Εθνικο Μετσοβιο Πολυτεχνειο
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Πληροφορικης και Συστηματων Υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων VLSI

Μεθοδολογιες Αυτοματοποιημενης Εξερευνησης και Συνθεσης Επαναδιαταξιμων Συνεπεξεργαστων Υλικου

Διδακτορικη Διατριβη

του

Σωτηριου Γ. Ξυδη

Αθήνα, Ιανουάριος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ VLSI

ΜΕΘΟΔΟΛΟΓΙΕΣ ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΗΣ ΕΞΕΡΕΥΝΗΣΗΣ ΚΑΙ ΣΥΝΘΕΣΗΣ ΕΠΑΝΑΔΙΑΤΑΞΙΜΩΝ ΣΥΝΕΠΕΞΕΡΓΑΣΤΩΝ ΥΛΙΚΟΥ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

ΣΩΤΗΡΙΟΥ Γ. ΞΥΔΗ

Συμβουλευτική Επιτροπή: ΚΙΑΜΑΛ ΠΕΚΜΕΣΤΖΗ
ΠΑΝΑΓΙΩΤΗΣ ΤΣΑΝΑΚΑΣ
ΝΕΚΤΑΡΙΟΣ ΚΟΖΥΡΗΣ

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 24η Ιανουαρίου 2011.

.....	.....	.....
-Κ. ΠΕΚΜΕΣΤΖΗ-	-Π. ΤΣΑΝΑΚΑΣ-	-Ν. ΚΟΖΥΡΗΣ-
-ΚΑΘ. ΕΜΠ-	-ΚΑΘ. ΕΜΠ-	-ΑΝ. ΚΑΘ. ΕΜΠ-
.....	.....	.....
-Δ. ΣΟΥΝΤΡΗΣ-	-Ν. ΜΗΤΡΟΥ-	-Γ. ΟΙΚΟΝΟΜΑΚΟΣ-
-ΕΠ. ΚΑΘ. ΕΜΠ-	-ΚΑΘ. ΕΜΠ-	-ΛΕΚΤΩΡ ΕΜΠ-

.....
-Γ. ΘΕΟΔΩΡΙΔΗΣ-
-ΕΠ. ΚΑΘ. ΠΑΝ. ΠΑΤΡΩΝ-

Αθήνα, Ιανουάριος 2011.

.....
(ΣΩΤΗΡΙΟΣ Γ. ΞΥΔΗΣ)
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Contents

List of Figures

List of Tables

Ευχαριστίες

Το κείμενο είναι σχεδόν έτοιμο... και η ''Ιθάκη'' φαίνεται να ξεπροβάλει στον ορίζοντα. Αντικρίζοντας για πρώτη φορά από τόσο κοντά τον προορισμό, η σκέψη μου κατακλύζεται από όλα εκείνα τα πρόσωπα που με συνόδευσαν σε αυτό το ταξίδι. Στο σημείο, λοιπόν, αυτό θα ήθελα να αφιερώσω λίγες γραμμές εκφράζοντας τις ευχαριστίες μου σε όλους αυτούς που συνέβαλαν, ο καθένας με το δικό του τρόπο, στην περιήγηση μου στα μονοπάτια της έρευνας.

Ξεκινώντας, θα ήθελα να ευχαριστήσω τα μέλη της επιτροπής παρακολούθησης της διατριβής. Πρώτα, τον επιβλέποντα Καθηγητή μου κ. Κιαμάλ Πεκμεστζή για την καθοδήγηση του και την εμπιστοσύνη του να με δεχθεί σαν συνεργάτη του. Πάνω από όλα όμως τον ευχαριστώ για την κριτική του στάση απέναντι στις ερευνητικές επιλογές μου και την συμπαράσταση του προς οποιαδήποτε ερευνητική κατεύθυνση ακολούθησα. Ευχαριστώ, επίσης, τον Καθηγητή κ. Παναγιώτη Τσανάκα και τον Αναπλ. Καθηγητή κ. Νεκτάριο Κοζύρη, οι οποίοι αποτέλεσαν για μένα ισχυρά πρότυπα επιστημονικής ακεραιότητας από τα προπτυχιακά μου ακόμα χρόνια. Θα ήθελα όμως να σταθώ και να ευχαριστήσω ιδιαίτερα ακόμα δύο ανθρώπους, η συμβολή των οποίων ήταν καθοριστική στη διαμόρφωση της πορείας μου ως διδακτορικός φοιτητής. Τον Επικ. Καθηγητή κ. Δημήτριο Σούντρη για την αμέριστη υποστήριξη του σε ερευνητικά και όχι μόνο θέματα, από την πρώτη στιγμή της γνωριμίας μας. Η έλευση του στο εργαστήριο με έφερε σε επαφή με νέες θεματικές περιοχές και τρόπους σκέψης που έπαιξαν σημαντικό ρόλο στο τελικό αποτέλεσμα της διατριβής. Επίσης, τον Λέκτορα κ. Γιώργιο Οικονομάκο που στάθηκε δίπλα μου στα αγωνιώδη πρώτα χρόνια του διδακτορικού. Ήταν αυτός που με παρότρυνε να μελετήσω την περιοχή των επαναδιατάξιμων συστημάτων σε μια περίοδο όπου η περιπλάνηση μέσα στον λαβύρινθο των ερευνητικών θεμάτων, φαινόταν αχανής.

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω τα υπόλοιπα μέλη της ομάδας του εργαστηρίου Μικρουπολογιστών και Ψηφιακών Συστημάτων. Τους Κωνσταντίνο Αναγνωστόπουλο, Δρ. Ισίδωρο Σιδέρη και Ζέφη Σκίνη που με καλοσώρισαν στο εργαστήριο. Τους μεταπτυχιακούς φοιτητές Δημήτριο Μπεκιάρη, Ηρακλή Αναγνωστόπουλο, Ιωάννη Κούτρα, Διονύση Διαμαντόπουλο, Χάρη Σιδηρόπουλο, Κωνσταντίνο Τσουμάνη, Ευστάθιο Σωτηρίου-Ξανθόπουλο, Νικόλαο Ζομπάκη και Μάρω Μπάκα και τους μεταδιδακτορικούς ερευνητές Δρ. Νικόλαο Αξελό, Δρ. Αντώνη Παπανικολάου, Δρ. Κωνσταντίνο Σιώζιο, Δρ. Αλέξανδρο Μπάρτζα για τις πολύωρες τεχνικές και φιλοσοφικές συζητήσεις και το υπέροχο κλίμα που έχουν δημιουργήσει. Θα ήθελα να αναφερθώ ειδικά στον Δρ. Αλέξανδρο Μπάρτζα για την άψογη συνεργασία και την υποστήριξη του σε πολλά επίπεδα. Εύχομαι σε όλους καλή συνέχεια. Μακάρι οι δρόμοι μας να σμίξουν ξανά στο μέλλον.

Οι τελευταίες γραμμές γράφονται δύσκολα... Πώς μπορεί κάποιος να εκφράσει με λόγια αυτό για το οποίο οι λέξεις ''ευχαριστώ'' και ''ευγνωμοσύνη'' ακούγονται τόσο λίγες; Πώς μπορεί κάποιος να αναφερθεί σε εκείνους που θέτουν πρόθυμα τον εαυτό

τους στην εκπλήρωση των δικών σου προσωπικών ονείρων; Στις τελευταίες αυτές γραμμές θα ήθελα λοιπόν να εκφράσω την αγάπη μου στην οικογένεια μου: στη μητέρα μου **Αγγελική**, στον πατέρα μου **Γιώργο**, στην αδελφή μου **Έφη**, καθώς και στη σύντροφο μου **Νάσια** με την οποία βαδίζουμε στο ίδιο μονοπάτι. Σας ευχαριστώ για την ανεκτίμητη στήριξη που μου προσφέρατε, την υπομονή σας, μα πάνω από όλα την παρουσία σας κοντά μου όλα αυτά τα χρόνια. Σε εσάς θα ήθελα να αφιερώσω την παρούσα διατριβή.

Σωτήριος Ξύδης, Ιανουάριος 2011

Abstract

Technological advances in micro-electronics, enabled the development of complex embedded computing devices, i.e. mobile phones, digital cameras, etc., which dominate the modern everyday life. Such type of systems usually execute a large but specific set of applications, which combine highly dynamic behavior together with high demands in computing power. The designers have to deal with the increased system complexity, in order to provide design solutions that satisfy a set of stringent functional and financial constraints. Today, it is broadly accepted that without the use of automated tools to optimize the hardware and software system's coefficients, designers are led to the adoption of sub-optimal design solutions. This thesis addresses the above problem by developing a set of methodologies for efficient design space exploration and architectural synthesis for digital signal processing coprocessors. Specifically, the proposed techniques target to (i) the development of customized software solutions for dynamic memory management of multi-threaded applications and (ii) the design of efficient customized and reconfigurable coprocessor architectures.

Regarding to the automated design space exploration methodologies, we model and analyze the basic building blocks of multi-threaded dynamic memory management for multi-core platforms with shared memory. We propose new algorithms for exploring and traversing the defined parameter space based on constrained orthogonal design space partitioning, which enables multi-objective optimization and automated code generation of application specific dynamic memory management. In addition, we propose a new approach for defining an extended design space for hardware coprocessor synthesis, which takes into account the combined impact of behavioral-algorithmic and architectural level parameters. New exploration algorithms are developed to enable fast and efficient exploration that converges to more globally optimal design solutions. Regarding to the architectural synthesis of reconfigurable coprocessors, we introduce the Flexibility Inlining technique for designing coarse-grained reconfigurable architectures at the circuit level. The proposed technique exploits the mirror symmetry found in ASIC implementations of arithmetic circuits and through appropriate RTL transformations achieves the design of a new reconfigurable micro-architectural template that exploits in a combined manner the architectural optimizations of horizontal parallelism, vertical parallel and operation chaining. In addition, we present a second micro-architectural template that use advanced arithmetic optimization techniques for designing flexible coprocessor datapaths. Each of the aforementioned reconfigurable architectures is complemented with new high level synthesis algorithms in order to enable automated mapping of applications onto the introduced datapaths.

We study the effectiveness of the proposed methodologies through multiple and extensive experimental evaluations of the proposed solutions in comparison with state-of-the-art design solutions. In any case, it seems that the adoption of the proposed methodologies leads to a significant shift of the design solutions towards more efficient implementations.

Περίληψη

Οι τεχνολογικές εξελίξεις στον τομέα της μικρο-ηλεκτρονικής επιτρέπουν σήμερα την ανάπτυξη πολύπλοκών ενσωματωμένων υπολογιστικών συσκευών π.χ. κινητά τηλέφωνα, ψηφιακές κάμερες, που κυριαρχούν τη σύγχρονη καθημερινότητα. Οι σχεδιαστές καλούνται να αντιμετωπίσουν την αυξημένη πολυπλοκότητα, παρέχοντας σχεδιαστικές λύσεις που ικανοποιούν ένα αντικρουόμενο σύνολο αυστηρών λειτουργικών και οικονομικών περιορισμών. Σήμερα, είναι γενικά αποδεκτό ότι χωρίς τη χρήση αυτοματοποιημένων εργαλείων που βελτιστοποιούν αισθητά τις συνιστώσες υλικού και λογισμικού του συστήματος, οι σχεδιαστές οδηγούνται στην υιοθέτηση υπο-βέλτιστων σχεδιαστικών λύσεων. Η παρούσα διατριβή στοχεύει στην αντιμετώπιση του παραπάνω προβλήματος μέσω της ανάπτυξης ενός συνόλου μεθοδολογιών σχεδίασης που επιτρέπουν αποδοτική εξερεύνηση και σύνθεση αρχιτεκτονικών λύσεων υλικού και λογισμικού. Πιο συγκεκριμένα, οι προτεινόμενες μεθοδολογίες αφορούν (i) την ανάπτυξη εξειδικευμένων λύσεων λογισμικού για δυναμική διαχείριση μνήμης πολυ-νηματικών εφαρμογών και (ii) τη σχεδίαση αποδοτικών λύσεων υλικού τόσο για επαναδιατάξιμες όσο και εξειδικευμένες αρχιτεκτονικές συνεπεξεργαστή.

Σε ότι αφορά στις μεθοδολογίες αυτοματοποιημένη εξερεύνησης, παρουσιάζεται η μοντελοποίηση και η ανάλυση σε βασικές συνιστώσες του χώρου σχεδίασης των δυναμικών διαχειριστών μνήμης για πολυ-νηματικές εφαρμογές. Προτείνονται νέοι αλγόριθμοι εξερεύνησης και διάσχισης του παραπάνω χώρου σχεδίασης βάσει μεθόδων ορθογώνιου διαχωρισμού των αποφάσεων, που επιτρέπουν την πολύ-κριτηριακή εξειδίκευση και την αυτοματοποιημένη παραγωγή λογισμικού των δυναμικών διαχειριστών μνήμης. Επιπλέον, προτείνεται μια νέα θεώρηση για την παραμετρική μοντελοποίηση του χώρου σχεδίασης εξειδικευμένων συνεπεξεργαστών υλικού, η οποία λαμβάνει υπόψη τις επιδράσεις των σχεδιαστικών αποφάσεων από τα επίπεδα αφαίρεσης συμπεριφοράς/αλγορίθμου και αρχιτεκτονικής, με στόχο τη συνδυασμένη βελτιστοποίηση της επίδοσης και της επιφάνειας υλικού. Νέοι αλγόριθμοι εξερεύνησης αναπτύχθηκαν, ώστε να επιτυγχάνεται γρήγορη αναζήτηση με αποδοτική σύγκλιση στις καθολικά βέλτιστες σχεδιαστικές λύσεις. Σε ότι αφορά στις μεθοδολογίες αρχιτεκτονικής σύνθεσης επαναδιατάξιμων συνεπεξεργαστών υλικού, παρουσιάζεται η τεχνική Εισαγωγή Ευελιξίας για κυκλωματική σχεδίαση επαναδιατάξιμων αρχιτεκτονικών αδρομερούς υφής. Η προτεινόμενη τεχνική εκμεταλλεύεται την κατοπτρική συμμετρία στις διασυνδέσεις εξειδικευμένων (μη-επαναδιατάξιμων) αριθμητικών κυκλωμάτων και μέσω κατάλληλων μετασχηματισμών, επιτυγχάνει τη σχεδίαση ενός νέου επαναδιατάξιμου μικρο-αρχιτεκτονικού πρότυπου, το οποίο συνδυάζει τον οριζόντιο και κάθετο παραλληλισμό με δυνατότητες υπολογιστικής αλυσίδωσης λειτουργιών. Επιπροσθέτως, προτείνεται και μια δεύτερη επαναδιατάξιμη αρχιτεκτονική συνεπεξεργαστή, η οποία βασίζεται στη χρήση μεθόδων αριθμητικής βελτιστοποίησης για σχεδίαση ευέλικτων συνεπεξεργαστών με αλυσίδωση λειτουργιών. Για κάθε επαναδιατάξιμο συνεπεξεργαστή αναπτύχθηκαν νέοι αλγόριθμοι σχεδίασης για αυτοματοποιημένη απεικόνιση εφαρμογών στις προτεινόμενες αρχιτεκτονικές.

Η μελέτη της αποδοτικότητας των προτεινόμενων μεθοδολογιών πραγματοποιήθηκε μέσω εκτενών πειραματικών αξιολογήσεων και συγκρίσεων με τις αντίστοιχες σχεδιαστικές λύσεις αιχμής. Σε κάθε περίπτωση, διαπιστώνεται πως η υιοθέτηση των μεθοδολογιών που αναπτύχθηκαν στην παρούσα διατριβή, οδηγεί σε σημαντική μετατόπιση των σχεδιαστικών λύσεων προς πιο αποδοτικές υλοποιήσεις.

0.1. Εισαγωγή

Ο χώρος του Αυτοματοποιημένου Ηλεκτρονικού Σχεδιασμού (Electronic Design Automation, EDA) αποτελεί ένα σημαντικό και συνεχώς αναπτυσσόμενο τομέα του ευρύτερου χώρου της σχεδίασης ολοκληρωμένων κυκλωμάτων (Integrated Circuits, IC). Το αυξημένο ενδιαφέρον για την ανεύρεση αποδοτικών EDA μεθοδολογιών οδηγείται σε μεγάλο βαθμό από τις εξελίξεις στις τεχνολογίες κατασκευής, οι οποίες ακολουθώντας πιστά το νόμο του Moore [23], επιτρέπουν σήμερα την ολοκλήρωση πολυ-πύρηνων επεξεργαστικών συστημάτων σε μια ψηφίδα (Multi-Processor System-on-Chip, MPSoC).

Τα ενσωματωμένα συστήματα (Ε.Σ.) αποτελούν υπολογιστικές συσκευές ειδικού σκοπού για συγκεκριμένο πεδίο εφαρμογής, π.χ. κινητά τηλέφωνα, ψηφιακές κάμερες. Σε αντιδιαστολή με τα υπολογιστικά συστήματα γενικού σκοπού, π.χ προσωπικοί υπολογιστές, φορητοί υπολογιστές, ένα ενσωματωμένο σύστημα καλείται να εκτελέσει συγκεκριμένες λειτουργίες υπό προκαθορισμένους περιορισμούς. Επιπλέον, τις περισσότερες φορές τα Ε.Σ. παρουσιάζουν μια εξαιρετικά δυναμική συμπεριφορά λόγω της συνεχούς αλληλεπίδρασης τους με το περιβάλλον λειτουργίας τους. Λαμβάνοντας υπόψη το εκάστοτε πεδίο εφαρμογής, οι σχεδιαστές Ε.Σ. αναζητούν αποδοτικές μεθοδολογίες σχεδίασης ικανές να παράγουν εξειδικευμένες λύσεις λογισμικού-υλικού που να ικανοποιούν τους σχεδιαστικούς περιορισμούς στο χαμηλότερο δυνατό κόστος, π.χ. επιφάνεια υλικού, κατανάλωση ισχύος.

Ο σχεδιασμός εξειδικευμένων συστημάτων είναι μια απαίτηση που δημιουργείται από την αυξημένη πολυπλοκότητα των σύγχρονων ενσωματωμένων εφαρμογών. Οι συμβατικές σχεδιαστικές προσεγγίσεις, π.χ. εξειδικευμένο λογισμικό για μονο-πύρηνες επεξεργαστικές πλατφόρμες υλικού, δεν καταφέρνουν να ανταπεξέλθουν στις σύγχρονες απαιτήσεις των Ε.Σ. Το Σχ. 0.1 δείχνει ένα τέτοιο παράδειγμα σχετικά με τη σημαντικότητα και την αναγκαιότητα για εξειδικευμένες λύσεις λογισμικού-υλικού. Η εφαρμογή αποκωδικοποίησης του MPEG-4 [24] απεικονίστηκε σε τέσσερις διαφορετικές αρχιτεκτονικές επεξεργαστή: (1) σε έναν επεξεργαστή ARM9 [25], (2) έναν ARM11 [25] (επεξεργαστής με πιο πλούσιο σύνολο εντολών για αποδοτική υποστήριξη λειτουργιών ψηφιακής επεξεργασίας σήματος (Digital Signal Processing, DSP), (3) έναν C6x VLIW από την Texas Instruments [26] (επεξεργαστής με υπερεξειδικευμένο σύνολο εντολών για λειτουργίες DSP) και (4) έναν δι-πύρηνο επεξεργαστή Intel Core 2 Duo [27]. Οι ARM9, ARM11 και C6x VLIW αποτελούν επεξεργαστές που προορίζονται για ενσωματωμένα συστήματα, ενώ αντίθετα ο Intel Core 2 Duo είναι επεξεργαστής γενικού σκοπού. Ο σχεδιαστικός περιορισμός αφορά τη διασφάλιση του ελάχιστου επιτρεπτού ρυθμού αποκωδικοποίησης (24 πλαίσια εικόνας ανά δευτερόλεπτο), προκειμένου η ποιότητα της εικόνας να είναι ανεκτή στον τελικό χρήστη. Η αξιολόγηση πραγματοποιήθηκε για τις μετρικές επίδοσης που αφορούν στο ρυθμό αποκωδικοποίησης και στην κατανάλωση ισχύος. Από το Σχ. 0.1 διαπιστώνεται ότι οι λύσεις λογισμικού που απεικονίστηκαν στους επεξεργαστές για ενσωματωμένα συστήματα, απέτυχαν να ικανοποιήσουν το σχεδιαστικό περιορισμό του ελάχιστου ρυθμού αποκωδικοποίησης. Ο δι-πύρηνος επεξεργαστής γενικού σκοπού ικανοποιεί το σχεδιαστικό περιορισμό αλλά με υπερβολική κατανάλωση ισχύος, η οποία οδηγεί σε ελάχιστη ενεργειακή αυτονομία του συστήματος. Από το παραπάνω παράδειγμα, διαπιστώνουμε ότι προκειμένου να ικανοποιήσουμε αυστηρούς σχεδιαστικούς περιορισμούς επιδόσεων μέσα σε αποδεκτά ενεργειακά όρια, επιβάλλεται μια συντονισμένη εξειδίκευση τόσο σε επίπεδο λογισμικού (π.χ παράλληλες/πολυνη-

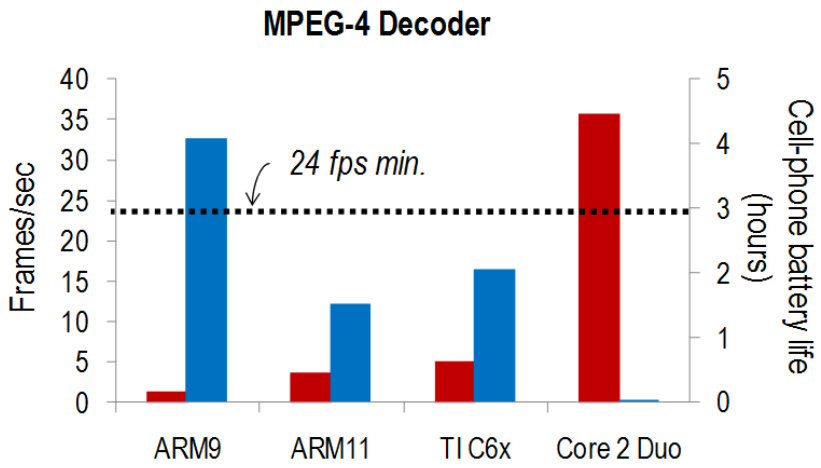


Figure 0.1.: Ανάγκη για εξειδίκευση Υλικού-Λογισμικού.

ματικές εφαρμογές) όσο και σε επίπεδο υλικού (π.χ. υποστήριξη συνεπεξεργαστών υλικού).

Με γνώμονα τις παραπάνω κινητήριες παρατηρήσεις, η παρούσα διατριβή στοχεύει στην ανάπτυξη ενός συνόλου αποδοτικών μεθοδολογιών σχεδίασης για σύγχρονες αρχιτεκτονικές Ε.Σ. Χωρίς βλάβη της γενικότητας θεωρούμε ένα αφηρημένο αρχιτεκτονικό πρότυπο Ε.Σ. (Σχ. 0.2) το οποίο αποτελείται από (1) ένα σύνολο μικροεπεξεργαστών οι οποίοι παρέχουν στην πλατφόρμα δυνατότητες παραλληλίας σε επίπεδο διεργασίας ή νήματος, (2) ένα σύνολο από επαναδιατάξιμους ή μη συνεπεξεργαστές υλικού για επιτάχυνση των υπολογιστικά απαιτητικών διεργασιών, (3) μια πολυεπίπεδη ιεραρχία μνήμης για αποθήκευση των δεδομένων και (4) ένα δίκτυο διασύνδεσης για τη μεταφορά των δεδομένων μεταξύ των στοιχείων επεξεργασίας και αποθήκευσης. Οι προτεινόμενες μεθοδολογίες σχεδίασης περιλαμβάνουν τεχνικές εξειδίκευσης των συνιστωσών λογισμικού που αναλαμβάνουν τη διαχείριση της δυναμικής μνήμης του συστήματος καθώς και των συνιστωσών υλικού για την επιτάχυνση των υπολογιστικά κρίσιμων τμημάτων της εφαρμογής σε επαναδιατάξιμους ή μη συνεπεξεργαστές.

0.2. Υπόβαθρο - Περιγραφή Χώρου Έρευνας

Το πλαίσιο της παρούσας διδακτορικής διατριβής καθορίζεται από δυο κεντρικούς άξονες, που οριοθετούν και προσδιορίζουν το χώρο έρευνας. Πιο συγκεκριμένα, ο πρώτος άξονας αφορά στις μεθοδολογίες αυτοματοποιημένης εξερεύνησης σχεδιαστικών χώρων (Design Space Exploration, DSE), ώστε ο σχεδιαστής να οδηγείται σε βελτιστοποιημένες σχεδιαστικές λύσεις λαμβάνοντας υπόψη πολλαπλούς παράγοντες κόστους και περιορισμούς. Ο δεύτερος άξονας αφορά στις μεθοδολογίες σύνθεσης αρχιτεκτονικής από υψηλό επίπεδο (Architectural ή High Level Synthesis, HLS) επαναδιατάξιμων συνεπεξεργαστών υλικού αδρομερούς υφής (Coarse-Grained Reconfigurable Coprocessors, CGRCs). Οι HLS μεθοδολογίες επιτρέπουν την αυτοματοποιημένη

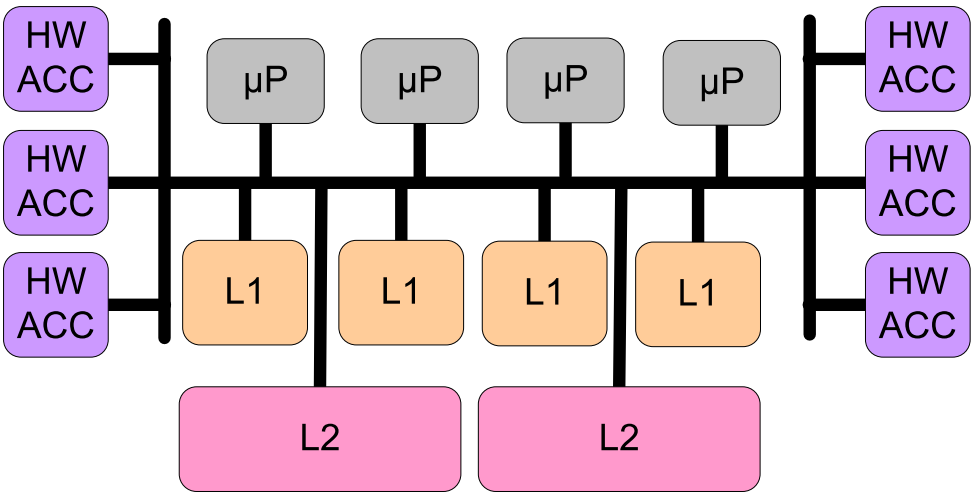


Figure 0.2.: Αφηρημένο αρχιτεκτονικό πρότυπο Ε.Σ. .

σύνθεση αρχιτεκτονικών υλικού απευθείας από την αλγοριθμική περιγραφή (π.χ. σε γλώσσα προγραμματισμού C) της επιθυμητής συμπεριφοράς επιτρέποντας την επιτάχυνση των υπολογιστικά απαιτητικών διεργασιών μέσω απεικόνισης τους σε υλικό, βελτιώνοντας παράλληλα την παραγωγικότητα του σχεδιαστή, καθώς δεν χρειάζεται να γνωρίζει χαμηλού επιπέδου σχεδιαστικές λεπτομέρειες. Επιπροσθέτως, οι αρχιτεκτονικές CGRC λειτουργούν ως ένα επιπλέον επίπεδο βελτιστοποίησης, επιτρέποντας επιτάχυνση των υπολογιστικά απαιτητικών διεργασιών με χαμηλότερο κατασκευαστικό κόστος, λόγω των αυξημένων δυνατοτήτων διαμοιρασμού υλικού. Οι δύο προαναφερόμενοι άξονες περιγράφονται συνοπτικά στις υπο-ενότητες που ακολουθούν.

0.2.1. Αυτοματοποιημένη Εξερεύνηση Σχεδιαστικών Χώρων

Η εξερεύνηση ενός σχεδιαστικού χώρου είναι η διαδικασία διερεύνησης πολλαπλών σχεδιαστικών παραλλαγών, προκειμένου ο σχεδιαστής να οδηγηθεί σε μια βέλτιστη λύση (στην περίπτωση που υποθέτουμε μονο-κριτηριακή αντικειμενική συνάρτηση κόστους). Στην περίπτωση που το προς μελέτη σχεδιαστικό πρόβλημα απαιτεί τη βελτιστοποίηση μιας πολύ-κριτηριακής συνάρτησης κόστους, π.χ. μια συνάρτηση κόστους που περιλαμβάνει βελτιστοποίηση της επιφάνειας υλικού και της καθυστέρησης για σχεδιαστικά προβλήματα σύνθεσης συνεπεξεργαστών υλικού, ή μια συνάρτηση κόστους που περιλαμβάνει βελτιστοποίηση του αποτυπώματος μνήμης και του αριθμού των προσπελάσεων στη μνήμη για σχεδιαστικά προβλήματα που αφορούν τη δυναμική διαχείριση μνήμης κ.τ.λ., τότε, αντί για μία μοναδική βέλτιστη λύση αναζητείται ένα σύνολο από βέλτιστες λύσεις (Pareto-βέλτιστες [28]). Η έννοια της Pareto κυριαρχίας μιας λύσης A σε σχέση με μια λύση B, οι οποίες έχουν χαρακτηριστεί βάσει του συνόλου των κριτηρίων βελτιστοποίησης, ορίζεται αν η λύση A είναι καλύτερη σε τουλάχιστον ένα κριτήριο βελτιστοποίησης χωρίς να είναι χειρότερη σε κανένα από τα υπόλοιπα κριτήρια. Οι Pareto-βέλτιστες λύσεις είναι ακριβώς οι λύσεις αυτές που παρουσιάζουν την παραπάνω ιδιότητα της Pareto κυριαρχίας σε σχέση με κάθε άλλη λύση από το σύνολο λύσεων. Προκειμένου να αξι-

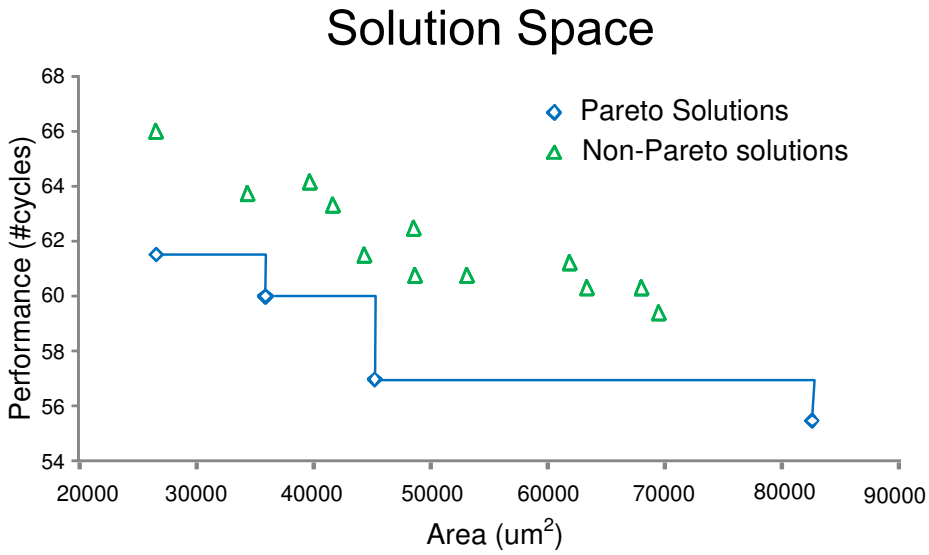


Figure 0.3.: Παράδειγμα Pareto κυριαρχίας .

ολογηθεί αν μια λύση ανήκει στο σύνολο των Pareto-βέλτιστων λύσεων, η τιμή που λαμβάνει η πολύ-κριτηριακή συνάρτηση κόστους για τη συγκεκριμένη σχεδιαστική λύση θα πρέπει να συγκριθεί εξαντλητικά σε σχέση με τις αντίστοιχες τιμές που προκύπτουν από το σύνολο των υπόλοιπων προς εξέταση σχεδιαστικών λύσεων. Το Σχ. 0.3 απεικονίζει έναν σύνολο λύσεων χαρακτηρισμένων βάσει των κριτηρίων καθυστέρησης (άξονας Υ) και επιφάνειας υλικού (άξονας Χ). Όλα τα σημεία ανήκουν στο χώρο των εφικτών λύσεων, όμως τα τέσσερα συνδεδεμένα σημεία αποτελούν το σύνορο των Pareto-βέλτιστων λύσεων.

Συνήθως, τα μεγέθη που αφορούν τα κριτήρια βελτιστοποίησης, π.χ. καθυστέρηση απόκρισης - επιφάνεια υλικού, σε μια πολύ-κριτηριακή συνάρτηση κόστους είναι στενά συνδεδεμένα μεταξύ τους και συνήθως αντικρουόμενα, με αποτέλεσμα η βελτιστοποίηση ενός μόνο κριτηρίου συχνά να επιφέρει σοβαρούς συμβιβασμούς στα υπόλοιπα κριτήρια. Στην πραγματικότητα, ένα σύστημα το οποίο έχει βελτιστοποιηθεί με μοναδικό κριτήριο την ελαχιστοποίηση της καθυστέρησης συνήθως αποτελεί μια από τις χειρότερες λύσεις σε ότι αφορά την επιφάνεια υλικού και αντιστρόφως.

Τα προβλήματα εξερεύνησης σχεδιαστικών χώρων συνήθως μοντελοποιούνται μέσω τριών δομικών στοιχείων, (1) του χώρου παραμέτρων, (2) του χώρου λύσεων και (3) της συνάρτησης απεικόνισης. Ο χώρος παραμέτρων ορίζεται από τις αποφάσεις που πρέπει να ληφθούν για την επίλυση του προβλήματος σχεδιασμού. Οι αποφάσεις αυτές επηρεάζουν τις τιμές των κριτηρίων βελτιστοποίησης, χωρίς όμως να τα αναπαριστούν. Η λήψη διαφορετικών αποφάσεων οδηγεί σε ένα διακριτό σύνολο σχεδιαστικών λύσεων, η κάθε μια από τις οποίες χαρακτηρίζεται από συγκεκριμένες τιμές σε ότι αφορά τα κριτήρια βελτιστοποίησης. Για παράδειγμα, ένας διαχειριστής δυναμικής μνήμης μπορεί να περιγραφεί από ένα σύνολο μηχανισμών π.χ. μηχανισμοί αναζήτησης ελεύθερων μπλοκ μνήμης, μηχανισμοί τεμαχισμού μπλοκ μνήμης κ.τ.λ., οι οποίοι δεν δίνουν καμία πληροφορία για τις τιμές των κριτηρίων βελτιστοποίησης, όπως π.χ. ίχνος μνήμης, χρόνος εκτέλεσης κ.τ.λ.. Ο χώρος λύσεων ορίζεται από τις τιμές που λαμβάνουν τα κριτήρια βελτιστοποίησης, όπως το κόστος του συστήματος,

η ταχύτητα εκτέλεσης και η κατανάλωση ισχύος. Τέλος, η συνάρτηση απεικόνισης είναι η διαδικασία η οποία απεικονίζει τα διανύσματα του χώρου παραμέτρων στα αντίστοιχα στιγμιότυπα του χώρου λύσεων. Προγράμματα προσομοίωσης, αναλυτικά μοντέλα, αφαιρετικά μοντέλα, συνδυασμοί διαδικασιών όπως η μεταγλώττιση-εκτέλεση για τις συνιστώσες λογισμικού ή αρχιτεκτονική σύνθεση και σύνθεση πυλών για τις συνιστώσες υλικού κ.τ.λ., μπορούν να χρησιμοποιηθούν ως έγκυρες συναρτήσεις απεικόνισης για την εξερεύνηση ενός σχεδιαστικού χώρου [29].

Οι αλγόριθμοι εξερεύνησης μπορούν να εφαρμοστούν είτε στο χώρο παραμέτρων είτε στο χώρο λύσεων. Ένα αφαιρετικό λογικό διάγραμμα ροής για τους αλγορίθμους εξερεύνησης στο χώρο παραμέτρων αποτελείται συνήθως από τρία βήματα: (1) συστηματική επιλογή ενός συνόλου από διανύσματα παραμέτρων, (2) παραγωγή του αντίστοιχου χώρου λύσεων και (3) αξιολόγηση του χώρου λύσεων και εξαγωγή του Pareto συνόρου. Αντίθετα, η τυπική προσέγγιση που ακολουθείται από τους αλγορίθμους εξερεύνησης στο χώρο λύσεων επικεντρώνεται στο συστηματικό περιορισμό του χώρου λύσεων (άρα και των εξεταζόμενων διανυσμάτων του χώρου παραμέτρων) κατά τη διάρκεια της διαδικασίας διερεύνησης π.χ. ο αλγόριθμος καθορίζει κατά τη διάρκεια εκτέλεσης του για το αν μπορεί να βρεθεί μια σχεδιαστική λύση η οποία θα ικανοποιεί τους περιορισμούς της σχεδίασης ενώ παράλληλα βελτιστοποιεί τα κριτήρια της αντικειμενικής συνάρτησης κόστους.

Η πολύ-κριτηριακή εξερεύνηση αποτελεί το πιο ρεαλιστικό σενάριο στη σχεδίαση ενσωματωμένων συστημάτων, δεδομένου ότι ο σχεδιαστής καλείται να επιλέξει μια βελτιστοποιημένη λύση η οποία ικανοποιεί αυστηρούς περιορισμούς. Στην πραγματικότητα, κάθε σχεδιαστικό πρόβλημα που συναντάται σε μια ροή σχεδίασης μπορεί να θεωρηθεί ως πρόβλημα πολύ-κριτηριακής εξερεύνησης. Σε τέτοιου είδους προβλήματα, η πολυπλοκότητα της διαδικασίας εξερεύνησης εξαρτάται από το μέγεθος του εξεταζόμενου χώρου λύσεων. Στην τυπική περίπτωση, οι παραγόμενοι χώροι λύσεων είναι υπερβολικά μεγάλοι με αποτέλεσμα μια προσέγγιση βασιζόμενη στην εξαντλητική εξερεύνηση να είναι πρακτικά αδύνατη λόγω του υπολογιστικού φόρτου. Επιπλέον, μεθοδολογίες εξερεύνησης βάσει τυχαίας δειγματοληψίας του χώρου λύσεων ή παραμέτρων δεν παρέχουν καμία εγγύηση σύγκλισης προς τις πραγματικά βέλτιστες λύσεις, αυξάνοντας το κίνδυνο για υιοθέτηση υπο-βέλτιστων σχεδιάσεων. Κατά συνέπεια, υπάρχει ανάγκη για αυτοματοποιημένες αλλά ταυτόχρονα στοχευμένες μεθοδολογίες εξερεύνησης, υπό την έννοια ότι οι αλγόριθμοι εξερεύνησης θα πρέπει να αναπτύσσονται με γνώμονα όχι μόνο την αυτοματοποίηση αλλά και την ενσωμάτωση γνώσης σχετικά με τη δομή του χώρου σχεδίασης ώστε να επιτυγχάνεται η σύγκλιση προς τα πραγματικά Pareto σύνορα, χωρίς να απαιτείται μια εξαντλητική προσέγγιση διερεύνησης.

Η παρούσα διατριβή επικεντρώνεται σε δύο προβλήματα εξερεύνησης που παρουσιάζονται κατά τη διάρκεια σχεδιασμού ενός ενσωματωμένου υπολογιστικού συστήματος. Το πρώτο πρόβλημα εξερεύνησης αφορά στη σχεδίαση εξειδικευμένων διαχειριστών λογισμικού δυναμικής μνήμης σε πολύ-νηματικές εφαρμογές. Πιο συγκεκριμένα, προτείνεται μια συστηματική μεθοδολογία εξερεύνησης η οποία εφαρμόζεται στο χώρο παραμέτρων του προβλήματος, αξιοποιώντας τις μεταξύ τους εξαρτήσεις για μείωση του χώρου αναζήτησης μέσω πρόωρης αναγνώρισης και εξάλειψης μη-Pareto -βέλτιστων λύσεων. Οι προτεινόμενες μεθοδολογίες παρέχουν εξειδικευμένη διαχείριση της δυναμικής μνήμης σε πολύ-νηματικές εφαρμογές προεκτείνοντας προηγούμενες ερευνητικές προσπάθειες οι οποίες λάμβαναν υπόψη μονο-νηματικές εφαρμογές [30] . Το δεύτερο πρόβλημα εξερεύνησης αφορά στη σύνθεση εξειδικευμένων

αρχιτεκτονικών συνεπεξεργαστών υλικού με σκοπό τη βελτιστοποίηση των κριτηρίων καθυστέρησης και επιφάνειας υλικού. Οι υπάρχουσες προσεγγίσεις δεν εξετάζουν τη συνδυασμένη επίδραση των αλγοριθμικών και των αρχιτεκτονικών παραμέτρων του συγκεκριμένου προβλήματος, με αποτελέσματα να αποκλείουν από το χώρο αναζήτησης ένα σημαντικό αριθμό λύσεων, οι οποίες συνήθως ανήκουν στο Pareto σύνορο. Προκειμένου να ξεπεραστεί ο παραπάνω περιορισμός, προτείνεται μια συστηματική μεθοδολογία εξερεύνησης στο χώρο λύσεων του προβλήματος η οποία λαμβάνει υπόψη ένα επαυξημένο χώρο παραμέτρων και ενσωματώνει γνώση της δομής του χώρου αυτού, ώστε να αποκλείει δυναμικά κατά τη διάρκεια εξερεύνησης σχεδιαστικές λύσεις που δεν συγκλίνουν σε Pareto αρχιτεκτονικές.

0.2.2. Επαναδιατάξιμες Αρχιτεκτονικές Υλικού

Οι επαναδιατάξιμες αρχιτεκτονικές υλικού [31] αποτελούν ένα καινούργιο και ταχέως αναπτυσσόμενο πεδίο έρευνας στο χώρο της ψηφιακής σχεδίασης. Εισάγουν ένα νέο υπολογιστικό μοντέλο το οποίο αυξάνει τη διαθέσιμη λογική πυκνότητα των κυκλωμάτων, επιτρέποντας τη σχεδίαση αρχιτεκτονικών που παρουσιάζουν τις υψηλές επιδόσεις των εξειδικευμένων κυκλωμάτων υλικού με τις δυνατότητες ευελιξίας/προγραμματισμού που προσφέρονται από αρχιτεκτονικές μικρο-επεξεργαστών. Η ευελιξία αυτή αυξάνει τις δυνατότητες για επαναχρησιμοποίηση του υλικού, καθώς ο ίδιος επαναδιατάξιμος συνεπεξεργαστής δύναται να μεταβάλλει τη λειτουργία του ανά διάφορα αμοιβαίως αποκλειόμενα χρονικά παράθυρα, ανάλογα με την εκάστοτε εφαρμογή που του ανατίθεται. Επομένως, στο ίδιο υλικό αντιστοιχίζονται περισσότεροι του ενός πυρήνες της εφαρμογής, σε αντίθεση με τους τυπικούς συνεπεξεργαστές υλικού ειδικού σκοπού, κάθε ένας από τους οποίους προϋποθέτει εκχώρηση υλικού.

Το Σχ. 0.4 συγκρίνει τις διάφορες τεχνολογίες υλοποίησης σε ότι αφορά την επίδοση (κατανάλωση) έναντι της υποστηριζόμενης ευελιξίας. Με τον όρο επίδοση, εννοούμε την καθυστέρηση εκτέλεσης μιας εφαρμογής, ενώ με τον όρο ευελιξία αναφερόμαστε στη δυνατότητα προγραμματισμού του συστήματος. Η απεικονιζόμενη τάση δείχνει πως η αύξηση της ευελιξίας του υλικού έχει αρνητική επίπτωση στην επίδοση, ή από μια διαφορετική σκοπιά η εξειδίκευση του υλικού ανάλογα με την εφαρμογή οδηγεί σε κέρδη επίδοσης. Οι προγραμματιζόμενοι επεξεργαστές εμφανίζουν τη μεγαλύτερη ευελιξία εις βάρος των σχετικά χαμηλών επιδόσεων, ενώ αντίστροφα τα κυκλώματα ειδικού σκοπού (τα Application Specific ICs (ASICs) και τα βελτιστοποιημένα σε επίπεδο τρανζίστορ ICs) παρουσιάζουν τις καλύτερες επιδόσεις αλλά με μηδενική ευελιξία. Οι επαναδιατάξιμες αρχιτεκτονικές τοποθετούνται στο ενδιάμεσο μεταξύ των δύο προηγούμενων ακραίων περιπτώσεων, περιορίζοντας έτσι το βαθμό ευελιξίας τους σε συγκεκριμένους τομείς εφαρμογών.

Ένας σημαντικός αριθμός επαναδιατάξιμων αρχιτεκτονικών έχει ήδη προταθεί στη βιβλιογραφία [32]. Οι επαναδιατάξιμες αρχιτεκτονικές διακρίνονται σε αρχιτεκτονικές με δυνατότητες λεπτομερούς και αδρομερούς επαναδιάταξης. Οι αρχιτεκτονικές λεπτομερούς επαναδιάταξης όπως τα FPGAs, υποστηρίζουν επαναδιάταξη στο επίπεδο του bit, (επιτρέποντας την απεικόνιση μεγάλου εύρους εφαρμογών) χαρακτηρίζονται όμως από υψηλή καθυστέρηση επαναδιάταξης και κατανάλωση ισχύος. Επιπλέον, αυτού του βαθμού η παρεχόμενη ευελιξία αφορά σε περιπτώσεις σχεδίασης που οι υπολογιστικές απαιτήσεις, είτε δεν είναι γνωστές εκ των προτέρων είτε διαφέρουν σημαντικά μεταξύ των εφαρμογών. Σε πολλές περιπτώσεις, αυτός ο ακραίος βαθμός

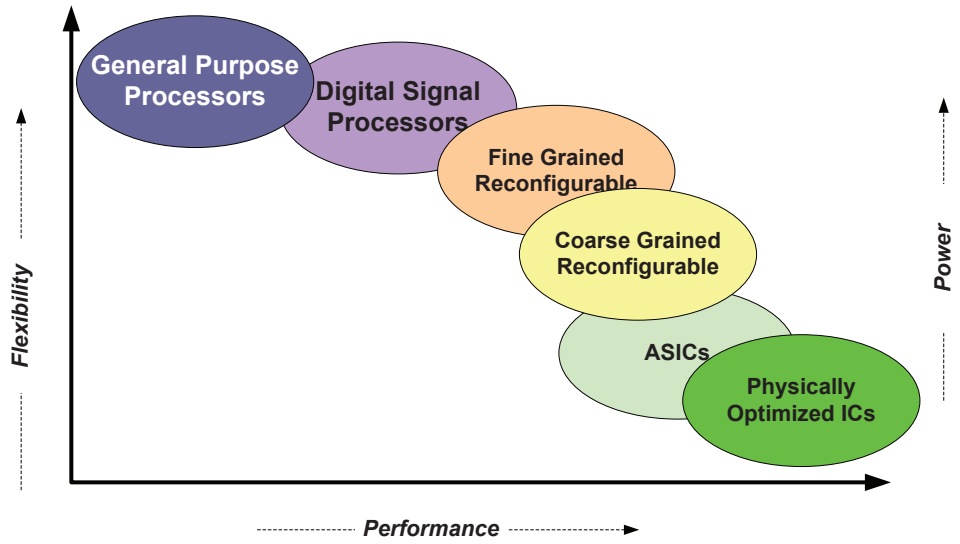


Figure 0.4.: Τοποθέτηση Τεχνολογιών Υλοποίησης Συστημάτων Υλικού .

ευελιξίας είναι περιττός και έχει ως αποτέλεσμα αυξημένο κόστος υλοποίησης, λόγω επιφάνειας υλικού και της κατανάλωσης ισχύος. Αντίθετα, στις αρχιτεκτονικές αδρομερούς επαναδιάταξης , η επαναδιάταξη επιτελείται στο επίπεδο των λειτουργικών μονάδων παρά στο επίπεδο του bit . Με τον τρόπο αυτό, η διαδικασία επαναδιάταξης επιτελείται ταχύτερα από εκείνη στα FPGAs . Επιπλέον, δεδομένου του εκάστοτε πεδίου εφαρμογής, εξειδικευμένες επαναδιατάξιμες αρχιτεκτονικές αδρομερούς υφής, προσαρμοσμένες στις ειδικές ανάγκες του συνόλου εφαρμογών, μπορούν να σχεδιαστούν ώστε να οδηγήσουν σε συστήματα με δραστικώς καλύτερα χαρακτηριστικά επιφάνειας υλικού, επίδοσης και κατανάλωσης ισχύος σε σχέση με τις αρχιτεκτονικές λεπτομερούς επαναδιάταξης.

Η παρούσα διατριβή επικεντρώνεται στη σχεδίαση επαναδιατάξιμων αρχιτεκτονικών επεξεργαστή αδρομερούς υφής για εφαρμογές που προέρχονται από το πεδίο DSP . Τυπικά, οι DSP εφαρμογές δομούνται από πολλούς υπολογιστικά απαιτητικούς πυρήνες που εκτελούν ένα μεγάλο αριθμό αριθμητικών πράξεων. Τέτοιου είδους εφαρμογές προσφέρουν πρόσφορο έδαφος για την ανάπτυξη μεθοδολογιών σχεδίασης με στόχο τη βελτιστοποίηση των κριτηρίων καθυστέρησης λειτουργίας και καταλαμβανόμενης επιφάνειας υλικού.

Στους DSP συνεπεξεργαστές, οι μονάδες πολλαπλασιασμού αποτελούν κρίσιμα δομικά στοιχεία λόγω της υψηλής πολυπλοκότητας (κυκλωματικής) και καθυστέρησης (μεγάλα κρίσιμα μονοπάτια) που παρουσιάζουν. Βάσει της παραπάνω ανάλυσης, εστιάζουμε στο σχεδιασμό επαναδιατάξιμων αρχιτεκτονικών οι οποίες θα αξιοποιούν με αποδοτικό τρόπο την επιφάνεια υλικού που έχει εκχωρηθεί στα κυκλώματα πολλαπλασιασμού. Τυπικά, οι αρχιτεκτονικές αδρομερούς επαναδιάταξης σχεδιάζονται και προδιαγράφονται στο επίπεδο της μικρο-αρχιτεκτονικής, όπου καθορίζονται οι δομικές μονάδες υλικού π.χ. αριθμητικοί τελεστές, οι οποίες είναι κοινές και διαμοιραζόμενες μεταξύ των αμοιβαίως αποκλειόμενων πυρήνων που απεικονίζονται στον επαναδιατάξιμο συνεπεξεργαστή. Ωστόσο, στην παρούσα διατριβή ακολουθούμε μια μάλλον διαφορετική προσέγγιση του σχεδιασμού από εκείνο χρησιμοποιείται

συνήθως για αρχιτεκτονικές αδρομερούς επαναδιάταξης. Συγκεκριμένα, εντοπίζουμε πως εξετάζοντας ενιαία το πρόβλημα σχεδίασης στο επίπεδο της μικρο-αρχιτεκτονικής και στο πιο λεπτομερές επίπεδο του bit, ο τυπικός σχεδιασμός αρχιτεκτονικών αδρομερούς επαναδιάταξης, που βασίζεται στην αρχή διαμοιρασμού του υλικού στο επίπεδο των δομικών μονάδων, μπορεί να επεκταθεί (1) με δυνατότητες διαμοιρασμού κυκλωματικών δομών που βρίσκονται στο επίπεδο του bit, παρέχοντας πιο αποδοτικές λύσεις αναφορικά με την καταλαμβανόμενη επιφάνεια υλικού καθώς και (2) με την ενσωμάτωση αριθμητικών βελτιστοποιήσεων, π.χ. οι επαναδιατάξιμες δομικές μονάδες να εκτελούν υπολογισμούς σε Carry-Save αριθμητική για επίτευξη υψηλότερων ταχυτήτων λειτουργίας.

Με βάση τις παραπάνω κινητήριες παρατηρήσεις, προτείνουμε δύο νέα ευέλικτα αρχιτεκτονικά πρότυπα αδρομερούς επαναδιάταξης, ένα για αρχιτεκτονικές υλικού που βασίζονται σε μονάδες πολλαπλασιασμού τύπου-πίνακα και ένα δεύτερο για αρχιτεκτονικές υλικού που βασίζονται σε μονάδες πολλαπλασιασμού τύπου-δέντρο. Το πρώτο αρχιτεκτονικό πρότυπο συνδυάζει τεχνικές διαμοιρασμού στο επίπεδο του bitκαι στο επίπεδο των δομικών μονάδων μαζί με τεχνικές αριθμητικής βελτιστοποίησης. Πιο συγκεκριμένα, εισάγουμε την τεχνική μετασχηματισμού Εισαγωγή Ευελιξίας, που επιτρέπει την απεικόνιση διαφόρων αριθμητικών τελεστών πάνω στη δομή διασύνδεσης πολλαπλασιαστών τύπου-πίνακα, παρέχοντας έτσι αυξημένες δυνατότητες επαναχρησιμοποίησης υλικού. Το δεύτερο αρχιτεκτονικό πρότυπο αφορά κυρίως στην χρήση τεχνικών αριθμητικής βελτιστοποίησης. Συγκεκριμένα, προτείνονται η σχεδίαση επαναδιατάξιμων δομικών μονάδων οι οποίες ενσωματώνουν προηγμένες τεχνικές αριθμητικής επανα-κωδικοποίησης, προκειμένου να καταστούν δυνατοί οι μη-διακοπτόμενοι υπολογισμοί στην αριθμητική αναπαράσταση Carry-Save, προσφέροντας υψηλότερες ταχύτητες λειτουργίας μέσω της μείωσης των μεγάλων αλυσίδων μεταφοράς κρατουμένου που χαρακτηρίζουν τις συμβατικές σχεδιάσεις υλικού.

0.2.3. Σύνθεση Αρχιτεκτονικής

Οι προτεινόμενες επαναδιατάξιμες αρχιτεκτονικές εισάγουν ένα νέο μικρο-αρχιτεκτονικό πρότυπο, τα χαρακτηριστικά του οποίου πρέπει να ληφθούν υπόψη κατά την διαδικασία απεικόνισης της εφαρμογής. Ο σχεδιαστής πρέπει να γνωρίζει τις μικρο-αρχιτεκτονικές λεπτομέρειες, ώστε να απεικονίσει αποτελεσματικά την εκάστοτε εφαρμογή, κάτι που περιορίζει συνήθως την παραγωγικότητα. Στην κλασσική περίπτωση ανάπτυξης λογισμικού ο περιορισμός αυτός αντιμετωπίζεται με την εκτεταμένη χρήση των γλωσσών προγραμματισμού και των μεταγλωττιστών οι οποίοι μεταφράζουν την αφαιρετική περιγραφή συμπεριφοράς της γλώσσας προγραμματισμού σε μια γλώσσα μηχανής πολύ πιο κοντά στην αρχιτεκτονική υλικού. Βασισμένη σε παρόμοιους στόχους, η κοινότητα της σχεδίασης υλικού έχει προτείνει τη χρήση HLSμεθοδολογιών για αυτοματοποιημένη σύνθεση αρχιτεκτονικής - επίσης γνωστή ως σύνθεση από υψηλό επιπέδου αφαίρεσης. Οι υπάρχουσες μεθοδολογίες αυτοματοποιημένης αρχιτεκτονικής σύνθεσης έχουν αναπτυχθεί με γνώμονα τη σχεδίαση κυκλωμάτων ειδικού σκοπού τύπου ASIC. Η ανάπτυξη μεθοδολογίων αυτοματοποιημένης σύνθεσης επαναδιατάξιμων αρχιτεκτονικών κρίνεται ιδιαίτερα σημαντική για το πεδίο των επαναδιατάξιμων συστημάτων, προκειμένου να καταφέρουν να ανταγωνιστούν την ASIC σχεδίαση.

Ο όρος σύνθεση αρχιτεκτονικής αναφέρεται στη διαδικασία η οποία μετασχηματίζει αλγοριθμικές περιγραφές σε ισοδύναμες κυκλωματικές δομές στο επίπεδο της μικρο-αρχιτεκτονικής. Οι κυκλωματικές δομές ορίζουν το ακριβές σχήμα διασύνδεσης μεταξύ δομικών μονάδων π.χ. αριθμητικοί τελεστές υλικού (ALUs, πολλαπλασιαστές), στοιχείων μνήμης (καταχωρητές, μπλοκ μνήμης), στοιχείων διασύνδεσης π.χ. πολυπλέκτες και μονάδων ελέγχου. Το πρόβλημα της αρχιτεκτονικής σύνθεσης μπορεί να διατυπωθεί με τον ακόλουθο τρόπο: *Δεδομένων της αλγοριθμικής περιγραφής μιας εφαρμογής, ενός συνόλου πόρων υλικού, ενός συνόλου σχεδιαστικών περιορισμών και μιας αντικειμενικής συνάρτησης, να καθοριστεί ένα σύνολο διασύνδεσης μεταξύ των διαθέσιμων πόρων υλικού λειτουργικά ισοδύναμο με την αλγοριθμική περιγραφή, τέτοιο ώστε να ικανοποιεί τους σχεδιαστικούς περιορισμούς και να ελαχιστοποιεί την αντικειμενική συνάρτηση.*

Στην περίπτωση των αρχιτεκτονικών αδρομερούς επαναδιάταξης, η σύνθεση αρχιτεκτονικής αναφέρεται στην απεικόνιση της αλγοριθμικής περιγραφής πάνω στα στοιχεία της επαναδιατάξιμου κυκλώματος. Τυπικά, οι αδρομερείς επαναδιατάξιμες αρχιτεκτονικές [33], αποτελούνται από δομικές μονάδες υλικού που είναι παρόμοιες με τις δομικές μονάδες που χειρίζονται οι αλγόριθμοι σύνθεσης αρχιτεκτονικής ASIC. Σε αυτή την περίπτωση, η υιοθέτηση των υπαρχόντων αλγορίθμων αρχιτεκτονικής σύνθεσης μπορεί να επιτευχθεί με ελάχιστες ή ακόμα καμία τροποποίηση. Αυτό δεν ισχύει για τα επαναδιατάξιμα αρχιτεκτονικά πρότυπα υλικού που προτείνονται στη παρούσα εργασία. Αυτό οφείλεται σε δύο λόγους. Ο πρώτος λόγος έγκειται στη διαφορετικότητα των μοντέλων των επαναδιατάξιμων δομικών μονάδων που προκύπτουν από τη συνδυασμένη ενσωμάτωση των τεχνικών διαμοιρασμού υλικού στο επίπεδο του bit και στο επίπεδο των δομικών μονάδων μαζί με τεχνικές αριθμητικής βελτιστοποίησης. Πιο συγκεκριμένα, οι τυπικοί αλγόριθμοι σύνθεσης αρχιτεκτονικής ASIC ή και συμβατικών επαναδιατάξιμων αρχιτεκτονικών προϋποθέτουν ότι οι δομικές μονάδες υλικού περιγράφονται από ένα μοντέλο 2-εισόδων-1-εξόδου. Το μοντέλο αυτό, αν και εφαρμόσιμο για απεικόνιση εφαρμογών στις προτεινόμενες αρχιτεκτονικές, οδηγεί σε υπο-βέλτιστες λύσεις καθώς το υποστηριζόμενο μοντέλο δομικών μονάδων είναι πιο σύνθετο και παρέχει δυνατότητες για πιο αποδοτικές απεικονίσεις. Ο δεύτερος λόγος είναι ότι οι προτεινόμενες επαναδιατάξιμες αρχιτεκτονικές επιτρέπουν το συνδυασμό πολλαπλών μικρο-αρχιτεκτονικών σχημάτων, πιο συγκεκριμένα ένα συνδυασμό οριζόντιου και κάθετου παραλληλισμού και υπολογιστικής αλυσίδωσης λειτουργιών, τα οποία πρέπει να αξιοποιηθούν κατά τη διάρκεια της σύνθεσης αρχιτεκτονικής για σχεδίαση ακόμη πιο βελτιωμένων λύσεων.

Στην παρούσα διδακτορική διατριβή, καθένα από τα προτεινόμενα επαναδιατάξιμα αρχιτεκτονικά πρότυπα συνοδεύεται από μια εξειδικευμένη μεθοδολογία αρχιτεκτονικής σύνθεσης που επιτρέπει την αποδοτική απεικόνιση αλγοριθμικών περιγραφών. Με τον όρο εξειδικευμένη, αναφερόμαστε στην προσπάθεια για εξειδίκευση γνωστών και αποδοτικών αλγορίθμων σύνθεσης υλικού οι οποίοι τροποποιήθηκαν κατάλληλα ώστε να προσαρμοστούν στα αρχιτεκτονικά μοντέλα. Ωστόσο, νέοι αλγόριθμοι σύνθεσης και τεχνικές μετασχηματισμού κώδικα προτάθηκαν, όπου αυτό θεωρήθηκε κρίσιμο για τη διασφάλιση της υψηλής ποιότητας των παρεχόμενων σχεδιαστικών λύσεων. Εν συντομία, οι προτεινόμενοι μετασχηματισμοί κώδικα επιτρέπουν την απεικόνιση της αλγοριθμικής περιγραφής σε ένα ενδιάμεσο μοντέλο αναπαράστασης το οποίο περιλαμβάνει χαρακτηριστικά των προτεινόμενων αρχιτεκτονικών. Με τον τρόπο επιτρέπουμε τη χρήση γνωστών αλγορίθμων βελτιστοποίησης (που αρχικά αναπτύχτηκαν για σύνθεση ASIC) απαλλαγμένων από τους περιορισμούς των τυπικών μοντέλων υλικού που χρησιμοποιούνται στην ASIC σχεδίαση. Επιπροσθέτως, προ-

τείνεται ένας νέος αλγόριθμος για την ανάθεση των χρονο-προγραμματισμένων λειτουργιών σε επαναδιατάξιμες δομικές μονάδες ώστε να μεγιστοποιείται η εκμετάλλευση του κάθετου παραλληλισμού σε επαναδιατάξιμες αρχιτεκτονικές που επιτρέπουν το συνδυασμό οριζόντιου και κάθετου παραλληλισμού και υπολογιστικής αλυσίδωσης λειτουργιών. Όλοι οι προτεινόμενοι αλγόριθμοι (καινούργιοι ή εξειδικευμένες εκδόσεις υπαρχόντων αλγορίθμων) δεν αναπτύχθηκαν απομονωμένα αλλά σχεδιάστηκαν με στόχο την αποδοτική σύνθεση της συνολικής επαναδιατάξιμης αρχιτεκτονικής. Για παράδειγμα, στο επαναδιατάξιμο αρχιτεκτονικό πρότυπο που επιτρέπει το συνδυασμό οριζόντιου και κάθετου παραλληλισμού και υπολογιστικής αλυσίδωσης λειτουργιών, η διαχείριση της υπολογιστικής αλυσίδωσης λειτουργιών πραγματοποιείται μέσω κατάλληλων μετασχηματισμών κώδικα, η διαχείριση του οριζόντιου παραλληλισμού πραγματοποιείται μέσω εξειδίκευσης γνωστών αλγορίθμων χρονο-προγραμματισμού ενώ η διαχείριση του κάθετου παραλληλισμού επιτυγχάνεται μέσω ενός νέου αλγόριθμου ανάθεσης λειτουργιών σε δομικές μονάδες.

0.3. Προσδιορισμός Ανοικτών Προβλημάτων

Στις προηγούμενες ενότητες παρουσιάστηκε το θεωρητικό υπόβαθρο στο οποίο κινήθηκε η παρούσα διδακτορική διατριβή. Η ενότητα αυτή συνοψίζει τα ανοικτά σχεδιαστικά προβλήματα που αντιμετωπίστηκαν στα πλαίσια της παρούσας διατριβής.

- **Πρόβλημα 1.** Υπάρχει έλλειψη αποτελεσματικών προσεγγίσεων σχετικά με μεθοδολογίες εξειδίκευσης για δυναμική διαχείριση μνήμης πολύ-νηματικών εφαρμογών. Οι υπάρχουσες προσεγγίσεις στηρίζονται σε έναν περιορισμένο χώρο παραμέτρων θεωρώντας κάθε νήμα εκτέλεσης ως μία ξεχωριστή εφαρμογή με αποτέλεσμα να αποτυγχάνουν στη μοντελοποίηση και αξιολόγηση των επιδράσεων που έχει ένα πολύ νηματικό περιβάλλον πάνω στη δυναμική διαχείριση μνήμης. Αυτό οδηγεί σε ένα μεγάλο ανεξερεύνητο τμήμα του χώρου παραμέτρων που επίσης οδηγεί σε απώλεια εξερεύνησης των πραγματικών Pareto βέλτιστων λύσεων.

- **Πρόβλημα 2.** Υπάρχει έλλειψη αποτελεσματικών προσεγγίσεων εξερεύνησης στο επίπεδο της σύνθεσης αρχιτεκτονικής υλικού καθώς, δεν λαμβάνεται υπόψη η συνδυασμένη επίδραση που έχουν οι σχεδιαστικές αποφάσεις από τον αλγοριθμικό και τον αρχιτεκτονικό χώρο παραμέτρων πάνω στην τελική υλοποίηση του κυκλώματος. Ένα σημαντικό μέρος του χώρου λύσεων μένει αποκλεισμένο από το χώρο αναζήτησης με αποτέλεσμα να μην επιτυγχάνεται η σύγκλιση προς το πραγματικό Pareto σύνορο.

- **Πρόβλημα 3.** Οι υπάρχουσες αρχιτεκτονικές συνεπεξεργαστών υλικού αδρομερούς επαναδιάταξης λαμβάνουν υπόψη το μικρο-αρχιτεκτονικό διαμοιρασμό του υλικού μεταξύ των εκχωρημένων δομικών μονάδων αποκλείοντας το διαμοιρασμό πιο βασικών κυκλωματικών δομών που υπάρχουν στο επίπεδο του κυκλώματος. Αυτό οδηγεί σε αυξημένη επιφάνεια υλικού για την κυκλωματική υλοποίηση επαναδιατάξιμων συνεπεξεργαστών υλικού με αποτέλεσμα την αύξηση του κόστους κατασκευής του ολοκληρωμένου.

- **Πρόβλημα 4.** Οι υπάρχουσες αρχιτεκτονικές συνεπεξεργαστών υλικού αδρο-

μερούς επαναδιάταξης δεν λαμβάνουν υπόψη τη χρήση αριθμητικών βελτιστοποιήσεων στις μικρο-αρχιτεκτονικές προδιαγραφές. Αυτό περιορίζει τις επαναδιατάξιμες αρχιτεκτονικές σε μικρότερες συχνότητες λειτουργίας με αποτέλεσμα να υπολείπονται σε ταχύτητα έναντι των μη-ευέλικτων ASIC λύσεων.

- **Πρόβλημα 5.** Οι αυστηρές απαιτήσεις για μεγάλη παραγωγικότητα και σύντομους χρόνους εισαγωγής του προϊόντος στην αγορά (time-to-market) επιβάλλουν περιορισμούς στην υιοθέτηση καινοτόμων αρχιτεκτονικών πρότυπων με πιο σύνθετα μοντέλα προγραμματισμού. Συνεπώς, βελτιστοποιημένες σχεδιαστικές λύσεις (π.χ. οι προτεινόμενες αρχιτεκτονικές αδρομερούς επαναδιάταξης) συχνά αποκλείονται λόγω της πολυπλοκότητας του προγραμματισμού τους.

Στις ακόλουθες ενότητες παρουσιάζονται οι λύσεις που προτείνονται στα πλαίσια της παρούσας διδακτορικής διατριβής για την επιτυχή αντιμετώπιση των παραπάνω προβλημάτων. Η ενότητα 0.4 αναφέρεται στο **Πρόβλημα 1**, παρουσιάζοντας ένα πλαίσιο συστηματικής εξερεύνησης για τη σχεδίαση εξειδικευμένων διαχειριστών δυναμικής μνήμης πολύ-νηματικών εφαρμογών. Η ενότητα 0.5 παρουσιάζει τις μεθοδολογίες εξερεύνησης που λαμβάνουν υπόψη τη συνδυασμένη επίδραση των αλγοριθμικών και αρχιτεκτονικών παραμέτρων σχεδίασης για αποδοτική εξερεύνηση αρχιτεκτονικής σύνθεσης συνεπεξεργαστών υλικού (**Πρόβλημα 2**). Οι ενότητες 0.6 και 0.7 παρουσιάζουν τις καινοτόμες αρχιτεκτονικές λύσεις για συνεπεξεργαστές υλικού αδρομερούς επαναδιάταξης σε συνδυασμό με τις αντίστοιχες μεθοδολογίες αρχιτεκτονικής σύνθεσης, που προτείνονται για την αντιμετώπιση των **Προβλημάτων 3-5**.

0.4. Συστηματική Μεθοδολογία Εξερεύνησης για Εξειδικευμένους Διαχειριστές Δυναμικής Μνήμης Πολύ-Νηματικών Εφαρμογών

Οι πολύ-νηματικές (multi-threaded) εφαρμογές προβλέπεται να κυριαρχήσουν την επόμενη γενιά ενσωματωμένων συστημάτων, καθώς ήδη παρατηρείται μια μετακίνηση των αρχιτεκτονικών προτύπων προς τις πολύ-πύρηνες πλατφόρμες υλικού, οι οποίες προσφέρουν συνδυασμό παραλληλίας σε επίπεδο εντολών και νήματος. Οι εφαρμογές αυτές θα πρέπει να αντιμετωπίσουν αποτελεσματικά την αυξημένη δυναμικότητα που εισάγουν τα πολλαπλά σενάρια χρήσης και η αυξημένη αλληλεπίδραση του συστήματος με το περιβάλλον. Ο αυξημένος δυναμισμός οδηγεί σε απροσδόκητες διακυμάνσεις στις απαιτήσεις μνήμης (ίχνος μνήμης, footprint) άγνωστες κατά το χρόνο σχεδίασης. Η ανάπτυξη δυναμικών εφαρμογών χρησιμοποιώντας τις απαισιόδοξες εκτιμήσεις (worst-case) για τη διαχείριση της μνήμης με στατικό τρόπο, επιβάλει αυξημένο κόστος τόσο στο μέγεθος της μνήμης όσο και στην κατανάλωση ενέργειας. Προκειμένου να αποφευχθούν τέτοιου είδους υπερεκτιμήσεις, οι προγραμματιστές καλούνται να αξιοποιήσουν λύσεις δυναμικής διαχείρισης της μνήμης [34].

Η δυναμική διαχείριση μνήμης (Dynamic Memory Management, DMM) αποτελεί συχνά περιοριστικό παράγοντα για την επίδοση και την επεκτασιμότητα των πολύ-νηματικών εφαρμογών [20]. Επιπλέον, επηρεάζει σημαντικά την κατανάλωση ενέργειας και το μέγεθος μνήμης του συνολικού συστήματος [35]. Ο δυναμικός διαχειριστής μνήμης αποτελεί εκείνη τη συνιστώσα λογισμικού που είναι υπεύθυνη για την οργάνωση των

δυναμικών δεδομένων στη μνήμη και την εξυπηρέτηση των αιτήσεων για δυναμική δέσμευση και αποδέσμευση μνήμης στην εκτελούμενη διεργασία. Στην περίπτωση που η εφαρμογή αιτείται δέσμευση μνήμης κατά τη διάρκεια εκτέλεσης για την αποθήκευση ενός αντικειμένου (π.χ. με χρήση των συναρτήσεων malloc/new στη C/C++), ο δυναμικός διαχειριστής επιστρέφει στην εφαρμογή τον δείκτη που αναφέρεται στη διεύθυνση της μνήμης που εκχωρήθηκε. Σε περίπτωση αίτησης για αποδέσμευση μνήμης (π.χ. με χρήση των συναρτήσεων free/delete στη C/C++), ο δυναμικός διαχειριστής μνήμης επιστρέφει μια Boolean τιμή ανάλογα με την επιτυχία ή την αποτυχία της διαδικασίας αποδέσμευσης.

Οι υπάρχουσες ερευνητικές προσπάθειες επικεντρώνονται στην ανάπτυξη μηχανισμών για γενικού σκοπού δυναμική διαχείριση μνήμης σε μονο-πύρηνα [34], [36] ή πολύ-πύρηνα συστήματα [37], [38], [39], [21]. Για τα ενσωματωμένα συστήματα, η εγγενής γενικότητα των παραπάνω προσεγγίσεων οδηγεί στην υιοθέτηση υποβέλτιστων σχεδιαστικών λύσεων του δυναμικού διαχειριστή μνήμης, καθώς δεν επιτυγχάνεται η εξειδίκευση του διαχειριστή στο συγκεκριμένο σύνολο εφαρμογών του συστήματος. Πρόσφατα, η εξειδικευμένη δυναμική διαχείριση μνήμης έχει προταθεί ως μια αποτελεσματική προσέγγιση για τη βελτίωση των επιδόσεων της εφαρμογής [40] ή τη μείωση του μεγέθους της μνήμης [30]. Ωστόσο, μεθοδολογίες εξειδίκευσης των δυναμικών διαχειριστών μνήμης έχουν παρουσιαστεί μόνο για μονονηματικές εφαρμογές. Επομένως, διαπιστώνεται ένα ερευνητικό και βιβλιογραφικό κενό αναφορικά με την ύπαρξη μεθοδολογιών εξειδίκευσης της δυναμικής διαχείρισης μνήμης που συναντάται σε πολύ-νηματικές εφαρμογές.

Η συγκεκριμένη ενότητα παρουσιάζει τη σχεδιαστική προσέγγιση που προτείνεται στην παρούσα διατριβή σχετικά με την ανάπτυξη αποδοτικών μεθοδολογιών εξερεύνησης για την αυτοματοποιημένη παραγωγή εξειδικευμένων δυναμικών διαχειριστών μνήμης πολύ-νηματικών εφαρμογών. Η κύρια συνεισφορά της διατριβής στο συγκεκριμένο πρόβλημα σχεδίασης μπορεί να συνοψιστεί στα ακόλουθα, (1) Ορίζεται και μοντελοποιείται ο χώρος σχεδίασης για εξειδικευμένη δυναμική διαχείριση μνήμης σε πολύ-νηματικές εφαρμογές. (2) Προτείνεται μια συστηματική μεθοδολογία για αποτελεσματική εξερεύνηση και διάσχιση των αποφάσεων του χώρου σχεδίασης. (3) Αναπτύσσονται κατάλληλα εργαλεία για αυτοματοποίηση της πολύ-κριτηριακής εξερεύνησης και παραγωγής του πηγαίου κώδικα των εξειδικευμένων δυναμικών διαχειριστών μνήμης.

0.4.1. Χώρος Αποφάσεων Δυναμικής Διαχείρισης Μνήμης Πολύ-Νηματικών Εφαρμογών

Ο προτεινόμενος χώρος αποφάσεων, που λειτουργεί ως χώρος παραμέτρων του συγκεκριμένου προβλήματος εξερεύνησης, καθώς και οι μεταξύ τους αλληλεξαρτήσεις απεικονίζονται στο Σχ. 0.5. Η λεπτομερής ανάλυση της σημασιολογίας κάθε σχεδιαστικής απόφασης καθώς και της κάθε εξάρτησης δίνεται στο κυρίως κείμενο της διδακτορικής διατριβής.

Ο συγκεκριμένος χώρος αποφάσεων είναι πλήρης για αρχιτεκτονικές συστημάτων με διαμοιραζόμενη οργάνωση μνήμης, υπό την έννοια ότι κάθε γνωστός δυναμικός διαχειριστής μνήμης μπορεί να μοντελοποιηθεί βάσει των απεικονιζόμενων σχεδιαστικών αποφάσεων. Με βάση μια παρόμοια ανάλυση, οι Atienza και άλλοι [30] έχουν

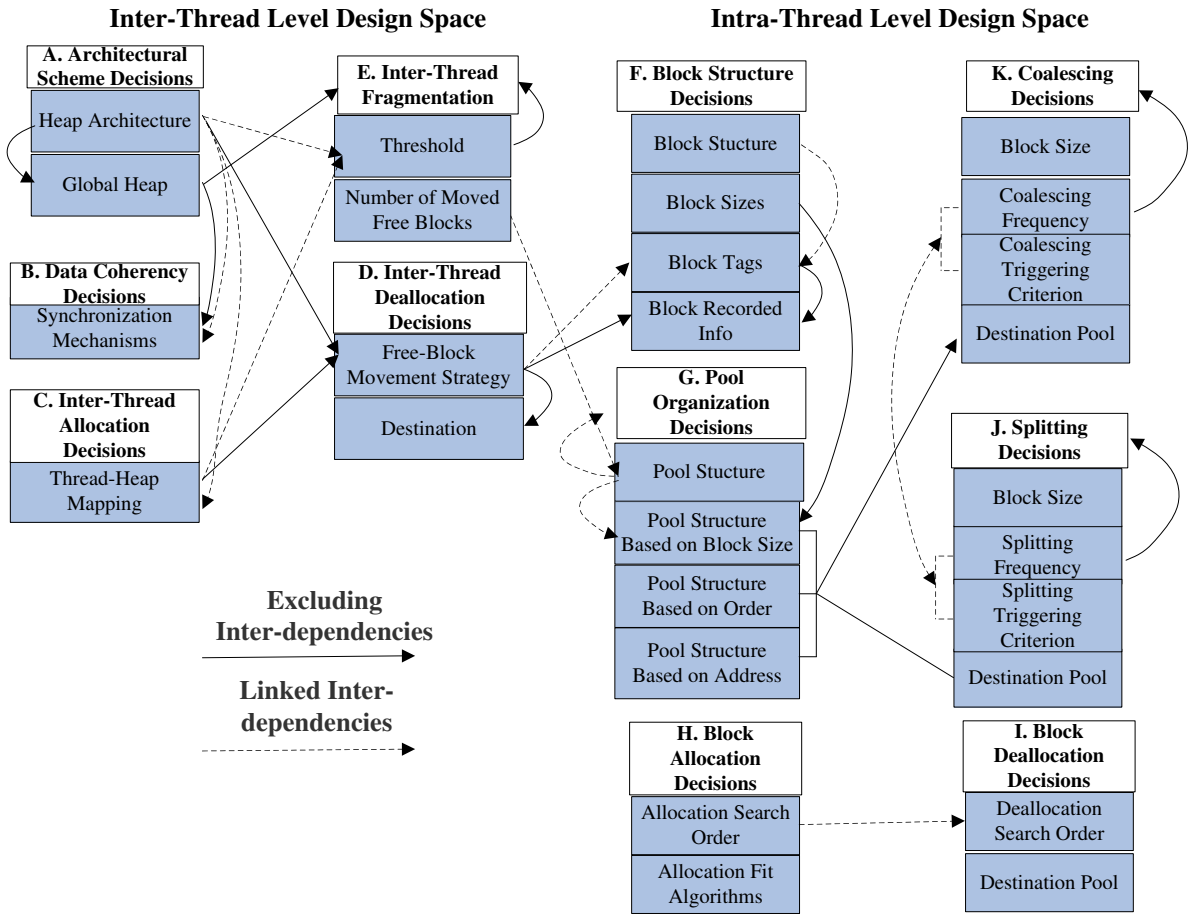


Figure 0.5.: Χώρος αποφάσεων και οι αντίστοιχες αλληλεξαρτήσεις για δυναμική διαχείριση μνήμης πολυ-νηματικών εφαρμογών .

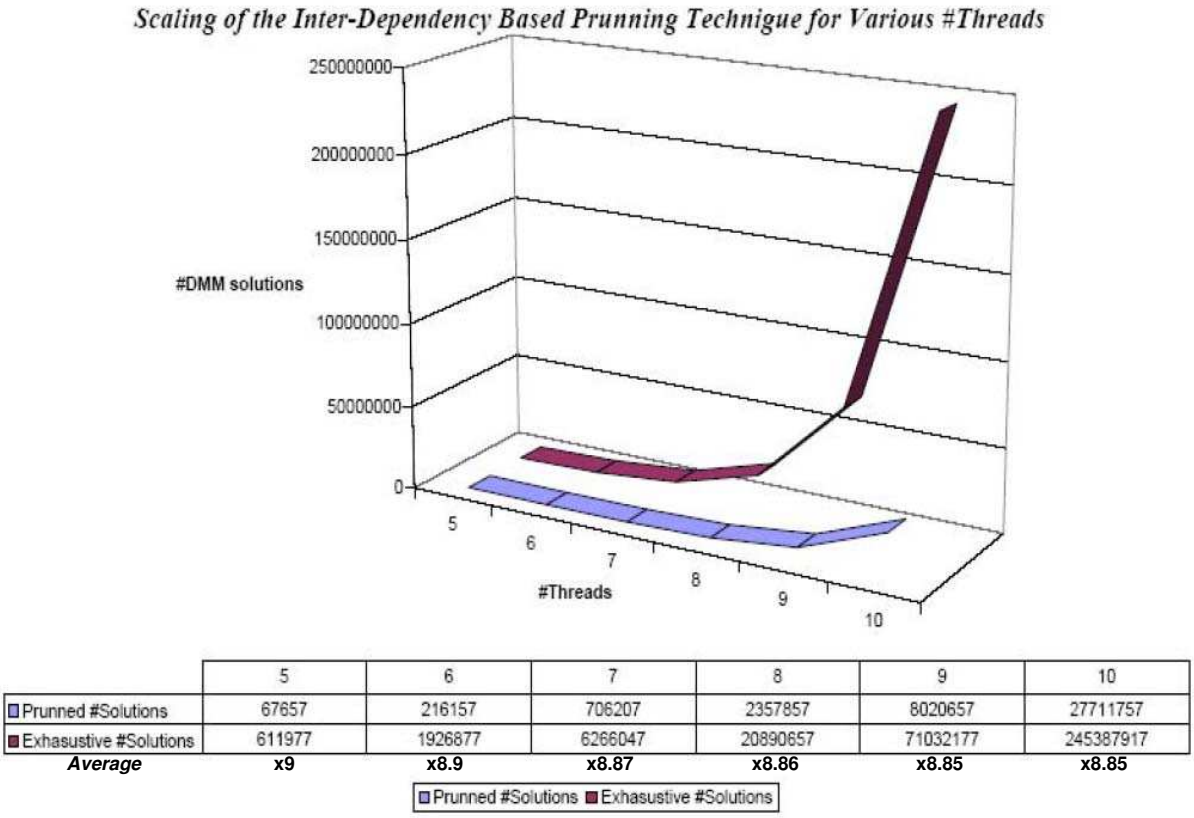
παρουσιάσει μια ταξινόμηση των σχεδιαστικών αποφάσεων που καλύπτουν το χώρο σχεδίασης δυναμικής διαχείρισης μνήμης για μονο-νηματικές εφαρμογές. Ο χώρος αποφάσεων, που προτείνεται στη παρούσα διατριβή, επεκτείνει το χώρο σχεδίασης μονο-νηματικών εφαρμογών με παραμέτρους που μοντελοποιούν τις σχεδιαστικές αποφάσεις που αφορούν στη δυναμική διαχείριση πολύ-νηματικών εφαρμογών. Πιο συγκεκριμένα αποτελείται από δύο υπο-χώρους αποφάσεων, το χώρο δια-νηματικών σχεδιαστικών αποφάσεων (Inter-Heap) και το χώρο ενδο-νηματικών σχεδιαστικών αποφάσεων (Intra-Heap). Οι όροι δια-νηματικός και ενδο-νηματικός χώρος σχεδίασης επιλέχθηκαν για την ποιοτική κατανόηση του διαχωρισμού των αποφάσεων, καθώς δεν περιγράφουν πλήρως το πεδίο εφαρμογής του κάθε σχεδιαστικού υπο-χώρου. Πιο συγκεκριμένα, οι σχεδιαστικές αποφάσεις αφορούν στον τρόπο διαχείρισης του χώρου διευθύνσεων που εκχωρείται για την αποθήκευση των δυναμικών δεδομένων. Ο χώρος αυτός διευθύνσεων ονομάζεται Σωρός (Heap). Δυο η περισσότερα νήματα εκτέλεσης δύναται να μοιράζονται ένα κοινό χώρο διευθύνσεων για τη δέσμευση η αποδέσμευση των δυναμικών δεδομένων. Επομένως, ο όρος σωρός έχει διττή έννοια καθώς σε μια περίπτωση μπορεί να αναφέρεται σε ολόκληρο το διαθέσιμο χώρο διευθύνσεων (περίπτωση δια-νηματικού χώρου αποφάσεων), ενώ σε μια άλλη μπορεί να αναφέρεται σε κάποιο υπο-χώρο διευθύνσεων του συνολικού σωρού, ο οποίος

αναλαμβάνει την αποθήκευση δυναμικών δεδομένων από συγκεκριμένα νήματα εφαρμογής (περίπτωση ενδο-νηματικού χώρου αποφάσεων). Σε κάθε περίπτωση, οι σχεδιαστικές αποφάσεις που περιλαμβάνονται στους δυο υπο-χώρους αφορούν στη διαχείριση αυτών των χώρων διευθύνσεων και όχι αυτού κάθε αυτού του νήματος εκτέλεσης. Η λογική ένωση αυτών των δύο σχεδιαστικών υπο-χώρων παράγει το συνολικό χώρο παραμέτρων.

Ο δια-νηματικός χώρος παραμέτρων περιλαμβάνει σχεδιαστικές αποφάσεις σχετικά με τη δυναμική διαχείριση μνήμης που δια-μοιράζονται μεταξύ των νημάτων εκτέλεσης. Πιο συγκεκριμένα, οι ακόλουθες κατηγορίες αποφάσεων περιλαμβάνονται: (1) Σχεδιαστικές αποφάσεις που αφορούν στη γενική αρχιτεκτονική του σωρού (Κατηγορία A). (2) Σχεδιαστικές αποφάσεις που αφορούν στη συνοχή των δυναμικών δεδομένων (Κατηγορία B). (3) Σχεδιαστικές αποφάσεις που αφορούν στη συνάρτηση ανάθεσης νημάτων εκτέλεσης σε σωρούς (Κατηγορία C). (4) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς μείωσης του κατακερματισμού μεταξύ διαφορετικών σωρών (Κατηγορία D). (5) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς αποδέσμευσης μπλοκ μνήμης από νήματα εκτέλεσης που έχουν ανατεθεί σε διαφορετικούς σωρούς (Κατηγορία E).

Ο ενδο-νηματικός χώρος παραμέτρων περιλαμβάνει τις σχεδιαστικές αποφάσεις που αφορούν στην εσωτερική δομή του κάθε εκχωρημένου σωρού. Πιο συγκεκριμένα, οι ακόλουθες κατηγορίες αποφάσεων περιλαμβάνονται: (1) Σχεδιαστικές αποφάσεις που αφορούν στην οργάνωση των μπλοκ μνήμης εντός ενός συγκεκριμένου σωρού (Κατηγορία F). (2) Σχεδιαστικές αποφάσεις που αφορούν στην οργάνωση των δεξαμενών μνήμης εντός ενός συγκεκριμένου σωρού (Κατηγορία G). (3) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς δέσμευσης ελεύθερου μπλοκ μνήμης (Κατηγορία H). (4) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς αποδέσμευσης μπλοκ μνήμης (Κατηγορία I). (5) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς διαχωρισμού μεγάλων μπλοκ μνήμης σε μικρότερα (Κατηγορία J). (6) Σχεδιαστικές αποφάσεις που αφορούν στους μηχανισμούς ενοποίησης μικρότερων μπλοκ μνήμης σε μεγαλύτερα (Κατηγορία K).

Στο Σχ. 0.5έχουν απεικονιστεί και το σύνολο των αλληλεξαρτήσεων οι οποίες ενυπάρχουν και διαδίδονται μεταξύ των σχεδιαστικών κατηγοριών ή μεταξύ των σχεδιαστικών αποφάσεων. Οι εξαρτήσεις αυτές ταξινομούνται σε δύο σύνολα. (1) Εξαρτήσεις αποκλεισμού (βέλη εξαρτήσεων με συνεχείς γραμμές), οι οποίες μοντελοποιούν την κατάσταση όπου η επιλογή μιας απόφασης (κεφαλή βέλους) μπλοκάρει την ενδεχόμενη επιλογή κάποιας άλλης απόφασης (τέλος βέλους) λόγω έλλειψης σημασιολογικής ή δομικής συνοχής. (2) Εξαρτήσεις σύνδεσης (βέλη εξαρτήσεων με διακεκομμένες γραμμές), οι οποίες μοντελοποιούν την κατάσταση όπου η επιλογή μιας απόφασης επηρεάζει την ενδεχόμενη επιλογή κάποιας άλλης απόφασης χωρίς όμως η επίδραση αυτή να μπορεί να μοντελοποιηθεί ή να εκτιμηθεί ποιοτικά για όλες τις περιπτώσεις. Οι εξαρτήσεις αποκλεισμού μπορούν να χρησιμοποιηθούν σαν κανόνες πρόωρης εξάλειψης των μη-σημασιολογικά ή μη-δομικά έγκυρων λύσεων, ώστε να μειωθεί σημαντικά ο χώρος αναζήτησης των προς διερεύνηση λύσεων. Η μείωση του χώρου αναζήτησης δεν επηρεάζει την ποιότητα των λύσεων, δεδομένου ότι οι εξαρτήσεις αποκλεισμού δεν αποτελούν ευριστική διαδικασία εξάλειψης λύσεων, καθώς μόνο οι μη έγκυρες λύσεις εξαιρούνται της εξερεύνησης. Το Σχ. 0.6 απεικονίζει τα αποτελέσματα μιας συγκριτικής μελέτης σχετικά με το μέγεθος του χώρου λύσεων (που προκύπτει από τις αποφάσεις μόνο το δια-νηματικού χώρου αποφάσεων) για αυξανόμενο αριθμό νημάτων εκτέλεσης, στις περιπτώσεις που οι



	5	6	7	8	9	10
Prunned #Solutions	67657	216157	706207	2357857	8020657	27711757
Exhasustive #Solutions	611977	1926877	6266047	20890657	71032177	245387917
Average	x9	x8.9	x8.87	x8.86	x8.85	x8.85

Prunned #Solutions Exhasustive #Solutions

Figure 0.6.: Συγκριτική μελέτη σχετικά με την επίδραση των αλληλεξαρτήσεων στο μέγεθος του χώρου λύσεων.

εξαρτήσεις αποκλεισμού λαμβάνονται και δεν λαμβάνονται υπόψη αντίστοιχα. Όταν οι εξαρτήσεις αποκλεισμού δεν λαμβάνονται υπόψη, ο χώρος λύσεων αυξάνεται εκθετικά με τον αριθμό των νημάτων της εφαρμογής. Στην περίπτωση που οι εξαρτήσεις αποκλεισμού συμπεριλαμβάνονται σαν κανόνες πρόωρης εξάλειψης λύσεων η αύξηση του χώρου λύσεων προσεγγίζει μια πιο γραμμική συμπεριφορά, παρέχοντας μια επιτάχυνση της τάξης του $9\times$.

0.4.2. Μεθοδολογία Εξερεύνησης

Κάθε πιθανή σχεδιαστική λύση δυναμικής διαχείρισης μνήμης μπορεί να μοντελοποιηθεί ως ένα διάνυσμα αποφάσεων του χώρου σχεδίασης που περιγράφηκε στην προηγούμενη ενότητα. Επιπλέον, ένα ξεχωριστό υπο-διάνυσμα αποφάσεων πρέπει να ληφθεί υπόψη για κάθε περίπτωση ανάθεση νήματος σε σωρό, σχετικά με τις αποφάσεις που αφορούν στο ενδο-νηματικό χώρο σχεδίασης Ωστόσο, η εξαντλητική διερεύνηση όλων των πιθανών συνδυασμών αποφάσεων απαιτεί την πλήρη απαρίθμηση (παραγοντικό του αριθμού των παραμέτρων) και αξιολόγηση όλων των δυνατών διανυσμάτων αποφάσεων, με αποτέλεσμα το μέγεθος του χώρου λύσεων να εκρήγνυται και να παύει να είναι υπολογιστικά διαχειρίσιμο. Προκειμένου να καταστεί διαχειρίσιμη

η πολυπλοκότητα αυτή, προτείνουμε μια μεθοδολογία εξερεύνησης που βασίζεται στον ορθογώνιο διαχωρισμό του χώρου σχεδίασης, ακολουθώντας το παράδειγμα του [41]. Παρατηρούμε πως ο δια-νηματικός υπο-χώρος και ο ενδο-νηματικός υποχώρος αποφάσεων είναι ορθογώνιοι μεταξύ τους με ιεραρχική διάδοση περιορισμών. Επομένως, αντί για μια ενοποιημένη εξερεύνηση ολόκληρου του χώρου σχεδιασμού, η προτεινόμενη προσέγγιση διαχωρίζει την εξερεύνηση σε δύο ορθογώνια προβλήματα με διάδοση περιορισμών. Το πρώτο πρόβλημα αφορά στην εξερεύνηση των σχεδιαστικών αποφάσεων του δια-νηματικού χώρου σχεδίασης, ενώ το δεύτερο αφορά στην εξερεύνηση των σχεδιαστικών αποφάσεων του ενδο-νηματικού χώρου σχεδίασης δεδομένων των δια-νηματικών αποφάσεων. Πιο συγκεκριμένα, η εξερεύνηση στο επίπεδο του δια-νηματικού χώρου σχεδίασης εκτελείται αρχικά παράγοντας ένα πρώτο Pareto σύνολο λύσεων. Αυτές οι Pareto λύσεις αποτελούν την είσοδο της ενδο-νηματικής εξερεύνησης, η οποία επιτελεί επιπλέον εξερεύνηση μόνο στα πεδία που αφορούν ενδο-νηματικές αποφάσεις διατηρώντας σταθερές τις αποφάσεις που αφορούν τη δια-νηματική διαχείριση.

Η προτεινόμενη μεθοδολογία εξερεύνησης αποτελείται από πέντε βασικά βήματα. Κάθε βήμα έχει υλοποιηθεί σαν ξεχωριστό εργαλείο λογισμικού ώστε να είναι δυνατή η εξειδίκευση του, αναλόγως του πεδίου εφαρμογής. Η συνολική μεθοδολογία υλοποιείται με σύνδεση των επιμέρους εργαλείων και απεικονίζεται στο Σχ. 0.7. Πιο συγκεκριμένα, παρακάτω δίδεται μια σύντομη περιγραφή καθενός από βήματα που αποτελούν τη μεθοδολογία εξερεύνησης:

1. Παραμετρική C++ βιβλιοθήκη σχεδιαστικών αποφάσεων: Η C++ βιβλιοθήκη λογισμικού υλοποίει κάθε μηχανισμό που περιγράφεται από τις αποφάσεις του δια-νηματιού και ενδο-νηματικού χώρου σχεδίασης. Βασίζεται στη τεχνολογία των επιπέδων mixin [42], προκειμένου να καθιστά δυνατή τη παραγωγή κώδικα οποιουδήποτε συνδυασμού αποφάσεων μέσω συναρμολόγησης mixin αντικειμένων. Αποτελεί το βασικό τμήμα της μεθοδολογίας εξερεύνησης, δεδομένου ότι σε αυτή στηρίζεται η αυτοματοποιημένη γέννηση κώδικα των διαφόρων δυναμικών διαχειριστών μνήμης που επιλέγονται από τα εργαλεία λήψης αποφάσεων και σχηματισμού των διανυσμάτων κατά, τη διάρκεια της εξερεύνησης.

2. Εργαλείο παραγωγής δια-νηματικών διανυσμάτων αποφάσεων: Το εργαλείο παραγωγής δια-νηματικών διανυσμάτων αποφάσεων λαμβάνει υπόψη τις εξαρτήσεις αποκλεισμού του δια-νηματικού υπο-χώρου και παράγει αυτόματα τα διανύσματα αποφάσεων που πρέπει να αξιολογηθούν δεδομένων των ορίων των παραμέτρων που δίνονται από τον σχεδιαστή. Σε αυτή τη φάση εξερεύνησης, όλοι οι σωροί μνήμης που αφορούν στις ενδο-νηματικές αποφάσεις θεωρούνται ομοιόμορφοι. Τα διάφορα διανύσματα αποφάσεων εισάγονται στη C++ βιβλιοθήκη και οι αντίστοιχες υλοποιήσεις των δυναμικών διαχειριστών μνήμης παράγονται και συνδέονται με την υπο-εξέταση εφαρμογή. Στη συνέχεια, η εφαρμογή εξομοιώνεται και οι αντίστοιχες μετρικές επίδοσης (π.χ. ίχνος μνήμης, αριθμός προσβάσεων, χρόνος εκτέλεσης κ.τ.λ.) εξάγονται μέσω κατάλληλων τεχνικών σκιαγράφησης κώδικα. Οι λύσεις αξιολογούνται και το Pareto σύνορο προκύπτει προκείμενου προωθηθεί στο εργαλείο εξερεύνησης ενδο-νηματικών διανυσμάτων αποφάσεων.

3. Εργαλείο παραγωγής ενδο-νηματικών διανυσμάτων αποφάσεων: Το εργαλείο παραγωγής ενδο-νηματικών διανυσμάτων αποφάσεων δέχεται σαν είσοδο το δια-νηματικό Pareto σύνορο λύσεων και τα όρια τιμών από το σχεδιαστή που

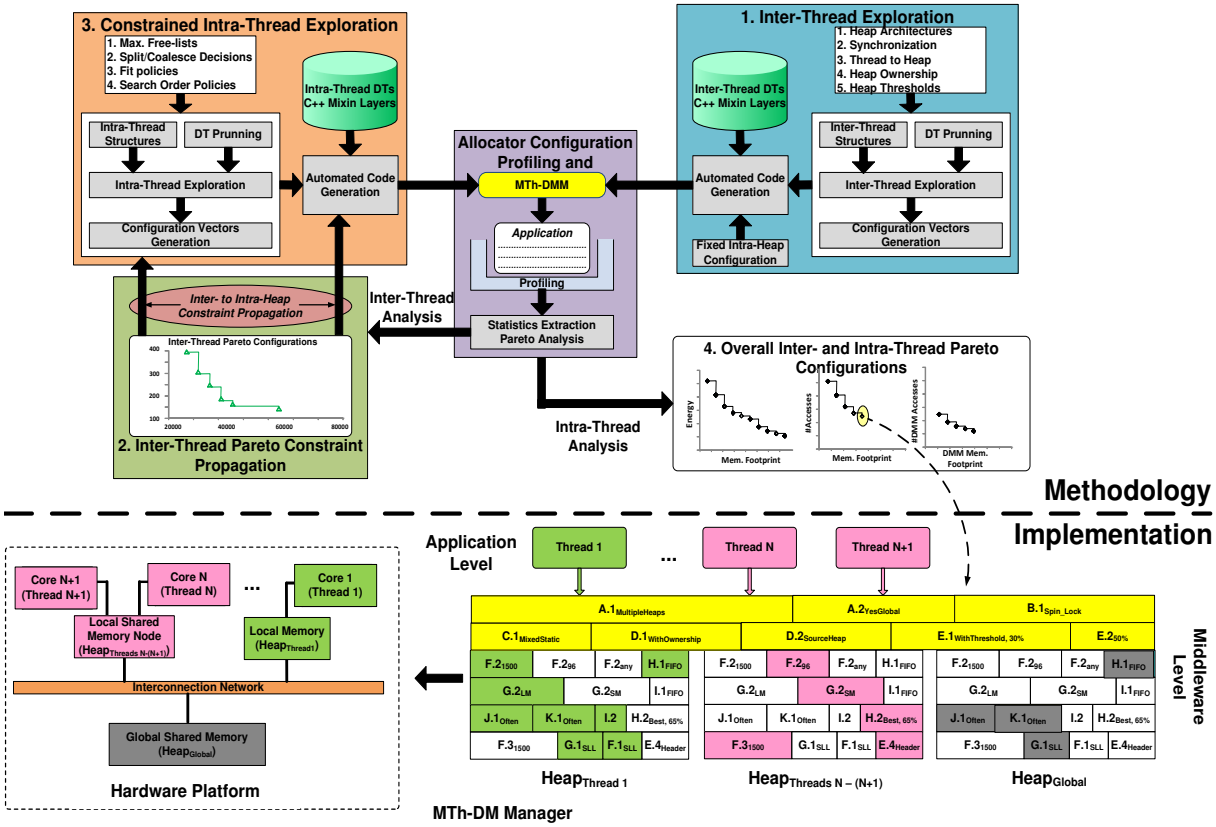


Figure 0.7.: Προτεινόμενη μεθοδολογία και εργαλείο εξερεύνησης .

αφορούν στην ενδο-νηματική εξερεύνηση. Λαμβάνοντας υπόψη τις εξαρτήσεις αποκλεισμού του ενδο-νηματικού υπο-χώρου, παράγει αυτόματα τα ενδο-νηματικά διανύσματα αποφάσεων που πρέπει να αξιολογηθούν και τα συγχωνεύει με τα διανύματα που περιγράφουν τα δια-νηματικά Pareto σημεία. Σε αυτή τη φάση εξερεύνησης, κάθε ενδο-νηματικός σωρός μνήμης μπορεί να έχει διαφορετική μορφοποίηση. Σε περίπτωση που τα όρια του χώρου παραμέτρων που εισάγει ο σχεδιαστής οδηγούν σε πολύ μεγάλο μέγεθος του εξεταζόμενου χώρου λύσεων, το εργαλείο ενδο-νηματικής εξερεύνησης παρέχει δυο ευριστικούς αλγορίθμους για μείωση του ίχνους μνήμης ή μείωση του αριθμού των προσβάσεων. Τα παραγόμενα διανύσματα αποφάσεων εισάγονται στη C++ βιβλιοθήκη και οι αντίστοιχες υλοποιήσεις των δυναμικών διαχειριστών μνήμης παράγονται και συνδέονται με τη υπο-εξέταση εφαρμογή. Στη συνέχεια, η εφαρμογή επανα-εξομοιώνεται και εξάγονται οι αντίστοιχες μετρικές απόδοσης. Οι λύσεις αξιολογούνται και το τελικό Pareto σύνορο παράγεται και επιστρέφεται στο σχεδιαστή.

4. **Εργαλείο εξαγωγής Pareto λύσεων:** Το εργαλείο εξαγωγής Pareto λύσεων δέχεται ως είσοδο τον χαρακτηρισμένο χώρο λύσεων που έχει προκύψει από τα εργαλεία εξερεύνησης καθώς και τα επιθυμητά κριτήρια βελτιστοποίησης και εξάγει τις λύσεις του εξεταζόμενου χώρου.

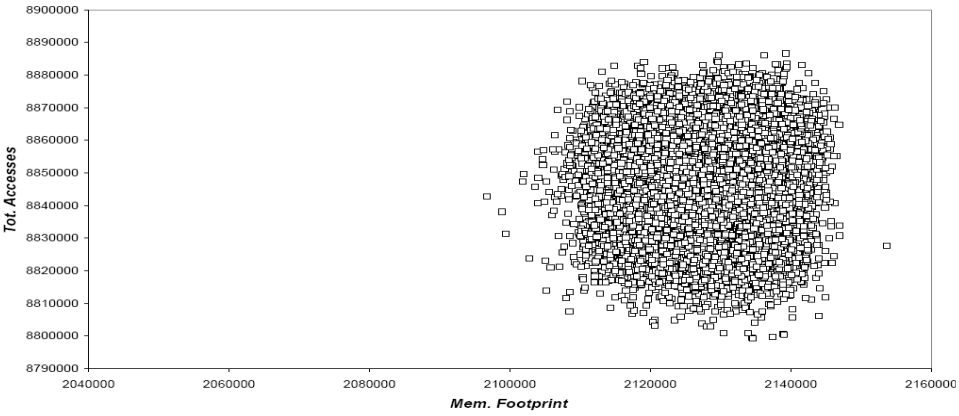


Figure 0.8.: Δια-νηματικός χώρος λύσεων της δικτυακής εφαρμογής .

0.4.3. Ενδεικτικά Αποτελέσματα Πειραματικής Αξιολόγησης

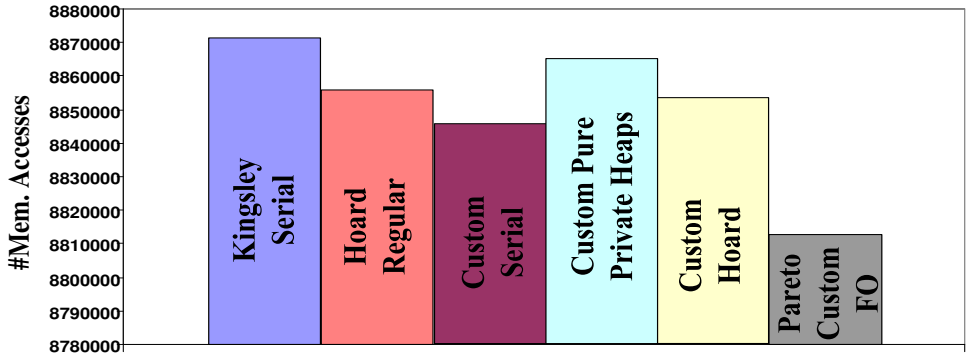
Η ενότητα αυτή παρουσιάζει κάποια ενδεικτικά πειραματικά αποτελέσματα για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης προσέγγισης. Περαιτέρω πειραματικά αποτελέσματα μπορούν να βρεθούν στο κυρίως κείμενο της διατριβής. Εφαρμόσαμε την παραπάνω μεθοδολογία εξερεύνησης σε μια πραγματική πολύ-νηματική δυναμική δικτυακή εφαρμογή. Η εφαρμογή αποτελείται από 5 νήματα που το καθένα ενεργοποιείται από ρεύματα δεδομένων. Κάθε νήμα εκτέλεσης επικοινωνεί ασύγχρονα με τα υπόλοιπα. Η εφαρμογή απεικονίστηκε σε ένα Linux Posix πολύνηματικό περιβάλλον για έναν τετρα-πύρηνο επεξεργαστή Intel Quad Core στα 2,66 GHz. Οι μετρικές αξιολόγησης αφορούν: (1) τον αριθμό προσβάσεων στη μνήμη, (2) το ίχνος μνήμης και (3) το μέγεθος του κώδικα του κάθε δυναμικού διαχειριστή μνήμης.

Το Σχ. 0.8 περιλαμβάνει μια δισδιάστατη απεικόνιση του χώρου λύσεων της εφαρμογής, η οποία προκύπτει από τη δια-νηματική διαδικασία εξερεύνησης με χαρακτηρισμό του κάθε σημείου ως προς το ίχνος μνήμης και τον αριθμό των προσβάσεων. Ο απεικονιζόμενος χώρος λύσεων αποτελείται από 67.655 σημεία. Δεδομένου του μεγάλου εύρους τιμών που παρουσιάζει το εικονιζόμενο σύνολο λύσεων, γίνεται κατανοητή η δυσκολία προσέγγισης του Pareto συνόρου και επομένως η σημαντικότητα της υιοθέτησης μεθοδολογιών αυτοματοποιημένης εξερεύνησης για την αποτελεσματική αντιμετώπιση του προβλήματος σχεδίασης εξειδικευμένων δυναμικών διαχειριστών μνήμης για πολύ-νηματικές εφαρμογές.

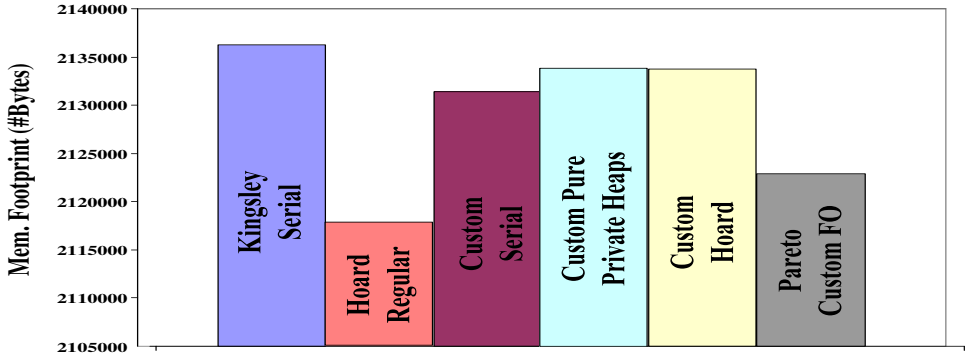
Προκειμένου να αξιολογήσουμε την αποδοτικότητα των λύσεων που παρέχονται από την προτεινόμενη μεθοδολογία, συγκρίνουμε εξειδικευμένους διαχειριστές έναντι των γενικού σκοπού δυναμικούς διαχειριστές μνήμης των Windows XP-Kingsley [20] και Hoard [21], που αποτελεί τεχνολογία αιχμής στους πολύ-νηματικούς διαχειριστές. Θεωρήσαμε τέσσερις παραλλαγές εξειδικευμένων διαχειριστών, η δομή των οποίων δίνεται μέσω των αποφάσεων σχεδίασης που ο καθένας υλοποιεί στο Σχ. 2.11a. Το Σχ. 2.11 δείχνει τα αποτελέσματα της αξιολόγησης. Σχετικά με τη μετρική του αριθμού προσβάσεων στη μνήμη, η Pareto λύση προσφέρει μια μέση μείωση των 46K σε προσπελάσεις μνήμης (Σχ.2.11b). Σε ότι αφορά το ίχνος μνήμης (Σχ. 2.11c), η Pareto λύση είναι η δεύτερη καλύτερη μετά τον Hoard, με μέση μείωση του ίχνους

MTh-DMMs	Inter-Heap Decisions	Brief Description of Intra-Heap Decisions
Kingsley Serial	$A.1_{Serial} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{AllinOne} \rightarrow D.1_{w/o} \rightarrow D.2_{Global} \rightarrow E.1_{No} \rightarrow E.2_{None}$	128 Size Freelists→ SLL organization→ 1 Generic Heap with FirstFit and Coalesce/Split functions
Hoard	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/} \rightarrow D.2_{Source} \rightarrow E.1_{Yes,\ 50\%} \rightarrow E.2_{All}$	Per size bins (max 128 bins)→ SLL organization with FirstFit allocation.
Custom Serial	$A.1_{Serial} \rightarrow A.2_{Yes} \rightarrow B.1_{Mutex} \rightarrow C.1_{AllinOne} \rightarrow D.1_{w/o} \rightarrow D.2_{Global} \rightarrow E.1_{No} \rightarrow E.2_{None}$	1 Generic Heap with DLL organization→ FirstFit and Coalesce/Split functions
Custom Pure Private Heaps	$A.1_{PurePrivateHeaps} \rightarrow A.2_{No} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/o} \rightarrow D.2_{Other} \rightarrow E.1_{No} \rightarrow E.2_{None}$	1 Fixed Size Freelist→ 1 Generic Heap with DLL organization→ FirstFit and Coalesce/Split functions
Custom Hoard	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/} \rightarrow D.2_{Source} \rightarrow E.1_{Yes,\ 70\%} \rightarrow E.2_{All}$	1 Generic Heap with SLL organization→ FirstFit and Coalesce/Split functions
Pareto FO	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{Mutex} \rightarrow C.1_{StaticAssignment} \rightarrow D.1_{w/o} \rightarrow D.2_{Other} \rightarrow E.1_{Yes,\ 90\%} \rightarrow E.2_{50\%}$	3 Fixed Size Freelists→ 1 Generic Heap with SLL organization→ FirstFit and Coalesce/Split functions

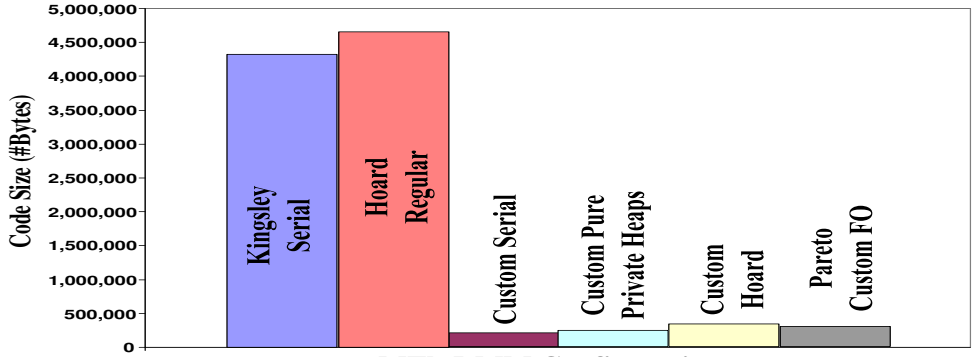
(a) Συνδυασμός Αποφάσεων Εξεταζόμενων Διαχειριστών



(b) Συγκριτική μελέτη αριθμού προσβάσεων στη μνήμη



(c) Συγκριτική μελέτη για το ίχνος μνήμης



(d) Συγκριτική μελέτη για το μέγεθος κώδικα

Figure 0.9.: Συγκριτική μελέτη μεταξύ διαχειριστών δυναμικής μνήμης .

της τάξης των 8.8KBytes. Η αποδοτικότητα του Hoard στο ίχνος μνήμης οφείλεται στην εσωτερική οργάνωση (μεγάλος αριθμός δεξαμενών μνήμης) των σωρών του Hoard. Ωστόσο, το Σχ. 2.11d δείχνει ότι αυτή η σύνθετη οργάνωση του σωρού κάνει τον Hoard τη χειρότερη λύση σε ότι αφορά το μέγεθος κώδικα του διαχειριστή μνήμης. Ο προτεινόμενος Pareto διαχειριστής, όπως και οι υπόλοιποι εξειδικευμένοι διαχειριστές, παρουσιάζουν σημαντικά κέρδη σχεδόν 14× σε σύγκριση με τους γενικού σκοπού δυναμικούς διαχειριστές μνήμης.

0.5. Συνδυασμένη Αλγοριθμική και Αρχιτεκτονική Εξερεύνηση Για Αυτοματοποιημένη Σύνθεση Υλικού

Οι μεθοδολογίες εξερεύνησης κατά τη διάρκεια της αυτοματοποιημένης σύνθεσης αρχιτεκτονικών υλικού, HLS, έχει σαν στόχο τον υπολογισμό εκείνων των σχεδιαστικών λύσεων οι οποίες σχηματίζουν το Pareto σύνορο, δηλαδή τα σχεδιαστικά σημεία που παρέχουν τους βέλτιστους συμβιβασμούς (trade-off) μεταξύ μετρικών αξιολόγησης. Οι μεθοδολογίες αυτοματοποιημένης αρχιτεκτονικής σύνθεσης υλικού βρίσκονται πλέον στην ώριμη φάση τους και μια συνεχώς αναπτυσσόμενη κοινότητα σχεδιαστών φαίνεται να τις υιοθετεί, προκειμένου να αντιμετωπίσει την αυξημένη πολυπλοκότητα σχεδίασης. Ωστόσο, η σχεδίαση από υψηλά επίπεδα αφαίρεσης στις μεθοδολογίες HLS συνοδεύεται με μια αύξηση των παραμέτρων και των αποφάσεων που πρέπει να ληφθούν υπόψη, με αποτέλεσμα οι σχεδιαστές να αντιμετωπίζουν πλέον το πρόβλημα εύρεσης μιας συλλογιστικής η οποία να επιτρέπει την εκτίμηση των παραγόμενων trade-offs και να επικουρεί στη διάσχιση του χώρου σχεδίασης. Επομένως, η ανάπτυξη αποδοτικών μεθοδολογιών και αυτοματοποιημένων εργαλείων εξερεύνησης του χώρου σχεδίασης της αρχιτεκτονικής σύνθεσης υλικού είναι σημαντική για την κοινότητα σχεδίασης υλικού.

Στην παρούσα διδακτορική διατριβή στοχεύουμε στο θεμελιώδες HLS πρόβλημα της αναζήτησης των σημείων που αποτελούν βέλτιστα trade-off μεταξύ των μετρικών βελτιστοποίησης καθυστέρησης λειτουργίας και επιφάνειας υλικού. Τυπικά, η αναζήτηση για την εύρεση των βέλτιστων trade-off σημείων έχει επικεντρωθεί στη μελέτη των διαφόρων αρχιτεκτονικών παραμέτρων, μη διερευνώντας σε βάθος την επίδραση που έχει ο συνδυασμός των αρχιτεκτονικών με τις αντίστοιχες συμπεριφορικές (αλγοριθμικές) παραμέτρους. Δείχνουμε ότι χρησιμοποιώντας τις υπάρχουσες προσεγγίσεις ένα μεγάλο ποσοστό των πραγματικών Pareto λύσεων αποκλείεται από το χώρο εξερεύνησης οδηγώντας τον σχεδιαστή στην υιοθέτηση υπο-βέλτιστων σχεδιαστικών λύσεων. Προκειμένου να αντιμετωπίσουμε την παραπάνω αδυναμία σύγκλισης προς τα καθολικά βέλτιστα, προτείνουμε μια νέα μεθοδολογία εξερεύνησης η οποία χειρίζεται σε ένα κοινό βρόγχο εξερεύνησης τις αλγοριθμικές μαζί με τις αρχιτεκτονικές παραμέτρους. Δεδομένου ότι οι αλγοριθμικές παράμετροι χειρίζονται στο επίπεδο του μεταγλωττιστή (compiler) ονομάζουμε τη νέα μεθοδολογία εξερεύνησης Compiler-In-The-Loop (CompInLoop). Η CompInLoop βασίζεται σε ένα διευρυμένο χώρο παραμέτρων ο οποίος μοντελοποιεί σε ένα ενιαίο χώρο τις αλγοριθμικές και τις αρχιτεκτονικές παραμέτρους που λαμβάνονται υπόψη κατά τη διάρκεια της εξερεύνησης. Η προτεινόμενη προσέγγιση μπορεί και να χαρακτηριστεί ως μεθοδολογία μετα-εξερεύνησης καθώς δεν προτείνει νέους αλγορίθμους για χρονοδρομολόγηση λειτουργιών ή ανάθεσης λειτουργιών σε πόρους υλικού όπως συμβαίνει συχνά στις HLS μεθοδολογίες, αλλά αντί αυτού προτείνει ένα σχήμα αποτελεσματικής δι-

αχείρισης των λύσεων που προκύπτουν από τα υπάρχοντα HLS εργαλεία. Η CompInLoop μεθοδολογία εξερεύνησης εφαρμόζεται στο χώρο λύσεων και βασίζεται στην εισαγωγή δομής στο χώρο αναζήτησης, στο σχηματισμό κατάλληλων άνω οριακών συνθηκών καθώς και σε ένα νέο ευριστικό αλγόριθμο αναζήτησης ο οποίος εκμεταλλεύεται την εισηγμένη δομή του χώρου λύσεων, ώστε να μειώσει περαιτέρω το χώρο αναζήτησης, όταν διαπιστώνει πως αξιολογεί λύσεις σε τοπικά μη-βέλτιστες περιοχές κορεσμού. Η προτεινόμενη CompInLoop μεθοδολογία εξερεύνησης υλοποιήθηκε επεκτείνοντας το εργαλείου αυτοματοποιημένης αρχιτεκτονικής σύνθεσης SPARK [43].

0.5.1. Μετακίνηση Προς Υψηλότερης Ποιότητας Pareto Σημεία

Οι υπάρχουσες μεθοδολογίες εξερεύνησης στο πεδίο της σύνθεσης αρχιτεκτονικής μπορούν να ταξινομηθούν στις ακόλουθες τρεις κατηγορίες, ανάλογα με τον τρόπο που χειρίζονται οι αλγοριθμικοί μετασχηματισμοί στο επίπεδο του μεταγλωττιστή. Το Σχ. 0.10 προσφέρει μια σχηματική απεικόνιση της βασικής δομής της κάθε μεθοδολογίας εξερεύνησης.

- **Εξερεύνηση Υπο-Βοηθούμενη από τον Compiler (Compiler assisted DSE, CompAssisted):** Η μεθοδολογία αυτή χρησιμοποιεί τις διάφορες βελτιστοποιήσεις κώδικα που προσφέρονται από το μεταγλωττιστή προκειμένου να παραχθεί μια βελτιστοποιημένη εσωτερική αναπαράσταση του κώδικα προς σύνθεση και στη συνέχεια διερευνά τις επίδρασεις διαφορετικών σεναρίων εκχώρησης πόρων υλικού επί της εσωτερικής αναπαράστασης. Οι εργασίες [1], [2], [3], [4], [5], αποτελούν χαρακτηριστικά παραδείγματα μεθοδολογιών που εμπίπτουν στη συγκεκριμένη κατηγορία.

- **Εξερεύνηση Υπο-Βοηθούμενη από τον Compiler Σε Συνδυασμό Με Επιλογή Περιόδου Ρολογιού (Compiler assisted with CLK selection DSE, CompAssisted-CLK):** Η μεθοδολογία αυτή επεκτείνει την CompAssisted λαμβάνοντας υπόψη την επίδραση της συχνότητας ρολογιού στην τελική κυκλωματική υλοποίηση π.χ. [6], [7], [8].

- **Εξερεύνηση Οδηγούμενη από τον Compiler (CompDirected):** Η μεθοδολογία αυτή θεωρεί δύο βρόγχους εξερεύνησης, ο πρώτος για εξερεύνηση των μετασχηματισμό στο επίπεδο του compiler και ο δεύτερος για εξερεύνηση των αρχιτεκτονικών παραμέτρων π.χ. [9], [10], [11], [12], [13], [14].

Η προτεινόμενη CompInLoop προσέγγιση διαφοροποιείται σημαντικά από τα προηγούμενα σχήματα εξερεύνησης στο ότι στοχεύει σε έναν πιο γενικό χώρο λύσεων. Πιο συγκεκριμένα, η CompInLoop μεθοδολογία εξερεύνησης όχι μόνο περιλαμβάνει το σύνολο των λύσεων που παράγονται από τα υπάρχοντα σχήματα εξερεύνησης, αλλά ταυτόχρονα επεκτείνει το χώρο αναζήτησης προς περιοχές οι οποίες περιέχουν Pareto λύσεις και οι οποίες αποκλείονταν της εξερεύνησης.

Προκειμένου να γίνουν κατανοητές οι δυνατότητες για μετακίνηση προς πιο αποδοτικά Pareto σύνορα με χρήση της προτεινόμενης CompInLoop μεθοδολογίας, αξιολογήσαμε τους χώρους λύσεων που αντιστοιχούν σε κάθε μια από τις παραπάνω

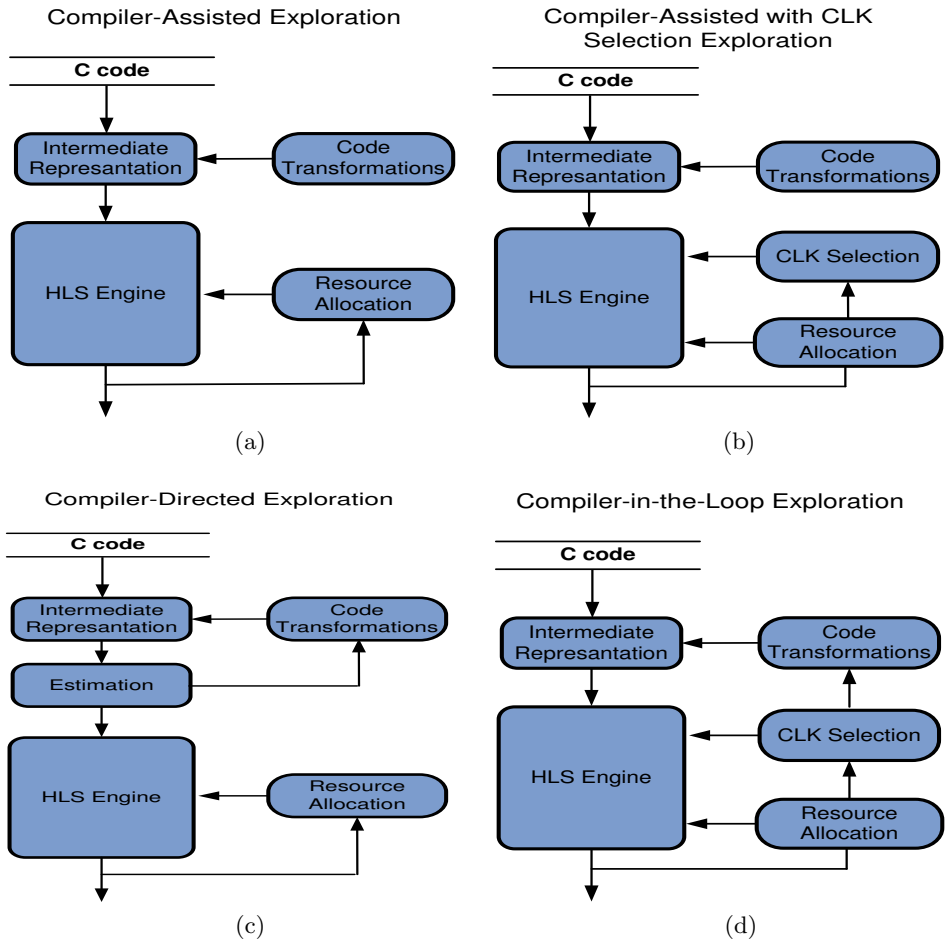


Figure 0.10.: Στρατηγικές εξερεύνησης : (a) Compiler assisted [1], [2], [3], [4], [5]. (b) Compiler assisted with CLK selection [6], [7], [8]. (c) Compiler directed [9], [10], [11], [12], [13], [14]. (d) Προτεινόμενη Compiler-in-the-loop.

μεθοδολογίες εξερεύνησης. Ο μετασχηματισμός συνημίτονου (1-D Discrete Cosine Transform, 1D-DCT)χρησιμοποιήθηκε ως η υποψήφια εφαρμογή προς εξερεύνηση. Θεωρήσαμε εξαντλητική εξερεύνηση του χώρου λύσεων για κάθε μεθοδολογία εξερεύνησης υπό τον περιορισμό η επιφάνεια υλικού της παραγόμενης κυκλωματικής υλοποίησης να μην ξεπερνά τα 65000um^2. Τα αποτελέσματα της συγκριτικής αξιολόγησης απεικονίζονται στο Σχ. 0.11.

Η προσεκτική μελέτη των διάφορων απεικονιζόμενων χώρων λύσεων αποκαλύπτει κάποιες βασικές παρατηρήσεις. Πρώτον, η αποτελεσματικότητα της κάθε μεθοδολογίας εξερεύνησης, σχετικά με την ποιότητα του εξαγόμενου Pareto συνόρου, εξαρτάται σε μεγάλο βαθμό από τον ‘‘ορατό’’ χώρο λύσεων (ο οποίος συσχετίζεται με τον αριθμό και το είδος των παραμέτρων εξερεύνησης). Λόγω των διαφορετικών ‘‘ορατών’’ χώρων λύσεων της κάθε μεθοδολογίας παρατηρούνται κενά βελτιστότητας (Σχ. 0.11) μεταξύ των DSE προσεγγίσεων. Δεύτερον, η κάλυψη ενός ευρύτερου χώρου αναζήτησης που οδηγεί σε νέα Pareto σημεία θα πρέπει να λαμβάνει υπόψη

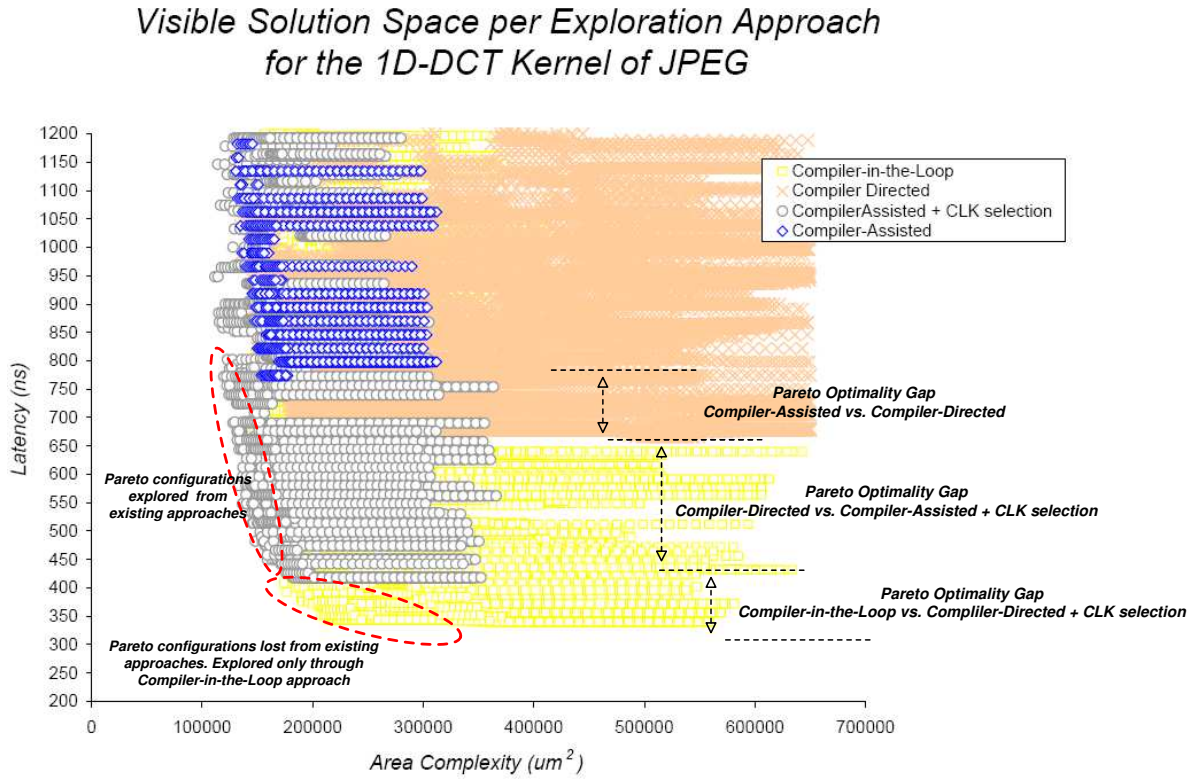


Figure 0.11.: Συγκριτική αξιολόγηση και κενά βελτιστότητας μεταξύ διαφορετικών μεθοδολογιών εξερεύνησης .

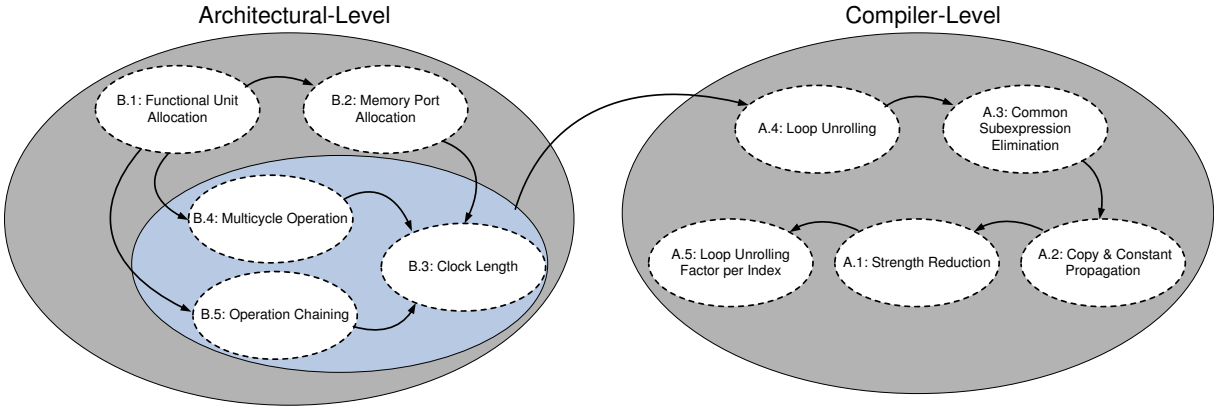


Figure 0.12.: CompInLoop Χώρος αποφάσεων και οι αντίστοιχες αλληλεξαρτήσεις .

την αλληλεπίδραση μεταξύ του συνδυασμού των παραμέτρων. Η αναζήτηση σε μια ευρύτερη περιοχή του χώρου λύσεων δεν εγγυάται ότι η εξερεύνηση θα κινηθεί προς την κατεύθυνση εύρεσης πιο αποδοτικών Pareto λύσεων, όπως φαίνεται από το κενό βελτιστότητας μεταξύ των CompDirected και το CompAssistedCLK μεθοδολογιών. Η CompInLoop μεθοδολογία εξερεύνησης επιτυγχάνει την αποτελεσματική αντιμετώπιση των παραπάνω παρατηρήσεων, στοχεύοντας σε έναν ευρύτερο χώρο παραμέτρων (αλγοριθμικό και αρχιτεκτονικό επίπεδο) αλλά ταυτόχρονα λαμβάνοντας υπόψη και την αξιολόγηση των αλληλεπιδράσεων που παράγονται από τους διάφορους συνδυασμούς των παραμέτρων. Πιο συγκεκριμένα, η προτεινόμενη προσέγγιση όχι μόνο περικλείει τους χώρους λύσεων που είναι ‘‘ορατοί’’ από τις υπάρχουσες μεθοδολογίες αλλά ταυτόχρονα επεκτείνει την ‘‘ορατότητα’’ της εξερεύνησης προς νέες περιοχές που περιέχουν σχεδιαστικές λύσεις υψηλότερης ποιότητας.

0.5.2. CompInLoop Χώρος Σχεδίασης

Ο προτεινόμενος χώρος αποφάσεων, που λειτουργεί ως χώρος παραμέτρων του συγκεκριμένου προβλήματος εξερεύνησης, καθώς και μια προτεινόμενη ιεράρχηση των αποφάσεων για την περίπτωση που τα κριτήρια βελτιστοποίησης αφορούν την καθυστέρηση και την επιφάνεια υλικού απεικονίζονται στο Σχ. 0.12. Η λεπτομερής ανάλυση της σημασιολογίας κάθε σχεδιαστικής απόφασης καθώς και της κάθε εξάρτησης δίνεται στο κυρίως κείμενο της διδακτορικής διατριβής.

Πιο συγκεκριμένα ο προτεινόμενος χώρος παραμέτρων αποτελείται από δύο υποχώρους αποφάσεων, το χώρο αλγοριθμικών σχεδιαστικών αποφάσεων στο επίπεδο του compiler (Compiler-Level) και το χώρο σχεδιαστικών αποφάσεων επίπεδο αρχιτεκτονικής (Architectural-Level). Η λογική ένωση των δύο υπο-χώρων αποφάσεων σχηματίζει τον συνολικό χώρο παραμέτρων του συγκεκριμένου προβλήματος.

Οι αλγοριθμικός χώρος παραμέτρων (Compiler-Level) περιλαμβάνει σχεδιαστικές αποφάσεις που αφορούν σε μετασχηματισμούς του πηγαίου κώδικα της αλγοριθμικής περιγραφής. Οι αποφάσεις αυτές δεν παρουσιάζουν άμεσο αντίκτυπο στην τελική αρχιτεκτονική του κυκλώματος, ωστόσο διαδίδουν περιορισμούς που επηρεάζουν σε

μεγάλο βαθμό την τελική αρχιτεκτονική. Αποφάσεις που καθορίζουν το μετασχηματισμούς βελτιστοποίησης όπως εξάλειψη και διάδοση κοινών υπο-εκφράσεων, ξεδίπλωμα βρόγχων, το βαθμό ξεδίπλωτος του κάθε βρόγχου κ.τ.λ. αποτελούν αποφάσεις που περιλαμβάνονται σε αυτό το επίπεδο.

Οι αρχιτεκτονικός χώρος παραμέτρων (Architectural-Level) περιλαμβάνει σχεδιαστικές αποφάσεις που έχουν άμεσο αντίκτυπο στην αρχιτεκτονική του τελικού κυκλώματος. Έτσι σχεδιαστικές αποφάσεις που περιλαμβάνονται στο συγκεκριμένο υπο-χώρο αφορούν στον αριθμό και στο είδος των πόρων υλικού (δομικές μονάδες, θύρες μνήμης κ.τ.λ), στη συχνότητα λειτουργίας της κυκλωματικής υλοποίησης, στον βαθμό αλύσιδωσης μεταξύ δομικών μονάδων κ.τ.λ.

Ο χώρος λύσεων περιλαμβάνει όλους τους δυνατούς συνδυασμούς (πλήρες παραγοντικό) που σχηματίζονται μεταξύ των αποφάσεων του χώρου παραμέτρων. Ο παραπάνω τρόπος σχηματισμού του CompInLoop χώρου λύσεων επιτρέπει την αξιολόγηση των επιπτώσεων του κάθε συνδυασμού παραμέτρων στην κυκλωματική υλοποίηση. Η προτεινόμενη ιεράρχηση των σχεδιαστικών αποφάσεων που απεικονίζεται στο Σχ. 0.12 ορίζει τη σειρά με την οποία οι παράμετροι πρέπει να αξιολογηθούν, ορίζοντας ουσιαστικά τη σειρά με την οποία οι σχηματίζονται οι βρόχοι εξερεύνησης στη μεθοδολογία της CompInLoop εξερεύνησης. Δεδομένου ότι το ενδιαφέρον της εξερεύνησης επικεντρώνεται στα trade-offs που παράγονται όταν σε μια συγκεκριμένη αρχιτεκτονική απεικονίζεται μια σημασιολογικά ισοδύναμη περιγραφή συμπεριφοράς, η ανάθεση των τιμών στις παραμέτρους αρχίζει από το επίπεδο αρχιτεκτονικής και στη συνέχεια για κάθε αρχιτεκτονική μορφοποίηση ανατίθενται και οι αντίστοιχες τιμές των παραμέτρων στο αλγοριθμικό επίπεδο. Σε αντίθεση με το χώρο παραμέτρων που συναντάται στην εξερεύνηση δυναμικών διαχειριστών μνήμης για πολύ-νηματικές εφαρμογές όπου σημασιολογικά μη-έγκυρες λύσεις μπορούν να ανιχνευτούν, ο CompInLoop χώρος παραμέτρων περιλαμβάνει μόνο έγκυρα διανύσματα, με αποτέλεσμα να μην είναι δυνατή μια πρόωρη εξάλειψη μη-έγκυρων σχεδιαστικά λύσεων. Η προτεινόμενη ιεράρχηση των αποφάσεων στον CompInLoop χώρο παραμέτρων έχει ως στόχο την εισαγωγή μιας κατάλληλης δομής στο χώρο λύσεων η οποία επιτρέπει την ανάπτυξη αποδοτικών ευριστικών κατά τη διάρκεια της εξερεύνησης.

0.5.3. CompInLoop Μεθοδολογία Σχεδίασης

Κάθε πιθανή σχεδιαστική λύση μπορεί να μοντελοποιηθεί ως ένα διάνυσμα αποφάσεων του CompInLoop χώρου σχεδίασης που περιγράφηκε στην προηγούμενη ενότητα. Ωστόσο, η εξαντλητική διερεύνηση όλων των πιθανών συνδυασμών αποφάσεων απαιτεί την πλήρη απαρίθμηση (πλήρες παραγοντικό του αριθμού των παραμέτρων) και αξιολόγηση όλων των δυνατών διανυσμάτων αποφάσεων, με αποτέλεσμα το μέγεθος του χώρου λύσεων να εκρήγνυται και να παύει να είναι υπολογιστικά διαχειρίσιμο. Προκειμένου να καταστεί διαχειρίσιμη η πολυπλοκότητα της εξερεύνησης, προτείνουμε μια μεθοδολογία που βασίζεται στο σχηματισμό κατάλληλων άνω οριακών συνθηκών που φράζουν το διαθέσιμο χώρο λύσεων και στην ανάπτυξη ενός νέου ευριστικού αλγορίθμου ο οποίος εκμεταλλεύεται την δομή του χώρου λύσεων, που παράγεται από την προτεινόμενη ιεράρχηση των αποφάσεων, προκειμένου να μειώσει περαιτέρω το χώρο αναζήτησης όταν διαπιστώνει πως αξιολογεί λύσεις σε τοπικά μη-βέλτιστες περιοχές κορεσμού.

Οι άνω οριακές συνθήκες αφορούν το μέγιστο αριθμό δομικών μονάδων πάνω από τον οποίο δεν έχει νόημα να ελεγχθούν σχεδιαστικές λύσεις. Η ανίχνευση των άνω οριακών συνθηκών επιτελείται με χρήση του αλγορίθμου χρονοδρομολόγησης As-Soon-As-Possible (ASAP) [44]. Για κάθε συνδυασμό των βαθμών ξεδιπλώματος των βρόγχων της εφαρμογής ορίζεται ξεχωριστό άνω όριο. Ο αλγόριθμος ASAP επιστρέφει τον χρονοπρογραμματισμό των λειτουργιών της εφαρμογής στη μέγιστη δυνατή παραλληλία (θεωρώντας την ύπαρξη άπειρων διαθέσιμων πόρων υλικού). Επομένως, ο αλγόριθμος εξερεύνησης δεν χρειάζεται να αξιολογήσει σχεδιαστικές λύσεις που ξεπερνούν το μέγιστο διαθέσιμο παραλληλισμό της εκάστοτε αλγοριθμικής αναπαράστασης.

Ο προτεινόμενος ευριστικός αλγόριθμος εκμεταλλεύεται τη δομή που εισάγει στο χώρο λύσεων η ιεράρχηση των αποφάσεων του χώρου παραμέτρων. Ουσιαστικά, η προτεινόμενη ιεράρχηση αποσυνθέτει το χώρο λύσεων σε μικρότερους υπο-χώρους λύσεων, ο καθένας από τους οποίους υλοποίει μια διακριτή συνάρτηση, έστω L, με μοναδική μεταβλητή τον αριθμό των διαθέσιμων μονάδων ALU . Το Σχ. 0.13 απεικονίζει τον υπο-χώρο λύσεων δυο γειτονικών συναρτήσεων L (γειτονικές ως προς τη τη δομή που εισάγει η προτεινόμενη ιεράρχηση των αποφάσεων). Υπάρχουν τόσες L όσοι και οι υπο-χώροι σχεδίασης. Παρατηρούμε ότι στην περίπτωση που ο κάθε υπο-χώρος λύσεων φτάνει σε ένα σημείο κορεσμού (ως προς τον αριθμό των ALU), η κλίση της συνάρτησης L που αντιστοιχεί στον κάθε υπο-χώρο μηδενίζεται. Ο προτεινόμενος ευριστικός αλγόριθμος εκμεταλλεύεται την παραπάνω παρατήρηση ώστε να διακόπτει την αναζήτηση σε έναν υπο-χώρο κάθε φορά που βρίσκει συνεχόμενες λύσεις της συνάρτησης L με μηδενική κλίση. Η ευριστική συμπεριφορά της προτεινόμενης προσέγγισης έγκειται στο γεγονός ότι μια μηδενική κλίση της συνάρτησης L δεν συνεπάγεται ότι η L βρίσκεται σε περιοχή κόρου. Προκειμένου να μετριαστεί η ευριστική συμπεριφορά, ο αλγόριθμος επαυξήθηκε με κατάλληλες μεταβλητές ελέγχου που καθορίζουν τόσο το βάθος αναζήτησης για την λήψη της απόφασης μηδενικής κλίσης όσο και τον τρόπο μετακίνησης μεταξύ διαφορετικών υπο-χώρων λύσεων σε περίπτωση εύρεσης λύσεων μηδενικής κλίσης.

Η πλήρης ανάλυση τόσο του θεωρητικού υποβάθρου βάσει του οποίου προκύπτουν οι άνω οριακές συνθήκες φραγμού του χώρου λύσεων καθώς όσο του προτεινόμενου ευριστικού αλγορίθμου, περιλαμβάνεται στο κυρίως κείμενο της διατριβής.

0.5.4. Ενδεικτικά Αποτελέσματα Πειραματικής Αξιολόγησης

Η ενότητα αυτή παρουσιάζει κάποια ενδεικτικά πειραματικά αποτελέσματα για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης προσέγγισης. Περαιτέρω πειραματικά αποτελέσματα μπορούν να βρεθούν στο κυρίως κείμενο της διατριβής. Εφαρμόσαμε την CompInLoop μεθοδολογία εξερεύνησης σε ένα σύνολο υπολογιστικά απαιτητικών εφαρμογών από το πεδίο ψηφιακής επεξεργασίας σήματος.

Δεδομένου ότι ο στόχος της CompInLoop μεθοδολογίας εξερεύνησης είναι να μετακινηθεί η σχεδίαση προς λύσεις υψηλότερης ποιότητας, σε σύγκριση με τις λύσεις που προκύπτουν από τις υπάρχουσες μεθοδολογίες εξερεύνησης, το Σχ. 0.14 απεικονίζει τα Pareto σύνορα που προέρχονται από τις CompAssisted, CompAssistedCLK, CompDirected και CompInLoop μεθοδολογίες εξερεύνησης για έξι εφαρμογές. Ο άξονας Υ αναφέρεται στην καθυστέρηση που συνδέεται με Pareto λύση. Ο άξονας Χ αν-

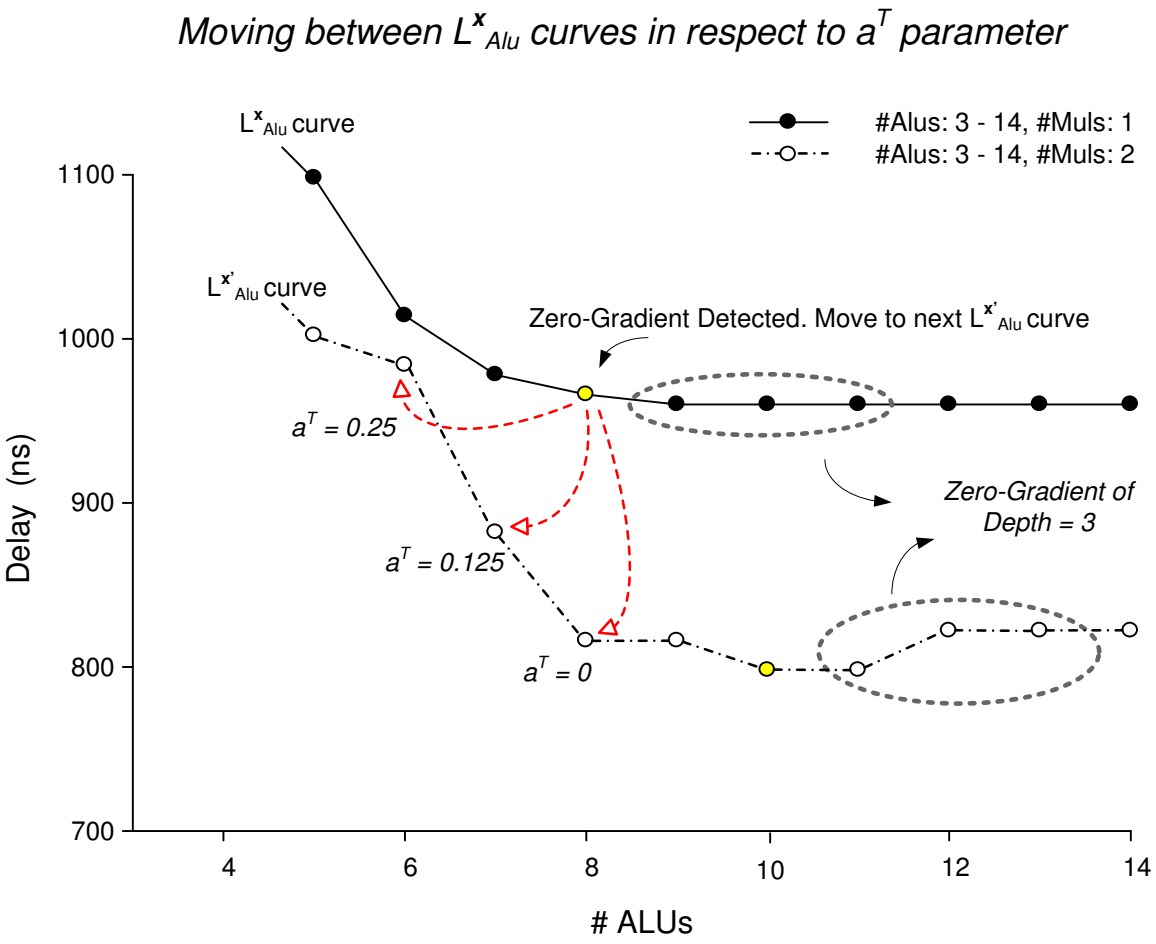
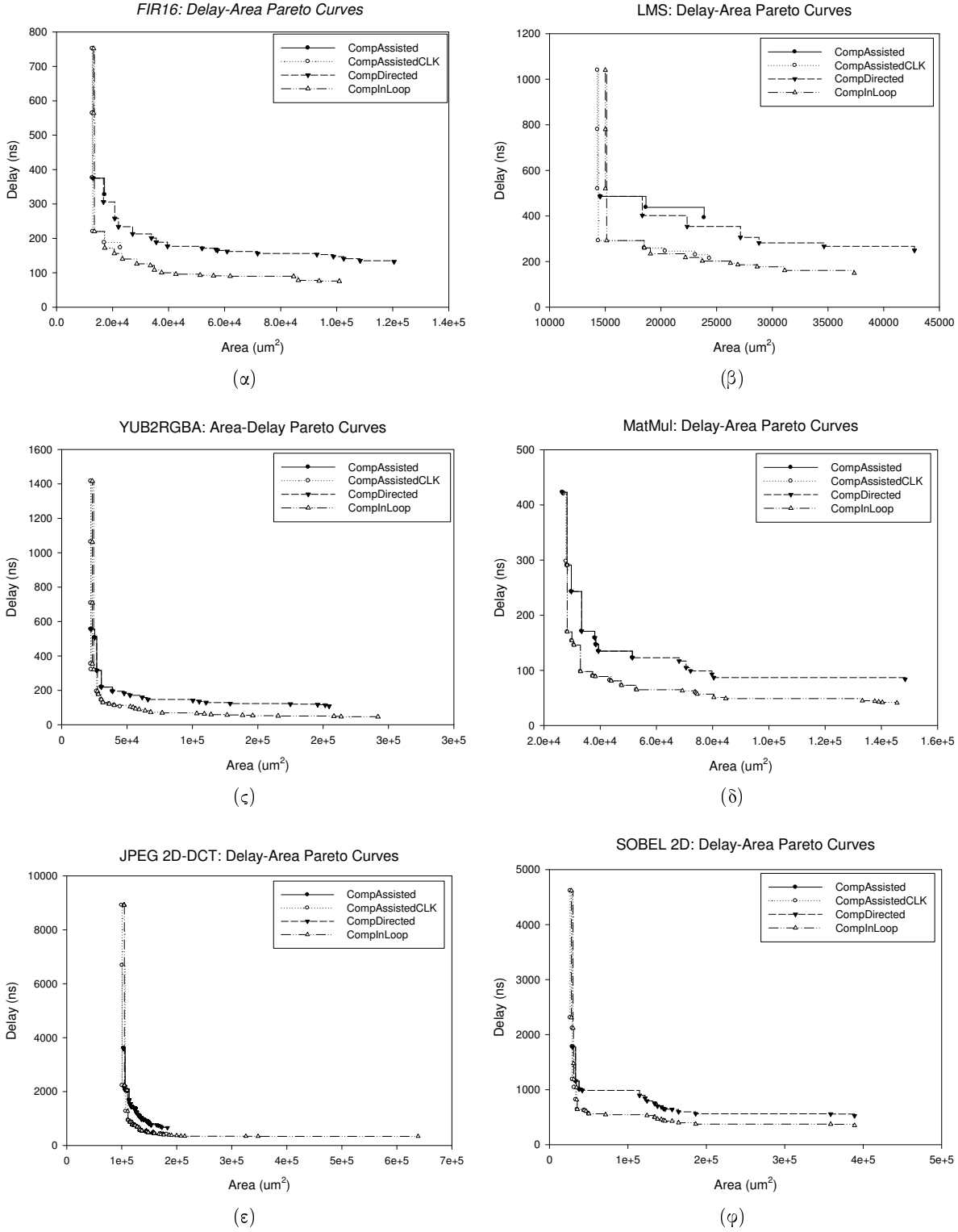


Figure 0.13.: Σημεία υπο-χώρων συναρτήσεων γειτονικών L.

τιπροσωπεύει το αντίστοιχο κόστος σε επιφάνεια υλικού. Τα πειραματικά αποτελέσματα δείχνουν ότι η CompInLoop μεθοδολογία εξερεύνησης επιστέφει υψηλότερης ποιότητας σημεία σχεδιασμού σε σχέση με τις υπόλοιπες στρατηγικές. Συγκεκριμένα, οι Pareto λύσεις που προκύπτουν από την προτεινόμενη CompInLoop εξερεύνηση κυριαρχούν σε όλες τις περιπτώσεις των λύσεων που προκύπτουν από τις υπόλοιπες μεθοδολογίες. Η ίδια συμπεριφορά παρατηρείται, επίσης, για το υπόλοιπο των εφαρμογών που δεν απεικονίζονται στο σχήμα. Πρέπει να υπογραμμιστεί πως η συγκεκριμένη σύγκριση πραγματοποιήθηκε θεωρώντας εξαντλητική αναζήτηση για τις CompAssisted, CompAssistedCLK και CompDirected προσεγγίσεις και ευριστική αναζήτηση για την προτεινόμενη CompInLoop μεθοδολογία. Επομένως, η CompInLoop μεθοδολογία, παρά την ευριστική φύση της εξερεύνησης, είναι σε θέση να βρίσκει πιο αποτελεσματικές σχεδιαστικές λύσεις σε σχέση με τις υπάρχουσες προσεγγίσεις. Τα παραπάνω μας πληροφορούν ότι η αναζήτηση είτε σε μικρούς χώρους σχεδιασμού, π.χ. στην περίπτωση της CompAssisted, ή σε μεγάλους χώρους λύσεων, π.χ. στην περίπτωση της CompDirected, μας οδηγεί σε υπο-βέλτιστες σχεδιαστικές λύσεις, αν δεν ληφθούν προσεχτικά υπόψη οι αλληλεπιδράσεις που προκύπτουν από το σύνολο του χώρου παραμέτρων.



Φιγυρε 0.14.: Συγκριτική μελέτη Delay-Area Pareto σημείων που προκύπτουν από διαφορετικές στρατηγικές DSE .

0.6. Αρχιτεκτονική Σύνθεση Συνεπεξεργαστών Υλικού Αδρομερούς Επαναδιάταξης Με Εισαγωγή Ευελιξίας

Η ευελιξία που προσφέρουν οι επαναδιατάξιμες αρχιτεκτονικές συνεπεξεργαστή επιτρέπει την επαναχρησιμοποίηση του υλικού, καθώς στον ίδιο επαναδιατάξιμο συνεπεξεργαστή απεικονίζονται περισσότεροι του ενός πυρήνες της εφαρμογής. Τυπικά, η επαναχρησιμοποίηση υλικού επιτυγχάνεται στο επίπεδο της μικρο-αρχιτεκτονικής όπου οι δομικές μονάδες υλικού π.χ. αριθμητικοί τελεστές, διαμοιράζονται μεταξύ των αμοιβαίως αποκλειόμενων πυρήνων που απεικονίζονται στον επαναδιατάξιμο συνεπεξεργαστή. Εξετάζοντας με ενιαίο τρόπο τη σχεδίαση στο επίπεδο της μικρο-αρχιτεκτονικής και στο επίπεδο κυκλώματος, η τυπική προσέγγιση διαμοιρασμού των δομικών μονάδων μπορεί να επεκταθεί με δυνατότητες διαμοιρασμού κυκλωματικών δομών που βρίσκονται στο επίπεδο του bit, παρέχοντας με τον τρόπο πιο αποδοτικές λύσεις αναφορικά με την καταλαμβανόμενη επιφάνεια υλικού. Η νέα οπτική που επιφέρει μια τέτοια επέκταση της τυπικής προσέγγισης σχεδίασης, γεννά την ανάγκη για ανάπτυξη νέων μεθοδολογιών σχεδίασης στα επίπεδα κυκλώματος, μικρο-αρχιτεκτονικής και αρχιτεκτονικής σύνθεσης προκειμένου οι σχεδιαστές να οδηγηθούν προς αποδοτικότερες υλοποιήσεις επαναδιατάξιμων αρχιτεκτονικών.

Η παρούσα διατριβή προτείνει ένα ολοκληρωμένο σύνολο μεθοδολογιών (επίπεδο κυκλώματος, μικρο-αρχιτεκτονικής, αρχιτεκτονικής σύνθεσης) οι οποίες επιτρέπουν τη σχεδίαση επαναδιατάξιμων αρχιτεκτονικών προτύπων με αυξημένες δυνατότητες διαμοιρασμού του εκχωρημένου υλικού. Στο επίπεδο κυκλώματος, παρουσιάζουμε την τεχνική της Εισαγωγής Ευελιξίας, η οποία επιτρέπει μέσω κατάλληλων μετασχηματισμών την απεικόνιση κυκλωματικών περιγραφών διαφόρων αριθμητικών τελεστών στην κυκλωματική δομή ενός πολλαπλασιαστή τύπου-πίνακα. Στο επίπεδο μικρο-αρχιτεκτονικής κάνοντας χρήση των δομικών μονάδων που προκύπτουν από εφαρμογή της Εισαγωγής Ευελιξίας, ορίζουμε ένα νέο αρχιτεκτονικό πρότυπο ευέλικτων συνεπεξεργαστών οι οποίοι υποστηρίζουν εγγενώς τη συνδυασμένη εκμετάλλευση του οριζόντιου παραλληλισμού, του κάθετου παραλληλισμού και της αλυσίδωσης λειτουργιών. Στο επίπεδο αρχιτεκτονικής σύνθεσης, αναπτύχθηκαν εξειδικευμένοι αλγόριθμοι σύνθεσης οι οποίοι επεκτείνουν τις υπάρχουσες μεθοδολογίες αρχιτεκτονικής σύνθεσης, προκειμένου να επιτευχθεί μια ροή σχεδίασης για αποδοτική σύνθεση αλγοριθμικών περιγραφών πάνω στο προτεινόμενο επαναδιατάξιμο αρχιτεκτονικό πρότυπο.

0.6.1. Επίπεδο Κυκλώματος: Τεχνική Εισαγωγή Ευελιξίας

Η τεχνική Εισαγωγή Ευελιξίας είναι ένα σύνολο μετασχηματισμών στο επίπεδο κυκλώματος για την σχεδίαση αποδοτικών επαναδιατάξιμων αριθμητικών μονάδων για εφαρμογές ψηφιακής επεξεργασίας σήματος. Βάσει της προτεινόμενης τεχνικής, οι κυκλωματικές περιγραφές των αριθμητικών τελεστών που υλοποιούν πρόσθεση/αφαίρεση στην Carry-Save (CS) αριθμητική αναπαράσταση απεικονίζονται στο κυκλώμα ενός CS πολλαπλασιαστή τύπου πίνακα. Με τον τρόπο αυτό, οι αριθμητικές λειτουργίες του πολλαπλασιασμού, της πρόσθεσης και της αφαίρεσης συγχωνεύονται σε ένα ενιαίο ευέλικτο κύκλωμα.

Η ευελιξία εισάγεται με μηδενική επιβάρυνση σε ότι αφορά στο κανονικό (και απο-

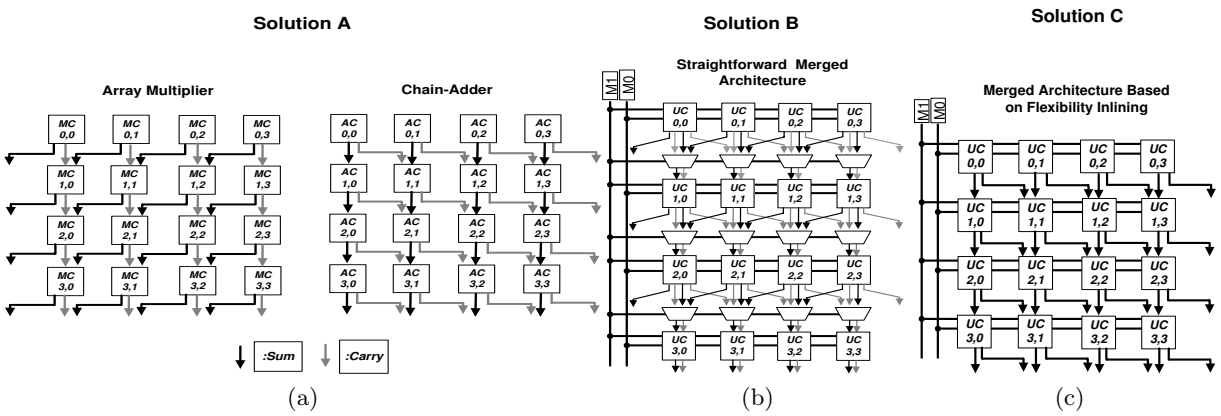


Figure 0.15.: a) Λύση 1: Κύκλωμα με 1 CS πολλαπλασιαστή τύπου-πίνακα και 1 CS αθροιστή 6 δεδομένων, b) Λύση 2: Συγχώνευση κυκλωμάτων χώρις Εισαγωγή Ευελιξίας, c) Λύση 3: Συγχώνευση κυκλωμάτων με Εισαγωγή Ευελιξίας.

δοτικό για VLSI σχεδίαση) σχήμα διασύνδεσης που παρουσιάζουν οι CS πολλαπλασιαστές τύπου-πίνακα. Μέσω κατάλληλων μετασχηματισμών κατοπτρικής συμμετρίας, εξάγονται τα κοινά σχήματα διασύνδεσης μεταξύ των αριθμητικών κυκλωμάτων. Βάσει των κοινών σχημάτων διασύνδεσης, επιτυγχάνεται η συγχώνευση μεταξύ του κανονικού (ή κατοπτρικού) κυκλώματος των αθροιστών/αφαιρετών με το κατοπτρικό (ή κανονικό) κύκλωμα του πολλαπλασιαστή πάνω από το κοινό σύστημα διασύνδεσης.

Δεδομένου ότι οι DSP εφαρμογές κυριαρχούνται από αριθμητικές λειτουργίες πρόσθεσης/αφαίρεσης/πολλαπλασιασμού η προτεινόμενη τεχνική αποτελεί μια ελκυστική λύση για επαναδιατάξιμες αρχιτεκτονικές που στοχεύουν στο συγκεκριμένο πεδίο εφαρμογών. Ας υποθέσουμε την περίπτωση κατά την οποία η σχεδίαση απαιτεί την εκχώρηση υλικού για τα κυκλώματα ενός CS πολλαπλασιαστή και ενός CS -αθροιστή για άθροιση 6 προσθετέων. Το Σχ. 0.15 απεικονίζει τρεις δυνατές σχεδιαστικές λύσεις. Με MC και AC συμβολίζονται τα βασικά κύτταρα 1-bit που υλοποιούν πολλαπλασιασμό και πρόσθεση, αντίστοιχα. Με UC συμβολίζονται τα κύτταρα τα οποία προκύπτουν μέσω της προτεινόμενης μεθοδολογίας και εκτελούν ενοποιημένα τις πράξεις του πολλαπλασιασμού, πρόσθεσης ή αφαίρεσης. Η πρώτη λύση Α (Σχ. 0.15a) χρησιμοποιεί τα δύο αριθμητικά κυκλώματα χωριστά. Δεδομένου ότι δεν επιτυγχάνεται κανένας διαμοιρασμός υλικού, η λύση αυτή παρουσιάζει υψηλό κόστος σε επιφάνεια υλικού. Η δεύτερη σχεδιαστική λύση, (Σχ. 0.15b), πραγματοποιεί συγχώνευση των δύο κυκλωμάτων, χωρίς να λαμβάνει υπόψη την κατοπτρική συμμετρία στο σχήμα διασύνδεσης. Παρά το γεγονός ότι επιτυγχάνεται διαμοιρασμός υλικού, η λύση συνεχίζει να είναι υπο-βέλτιστη καθώς ένας μεγάλος αριθμός πολυπλεκτών εισάγεται ώστε να διασφαλιστεί η ορθή διάδοση των σημάτων. Η τρίτη σχεδιαστική λύση (Σχ. 0.15c) πραγματοποιεί συγχώνευση των δύο αριθμητικών κυκλωμάτων σύμφωνα με την τεχνική της Εισαγωγής Ευελιξίας. Η προτεινόμενη τεχνική, λαμβάνοντας υπόψη τις δυνατότητες για διαμοιρασμό υλικού που προκύπτουν από την κατοπτρική συμμετρία, παράγει μια βελτιστοποιημένη συγχωνευμένη κυκλωματική αρχιτεκτονική χωρίς την επιβάρυνση των πολυπλεκτών που παρουσιάζονται στη δεύτερη λύση.

Η τεχνική Εισαγωγή Ευελιξίας αποτελείται από δύο βήματα, (1) το μετασχηματισμό

Table 0.1.: ***UC Implementation Strategies***.

Cell	Area (um^2)	Power (uW)	Area Overheads	Power Overheads
UC_{STRFW1}	370.2	271.8	71.6%	79%
UC_{STRFW2}	275.1	255.3	27.5%	68.2%
UC_{SUBOPT}	232.6	213.9	7.8%	40.9%
UC_{OPT}	215.7	151.8	0	0
MC	93.4	63.2	-56.7%	-58.3%

ομοιομορφίας ο οποίος μετασχηματίζει κατάλληλα τις κυκλωματικές περιγραφές των αριθμητικών τελεστών ώστε να αξιοποιηθεί ένα κοινό σχήμα διασύνδεσης και (2) τη σχεδίαση του ενοποιημένου κυττάρου (**Unified Cell, UC**) υπολογισμού **1-bit** που συγχωνεύει αποδοτικά τις λειτουργίες των βασικών κυκλωματικών κυττάρων **1-bit** της πρόσθεσης/αφαίρεσης/πολλαπλασιασμού. Μια λεπτομερής περιγραφή της **Εισαγωγής Ευελιξίας** παρουσιάζεται στο κυρίως τμήμα της διδακτορικής διατριβής.

Ο μετασχηματισμός ομοιομορφίας αποτελείται από τα τέσσερα βήματα που απεικονίζονται στο **Σχ. 0.16**. Στο πρώτο βήμα, επιλέγεται η κυκλωματική περιγραφή που θα αποτελέσει την αρχιτεκτονική αναφοράς, δηλαδή την κυκλωματική αρχιτεκτονική που θα διατηρήσει το σχήμα διασύνδεσης της, ώστε πάνω σε αυτό να απεικονιστούν τα δίκτυα διασύνδεσης των υπόλοιπων αριθμητικών τελεστών. Στο **Σχ. 0.16** η αρχιτεκτονική ενός **CS** αθροιστή **6** δεδομένων θεωρήθηκε ως η αρχιτεκτονική αναφοράς, δεδομένου ότι στις περισσότερες των εφαρμογών, ο αριθμός των πράξεων άθροισης είναι μεγαλύτερος των πράξεων πολλαπλασιασμού. Στο δεύτερο βήμα, εφαρμόζεται ένας μετασχηματισμός κατοπτρικής συμμετρίας στις κυκλωματικές περιγραφές των υπόλοιπων αριθμητικών τελεστών, ο οποίος αποκαλύπτει ένα κοινό σχήμα διασύνδεσης μεταξύ των εξεταζομένων κυκλωμάτων. Το σχήμα διασύνδεσης είναι κοινό δομικά αλλά όχι και σημασιολογικά, καθώς ο ίδιος δομικά φορέας σήματος (καλώδιο διασύνδεσης) χρησιμοποιείται από διαφορετικά σήματα σε κάθε κυκλωματική περιγραφή. Στο τρίτο βήμα, η παραπάνω αναντιστοιχία αντιμετωπίζεται με επανασχεδιασμό της κυκλωματικής διεπαφής των βασικών αριθμητικών κυττάρων. Τέλος, στο τέταρτο βήμα, παράγεται το κυκλωματικό πρότυπο των συγχωνευμένων αριθμητικών τελεστών.

Η σχεδίαση του ενοποιημένου κυττάρου λειτουργιών, **UC**, αφορά στη σχεδίαση της εσωτερικής δομής του κυττάρου ώστε να μεγιστοποιηθεί ο διαμοιρασμός υλικού. **Χ**ρησιμοποιήθηκε η συγχώνευση των πινάκων αληθείας των κυττάρων που δομούν τα αριθμητικά κυκλώματα με χρήση του εργαλείου λογικής σύνθεση **ESPRESSO** [45]. **Ο** πίνακας **0.1** συγκρίνει την προτεινόμενη προσέγγιση UC_{OPT} σε σχέση με άλλες εναλλακτικές σχεδίασης του **UC** έχοντας ως σημείο αναφοράς το μη-ευέλικτο κελί του πολλαπλασιαστή, **MC** . **Η** μέθοδος συγχώνευσης λειτουργιών στην περίπτωση του UC_{OPT} πραγματοποιείται με το μικρότερο κόστος υλοποίησης.

0.6.2. Επίπεδο Μικρο-Αρχιτεκτονικής

Στο επίπεδο της μικρο-αρχιτεκτονικής προδιαγράφονται τα χαρακτηριστικά και η οργάνωση της συνολικής αρχιτεκτονικής σε ότι αφορά τις δομικές μονάδες, το σχήμα διασύνδεσης, το μοντέλο υπολογισμού κ.τ.λ. και ταυτόχρονα εξάγονται τα υπολογι-

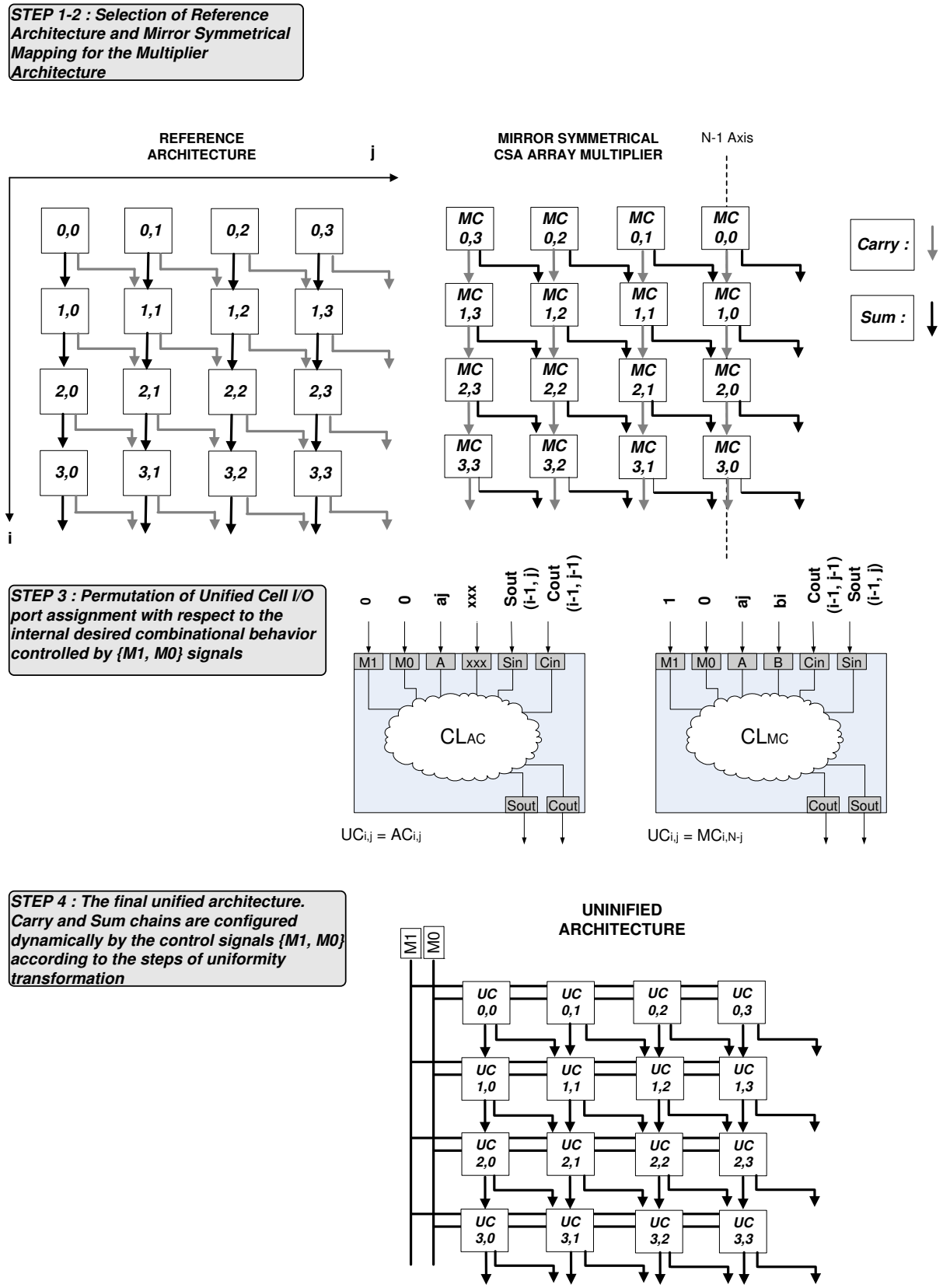


Figure 0.16.: Μετασχηματισμός Ομοιομοφίας .

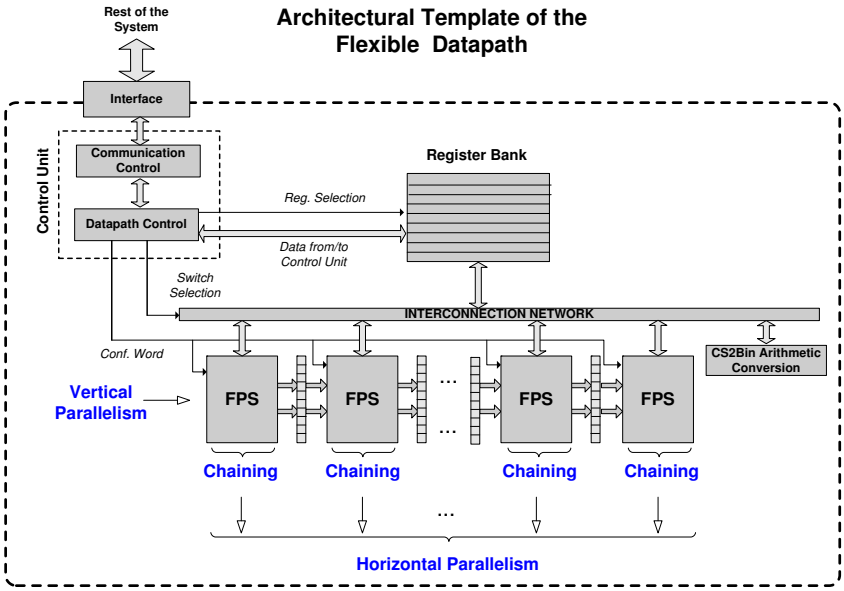


Figure 0.17.: Προτεινόμενη μικρο-αρχιτεκτονική οργάνωση.

στικά μοντέλα των δομικών μονάδων υλικού (αντίστοιχα με τις εντολές ενός μικρο-επεξεργαστή), τα οποία θα χρησιμοποιηθούν στη φάση της σύνθεσης αρχιτεκτονικής για την απεικόνιση των αλγοριθμικών περιγραφών πάνω στην επαναδιατάξιμη αρχιτεκτονική.

Το Σχ. 0.17 απεικονίζει ένα αφαιρετικό μοντέλο του προτεινόμενου επαδιατάξιμου μικρο-αρχιτεκτονικού προτύπου. Πιο συγκεκριμένα, αποτελείται από έξι δομικά στοιχεία: (1) τα ευέλικτα στάδια της αρχιτεκτονικής σωλήνωσης (Flexible Pipeline Stage, FPS) που αποτελούν τις δομικές μονάδες υπολογισμού, (2) τους καταχωρητές μεταξύ κάθε σταδίου της σωλήνωσης, (3) το μητρώο καταχωρητών για αποθήκευση των ενδιάμεσων δεδομένων, (4) το δίκτυο διασύνδεσης, (5) τη μονάδα που μετατρέπει τα δεδομένα από την CS στη δυαδική αριθμητική αναπαράσταση (CS2Bin) και (6) τη μονάδα ελέγχου που οδηγεί τη συνολική αρχιτεκτονική (οδήγηση σημάτων επιλογής και διαμόρφωσης στις μονάδες FPS και στο δίκτυο διασύνδεσης). Η προτεινόμενη αρχιτεκτονική υιοθετεί ένα μοντέλο υπολογισμού στο οποίο τα σήματα ελέγχου παρέχονται από τη μονάδα ελέγχου και τα δεδομένα φορτώνονται στο μητρώο καταχωρητών πριν από τη φάση εκτέλεσης. Η αρχιτεκτονική έχει σχεδιαστεί για μήκος λέξης δεδομένων 16-bit, το οποίο θεωρείται κατάλληλο για τις υπολογιστικά απαιτητικές DSP εφαρμογές [46].

Το Σχ. 0.17 απεικονίζει τον τρόπο με τον οποίο το ευέλικτο τμήμα της αρχιτεκτονικής (μονάδες FPS) επιτρέπει το συνδυασμό τριών αρχιτεκτονικών βελτιστοποιήσεων, οριζόντιος παραλληλισμός, κάθετος παραλληλισμός, αλυσίδωση εκτέλεσης λειτουργιών. Ο οριζόντια παραλληλισμός αφορά στον αριθμό των λειτουργιών που μπορούν να εκτελεστούν ταυτόχρονα από την αρχιτεκτονική. Υποστηρίζεται στο επίπεδο της αρχιτεκτονικής οργάνωσης και είναι ανάλογος του αριθμού των μονάδων FPS. Ο κατακόρυφος παραλληλισμός υποστηρίζεται επίσης στο επίπεδο της αρχιτεκτονικής οργάνωσης και αφορά στις δυνατότητες υπολογιστικής σωλήνωσης (pipelining) που παρέχονται από το προτεινόμενο αρχιτεκτονικό πρότυπο μέσω των

ενδιάμεσων καταχωρητών. Ένας υπολογισμός που ακολουθεί ροή υπολογιστικής σωλήνωσης μπορεί να ξεκινήσει και να τερματιστεί σε οποιοδήποτε μονάδα FPS. Με τον τρόπο αυτό, επιτυγχάνεται η συνεργασία μεταξύ οριζόντιου και κάθετου παραλληλισμού καθώς πολλαπλές υπολογιστικές σωληνώσεις (ο μέγιστος αριθμός καθορίζεται από το βαθμό οριζόντιου παραλληλισμού) είναι δυνατόν να εκτελούνται από τις μονάδες FPS. Η δυνατότητα για αλυσίδωση λειτουργιών προκύπτει από την εσωτερική δομή των μονάδων FPS. Κάθε μονάδα FPS προκύπτει από εφαρμογή της τεχνικής Εισαγωγή Ευελιξίας. Μια μονάδα FPS μπορεί να συγχωνεύσει την εκτέλεση λειτουργιών με εξαρτήσεις δεδομένων, προκειμένου να εκτελεστούν σε ένα κύκλο ρολογιού. Αυτή η δυνατότητα ταχείας εκτέλεσης πολλαπλών λειτουργιών σε ένα κύκλο ρολογιού επιτυγχάνεται μέσω της CS αριθμητικής αναπαράστασης, η οποία εξαλείφει τις μεγάλες αλυσίδες διάδοσης κρατουμένου που παρουσιάζονται στα κυκλώματα δυαδικής αριθμητικής αναπαράστασης. Το Σχ. 0.19 απεικονίζει τη βιβλιοθήκη των υποστηριζόμενων λειτουργιών που μπορούν να εκτελεστούν από μια μονάδα FPS.

0.6.3. Επίπεδο Αρχιτεκτονικής Σύνθεσης

Η προτεινόμενη μεθοδολογία αρχιτεκτονικής σύνθεσης δέχεται ως είσοδο το σύνολο των πυρήνων λογισμικού $K = K_1, ..., K_i, ..., K_N\}$, οι οποίοι πρόκειται να απεικονιστούν πάνω στην επαναδιατάξιμη αρχιτεκτονική υλικού. Κάθε πυρήνας, K_i, μοντελοποιείται ως ένας ξεχωριστός κόμβος στο Γράφο Ροής Ελέγχου (Control-Flow Graph, CFG). Η προτεινόμενη ροή αρχιτεκτονικής σύνθεσης απεικονίζει κάθε πυρήνα πάνω στο σύνολο των κοινών πόρων υλικού που δομούν την επαναδιατάξιμη αρχιτεκτονική (π.χ. μονάδες FPS, CS2Bin κυκλώματα, μητρώα καταχωρητών κ.τ.λ.). Με τον τρόπο αυτό, όλοι οι πυρήνες μοιράζονται τους ίδιους πόρους υλικού. Στην έξοδο της διαδικασίας αρχιτεκτονικής σύνθεσης παράγεται ένα βελτιστοποιημένο στιγμιότυπο της προτεινόμενης ευέλικτης αρχιτεκτονικής σε συνθέσιμη Verilog RTL [47], που υλοποιεί τις απεικονισμένες αλγοριθμικές περιγραφές.

Το Σχ. 0.18 απεικονίζει τη συνολική ροή της προτεινόμενης μεθοδολογίας αρχιτεκτονικής σύνθεσης. Η αλγοριθμική περιγραφή δίνεται από το σχεδιαστή στη γλώσσα C. Από τον κώδικα C παράγεται η ενδιάμεση αναπαράσταση του Γράφου Ροής Ελέγχου-Δεδομένων (Control-Data-Flow Graph, CDFG) με χρήση συναρτήσεων ανάλυσης (λεκτική και συντακτική ανάλυση, ανάλυση εξαρτήσεων δεδομένων και ελέγχου) που είναι διαθέσιμες σε τυπικούς μεταγλωττιστές λογισμικού. Ο CDFG αποτελεί την είσοδο της προτεινόμενης ροής αρχιτεκτονικής σύνθεσης. Συγκεκριμένα, αποτελείται από δύο φάσεις:

- Φάση 1: Στη φάση αυτή, ο CDFG γράφος μετασχηματίζεται και βελτιστοποιείται προκειμένου να περιγράψει την αλγοριθμική συμπεριφορά σε ένα μοντέλο ενδιάμεσης αναπαράστασης πιο κοντά στα μικρο-αρχιτεκτονικά χαρακτηριστικά της προτεινόμενης επαναδιατάξιμης αρχιτεκτονικής. Αρχικά, κάθε λειτουργία πολλαπλασιασμού του CDFG μετασχηματίζεται σε μια υπολογιστικής σωλήνωση μικρότερου εύρους πολλαπλασιασμών ανάλογα με το εύρος των δεδομένων εισόδων (σε περίπτωση πολλαπλασιασμού με σταθερούς όρους). Στη συνέχεια, οι υπόλοιπες πρωτογενείς λειτουργίες/κόμβοι του CDFG ομαδοποιούνται σε μεγαλύτερους κόμβους που αντιστοιχούν σε λειτουργίες FPS της βιβλιοθήκης στο Σχ. 0.19. Με τον τρόπο αυτό αξιοποιούνται οι δυνατότητες για αλυσίδωση λει-

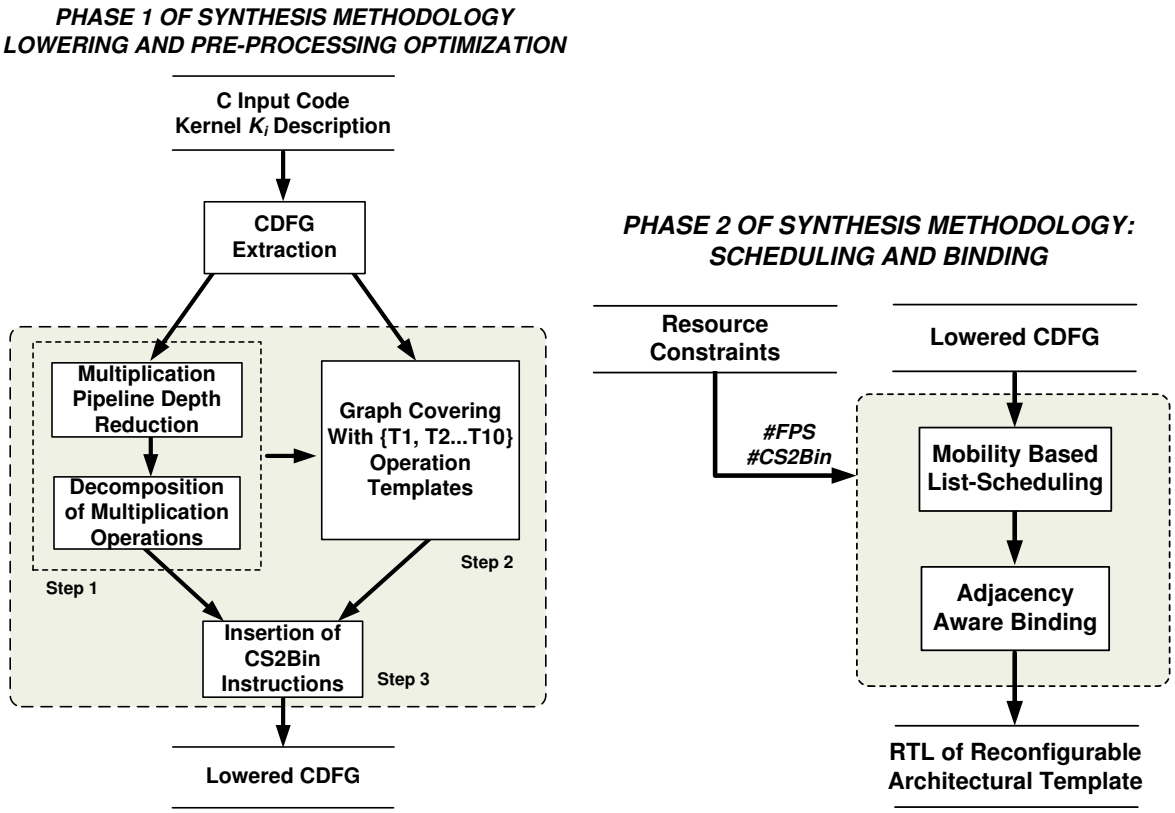


Figure 0.18.: Μεθοδολογία σύνθεσης αρχιτεκτονικής .

τουργιών που υποστηρίζονται από την προτεινόμενη επαναδιατάξιμη αρχιτεκτονική. Στο τελευταίο βήμα, πραγματοποιείται η εισαγωγή κατάλληλων εντολών μετατροπής των δεδομένων από την CS στη δυαδική αριθμητική αναπαράσταση σύμφωνα με τα χαρακτηριστικά της αρχιτεκτονικής. Με την ολοκλήρωση των παραπάνω διαδικασιών ο αρχικός CDFG μετασχηματίζεται σε έναν ισοδύναμο γράφο, L-CDFG , ο οποίος περιγράφει την αλγοριθμική περιγραφή με δομές πιο κοντά στο υλικό της αρχιτεκτονικής.

Φάση 2: Στη φάση αυτή, η μετασχηματισμένη εσωτερική αναπαράσταση χρονοδρομολογείται βάσει των περιορισμών στους πόρους υλικού ενός συγκεκριμένου αρχιτεκτονικού στιγμιότυπου. Στη συνέχεια οι χρονοδρομολογημένες λειτουργίες ανατίθενται στις μονάδες FPS και παράγεται το αντίστοιχο κύκλωμα ελέγχου μαζί με τα σήματα οδήγησης. Η συνάρτηση χρονοδρομολόγησης υλοποιεί έναν χρονοδρομολογητή τύπου λίστας [44] ο οποίος αξιοποιεί τον οριζόντιο παραλληλισμό της εφαρμογής αξιοποιεί την κινητικότητα λειτουργιών στον L-CDFG και τον αριθμό των διαθέσιμων μονάδων FPS. Δεδομένου ότι η χρονοδρομολόγηση δεν λαμβάνει υπόψη τις δυνατότητες για κάθετο παραλληλισμό που προσφέρονται από την προτεινόμενη επαναδιατάξιμη αρχιτεκτονική, αναπτύχθηκε ένας νέος αλγόριθμος ανάθεσης των χρονοδρομολογημένων λειτουργιών του L-CDFG σε μονάδες FPS. Ο προτεινόμενος αλγόριθμος ανάθεσης λαμβάνει υπόψη η φυσική γειτνίαση των μονάδων FPS ώστε να μεγιστοποιήσει τη χρήση

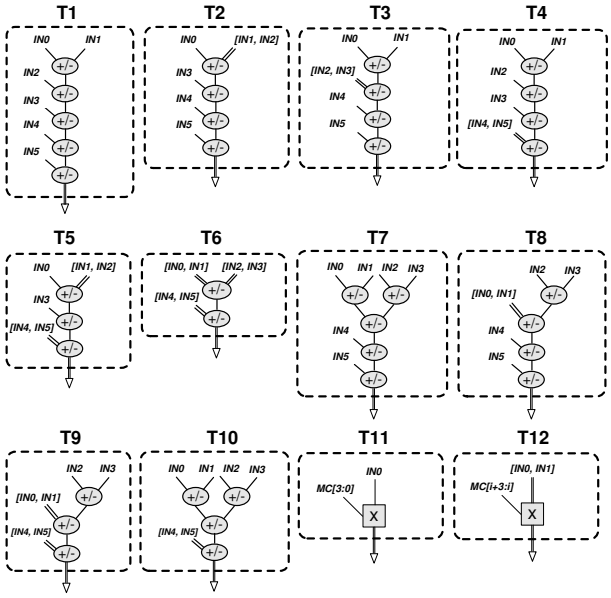


Figure 0.19.: Βιβλιοθήκη Λειτουργιών FPS.

των ενδιάμεσων καταχωρητών σωλήνωσης, απεικονίζοντας L-CDFG λειτουργίες με εξαρτήσεις δεδομένων σε γειτονικές μονάδες FPS.

0.6.4. Ενδεικτικά Αποτελέσματα Πειραματικής Αξιολόγησης

Η ενότητα αυτή παρουσιάζει κάποια ενδεικτικά πειραματικά αποτελέσματα για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης προσέγγισης. Περαιτέρω πειραματικά αποτελέσματα μπορούν να βρεθούν στο κυρίως κείμενο της διατριβής. Η πειραματική αξιολόγηση που παρουσιάζεται στην ενότητα αυτή αφορά στην αποδοτικότητα της προτεινόμενης αρχιτεκτονικής για DSP εφαρμογές που απεικονίστηκαν μέσω της προτεινόμενης μεθοδολογίας αρχιτεκτονικής σύνθεσης.

Ένα σύνολο από υπολογιστικά απαιτητικούς πυρήνες DSP απεικονίστηκαν σε τέσσερις επαναδιατάξιμες αρχιτεκτονικές υλικού. Συγκεκριμένα, εξετάστηκαν οι εξής αρχιτεκτονικές λύσεις: (1) η προτεινόμενη ευέλικτη αρχιτεκτονική που περιλαμβάνει τέσσερις μανάδες FPS και μία μονάδα CS2Bin, (2) μια αρχιτεκτονική τύπου CRISP [15] που περιλαμβάνει τέσσερις επαναδιατάξιμες δομικές μονάδες, (3) μια αρχιτεκτονική τύπου FCC [16] με δύο κόμβους FCC και (4) η αρχιτεκτονική τύπου NISC [17] με 4 ALUs και 1 πολλαπλασιαστή.

Συγκρίνουμε την αποδοτικότητα της προτεινόμενης προσέγγισης βάσει του γινομένου των μετρικών επιφάνειας υλικού και καθυστέρησης λειτουργίας (Area-Delay, AD). Στο Σχ. 0.20a το AD της προτεινόμενης αρχιτεκτονικής απεικονίζεται σε σχέση με το αντίστοιχο AD της CRISP. Το Σχ. 0.20b δείχνει τη σύγκριση της AD μετρικής σε σχέση με την FCC, ενώ το Σχ. 0.20c την αντίστοιχη σύγκριση με τη NISC. Συγκεκριμένα, οι τιμές των AD έχουν κανονικοποιηθεί ως προς με τα αντίστοιχα AD των υπόλοιπων αρχιτεκτονικών. Επομένως, η διακεκομμένη γραμμή (AD = 1) αναφέρεται

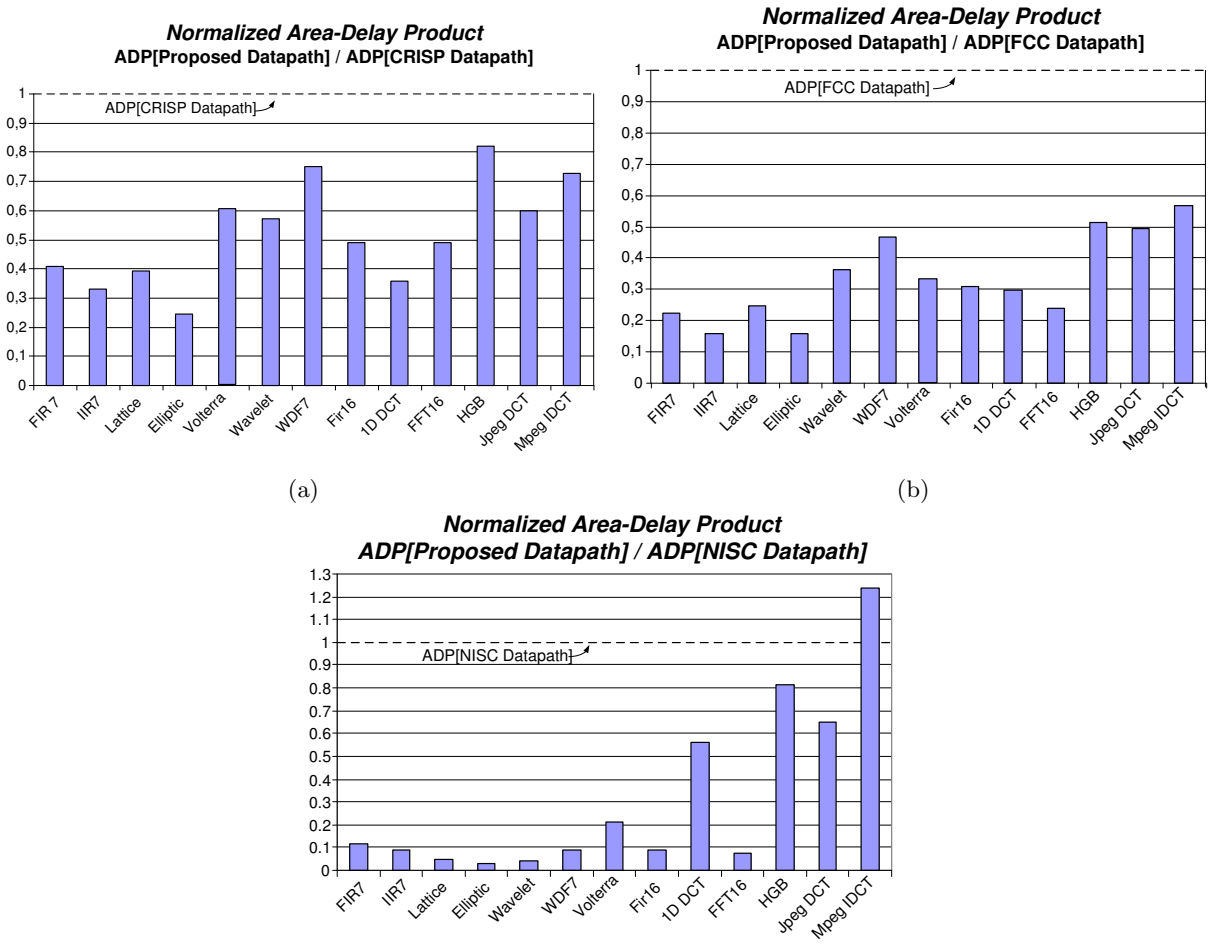


Figure 0.20.: Μετρική AD: Προτεινόμενη αρχιτεκτονική vs CRISP [15], b) Προτεινόμενη αρχιτεκτονική vs FCC [16], c) Προτεινόμενη Αρχιτεκτονική vs NISC [17].

στο AD είτε της CRISP είτε της FCC ή της NISC αρχιτεκτονικής. Σε σύγκριση με τις CRISP και FCC , η προτεινόμενη αρχιτεκτονική είναι πιο αποτελεσματική σε όλες τις περιπτώσεις, παρέχοντας λύσεις που προσφέρουν ταχύτερες υλοποιήσεις και μικρότερη επιφάνεια υλικού σε σχέση με τις άλλες δύο προσεγγίσεις. Πιο συγκεκριμένα, σε σύγκριση με την CRISP (Σχ. 0.20α), το AD κυμαίνεται μεταξύ 24,5% (Elliptic) και 82% (HGB). Σε σύγκριση με την FCC (Σχ. 0.20b), το AD κυμαίνεται μεταξύ 15,8% (IIR7 και Elliptic) και 56,5% (Mpeg IDCT). Σε σύγκριση με την NISC αρχιτεκτονική, η προτεινόμενη λύση παρέχει σημαντικά οφέλη AD για μια μεγάλη κατηγορία πυρήνες (FIR7, IIR7, Lattice, Elliptic, Wavelet, WDF7, Volterra, FIR16, FFT16) που κυμαίνονται μεταξύ 8,6% (IIR7) και 20,9% (Volterra). Σε εφαρμογές με δομές βρόχου (1D DCT, HGB, Jpeg DCT), η προτεινόμενη αρχιτεκτονική παρέχει AD τιμές που κυμαίνονται από 55,9% (1D DCT) έως 81,6% (HGB), σε σύγκριση την NISC. Για την περίπτωση του Mpeg IDCT,η προτεινόμενη λύση είναι λιγότερο αποτελεσματική από τη NISC, δεδομένου ότι ο συγκεκριμένος πυρήνας επηρεάζεται πολύ από τις βελτιστοποιήσεις κώδικα του NISC μεταγλωττιστή.

0.7. Αρχιτεκτονική Σύνθεση Ευέλικτων Συνεπεξεργαστών Υλικού Βάσει Αριθμητικά Βελτιστοποιημένης Αλυσίδωσης Λειτουργιών

Στην προηγούμενη ενότητα, παρουσιάσαμε μια προσέγγιση αρχιτεκτονικής σύνθεσης για το σχεδιασμό ευέλικτων DSP συνεπεξεργαστών υλικού, που ενσωματώνουν μονάδες πολλαπλασιαστή τύπου-πίνακα. Ωστόσο, η απεικόνιση σύνθετων υπολογισμών σε μονάδες υλικού που ενσωματώνουν πολλαπλασιαστές τύπου-δέντρου, έχει αποδειχθεί εξαιρετικά αποτελεσματική σε ότι αφορά τη βελτιστοποίηση της καθυστέρησης των DSP εφαρμογών. Στην ενότητα αυτή, προτείνουμε μια δεύτερη αρχιτεκτονική λύση και μεθοδολογία σύνθεσης που στοχεύουν στη σχεδίαση ευέλικτων αρχιτεκτονικών αδρομερούς επαναδιάταξης που ενσωματώνουν σύνθετες μονάδες με πολλαπλασιαστές τύπου-δένδρο.

Εξετάζοντας τις ευέλικτες αρχιτεκτονικές [16], [48], [49] αλλά και τις ASIC υλοποιήσεις [50], [7] υψηλών επιδόσεων, προτείνεται η χρήση σύνθετων δομικών μονάδων με αλυσίδωση λειτουργιών για την αποδοτική απεικόνιση σε υλικό των πρωτογενών αριθμητικών πράξεων της αλγοριθμικής περιγραφής. Δεδομένου ότι οι υπάρχουσες μεθοδολογίες αρχιτεκτονικής σύνθεσης βασίζονται στη χρήση αλγορίθμων βελτιστοποίησης που εφαρμόζονται στο επίπεδο αρχιτεκτονικής, οι παραπάνω μεθοδολογίες παράγουν αρχιτεκτονικές υλικού λαμβάνοντας υπόψη μόνο τα σχήματα διασύνδεσης των κόμβων στους DFG γράφους της αλγοριθμικής περιγραφής. Αυτό έχει σαν αποτέλεσμα να μην λαμβάνονται υπόψη τεχνικές βελτιστοποίησης από τα χαμηλότερα επίπεδα σχεδίασης π.χ. CS αριθμητική βελτιστοποίηση στο επίπεδο του κυκλώματος. Ο αποκλεισμός των αριθμητικών βελτιστοποιήσεων στο επίπεδο της αρχιτεκτονικής σύνθεσης έρχεται μαζί με τον εγγενή περιορισμό, ότι η καθυστέρηση των κρίσιμων μονοπατιών της αρχιτεκτονικής καθορίζεται από τις μεγάλες αλυσίδες διάδοσης κρατουμένου, που παρουσιάζει η σχεδίαση κυκλωμάτων στην τυπική δυαδική αριθμητική αναπαράσταση, π.χ. αριθμητική συμπληρώματος ως προς 2.

Προκειμένου να εξαλειφθούν αυτές οι αλυσίδες διάδοσης κρατουμένου, ορισμένες πρόσφατες ερευνητικές προσπάθειες, π.χ. [19], [18], [51], μελέτησαν την ενσωμάτωση της CS αριθμητικής βελτιστοποίησης στο επίπεδο αρχιτεκτονικής. Παρόλα αυτά, οι μελέτες αυτές εφαρμόζονται μόνο σε μη-ευέλικτες ASIC αρχιτεκτονικές. Οι μεθοδολογίες σχεδίασης που παρουσιάζονται σε αυτή την ενότητα έχουν στόχο την ανάπτυξη σύνθεση αρχιτεκτονικής που θα αξιοποιεί τεχνικές αριθμητικής βελτιστοποίησης σε σύνθετες επαναδιατάξιμες μονάδες υλικού για τη σχεδίαση ευέλικτων DSP συνεπεξεργαστών υλικού.

Ο κύριος στόχος της CS αριθμητικής βελτιστοποίησης είναι να μεγιστοποιήσει το εύρος των υπολογισμών οι οποίοι πραγματοποιούνται σε CS αριθμητική αναπαράσταση, ώστε να μειωθεί η καθυστέρηση στα κρίσιμα μονοπάτια του DFG γράφου της εφαρμογής. Η κοινή πρακτική στην εφαρμογή CS αριθμητικής βελτιστοποίησης στο επίπεδο της αρχιτεκτονικής σύνθεσης είναι η ανάπτυξη μετασχηματισμών αναδιάταξης των λειτουργιών του DFG, προκειμένου να επιτευχθούν κατάλληλες ομαδοποιήσεις πρωτογενών λειτουργιών (αθροίσεις, αφαιρέσεις) του DFG ώστε να υπολογιστούν σε CS μονάδες υλικού (CS συμπιεστές).

Κάθε φορά που ο υπολογισμός δεν μπορεί να εκτελεστεί στην CS αριθμητική ανα-

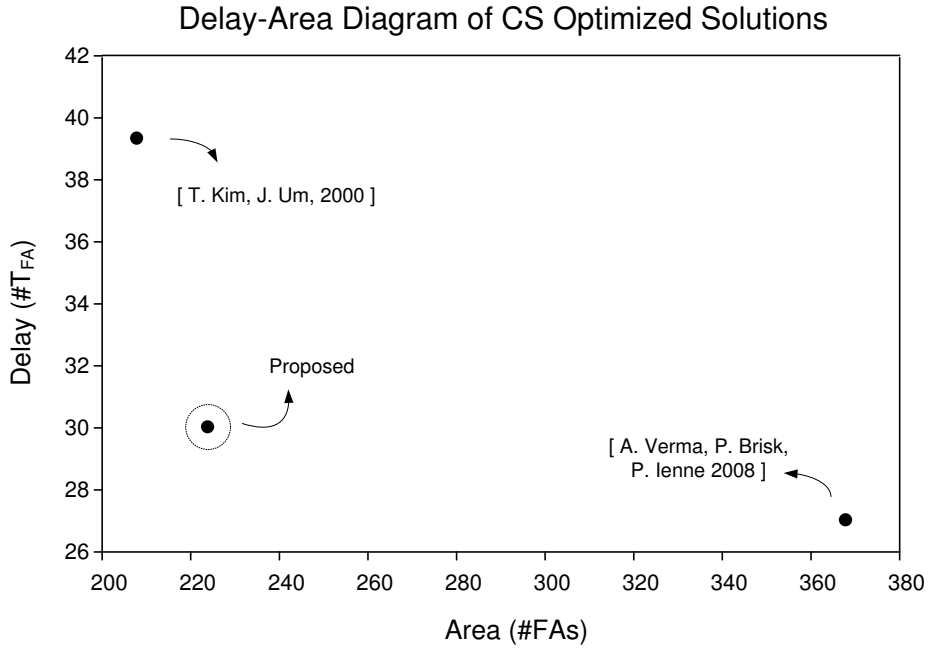


Figure 0.21.: Τοποθέτηση της προτεινόμενης λύσης σε σχέση με τις υπολοιπες τεχνικές CS βελτιστοποιήσεις [18], [19].

παράσταση, π.χ. όταν παρεμβάλλεται κάποιος λογικός τελεστής, η μια πράξη πολλαπλασιασμού, γίνεται η προσθήκη μιας λειτουργίας άθροισης με μεταφορά κρατουμένου για τη μετατροπή των δεδομένων από τη CS αριθμητική αναπαράσταση σε αριθμητική αναπαράσταση συμπληρώματος ως προς 2 (CS2Bin λειτουργία). Πρόσφατα, οι Verma [19] και Kim [18] πρότειναν ένα σύνολο κανόνων για το μετασχηματισμό του αρχικού DFG σε έναν σημασιολογικά ισοδύναμο γράφο οπού ένας αυξημένος αριθμός λειτουργιών έχει απεικονιστεί σε CS συμπιεστές. Τέτοιου είδους μετασχηματισμοί εφαρμόζονται πριν από τη σύνθεση και μπορούν να χρησιμοποιηθούν άμεσα από την προσέγγισή που προτείνουμε.

Οι παραπάνω τεχνικές στοχεύουν στην αναδιάταξη λειτουργιών άθροισης ώστε να μη παρεμβάλλονται ενδιάμεσα λογικοί τελεστές π.χ. ολίσθηση, λογικό KAI κ.τ.λ. Έτσι, η αποδοτικότητα τους περιορίζεται όταν εφαρμόζονται σε DFG γράφους που κυριαρχούνται από λειτουργίες πολλαπλασιασμού, π.χ. σε εφαρμογές DSP. Πιο συγκεκριμένα, οι Kim και άλλοι [18] αντιμετωπίζουν τις πράξεις πολλαπλασιασμού σαν συνοριακές λειτουργίες και εισάγουν έναν κατάλληλο αριθμό λειτουργιών CS2Bin, αυξάνοντας τη καθυστέρηση του DFG. Οι Verma και άλλοι [19] προτείνουν τη χρήση της επιμεριστικής ιδιότητας, επιμερίζοντας τον CS πολλαπλασιασμό σε δύο, με αποτέλεσμα να αυξάνεται αισθητά η επιφάνεια υλικού του κυκλώματος. Στην παρούσα διδακτορική διατριβή, προτείνουμε τη χρήση τεχνικών αριθμητικής επανα-κωδικοποίησης, [52], προκειμένου να επιτύχουμε σχεδιαστικές λύσεις με πιο αποδοτικά trade-off καθυστέρησης και επιφάνειας υλικού (Σχ. 0.21).

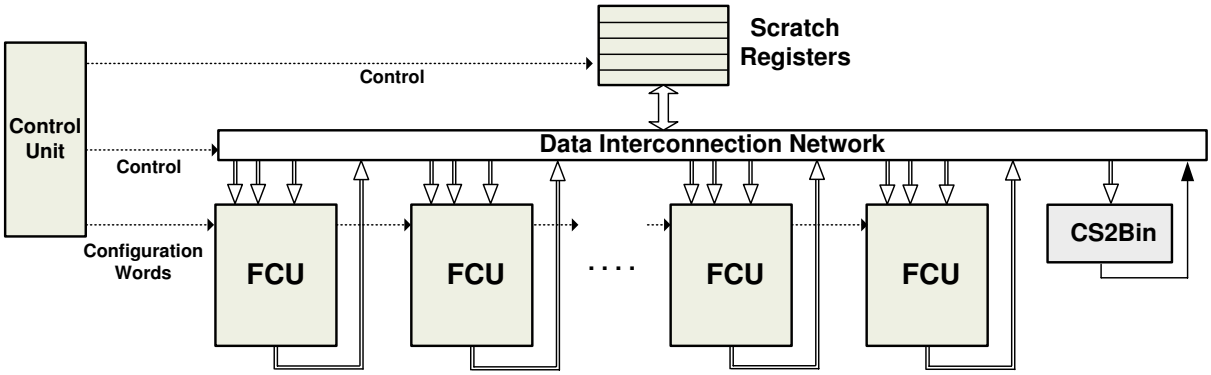


Figure 0.22.: Η προτεινόμενη ευέλικτη αρχιτεκτονική .

0.7.1. Προτεινόμενη Ευέλικτη Αρχιτεκτονική

Το Σχ. 0.22 απεικονίζει ένα αφαιρετικό μοντέλο της προτεινόμενης ευέλικτης αρχιτεκτονικής. Αποτελείται από πέντε δομικά στοιχεία: (1) τις ευέλικτες μονάδες υπολογισμού (Flexible Computational Units, FCU), (2) το μητρώο καταχωρητών (3) το δίκτυο διασύνδεσης, (4) τη μονάδα που μετατρέπει τα δεδομένα από την CS στη δυαδική αριθμητική αναπαράσταση (CS2Bin) και (5) τη μονάδα ελέγχου που οδηγεί τη συνολική αρχιτεκτονική (οδήγηση σημάτων επιλογής και διαμόρφωσης). Η προτεινόμενη αρχιτεκτονική υποθέτει μήκος λέξης δεδομένων των 16-bit και υιοθετεί ένα μοντέλο υπολογισμού στο οποίο τα σήματα ελέγχου παρέχονται από τη μονάδα ελέγχου και τα δεδομένα φορτώνονται στο μητρώο καταχωρητών πριν από τη φάση εκτέλεσης.

Η προτεινόμενη αρχιτεκτονική είναι σε θέση να εκτελεί υπολογισμούς σε δεδομένα σε CS αριθμητική αναπαράσταση ή σε αριθμητική αναπαράσταση συμπληρώματος ως προς 2. Με τον τρόπο αυτό, οι υπολογισμοί εκτελούνται σε CS και η μετατροπή των δεδομένων από CS σε συμπλήρωμα ως προς 2 λαμβάνει χώρα στο τέλος κάθε εκτέλεσης του πυρήνα. Ο αριθμός των FCUs καθορίζεται κατά τη διάρκεια σχεδιασμού σύμφωνα με το επιθυμητό οριζόντιο παραλληλισμό και τους περιορισμούς υλικού που θέτει ο σχεδιαστής. Το μητρώο καταχωρητών αναλαμβάνει την αποθήκευση των ενδιάμεσων αποτελεσμάτων και την ανταλλαγή και επικοινωνία των μεταβλητών μεταξύ των FCUs. Το δίκτυο διασύνδεσης αναλαμβάνει την επικοινωνία μεταξύ του μητρώου καταχωρητών και των FCUs.

Η εσωτερική δομή του FCU έχει σχεδιαστεί για να επιτρέπει ευέλικτη αλυσίδωση λειτουργιών σε υψηλές συχνότητες λειτουργίας. Η αλυσίδωση λειτουργιών επιτρέπει τη χρονοδρομολόγηση πολλών εξαρτημένων πρωτογενών λειτουργιών στον ίδιο κύκλο ρολογιού, εξαλείφοντας την εγγραφή των ενδιαμέσων αποτελεσμάτων σε κάποιον καταχωρητή. Με τον τρόπο αυτό επιτυγχάνεται βελτίωση τόσο της συνολικής καθυστέρησης όσο και της επιφάνεια υλικού της αρχιτεκτονικής. Η έννοια της ευέλικτης αλυσίδωσης λειτουργιών επιτρέπει το διαμοιρασμό υλικού μεταξύ σύνθετων μονάδων υλικού.

Το Σχ. 0.23 απεικονίζει ένα αφηρημένο μοντέλο της προτεινόμενης μονάδας FCU. Φαίνεται, επίσης, η εσωτερική δομή του κόμβου FCU (λεπτομερές μοντέλο), καθώς

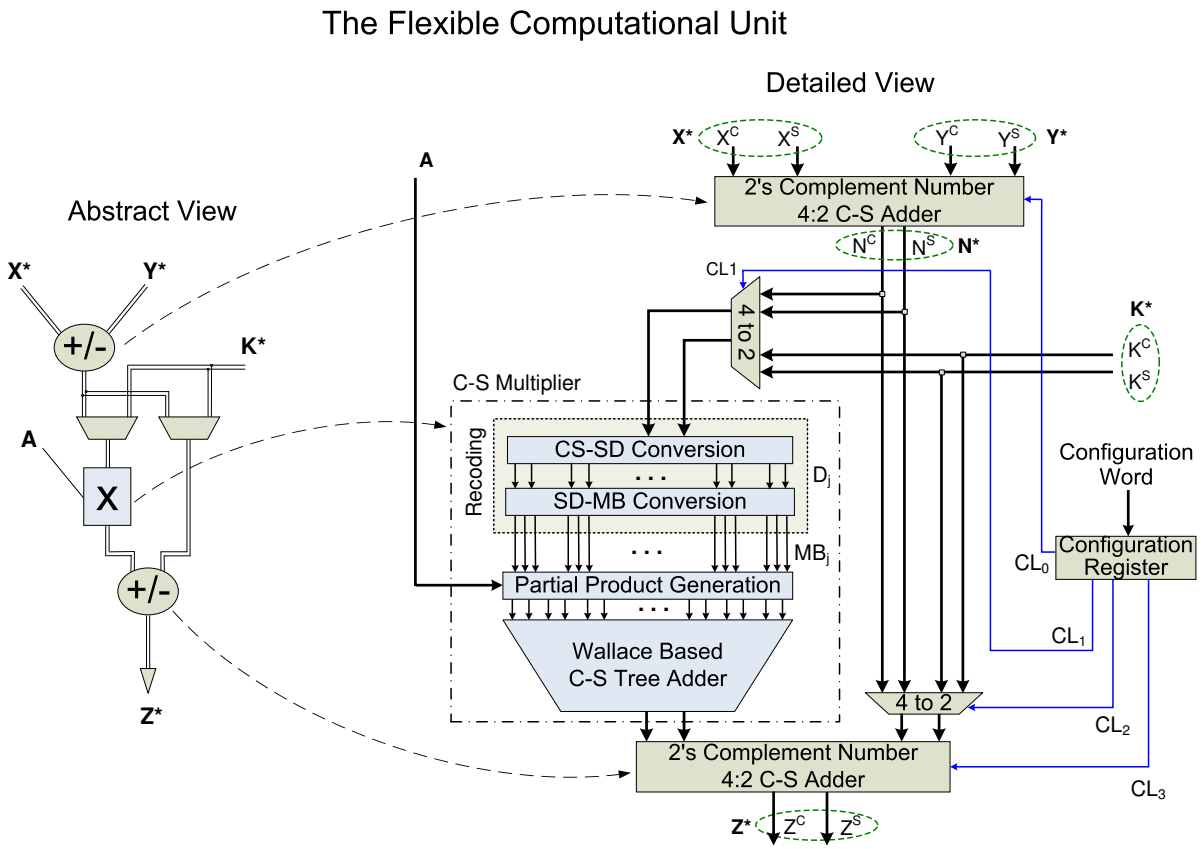


Figure 0.23.: Η μονάδα FCU.

και ο τρόπος που το αφηρημένο μοντέλο απεικονίζεται πάνω στο λεπτομερές. Ο υπολογισμός εντός της FCU επιτελείται από CS κυκλώματα. Κάθε FCU ενσωματώνει (1) δύο μονάδες 4:2 CS συμπιεστών που επιτρέπουν λειτουργίες πρόσθεσης και αφαίρεσης να εκτελεστούν, ελεγχόμενα από τα σήματα ελέγχου CL_i, πριν και μετά τη λειτουργία του πολλαπλασιασμού, (2) έναν πολλαπλασιαστή τύπου-δεντρο (γεννήτορας μερικών γινομένων και μια δομή δεντρικού αθροιστή) για γρήγορη εκτέλεση λειτουργιών πολλαπλασιασμού και (3) κατάλληλα κυκλώματα αριθμητικής επανα-κωδικοποίησης προκειμένου ένα CS δεδομένο να χρησιμοποιείται ως είσοδος στον πολλαπλασιαστή τύπου-δέντρο χωρίς να χρειάζεται να μετατραπεί σε συμπλήρωμα ως προς 2. Το κύκλωμα αριθμητικής επανα-κωδικοποίησης μετατρέπει έναν CS αριθμό σε ένα σύνολο ψηφιών Modified Booth (MB) [53]. Το κύκλωμα επανα-κωδικοποίησης υλοποιείται σύμφωνα με το [52]. Επιτυγχάνει την μετατροπή CS σε MB ψηφία με πολύ μικρή καθυστέρηση ανεξάρτητη του μήκους λέξης. Πιο συγκεκριμένα, η καθυστέρηση επανακωδικοποίησης είναι σε κάθε περίπτωση ίση με την καθυστέρηση 2 κελιών πλήρους αθροιστή (Full Adder, FA) σε σειρά. Σημειώνεται πως η καθυστέρηση μετατροπής CS σε συμπλήρωμα ως προς 2, με χρήση αθροιστών διάδοσης-κρατουμένου ισούται με N, όπου N είναι το μήκος λέξης των δεδομένων. Περισσότερες λεπτομέρειες σχετικά με τη σχεδίαση του κυκλώματος επανα-κωδικοποίησης δίνονται στο κυρίως κείμενο της διατριβής.

Μέσω της εκτενούς χρήσης CS τελεστών καθώς και της ενσωμάτωση του κυκλώματος

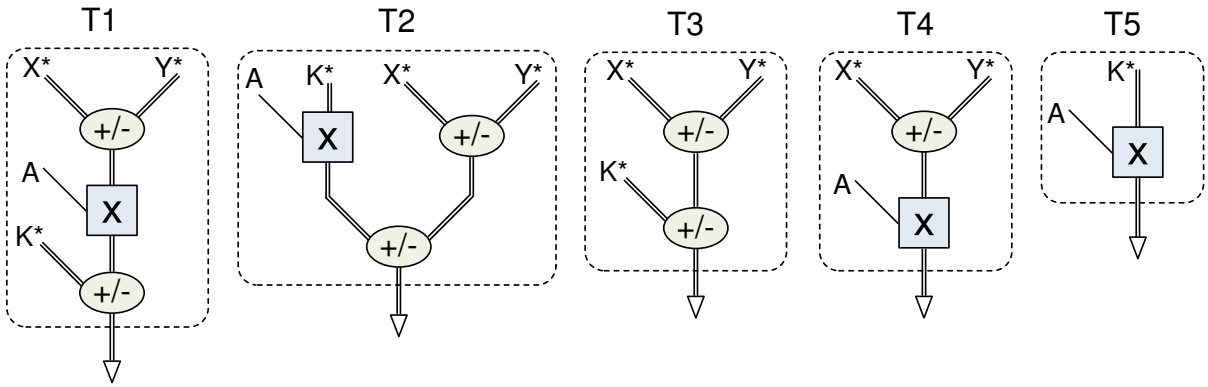


Figure 0.24.: Η βιβλιοθήκη λειτουργιών της μονάδας FCU.

αριθμητικής επανα-κωδικοποίησης από CS σε MB , το κάθε FCU επιτυγχάνει τον υπολογισμό σύνθετων λειτουργίων, σε πολύ υψηλές συχνότητες λειτουργίας. Επιπλέον, η ευελιξία που προσφέρεται από την εσωτερική οργάνωση του κάθε FCU επιτρέπει τον υπολογισμό ενός συνόλου σύνθετων λειτουργιών, όπως αυτές που φαίνονται στο Σχ. 0.24. Η ευελιξία αυτή μπορεί να αξιοποιηθεί για την απεικόνιση υπολογιστικά απαιτητικών εφαρμογών. Επομένως, το σύνολο της προτεινόμενης αρχιτεκτονικής αποτελεί μια αποδοτική λύση για αρχιτεκτονικές συνεπεξεργάστη DSP εφαρμογών.

0.7.2. Μεθοδολογία Αρχιτεκτονικής Σύνθεσης

Για την αποδοτική απεικόνιση DSP εφαρμογών πάνω στην προτεινόμενη αρχιτεκτονική, αναπτύχθηκε μια εξειδικευμένη μεθοδολογία αρχιτεκτονικής σύνθεσης. Η συνολική ροή της προτεινόμενης μεθοδολογίας απεικονίζεται στο Σχ. 0.25. Αποτελείται από 3 φάσεις, οι οποίες περιγράφονται συνοπτικά παρακάτω:

Φάση 1: Ο DFG γράφος της εφαρμογής εξάγεται από την αλγοριθμική περιγραφή εισόδου. Οι δείκτες διευθυνσιοδότησης των διαφόρων πινάκων δεδομένων προυπολογίζονται στατικά και αντικαθιστώνται με ισοδύναμες μεταφορές μεταξύ των καταχωρητών της αρχιτεκτονικής. Με τον τρόπο αυτό, οι μονάδες FCU επικεντρώνονται στους υπολογισμούς του DFG. Εξάλλου, η προτεινόμενη αρχιτεκτονική παράγει δεδομένα στην CS αριθμητική παράσταση, η οποία είναι δεν είναι αποτελεσματική για αριθμητική δεικτών. Επιπλέον, ο DFG γράφος ξεδιπλώνεται ανάλογα με τον αριθμό των λειτουργιών πολλαπλασιασμού και των διαθέσιμων μονάδων FCU. Με τον τρόπο αυτό, ο οριζόντιος παραλληλισμός εξισορροπείται με το βαθμό χρησιμοποίησης του υλικού, δεδομένου ότι οι πολλαπλασιαστές είναι οι μονάδες με την μεγαλύτερη επιφάνεια υλικού σε κάθε FCU.

Φάση 2: Σε αυτή τη φάση επιτελείται η απεικόνιση των πρωτογενών λειτουργιών του DFG σε CS λειτουργίες που μπορούν να εκτελεστούν από τις μονάδες FCU. Αρχικά, ο γράφος DFG μετασχηματίζεται σε έναν DFG με CS λειτουργίες. Στη συνέχεια, οι CS λειτουργίες του μετασχηματισμένου DFG ομαδοποιούνται σε μεγαλύτερα μπλοκ, ώστε να απεικονιστούν στα στοιχεία της βιβλιοθήκης λειτουργιών

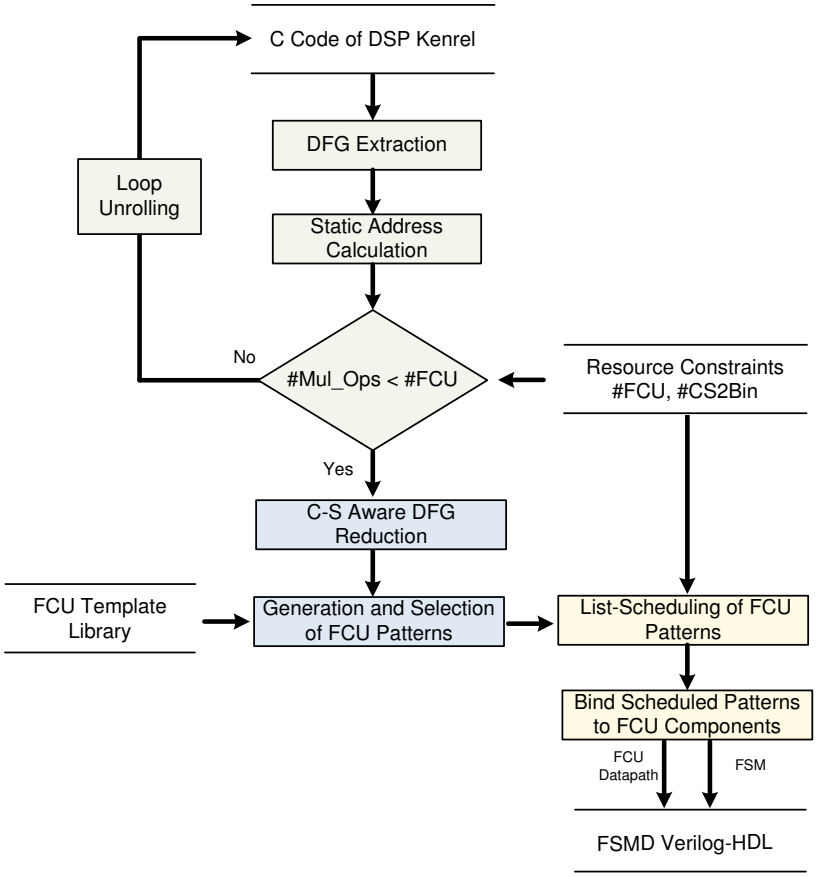
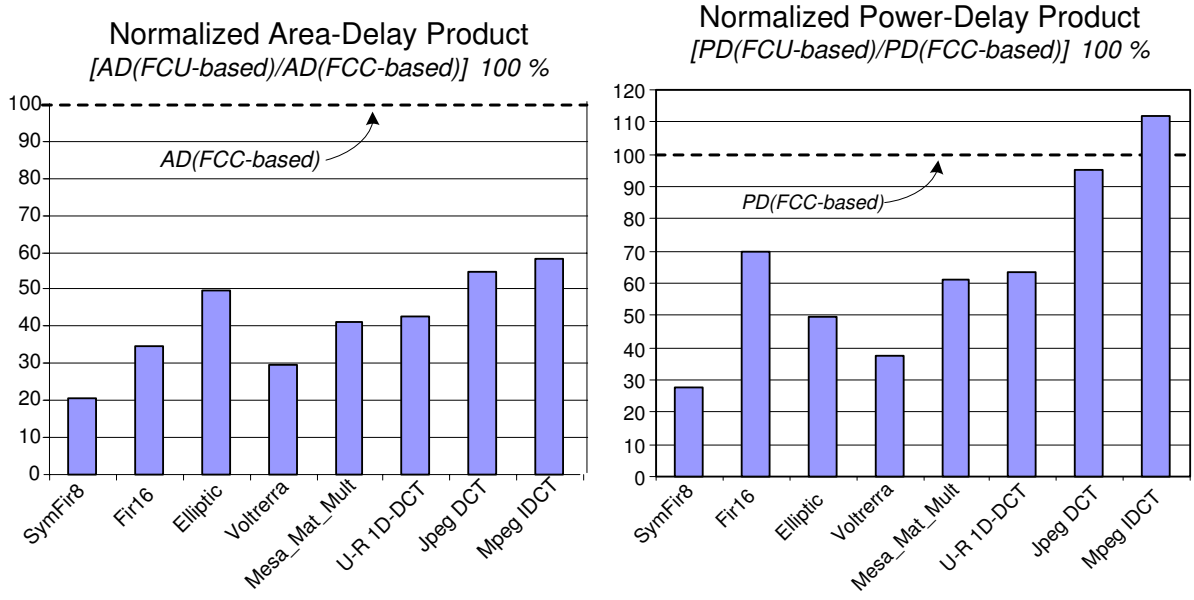


Figure 0.25.: Προτεινόμενη ροή σχεδίασης.

του **FCU**, με κριτήριο την ελαχιστοποίηση της καθυστέρησης του κρίσιμου μονοπατιού. Η απεικόνιση σε **CS** λειτουργίες του **FCU** εκμεταλλεύεται το εγγενές χαρακτηριστικό των **CS** κυκλωμάτων τα οποία συγχωνεύουν πολλαπλές λειτουργίες άθροισης/αφαίρεσης σε μια [54]. Με τον τρόπο αυτό, το μέγεθος του **DFG** συρρικνώνεται, προσφέροντας ευκαιρίες για ταχύτερη χρονοδρομολόγηση.

Φάση 3: Στη φάση αυτή, ο μετασχηματισμένος **DFG** χρονοδρομολογείται βάσει των περιορισμών στους πόρους υλικού (αριθμός μονάδων **FCU** και **CS2Bin**) ενός συγκεκριμένου αρχιτεκτονικού στιγμιότυπου. Η συνάρτηση χρονοδρομολόγησης υλοποιεί έναν χρονοδρομολογητή τύπου λίστας [44] ο οποίος αξιοποιεί τον οριζόντιο παραλληλισμό της εφαρμογής και αξιοποιεί την κινητικότητα λειτουργιών και τον αριθμό των διαθέσιμων μονάδων **FCU**. Στη συνέχεια οι χρονοδρομολογημένες λειτουργίες ανατίθενται στις μονάδες **FCU**,παράγονται τα κατάλληλα σήματα ελέγχου και εξάγεται η συνθέσιμη περιγραφή της αρχιτεκτονικής.



(a) (b)

Figure 0.26.: Σύγκριση μετρικών σχεδίασης 1) Area-Delay b) Power-Delay.

0.7.3. Ενδεικτικά Αποτελέσματα Πειραματικής Αξιολόγησης

Η ενότητα αυτή παρουσιάζει κάποια ενδεικτικά πειραματικά αποτελέσματα για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης προσέγγισης. Περαιτέρω πειραματικά αποτελέσματα μπορούν να βρεθούν στο κυρίως κείμενο της διατριβής. Η πειραματική αξιολόγηση που παρουσιάζεται στην ενότητα αυτή αφορά στην αποδοτικότητα της προτεινόμενης αρχιτεκτονικής για DSP εφαρμογές που απεικονίστηκαν μέσω της προτεινόμενης μεθοδολογίας αρχιτεκτονικής σύνθεσης.

Ένα σύνολο από υπολογιστικά απαιτητικούς πυρήνες DSP απεικονίστηκαν. Συγκεκριμένα, εξετάστηκαν οι εξής δύο αρχιτεκτονικές λύσεις: (1) η προτεινόμενη ευέλικτη αρχιτεκτονική που περιλαμβάνει τέσσερις μονάδες FCU και μία μονάδα CS2Bin, (2) μια αρχιτεκτονική τύπου FCC [16] με δύο κόμβους FCC .

Συγκρίνουμε την αποδοτικότητα της προτεινόμενης προσέγγισης βάσει του γινομένου των μετρικών επιφάνειας υλικού και καθυστέρησης λειτουργίας (Area-Delay, AD) και του γινομένου των μετρικών κατανάλωσης ισχύος και καθυστέρησης λειτουργίας (Power-Delay, PD).Στο Σχ. 0.26a το γινόμενο AD της προτεινόμενης αρχιτεκτονικής απεικονίζεται σε σχέση με το αντίστοιχο γινόμενο AD της FCC. Το Σχ. 0.26b δείχνει τη σύγκριση της μετρικής PD σε σχέση με τη FCC. Οι τιμές των AD και PD έχουν κανονικοποιηθεί ως προς με τα αντίστοιχα γινόμενα AD και PD της αρχιτεκτονικής. Επομένως, η διακεκομμένη γραμμή (AD/PD = 1) αναφέρεται στο AD/PD της FCC αρχιτεκτονικής. Οι τιμές AD της προτεινόμενης σε σχέση με την FCC αρχιτεκτονική κυμαίνονται μεταξύ 20,8%-58,4%. Το αντίστοιχο εύρος τιμών για την μετρική PD κυμαίνεται μεταξύ 27,7% -111% (περίπτωση MPEG).

0.8. Συμβολή Διδακτορικής Διατριβής

Η διδακτορική διατριβή ''Μεθοδολογίες Αυτοματοποιημένης Εξερεύνησης και Σύνθεσης Επαναδιατάξιμων Συνεπεξεργαστών Υλικού'' προτείνει ένα σύνολο μεθοδολογιών για αποδοτική εξερεύνηση των σχεδιαστικών χώρων που προκύπτουν από το διαχωρισμό της εφαρμογής σε συνιστώσες λογισμικού-υλικού, και αποδοτική σχεδίαση επαναδιατάξιμων αρχιτεκτονικών υλικού για απεικόνιση εφαρμογών ψηφιακής επεξεργασίας σήματος.

Συγκεκριμένα, η συμβολή της διατριβής συνοψίζεται στα παρακάτω:

1. Επιτεύχθηκε η ανάλυση σε βασικές συνιστώσες, των αποφάσεων του χώρου σχεδίασης που αφορά στη δυναμική διαχείριση μνήμης πολυπύρηνων υπολογιστικών συστημάτων με διαμοιραζόμενη ιεραρχία μνήμης. Προτάθηκαν νέοι ευριστικοί αλγόριθμοι εξερεύνησης και διάσχισης του παραπάνω χώρου σχεδίασης βάσει μεθόδων ορθογώνιου διαχωρισμού των αποφάσεων, οι οποίοι επιτρέπουν την πολύ-κριτηριακή βελτιστοποίηση και την αυτοματοποιημένη παραγωγή λογισμικού δυναμικών διαχειριστών μνήμης εξειδικευμένων στην εκάστοτε εφαρμογή.

2. Προτάθηκε μια νέα θεώρηση για την παραμετρική μοντελοποίηση του χώρου σχεδίασης συνεπεξεργαστών υλικού, η οποία λαμβάνει υπόψη τις επιδράσεις των σχεδιαστικών αποφάσεων από τα επίπεδα αφαίρεσης συμπεριφοράς/αλγορίθμου και αρχιτεκτονικής, με στόχο τη συνδυασμένη βελτιστοποίηση της επίδοσης και της επιφάνειας υλικού. Επιπροσθέτως, προτάθηκαν νέοι ευριστικοί αλγόριθμοι εξερεύνησης του παραπάνω χώρου σχεδίασης, οι οποίοι επιτυγχάνουν γρήγορη αναζήτηση με αποδοτική σύγκλιση στις καθολικά βέλτιστες σχεδιαστικές λύσεις.

3. Προτάθηκε η τεχνική Εισαγωγή Ευελιξίας για τη κυκλωματική σχεδίαση επαναδιατάξιμων αρχιτεκτονικών αδρομερούς υφής. Η προτεινόμενη τεχνική εκμεταλλεύεται την κατοπτρική συμμετρία στις διασυνδέσεις εξειδικευμένων (μη-επαναδιατάξιμων) αριθμητικών κυκλώματων και μέσω κατάλληλων μετασχηματισμών στο επίπεδο του bit επιτυγχάνει τη σχεδίαση του επαδιατάξιμου ισοδύναμου κυκλώματος.

4. Βάσει της τεχνικής Εισαγωγή Ευελιξίας, προτάθηκε ένα νέο επαναδιατάξιμο μικρο-αρχιτεκτονικό πρότυπο συνεπεξεργαστή το οποίο επιτρέπει σε επίπεδο υλικού τη συνδυασμένη εκμετάλλευση των αρχιτεκτονικών βελτιστοποιήσεων: οριζόντιου παραλληλισμού, κάθετου παραλληλισμού και αλυσίδωσης λειτουργιών με χαμηλή κυκλωματική πολυπλοκότητα.

5. Στη συνέχεια προτάθηκαν νέοι αλγόριθμοι σχεδίασης απο υψηλό επίπεδο για την αυτοματοποιημένη απεικόνιση εφαρμογών από το επίπεδο συμπεριφοράς σε κυκλωματικές υλοποιήσεις των επαναδιατάξιμων αρχιτεκτονικών που προέκυψαν από την τεχνική Εισαγωγή Ευελιξίας. Οι προτεινόμενοι αλγόριθμοι επιτυγχάνουν τη συνδυασμένη αξιοποίηση των αρχιτεκτονικών, κυκλωματικών και χωροθετικών χαρακτηριστικών του επαναδιατάξιμου συνεπεξεργαστή.

6. Βάσει της τεχνικής Εισαγωγή Ευελιξίας, προτάθηκε ένα νέο επαναδιατάξιμο μικρο-αρχιτεκτονικό πρότυπο συνεπεξεργαστή το οποίο επιτρέπει σε επίπεδο

υλικού την συνδυασμένη εκμετάλλευση των αρχιτεκτονικών βελτιστοποιήσεων: οριζόντιου παραλληλισμού, κάθετου παραλληλισμού και αλυσίδωσης λειτουργιών με χαμηλή κυκλωματική πολυπλοκότητα.

7. Προτάθηκε μια νέα επαναδιατάξιμη αρχιτεκτονική συνεπεξεργαστή η οποία βασίζεται στη χρήση μεθόδων αριθμητικής βελτιστοποίησης για σχεδίαση ευέλικτων χειριστών δεδομένων με αλυσίδωση λειτουργιών. Επιπλέον, προτάθηκε αλγόριθμος μετασχηματισμού συμπεριφορικών περιγραφών κώδικα σε ισοδύναμες αριθμητικά βελτιστοποιημένες περιγραφές, οι οποίες επιτρέπουν την αυτοματοποίηση της απεικόνισης εφαρμογών στην προτεινόμενη επαναδιατάξιμη αρχιτεκτονική.

0.9. Συμπεράσματα και Μελλοντικές Προεκτάσεις

Στην ενότητα αυτή, παρουσιάζονται συνοπτικά τα κύρια συμπεράσματα που προέκυψαν από την εφαρμογή των προτεινόμενων μεθοδολογιών σε αντιπροσωπευτικά σενάρια ελέγχου. Επαληθεύσαμε το συμπέρασμα που έχει αναφερθεί και σε άλλες συγγενείς ερευνητικές προσπάθειες, πως υπάρχει ισχυρή εξάρτηση μεταξύ των αλγόριθμων σχεδίασης και της αποτελεσματικότητας της τελικής σχεδιαστικής λύσης. Ωστόσο, το πιο βασικό συμπέρασμα αυτής της εργασίας είναι ότι η αποτελεσματικότητα της σχεδιαστικής λύσης είναι ισχυρώς συσχετισμένη, εκτός των αλγορίθμων σχεδιασμού, τόσο από την πληρότητα του οριζόμενου χώρου των παραμέτρων εξερεύνησης όσο και από τις αλληλεπιδράσεις μεταξύ των διαφόρων επιπέδων αφαίρεσης. Ενώ στην τυπική περίπτωση, οι αλγόριθμοι σχεδιασμού προτείνουν τεχνικές για αποδοτική διάσχιση ενός συγκεκριμένου επιπέδου αφαίρεσης, η προσέγγιση αυτή περιορίζει το σύνολο των παραμέτρων προς διερεύνηση με αποτέλεσμα να μην οδηγεί προς καθολικά βέλτιστες λύσεις λόγω του υψηλού κατακερματισμού του χώρου σχεδίασης. Προκειμένου να ξεπεραστούν τέτοιου είδους περιορισμοί, οι σχεδιαστικοί αλγόριθμοι θα πρέπει να εξελιχθούν, ώστε να λαμβάνουν υπόψη πιο πολύπλοκους χώρους παραμέτρων καθώς και δυνατότητες βελτιστοποίησης που παρουσιάζονται σε περισσότερα του ενός επίπεδα αφαίρεσης. Φυσικά, η ανάπτυξη σχεδιαστικών αλγορίθμων που θα εφαρμόζονται σε ένα εντελώς απο-κατακερματισμένο χώρο σχεδίασης δεν είναι ακόμη εφικτή, κυρίως λόγω των τεράστιων χώρων παραμέτρων και λύσεων που δημιουργούνται αλλά και λόγω των δυνατοτήτων των σημερινών υπολογιστικών συστημάτων. Επομένως, η παρούσα διατριβή προτείνει τεχνικές σχεδίασης που επιτρέπουν την ανάπτυξη αλγορίθμων εξερεύνησης και βελτιστοποίησης που στοχεύουν όχι σε ένα τελείως από-κατακερματισμένο χώρο σχεδίασης, αλλά σε ένα χώρο σχεδιασμού με πιο αδρομερή κατακερματισμό σε σχέση με τις προηγούμενες ερευνητικές προσεγγίσεις.

0.9.1. Συμπεράσματα

Πιο συγκεκριμένα, όσον αφορά στη διαχείριση δυναμικής μνήμης πολύ-νηματικών εφαρμογών, δείξαμε ότι υπάρχει ισχυρή εξάρτηση μεταξύ των μηχανισμών που υλοποιούν τη δομή/αρχιτεκτονική λογισμικού ενός διαχειριστή δυναμικής μνήμης και των μετρικών κόστους, όπως το ίχνος μνήμης, ο αριθμός των προσβάσεων, το

μέγεθος κώδικα του διαχειριστή κ.λ.π. Στην περίπτωση που το σύνολο των εφαρμογών είναι a-priori γνωστό στο σχεδιαστή, όπως στην περίπτωση των ενσωματωμένων συστημάτων, η εξειδίκευση της δομής του λογισμικού του διαχειριστή οδηγεί σε βελτιστοποιημένες λύσεις. Επιπροσθέτως, η αρχιτεκτονική του λογισμικού διαχείρισης της δυναμικής μνήμης μπορεί να εξειδικευτεί βάσει ενός συνόλου ορθογώνιων σχεδιαστικών αποφάσεων, οι οποίες ορίζουν τον αντίστοιχο χώρο παραμέτρων της εξερεύνησης.

Οι σχεδιαστικές αποφάσεις οργανώνονται ιεραρχικά σε δύο ορθογώνιους υπό-χωρους παραμέτρων, (1) στον δια-νηματικό χώρο παραμέτρων ο οποίος περιλαμβάνει τις καθολικές σχεδιαστικές αποφάσεις που καθορίζουν την αρχιτεκτονική του δυναμικού διαχειριστή, λαμβάνοντας υπόψη το σύνολο των νημάτων της εφαρμογής και (2) στον ενδό-νηματικό χώρο παραμέτρων ο οποίος περιλαμβάνει τοπικές σχεδιαστικές αποφάσεις που καθορίζουν την αρχιτεκτονική συγκεκριμένων τμημάτων του δυναμικού διαχειριστή λαμβάνοντας υπόψη το υποσύνολο των νημάτων που διαχειρίζονται τη συγκεκριμένη δεξαμενή μνήμης. Δείξαμε ότι υπάρχει ισχυρή συσχέτιση τόσο μεταξύ των παραμέτρων του ίδιου υπο-χώρου, όσο και μεταξύ των δύο υπο-χώρων και πως οι συσχετισμοί αυτοί πρέπει να λαμβάνονται υπόψη προκειμένου να καθοριστούν οι κατάλληλες ιεραρχήσεις των σχεδιαστικών αποφάσεων ανάλογα με το επιδιωκόμενο κριτήριο βελτιστοποίησης. Δεδομένου του τεράστιου χώρου σχεδίασης, η ανάλυση των παραπάνω συσχετισμών λειτουργεί επικουρικά στην αποτελεσματική αυτοματοποίηση της εξερεύνησης. Οι συσχετισμοί αυτοί μπορούν να θεωρηθούν ως σημασιολογικοί περιορισμοί οι οποίοι διαδίδονται στο χώρο του σχεδιασμού και επιτρέπουν την ανάπτυξη αποτελεσματικών μηχανισμών εξάλειψης των σημασιολογικά μη-έγκυρων λύσεων, μειώνοντας δραστικά το μέγεθος του χώρου λύσεων προς διερεύνηση χωρίς να υποβαθμίζουν την ποιότητα του αποτελέσματος της εξερεύνησης. Επιπλέον, οι διαφορετικές ιεραρχήσεις αποφάσεων μπορούν να αξιοποιηθούν για την ανάπτυξη ευριστικών αλγορίθμων για ικανοποιητική διάσχιση του χώρου λύσεων με λογικές καθυστερήσεις. Τέλος, τα πειραματικά δεδομένα έδειξαν ότι ο χώρος των λύσεων, έχει αρκετά αόριστη δομή που καθιστά εξαιρετικά δύσκολο ακόμη και για έναν έμπειρο σχεδιαστή να περιηγηθεί σε αυτόν στοχευμένα, προς τις Pareto βέλτιστες λύσεις, χωρίς τη βοήθεια εργαλείων αυτοματισμού.

Σχετικά με το θεμελιώδες πρόβλημα εξερεύνησης στη σύνθεση αρχιτεκτονικής, αυτό της εύρεσης των Pareto λύσεων με κριτήρια βελτιστοποίησης την καθυστέρηση λειτουργίας και την καταλαμβανόμενη επιφάνεια υλικού, εντοπίσαμε την ισχυρή συσχέτιση των παραμέτρων μεταξύ του αλγοριθμικού (επίπεδο μεταγλωττιστή) και αρχιτεκτονικού χώρου παραμέτρων. Παρόλο που οι αλγοριθμικοί μετασχηματισμοί στο επίπεδο του μεταγλωττιστή έχουν μελετηθεί εκτενώς στον τομέα της αρχιτεκτονικής σύνθεσης, παρατηρήσαμε πως οι τελευταίοι διερευνώνται συνήθως αυτόνομα οδηγούμενοι από το σχεδιαστή, με αποτέλεσμα να μην μελετάται με έναν καθολικό τρόπο η αλληλεπίδραση τους με τις αρχιτεκτονικές αποφάσεις. Δείξαμε ότι η υιοθέτηση των προσεγγίσεων αυτών έχει σαν αποτέλεσμα να μην διερευνάται ένα σημαντικό τμήμα σχεδιαστικών λύσεων, το οποίο συνήθως περιλαμβάνει το Pareto σύνορο λύσεων. Προκειμένου να αντιμετωπιστεί αυτή η σχεδιαστική ανεπάρκεια, επιχειρήσαμε τη μοντελοποίηση ενός επαυξημένου χώρου παραμέτρων, ο οποίος λαμβάνει υπόψη συνδυασμένα τις αποφάσεις στο επίπεδο των αλγοριθμικών μετασχηματισμών όσο και τις αποφάσεις στο επίπεδο αρχιτεκτονικής. Σε αντίθεση με το χώρο σχεδίασης που μελετήθηκε στην περίπτωση της δυναμικής διαχείρισης μνήμης, ο συγκεκριμένος χώρος σχεδίασης δεν χαρακτηρίζεται από τόσο ισχυρές εξαρτήσεις και διάδοση περιορισμών μεταξύ των παραμέτρων του ώστε να επιτρέπει τον έγκαιρο

προσδιορισμό των σημασιολογικά μη-έγκυρων λύσεων. Λόγω του παραπάνω χαρακτηριστικού γνωρίσματος του συγκεκριμένου σχεδιαστικού χώρου, προτείναμε μια στρατηγική εξερεύνησης η οποία εφαρμόζεται στο χώρο λύσεων και όχι στο χώρο παραμέτρων. Ωστόσο, ακόμη και σε αυτή την περίπτωση των χαλαρών εξαρτήσεων στο χώρο παραμέτρων, δείξαμε πως μέσω κατάλληλης ιεράρχησης των παραμέτρων επιτυγχάνεται η αποσύνθεση του τεράστιου χώρου λύσεων, σε μικρότερους υποχώρους που μπορούν να διερευνηθούν πιο αποτελεσματικά. Με βάση αυτή την αποσύνθεση του χώρου λύσεων, αποδείξαμε ότι υπάρχουν άνω οριακές συνθήκες για κάθε υπο-χώρο λύσεων. Επιπλέον, δείξαμε ότι ο αποσυντεθημένος χώρος λύσεων έχει μια καλώς ορισμένη δομή, η οποία αξιοποιείται κατά τη διάρκεια εξερεύνησης μέσω ενός καινούργιου ευριστικού αλγορίθμου, που βασίζεται στην κλίση που παρουσιάζουν τα σημεία του κάθε υπο-χώρου. Η πειραματική αξιολόγηση έδειξε πως, η προτεινόμενη στρατηγική εξερεύνησης επιτυγχάνει τη μετατόπιση της καμπύλης εξερεύνησης (οι Pareto λύσεις που προκύπτουν μετά το τέλος μιας εξερεύνησης) προς πιο αποδοτικές σχεδιαστικές λύσεις σε σύγκριση με τις υπάρχουσες μεθοδολογίες, με αποτελεσματική σύγκλιση προς το καθολικό Pareto σύνορο και με σημαντικά κέρδη όσον αφορά στη διάρκεια της εξερεύνησης (έως 8× φορές ταχύτερα) σε σύγκριση με την εξαντλητική διερεύνηση.

Όσον αφορά στη σχεδίαση αρχιτεκτονικών αδρομερούς επαναδιάταξης μελετήσαμε τις δυνατότητες βελτιστοποίησης που παρουσιάζονται, όταν η αρχιτεκτονική προδιαγράφεται λαμβάνοντας υπόψη τόσο το μικρο-αρχιτεκτονικό όσο και το κυκλωματικό επίπεδο αφαίρεσης. Δείξαμε ότι η υιοθετώντας μια συνδυασμένη οπτική των δύο επίπεδων αφαίρεσης καθιστά δυνατή τη σχεδίαση επαναδιατάξιμων αρχιτεκτονικών που επιτρέπουν το διαμοιρασμό υλικού τόσο μεταξύ των δομικών μονάδων όσο και μεταξύ πιο βασικών κυκλωματικών δομών στο επίπεδο του bit, παρέχοντας πιο αποτελεσματικές λύσεις ευέλικτων κυκλωμάτων, σε ότι αφορά στην καταλαμβανόμενη επιφάνεια υλικού και του βαθμού επαναχρησιμοποίησης του. Επιπροσθέτως, επιτρέπει την εφαρμογή αριθμητικών βελτιστοποιήσεων στη φάση του σχεδιασμού της μικροαρχιτεκτονικής, με αποτέλεσμα τη δημιουργία ευέλικτων κυκλωμάτων υψηλών επιδόσεων. Από τη μελέτη αυτή, προέκυψαν δύο νέα επαναδιατάξιμα αρχιτεκτονικά πρότυπα για εφαρμογές που συναντώνται στο πεδίο της ψηφιακής επεξεργασίας σήματος. Πιο συγκεκριμένα, η συνδυασμένη μελέτη των δυο επιπέδων αφαίρεσης (μικρο-αρχιτεκτονικό και κυκλωματικό) και των βελτιστοποιήσεων που αυτά προσφέρουν, επιτρέπει τη σχεδίαση ευέλικτων δομικών μονάδων υλικού υψηλών επιδόσεων, χαμηλής κυκλωματικής πολυπλοκότητας και μεγάλης λειτουργικής πυκνότητας. Επιπλέον, αναγνωρίζοντας τα χαρακτηριστικά των υψηλών επιδόσεων που προσφέρει η αριθμητική Carry-Save, ενσωματώθηκαν προηγμένες τεχνικές αριθμητικής επανακωδικοποίησης που επιτρέπουν το μη-διακοπτόμενο υπολογισμό στη συγκεκριμένη αριθμητική αναπαράσταση. Επιπλέον, δείξαμε ότι η από κοινού αξιοποίηση πολλαπλών αρχιτεκτονικών βελτιστοποιήσεων, πιο συγκεκριμένα του οριζόντιου, του κάθετου παραλληλισμού και της αλυσίδωσης υπολογιστικών λειτουργιών, μαζί με βελτιστοποιήσεις στο επίπεδο της αριθμητικής, σχηματίζει μια πολλά υποσχόμενη λύση για αρχιτεκτονικές αδρομερούς επαναδιάταξης. Αυτή η ενοποιημένη οπτική των επιπέδων αφαίρεσης μικρο-αρχιτεκτονικής και κυκλώματος, προκαλεί αυξημένη πολυπλοκότητα στα μοντέλα προγραμματισμού που υποστηρίζονται από τις υπάρχουσες μεθοδολογίες αρχιτεκτονικής σύνθεσης. Αν και οι υπάρχουσες μεθοδολογίες αρχιτεκτονικής σύνθεσης είναι αρκετά αποτελεσματικές για αρχιτεκτονικές που δομούνται από συμβατικά μοντέλα μονάδων υλικού, αποτυγχάνουν στο να υποστηρίξουν βελτιστοποιημένες λύσεις για αρχιτεκτονικές που υιοθετούν πιο πολύπλοκα μοντέλα μονάδων. Προτείναμε τη διαχείριση της αυξημένης πολυπλοκότητας του μοντέλου

προγραμματισμού στα υψηλά επίπεδα αφαίρεσης, ώστε να επιτραπεί η επαναχρησιμοποίηση καθιερωμένων αλγορίθμων και εργαλείων σύνθεσης που προέρχονται από το πεδίο της αρχιτεκτονικής σύνθεσης ASIC. Για την αντιμετώπιση του προβλήματος, αναπτύξαμε κατάλληλους μετασχηματισμούς ώστε να επιτρέπει η απεικόνιση μοντέλων ενδιάμεσης αναπαράστασης που χρησιμοποιούνται σε εργαλεία αρχιτεκτονικής σύνθεσης ASIC πάνω στο πιο πολύπλοκο μικρο-αρχιτεκτονικό μοντέλο που εισάγουν τα προτεινόμενα επαναδιατάξιμα αρχιτεκτονικά πρότυπα. Η πειραματική αξιολόγηση σε επίπεδο κυκλώματος έδειξε ότι οι προτεινόμενες ευέλικτες αρχιτεκτονικές είναι σε θέση να λειτουργούν σε ένα σημαντικά μεγαλύτερο φάσμα των συχνοτήτων ρολογιού και με χαμηλότερη γενικά επιφάνεια από τα τις υπάρχουσες αρχιτεκτονικές αδρομερούς επαναδιάταξης. Επιπλέον, βάσει ενός αντιπροσωπευτικού συνόλου εφαρμογών ψηφιακής επεξεργασίας σήματος, δείξαμε ότι η προτεινόμενη προσέγγιση προσφέρει σημαντικές βελτιώσεις τόσο σε ταχύτητα εκτέλεσης των εφαρμογών όσο και σε επιφάνεια υλικού που καταλαμβάνει το κύκλωμα του επαναδιατάξιμου συνεπεξεργαστή, σε σχέση με τις υπάρχουσες προσεγγίσεις.

0.9.2. Μελλοντικές Προεκτάσεις

Η παρούσα διδακτορική διατριβή προτείνει ένα σύνολο καινοτόμων μεθοδολογιών σχεδίασης με στόχο να επιτρέψει στους σχεδιαστές να μετακινηθούν προς βελτιστοποιημένες λύσεις συστημάτων. Τα εργαλεία σχεδίασης που υλοποιούν τις προτεινόμενες μεθοδολογίες καθώς και τα θεωρητικά και ποσοτικά συμπεράσματα που εξήχθησαν από την παρούσα διατριβή μπορούν να χρησιμοποιηθούν ως βάση για την αντιμετώπιση νέων προβλημάτων σχεδίασης που προκύπτουν από τις νέες τεχνολογικές προκλήσεις. Επιγραμματικά, οι ακόλουθες ερευνητικές επεκτάσεις αναγνωρίζονται, σε σχέση με τις υπόλοιπες ερευνητικές δραστηριότητες που έχουν αναπτυχθεί στα πλαίσια του εργαστηρίου Μικρο-υπολογιστών και Ψηφιακών Συστημάτων.

- Επέκταση των εξειδικευμένων διαχειριστών δυναμικής μνήμης με μηχανισμούς που θα επιτρέπουν προσαρμοστικότητα της λειτουργίας τους κατά τη διάρκεια εκτέλεσης προκειμένου να διασφαλιστεί η αξιοπιστία λειτουργίας σε δυναμικά περιβάλλοντα.

- Συνδυασμένη εξερεύνηση των παραμέτρων στο επίπεδο της δυναμικής διαχείρισης μνήμης και στο επίπεδο των μικρο-αρχιτεκτονικών χαρακτηριστικών της πλατφόρμας υλικού προκειμένου να διερευνηθούν οι αλληλεπιδράσεις μεταξύ των παραμέτρων λογισμικού και υλικού.

- Συνδυασμένη εξερεύνηση αλγοριθμικών-αρχιτεκτονικών-θερμικών παραμέτρων με στόχο την αποφυγή προβλημάτων αξιοπιστίας του υλικού στις χαμηλές τεχνολογίες ολοκλήρωσης λόγω της αυξημένης πυκνότητας ισχύος.

- Ανάπτυξη μεθοδολογιών ενσωμάτωσης των αρχιτεκτονικών αδρομερούς επαναδιάταξης σε επαναδιατάξιμα συστήματα λεπτομερούς επαναδιάταξης, ώστε να αξιοποιηθούν τα χαρακτηριστικά και των δύο τεχνολογιών σε μια ενιαία πλατφόρμα υλικού.

1. Introduction

A prudent question is one-half of wisdom.

Francis Bacon

Electronic Design Automation (EDA) is a continuously growing research area in the field of Integrated-Circuit (IC) design. A large number of researchers and engineers from both the academia and industry have extensively work on EDA problems during the last decades [55]. This interest for efficient EDA methodologies is based on the enormous advances in system integration, which still follows very closely the Moore's Law [23]. The research outcome of this PhD Thesis is a set of design methodologies spanning across the front-end design layers and targeting to aid designers to move towards optimized system solutions. This introductory chapter serves as a guide for the reader to understand the targeted research field and the thesis organization.

The context of this thesis is delimited through four design axes which also define the targeted research space. Specifically, embedded systems form the first axis, which actually determines the targeted hardware platform (Section 1.1). The second axis concerns the automated exploration of the solution spaces occurred during the design procedure, which enables the designers to quickly evaluate various design trade-offs and to reveal higher quality design configurations (Section 1.2). The third axis concerns the architectural synthesis of reconfigurable datapaths which serves as a second optimization layer enabling both higher performance through hardware acceleration and lower implementation costs through advanced hardware sharing between dedicated coprocessors (Section 1.3). The last axis refers to a wider design flow that forms a globalized view of the system design procedure, playing the role of an "umbrella" environment including and instantiating the methodologies proposed in this dissertation (Section 1.4).

1.1. Embedded Systems

Embedded systems are a special class of computing devices that are tailored to the needs of certain applications or application domains i.e. mobile phones, digital cameras, anti-block breaks etc. Despite with the general purpose computing devices i.e. PCs, laptops, embedded systems are focused on the execution of predefined tasks [56]. Furthermore, they exhibit high dynamic behavior through the continuous interaction (sensing and reaction) with the system's environment. Due to their application specific nature, designers are seeking for design methodologies that are able to customize the final system configuration in such a manner that the performance constraints are met under the lowest manufactural cost (silicon area, power and energy budget).

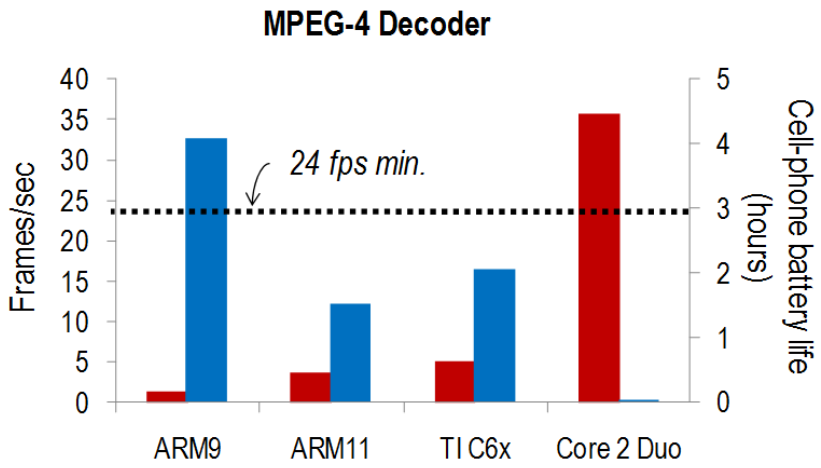


Figure 1.1.: Need for Customization.

System customization is a requirement generated from the increased complexity of embedded applications. The conventional approach of software customization for single processor embedded devices are now far away from modern application's requests. Fig. 1.1 depicts a case study based on the Moving Picture Experts Group (MPEG) Decoder [24] application (found in a large set of embedded systems) to reason about why customization is essential for modern embedded architectures. The tested application has been mapped onto four different computing devices, incorporating an ARM9 processor [25], an ARM11 processor [25] (ARM customized for DSP operations), a C6x VLIW processor from Texas Instruments [26] and a general purpose Intel Core 2 Duo platform [27], respectively. The evaluation is performed taking under consideration the performance and energy metrics. The quality of service (QoS) constraint that the system has to guarantee is the 24 frames per second. The left bar in Fig. 1.1 depicts the achieved frame rate while the right bar depicts the energy consumption in terms of battery life. As shown, software only solutions fail to satisfy the performance constraints of the application, despite the general purpose multi-core architecture, which satisfies the constraint in the expense of very low energy efficiency. In order to satisfy such strict performance constraints keeping the system's implementation in acceptable energy budgets, a synergetic software (in terms of supported parallelization) and hardware (in terms of hardware acceleration) customization approach has to be considered.

Motivated from the previous observations, this thesis targets embedded system architectures that are able to support software and hardware customization, like the ones illustrated in Fig. 1.2. The targeted system template consists of (i) a set of programmable microprocessors (μP) for task concurrency, (ii) a set of dedicated and/or reconfigurable hardware accelerators (HW ACC), (iii) a communication network for interconnecting the various modules and (iv) a multi-level memory hierarchy for storage.

***Processing Subsystem*: The programmable microprocessors are responsible to execute the software coefficients (assembly instructions of their assigned software tasks) in a concurrent manner. The computationally intensive software tasks that form performance bottlenecks of the application are characterized as hardware coefficients which are further mapped onto either the dedicated or reconfigurable hardware accelerators through High Level Synthesis**

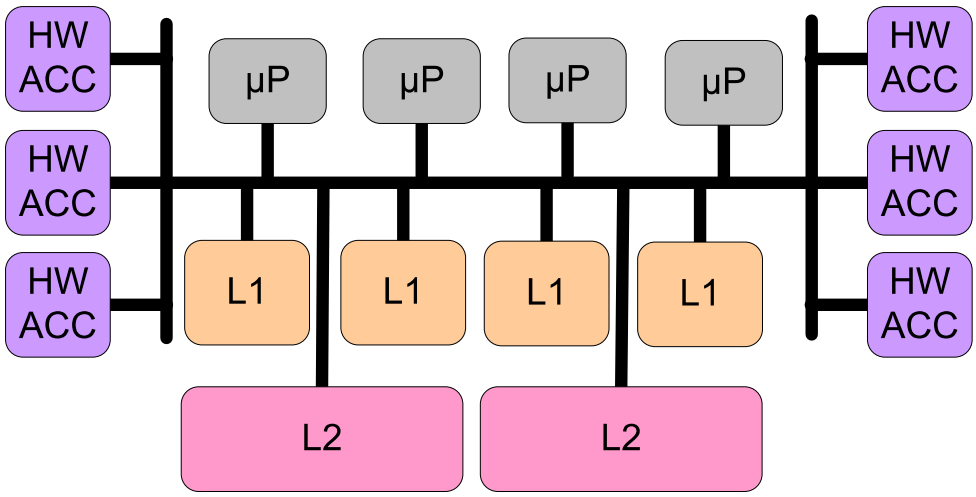


Figure 1.2.: Targeted System Architecture.

(HLS) [57] tools. Although, dedicated accelerators are different from the reconfigurable ones, we considered them together in this point of our presentation driven by the fact that both modules are used for acceleration of the computationally intensive tasks. In the second part of this thesis, we present reconfigurable accelerator architectures that are able to map onto the same silicon the functionality of a large set of dedicated accelerators improving the area utilization and cost.

***Memory Subsystem*: The multi-level memory hierarchy concerns the memory organization for data and instruction storage [58]. Multi-level memory organization enables the small portions of data that are stored in the off-chip memory to be stored also inside the chip in order to reduce access latencies. In modern computing architectures the on-chip memory hierarchy is organized in two levels (L1 and L2 memories). L2 memories has a larger size than the L1 memories with slower access times and higher power consumption. Regarding the memory subsystem, this thesis is focused on the customization of the software mechanisms for a specific class of memory management, that of dynamic memory management in multi-threaded and multi-core environments. Thus, we do not focus on the customization of the memory hierarchy itself, but on the customization of the memory management mechanisms for the targeted system architectures.**

***Network Subsystem*: Finally, the interconnection network is responsible for the data transfers between memory system and the internal registers of the programmable microprocessors and the hardware accelerators. Shared bus and Network-on-Chip interconnection schemes are the most common interconnection structure. It is important to be mentioned that the interconnection architecture is a system's resource that can be also customized to the application specific needs in an orthogonal way than the rest of the architecture [41]. The methodologies and architectures presented in this thesis are independent from the underlying interconnection scheme, thus we leave interconnection customization out of the scope of our investigation.**

1.2. Design Space Exploration

Design space exploration (DSE) is the procedure of investigating multiple implementation variants in order to conclude in an optimal solution, in case of a single objective cost function is assumed. In case that multiple optimization objectives are considered, i.e. area cost, delay, power for datapath synthesis, or memory footprint, fragmentation, memory accesses for dynamic memory management, a set of Pareto-optimal solutions exists (see Definition 1) rather than a single optimal one. These objectives usually exhibits tight connections between each other, thus optimizing with a single objective in mind may reveal severe trade-offs with respect to the other objectives. In fact, a system configuration, which is the best from the performance point of view, can be the worst in terms of power consumption and vice versa. In order to evaluate one solution whether it is Pareto-optimal with respect to a set of solutions, all objective values of the design must exhaustively be compared with the corresponding objective values of every other design in the set.

Definition 1. *Pareto criterion for dominance.* Given k objectives to be minimized without loss of generality and two solutions A and B with values $(a_0, a_1, ..., a_{k-1})$ and $(b_0, b_1, ..., b_{k-1})$ for all objectives, respectively, solution A dominates solution B if and only if $\forall i \in \{0 \leq i \leq k$: $a_i \leq b_i\}$ and $\exists j : a_i < b_i$.

That means, a superior solution is at least better in one objective while being at least the same in all other objectives. A more rigorous definition of *strict dominance* requires A to be better in all objectives compared to B, whereas the less strong definition of *weak dominance* only requires the condition $\forall i \in \{0 \leq i \leq k$: $a_i \leq b_i\}$.

Definition 2. *Pareto-optimal solution.* A solution is called Pareto-optimal if it is not dominated by any other solution. Non-dominated solutions form a Pareto-optimal set (exact Pareto front) in which neither of the solutions is dominated by any other solution in the set.

An example illustrating the notion of Pareto dominance is illustrated in Fig. 1.3. We assume a design space with two objectives for minimization, area-cost (x-axis) and execution time of a design (y-axis). All design points included in the 2D-plot form the feasible region. Thus, without further insights into the design problem, all depicted design points are reasonable solutions. The four connected design points forms the Pareto front. The rest of the design points are dominated by the Pareto-optimal ones.

In general, design space exploration problems are formulated through (i) the *parameter space*, (ii) the *solution space* and (iii) the *mapping function*. The *parameter space* is defined by properties of the design problem that do not represent immediate design objectives, but natural characteristics of the design space. In the context of DSE, the dimensions of the problem space often coincide with the axes-parameters of the design space. For instance, a dynamic memory manager can be described by a set of fitting policies, coalescing mechanisms etc. that do not give any insights into primary objectives, such as the memory footprint consumed or the performance of the allocator. The *solution space* is defined by the primary objectives of the design space exploration, i.e. system cost, speed, and power dissipation. Finally, the *mapping function* is the function that maps the configurations defined in the *parameter space* to the corresponding instances in the solution space. Simulators, analytical models, high level models, compilation-execution phases, even

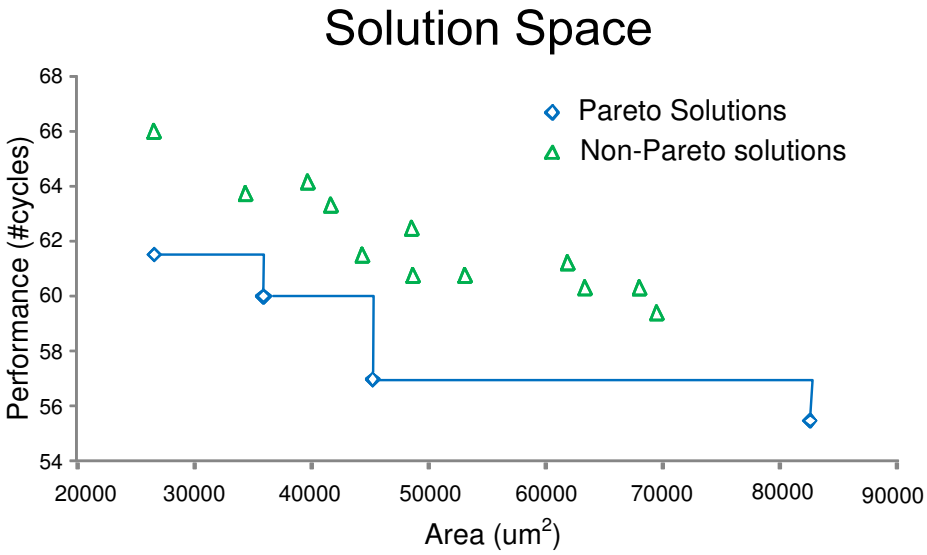


Figure 1.3.: Pareto dominance.

synthesis-implementation phases can be used as valid mapping functions in a exploration framework [29].

Exploration approaches work either to the *parameter space* or to the *solution space*. In the former case, the exploration algorithm (i) systematically chooses system configurations instantiated in the *parameter space*, (ii) evaluates the selected configurations in order to check their feasibility and (iii) extracts the Pareto set of the examined set. In the later case, the exploration algorithm systematically constrains the set of feasible designs, i.e., the algorithm determines during exploration whether a suitable design can be found that satisfies the constraints while optimizing other objectives.

In system design, the multi-objective exploration case forms the most realistic scenario, since the designer has to deliver optimized solutions within specific objective budgets. Actually, each design task included in a system implementation flow can be considered as a separate multi-objective exploration problem. In such design problems the complexity of the exploration process depends on the size of the examined solution space. Given that arbitrary parameter bindings usually make the solution space quite large, an exhaustive evaluation of all possible parameter combinations quickly becomes intractable. In addition, randomly sampling the design space has no guarantee that the exploration converge to the exact Pareto front, leading to potential highly suboptimal solutions. Consequently, there is a need for automated and disciplined exploration methodologies that incorporate knowledge on the structure of the design space without searching the design space extensively, in order to converge to an approximate Pareto set close to the exact Pareto front.

In this thesis, we focus on two design space exploration problems faced during system synthesis, after the system specification has been partitioned into its corresponding software and hardware coefficients (see Section 1.4). For both exploration problems thorough theoretical analysis is provided and the corresponding automated exploration and synthesis tools have been developed to provide a seamless path from exploration down to implemen-

tation.

The first exploration problem occurs during software synthesis and concerns the development of customized services for dynamic memory management (Chapter 2). The targeted exploration problem aims to deliver a systematic way for generating Pareto-optimal and application-specific dynamic memory management software for multi-threaded applications mapped onto multi-core platforms. The specific exploration approach works on the parameter space exploiting dependencies among the parameters of the design space in order to early prune non-optimal configurations. To the best of author knowledge this is the first time that both systematic parameter modeling and exploration algorithms are attempted for customized *multi-threaded* dynamic memory management. Previous work on dynamic memory management exploration has been proposed only for single-threaded applications running onto single processor embedded platforms [30].

The second exploration problem occurs during hardware synthesis of the coefficients selected for acceleration (Chapter 3). The targeted problem aims to explore the Delay-Area trade-offs delivering Pareto coprocessor implementations close to the exact Pareto-optimal ones. Although there have been already proposed exploration algorithms targeting the specific trade-offs, the existing approaches for Delay-Area exploration regarding coprocessor synthesis provide suboptimal solutions, due to the fact that the impact of behavioral- and architectural-level parameters is not explored in a combined manner. In other words, the interaction between the structure of behavioral code and the architectural optimizations are silently assumed to be independent. We leverage this limiting factor by modeling an extended parameter space including decisions from both behavioral and architectural abstraction levels and by proposing an exploration strategy which handles the overall parameter space in a combined manner. The specific exploration approach works on the solution space exploiting knowledge on the parameter space to prune non-optimal design solutions during the exploration's runtime.

1.3. Architectural Synthesis of Reconfigurable Datapaths

1.3.1. Reconfigurable Architectures

The advent of Reconfigurable Computing (RC) [31] has generated a whole new research field in the area of digital design. The new computational model augments the available logical density of the circuits, binding the spatial (high parallelism) and the temporal (high programmability-flexibility) models. Along with the high integration densities provided by the current VLSI technologies, designers are able to design configurable System-on-Chips (SoCs) [59], [60]. Reconfigurable hardware is ideal for use in SoCs as it executes applications with the efficiency of dedicated hardware accelerators, and yet maintains a level of flexibility not available with more traditional full custom circuitry. Hardware flexibility enables cost efficient SoC implementations due to increased opportunities for coprocessor level hardware reuse. Thus, on the area allocated for a single reconfigurable coprocessor more than one application kernels are mapped, rather than requiring a separate custom circuit for each one.

Fig. 1.4 compares the various implementation technologies in terms of efficiency (performance, power) versus supported flexibility. By performance, we mean the amount of clock

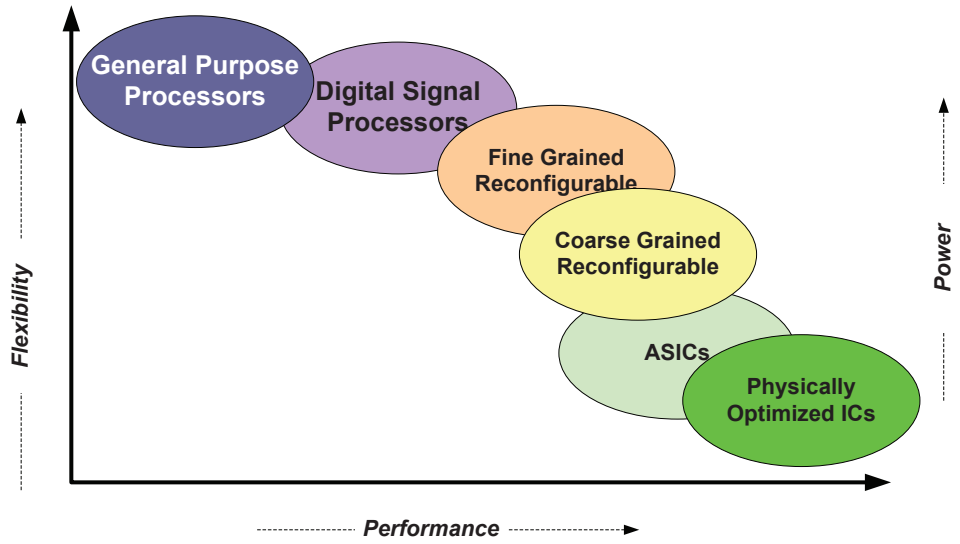


Figure 1.4.: Positioning of the available technologies.

cycles it takes to run a specific application program under certain specified conditions, while flexibility refers to the programmability of the system. The depicted trend is that increasing hardware flexibility has a negative impact of the performance metric, or from a different point of view customizing the hardware to the application leads to performance gains. On the one side, there are the programmable processors (general-purpose or Digital Signal Processor (DSP) which exhibit the highest flexibility in the expense of relatively poor performance, whereas on the completely opposite side the application specific circuitry (ASIC and physically optimized ICs) is placed, which is very performance efficient, but too inflexible. Reconfigurable architectures make a trade-off between these two extremes cases, thereby limiting their flexibility to a particular algorithm domain.

A significant number of reconfigurable architectures have already been proposed, varying mostly on the granularity's degree. An overview of the most popular reconfigurable architectures can be found in [32]. Fine-grained reconfigurable architectures, such as classical Field Programmable Gate Arrays (FPGAs), favor bit-level operations and mapping universality, but suffer from high reconfiguration delays and power consumption. Furthermore, this flexibility has its place for situations where the computational requirements are either not known in advance or vary considerably among the needed applications. However, in many cases this extreme level of flexibility is unnecessary and would result in significant overheads of area, delay and power consumption. Coarse-grained reconfigurable architectures eliminate the disadvantages of fine-grained ones and preserve universality at most cases, but operate only on word-length data formats. In coarse-grained architectures the reconfigurable unit is a specialized hardware that supports logic reconfiguration. The reconfiguration procedure is much faster than that found in FPGAs. In addition, given the application domain, reconfigurable data paths customized to the application's domain specific requirements could be designed, being drastically more area- and power- efficient than fine-grained implementations.

In this thesis we target coarse-grained reconfigurable architectures for applications found in the field of Digital Signal Processing (DSP). DSP applications are dominated by many

computational intensive kernels performing a large number of arithmetic operations. We are mainly interested on the Delay-Area trade-offs delivered by these arithmetic operations in DSP datapath implementations. In such datapaths, the multiplication units form the major area and performance "hungry" component due to their high complexity and large critical paths, respectively.

We focus our research on designing reconfigurable architectures exploiting the allocated area of multiplication units. Conventional coarse-grained reconfigurable datapaths are specified at the micro-architectural level by allocating a number of hardware resources i.e. arithmetic operators, which are shared between different configurations (functional-unit sharing). However, we follow a rather different design approach than the one usually used for coarse-grained architectures. Specifically, we identify that incorporating together the datapath views at the micro-architectural and bit-structural levels, the conventional functional-unit sharing can be complement with (i) bit-level sharing leading to more area efficient solutions and (ii) arithmetic level optimizations, i.e. computing in Carry-Save arithmetic representation, leading to more efficient datapaths.

Based on the above motivational observations, we present two new reconfigurable/flexible architectures: (i) one for datapaths invoking array-based multiplication units (Chapter 4) and (ii) the second one for datapaths invoking tree-based multiplication units (Chapter 5). The former case combines both bit-level sharing and arithmetic level optimizations. In particular, we introduce the Flexibility Inlining datapath transformation technique that enables the mapping of various arithmetic operators onto the regular structure of array-based multipliers, thus delivering high area reusability. The later case is focused mainly on the incorporation of arithmetic level optimizations. Specifically, a coarse-grained reconfigurable architecture is introduced based on functional units implementing flexible operation-templates with advanced arithmetic recoding techniques to enable computation to be maximally performed in CS representation, alleviating the large carry-propagation chains found in conventional implementations.

Although the high efficiency of the proposed reconfigurable datapaths, they introduce new micro-architectural characteristics that have to be taken into account during application mapping. Thus, the designer has to be aware on the micro-architectural details of each datapath solution in order to efficiently map the desired application. As in conventional software development, the awareness of the micro-architectural details limits the programmers/designers productivity. In order to tackle this limitation, software community is based on the extensive use of compiler infrastructures. Based on similar objectives, the hardware community has proposed the usage of architectural HLS frameworks. Given the new micro-architectural features of the proposed reconfigurable datapaths, the development of enhanced architectural synthesis methodologies is required in order to both abstract the micro-architectural details of the proposed datapaths and perform efficient application mapping.

1.3.2. Architectural Synthesis

Architectural synthesis, HLS, methodologies in the sense of high-level design entry tools, is essential for coarse-grained reconfigurable architectures to compete with application specific architectures. Architectural synthesis is the process of creating a structural micro-architectural level representation from a behavioral description of an application in a seamless

way. A structural representation defines an exact interconnection between a set of architectural resources i.e. datapath components (Arithmetic Logic Units (ALUs), multipliers (Muls)), memory components (registers, memory blocks), interconnection components (multiplexors, I/O), control units. The architectural synthesis problem can be formulated in the following manner: Given the algorithmic description of an application, a set of fully characterized architectural resources, a set of constraints and an optimization function, determine a fully connected set of resources that conforms to the given constraints and minimizes the objective function.

The algorithmic description refers to the set of operations, i.e. addition, multiplication etc., and the corresponding data and control dependencies that implement the desired behavior. Architectural synthesis can be split into two sub-problems: operation scheduling and resource binding. Operation scheduling is the procedure that determines the start time for each operation. The start times must follow the data-dependencies of the algorithmic description. Regarding the Delay-Area trade-offs, there are two variants of the scheduling problem (i) the resource-constrained and (ii) timing constrained scheduling [44]. Both scheduling variants have been proved to be NP-complete problems. In this thesis, we focus on the resource constrained scheduling variant, which is formulated as follows: Given the algorithmic description (operations and data-dependencies) and the maximum resource constraints, find a schedule that minimizes latency while satisfying the constraints. Resource binding is the procedure that assigns hardware resources to one or more operations. It greatly affects the area and latency of the circuit as it dictates the amount of steering logic and memory components of the circuit. The resource binding procedure may be performed before or after scheduling. It is most often performed after scheduling, so we focus on that methodology. In this case, the scheduling generates additional constraints on the possible resource bindings. For example, operations that are scheduled to execute during the same time step cannot share the same resource.

In the case of coarse-grained reconfigurable architectures, architectural synthesis refers to the mapping of the algorithmic description of the application kernel onto the elements of the reconfigurable datapath. In conventional coarse-grained architectures [33], the reconfigurable datapath consists of functional units that are similar with the functional-units in ASIC architectural synthesis. Thus, the adoption of existing architectural synthesis algorithms (scheduling-binding) is performed with little or none modification. This is not the case for the reconfigurable datapaths that this thesis introduces. That is due to two reasons. The first reason is that the incorporation of bit-level sharing together with arithmetic level optimization during architecture specification results in functional units with different resource models than the resources implementing primary operations i.e. 2-input-1-output addition, multiplication etc. The second reason is that the proposed datapaths are exhibiting interesting combinations of multiple micro-architectural features, i.e. joint horizontal/vertical parallelism and operation chaining, that have to exploited during architectural synthesis to deliver optimized solutions.

In this dissertation, each of the introduced reconfigurable datapaths is complemented with a properly tuned architectural synthesis methodology. By the term "properly tuned", we refer to our attempt to reuse existing and well establish knowledge on the field architectural synthesis, thus not reinventing the wheel. However, new synthesis algorithms and pre-synthesis (source-to-source) transformation techniques are proposed wherever this was considered crucial for guaranteing the high quality of the delivered datapath solutions. Regarding pre-synthesis transformation techniques, we propose a set of refinement algorithms for transforming the high-level algorithmic description based on primitive resource models

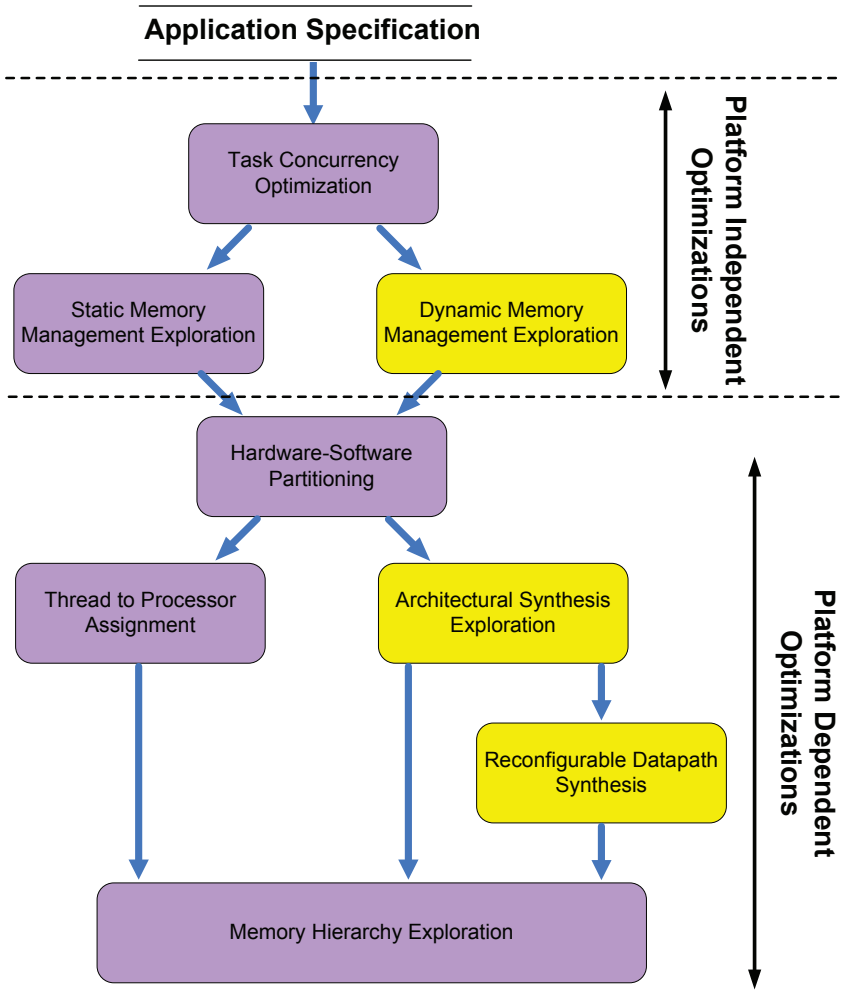


Figure 1.5.: The overall system design flow.

to an equivalent one optimized for efficient mapping onto the resource models supported by the new reconfigurable architectures. These pre-synthesis transformations enables the designer to use well known synthesis optimizations (initially developed for ASIC architectural synthesis) during the application mapping onto the introduced architectures. Regarding new synthesis techniques, an efficient resource binding algorithm was developed in order to exploit the opportunities for fine-grained vertical parallelism found into reconfigurable datapaths generated through the Flexibility Inlining.

1.4. Description of the Overall Design Flow

The complete system design flow is depicted in Fig. 1.5. The methodologies developed in this thesis address specific design problems denoted by yellow boxes. In particular, we target the design problems concerning (i) the dynamic memory management of multi-threaded applications, (ii) the exploration for Delay-Area efficient hardware coprocessor

synthesis and (iii) the architectural optimization of the hardware coprocessors mapped onto reconfigurable datapaths. The rest of the boxes, included in the illustrated design flow, forms differing research topics in the field of digital system design that have been extensively addressed by other research groups.

The complete design flow includes a various number of optimization steps. Given the application specification, platform independent optimizations are performed first, followed by the corresponding platform dependent ones. The concept behind platform independent optimizations is that their appliance does not require the complete specification of the platform. However, the design decisions that are taken during these steps has a great impact in platform specific metrics and in many cases works as guide of the platform dependent optimizations. For example, the design decisions concerning the platform independent step of dynamic memory management have a great impact on the memory access patterns which are highly correlated with the overall power consumption. In addition, the same decisions will guide the thread to processor mapping and the memory hierarchy exploration steps [61]. On the other hand, platform dependent optimizations are closely coupled with a specific platform instance. For example, the software-hardware partitioning step depends on the hardware coprocessors and the processor types included in the platform, while the thread-to-processor mapping step depends on the physical adjacency of the processor cores onto the floorplanned system, i.e. [62].

Platform independent optimizations can be viewed as high level source-to-source transformations that optimizes the initial application specification. They include the following optimization steps:

- Task Concurrency Optimization: This step performs the parallelization of the initial application specification. Source-to-source rewriting passes can be employed to parallelize the application according to task-level dependencies [63]. A parallel version of the code is generated based on first-in first-out (FIFO) communication directives and synchronization mechanisms.

- Static Memory Management Exploration: Static memory refers to the manipulation of statically allocated data, i.e. arrays. The main feature of static data is that their allocation is known during compile time. Thus, the multi-threaded version of the application is further analyzed and explored in order to incorporate scheduled data transfers of static data between the main memory and the local memories (Data Transfer and Storage Exploration (DTSE) methodology [58], [64]) .

- Dynamic Memory Management (DMM) Exploration: Dynamic memory refers to the manipulation of dynamically allocated data, i.e. data handled through malloc and free functions. The combination of malloc and free functions form the dynamic memory manager. The main objective of this step is to generate customized dynamic memory managers tailored to the applications specific needs. We propose to perform dynamic memory management customization, based on systematic exploration applied to a newly introduced parameter space (Chapter 2). The customized dynamic memory manager replaces the general purpose one.

The outcome of the platform independent optimization steps is a transformed version of the initial application specification optimized regarding the task level concurrency and memory management features. The platform dependent optimization steps further refine

the transformed application according to platform specific constraints. We identify two types of platform specific constraints, (i) resource constraints and (ii) topology constraints. Resource constraints refer to the allocated processing and storage elements, i.e. number of processors and on-chip memories, type of processors (programmable processors, DSPs, ASIC accelerators, reconfigurable accelerators). The topology constraints refer to the on-chip interconnection structure i.e. single bus, segmented bus, NoC etc. Given the aforementioned constraints, the following five steps form the platform dependent optimization phase:

- **Hardware-Software Partitioning: In this step the application specification is co-designed and partitioned in its hardware and software coefficients [65]. The software coefficients are mapped onto programmable processors, while the hardware coefficients are mapped onto hardware accelerators. In typical co-design environments, the partitioning is performed according to the computational characteristics of each thread. Thus, computationally intensive tasks and functions are usually mapped to hardware, while the control-intensive parts of the application are mapped onto software.**

- **Thread to Processor Assignment: The thread to processor assignment step maps the software coefficients to processors according to the platform specific resource and interconnection constraints. A lot of past and recent activities both from the embedded and parallel computing research areas have proposed efficient solution for the specific problem [66].**

- **Architectural Synthesis Exploration: Architectural synthesis exploration step refers to the generation of optimized hardware coprocessors. It has a two-fold objective. The first objective is the automatic generation of the circuit-level implementation of the hardware coefficients described in the algorithmic level. High level synthesis frameworks can be used for this procedure [57], [67]. The second objective is to deliver Pareto-optimal datapath solutions regarding designer specific constraints. For this second objective, automatic exploration of differing architectural synthesis parameters has to be performed in order to find the Pareto front. We propose a new exploration framework which moves the explored Pareto front towards higher quality datapath solutions, taking into consideration the combined interaction of compiler-level and architectural parameters (Chapter 3).**

- **Reconfigurable Datapath Synthesis: In this step the Pareto-optimal configurations of the hardware coefficients that are mapped onto reconfigurable coprocessor architectures is performed. We focus on coarse-grained reconfigurable architectures targeting DSP application acceleration. Towards this direction, we propose two new coarse-grained reconfigurable architectural templates complement with the corresponding architectural synthesis methodologies. The proposed methodologies exploit functional-unit and bit-level sharing together with arithmetic level optimizations for delay-area optimized reconfigurable solutions (Chapters 4-5).**

- **Memory Hierarchy Exploration: In this step the application's data structures are mapped to specific memory modules forming the on-chip memory hierarchy. After the architecture specification of the processing elements, the memory hierarchy exploration generates the configuration of the on-chip memory system taking into consideration the access patterns and the memory footprint of each processing element and the physical locations of the memory modules onto the multi-core platform [68].**

1.5. Thesis Contributions

Sections 1.1-1.4 discussed our basic research field of interest. In this section, we present the specific problems addressed in this PhD thesis.

Problem 1. ***There is lack of efficient approaches regarding application-specific dynamic memory management for multi-threaded applications. The existing approaches consider a limited parameter space concerning each thread as a single application, failing to capture the impact that the multi-threaded environments have onto the dynamic memory management. This results to a large unexplored fraction of the parameter space which further leads to loss of optimal solutions.***

Proposed Solution. We introduce a new design space for Multi-Threaded Dynamic Memory Management (MTh-DMM) in Multi-Processor SoCs (MPSoCs). Based on the introduced design space, we propose a systematic exploration methodology to efficiently traverse through the decisions of the parameter space. Software tools have been developed to automate the exploration procedure and the code generation of application-specific MTh-DMM configurations under various optimization objectives, i.e. memory footprint, memory accesses, fragmentation, execution time etc. The overall exploration procedure is automated and transparent to the designer leading to high productivity gains (Chapter 2).

Problem 2. ***Existing architectural synthesis approaches do not take into account trade-offs produced by the combined impact of architecture level optimizations together with behavioral level optimizations. This results to a large unexplored fraction of the design space and to the loss of optimal solutions.***

Proposed Solution. A new augmented design space has been modeled considering both architecture and behavioral level optimizations. In order to efficiently traverse the augmented design space, efficient exploration techniques have been developed. In each case, the proposed methodology delivers more optimized design solutions than the existing exploration approaches and converge to the global optima in a quick manner. Appropriate tools have been developed to automate the proposed design space exploration methodology. The overall exploration procedure is transparent to designer and thus the discovery and convergence towards higher quality solutions comes along with high productivity gains (Chapter 3).

Problem 3. ***Existing coarse-grained reconfigurable architectures take into account only sharing at the Functional Unit (FU) level between identical types of FUs, thus delivering high area overheads.***

Proposed Solution. A novel circuit level design technique, called Flexibility Inlining, has been developed which generates area efficient coarse-grained reconfigurable architectural

templates taking into consideration bit-level hardware sharing. Flexibility Inlining enables both (i) the conventional sharing between identical type FUs together with (ii) the hardware sharing between of FUs of different type and functionality (i.e. adders and multipliers), based on the efficient datapath merging of single or chained Carry-Save (CS) additions/subtractions and CS array-multiplication. Common interconnection schemes among the merged arithmetic operators are exposed through a series of architectural- and bit-level transformations, which exploits the structural symmetries. Thus, operation sharing is enabled with zero overhead on the interconnection scheme of the original arithmetic datapaths. The two-level hardware sharing enabled by Flexibility Inlining results in area efficient reconfigurable datapaths without compromising execution delay of the reconfigurable datapath (Chapter 4).

Problem 4. ***Existing coarse-grained reconfigurable architectures do not take into account arithmetic level optimizations during architecture specification leading to major delay overheads in comparison with ASICs.***

Proposed Solution. The CS arithmetic optimization technique has been considered during architectural design of coarse-grained architectures. CS arithmetic optimization has been enabled at the bit-level sharing during the appliance of Flexibility Inlining technique (Chapter 4). In addition, existing design methodologies for reconfigurable architectures, which are based on single FU sharing, have been extended in order to take into account CS arithmetic optimization (Chapter 5).

The techniques developed for Problem 3 and Problem 4 give solution to a fundamental problem in circuit design: How to design area efficient circuits without compromising operating frequency and vice versa.

Problem 5. ***Productivity and strict time-to-market requirements impose severe constraints in the adoption of new architectural templates based on complex resource models. Thus, optimized solutions (i.e. the proposed coarse-grained reconfigurable architectures) are often excluded due to their programming complexity imposed by the underlying hardware resource model.***

Proposed Solution. New high-level synthesis algorithms and tools have been developed which automatically map behavioral descriptions (i.e. C descriptions) onto the proposed reconfigurable architectural templates. The synthesis algorithms take into consideration the inherent features of the architectural template delivering optimized mappings for each application kernel. For the first time, the combined exploitation of the architectural optimizations: (i) horizontal parallelism, (ii) vertical parallelism and (iii) operation chaining have been considered (Chapter 4). ***Consequently, we filled an existing gap found in current datapath synthesis literature, since the aforementioned architectural optimizations did***

not treated in a holistic way. Also a new source-source transformation has been developed to efficiently utilized CS arithmetic optimizations during datapath synthesis (Chapter 5). In both cases, the complexity of the underlying hardware resource model is becoming transparent to the designer. By this way, large applications can be coded in standard programming languages and then mapped onto optimized flexible datapath solutions, no matter how complex is their resource model. Thus, higher design productivity is achieved for reconfigurable datapaths.

Furthermore, the proposed methodologies delivers high quality coprocessor solutions, *achieving significant gains in both execution time and area complexity compared to existing state-of-the-art reconfigurable architectures i.e. [15], [16].*

The above discussion is summarized in Fig. 1.6 showing the research tree of this PhD thesis.

1.6. Thesis Organization

The dissertation is organized in two parts. We adopted a top-down approach to present the technical contributions, following the design flow discussed in Section 1.4. Each part consists of two chapters. Part I is focused on the problem of design space exploration, while Part II concentrates on the problem of architectural synthesis for optimized coarse-grained reconfigurable coprocessor architectures. Specifically, the remainder consists of the following chapters:

In Chapter 2, we address the problem of custom DMM in MPSoC architectures. We introduce a extended design space that captures in a global, modular and parameterized manner the primitive building blocks of multi-threaded DMM. A systematic exploration methodology working at the parameter space is proposed to efficiently traverse the extended design space. The software tools supporting the proposed methodology along with the heuristics used are also presented. Experimental results evaluating the proposed methodology are presented using a real-life multi-threaded dynamic network application as a case-study.

Chapter 3 targets the problem of design space exploration during high level synthesis for discovering delay-area optimal trade-off points. While previous research attempts silently neglects the trade-offs produced from the combined impact of behavioral-level together with architectural-level parameters, we propose a novel design space exploration methodology that studies an extended instance of the solution space considering the effects of combining compiler- and architectural-level transformations. We show that exploring the design space in a global manner reveals new Pareto trade-off points, thus shifting towards higher quality design solutions. We present a new exploration algorithm based on a combination of upper-bounding conditions together with gradient-based heuristic pruning to efficiently traverse the extended search space. Experimental evaluation, based on an automated exploration framework implementing the proposed algorithm, shows significant quality improvements without compromising the optimality (Pareto accuracy) of the discovered solutions. In addition, significant runtime reductions are reported compared to exhaustive exploration of the solution space.

In Chapter 4, we introduce the circuit-level Flexibility Inlining technique along with architectural synthesis techniques for high performance and area efficient coarse-grained reconfigurable datapaths, targeting computationally intensive DSP kernels. The circuit-level design methodology is analyzed in detail and an area-time efficient reconfigurable architecture is presented. Based on the introduced coarse-grained reconfigurable/flexible architectural template, we propose a methodology which enables the combined exploitation of the horizontal and vertical parallelism along with the operation chaining opportunities found in the application's behavioral description. Efficient synthesis techniques exploiting these architectural optimization concepts from a higher level of abstraction are presented. Experimental results including quantitative and qualitative comparisons with existing reconfigurable arithmetic cores and exploration results of the proposed reconfigurable architecture are provided.

Chapter 5 presents an alternative synthesis methodology targeting to template-based flexible datapaths with tree-based multipliers. We introduce the consideration of arithmetic level optimizations for template based datapath synthesis. A high performance architecture for the implementation of DSP kernels is proposed, based on flexible and arithmetically optimized components able to perform a large set of operation templates. A synthesis methodology for optimized mapping of DSP kernels onto the proposed architecture is also presented. Experimental results showing significant gains in execution time, active chip area and power dissipation in comparison to state-of-the-art template-based flexible datapaths incorporating tree multipliers are also included.

Finally, Chapter 6 concludes the dissertation and highlights possible future extensions of this work.

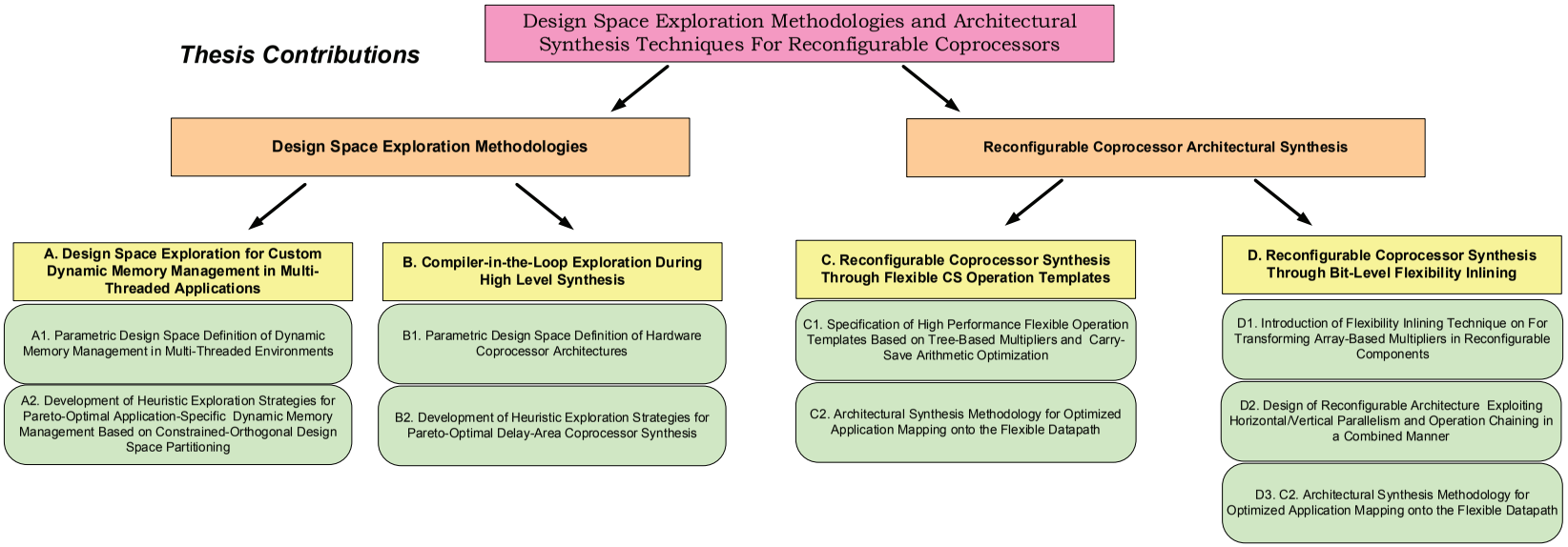


Figure 1.6.: Research tree of the dissertation.

Part I.

Multi-Level Design Space Exploration for System and Datapath Optimization

Truth is much too complicated to allow anything but approximations.

John von Neumann

The first part of this thesis is focused on the problem of design space exploration. Design space exploration is the procedure that evaluates the solution space in order to return a set of design points that form the optimal trade-offs according to some design criterions (execution delay, circuit area, dissipated power etc.). Pareto optimality [28] specifies that one design solution dominates the other when it is at least as good in all of the examined criteria and also strictly better in at least one of them. Thus, designers are interested mostly on Pareto-optimal configurations. The enumeration of all possible solutions in order to extract the Pareto frontier is an extremely time-consuming task, since the size of the design space is usually huge. Thus, DSE methodologies are focused on the development of strategies for efficient traversal of the available design space.

Exploration strategies are highly dependent on the domain and the context of each targeted design space i.e. architectural synthesis domain for coprocessors [44], platform based design domain [69], processor micro-architecture domain [70], interconnection architecture domain [71], memory management domain [72] etc. In order to compensate this large diversity of domains and contexts in DSE, designers are typically using meta-heuristic approaches [73] based on the conviction that the algorithms are going to converge to the exact Pareto solutions. However, the usage of meta-heuristics as "black boxes" leaves limited opportunities to the expert designer to incorporate knowledge of the specific structure of the design space. In this thesis, we adopt a different approach developing efficient exploration strategies exploiting the knowledge on the context of the targeted domain.

In order to manage the diversity of differing domains and contexts of each design space, we establish a top-down conceptual analysis framework that enable us to develop efficient exploration strategies. This conceptual analysis framework is applied prior to the development of the actual exploration methodology, being actually its enabler. Fig. 1.7 depicts an abstract view of this approach. Given the design space (it can be provided by an expert or through extensive search into the literature), the conceptual framework performs three hierarchical orthogonalizations (subspace-level, category-level and decision-level). The result of each orthogonalization guides the one in the next level. Through this hierarchical orthogonalization, the initial design space is decomposed in its orthogonal coefficients (orthogonal decisions). Decision orthogonalization limits redundancy among the examined configurations (each examined configuration can be viewed as a vector of orthogonal decisions). For each orthogonalization step, a conceptual analysis of the dependencies existing between the each element is performed, whenever the structure of the examined design space permits it. Through dependency analysis, we extract the potential influence of each parent element to each of its children (dependent) elements. Using this knowledge the designer of the exploration strategy identifies semantically non-valid configurations (if any) that are excluded from further exploration (pruned) without worrying on excluding potential optimal solutions. In addition, the dependency analysis at the decision level enables the designer to better visualize the design space and extract guidelines concerning potential decision orderings. Decision ordering refers to the actual traversal of the design space during exploration, forming the first step for developing the exploration strategy. Having as inputs the results of this conceptual analysis, the designer is equipped with an insight on the internal structure of the design space in order to develop the domain specific exploration strategy.

We are interest in the case that the goal is to develop an exploration framework like the one shown at the bottom of Fig. 1.7. Without loss of generality, we assume that the exploration framework is composed by (i) an exploration heuristic (structured according to the aforementioned conceptual analysis), (ii) the evaluation framework (the actual function is explored) and (iii) a Pareto analyzer module that extracts the Pareto front of the explored configurations. In this thesis, we applied the aforementioned conceptual framework and develop efficient exploration methodologies for two distinct DSE problems from different domains:

1. one occurred during system level design regarding customized dynamic memory management for multi-threaded applications and

2. another one occurred during high level synthesis of coprocessor architectures.

At the first studied DSE instance, we address the problem of customizing software implementations of dynamic memory managers in the context of multi-threaded applications on MPSoC platforms. Multiple use-case scenarios and increased interaction with the environment expose the high dynamic behavior of these systems. The increased dynamism leads to unexpected memory footprint variations unknown at design time. Developing dynamic multi-threaded applications, using worst-case estimates for managing memory in a static manner, would impose severe overheads in memory footprint and power consumption. Thus, developers are motivated to efficiently utilize dynamic memory in order to avoid such type of costly over-estimations. However, the inherent generality of existing dynamic memory management imposes sub-optimal solutions to be delivered in respect to the target application. Application specific dynamic memory management is an effective way to over pass these inefficiencies improving performance and/or reducing memory footprint. We define the design space for MTh-DMM and propose an automated software-supported systematic methodology to efficiently explore and traverse through the decisions of the MTh-DMM design space taking into account various optimization criteria (Chapter 2).

Concerning the DSE for coprocessor synthesis, we target to the fundamental problem of exploring the Delay-Area trade-offs during high level synthesis. Existing approaches in this area are far from delivering the most efficient design solutions. That is because neither the impact of code transformations on the behavioral description nor the full set of architectural optimizations during scheduling, are considered as effective design parameters. Usually, code-level transformations (i.e. loop unrolling) and architectural level optimizations (i.e. operation chaining) are treated as user-guided pre-defined parameters during exploration. In a common case, such exploration strategies deliver suboptimal design points since the interaction between the structure of behavioral code and the architectural optimizations are silently assumed to be independent. In order to address this inefficiency, we have formulate an exploration methodology which accounts for both the loop unrolling code transformation and operation chaining architectural optimization as parameters of the design space (Chapter 3). The proposed methodology explores the extended design space in a quick and efficient manner and delivers exploration curves which dominate the curves generated by the existing exploration strategies.

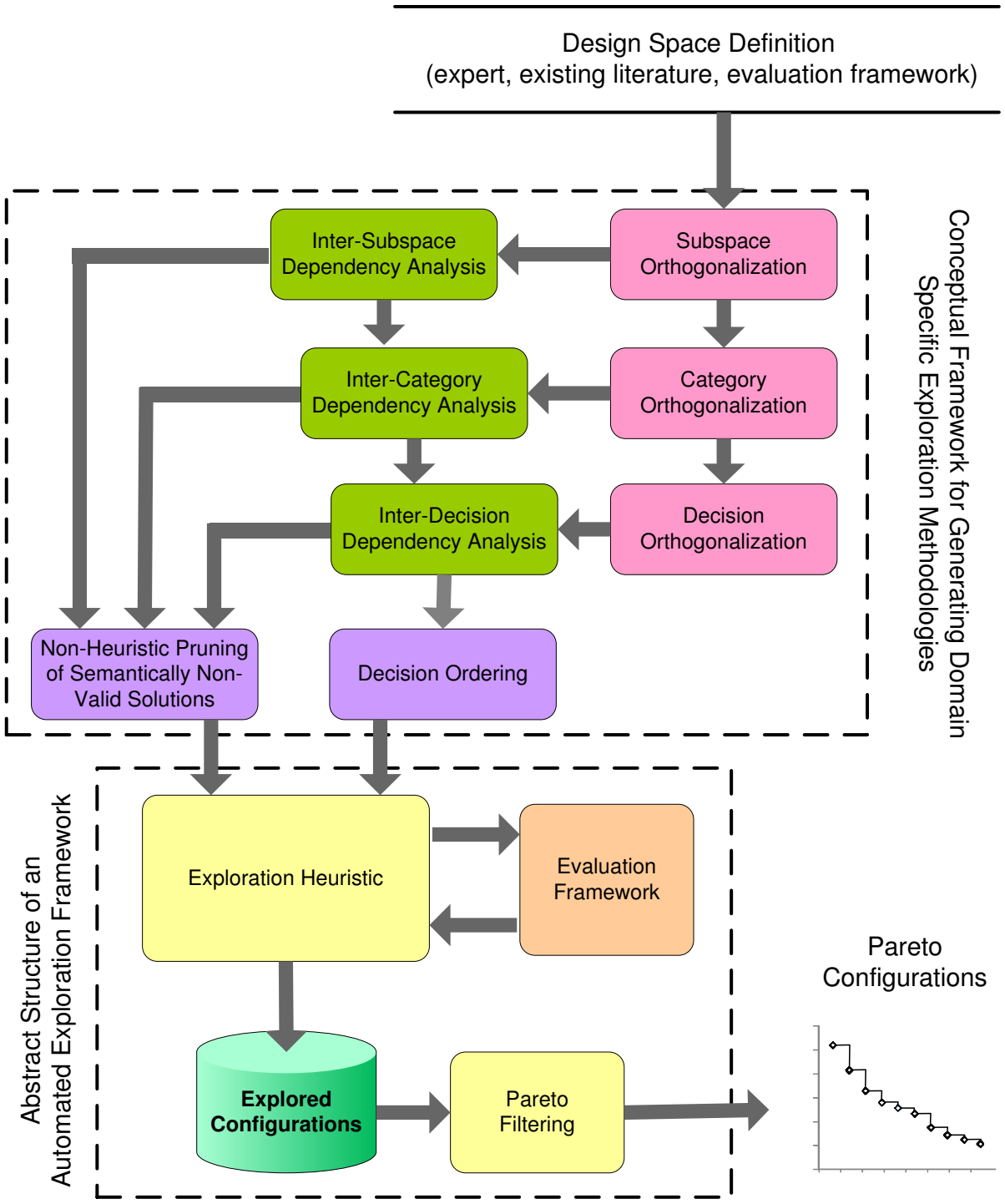


Figure 1.7.: The conceptual framework for domain specific DSE.

2. A Systematic Exploration Methodology for Application-Specific Multi-Threaded Dynamic Memory Management

In this chapter we address the problem of custom dynamic memory management in MPSoC architectures. Customization is enabled through the definition of a design space that captures in a global, modular and parameterized manner the primitive building blocks of multi-threaded DMM. A systematic exploration methodology is proposed to efficiently traverse the design space. Customized Pareto DMM configurations are automatically generated through the development of software tools implementing the proposed methodology. Experimental evaluation based on a real-life multi-threaded dynamic network application show that the proposed methodology delivers higher quality (application-specific) solutions in comparison with state-of-the-art dynamic memory managers together with 62% exploration runtime reductions.

2.1. Introduction

Recent advances in VLSI process technology enabled the shifting from single-processor SoC to multi-processor SoC architectures. The emerging market of embedded devices highly adopts the MPSoC architectural template [74], [75]. In order to support multiple services and heterogeneous applications i.e. multimedia, wireless communications etc. MPSoC platforms exploit parallelism in coarser levels (thread-level) than the instruction-level (ILP) found in common SoCs. Thus, multi-threaded applications are becoming increasingly prevalent for next generation embedded systems. Furthermore, multiple use-case scenarios and increased interaction with the environment expose the high dynamic behavior of these systems. The increased dynamism leads to unexpected memory footprint variations unknown at design time. Developing dynamic multi-threaded applications, using worst-case estimates for managing memory in a static manner, would impose severe overheads in memory footprint and power consumption. To avoid such type of costly over-estimations, developers are motivated to efficiently utilize dynamic memory [34].

However, dynamic memory management often forms the main performance and scalability bottleneck of multi-threaded applications [20]. In addition, it greatly affects the energy and memory consumption of the overall system [35]. Dynamic Memory (DM) managers are responsible for organizing the dynamically allocated data in memory and servicing the application memory requests (allocation/deallocation) at run-time. In case of a memory request for allocation of a new object (using the `malloc/new` library functions in C/C++), the DM manager returns to the application the pointer, which refers to the memory position of the allocated object. In case of a request for memory deallocation (using the

`free/delete`), the DM manager returns either a true or a false value in respect to success of the deallocation process.

Extensive research has been conducted for general-purpose dynamic memory management, which target either the single processor [34], [36] or the multi-processor domain [37], [38], [39], [21]. The inherent generality of existing DMM imposes sub-optimal solutions to be delivered in respect to the target application. Recently, customized (application-specific) DMM has been proposed as an effective way either to improve performance [40] or to reduce memory footprint [30]. *However, DMM customization has been considered only for single-threaded applications running onto single-processor platforms.* Specifically, a semantic and bibliographic gap is reported considering the existence of methodologies for designing efficient custom DM managers for multi-threaded applications.

We address the problem of customizing DM managers in the context of multi-threaded applications on MPSoC platforms. To the best of our knowledge, this is the first work targeting the aforementioned problem. *Our main contributions are: (i) Define the design space for multi-threaded dynamic memory management in MPSoCs. (ii) Propose a systematic methodology to efficiently explore and traverse through the decisions of the design space. (iii) Develop software tools to automate the exploration procedure and the code generation of application-specific MTh-DMM configurations under various optimization criteria.*

The design space is defined through platform-independent Decision Trees (DTs) that capture both the inter- and intra-heap level decisions for composing software customized MTh-DMM configurations. A componentwise C++ library, based on Mixin layers [42], has been developed to implement the design decisions found into the extended MTh-DMM search space. By this way, the various MTh-DMM solutions are composed in layered manner. An exploration methodology is proposed to efficiently traverse the available design space returning a Pareto set [28] of customized MTh-DMM configurations. The methodology is supported through proper software tools, which perform the exploration and the source code generation of the Pareto MTh-DMM configurations in a fully automated manner. The efficiency of the exploration methodology is evaluated through extensive experimentation on a real dynamic multi-threaded network application. Comparative results show that the proposed methodology delivers customized MTh-DMMs with significant gains on critical performance metrics i.e., memory footprint, number of memory accesses, etc. Also, design-time reductions of 62% (up to 22 days) are achieved comparing with hard-coded customization from an expert MTh-DMM designer.

The rest of the chapter is organized as follows. A short overview of the related work is presented in Sections 2.2-2.3. The design space is introduced in Section 2.4, whereas the exploration methodology is described in Section 2.5. The evaluation of the proposed approach is performed in Section 2.6 and, finally conclusions are drawn in Section 2.7.

2.2. Preliminary Discussion

Since, we target to embedded systems, both general purpose (i.e. performance, scalability, false sharing, memory fragmentation) and application specific (i.e. energy consumption, memory footprint, code size) metrics have to be considered.

2.2.1. Heap-Based Dynamic Memory Management

The memory pool responsible for allocation or deallocation of arbitrarily-sized blocks in arbitrary order that will live an arbitrary amount of time is called heap. In order to conceptually position the heap region, Fig. 2.1 depicts a typical memory layout of a running process, assuming a C application running onto a Linux-OS (the organization of process address space is determined by the Operating System (OS) and the programming language). In brief, the ".text" segment stores the compiled code of the program that forms the running process, the ".data" segment contains global and static variables used by the program that are initialized. while the ".bss" segment contains all the uninitialized global variables and static variables that are initialized to zero by default. The stack is the section of memory allocated for variables within functions or temporary storage of information, using a Last In First Out (LIFO) scheme.

Between the stack and the ".bss" segment, it lays the heap memory pool. The heap is the memory region that the dynamically allocated data objects are stored. Since the size of the allocated data is unknown until the actual runtime, the heap is managed by dividing it into blocks able to service the runtime memory request. The unallocated (free) blocks are organized based on dynamic data structures (i.e. single linked lists, double link lists, trees, etc.), usually called free-lists. In many cases several free-lists exist inside the heap address space. During an allocation request the dynamic memory manager searches the free-lists in order to find an available free block to return. In order to reduce the searching overhead, a common practice is to manage free-lists that handle memory blocks of a specific size, called fix-sized free-lists or fixed-lists. By this way the searching is reduced since the manager knows where to search first. Fixed-lists can be viewed as a caching mechanism of the dynamic memory manager. The size of the heap during the execution is defined by the brk pointer. Thus, each time the application requests for memory allocation, the dynamic memory manager either returns a pointer to an unused block found into the heap or requests for additional heap memory from the operating system using the sbrk function (actually requests for moving the brk pointer).

Dynamic memory managers are operate on the heap memory space. They are responsible for organizing the dynamically allocated data into the heap and servicing the application's memory requests (allocation/deallocation) at run-time. In case of a memory request for allocation of a new object, the dynamic memory manager returns to the application the pointer, which refers to the memory position of the allocated object. In case of a memory request for deallocation of an already dynamically allocated object, the dynamic memory manager returns to the application either a true or a false value in respect to success of the deallocation process. In C/C++ programming language dynamic memory management is performed through the malloc/new functions for allocation and free/delete functions for deallocation, respectively.

2.2.2. MTh-DMM Performance Metrics

Historically, the efficiency of single-threaded dynamic memory managers was evaluated according to conventional metrics such as performance and memory fragmentation (internal and external). However, in the field of multi-threaded dynamic memory managers both conventional and new defined metrics are required in order to perform accurate evaluation of their efficiency [21]. That is because of new design constraints are taken into account

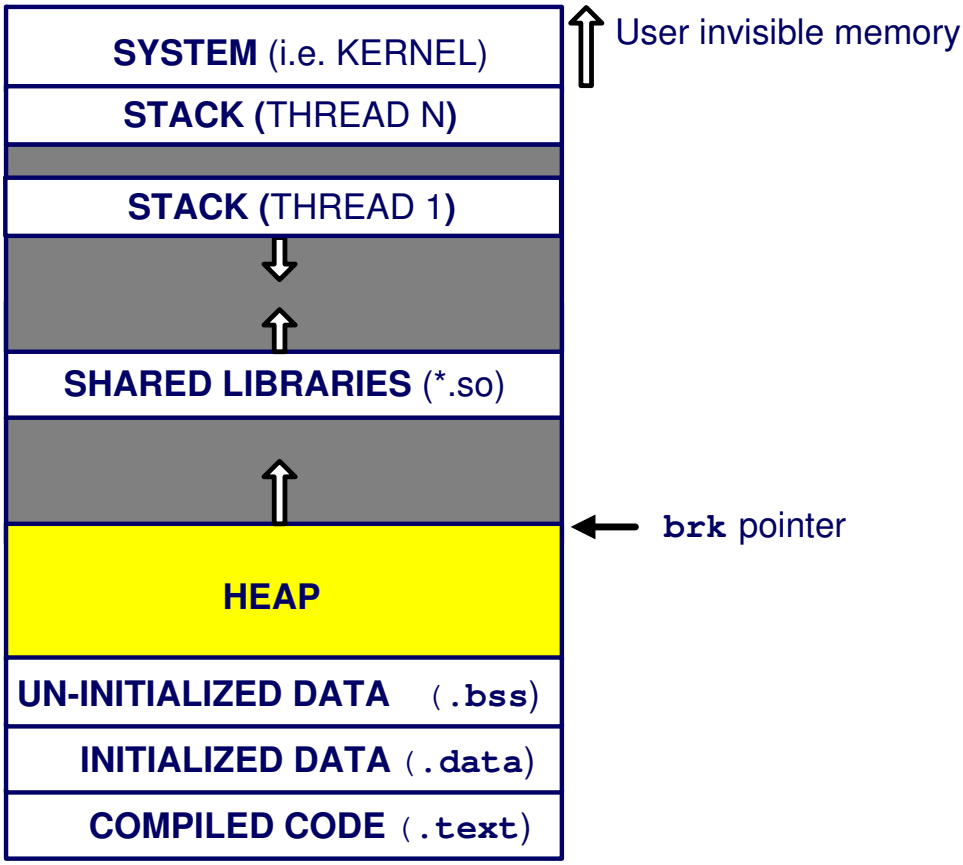


Figure 2.1.: Memory layout of process address space.

for the case of multi-threaded applications i.e. scalability of the solution and avoidance of false sharing. Furthermore, design constraints that come from the embedded computing community (i.e. energy consumption) have to also be taken into account. Thus, in order to design efficient multi-threaded dynamic memory allocators, the following metrics should have to be considered:

1. Performance: A multi-threaded dynamic memory allocator should perform memory operations (i.e., malloc and free) about as fast as a state-of-the-art serial memory allocator. By this way, performance is guaranteed even when a multi-threaded program executes on a single-processor. While in single-threaded applications performance is affected mainly by the fit and search policies of the allocator, in multi-threaded applications also the synchronization mechanisms and strategies have a great impact in the allocator's performance.

2. Scalability: A scalable multi-threaded dynamic memory allocator should guarantee that as the number of processors in the system grows, the performance of the allocator also grows/scale linearly with the number of processors to ensure scalable application performance. The un-scalable behavior of the memory allocator makes it the bottleneck of the overall application.

3. **Heap False Sharing: Heap false sharing refers to the situation in which threads with distinct heaps inadvertently share data. For example, data allocated from thread 1 into heap 1 are freed from of thread 2 into heap 2. In many access patterns, such a situation may impose extensive memory consumption. Considering a platform dependent view, heap corruption resembles the false sharing of cache lines.**

4. **Memory Fragmentation: In general fragmentation is defined as the maximum amount of memory allocated from the operating system divided by the maximum amount of memory required by the application. In multi-threaded memory allocators there are three types of fragmentation:**

   a) **Internal Fragmentation: It happens when the allocator returns a memory block that is larger than the initial size request.**

   b) **External Fragmentation: It happens when a memory request cannot be served even if there is available memory space that can serve the request (i.e. if free blocks were merged).**

   c) **Memory Blowup: This is a special kind of fragmentation found in multi-threaded memory allocators. Specifically, it refers to the situation in which the increase in memory consumption caused when a concurrent allocator reclaims memory freed by the program but fails to use it to satisfy future memory requests. It is defined as the maximum amount of memory allocated by a given allocator divided by the maximum amount of memory allocated by an ideal uni-processor allocator [21].**

5. **Energy Consumption: This design metric comes from the field of embedded computing in which battery lifetime forms a critical resource. Energy consumption for DMM has a direct relation with the number of memory accesses that the allocator performs to service a memory request.**

2.3. Related Work: Analysis of Heap Architectures

General-purpose MTh-DMM can be classified according to the organization of the heap memory pool [21]. A similar taxonomy for the case of single-threaded DMM can be found in the Wilson's extensive report [34]. Each multi-threaded allocator [2] presented in the existing literature can be assigned to one of these classes. Regarding the organization of the heap memory space, two major classes of heap architectures can be identified, namely the single-heap and the multiple-heap architecture.

2.3.1. Class A. Single Heap MTh-DMM

A.1 Serial Single Heap MTh-DMM **assumes a global shared heap, protected through locking before each (de)allocation request. Serialization of memory operations and heavy lock**

[2] In the reminder, we are going to use interchangeably the terms allocator and dynamic memory manager.

contention are introduced, forming a serious bottleneck in the case of multi-threaded applications. Since all the threads operate on the same heap, false sharing is heavily induced. Many operating systems provide such type of memory allocators in their default library [20]. Existing customization methodologies [40], [30] can be applied in straightforward manner in this class of allocators.

A.2 Concurrent Single Heap Allocators implement the heap as a concurrent data structure, such as a concurrent B-tree [39], [76] or a freelist with locks either on each free block or on the entire freelist [77], [78]. In the general case of random access patterns, they present better scalability and performance characteristics than the serial single heap allocators. However, in case that most allocations request a limited range of sizes [79], concurrent single heap allocation reduces to the case of serial single heap. The extensive use of locks makes them quite expensive in terms of energy and performance. Each synchronization operation requires one or more memory access, so their extensive use contributes in a negative way to the overall energy consumption. Customization is expected to have a positive impact in both performance and energy metrics of such allocators.

2.3.2. Class B. Multiple Heap MTh-DMM

B.1 Pure Private Heaps Allocators assume that each thread has its own private heap used for every DM operation and never accesses any other [80], [81]. Specifically, the allocator allocates/frees memory from/to its private heap. Both commercial i.e. STL's (Standard Template Library) [82], pthread alloc [80] and academic i.e. Cilk 4.1 [81] adopt the pure private heaps scheme. Each heap is *purely private* and each thread never accesses any other heap for any DM operation. In terms of performance and scalability, pure private heaps form a very efficient solution since in an ideal situation minimum lock contention is induced. However, if producer-consumer allocation patterns exists between different threads, memory consumption may become unbounded [21]. Thus, exploration and customization is required to evaluate the efficacy of pure private heaps DMM solutions in an application-specific basis.

B.2 Private Heaps Allocators with Ownership extend the pure private heaps class with heap ownership mechanisms [38], [83], [84]. Many high performance MTh-DM managers i.e. Ptmalloc [84] and LKmalloc [38] belongs to this class. Ownership mechanisms eliminate false heap sharing, by forcing the DM manager to deallocate memory blocks to the thread's heap that they originally had been allocated, no matter which thread requested the deallocation. In terms of performance and scalability, private heaps with ownership DMM is more efficient than single serial and concurrent heaps, since each thread or group of threads operates on its own heap and the lock contention is significantly smaller. Each thread or group of threads operates on its own heap and the lock contention is small delivering efficient performance and scalability. In an application-specific context, the ownership mechanisms and the various thread groupings (thread-to-heap mapping mechanisms) introduce new trade-off parameters that have to be explored for optimized custom MTh-DMM.

B.3 Private Heaps Allocators with Threshold move blocks of memory between a hierarchy of heaps shared by multiple threads [37], [21], [85]. When a private heap has more than a certain amount of free memory (crossing a threshold value), some portion of the free memory is moved to a shared heap. This strategy bounds memory blowup to a constant

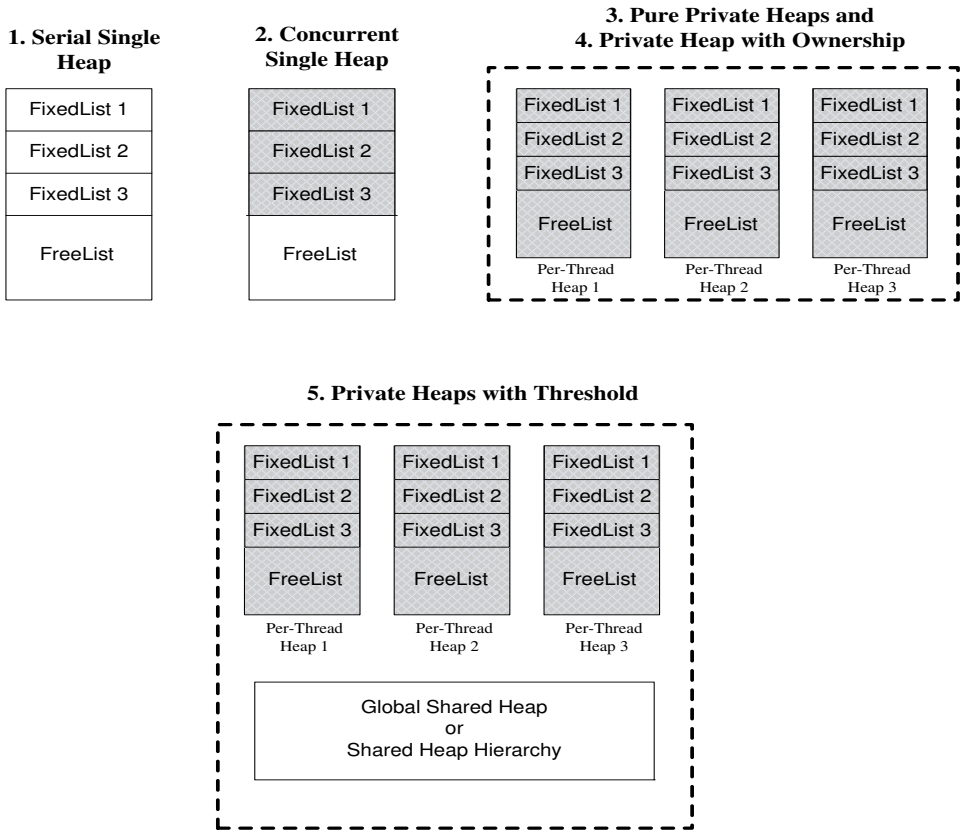


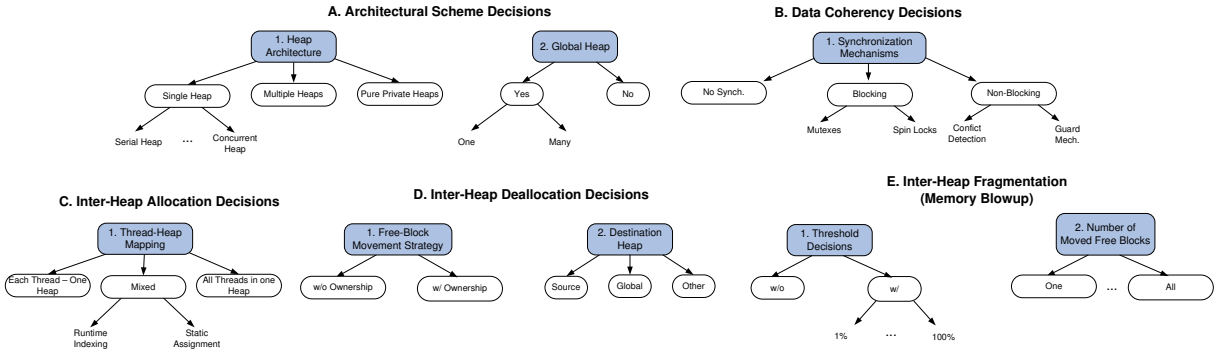
Figure 2.2.: Heap organization schemes for MTh-DMM.

factor, since no heap may hold more than some fixed amount of free memory [21]. However, the threshold value need to be carefully determined since a rather small value will initiate a lot of memory movement degrading performance, while a rather large threshold may induce large memory waste. Such trade-offs can be efficiently utilized through customization of the allocator according to the application specific needs. Characteristic DM managers of this class are the DYNIX kernel memory allocator [85], the allocator proposed by Vee and Hsu [37], the Hoard allocator by Berger [21] and its lock-free variants [86], [87].

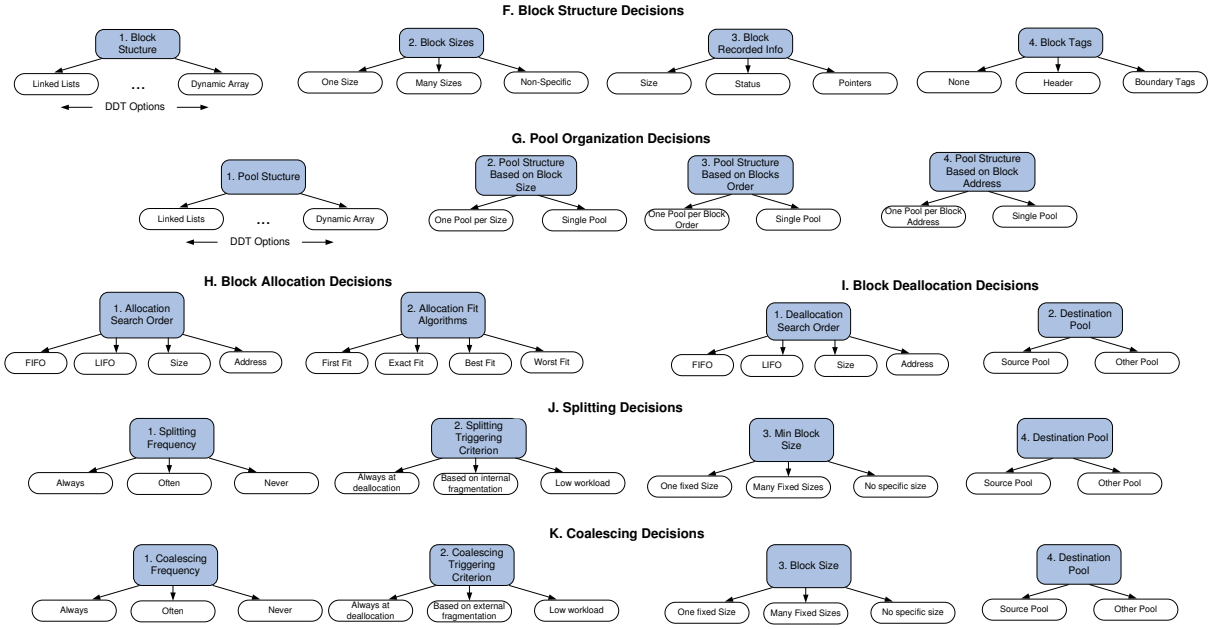
Fig. 2.2 illustrates in an abstract manner the various heap organizations that each of the dynamic memory management classes imposes. Grey areas depict the heap regions that can be accessed in parallel from different threads. The following subsections further elaborated on each of the aforementioned multi-threaded allocator classes and reference relative work and research activities.

2.4. Definition of MTh-DMM Design Space

The proposed design space is shown in Fig. 2.3. It is exhaustive in the sense that it models and instantiates every known MTh-DMM scheme. Based on a similar analysis, Atienza



(a) Inter-heap level design space.



(b) Intra-heap level design space.

Figure 2.3.: Complete MTh-DMM design space using decision trees.

et al. [30] have proposed a taxonomy of design decisions covering the design space for single-threaded DM management for applications running on single-processor platforms. *We extend the single-threaded DMM design space in order to efficiently model and capture decisions found in the field of multiprocessor and multi-threaded application domain. Any known dynamic memory allocator (multi-threaded or not) found in the literature can be modeled through the proposed design space.* The extended design space remains platform-independent and applicable to any shared memory MPSoC platform after a platform-dependent refinement. Decisions concerning both the inter- and intra-heap levels are identified. The union of these levels returns the overall MTh-DMM design space. The inter-heap level (Fig. 2.3a) captures decisions that are shared between the threads such as heap organization, thread to heap mapping policies etc. The intra-heap level (Fig. 2.3b) manages decisions which enable the customization of each instantiated heap's internal structure i.e. number of freelists, allocation mechanisms etc. These decisions can be seen as primitive building blocks offering modular composition of MTh-DMMs.

2.4.1. Inter-Heap Level Design Decisions

Fig. 2.3a depicts the inter-heap level decisions for MTh-DMM. The decisions have been composed into DTs and classified into five categories:

Category A. Architectural Scheme Decisions consist of two DTs concerning the heap organization of the MTh-DMM to exploit the available thread-level parallelism. The *Heap Architecture* DT defines the top-level architecture of the DM manager. The *Single Heap* decision implies a global shared heap. *Pure Private Heaps* organization implies that the number of available heaps equals the number of running threads. Thus, memory requests can be served with maximum parallelization. *Multiple Heaps* DM architectures form a intermediate solution which enables the sharing of heaps between different threads. By this way, dynamic memory requests are served in a parallel manner but perhaps not the maximum one. The *Global Heap* DT handles the decision of the existence or not of a global heap. The existence of a global heap affects the overall heap architecture of the MTh-DMM since it enables the sharing and the movement of the deallocated blocks. In combination with the decision of the *Heap Architecture* DT, it determines the overall architectural scheme of the allocator.

Category B. Data Coherency Decisions are modeled through a single DT. Since multiple-threads are running in parallel, multiple memory requests rise concurrently from different sources, corrupting data and functionality if coherency mechanisms are ignored. The *Synchronization Mechanisms* DT handles the decisions considering data coherency of the available heaps. In case that each thread operates on each own separate space without interacting with the heap space of other threads, i.e. Pure Private Heaps architecture, no synchronization mechanisms are required. In all other cases the synchronization can be ensured either using blocking primitives or nonblocking mechanisms. *Blocking* primitives are based on the mutual exclusion of the threads and they can be either mutexes or spin locks [88]. Locks are classified by what happens when the locking prevents the progress of the thread. *Mutexes* block the execution of the thread requesting the lock until it is allowed to access the locked resource. On the other hand, *Spin Locks* are locks where the thread spins until the lock becomes available, thus being very efficient if threads are to be blocked for a short period of time. Spin locks are very efficient if threads are only likely to be blocked for a short period of time, as it avoids the overhead of thread re-scheduling, but they are wasteful if the lock is held for a long period of time. *Non Blocking* synchronization avoids thread blocking. After each operation on a shared resource of the system, specialized mechanisms are triggered to validate the legality of the operation i.e. conflict detection mechanisms [89], guarding mechanisms [90] and atomic transactions [86]. Here, we consider MTh-DMM implementations based on blocking synchronization primitives, since non-blocking synchronization requires complex hardware and OS support [91] usually inappropriate for embedded solutions.

Category C. Inter-Heap Allocation Decisions manage the way threads allocate memory in the inter-heap level. Allocation in this level is strongly connected with decisions considering heap sharing and thread to heap mapping. Allocation decisions of finer granularity i.e. fit policies etc. are included into the intra-heap design space. At the inter-heap level, allocation decisions reduce to the mapping of threads to heaps. *Thread to Heap Mapping* DT determines which of the available heaps will serve the DM request in respect to the thread's ID. The "*Each Thread-One Heap*" and "*All Threads-One Heap*" decision leaves form the two boundary cases. In the former case each thread occupies its own heap and all the allocation requests are served according to one-to-one mapping [80], [81]. The latter case

imposes the existence of only one heap through which all the allocation requests are served based on a many-to-one mapping function [34]. The "*Mixed*" decision defines that the heaps are shared between threads and that a many-to-many mapping function is required in order to determine the assignment of threads to available heaps. Runtime indexing [21] or static thread-to-heap assignment (if #threads is known a-priori) can be incorporated. A hash mapping function can be used in order to enable assignment at run-time according to thread's ID [21]. In case that the number of threads is known a-priori, static thread-to-heap assignment can be also incorporated.

Category D. Inter-Heap Deallocation Decisions deal with the actions required in any MTh-DM manager to satisfy the deallocation memory requests. The *Free Block Movement Strategy* DT manages the decision of including heap-ownership mechanisms into the DM allocator. Heap ownership bounds memory blowup and eliminates false sharing [38] at the cost of increased memory footprint and synchronization. Heap ownership mechanisms annotate the allocated memory blocks with an extra tag including a pointer to the owner heap. Thus, when deallocation is performed, the DM manager checks the ownership tag and frees the memory block to its owner heap freelist. If heap ownership is disabled, the deallocated block will be inserted in the freelist of the thread initiated its deallocation. Heap ownership mechanisms require the usage of advanced synchronization and redirection mechanisms. The *Destination Heap* DT determines the heap that the memory blocks are going to be placed after their deallocation. The deallocated blocks can be placed either to the owner or to a shared global or to another assigned heap.

Category E. Inter-Heap Fragmentation Decisions refer to the memory blowup problem in MTh-DMM. Internal and external fragmentation are handled at the intra-heap level. The *Threshold* DT determines the inclusion of threshold mechanisms applied onto the heaps of the MTh-DMM. Thresholds enable the movement of a portion of free memory within a heap, to a global shared heap in order to be reused by another one. The heaps are extended with an upper threshold value, which is often expressed as a percentage of free memory against allocated memory [21]. When the free memory inside a heap crosses the specified threshold, a portion the available free blocks are moved to a global heap. In the general case, the threshold value can be different between heaps according to their allocation/deallocation patterns, thus exploration is needed . The *Number of Moved Free Blocks* DT determines the portion of free memory that is going to be moved from the heap's freelists to the shared global heap. The moved free blocks are placed to the freelist of the global heap in order to be ready for use from another heap.

2.4.2. Intra-Heap Level Design Decisions

Fig. 2.3b depicts the intra-heap level DTs. Each type of heap (i.e. local/global, single/concurrent, per-thread/shared) of a MTh-DMM can be further customized according the intra-heap DTs. They are classified into the following six main categories: (i) Block Structure, (ii) Pool Organization, (iii) Block Allocation, (iv) Block Deallocation, (v) Splitting Blocks, (vi) Coalescing Blocks. These DTs match with the decisions found in the single-processor single-threaded applications DMM design space [30]. Thus, we describe in a brief manner the intra-heap level DMM decisions.

Category F. Intra-Heap Block Structure Decisions handle the data structures which organize the memory blocks inside each heap of the MTh-DMM. The *Block Structure* DT

includes the combinations of Dynamic Data Types (DDTs) required to represent and construct any dynamic data representation inside a heap. The *Block Sizes* DT includes the different sizes supported by each local heap. Many DM managers i.e. Buddy, Segregated Fit etc. possess many different block sizes (fixed or not) [34]. The *Block Tags* and the *Block Recorded Info* trees define the added information inside each block. For instance, the *Pointers* leaf takes into account the different pointers stored within the blocks in double linked lists, heap ownership info, etc.

Category G. Intra-Heap Pool Organization Decisions define per heap pools' organization i.e. single pool, one pool per size, traversing order etc. The *Pool Structure* DT specifies the various pools of each heap and their internal DDT structures [92], [93]. In the *Pool Structure Based on Block Size* DT, the leaf one pool per size means that there is one specific pool per defined block size. For example, if we have defined 3 different block sizes (8-byte blocks, 64-byte blocks and 128-byte blocks), then we have three different pools. On the other hand, one common pool for all block sizes would initiated. The *Pool Structure Based on Blocks Order* DT organizes the pools of each heap according to different traversing orders (LIFO, FIFO and address). In the Block tree, the one pool per block leaf means that there is one specific pool per block. For example, if we have defined one pool per block there would be a different pool for each block (an extreme but also possible case).

Categories H and I. Intra-Heap Block Allocation/Deallocation Decisions deal with the operations to satisfy the DM allocation and deallocation requests. The *Allocation Search Order* DT handles the traversing mechanisms inside each heap or each pool of the heap. The *Allocation Fit Algorithms* DT models the fitting algorithms (i.e. FirstFit, BestFit, etc.) [34]. *Destination Pool* DT handles the decisions about the destination (source or other pool) of the deallocated block inside each heap.

Categories J and K. Intra-Heap Splitting/Coalescing Decisions formalize in two almost identical sets the decisions to handle the techniques for coalescing and splitting blocks [34]. The *Block Size* DTs include the different sizes that the DMM use as minimum and maximum block sizes inside each heap. The *Splitting/Coalescing Frequency* DTs determine the threshold logic for coalescing and splitting the blocks. The *Splitting/Coalescing Triggering Criterion* DTs contain the reasons for the allocator to split or coalesce blocks. The *Destination Pool* DTs define if the new block, that is created from the splitting or coalescing mechanism, ends up in the same pool or not.

2.4.3. Decision Dependencies within the MTh-DMM Space

Inter-Heap Dependencies After the definition of the DTs for MTh-DMM, we identify their possible inter-dependencies. The selection of certain leaves in some DTs heavily affects the coherency of decisions in other ones. Thus, DTs possess various inter-dependencies. The whole set of inter-dependencies for the design space is shown in Fig. 2.4. The MTh-DMM inter-dependencies can be classified in two main groups. Excluding inter-dependencies (solid arrows) are generated when a DT disables either semantically or structurally the use of other categories, DTs or leaves. Linked inter-dependencies are generated in cases which a DT affect other DTs, but not disable their use. Single ending dependencies indicate that the source of the arrow affects its destination and must be selected first. Double ending dependencies mean mutual influence in which a decision order is not clearly identi-

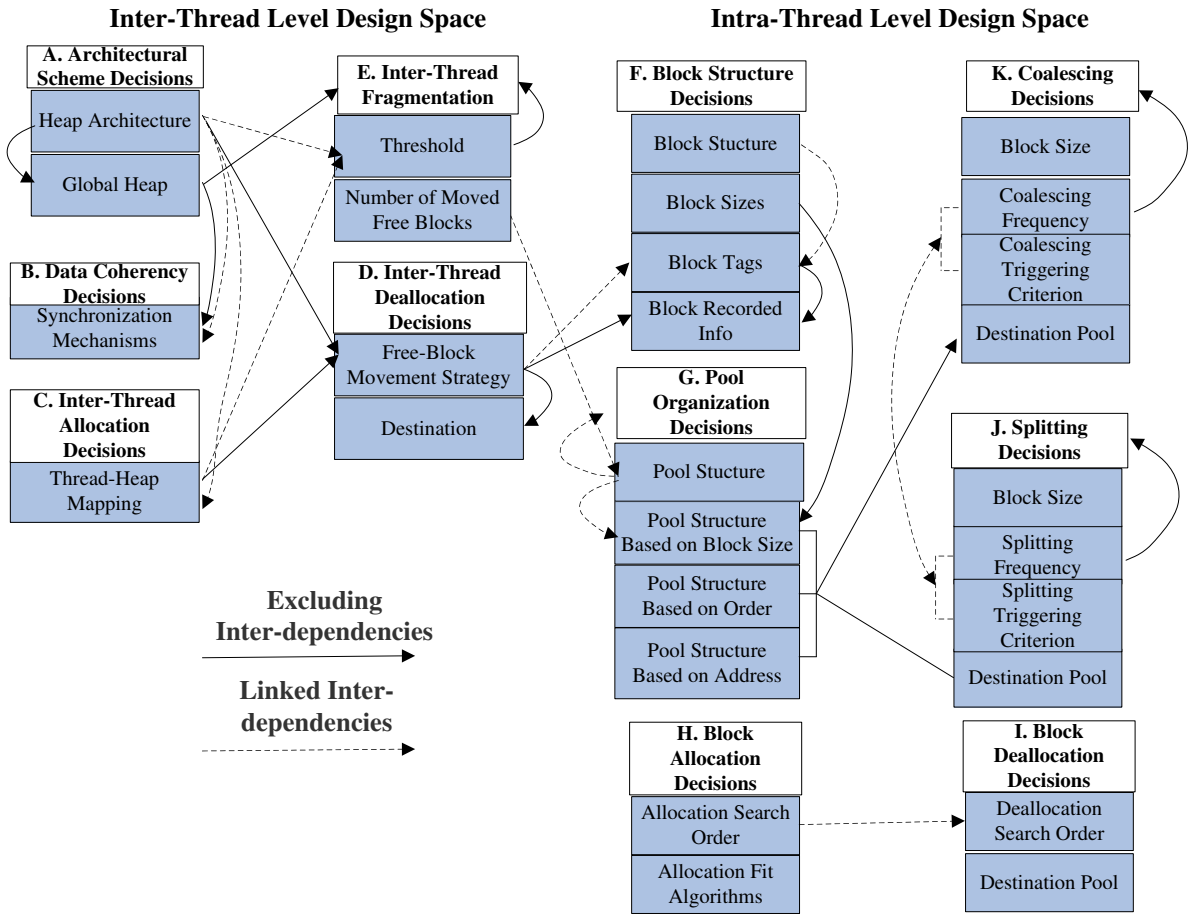


Figure 2.4.: Inter-dependencies among the DTs of MTh-DMM design space.

fiable.

We identified the following excluding inter-heap dependencies:

1. **A.1 → A.2: Inside the Architectural Scheme Decisions category, if either the Single Heap or the Pure Private Heaps leaf from the Heap Architecture tree is selected, then the Global Heap tree cannot be used. In the former case of Single Heap all the threads share the same heap so the existence of a separate global heap will be redundant. Additionally, Pure Private Heaps by definition exclude the existence of a global heap.**

2. **A.1 → D.1: The Pure Private Heaps leaf of Heap Architecture tree eliminates by definition the With Ownership leaf in the Free Block Movement Strategy tree since each thread operates only on its dedicated heap.**

3. **A.2 → B.1: In the Architectural Scheme Decisions category, the leaf Yes of the Global Heap tree excludes the No Synchronization leaf of Synchronization Mechanisms tree in Data Coherency Decisions category. The global heap is a shared resource that each**

thread can access and so the consistency of their data has to be ensured.

4. **A.2 → E: The leaf No in the Global Heap tree of Architectural Scheme Decisions category excludes the whole Inter-Thread Fragmentation category since the deallocated blocks would not be able to shared and reused from other heaps of the multi-threaded dynamic memory manager.**

5. **C.1 → D.1: In the Inter-Heap Allocation Decisions category, the leaf All Threads in One Heap of the Thread-Heap Mapping tree disables the selection of the With Ownership leaf in the Free-Block movement Strategy tree found into the Inter-Heap Deallocation Decisions category since all the thread will share the same heap.**

6. **D.1 → D.2: Inside the Inter-Thread Deallocation Decisions category, the selection of Without Ownership leaf of the Free-Block Movement Strategy tree disable the selection of the Source leaf of Destination tree which is highly correlated with the heap's ownership.**

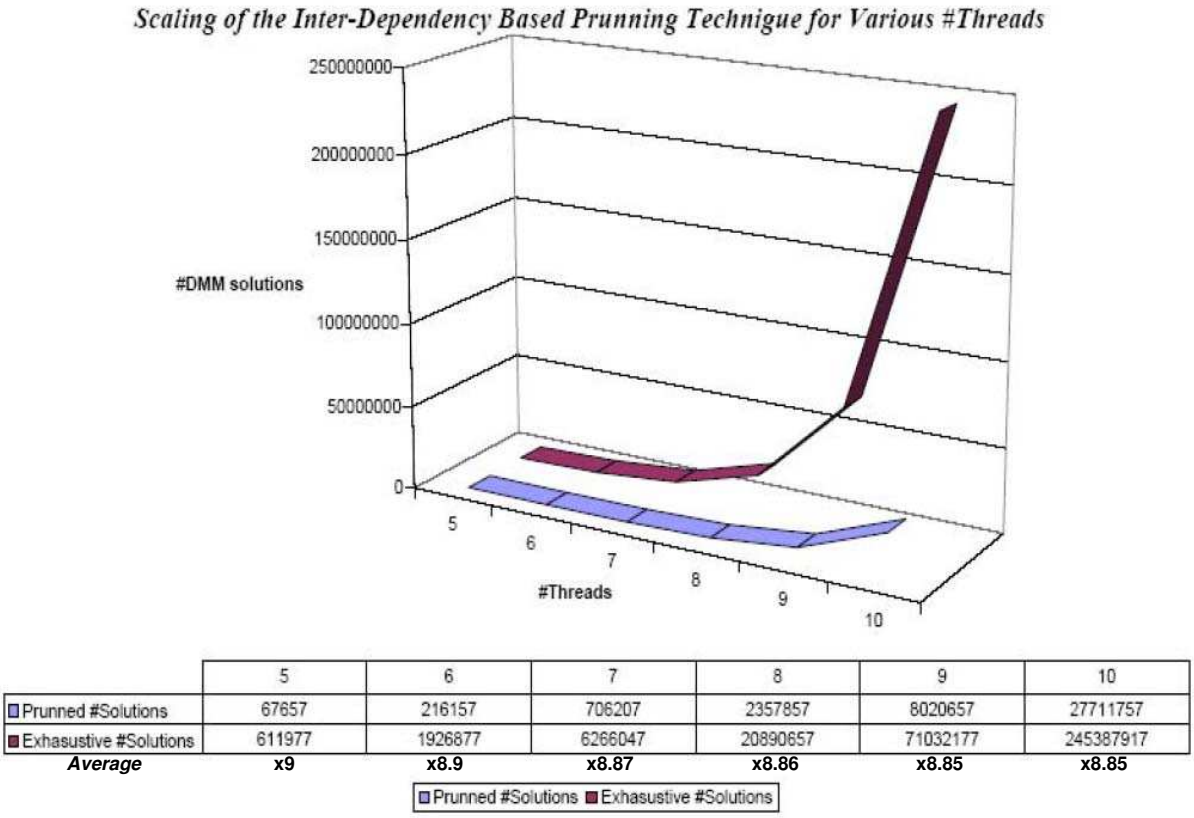
7. **D.1 → F.4: Also in the Inter-Heap Deallocation Decisions category, we identified an excluding interdependency between the inter- and the intra- thread design space. Specifically, if the leaf With Ownership is selected from the Free-Block Movement Strategy tree, then the Pointers leaf of the Block Recorded Info tree found in the Intra-Heap Design Space has to be enabled in order to incorporate the owner heap address for proper redirection of the deallocation operation.**

8. **E.1 → E: Finally, inside the Inter-Heap Fragmentation category, if the Without Threshold leaf in the Threshold tree is selected, there is no point to use the whole the category.**

Excluding DT inter-dependencies can be viewed in the context of *pruning directives*. Incorporating excluding independencies during MTh-DMM exploration greatly reduces the size of the search space. This reduction on the search space comes without compromising the quality of the solutions, since excluding inter-dependencies do not invoke any heuristic pruning on the solution space. Only the semantically or structurally meaningless solutions are excluded from exploration. These are solutions either not feasible or far from optimal. Thus, pruning the MTh-DMM design space according to the identified excluding independencies lead to significant exploration's acceleration. Fig. 2.5 depicts comparative results between pruned and exhaustive MTh-DMM exploration at the inter-heap level by increasing the number of threads. In case that exhaustive exploration is considered, an explosion on the size of the solution space is reported. On the other hand, pruned exploration based on excluding inter-dependencies delivers a rather constant acceleration rate of $\times 9$ with the solution space scaling in a less aggressive manner, in respect to the increasing number of threads.

Concerning the linked inter-heap dependencies, the following have been identified:

1. **A.1 → B.1: The decision taken in the Heap Architecture tree found into the Architectural Scheme Decisions category affects heavily the selection of the Synchronization Mechanisms tree. For example, selecting the Pure Private Heaps architecture the No Synchronization leaf of Synchronization Mechanisms tree can be selected. However, in case that either the Single Heap or the Multiple Heaps architecture is selected,**



	5	6	7	8	9	10
Prunned #Solutions	67657	216157	706207	2357857	8020657	27711757
Exhasustive #Solutions	611977	1926877	6266047	20890657	71032177	245387917
Average	**x9**	**x8.9**	**x8.87**	**x8.86**	**x8.85**	**x8.85**

Figure 2.5.: Scalability of pruned in comparison with an exhaustive inter-heap exploration under various #threads

then either blocking or non blocking synchronization primitives has to be selected in order to ensure heap data coherency.

2. **A.1 → C.1: A leaf choice in the Heap Architecture tree influences the Inter-Heap Allocation Decisions category. For example, selecting the single Heap leaf guide the Thread-Heap Mapping tree to select the All Threads-One Heap mapping decisions. This is not the case when the Multiple Heaps leaf is selected in Heap Architecture tree, since this architectural selection triggers the selection of both the Mixed and Each Thread-One Heap leaves in order to be evaluated. A Pure Private Heaps Selection at the Heap Architecture tree implies that an Each Thread-One Heap mapping strategy will be invoked in the dynamic memory allocator.**

3. **A.1 → E.1: Heap Architecture tree also affects the Inter-Heap Fragmentation decisions concerning the final choice in the Threshold tree. Single Heap and Pure Private Heaps architectures impose the selection of the Without Threshold leaf, since in both cases a global heap is redundant to the dynamic memory manager. However, in case of the Multiple Heaps leaf is selected, from the Heap Architecture tree, all the possible combinations form legal decisions.**

4. **C.1 → E.1: Inside the Inter-Heap Allocations Decisions category, the decision concern-**

ing the Thread-Heap Mapping tree affects heavily the Threshold tree in Inter-Heap Fragmentation category. In general, the decision to imply or not threshold mechanisms to the multiple heaps or single heap of the dynamic memory manager is sensitive to the dynamic behavior of the multi-threaded application. However, the decision of the thread to heap mapping has to been taken before the existence or not of thresholds mechanisms is evaluated.

5. D.1 → F.3: The Free-Block Movement Strategy tree found into the Deallocation Decisions of the Inter-Heap design space influences in a strong manner decisions in the Intra-Heap design space. The selection of the With Ownership leaf in Free-Block Movement Strategy tree affects the Block tags tree in Block Structure Decisions (Intra-Heap Decision) by imposing the usage of either a single header or a boundary tag in respect to the allocation/deallocation behavior of the dynamic multi-threaded application.

6. E.1 → G.1: The selection of the With Threshold leaf in Threshold tree affects also the structure of each pool inside the heaps (Pool Structure tree). Specifically, there is trade-off between the move operations and mechanisms at the inter-heap level and the dynamic behavior of the application, which determines the organization inside each heap at the intra-thread level. For example, the appliance of threshold mechanisms implies a SLL structure in order to perform the move operation fast and with a little number of memory accesses but the allocation behavior assigned to a heap implies the usage of a DLL structure for efficient manipulation of the dynamically allocated data. Given that the decisions made in the inter-heap level are propagated, as constraints to the intra-thread level the selection concerning the Threshold tree will be made first and proper cost functions concerning the performance, fragmentation and energy will further refine the pool structure inside each heap.

Intra-Heap Dependencies In the intra-thread design space the following excluding inter-dependencies have been identified [30]:

1. F.3 → F.4: Inside the Block Structures Decisions category, if the None leaf from the Block Tags tree is selected, then the Block Recorded Info tree cannot be used. Clearly, there would be no memory space to store the recorded info inside the block.

2. F.2 → G.2: The One Size leaf from the Block Sizes tree, excludes the use of the Pool Structure Based on Size tree in the Pool Organization Decisions category. This occurs because the one block size leaf does not allow us to define any new block size.

3. K.2 → K: As in the previous cases, if the Never leaf is chosen inside the Splitting/Coalescing Frequency tree of the Splitting/Coalescing Decisions categories, there is no point to use the whole Coalescing and Splitting blocks categories.

4. G.{2, 3, 4} → {K, J}.4: Finally, the sum of the orthogonal trees Pool Structure Based on Size/Order/Access inside the Pool Organization Decisions category influence the Destination Pool trees of Splitting and Coalescing categories, since the election of the One leaf in all of them forbids the option to use the Other pool leaf in both cases.

Concerning the linked intra-heap dependencies, the following have been identified:

1. **G.1 → G: The decision taken in the Pool Structure tree affects heavily the whole Pool Organization Decisions category based on criterion category. This happens because there are data structures that limit or do not allow the pool to be divided in the complex ways that the criterions of this category suggest.**

2. **H.1 → I.1: If a leaf choice is made in the Allocation Search Order tree within Block Allocation/Deallocation Decisions categories, it heavily affects the other since they are strongly tied together. Therefore, it is highly dubious that they follow different orders inside the same pool, although our search space does not restrict the designer to do it. Nevertheless, the Allocating blocks category has got more influence and involves more complex behavior in the final characterization of the dynamic memory manager. It is due to the existence of the Allocation Fit Algorithms tree inside it (clearly, the presence of the Allocation Fit Algorithms tree increases the cost of the Block Allocation Decisions category compare to Block Allocation Decisions one in memory accesses, time consumption (performance) and overall power in general), which does not appear in the Block Allocation Decisions category. Therefore, the dashed arrow is unidirectional and goes from the Block Allocation Decisions category to the Block Deallocation Decisions category, as it is depicted in Fig. 2.4.**

3. **J.{2, 3} → K.{2, 3}: The Splitting/Coalescing Frequency and Triggering Criterion trees inside the Coalescing or Splitting Decisions categories strongly influences each other. However, they are linked together with a double dash arc (in Fig. 2.4 there is only one double dashed arc between the two different trees with double dashed arcs, otherwise the figure would be too difficult to read). However, it represents two arcs, one for each pair of trees with the same name in the two categories) since it is not possible to decide which category has more influence in the final result. In fact, it depends on the complexity we want to give to each category. For example, if we decide to use the External Fragmentation leaf as the coalescing criterion in the Coalescing Decisions category, but the Always at Deallocation leaf for the Splitting Decisions category, then the Splitting category has much more impact in the cost. However, it could be also the other way around. Thus, it is specific for the application under study and maybe be decided based on other features of the system (e.g. final platform, memory hierarchy, etc.), which be analyzed for each particular case.**

2.5. Constrained-Orthogonal Exploration Methodology for Customized MTh-DMM

The goal of MTh-DMM exploration is to generate Pareto sets [28] of customized MTh-DMM, tailored to the designer's constraints and the application's specific needs. Each possible MTh-DMM solution can be modeled as a vector of DT's decisions. However, exhaustive search based on factorial enumeration of the number of DT's leaves explodes the size of the MTh-DMM solution space i.e. up to 612000 solutions for a 5-threaded application. Inspired by [41], we propose a methodology based on the constraint-orthogonal partition of the MTh-DMM design space, in order to handle the exploration's complexity. We observe that the semantics of the inter- and intra-heap design subspaces are constraint-orthogonal. That is because, inter-heap level DTs includes decisions of globalized and shared policies, while intra-heap level defines heaps' local customization policies in respect to the inter-heap decisions. Thus, instead of exploring the unified MTh-DMM design space, the

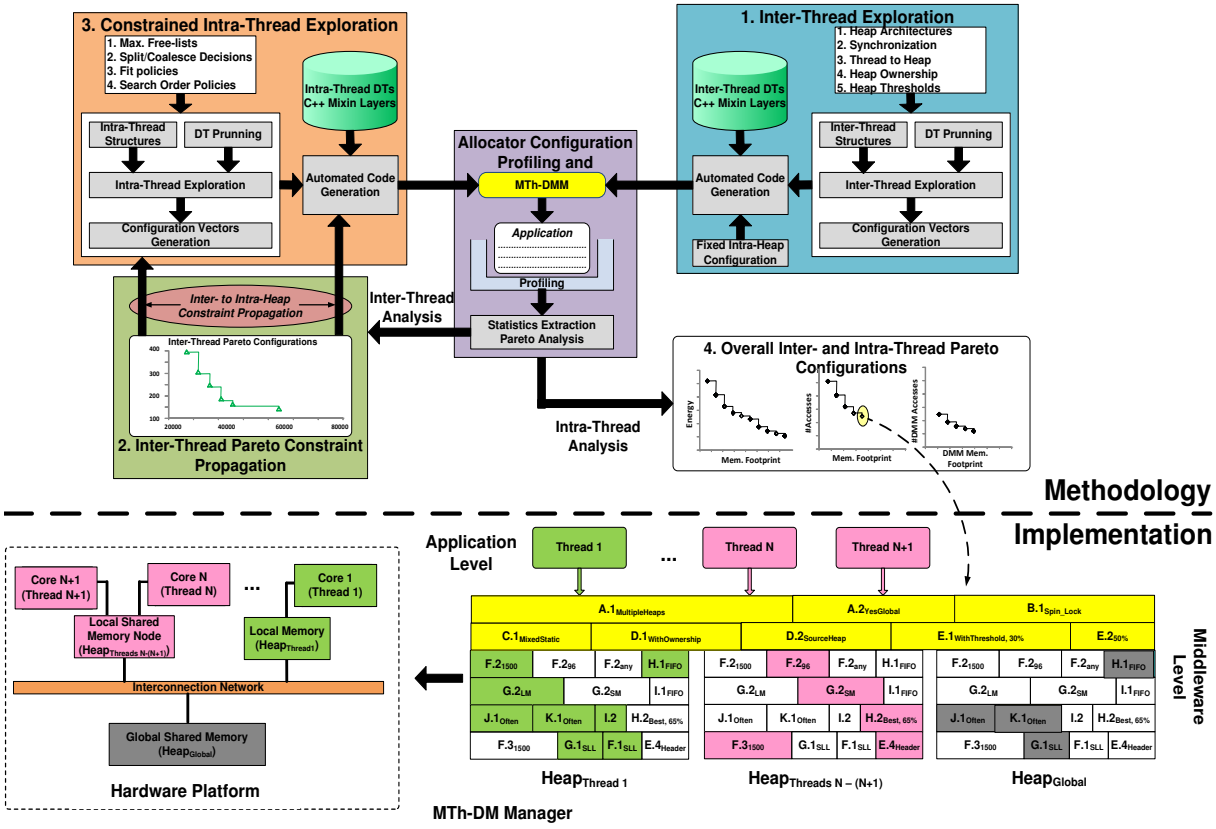


Figure 2.6.: MTh-DMM exploration methodology and tool.

exploration is partitioned in two constrained-orthogonal problems. The first problem refers to the exploration of inter-heap decisions, while the second one to the exploration decisions available in intra-heap design space. The inter-heap MTh-DMM exploration problem generates a Pareto set of solutions. These Pareto configurations are propagated as constraints to the intra-heap level exploration for further customization.

The major steps of the proposed exploration methodology are summarized as follows:

1. Given the dynamic application, its native source code is annotated with proper profiling constructs that capture the dynamic memory behavior of the application [94].

2. Inter-heap level exploration is performed to automatically generate the source code of valid MTh-DMM solutions. The Pareto inter-heap MTh-DMM solutions are extracted and propagated to the intra-heap level exploration.

3. Intra-heap level exploration customizes each heap individually found in the propagated inter-heap Pareto solutions. The combined intra- and inter-heap level Pareto MTh-DMM solutions are extracted and returned to the designer as the final customized Pareto MTh-DMM solutions.

Fig. 2.6 depicts the tool flow developed to automate the proposed MTh-DMM exploration

methodology. It consists of four steps: (i) inter-heap level MTh-DMM exploration, (ii) inter-heap level analysis and Pareto constraint propagation, (iii) constrained intra-heap level exploration, (iv) analysis and Pareto extraction of customized MTh-DMMs.

Step 1: Inter-heap level MTh-DMM exploration (Fig. 2.6) searches the corresponding solution space to find optimal combinations of design decisions for the dynamic application under study. The designer can either guide the exploration procedure by setting proper ranges to the exploration parameters (= ranges of decision leaf nodes found into inter-heap exploration) or let the tool perform exploration with default parameters. The exploration procedure effectively prunes the solution space by exploiting the inter-heap inter-dependencies. After the determination of the exploration parameters, the structures representing the examined decision trees of interest are initialized. The inter-dependency aware exploration loops are generated producing the inter-heap exploration script.

The inter-heap exploration script generates the MTh-DMM configuration set. A configuration vector lists each inter-heap decision that produced the specific solution. The MTh-DMM configuration vectors are fed into an automated code generator module that produces the C++ implementation of each dynamic memory manager according to the specified decisions. The code generator is linked with a C++ library containing modular software implementations [42] of each decision found into the inter-heap DTs. Intra-heap specific DMM decisions are not taken into account at this level of exploration. Thus, the internal structure of the heaps is not customized. A base FirstFit heap with a general FIFO-based freelist has been considered for servicing DM requests.

The source code of each inter-heap MTh-DMM solution is linked to the dynamic application source code in order to evaluate its impact on the overall performance metrics. To collect and analyze statistics of the dynamic application when various MTh-DMM managers are invoked, the application's source code is instrumented according to [94]. During application execution, we capture the memory behavior (dynamic and static) of the overall application and each thread separately. By the term simulation, we mean compilation and execution of the application under differing DMM configurations. Using the same input trace each automatically generated MTh-DMM solution is evaluated. For each examined solution we collect statistics concerning: (i) the number of memory accesses (both of the overall application and of the DMM manager isolated), (ii) the maximum memory footprint requested by the both the dynamic application or the DMM manager individually, (iii) the execution time (in usec) for the overall application, iv) the per-thread predominant block sizes (information used during intra-heap exploration and customization). This evaluation procedure is used for both inter- and intra-heap customized solutions. From these primitive statistics, we can evaluate other platform dependent metrics too, i.e. energy consumption.

Step 2: A Pareto analyzer module is invoked to extract the MTh-DMM solutions presenting the most efficient trade-offs. In case that the statistics collection is performed to evaluate the inter-heap explored DMM solutions, the extracted Pareto points form the Pareto constraints to be propagated to the intra-heap level exploration tool. In case that the evaluation is performed onto the intra-heap level customized MTh-DMM solution, the extracted Pareto curve includes the final Pareto MTh-DMM configurations, which are returned to the designer.

Step 3: Intra-heap level exploration tool performs the extra refinement of the inter-heap Pareto solutions. Each inter-heap Pareto solution is propagated to the intra-heap explo-

ration tool forming its constraints. Intra-heap exploration customizes the internal heap structure considering as fixed the inter-heap decisions defined in the propagated Pareto point. The software architecture of intra-heap exploration tool is similar to the corresponding inter-heap exploration tool. *Each instantiated heap of the inter-thread propagated Pareto set has to be customized.* Since each heap serves the DM requests of certain threads a homogeneous customization and exploration strategy it would lead to suboptimal solutions, except the case that all the threads presents the same dynamic behavior. The latter case corresponds to synthetic or simplified benchmark programs (i.e. threadtest.c [21]) rather than to real-life multi-threaded applications. However, the exploration and customization of each instantiated heap grows exponentially the complexity of the problem, since each possible heap customization of *Heap_A* has to be explored together with each possible heap customization of *Heap_B* for a two heap Pareto inter-heap solution. *We address this complexity explosion through two different strategies considering intra-heap exploration: (i) an access-oriented (AO) heuristic and (ii) a footprint-oriented (FO) heuristic*, according to the guidelines proposed in [30] for single-threaded DMM customization.

The intra-heap AO heuristic excludes from the exploration the intra-heap level decisions that inherently increase the number of memory accesses. Thus, splitting and coalescing decisions are not taken into account during exploration since they both require an increased number of accesses in order to generate the coalesced/splitted blocks. Similarly, FirstFit de/allocation strategy is preferable than the BestFit one. In addition, the instantiation of application specific fixed-sized free-lists inside each heap reduces the number of memory accesses. The fixed-sized freelists memory allocation requests are served with less memory accesses than in non-fixed-sized freelists, since in the latter case the DM manager has to traverse and search inside the DDT structures of the freelist in order to find the fitting memory block.

On the other hand, in the FO heuristic splitting/coalescing decisions are taken into account for various ranges of interest, since they both target to lowering the memory waste by generating at the runtime new proper memory blocks for the dynamic memory requests. In the same sense, BestFit allocation strategies are invoked for exploration. By searching for better block fittings the memory footprint is reduced since the better the fitting the lower the memory waste during allocation (through reducing internal fragmentation). As in the AO heuristic, the instantiation of fixed-size free-lists inside each heap has positive impact on the maximum requested memory footprint. In case of free blocks being accommodated into fixed-sized free-lists, then a memory request for allocation of that size will be served with zero memory waste.

Step 4: Each intra-heap customized MTh-DMM solution is revaluated by reusing the tool described in Step 2. The final Pareto curve is extracted and returned to the designer consisting of inter- and intra-customized MTh-DMM solutions.

2.6. Case Study: Evaluating Custom MTh-DMM for a Multi-Threaded Wireless Application

We evaluate the effectiveness of the proposed approach based on a real-life case study of a dynamic multi-threaded wireless application. The application consists of 5 kernels which

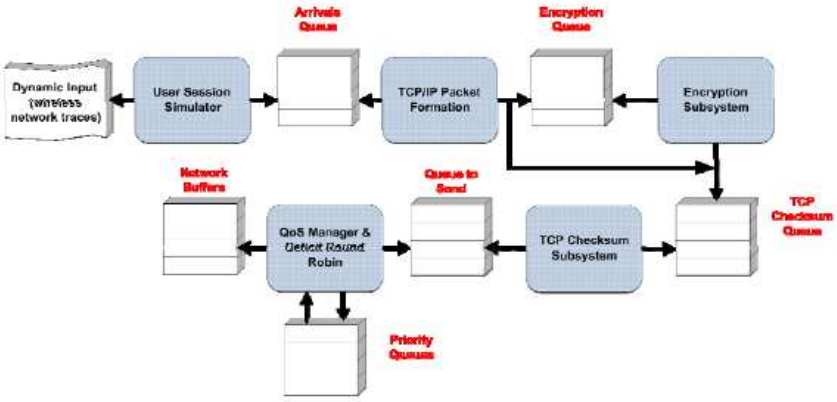


Figure 2.7.: The multi-threaded application used to evaluate the multi-threaded allocators. The boxes represent the different threads/kernels that communicate through asynchronous FIFO queues.

are triggered by wireless streams. Each kernel corresponds to a thread and communicates asynchronously with the other threads. The application was mapped, profiled and measured on top of a Linux multi-threaded environment for an Intel Quad Core MPSoC platform operating at 2.66 GHz. Our evaluation criteria are: (i) memory accesses, (ii) footprint and (iii) allocator code size.

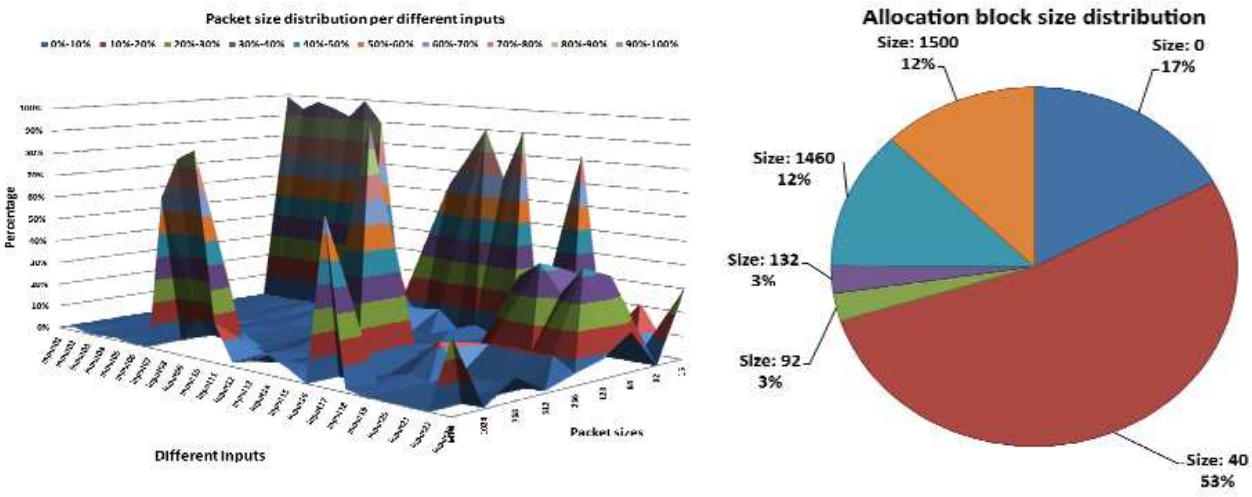
2.6.1. Description of the Dynamic MTh-Application

In order to validate our approach, we have modeled the following concurrent threads in an application (presented in Fig. 2.7), which are triggered by wireless streams in a Linux multi-threaded environment, relevant for embedded systems (each application kernel is executed on its own independent thread and communicates asynchronously with the other threads. A detailed description of the application, the profiling constructs, the measurement procedure etc. can be found in [94].

All the communication queues have locking mechanisms to ensure proper synchronization between threads:

1. **Simulated VoIP, FTP and Web browsing activity. This thread feeds into the application each packet of the input trace. The information available for every packet is: timestamp, source and destination IP addresses and port numbers, and size. This kernel just allocates the necessary memory space for the packet's data (not including the header) and thus simulates user applications without their corresponding processing requirements.**

2. **TCP/IP packet formation (it builds the complete TCP/IP packet filling in the header fields). Reflecting the entry point to the operating system like a write() system call, this thread builds the complete packet filling in the header fields that are available (source and destination IP addresses and port numbers). The total size of the packet is increased in 40 Bytes due to the headers. This kernel then writes the new packet**



(a) (b)

Figure 2.8.: Multi-threaded application characterization a) Network packet size distribution, b) Block sizes distribution.

into the queue for encryption or the queue for TCP checksum, according to whether the connection is encrypted or not.

3. **Encryption (packets that belong to an encrypted connection are processed with the DES algorithm). This thread accesses the data field of the packet in blocks of 8 bytes, and after finishing the encryption work it pushes the packet into the queue for TCP checksum.**

4. **TCP checksum. Calculated applying the 16 bit one's complement sum to the whole TCP packet and the so-called "IP pseudo header" as described in the RFC793. The contents of the packet are thus accessed consecutively using 16-bit operations. Once the CRC field of the packet's header is filled it is handled to the next queue.**

5. **The QoS manager builds a prioritized list of destinations. When a packet arrives to this subsystem it is queued in one of the priority classes. Packets are extracted from them and forwarded to the network adaptor according to a simplified Deficit Round Robin (DRR) algorithm. When a packet is forwarded its queue credit is reduced proportionally to the size of the packet. The forwarding of packets to the network adapter is simulated as a copy to a circular buffer in memory that can be traced by our profiling and analysis tools.**

The memory accesses that occur when an input triggers the system are:

1. **When a thread receives a packet it is removed from the queue it resided, thus freeing memory space for new packets. When a thread finishes the processing of the packet, the packet is written to the corresponding queue. For example when the packet is formed, according to its destination port, it is forwarded to the entry queue of an encryption thread or to the queue of an Internet checksum thread. If the "destination"**

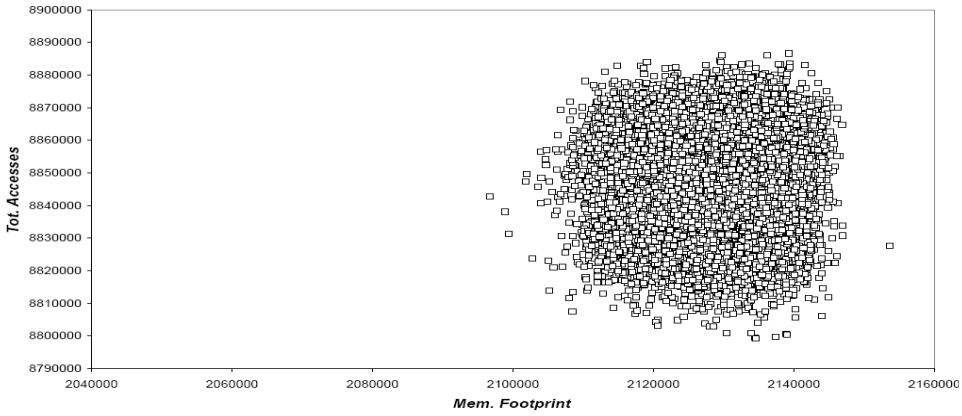


Figure 2.9.: Inter-heap solution space.

is full, the thread waits until there is enough space (in the meanwhile no new packets are processed by that thread). The system works as a pipeline; thus, no packet is discarded once the first thread accepts it.

2. **A thread processes a packet. Each thread performs different operations on the packet, accessing specific packet fields, thus performing different memory accesses. For example, a TCP/IP packet formation thread adds the packet header (size of 40 Bytes) to the raw data, thus forming a packet, writing relevant values the corresponding fields (IP addresses, port numbers, etc.). An encryption thread would encrypt all the data of the packet, etc. This type of processing can be potentially performed by one thread while the DMA performs the transfers for other thread.**

   The application dynamically allocates and deallocates data according to the characteristics of the incoming network trace. For example the distribution of the incoming packet sizes is shown in Fig. 2.8a, whereas the size distribution of the allocated blocks is shown in Fig. 2.8b.

2.6.2. Evaluating the MTh-DMM Solution Quality

A solution space of 67,655 valid and semantically disjoint MTh-DMM configurations has been generated. Fig. 2.9 displays a 2D diagram of the solution space characterized through the total number memory accesses versus the maximum memory footprint needed for each MTh-DMM configuration. Each point corresponds to a unique MTh-DMM solution applied to the network application. From the 2D diagram of Fig. 2.9, only the points delivering the best #Accesses vs. Footprint trade-offs are needed to be further explored. The best trade-off values (Pareto points) are extracted and propagated as constraints to the intra-heap level (AO or FO) exploration.

Fig. 2.10 depicts the three Pareto curves generated after the implication of each exploration (inter-heap, AO and FO intra-heap). The customization of inter-heap Pareto solutions is graphically depicted through the inter-heap Pareto curve shifting towards MTh-DMM solutions with either less memory accesses (in case of AO heuristic) or lower required

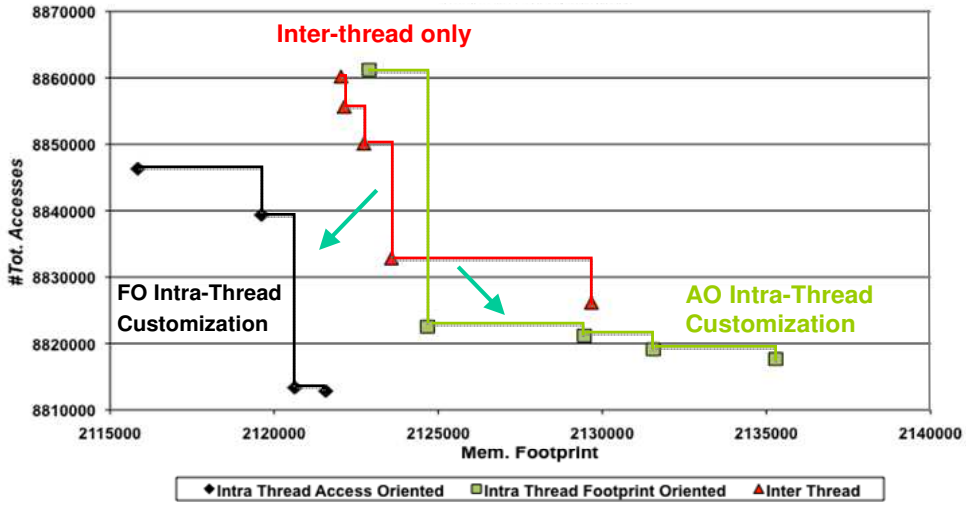


Figure 2.10.: Shifting of Pareto inter-heap solutions.

memory footprint (in case of FO heuristic).

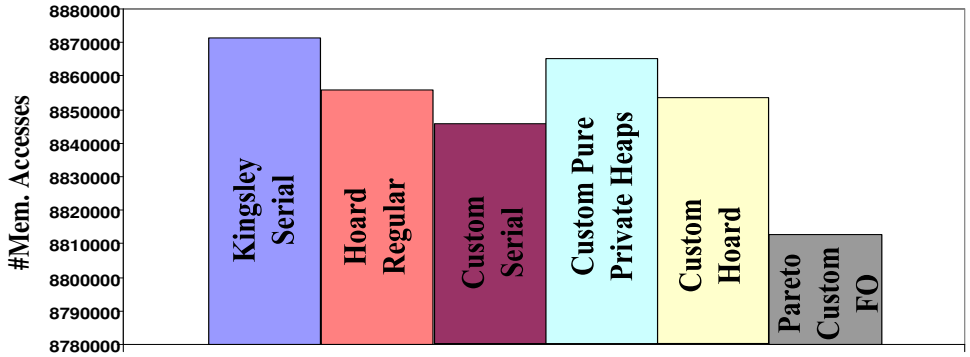
We have also compared custom implementations of MTh-DM managers with Windows-XP Kingsley [20] and Hoard [21] allocators (Fig. 2.11). We considered four variants of custom MTh-DMMs. Specifically, we considered a custom serial heap (Non-Pareto), a custom pure private heaps allocator (Non-Pareto), a custom Hoard-like DM manager (Non-Pareto) and best Pareto (custom) FO MTh-DMM solution extracted from the final Pareto curves (Fig. 2.10). The structure of these allocators is shown in Fig. 2.11a. Fig. 2.11 shows the efficiency of MTh-DMM solution generated by the proposed methodology. In terms of number of memory accesses, the FO MTh-DMM delivers an average reduction of 46K in memory accesses (Fig. 2.11b). In terms of memory footprint (Fig. 2.11c), the FO MTh-DMM is the second best after the Hoard allocator, with an average footprint reduction of 8.8KBytes. Hoard's efficiency in memory footprint is due to its internal heap organization with a large set of size-bins. However, Fig. 2.11d depicts that this complex heap organization makes Hoard the worst solution regarding the allocators code size. Enormous gains in terms of code size reductions of approximately $\times 14$ are reported for all customized MTh-DMM solutions in comparison with the general-purpose ones.

2.6.3. Evaluating Design Time Reductions

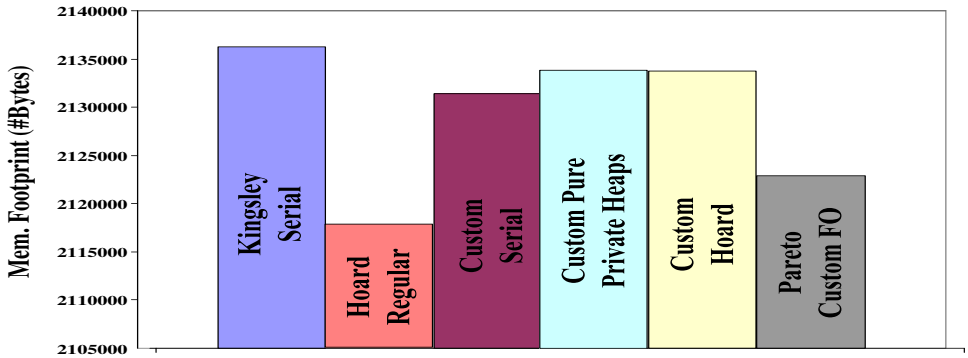
Finally, we evaluated the design time reductions delivered by the automated exploration tools over the hard-coded MTh-DMM exploration existed until now. In order to evaluate exploration's run-time, we consider the two following estimation models:

MTh-DMMs	Inter-Heap Decisions	Brief Description of Intra-Heap Decisions
Kingsley Serial	$A.1_{Serial} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{AllinOne} \rightarrow D.1_{w/o} \rightarrow D.2_{Global} \rightarrow E.1_{No} \rightarrow E.2_{None}$	128 Size Freelists→ SLL organization→ 1 Generic Heap with FirstFit and Coalesce/Split functions
Hoard	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/} \rightarrow D.2_{Source} \rightarrow E.1_{Yes,\ 50\%} \rightarrow E.2_{All}$	Per size bins (max 128 bins)→ SLL organization with FirstFit allocation.
Custom Serial	$A.1_{Serial} \rightarrow A.2_{Yes} \rightarrow B.1_{Mutex} \rightarrow C.1_{AllinOne} \rightarrow D.1_{w/o} \rightarrow D.2_{Global} \rightarrow E.1_{No} \rightarrow E.2_{None}$	1 Generic Heap with DLL organization→ FirstFit and Coalesce/Split functions
Custom Pure Private Heaps	$A.1_{PurePrivateHeaps} \rightarrow A.2_{No} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/o} \rightarrow D.2_{Other} \rightarrow E.1_{No} \rightarrow E.2_{None}$	1 Fixed Size Freelist→ 1 Generic Heap with DLL organization→ FirstFit and Coalesce/Split functions
Custom Hoard	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{SpinLock} \rightarrow C.1_{EachThreadOneHeap} \rightarrow D.1_{w/} \rightarrow D.2_{Source} \rightarrow E.1_{Yes,\ 70\%} \rightarrow E.2_{All}$	1 Generic Heap with SLL organization→ FirstFit and Coalesce/Split functions
Pareto FO	$A.1_{MultipleHeaps} \rightarrow A.2_{Yes} \rightarrow B.1_{Mutex} \rightarrow C.1_{StaticAssignment} \rightarrow D.1_{w/o} \rightarrow D.2_{Other} \rightarrow E.1_{Yes,\ 90\%} \rightarrow E.2_{50\%}$	3 Fixed Size Freelists→ 1 Generic Heap with SLL organization→ FirstFit and Coalesce/Split functions

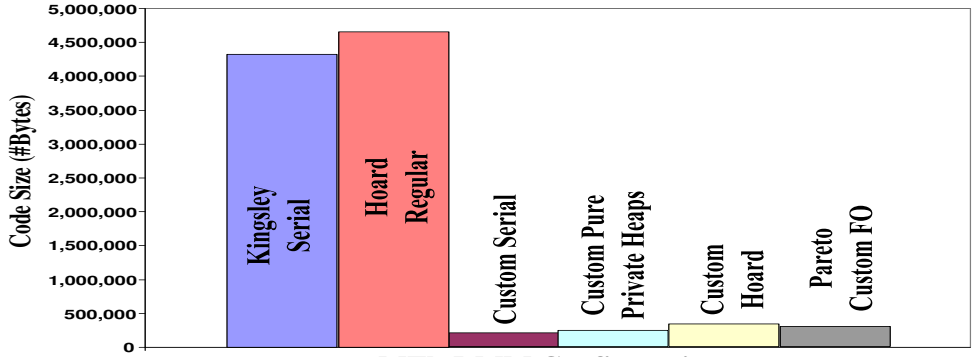
(a) Decision combination of DM managers



(b) #Mem. access comparison for various MTh-DM managers



(c) Mem. footprint comparison for various MTh-DM managers



(d) Code size of various MTh-DM managers

Figure 2.11.: Composition and comparative results for Kingsley-XP [20], Hoard [21] and various custom MTh-DM managers.

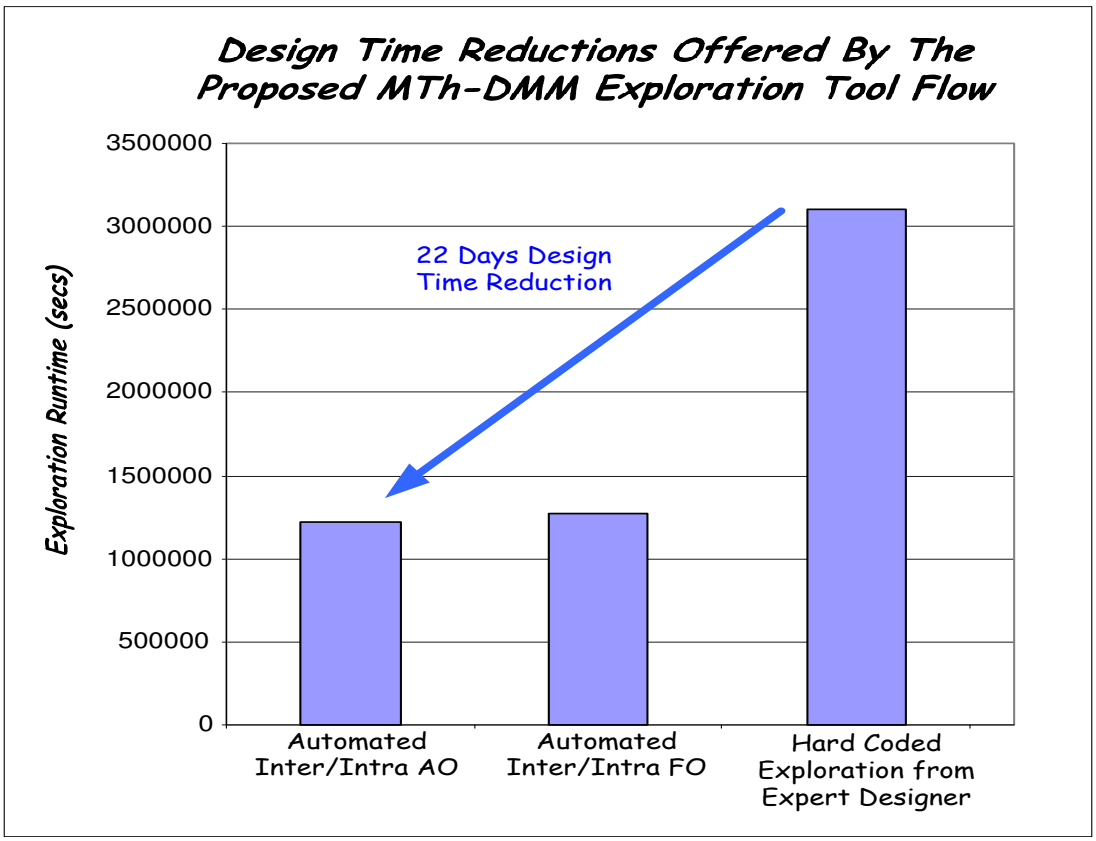


Figure 2.12.: Design time reductions.

$$T_{AutoExplor} = N_{Confs} \times (T_{Code\_Gen} + T_{Compile} + T_{Exec}) \tag{2.1a}$$
$$T_{HardCoded} = N'_{Confs} \times (T_{Code\_Writ} + T_{Compile} + T_{Exec}) \tag{2.1b}$$
$$T_{Code\_Gen} = 1sec \tag{2.1c}$$
$$T_{Compile} = 15sec \tag{2.1d}$$
$$T_{Exec} = 2sec \tag{2.1e}$$
$$T_{Code\_Writ} = 15min = 900sec \tag{2.1f}$$
$$N^{AO}_{Confs} = 67920 \tag{2.1g}$$
$$N^{FO}_{Confs} = 70391 \tag{2.1h}$$
$$N^{InterThread}_{Confs} = 67655 \tag{2.1i}$$
$$N'_{Confs} = 0.05 \times N^{InterThread}_{Confs} \tag{2.1j}$$

Eq. (1a) estimates the time needed each exploration approach (AO or FO) to extract final Pareto solutions. The time constants T_{Code_Gen}, $T_{Compile}$, T_{Exec} correspond to real values extracted from our exploration tools. The T_{Code_Writ} corresponds to the time needed for a designer familiar with the C++ library to code a MTh-DMM configuration. The N^{AO}_{Confs} and N^{FO}_{Confs} constants refer to the number of MTh-DMM configurations that were evaluated from the proposed tool flow according to the selected intra-heap heuristic *(AO, FO)*. For hard coded exploration,

we assume an expert designer exploring only the 5% of the solutions found in the inter-heap design space. This is an extra optimistic estimation for the designer. The proposed exploration methodology evaluates in a global manner (since both the inter- and the intra-heap design spaces are explored) up to $\times$2.5 faster than the expert designer. In absolute numbers, an expert designer needs approximate 36 days to explore the 5% of the inter-heap design space while the proposed flow explores the 100% of the inter-heap level design space and further customizes the extracted Pareto MTh-DMM solutions at the intra-heap level in less than 14 days (Fig. 2.12).

2.7. Conclusions

We presented a systematic methodology and the corresponding software tools developed to explore efficiently the design space of multi-threaded dynamic memory management for MPSoC platforms. Customized Pareto dynamic memory managers are generated delivering significant gains in memory footprint, memory accesses and code size.

3. Compiler in Loop Design Space Exploration During High Level Synthesis

Design space exploration during high level synthesis targets to the computation of those design solutions which form optimal trade-off points. This quest for optimal trade-offs has been focused on studying the impact of various architectural-level parameters during high level synthesis algorithms, silently neglecting the trade-offs produced from the combined impact of behavioral-level together with architectural-level parameters. We propose a novel design space exploration methodology that studies an extended instance of the solution space considering the effects of combining compiler- and architectural-level transformations. It is shown that exploring the design space in a global manner reveals new trade-off points and thus shifting towards higher quality design solutions is performed. We use a combination of upper-bounding conditions together with gradient-based heuristic pruning to efficiently traverse the extended search space. Our exploration framework delivers significant quality improvements without compromising the optimality (Pareto accuracy) of the discovered solutions, together with significant runtime reductions compared to exploring exhaustively the solution space at every allocation scenario.

3.1. Introduction

The continuous shrinking in transistor sizes, offered by current VLSI technologies, has led design methodologies towards high abstraction layers to handle the augmented complexity. High Level Synthesis is the driving force of design abstraction offering an automated and seamless path from high level behavioral specifications down to circuit level implementations. Being now in its mature phase [57], a continuously growing community of IC designers have adopted HLS techniques to tackle design complexity and meet tight time-to-market requirements. However, the rise of design abstraction exposed a large number of inter-dependent design parameters that have to be explored in order to discover the optimal solutions. Various implementations can be associated with each behavioral description and designers have faced the problem of reasoning on differing trade-offs among the set of design parameters. Thus, efficient exploration methodologies accompanied with automated tools are of great importance for a quick and concrete evaluation of the design space [29].

Design Space Exploration is the procedure that evaluates the solution space in order to return a set of Pareto-optimal [28] design points according to some design criterions (execution delay, circuit area, dissipated power). Pareto optimality [28] specifies that one design solution dominates the other when it is at least as good in all of the criteria and also strictly better in at least one of them. Thus, designers are interested mostly on Pareto optimal configurations. The evaluation of all possible design solutions to extract the Pareto frontier, is an extremely time-consuming task, since the size of the design space is usually huge. Thus, DSE methodologies are focused on the development of strategies for efficient traversal of the available design space.

In this article, we address the fundamental problem of exploring the Delay-Area trade-offs during HLS [44]. Delay-Area exploration curves can be generated through iterative scheduling the behavioral description of the application under different area or timing constraints. However, even if sophisticated scheduling algorithms are implied, such an approach is far from delivering the most efficient design solutions because the combination of code- and architectural-level optimizations are not considered as effective design parameters. In the common case, they are treated as user-guided pre-defined parameters during exploration, thus silently assuming the interaction between the structure of behavioral code and the architectural optimizations to be independent.

We show that using existing DSE methodologies, a large portion of Pareto-optimal design points are excluded from the final exploration curve. To address this inefficiency, a Compiler-In-the-Loop (CompInLoop) exploration methodology is proposed, which goes beyond the traditional exploration approaches by accounting an extended instance of the design space where both code transformations and architectural optimizations are treated as exploration parameters. We define the extended design space as a forest of decision trees, each modeling an exploration parameter. The search space explosion is managed by employing proper bounding conditions accompanied with a gradient-based look-ahead heuristic pruning scheme. Gradient-based look-ahead pruning trades accuracy/quality of the derived curve for exploration's runtime and vice versa, by characterizing large or small portions of design solutions inside the already bounded space. An exploration framework implementing the proposed methodology has been build extending the SPARK-HLS tool [43].

Extensive experimentation has been conducted based on real-life computationally intensive benchmarks to evaluate the effectiveness of our approach. In each case, a shift of the exploration's curve towards higher quality Pareto solutions is reported in comparison to the existing design exploration methodologies. Specifically, the proposed CompInLoop DSE approach delivers average gains of 68.28% in respect to the second best DSE approach, regarding the accuracy metrics of the approximated Pareto curve. In comparison to the full exhaustive exploration, exploration speedup of 8× in exploration's runtime is reported.

The rest of the article is organized as follows. Section 3.2 discusses the related work and the main differentiators of our approach. In Section 3.3, the observations that motivated this work are introduced. The architectural description of the targeted datapaths is presented in Section 3.4. Section 3.5 defines the design space. In Section 3.6, the algorithms that implement the proposed exploration methodology are presented and analyzed. Section 3.7 evaluates the proposed methodology through a rich set of experimental data. Finally, we conclude in Section 3.8.

3.2. Related Work

Design space exploration spans to almost all the design abstraction layers, i.e. system-level, platform-level, HLS level, Register Transfer Level (RTL) etc. In [29] an in depth analysis and categorization of existing exploration methodologies for various abstraction layers can be found. In this work, we targets to the field of design space exploration during HLS [95].

Design space exploration during HLS can be seen as a meta-layer that manages the solutions generated by operation scheduling. It strongly depends on the efficacy of the underlying scheduling algorithms, but it is not strictly coupled with them since it is not restrictive on the actual implementation of the DSE algorithm. Thus, many DSE frameworks have been proposed using various implementations for the scheduling algorithms [1], [2], [3]. In the former DSE methodologies, the exploration itself is performed exhaustively traversing the entire solution space. However, neither the structure of the behavioral description nor the architectural level optimizations (i.e. multi-cycle operations, operation chaining) are considered as exploration parameters, resulting in a large set of unexplored configurations, which usually dominates the derived exploration curve.

Exploring architectural optimizations during HLS i.e. operation chaining [6], CLK selection [96] etc. has great impact on the final datapath implementation. Conventional operation chaining targeted data-flow behavioral descriptions [6], [7] for performance improvement. Recently, operation chaining across conditional boundaries has been enabled [97] for designs with mixed data-flow and control-flow structures. Many researchers have also proposed to perform DSE by investigating the duality between timing- and resource-constrained scheduling problems [4], [8], [5]. The exploitation of the duality between the scheduling problems reports high quality results. However, the impact of the structure of the behavioral description (i.e. loop unrolling) has been ignored and operation chaining is only partially accounted [5] during exploration, leading to unexplored Pareto-optimal solutions.

During HLS phase, the code structure of the behavioral description heavily influ-

ence the final datapath's implementation [9]. Exploration on how compiler-level transformations impacts the code structure and the final datapath description have been considered in [10], [11]. However, exploration is focused on code-level transformations, without encountering the combined impact of code structure and architectural optimizations. In [12], loop unrolling has been explored in the context of multiple kernel instances that can be executed in parallel under imposed by a FPGA platform. The exploration is performed at the granularity of the entire kernel, thus the impact of unrolling on the execution of primary operations is not investigated. Recently, the impact of the code's structure has been evaluated in the field of Application Specific Instruction Processor (ASIP) design for exposing more beneficial instruction extensions [13], [14]. These works concentrate on the instruction identification problem rather than the design space exploration problem for Pareto-optimal coprocessor instantiations which this work targets to.

Regarding to the aforementioned research works, we can categorize the DSE approaches applied during HLS into three major classes illustrated in Fig. 3.1:

1. Compiler assisted DSE (CompAssisted) explores the impact of various resource allocation scenarios and uses the compiler-level optimization to generate optimized intermediate representation i.e. [1], [2], [3], [4], [5].

2. Compiler assisted with CLK selection DSE (CompAssistedCLK) extends the previous category by enabling CLK selection into the exploration loop i.e. [6], [7], [8].

3. Compiler-directed DSE (CompDirected) assumes two exploration loops, one for exploring code-level optimizations and another for exploring various resource allocation i.e. [9], [10], [11], [12], [13], [14].

The main differentiator of our approach is that we address an enhanced solution space considering trade-offs from an extended set of design parameters (Fig. 3.1d). CompInLoop exploration not only includes all the solutions generated from the aforementioned DSE strategies, but also reveals a new solution space which had remained unexplored until now (Section 3.3). By this way, the design space is explored in a more global manner, exhibiting new Pareto configurations. The proposed exploration methodology can be viewed as a meta-DSE strategy which enables the joint evaluation of a compile- and architectural-level decisions during HLS. Compiler-in-the-Loop exploration has been proposed in [98] for horizontally partitioned cache architectures for micro-processor design. Considering the HLS domain, Schafer and Wakabayashi [99] have proposed a similar approach in which the HLS tool is seen as a "black-box" manageable from the explorer. They employ user-defined parameter clustering to aggressively prune the available solution space at the up-front. Thus, the designer is responsible for defining the search spaces of interest. Here, we adopt a different approach by developing techniques to effectively prune non-promising solutions across the overall design space and not only

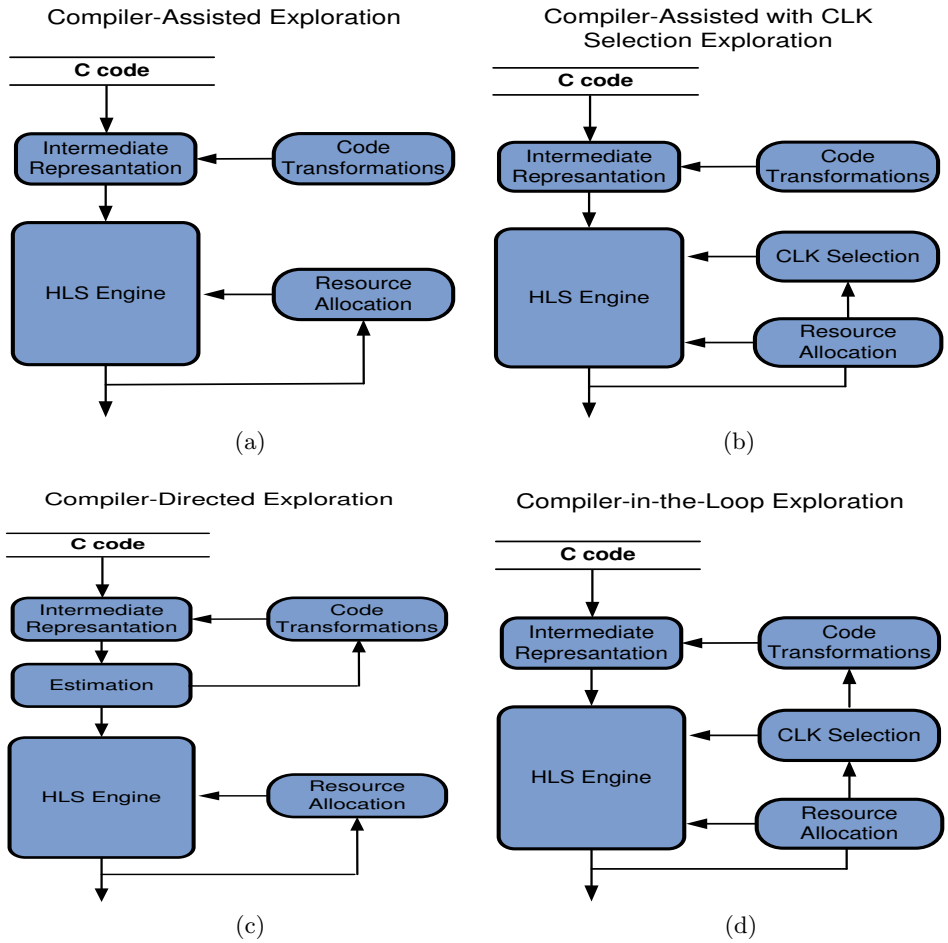


Figure 3.1.: Exploration strategies during HLS: (a) Compiler assisted exploration [1], [2], [3], [4], [5]. (b) Compiler assisted with CLK selection exploration [6], [7], [8]. (c) Compiler directed exploration [9], [10], [11], [12], [13], [14]. (d) Proposed compiler-in-the-loop exploration.

in specific regions.

We extend previous work by (i) integrating more accurate area models in the DSE framework, (ii) exploring a larger set architectural-level parameters including CLK selection, multi-cycle functional units and memory ports, (iii) modeling and defining the new design space and (iv) developing new exploration algorithms to handle the augmented complexity of the solution space.

3.3. CompinLoop Motivational Observations: Identifying Unexplored Pareto Solutions

This section demonstrates the quality improvements offered from the Compiler-in-the-Loop exploration. Specifically, we evaluate the proposed exploration approach over the existing DSE strategies. As driving application, we considered the 1D Discrete Cosine Transform (1D-DCT) kernel found into the specification of Joint Photographic Experts Group (JPEG) standard [100]. Given the behavioral description of the targeted application, the main objective is to return to the designer an exploration curve including the best delay-area trade-off points. Regarding such trade-offs, each design solution is included in a 2D-diagram, where the x-axis accounts for the area cost while the y-axis accounts for the delay.

The 1D-DCT kernel has been exhaustively explored considering (i) ComAssisted, (ii) ComAssistedCLK, (iii) CompDirected and (iv) CompInLoop DSE strategies, with an upper area constraint of $Area_{MAX} = 65000um^2$. Fig. 3.2 shows separately the search space explored by each DSE strategy.

Studying carefully the various search spaces reveals several key insights. First, the efficiency of each DSE, concerning the Pareto optimality, depends on the "visible" solution space. The varying visibility of the search space generates Pareto-optimality gaps (Fig. 3.2) among the DSE methodologies. Second, the coverage of a larger search space that leads to new Pareto trade-offs has to be take onto account the interaction of the variable parameters of the design space. By searching into a larger space does not guarantee that the exploration will move towards higher quality trade-offs, i.e. the depicted Pareto-optimality gap in Fig. (Fig. 3.2) between the CompDirected and the CompAssistedCLK DSE strategies. Thus, an efficient DSE strategy has to account not only for a larger search space but also for solution spaces that capture a larger set of interactions between the exploration parameters.

Compiler-in-the-Loop exploration methodology has been developed based on the aforementioned motivational observations. Thus, we target to an exploration methodology that is able to capture and evaluate the interaction between compiler- and architectural-level parameters. The adoption of the proposed DSE method leads to the revealing of a new search space, unexplored from the other exploration strategies (Fig. 3.2). This newly revealed search space includes the actual Pareto design solutions. It is worth noticing that CompInLoop exploration considers the extended superset of the other search spaces. Thus, it includes into its search space all the Pareto configurations generated from the alternative DSE strategies. For example, the Pareto solutions, generated from the CompAssistedCLK exploration in Fig. 3.2, are also explored in the case of CompInLoop exploration strategy. Since the number of explored design solutions is augmented, we focus on the development of algorithms for CompInLoop exploration which efficiently bounds and prunes the

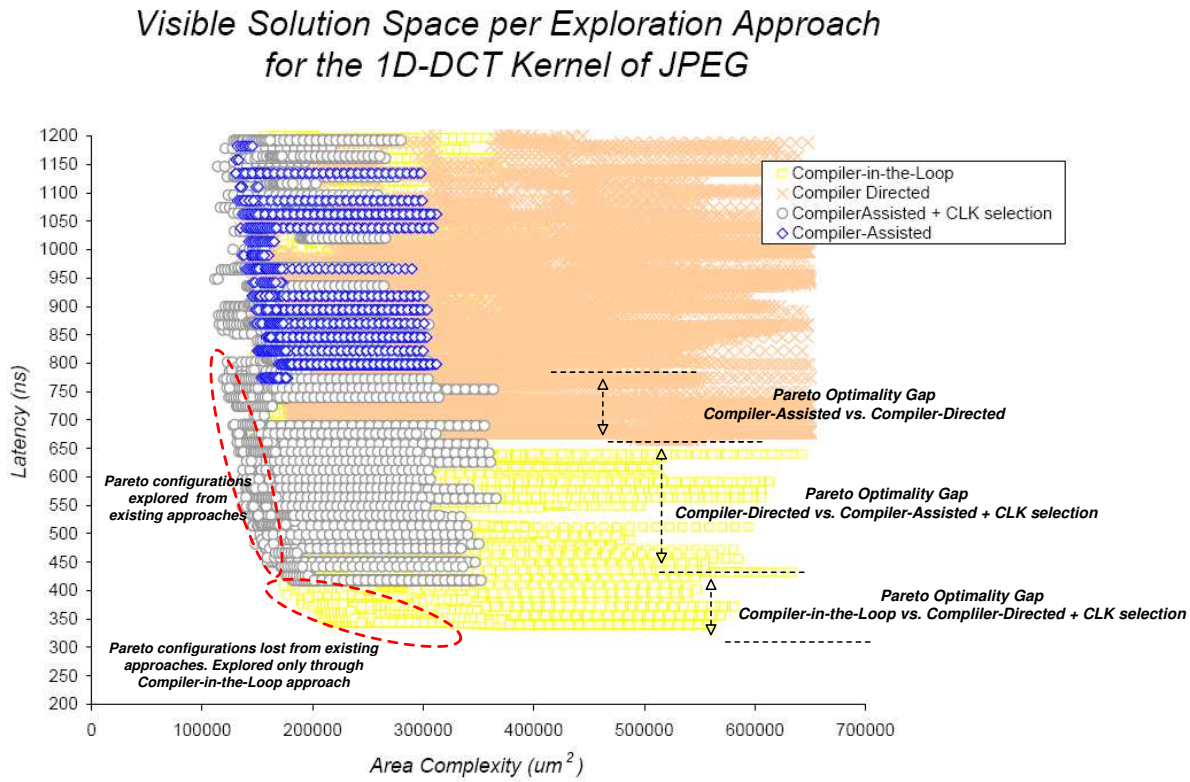


Figure 3.2.: Extend DSE's visibility through Compiler-in-the-Loop exploration.

available search space.

3.4. Area Estimation of the Targeted Architectural Template

In order to evaluate the area complexity of the explored solutions in a quick manner, area estimation is performed prior to logic synthesis (pre-synthesis area estimation). By this way, we may loose some estimation accuracy, since the result of gate sizing [101] is not taken into account, but we have large gains concerning exploration's runtime since the time consuming logic synthesis phase is bypassed. Pre-synthesis area estimation is a common practice in HLS exploration tools [95] for rapid area cost evaluation. However, most of the existing DSE frameworks [5], [11], [96], [10], [1] are based on rather simplified datapath models for area estimation, considering only the number of Alu and Mul units for area estimation.

We perform pre-synthesis area estimation based on a realistic datapath model. The adoption of realistic datapath model enables more accurate evaluation of the costs associate with each examined solution. Since the CompInLoop exploration approach considers compiler-level decisions, their effect onto the final datapath has to be evaluated. Compiler-level decisions have impact on datapath elements, i.e. number of allocated registers, complexity of the control unit [9] etc., which are not modeled efficiently using simplified area estimators.

Fig. 3.3 shows the architectural template that is used for area estimation. It is based on a typical structure found in hardware accelerators. It consists of (i) the datapath components (Alus and Muls) which perform the actual computation, (ii) the register bank for locally storing the intermediate results, (iii) the steering logic that moves data from the register bank to the datapath component and vice versa, (iv) the memory interface components (memory ports and Load/Store (LD/ST) units) to read/write data from/to the memory and (v) the control unit which generates and propagates control signals in a cycle-by-cycle basis. The total area complexity for the adopted architectural template is:

$$Area_{Total} = Area_{Datapath} + Area_{Regs} + Area_{MemIF} + Area_{Steering} + Area_{Ctrl} \tag{3.1}$$

The area complexity for the datapath, the memory interface components and the register bank is calculated as follows:

$$Area_{Datapath} = Alu \times Area_{Alu} + Mul \times Area_{Mul} \tag{3.2}$$

$$Area_{MemIF} = \#LD/ST \times Area_{LD/ST} + \#MemPort \times Area_{MemPort} \tag{3.3}$$

$$Area_{Regs} = \#Registers \times Area_{Register}^{Nbit} \tag{3.4}$$

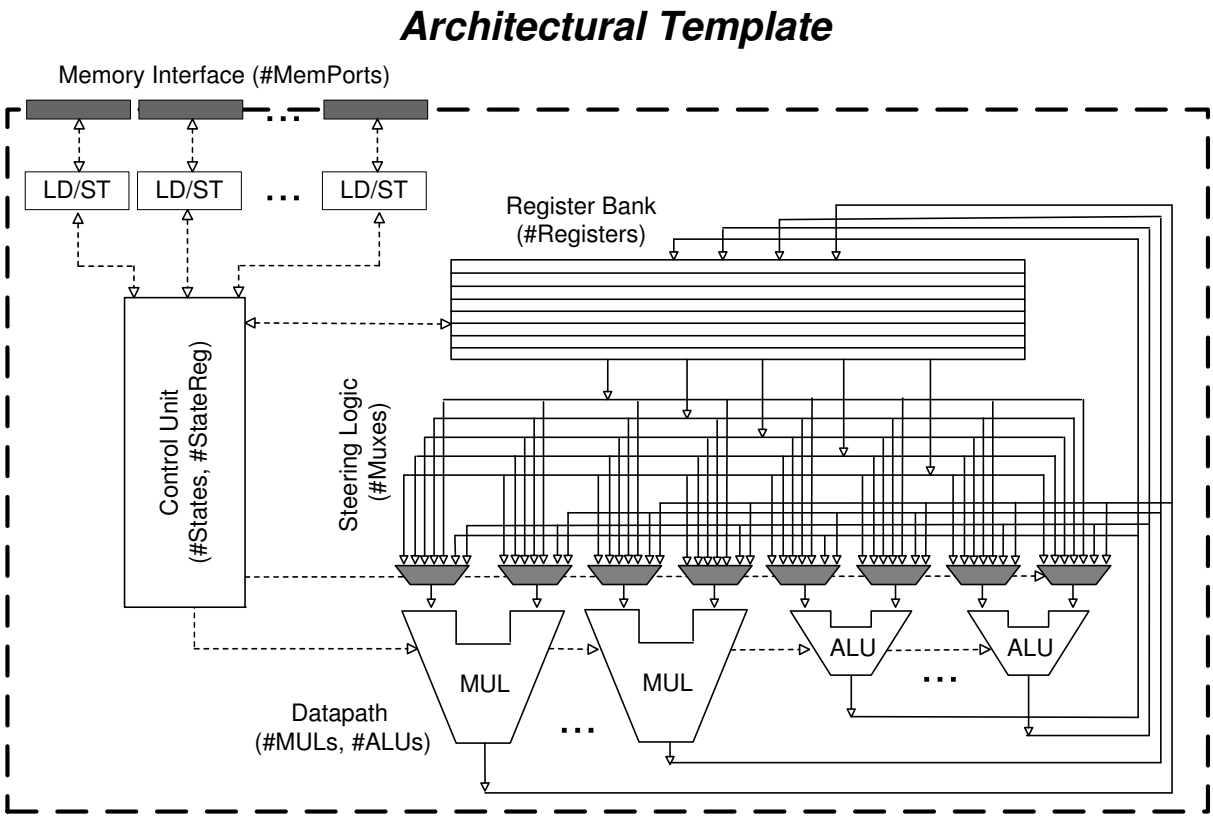


Figure 3.3.: Targeted architectural template.

The number of datapath components ($\#Alu, \#Mul$) and memory interface components ($\#LD/SC$) varies in respect to resource allocation scenario that each examined solution exhibits. Differing resource allocations generates different scheduling solutions, thus heavily affecting both the delay and the area cost of each solution. The size of the register bank ($\#Registers$) is extracted after the register allocation phase [44] which follows the operation scheduling of the behavioral kernel.

The area complexity of the steering logic ($Area_{Steering}$) is dominated by the number and the type of the multiplexors at the inputs of each FU. Since CompInLoop exploration takes place at a pre-synthesis level, we avoid performing interconnection optimization, thus estimating the area of steering logic in a coarse analytical manner. There are two main reasons that guided this decision. The first reason considers the impact of interconnection optimization on the exploration's runtime. Applying such optimization procedures for each examined design solution will greatly increment the runtime given the CompInLoop's extended design space. The second reason is due to the large impact that interconnection has in current VLSI technologies [102]. Since accurate interconnection optimization requires physical information, we leave such type of decisions to be made at lower abstraction levels (i.e. at the post-synthesis or post-placement level) than the one that CompInLoop exploration targets to. However, the proposed exploration methodology is not restrictive and it can be extended in a straightforward manner to incorporate high level interconnection optimization [103], [104].

We estimate $Area_{Steering}$ based on the following model:

$$Area_{Steering} = \#Mux \times (\#Operands_{PerMux} - 1) \times Area_{Mux2to1} \tag{3.5}$$

The $\#Mux$ is derived from the resource allocation of datapath components. Since we assume 2-input-1-output FUs, one multiplexer is allocated in each port (Eq. 6). We assume that each allocated multiplexer is decomposed into a tree constructed with 2-to-1 multiplexer cells. Since each 2-to-1 multiplexer routes two input operands, there will be needed $(\#Operands_{PerMux} - 1)$ 2-to-1 multiplexer cells. The $\#Operands_{PerMux}$ equals to the the number of different states that forms the FSM of the control unit (Eq. 7). That is because, in each state new data (operand) has to be propagated to the datapath components in order to perform the scheduled computations. A similar approach has been also used in [105] for estimating the area of steering logic in reconfigurable streaming accelerators. Thus:

$$\#Mux = 2 \times (\#Alus + \#Muls) \tag{3.6}$$

$$\#Operands_{PerMux} = \#States_{FSM} \tag{3.7}$$

Finally, the area complexity of the control unit is estimated according the area of (i) the registers storing the FSM states, (ii) the registers for the selection signals of the

Table 3.1.: Resource Characterization Based on TSMC 0.13 um Technology Library.

Resource Unit	Delay (ns)	Area (um^2)
ALU	3.00	4000
MUL	4.00	12000
LD/ST & MemPort	3.00	4200
Register 16-bit	0.15	550
Mux2to1	0.10	21
Register 1-bit	0.20	34

steering logic and (iii) the registers for the selection signals of the Alu components. The following formula occurs:

$$Area_{Ctrl} = Area_{States} + Area_{Mux}^{SelectionBits} + Area_{Alu}^{SelectionBits} \tag{3.8}$$

Given the results of operation scheduling, the FSM allocates a N_1-bit register to handle the encoded state space, where $N_1 = \lceil log_2 \#States_{FSM} \rceil$. In addition, a N_2-bit register is allocated to propagate the selection signals to the multiplexors of the steering logic, $N_2 = \#Mux \times (\#Operands_{PerMux} - 1)$. Finally, for each allocated Alu unit a N_3-bit register is allocated to control its operation, $N_3 = \#Alu \times (\lceil log_2 \#Operation_{Alu} \rceil$. Thus, the following equations stands for the calculation of left-hand side of Eq. 8.

$$Area_{States} = N_1 \times Area_{Register}^{1bit} \tag{3.9}$$

$$Area_{Mux}^{SelectionBits} = N_2 \times Area_{Register}^{1bit} \tag{3.10}$$

$$Area_{Alu}^{SelectionBits} = N_3 \times Area_{Register}^{1bit} \tag{3.11}$$

The presented area estimation model is based on the area measurement of some primitive components of the targeted resource library, namely the $Area_{Alu}$, $Area_{Mul}$, $Area_{LD/ST}$, $Area_{MemPort}$, $Area_{Register}^{Nbit}$, $Area_{Mux2to1}$, $Area_{Register}^{1bit}$. We derive the area complexity of each primitive component through post-synthesis characterization using the Synopsys Design Compiler synthesis tool [106]. Throughout this chapter, datapaths of 16-bit word-length and the TSMC 0.13 um standard cell library [107] has been considered in order to derive the area and delay estimation of the hardware resources. Table 3.1 reports the area and delay estimations of each primitive component based on 0.13 um library. Using the specific technology library is not a limiting factor for the proposed architectural template and exploration methodology, since they are general enough to incorporate resource libraries of various bit-widths, technology libraries and/or hardware resources.

3.5. Definition of the Design Space

The design space considered during CompInLoop exploration is composed by a set of design decisions. The combination of different design decisions generates different trade-off points. In order to generate efficient implementations, we have to systematically classify the design decisions which handle all the possible decision combinations. In this section, we model the design space considered in CompInLoop exploration by systematically classifying the available design decisions into DTs. Each design decision consists a root node of a different DT. All the possible parameters/values that can be taken for the root decision form the leaf nodes. We organize the DTs in an orthogonal manner in order to reduce redundancy of the examined solution space.

DTs have been successfully used to organize the design space found in the field of dynamic memory management [108], Chapter 2. The usage of DTs to model the CompInLoop design space targeting the HLS field, has a twofold aim: (i) DTs can be used as a conceptual analysis framework that enables the qualitative analysis and evaluation of different decision combinations. Through this conceptual analysis, exploration strategies can be formed to efficiently traverse the design space taking into account the inherent features of each design decision. (ii) Each possible design solution can be generated in a modular way binding together different design decision parameters/values found in the DTs. We have grouped the DTs forming the CompInLoop design space in two classes: (i) Compiler-level and (ii) Architectural-level.

3.5.1. Compiler-Level DTs

Compiler-level DTs (Fig. 3.4) capture decisions (transformations) imposed at the source code level of the behavioral description. These decisions do not alter in a direct manner the datapath's architecture. However, they propagate constraints that greatly affects the datapath's architecture.

The A.1, A.2 and A.3 are binary DTs, handling decisions on the appliance of source transformations. Strength Reduction (A.1), Copy and Constant Propagation (A.2) and Common Subexpression Elimination (A.3) are well known compiler-level optimizations [109]. These optimizations have beneficial results on the behavioral description, thus they are considered always enabled. In example, Strength Reduction performs replacement of costly (in terms of operator complexity) operations with an equivalent but less expensive operation set (i.e. a multiplication operation with fixed coefficients can be replaced by a combination of add/subtract and shift operations).

The A.4 and A.5 DTs concern loop unrolling decisions. They are orthogonal DTs

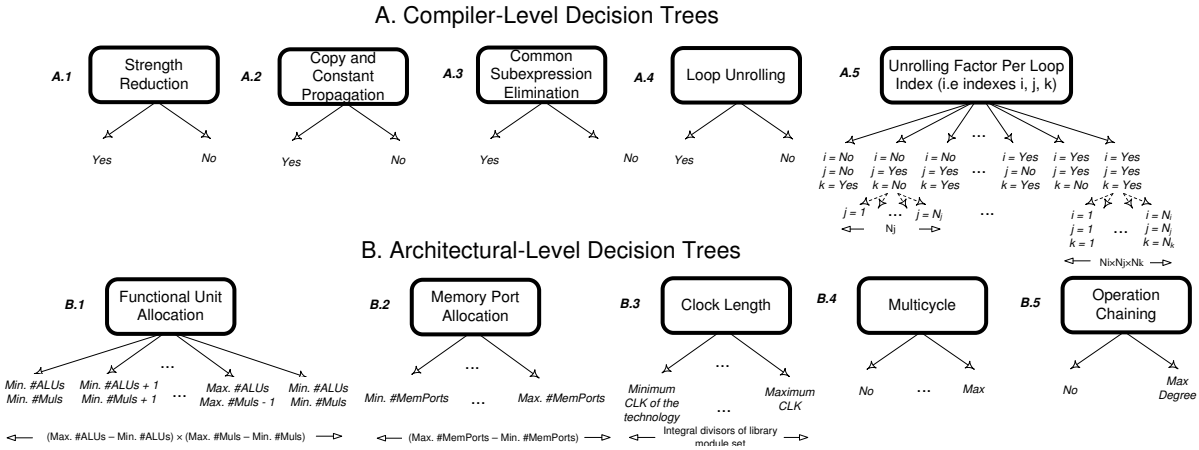


Figure 3.4.: Definition of the CompInLoop design space.

but highly correlated to each other. A.4 DT handles the decision whether to apply or not the loop unrolling transformation. The A.5 DT handles the decisions of which indexes (or index in case of single loop behavioral descriptions) and how many times (Loop Unrolling Factor, LUF) have to be unrolled. The number of leaf nodes for A.5 is application specific, since it depends on the number of loop indexes and their bounds found into the behavioral specification. In Fig. 3.4, A.5 DT is assumed for the case of three loop indexes (i, j, k) with N_i, N_j, N_k loop bounds.

The presented set of compiler-level DTs is complete for the SPARK-HLS tool [110] upon which we built the proposed DSE framework. In case that an alternative HLS tool is adopted supporting a reacher set of loop transformations i.e. loop fusion, loop splitting, loop tiling etc., each of the supported loop transformations can be added as extra DT in a straightforward manner. Without loss of generality, the following analysis considers the five presented compiler-level DTs.

3.5.2. Architectural-Level DTs

Architectural-level DTs include decisions with direct impact on the architecture of the final datapath. In particular decisions concerning resource allocation (FUs and memory ports), operator types and clock length selection are defined in this category.

The B.1 DT models the allocation scenarios considering the FUs of the datapath. In the same context, B.2 DT models the structure of the memory interface. We assume that each memory port is attached with a LD/ST unit. The designer provides the bounds of a range of interest in terms of minimum and maximum number of FUs and memory ports (i.e. Min./Max. ALUs, Min./Max. Muls, Min./Max.

MemPorts). Given the boundaries, allocation scenarios are derived from all the possible combinations inside that range (B.1 and B.2 leaf nodes). The number of leaf nodes is user specific, since the range of interest is specified by the designer. Each of the allocation scenarios included in B.1 and B.2 form an instance of the resource constraints that the HLS operation scheduling procedure has to satisfy during delay minimization.

The B.3 DT includes design decisions concerning the allocated clock frequency (CLK selection decisions). The clock length has a significant impact on the resulting design, since it directs the incorporation of multi-cycle (B.4) or chained operations (B.5) during scheduling. Exploring exhaustively the impact of each possible CLK length on the scheduling decisions is a very time-consuming task. We adopt the CLK selection methodology proposed by Blythe and Walker [96], which calculate a tighter set of CLK candidates. The set of the candidate CLK lengths (integer values) derives by taking the ceiling of the integral divisors of the delay associated with each functional unit. The minimum CLK length is determined by the minimum acceptable register-to-register delay, defined in each technology library. For example, TSMC 0.13 um technology library [107] assumes 1 ns as the minimum register-to-register delay propagation.

Operation multi-cycling (B.4 DT) and operation chaining (B.5 DT) decisions has a great impact on both the datapath structure and the operation scheduling. Multi-cycling enables faster CLK lengths by allocating more than one CLK cycle to each operation. Operation chaining removes the intermediate registers in data-dependent operations, thus enabling complex functional unit operations. It has also great impact on the datapath's implementation, i.e. augmentation of CLK period to incorporate the chained operators in a single cycle, less number of allocated registers etc. Both multi-cycling and chaining impact CLK selection decisions, by forcing the inclusion or the exclusion of specific CLK lengths from the candidate CLK set. Let us consider the case that the designer seeks to explore design solutions with (i) single-cycle Alus, (ii) multi-cycle Alus and (iii) chaining of two Alus. Assuming a 0.13 um technology library [107] and the ns as time-unit for CLK resolution, the set of candidate CLK lengths is formed by the following CLK periods $T_{Alu}^{Single-cycle} = 3ns$, $T_{Alu}^{Multi-cycle_1} = 1ns$, $T_{Alu}^{Multi-cycle_2} = 2ns$, $T_{Alu-Alu}^{Chained} = 6ns$. In this example, the set $Div(i) = \{1, 3, 6\}$ are the integral divisors of $T_{Alu-Alu}^{Chained} = 6ns$. Thus, the $T_{Alu}^{Multi-cycle_2} = 2ns$ has been derived from the $\lceil \frac{T_{Alu-Alu}^{Chained}}{Div(2)} \rceil = \lceil \frac{6}{3} \rceil = 2ns$. The CLK resolution can use different time-unit than ns i.e. ps, thus delivering a larger set of integer integral divisors. In each case, the lower acceptable register-to-register delay of each technology library i.e. 1000 ps for TSMC 0.13 um, has to be respected.

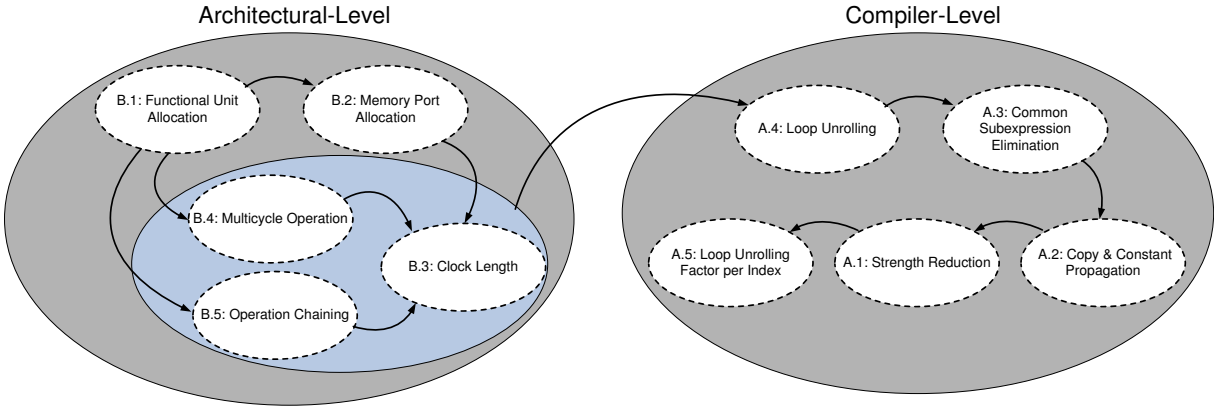


Figure 3.5.: Proposed DT traversing order.

3.5.3. Decision Ordering

The search space to be explored includes all possible combinations of the DT parameters (DT leaves). Thus, the number of explored solutions is formed through a full factorial of the instantiated design space. This type of search space enables the evaluation of the effects of each parameter on the examined solution.

Decision ordering defines the order that each solution included in the search space is evaluated. In [108], semantically non-valid or uninterest solutions are early identified (prior actual evaluation) and pruned through proper decision ordering. However, they evaluate dynamic memory management decisions rather than decisions concerning datapath synthesis, thus their design space is completely different from the one define here. Unfortunately, CompInLoop search space includes only valid solutions, making early pruning a risky approach. In context of CompInLoop exploration, decision ordering enables the definition of bounding rules and the development of efficient heuristics to be used during exploration's execution rather than early solution pruning.

Fig. 3.5 depicts the proposed decision order for CompInLoop Delay-Area exploration during HLS. Decision ordering suggests the order in which the exploration loops are structured in the CompInLoop DSE tool. We are interest on the trade-offs generated from the various architectural templates when functional equivalent code structures of the initial behavioral description are mapped onto them. Thus, architectural-level decisions are evaluated prior to the compiler-level ones.

Functional unit allocation scenarios (B.1 DT) are explored at a first-level followed by decisions on the memory interface structure (B.2 DT). forming the inner exploration loops. This set of decisions designate operation/instruction level parallelism, ILP, of each instantiated architecture, which depends on both the allocated FUs

and the supported data parallelism. After characterizing each architectural instance in terms of its supported ILP, we further explore them, considering the impact of different CLK lengths (B.3 DT). CLK length selection affects both the delay and the area of the explored solutions, since this exploration loop inherently explores also the impact of multi-cycling (B.4 DT) and operation chaining (B.5 DT). Even if the supported ILP is defined in the inner exploration loops, multi-cycling and operation chaining leads to differing scheduling and register allocation solutions.

After completing architectural-level characterization, we further explore at the compiler-level the trade-offs of altering the code structure of the initial behavioral description. Decisions concerning the invocation of loop unrolling (A.4 DT) are considered first. For a behavioral description including nested loops, one exploration loop per LUF is formed in the CompInLoop DSE tool. The LUFs' exploration loops follows the loop nesting found into the original source code of the behavioral description. Each combination of the LUFs generates a different behavioral instance (code structure) of the initial description. For each considered behavioral instance, code optimization decisions (A.3, A.2 and A.1 DTs) are applied. The ordering A.3→A.2→A.1 is proposed taking into account the opportunities that each optimization generates for the following ones [97].

3.6. Compiler-in-the-Loop Exploration Methodology

In this section, we describe the CompInLoop DSE methodology. We assume that the set of exploration parameters and their values are provided by the designer. Thus, the overall search space is defined by the full factorial of the exploration parameters. The proposed DSE is applied onto the ordered search space resulted from the DT ordering (Section 3.5.3). According to this ordering, we developed a pruning strategy based on a set of upper bounding conditions and a look-ahead heuristic algorithm to manage the increased size of the search space.

Fig. 3.6 summarizes the procedure of the our CompInLoop-DSE framework. The designer provides as inputs: (i) the behavioral description in C code, (ii) the set of exploration parameters along with their minimum and maximum values and (iii) the $Depth_k$ and a^T parameters which guide the look-ahead heuristic pruning. The output of the exploration procedure is an approximate Delay-Area Pareto curve.

3.6.1. Definition of Upper Bounding Conditions

The proposed CompInLoop DSE consists of several exploration loops (lines 7-9, 16, 20-22) formed according to the proposed DT ordering. The outer loops encounter

for the exploration of loop unrolling parameters (A.4 and A.5 DTs). After the LUF values are assigned, the Control-Data-Flow-Graph (CDFG) of the behavioral description is formed (line 11). CDFG alters only when different LUF values are assigned. Thus, the CDFG is generated once for each LUF assignment and it is reused in the subsequent exploration loops. The compiler-level source code optimizations (A.3, A.2 A.1 DTs) are applied to each generated CDFG following the proposed DT ordering (lines 12-14).

The following exploration loops, in Fig. 3.6, consider the CLK length selection, and generate the various resource allocation scenarios regarding the #MemPorts, #Muls, #Alus exploration parameters. The number of examined resource allocation scenarios is a critical factor of the overall exploration's runtime. The more the examined allocation scenarios, the larger the runtime of the exploration procedure. We constraint CompInLoop exploration to search only design solutions that are able to deliver delay gains, by defining the upper bound of resource allocation for each generated CDFG. The upper bounding conditions on resource allocation are extracted based on the following Lemmas:

Lemma 3.6.1.1 ***Given the CLK length and the LUF assignment of a CDFG, the resources allocated from the As-Soon-As-Possible (ASAP) schedule of the CDFG,*** $C_{ASAP} = \{Delay_{ASAP}, Res_{ASAP}\}$***, where*** $Res_{ASAP} = \{\#Alus_{ASAP}, \#Mul_{ASAP}, \#MemPorts_{ASAP}\}$***, forms the minimum delay solution.***

The proof of this lemma is straightforward, since it is impossible to get a solution faster than the ASAP solution, for a specific CDFG and given the CLK period.

Lemma 3.6.1.2 ***Given the CLK length, the LUF assignment of a CDFG and an optimizing scheduling algorithm, any resource allocation*** $Res_i = \{\#Alus_i, \#Mul_i, \#MemPorts_i\}$***, with*** $(\#Alus_i \geq \#Alus_{ASAP}) \bigwedge (\#Muls_i \geq \#Muls_{ASAP}) \bigwedge (\#MemPorts_i \geq \#MemPorts_{ASAP})$***, is not a dominant design solution in respect to the*** C_{ASAP}***.***

Let us assume that given the set of the CLK length, the LUF assignment of the CDFG and the scheduling algorithm, there exists a design solution $C_i = \{Delay_i, Res_i\}$ **with allocation** $(\#Alus_i \geq \#Alus_{ASAP}) \bigwedge (\#Muls_i \geq \#Muls_{ASAP}) \bigwedge (\#MemPorts_i \geq \#MemPorts_{ASAP})$**, that dominates** C_{ASAP}**. Dominance imposes that either** $Delay_i < Delay_{ASAP}$ **or** $Area_i < Area_{ASAP}$**. The assumption that** $Delay_i < Delay_{ASAP}$ **leads to a contradiction, since from Lemma 3.6.1.1** $Delay_{ASAP}$ **forms the minimum delay solution. In addition, the assumption that** $Area_i < Area_{ASAP}$ **leads also to a contradiction. That is because, the scheduling algorithm will schedule ASAP the CDFG since there are available resources to do so. Thus, any further resources found in** C_i **will be redundant and add some area,** $\Delta Area > 0$**, to the** $Area_{ASAP}$**. Thus,** $Area_i = Area_{ASAP} + \Delta Area \Leftrightarrow Area_i > Area_{ASAP}$**.**

Lemma 3.6.1.2 defines the upper bounding conditions in respect to the examined resource allocation scenarios. It implies that there is no need to examine design solutions with allocated resources greater than the allocated resources of the C_{ASAP} solution. Since multi-cycling and operation chaining is controlled through the CLK length selection, the ASAP schedule of each CDFG is performed for each explored CLK length (line 18 in Fig. 3.6), deriving the ASAP resource allocation and the minimum delay solution. The C_{ASAP} solution is used in the inner exploration loops as the upper boundary design point for the specific CDFG. Each solution exceeding the Res_{ASAP} configuration, is not considered as Pareto-optimal candidate and avoids examination (lines 23-26).

In the general case that a resource allocation scenario has to be examined, the CDFG is scheduled according to the specified CLK length (line 28). We consider a minimum delay resource constrained operation scheduling algorithm found in [97]. Each examined resource allocation (#Alus, #Muls, #MemPorts) forms the actual resource constraints that the scheduler has to satisfy. From the scheduled CDFG, we extract the number of control steps (line 29) and the number of allocated registers (line 30). The area cost of the examined solution is estimated (line 31) according to the area model presented in Section 3.4.

3.6.2. Gradient-Based Heuristic Pruning

The ordering of exploration loops in CompInLoop-DSE decomposes the search space into a set partial exploration curves. For each parameter assignment regarding the #Muls, #MemPorts, CLK, $\mathbf{LUF}_i$, ..., $\mathbf{LUF}_N$ parameters, there exist a partial exploration curve $\mathbf{L}^{\mathbf{x}}_{Alu}$ depending on the number of Alu components, where x is the vector representing the assignment of the remaining exploration parameters. For each $\mathbf{L}^{\mathbf{x}}_{Alu}$ of the search space, the following Lemma holds:

Lemma 3.6.2.1 ***Every L^{x}_{Alu} curve exhibits a saturation point (same delay for increasing #Alus) above which the examined point are not a dominant design solution in respect to the specific L^{x}_{Alu}.***

For any behavioral description, the parameter assignment vector x defines the corresponding CDFG under the CLK length #MemPorts and #Mul resource constraints. The ASAP scheduling of the CDFG with CLK length returns the $\#Alus_{ASAP}$ allocation. For each design solution included in $\mathbf{L}^{\mathbf{x}}_{\#Alu}$ with #Alus greater than the $\#Alus_{ASAP}$, the delay will be the same as in the ASAP solution, since there is no further available parallelism to be exploited. The non-dominance for these design solutions is proved in the same way as in Lemma 6.0.2.

Lemma 3.6.2.1 provides a key insight for the exploration of Alu's allocation scenarios. It says that given the CDFG, the CLK length and the MemPort and Mul

```
 1: input:    kernel.c: Behavioral kernel description in C;
 2: input:    Range of interest for: {#ALUs, #MULs, #Mem_Ports, #CLK, #LUF_i, ...};
 3: input:    Depth_k : Depth of look-ahead gradient searching;
 4: input:    a^T : Parameter controlling solution pruning;
 5: output:   Pareto Exploration_Curve;
 6:
 7: for(LUF_n= min_LUF_n ; LUF_n < max_LUF_n ; LUF_n++) {
 8:  …
 9:   for(LUF_i= min_LUF_i ; LUF_i < max_LUF_i ; LUF_i++) {
10:
11:     CDFG[LUF_i, …, LUF_n] ← C2CDFG(kernel.c, LUF_i, …, LUF_n);
12:     doCommonSubexpressionElimination(CDFG[LUF_i, …, LUF_n]);
13:     doCopyPropagation(CDFG[LUF_i, …, LUF_n]);
14:     doStrengthReduction(CDFG[LUF_i, …, LUF_n]);
15:
16:     for(CLK = min_CLK ; CLK < max_CLK ; CLK++) {
17:
18:      {ASAP_Alus, ASAP_Muls, ASAP_MemPorts, ASAP_Delay} ← ASAP (CDFG[LUF_i, …, LUF_n], CLK);
19:
20:        for(MemPorts_no = min_MemPorts ; MemPorts_no < max_MemPorts ; MemPorts_no++) {
21:          for(MUL_no = min_MULs ; MUL_no < max_MULs ; MUL_no++) {
22:            for(ALU_no = min_ALUs ; ALU_no < max_ALUs ; ALU_no++) {
23:             /* Checking Exploration's Upper Bounds */
24:             if (ALU_no > ASAP_Alus) && (MUL_no > ASAP_Muls) && (MemPorts_no > ASAP_MemPorts){
25:               break;
26:              }
27:
28:             sched_CDFG ← Schedule (CDFG[LUF_i, …, LUF_n], CLK, MemPorts_no, MUL_no, ALU_no);
29:             Contol_Steps ← Extract_Control_Steps (sched_CDFG);
30:             Registers ← RegAlloc(sched_CDFG);
31:             Area_Cost ← Area_Estimation (ALU_no, MUL_no, MemPorts_no, Registers, Control_Steps);
32:
33:             /* Look-ahead for Zero-Gradient Pruning */
34:             Grad← Evaluate_Gradient ((Area_Cost, Control_Steps), Depth_k);
35:             switch Grad
36:                   case  (Grad < 0) :  Insert (Examined_Curve, xi);
37:                   case  (Grad ≥ 0) :  if (MUL_no != MUL_no - 1){
38:                                        MUL_no ← min_MULs;
39:                                        ALU_no ← min_ALUs;
40:                                        MemPorts_no ← MemPorts_no + 1;
41:                                        break;
42:                                       }
43:                                       else{
44:                                        MUL_no ← MUL_no + 1;
45:                                        ALU_no ← (1 - a^T) × ALU_no; /* Pruning Control */
46:                                        break;
47:                                       }
48:     }
49:    }
50:   }
51:  }
52: …
53: }
54: Exploration_Curve ← Pareto_Extraction(Examined_Curve);
```

Figure 3.6.: Algorithm of CompInLoop exploration framework.

allocations, the $\mathbf{L}^{\mathbf{x}}_{Alu}$ curve presents zero-gradient (or non-decreasing gradient to cover the heuristic nature of the scheduling algorithm) for Alu allocations exhibiting larger Alu count than the Alu_{ASAP}.

Gradient-based heuristic is based on the above observations to effectively prune the number of examined solutions. Specifically, whenever a zero-gradient segment is identified in the examined $\mathbf{L}^{\mathbf{x}}_{Alu}$, the proposed heuristic stops the Alu exploration of the specific $\mathbf{L}^{\mathbf{x}}_{Alu}$ and moves the exploration to the $\mathbf{L}^{\mathbf{x'}}_{Alu}$, where x' follows the exploration loops in Fig. 3.6.

In reality, zero-gradient segments in a specific $\mathbf{L}^{\mathbf{x}}_{Alu}$ may occur prior to the Alu_{ASAP} configuration. For example, when examining the #Alu and #Alu+1 configurations in a $\mathbf{L}^{\mathbf{x}}_{Alu}$ a zero-gradient may be detected, however when #Alu+2 is examined then a decreasing gradient may be detected. This feature implies the heuristic nature of the gradient-based pruning. These zero-gradient segments are local and have to be de-characterized as proper design points for moving to the $\mathbf{L}^{\mathbf{x'}}_{Alu}$. Thus, examining zero-gradient only between two successive design solutions may lead to the loss of Pareto solutions. In order to avoid such situations, it is preferable that zero-gradient segments to be evaluated for a $Depth > 2$ of successive design solutions. The user defined $Depth_k$ parameter in the CompInLoop DSE (line 3 in Fig. 3.6) defines the depth that the gradient-based heuristic has to look-ahead in order to evaluate if the zero-gradient segment is local or not. Thus, for $Depth_k = k$ the condition that verifies a zero-gradient segment in a $\mathbf{L}^{\mathbf{x}}_{Alu}$ curve is formed as:

$$(\frac{\Delta Delay}{\Delta Area}|^{Alu_{i+1}}_{Alu_i} = 0) \bigwedge ... \bigwedge (\frac{\Delta Delay}{\Delta Area}|^{Alu_{i+k}}_{Alu_i} = 0) \tag{3.12}$$

High values of the $Depth_k$ parameter guarantee higher finding ratios of Pareto solutions in the expense of increasing exploration runtime. On the other hand, small $Depth_k$ values improve the overall exploration run-time in the expense of degrading the accuracy of Pareto solutions.

Each examined solution that the look-ahead evaluation reports a decreasing gradient, is characterized as a promising one and it is inserted in the examined set (line 36 in Fig. 3.6) in order to be further analyzed during Pareto curve computation (line 54).

The proposed heuristic takes special care to realize the moving from $\mathbf{L}^{\mathbf{x}}_{Alu}$ to $\mathbf{L}^{\mathbf{x'}}_{Alu}$, in case of a zero-gradient segment of $Depth_k$ is identified (lines 37-47). When such a pivot move takes place, the exploration in the $\mathbf{L}^{\mathbf{x'}}_{Alu}$ can start from various positions in respect to the number of Alus. The most conservative approach is to start the exploration of each $\mathbf{L}^{\mathbf{x'}}_{Alu}$ from the min_Alus position. The possibility to exclude Pareto solution from the examined set is minimized in the expense of higher exploration run-time due to large set of examined configurations. Also, empirical observations show that the Pareto solutions at the lower area regions are formed from solutions with small number of Muls components. Thus, searching for area

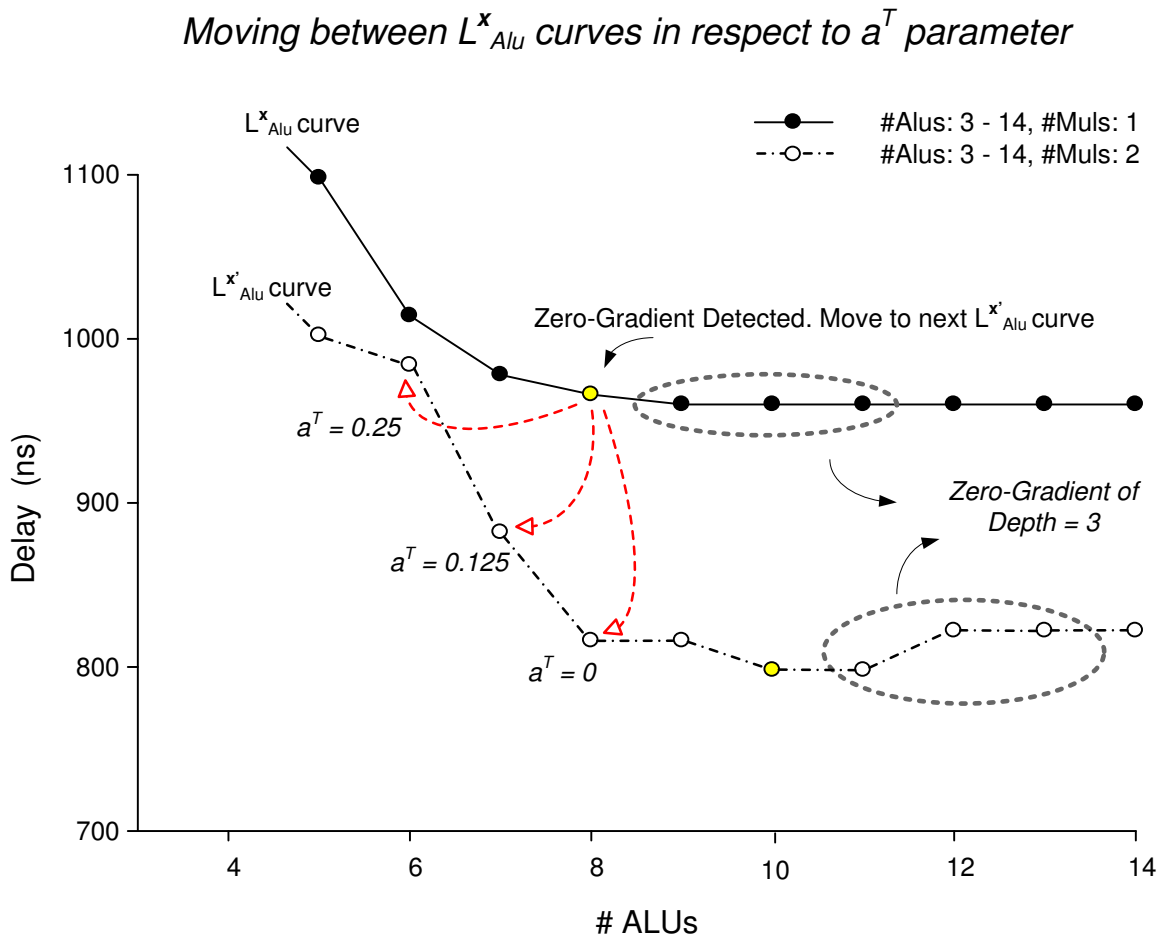


Figure 3.7.: Impact of a^T parameter during gradient-based pruning heuristic.

efficient trade-offs in $\mathbf{L}^{\mathbf{x}}_{Alu}$ curves with large number of #Muls may lead to evaluate many non-Pareto solutions. On the other side, the most aggressive approach is to start the exploration of each $\mathbf{L}^{\mathbf{x'}}_{Alu}$ from the last explored #Alus position. By this way, larger portions of the design space are excluded from examination under the risk of loosing Pareto solutions.

We introduce the a^T user parameter to guide a more balanced moving from $\mathbf{L}^{\mathbf{x}}_{Alu}$ to $\mathbf{L}^{\mathbf{x'}}_{Alu}$. The a^T parameter is a percentage defining the starting position of the $\mathbf{L}^{\mathbf{x'}}_{Alu}$ curve exploration in respect to the last explored #Alus position in $\mathbf{L}^{\mathbf{x}}_{Alu}$ (line 45). Fig. 3.7 depicts the impact of a^T parameter during the $\mathbf{L}^{\mathbf{x}}_{Alu}$ to $\mathbf{L}^{\mathbf{x'}}_{Alu}$ curve moving, when a $Depth_k = 3$ parameter is assumed for the 1D DCT kernel. The $\mathbf{L}^{\mathbf{x}}_{Alu}$ curve is examined until #Alus = 8, where a zero-gradient of depth 3 is detected. The $\mathbf{L}^{\mathbf{x'}}_{Alu}$ is formed by incrementing the #Muls exploration parameter. Varying the value of the a^T parameter, the exploration of $\mathbf{L}^{\mathbf{x'}}_{Alu}$ starts from different #Alus positions. The larger the a^T the larger the set of examined solutions. The exploration of $\mathbf{L}^{\mathbf{x'}}_{Alu}$ stops when a segment of non-increasing gradient is detected.

In case of the L^{x}_{Alu} is the last curve before incrementing the #MemPorts, the moving to the $L^{x'}_{Alu}$ is realized without considering the value of a^T parameter (lines 37-42 in Fig. 3.6). We follow the conservative approach in this case, starting the exploration of $L^{x'}_{Alu}$ from the min_Alus position, in order to avoid the exclusion of promising solutions from the examined set.

3.7. Experimental Results

In this section, we experimentally evaluate the proposed CompInLoop exploration approach. The analysis of experimental data is organized in three subsections. We present our experimental setup in Subsection 3.7.1. In Subsection 3.7.2, we discuss the quality assessment of design solutions derived from the CompInLoop exploration framework. Finally, Subsection 3.7.3 validates the sensitivity of the design space in respect to $Depth$ and a^T parameters of the CompInLoop exploration algorithm.

3.7.1. Experimental Setup

In order to implement the proposed CompInLoop exploration methodology, an automated HLS exploration framework has been developed. The exploration framework has been built utilizing the HLS libraries of the academic SPARK C-to-RTL tool [110]. SPARK has been extended with mechanisms such (i) the computation of the resources' upper bounds of each CDFG under various CLK lengths, (ii) the decision making procedure performed during the gradient-based solution pruning, (iii) the iterative scheduling under various resource, CLK and LUF parameters/constraints generated from the decision making engine, (iv) the extraction of proper statistics (area, delay, cycles etc.) according to the architectural template (Section 3.4) and the targeted technology library and (v) the Pareto set derivation of the explored solutions. The designer provides to the CompInLoop exploration framework the behavioral C code description of the application, along with the range of interest of the design parameters of Section 3.5. After exploration"s completion, the Pareto design solutions (solution presenting the best non-dominated trade-offs) are delivered to the designer. Through the SPARK's RTL code generator, a synthesizable RTL description of each of the Pareto solutions can be delivered to the designer for logic and physical synthesis using industrial tools. The CompInLoop exploration framework was installed and run on a Linux Xeon server at 2.33 GHz with 4 GB RAM.

A representative set of benchmarks found into real-life DSP and multimedia applications has been build to form the testing cases for evaluating the proposed exploration methodology. The benchmark suite consists of 10 computationally intensive kernels of various sizes and complexities. Specifically, we considered (i)

Table 3.2.: Kernel Characterization and Range of Exploration Parameters.

Kernel	**{#BBs, #Ops}**	**#ALUs**	**#Muls**	**CLKs**	**#MemPorts**	**LUFs**	**Sol. Space**
FIR16	{11, 19}	$\in$ {1, ..., 24}	$\in$ {0, ..., 12}	{1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 15}	**179712**
LMS	{11, 34}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 15}	**239616**
JPEG 1D DCT	{6, 87}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 7}	**119808**
YUB2RGBA	{5, 39}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 7}	**119808**
MatMul	{5, 45}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 3}	**59904**
SQRT	{14, 29}	$\in$ {1, ..., 24}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 2}	$LUF_1 \in$ {0, ..., 10}	**37440**
Jpeg 2D DCT	{10, 190}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 7} $LUF_2 \in$ {0, ..., 7}	**958464**
Mpeg 2D IDCT	{24, 367}	$\in$ {1, ..., 20}	$\in$ {0, ..., 8}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 3} $LUF_2 \in$ {0, ..., 2}	**77760**
Gauss Blur 2D	{30, 83}	$\in$ {1, .., 24}	$\in$ {0, ..., 12}	$\in$ {1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 4}	$LUF_1 \in$ {0, ..., 7} $LUF_2 \in$ {0, ..., 7}	**479232**
SOBEL 2D	{23, 98}	$\in$ {1, ..., 32}	$\in$ {0, ..., 12}	{1, 2, 3, 4, 6, 8}	$\in$ {1, ..., 6}	$LUF_1 \in$ {0, ..., 7} $LUF_2 \in$ {0, ..., 3}	**479232**

a custom 16th-order FIR filter (FIR16), (ii) an adaptive Least-Mean-Square filter (LMS) from [111], (iii) the row-wise Discrete Cosine Transformation of JPEG (JPEG 1D DCT) [100], (iv) the YUB to RGBA filter (YUB2RGBA) [112], (v) the MESA 4×4 Matrix Multiplication algorithm [100], (vi) the Square-Root function (SQRT) based on a $12th$ order Taylor approximation [111], (vii) the 2D DCT transformation of JPEG (JPEG 2D DCT) [100], (viii) 2D Inverse DCT kernel of MPEG application (Mpeg IDCT) [113], (ix) the Gauss Blur image transformation (Gauss Blur 2d) found into Cavity Detector algorithms, (x) the 8×8 Sobel edge detection image filter (2D SOBEL). The complexity of each kernel in terms of number of operations and basic blocks is shown in the second column of Table 3.2. Columns 3-7 summarizes the ranges of the exploration parameters considered in each case, while the last column of Table 3.2 reports the size of the overall search space in respect to the full factorial of the exploration parameters.

3.7.2. Quality Improvements on Design Solutions

Since the goal of the CompInLoop exploration methodology is to move towards higher quality design solutions, this subsection studies the quality improvements delivered by the proposed exploration framework in comparison with alternative exploration schemes. We implemented five different DSE strategies, regarding the categorization presented in Section 3.2. For all the implementations, the scheduling libraries of SPARK HLS tool has been used to guarantee judicious comparisons. Specifically, we considered:

1. Compiler assisted DSE (CompAssisted): Exhaustive exploration of design solutions by iteratively evaluating various resource allocations under standard code-level optimizations.

2. Compiler assisted with CLK selection DSE (CompAssistedCLK): Exhaustive exploration of design solutions by iteratively evaluating various resource allocations and CLK assignments under standard code-level optimizations.

3. Compiler-directed DSE (CompDirected): Exhaustive exploration of design solutions evaluating in a separated manner compiler-level decisions followed by resource allocation exploration.

4. Compiler-in-Loop DSE (CompinLoop): The proposed CompInLoop exploration strategy.

5. Exhaustive Compiler-in-Loop DSE (ExCompinLoop): Exhaustive exploration of the full factorial of the decisions found in the CompInLoop design space. ExCompinLoop would be used as the baseline to evaluate both the quality of the Pareto solutions delivered by the former DSE approaches and the speedups obtained by the proposed CompInLoop DSE.

The first set of experiments concerns the effectiveness of proposed approach to generate higher quality design solutions in comparison to the existing DSE strategies. All the kernels of the benchmark suite (Table 3.2) were explored according to each of the previous DSE strategies. Each DSE strategy returns a 2D Delay-Area Pareto diagram including the design solutions which form the optimal trade-off points. The y-axis represents the delay associated with each scheduled Pareto solution. The x-axis represents the area cost (um^2), estimated according to the architectural template presented in Section 3.4, of each solution after register allocation. Fig. 3.8 depicts the aggregated Delay-Area Pareto diagrams derived by the CompAssisted, CompAssistedCLK, CompDirected, CompInLoop for six kernels found in our benchmark suite. For the CompInLoop, we considered $Depth = 4$ and $a^T = 0.8$ of the examined kernels. The results from Fig. 3.8 shows that CompInLoop exploration discovers higher quality design points than the other strategies. We have to underline that we compare the exhaustive version of the other DSE strategies with the heuristic version of proposed CompInLoop methodology. Thus, CompInLoop is able to find more efficient than its competitors, despite of its heuristic nature. Specifically, the CompInLoop Pareto curves dominate in all cases the solutions discovered from the CompAssisted and CompDirected DSE strategies. The same behavior is observed also for the rest of the kernels not depicted in Fig. 3.8. This inform us that searching either small design spaces i.e. the CompAssisted DSE case or large design spaces i.e. CompDirected DSE case without taking into account the interactions of the whole set of exploration parameters (naive orthogonalization) leads to inefficient designs. In comparison with the CompAssistedCLK DSE strategy, the proposed CompInLoop approach also finds higher quality design solutions. CompAssistedCLK DSE seems to approximate CompInLoop Pareto curves in a better manner than CompAssisted and CompDirected strategies. Fig. 3.8 shows that in some cases CompAssistedCLK delivers a subset of CompInLoop Pareto solutions. In two test cases (LMS and Gauss Blur 2D), CompAssistedCLK also produces some Pareto solutions that dominate a small number of CompInLoop designs. However, these "good" approximations of CompAssistedCLK are limited only to solution subspaces leveraging low area costs. Such solution subspaces are characterized by reduced operation-level parallelism (proportional to resource count), thus the CLK length selection is the dominant performance factor. We can safely infer that compiler-level optimizations and especially loop-unrolling has a greater impact when the underlying architecture supports high degrees of operation level parallelism. In architectures supporting low degrees of operation level parallelism, the selection of fast CLK lengths should be considered.

Previous analysis indicated that CompInLoop exploration produces higher quality results than the DSE methods used so far. However, some fundamental questions are raised: *How much better quality solutions do we deliver? Does the proposed CompInLoop exploration converge or approximates the exact Pareto curve of the considered ComInLoop design space?* In order to address these questions, we compare the approximate Pareto curves generated from each of the aforementioned DSE approaches with the exact Pareto curve generated from the ExCompInLoop exploration. ExCompInLoop DSE generates the exact Pareto curve since it ex-

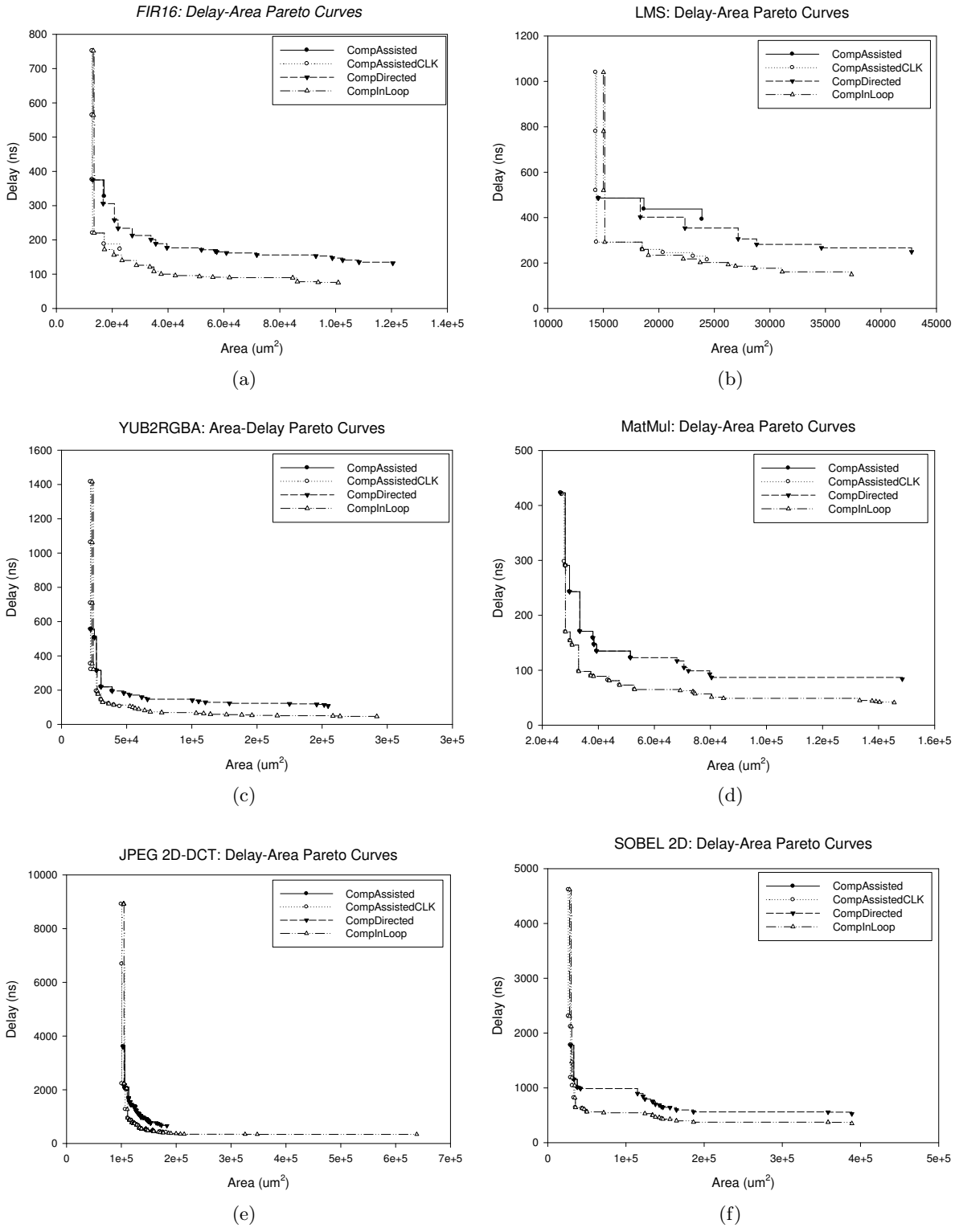


Figure 3.8.: Comparative Delay-Area Pareto curves delivered by differing DSE strategies.

plores exhaustively the all possible configurations of the search space. For each benchmark, the cardinality of the ExCompInLoop search space is given in the last column of Table 3.2. We considered two performance metrics used for measuring the accuracy the of Pareto sets generated from different multi-objective optimization algorithms, namely the Solution Coverage and the Generational Distance metric [114].

Solution Coverage (SL) is a cardinality-based metric which calculates the ratio of the non-dominated solutions found in each approximate Pareto curve in respect to the exact Pareto curve. Thus, the higher the coverage the better accuracy is achieved by the DSE. Table 3.3 reports the values of the SL metric for each benchmark kernel in respect to the adopted DSE strategy. The proposed CompInLoop exploration achieves the higher SL values both average and per benchmark in comparison the other DSE candidates. This indicates the importance of exploring the CompInLoop design space since most of the solutions that form the exact Pareto curve are included in the CompInLoop search space, with a limited number existing in the other search subspaces. Specifically, the average SL value for the CompInLoop exploration is 60.1% and average gains of 1667.9%, 41.4%, 355.3% are reported in comparison with the CompAssisted, CompAssistedCLK and CompDirected strategies, respectively.

However, the SL metric does not provide any insight on the distribution of the solutions and the relative quality of the solution set. For example, how far from the exact Pareto curve is laying the approximate Pareto one? For this reason, we employ the Generational Distance (GD) metric to evaluate our solution quality. The GD is an accuracy-based metric which calculates the average distance between the approximate and the exact Pareto curves. The lower the distance between the two curves, the better the approximation of the DSE strategy. The GD metric is computed according to the following formula:

$$GD(E,A) = \frac{1}{|E|}\left(\sum_{i\in E} d_i^q\right)^q \tag{3.13}$$

$$d_i = \min_{p\in A}\left\{\sqrt[2]{\sum_{k\in\{Area,Delay\}} (f_k(E_i) - f_k(A_i))^2}\right\} \tag{3.14}$$

where f_k represents the objective functions to be minimized (Area and Delay) and E, A is the exact (E) and approximate (A) Pareto curves, respectively. We set $q = 2$ to refer to Euclidean distance, since we are working in the 2D solution space. Table 3.4 summarizes the GD values computed for each examined DSE approach (Columns 2-5). Moving towards more complex design spaces (CompAssisted → CompInLoop) the accuracy of the DSE is improved. The last two columns of Table 3.4 reports in detail the gains delivered by CompInLoop exploration in respect to the CompAssistedCLK and CompDirected strategies. CompAssisted strategy delivers the worst approximation and is excluded for detailed comparisons. The proposed CompInLoop methodology delivers the lowest distance among all other

Table 3.3.: Comparison of Solution Coverage in respect to the Exact Pareto Curve

	Solution Coverage ExCompInLoop vs.			
Kernel	CompAssisted	CompAssistedCLK	CompDirected	CompinLoop
FIR16	0	20.0	0	70
LMS	0	25.0	0	31.3
JPEG 1D DCT	7.0	76.0	66	76
YUB2RGBA	0.0	42.8	0	71.4
MatMul	8.3	58.3	8.3	87.5
SQRT	14.3	87.5	14.3	28.6
JPEG 2D DCT	0.0	5.6	0.0	74.3
MPEG 2D IDCT	0.0	60.0	0	73.3
Gauss Blur 2D	0.0	2.4	0	14.8
SOBEL 2D	4.3	47.8	43.4	73.9
Average	**3.4**	**42.5**	**13.2**	**60.1**

Table 3.4.: Generational Distance Comparison in respect to the Exact Pareto Curve

	Generational Distance ExCompInLoop vs.				***Gains (%) CompInLoop vs.***	
Kernel	CompAssisted	CompAssistedCLK	CompDirected	CompinLoop	CompAssistedCLK	CompDirected
FIR16	36149.54	31800.82	5782.33	5378.66	83.1	6.9
LMS	3392.50	2912.98	758.20	792.31	72.8	-4.4
JPEG 1D DCT	6451.38	3725.14	2325.67	510.22	86.3	78.1
YUB2RGBA	55765.38	51702.07	6731.07	1146.59	97.7	82.9
MatMul	25158.36	23782.16	3125.63	158.83	99.3	94.9
SQRT	3404.98	163.27	1080.71	235.40	-44.2	78.2
JPEG 2D DCT	5660.83	3004.58	4483.48	384.18	87.2	91.4
MPEG 2D IDCT	10430.14	10145.25	9197.74	1008.29	90.1	89.1
Gauss Blur 2D	15301.15	9799.16	3538.14	1027.91	89.5	70.9
SOBEL 2D	77116.03	71372.29	2580.70	344.52	99.5	86.6
Average	23883.03	20840.77	3960.37	1098.69	**76.13**	**68.28**

DSE approaches, indicating that it approximates in an efficient manner the exact Pareto curve. Only for the LMS kernel, the CompDirected DSE approximates the exact Pareto curve 4.4% more effectively than the proposed approach. An average GD factor of 1089.69 is reported, leading to 76.13% and 68.28% average gains in comparison with the CompAssistedCLK and CompDirected strategies, respectively.

3.7.3. Exploration Sensitivity Over $Depth$ and a^T Parameters

The proposed gradient-based heuristic pruning algorithm used for CompInLoop exploration is controlled through the $Depth_k$ and a^T parameters, specified by the designer. Both parameters trades exploration runtime for accuracy of the approximated Pareto curve. In this subsection, we study the impact of the $Depth_k$ and a^T parameters on exploration's efficiency (approximation accuracy and the exploration runtime).

In order to evaluate the impact on the approximation accuracy, explorations performed under different $Depth_k$ and a^T assignments. For each generated approximate Pareto, we calculate the GD metric in respect to the exact Pareto (Eq. 13). Specifically, the parameters' values ranged between $Depth_k \in \{1, 2, 4, 6\}$ and $a^T \in \{20\%, 40\%, 60\%, 80\%, 100\%\}$. For each $Depth_k$, the corresponding GD values, varying the a^T parameter, are depicted in Fig. 3.9. As expected, the larger the values of $Depth_k$ and a^T parameters, the more accurate the approximation of the Pareto configurations (closer to the exact Pareto trade-offs). Large values of these parameters guides the pruning algorithm to search deeper and cover a larger portion of the solution space to decide the efficiency of each design solution. In addition, it is observed that the positive impact of the a^T parameter is greater for small values of the $Depth_k$ parameter. The a^T parameter operate as an amortization component during the pruning procedure. Thus, the more aggressive the pruning (i.e. small $Depth_k$), the higher the impact of the amortization. The same amortization effects occur for high $Depth_k$ in configuration with small a^T values. However, the amortization impact of the $Depth_k$ parameter seems to be smaller than the impact of a^T. For example, the amortization impact of $Depth_k$ on the the MatMul case is negligible. The structure of the MatMul's behavioral code presents many and large zero-gradient segments in its $\mathbf{L}^{\mathbf{x}}_{Alu}$ curves, located far away the Alu_{ASAP} configuration. Thus, exact Pareto solutions are early pruned from the evaluation. Further increment of the $Depth_k$ value will compensate this early solution pruning in the expense of increased exploration runtime. It is also noticeable that even for small values of the parameters i.e. $Depth_k = 2$ and $a^T = 20\%$, the proposed CompInLoop exploration methodology delivers smaller distances from the exact Pareto curve than the the CompAssisted, CompAssistedCLK and CompDirected DSE approaches. This indicates that the proposed DSE remains the most efficient exploration solution even in its worst case accuracy configurations.

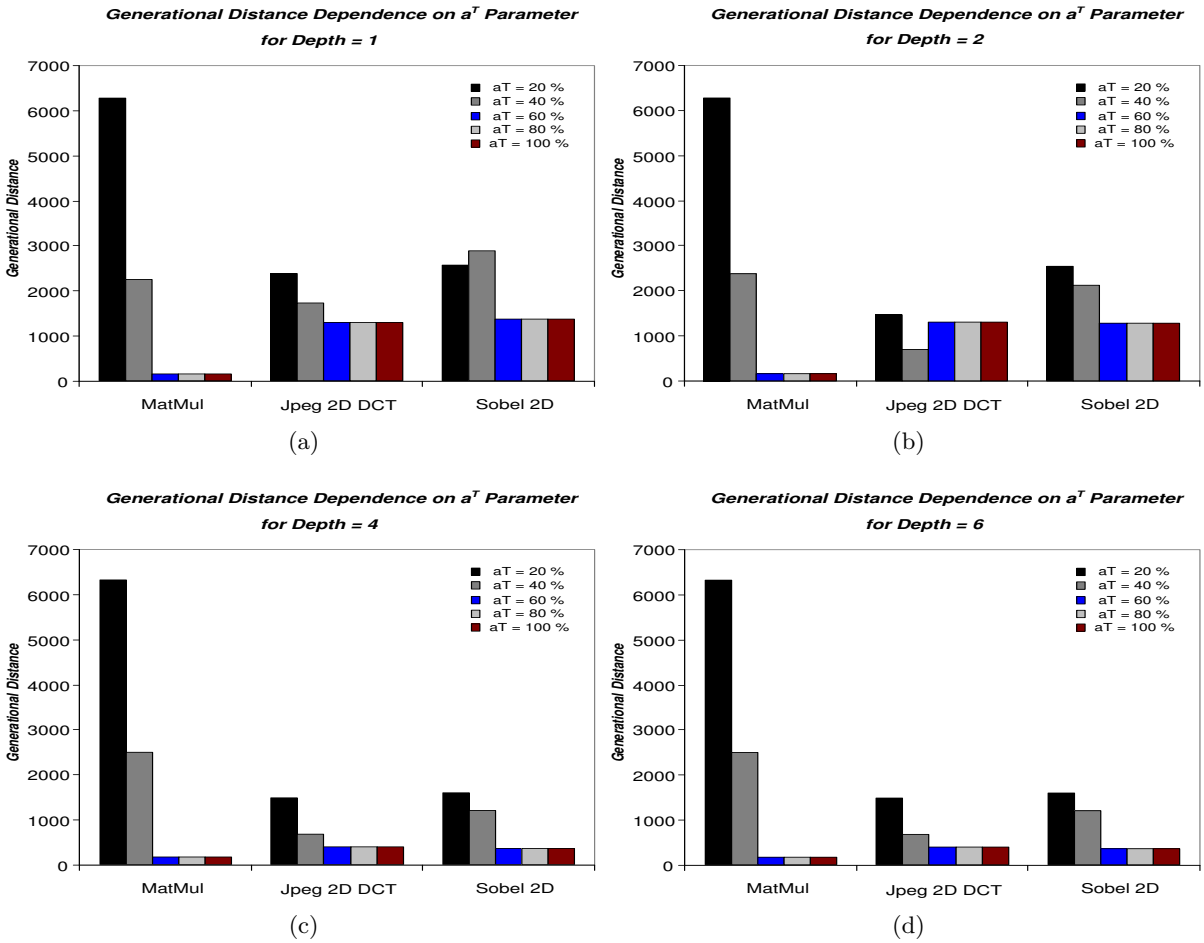


Figure 3.9.: Impact of the $Depth_k$ and a^T parameters on the exploration approximation efficiency in respect to the GD metric.

The final set of experiments concerns the evaluation of the exploration runtime. In order to eliminate both the dependence on the architecture of the host computer and the impact of other application workloads running concurrently with the exploration framework, we evaluate the runtime efficiency by measuring the percentage of solutions explored from the various DSE approaches. This does not mean that the delay of evaluating a single solution is the same across the overall search space. It is well known that there is a strong dependency of the operation scheduling on the size of each explored CDFG [44]. Fig. 3.10 depicts a comparative histogram on the number of design solutions explored by each DSE approach (CompAssisted, CompAssistedCLK, CompDirected, CompInLoop). We consider two configurations of the proposed CompInLoop DSE. The first configuration is customized for runtime efficiency with $Depth_k = 2$ and $a^T = 20\%$, while the second one is tuned for high accuracy with $Depth_k = 4$ and $a^T = 80\%$. The results are normalized for each kernel in respect to the solutions explored by the ExCompInLoop DSE (last column of Table 3.2). CompAssisted exploration reports the lowest runtime , exploring a very

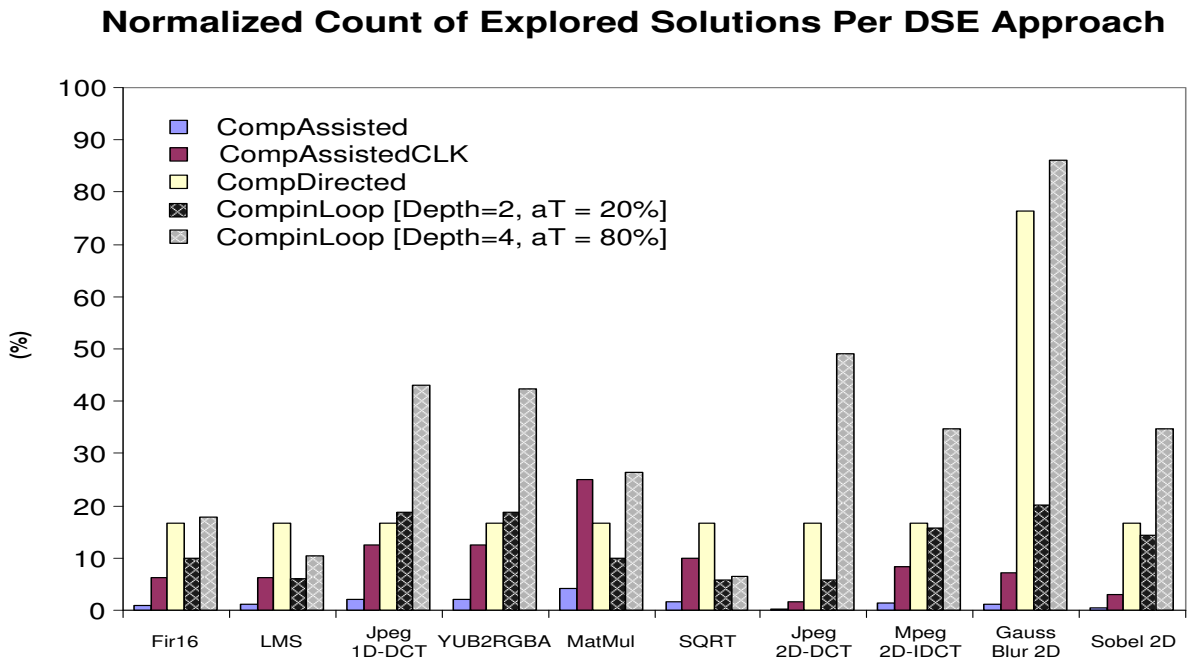


Figure 3.10.: Normalized percentage of the explored search space.

small portion of the solution space which does not include efficient design trade-offs as analyzed in Subsection 3.7.2. As expected, the accuracy-oriented CompInLoop configuration presents the highest exploration runtimes among the rest of DSE alternatives. In average, it explores the 35.1% of the solution space, 25.9% and 12.5% larger than the CompAssistedCLK and CompDirected, respectively. In case of the LMS, the MatMul and the SQRT kernels, it delivers exploration runtime close or even lower than the CompAssistedCLK and CompDirected approaches. The runtime-oriented CompInLoop configuration explores in average the 12.5% of the solution space. Specifically, it explores only 3.24% more design solutions in comparison to CompAssistedCLK. The proposed runtime-oriented CompInLoop is more efficient than the he CompDirected DSE, since it achieves to explore 10.1% faster the wider CompInLoop solution space. Moreover, it is also noticeable that the runtime-oriented CompInLoop ($Depth_k = 2$, $a^T = 20\%$) remains still more efficient in terms of Pareto curve accuracy than the CompAssistedCLK and CompDirected approaches, delivering GD gains of 57.8% and 17.4%, respectively. The fact that the proposed CompInLoop approach is able to deliver better Pareto design solutions than the existing DSE approaches with similar exploration runtimes, makes it an attractive and practical DSE alternative.

3.8. Conclusions

In this chapter, we proposed a Compiler-in-Loop exploration approach to move towards higher quality delay-area trade-offs during high-level synthesis. A parameterizable design space based on decision trees has been introduced to serve as a conceptual framework able to model the design decisions and reason about their interactions. To deal with the increased size of the solution space, a systematic exploration methodology has been developed based on upper bounding and effective solution pruning exploiting the structural features of the introduced design space. Through extensive experimentation, considering a representative set of real-life benchmarks, we showed that the proposed methodology outperforms the existing exploration approaches by both converging to better trade-off points, closer to the exact Pareto curve and maintaining reasonable runtime.

Part II.

Architectural Synthesis of Delay-Area Optimized Coarse-Grained Reconfigurable Coprocessors

Intelligence is the ability to adapt to change.

Stephen Hawking

Market demands urge embedded systems to incorporate multiple DSP applications from different domains, such as multimedia, telecom and wireless communications. DSP applications perform computationally intensive operations with lightweight control structures and spend most of their execution time in small code segments, called kernels. Many research activities from the HLS and the ASIP design research areas [115], [116], [7], [6], [48] have proposed hardware acceleration of these kernels. However, the allocation of multiple specialized hardware accelerators imposes high silicon area overheads.

Reconfigurable computing [31] has been proposed as a new paradigm which addresses the design requirements of hardware flexibility and high performance, found in modern multi-standard embedded systems. By binding together the configurability of software solutions along with the high performance of specialized hardware architectures, it closes the existing gap between flexibility and high performance.

A large number of fine- and coarse-grained reconfigurable architectures have been presented in [31], [32], respectively. The former architectures favor bit level operations and exhibit high power dissipation and limited performance gains. The latter ones operate at the word level of granularity exhibiting power and performance features comparable to ASIC implementations. The coarse-grained architectures can be further categorized in respect to their interconnection scheme in: (i) array-based [117], [118] and (ii) row-based [119], [120], [15], [49], [16]. Row-based reconfigurable architectures exhibit low area complexity, high hardware utilization and relatively small configuration words. In contrary, the aforementioned characteristics are not efficiently supported by the array-based architectures. Moreover, array-based architectures (also a limited number of the row-based i.e. [120], [49]) are based on pre-defined interconnection structures and register allocations which limit the customization of the flexible datapath to the application's domain needs [32]. On the other hand, row-based architectures [119], [16], [121], [122] are synthesized according to: (i) the resource constraints (i.e. the number of processing elements) for the flexible datapath and (ii) the set of kernels that are going to be mapped. Thus, the kernels' mapping and the final architecture instantiation are jointly optimized, delivering flexible datapaths tailored to the application's needs. We target to the DSP application domain and previous analysis indicates the reasons that guided our work to consider the design space of row-based coarse-grained reconfigurable architectures.

Thus, in this part, we present optimized coarse-grained reconfigurable datapath architectures and the corresponding high level synthesis methodologies for the design

of high performance and area efficient flexible coprocessors, targeting computationally intensive DSP kernels of embedded applications.

Our approach is shaped by two well known trends in VLSI design

- **The architecture efficiency gap i.e. how to build optimized hardware architectures and**

- **the productivity gap i.e. how to map efficiently high level behavioral descriptions on hardware architectures.**

Targeting the architecture efficiency gap, two new coarse-grained reconfigurable architectural templates are proposed: (i) one for coprocessors invoking array-based multiplication units and (ii) the second one for coprocessors invoking tree-based multiplication units. Targeting the productivity gap, new high-level synthesis algorithms and tools have been developed to automatically map behavioral descriptions in C onto the introduced reconfigurable architectural templates.

The proposed reconfigurable architectural templates enables high performance and low area DSP kernel implementations, thus being extremely promising solutions. The proposed solutions have been evaluated against a large set of state-of-the-art reconfigurable datapaths and through a large number of representative DSP kernels. Comparative results show the efficiency of the proposed reconfigurable solutions both in latency and area complexity.

The remainder of this part is organized as follows. Chapter 4 presents the flexibility inlining technique which inserts bit-level flexibility to array-based multiplication units and generates reconfigurable architectures exploiting horizontal/vertical parallelism and operation chaining architectural optimizations in a combined manner. Chapter 5 presents a reconfigurable datapath with flexible Carry-Save operation templates for arithmetically optimized operation chaining 5.

4. Architectural Synthesis for Coarse-Grained Reconfigurable Datapath With Bit-Level Inlined Flexibility

In this chapter, we introduce a circuit-level design methodology along with architectural synthesis techniques for high performance and area efficient coarse-grained reconfigurable datapaths, targeting computationally intensive DSP kernels of embedded applications. The circuit-level design methodology is analyzed in detail and an area-time efficient reconfigurable kernel architecture is presented. The proposed technique inlines flexibility into custom CS arithmetic datapaths exploiting a stable and canonical interconnection scheme. The canonical interconnection is revealed by a transformation, called Uniformity Transformation, imposed on the basic architectures of CS-multipliers and CS-chain-adders/subtractors. Based on the novel coarse-grained reconfigurable/flexible architectural template, we propose a methodology which enables the combined exploitation of the horizontal and vertical parallelism along with the operation chaining opportunities found in the application's behavioral description. Efficient synthesis techniques exploiting these architectural optimization concepts from a higher level of abstraction are presented and analyzed. Average latency and area reductions up to 33.9% and 53.9% are reported respectively and higher hardware area utilization, compared to previously published high performance coarse-grained reconfigurable datapaths. Also, experimental results including quantitative and qualitative comparisons with existing reconfigurable arithmetic cores and exploration results of the proposed reconfigurable architecture are provided.

4.1. Introduction

This chapter introduces operation level reconfigurability in arithmetic datapaths which incorporate a significant degree of computation density (i.e., array multipliers). Specifically, we present bit-level design techniques which enable operation level reconfigurability to be efficiently incorporated into custom arithmetic datapaths. The generated architecture is a coarse-grain reconfigurable datapath which mainly targets ASIC implementation technologies. It provides fast implementations for the set of mapped operations due to the CS-based logic, which eliminates the time consuming carry-propagation. The fast arithmetic operations and the stable inter-

connection scheme increase the opportunities for operation chaining which has been proved a beneficial technique for DSP algorithms [7].

At the circuit-level, we present the techniques which enable operation level reconfigurability to be efficiently incorporated into custom arithmetic datapaths. We apply these techniques in order to introduce an area, time and power efficient reconfigurable datapath architecture, considering the DSP application domain. The proposed techniques incorporate flexibility by mapping together the behaviors of a CS multiplier, a CS adder and a CS subtractor onto a stable interconnection scheme. At the micro-architectural level, we introduce a new synthesis methodology for the introduced reconfigurable architecture, for high performance and area efficient flexible datapaths. The area efficiency and the high utilization of the underlying hardware remove the inherent limitation of coarse-grained reconfigurable architectures, where large silicon area portions remains unutilized during execution. The area efficiency comes without performance degradation. This is particularly critical for modern embedded DSP applications, which impose tight latency constraints.

At the architecture-level, coarse-grained reconfigurable datapaths demand the close coupling between the applications' behavior and the structure of the underlying hardware. This requirement has made HLS techniques [44] an attractive solution for the design of coarse-grained reconfigurable datapaths. Many architectural HLS optimization concepts like operation level parallelism (horizontal parallelism), pipelined execution (vertical parallelism) or operation chaining, have been incorporated in the architecture specification of coarse-grained datapaths, especially for row-based ones. However, there is no any known approach in the literature that exploits the advantages of: (i) horizontal parallelism, (ii) vertical parallelism and (iii) operation chaining, into a single flexible architecture. Several coarse-grained architectural templates have been proposed (Section 4.2), which combine in an efficient manner up to two architectural HLS optimization concepts in their datapath specification. Consequently, there exists a gap in current literature considering the aforementioned architectural optimizations in a holistic way. We prove that designing flexible datapaths based on the combined exploitation of the three architectural optimization concepts, deliver solutions with higher performance and better area complexity than the existing approaches. We achieve this goal through the development of a specialized datapath architecture and novel design synthesis methodologies.

The main contributions of this chapter are:

- **The Flexibility Inlining technique is introduced. It is enabled through a transformation at the Register Transfer Level (RTL) of abstraction, named Uniformity Transformation. Uniformity Transformation reveals an interconnection scheme that remains stable and canonical between the configurations of CS-chained adders, CS-chained subtractors and CS-array multipliers.**

- **An optimized unified cell (UC) that combines the behaviors of multiplication, addition and subtraction, is presented.**

- **Based on the appliance of Flexibility Inlining, we present a novel coarse-grained reconfigurable architectural template and we analyze it in detail, at the circuit level, based on a specific architecture instantiation.**

- **Evaluation of the proposed architecture is conducted through extensive experimentation and explorative results.**

- **A new design strategy is proposed for realizing flexible datapaths, based for the first time on the combined exploitation of the architectural optimizations: (i) horizontal parallelism, (ii) vertical parallelism and (iii) operation chaining. A comprehensive HLS methodology is developed for synthesizing high performance and area efficient DSP kernels onto the introduced flexible datapath templates.**

At the circuit-level, the effectiveness of our approach is proven through a rich set of experimental data. Qualitative comparisons show that the proposed architecture is able to handle a large set of mapped arithmetic behaviors without wasting computational resources. Quantitative comparisons with dedicated and previously published reconfigurable architectures are also included. Specifically, the proposed architecture delivers gains up to 46.9% in area coverage, 296.5% in clock frequency, 25.2% in power dissipation and 32.75% in Mega Operations Per Second (MOPS) comparing to the reconfigurable Multiply-Accumulate (MAC) unit in [22]. In comparison to dedicated MACs, the proposed architecture delivers gains up to 218.6% in MOPS and 163.33% in energy efficiency (MOPS/mW). Finally, exploration results, altering the reconfigurability/flexibility incorporated into the proposed architecture, show an almost linear scaling behavior with regard to area complexity and power consumption. At the micro-architectural level, extensive experimentation based has been conducted based on a representative set of DSP benchmarks, showing that the proposed synthesis methodology achieves significant gains in terms of execution time and area complexity compared to previously published coarse-grained reconfigurable datapaths. Specifically, the comparison with the row-based coarse-grained reconfigurable datapath in [15] have shown an average improvement of 33.9% and 23.5% in latency and area, respectively. Compared to reconfigurable architectures with aggressive chaining opportunities [16], gains of 28.4% in execution time and 53.9% in area complexity are reported. Assuming reconfigurable datapaths with horizontal/vertical parallelism and data forwarding features [17], our methodology delivers an average improvement up to 70% and 30.9% in latency and area, respectively. The Area-Delay product and area utilization metrics further prove the efficiency of our approach. Finally, the scalability of the proposed solution has been evaluated showing its advantageous features considering the execution latency and bit-width scaling.

The remainder of this chapter is organized as follows. Section 4.2 discusses related work. In Section 4.3, we present the Flexibility Inlining design technique which enables the bit-level datapath merging. Circuit level analysis of the proposed reconfigurable datapath is presented in Section 4.4. Section 4.5 provides the micro-architectural abstractions of the proposed coarse-grained reconfigurable datapath and how they are exploited in an architectural synthesis framework developed for mapping behavioral kernels onto the proposed reconfigurable datapath. Section 4.6 experimentally evaluates the efficiency of the reconfigurable solution in comparison with state-of-the-art reconfigurable architectures.

4.2. Related Work

This section presents the related work on row-based coarse-grained reconfigurable architectures. We focus on the architectural optimizations concepts incorporated into their datapath.

Horizontal parallelism offers execution of multiple operations in parallel. Since it is a standard optimization for hardware accelerators, most of the reconfigurable architectures [119], [120], [15], [49], [16] support this feature.

Horizontal and vertical parallelism are combined in [119], [120]. In [119], a reconfigurable architecture is presented which maps the computation onto a 1-D linear pipeline of coarse-grained components i.e. ALUs, RAMs etc. Although, they consider pipelining at the inter-functional unit level, no support for pipelined components is considered, thus producing large time slacks between fast and slow functional units. PipeRench [120] reconfigurable architecture consists of identical stripes, organized in a pipelined manner. Each stripe consists of parallel processing elements enabling horizontal parallelism in an intra-stripe manner. In case that multiple stripes are allocated, PipeRench becomes an array of processing elements, which increases the area complexity of the final implementation.

A combination of operation chaining and horizontal parallelism is reported in [49], [16]. In [16], a flexible datapath architecture based on aggressive operation chaining of reconfigurable units is proposed. However, their approach introduces two main drawbacks. The operating frequency of the datapath is defined in all cases by the critical path of the slowest chained computational components, even if these slow components are not utilized in every control step. In addition, large area overheads with low utilization are imposed, since only half of the computational components are active per control step in every flexible cell. The same drawbacks stand also for the tile architecture in [49], with a smaller impact due to the less aggressive operation chaining capabilities.

The aforementioned coarse-grained reconfigurable architectures take into account

either one [15] or combinations of up to two [119], [120], [49], [16] architectural optimizations during datapath's specification. The main differentiator of our approach is that we encompass the whole set of architectural optimization in a single flexible architecture and we present a synthesis methodology which exploits these architectural features delivering high performance and area efficient datapath solutions. Recently, the NISC (No-Instruction-Set-Computer) technology [17] has been presented, trying also to apply the three architectural optimization concepts in the datapath's specification. However, we manage vertical parallelism and operation chaining at a finer degree of granularity.

4.3. The Flexibility Inlining Technique

This Section introduces the Flexibility Inlining technique. Flexibility Inlining maps efficiently together the RTL structures of a CS multiplier, a CS-adder and a CS-subtractor. It is enabled through a transformation at the RTL, named Uniformity Transformation. Uniformity Transformation reveals an interconnection scheme that remains stable and canonical between the configurations of CS-chained adders, CS-chained subtractors and CS-array multipliers.

4.3.1. The Flexibility Inlining Technique

Motivation and Problem Description

The functional generality of most of the reconfigurable architectures introduced, is performed by both interconnection's and functional block's reconfigurability. Classical FPGA structures use switch matrices for programmable routing, while coarse-grained reconfigurable arrays use mostly complex bus-based or highly multiplexer-based interconnections enabling the transfers between the configurable blocks. These complex interconnection schemes impose a significant area overhead and also a reconfiguration delay overhead due to the augmented number of configuration bits needed to control the interconnections.

On the other hand, from an algorithmic point of view, DSP applications are based mainly on four operative modes, namely, addition, subtraction, multiplication and shifting. This can be easily confirmed by profiling the main DSP kernels' dataflow-graphs. Thus, the ability of performing the previously mentioned primitive operations, by maintaining temporal flexibility and stable/simple interconnection structure, can expose high area and reconfiguration delay gains comparing with conventional coarse-grained architectures. Flexibility Inlining enables the mapping of the basic computational intensive operations (addition, subtraction, multiplication)

onto the structure of an array multiplier, while maintaining the efficient routing between the initial multiplier's basic cells.

Consider a $N \times N$ array multiplier based on CS-adders [123]. The array consists of N^2 multiplier cells (MCs) and their interconnections. The structure of such a multiplier is given in Fig. 4.1a for $N = 4$. Let i be the array's line index, and j the array's column index. The overall architecture can be described by the following relations:

$$MC_{i,j} : a_j \times b_i \times si_{i,j} \times ci_{i,j} \rightarrow so_{i,j}, co_{i,j} \tag{4.1a}$$
$$si_{i,j} = 0, \qquad i = 0 \tag{4.1b}$$
$$si_{i,j} = so_{i-1,j+1}, \qquad i \in \{1, N-1\} \tag{4.1c}$$
$$ci_{i,j} = 0, \qquad i = 0 \tag{4.1d}$$
$$ci_{i,j} = co_{i-1,j}, \qquad i \in \{1, N-1\} \tag{4.1e}$$
$$a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j} \in \{0, 1\} \tag{4.1f}$$

Relation 4.1a defines that each basic cell is a multi-input-multi-output function targeting the Boolean domain, as imposed by relation 4.1f. The order of the binary variables in the specification function 1a also defines the instantiated input/output port order at basic cell level. Relations 4.1b to 4.1f refer to the routing properties of the CS-multiplier's architecture. Actually, they determine the source of cell's inputs (s_i, c_i).

The array's structure of CS-adders for chain addition, has significant similarities with the multiplier's one. In both cases N^2 cells are required and the basic component of full addition (AC) is present. An illustrative example of 4×4 CS chain-adder is given in Fig. 4.1b. The corresponding relations that describe the chain-adder structure are the following ones:

$$AC_{i,j} : a_j \times si_{i,j} \times ci_{i,j} \rightarrow so_{i,j}, co_{i,j} \tag{4.2a}$$
$$si_{i,j} = x_{0,j}, \qquad i = 0 \tag{4.2b}$$
$$si_{i,j} = so_{i-1,j}, \qquad i \in \{1, N-1\} \tag{4.2c}$$
$$ci_{i,j} = y_{0,j}, \qquad i = 0 \tag{4.2d}$$
$$ci_{i,j} = co_{i-1,j-1}, \qquad i \in \{1, N-1\} \tag{4.2e}$$
$$a_j, x_{0,j}, y_{0,j}, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j} \in \{0, 1\} \tag{4.2f}$$

Relation 4.2a defines each basic cell's Boolean function, equations 4.2b to 4.2e refer to the routing properties of the structure and the ordered binary variables in the specification function 4.2a define the instantiated input/output port order at

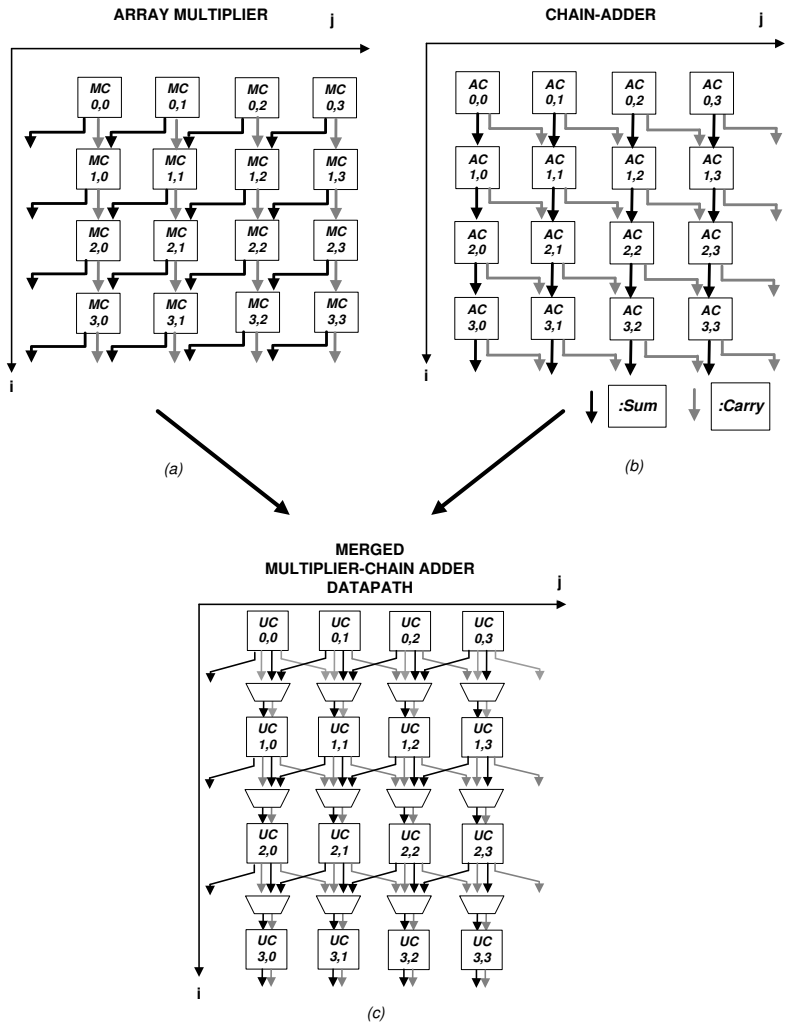


Figure 4.1.: a) 4×4 Multiplier, b) 4×4 Chain-Adder, c) Straightforward Unified Architecture.

basic cell level. The same relations also apply in chain-subtraction, with the only difference in the basic cell's truth table.

Comparing Fig. 4.1a with Fig. 4.1b, the routing differences between the two structures are exposed. The routing scheme, for both architectures, is denoted by relations 4.1c and 4.1e for the array-multiplier and by relations 4.2c and 4.2e for the chain-adder, respectively. Actually, these relations are depicting the dependence of a cell with the cells located on the previous row and the neighbor columns. Relation 1c states that the multiplier's input si of $MC_{i,j}$ results from the cell's output so of the previous row $(i-1)$ and the right column (j+1). Respectively, relation 4.1e denotes that input ci results from the cell's output co of the previous row $(i-1)$ and the same column (j). In the chain-adder's case, relations 4.2c and 4.2e state that $AC_{i,j}$ receives its input signal si from the cell's output so of the previous row $(i-1)$ and the same column (j), while the input signal ci is the cell's output co of the previous row $(i-1)$ and the left column $(j-1)$.

Finally, let us consider a unified cell (UC) (Fig. 4.1c) that is able to function as MC or as AC or as subtraction cell (SC). The function of the cell depends on a set of appropriate selection signals which control the operating mode of the unified cell. An optimized implementation of the UC is presented at Subsection 4.3.1. The number of inputs and outputs for the UC derived by the union of the basic cells' input ports. In our case, the port declaration of the unified cell is going to be the following:

$$\begin{aligned} Ports_{MC} \cup Ports_{AC} \cup Ports_{SC} & \\ &= \{a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &\quad \cup \{a_j, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &\quad \cup \{a_j, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \\ &= \{a_j, b_i, si_{i,j}, ci_{i,j}, so_{i,j}, co_{i,j}\} \end{aligned} \tag{4.3}$$

As stated previously, mapping together multiplication, addition, subtraction, chain addition and chain subtraction operations on a single circuit, and changing dynamically the configuration between these behaviors can lead to a high performance reconfigurable functional unit, especially for the DSP domain. However, the combinative mapping of the above configurations in a straightforward manner, as in Fig. 4.1c, suffers from the described routing dissimilarity, and from the functional dissimilarity occurring between each basic cell configuration context. Routing dissimilarity imposes a complex interconnection scheme between the cells. Actually, a 2 to 3 switch circuit is needed for every $UC_{i,j}$ to route appropriately the cell's outputs. This scheme is area inefficient, augments the architecture's configuration word-length and imposes a highly complex interconnection routing between the cells.

With Flexibility Inlining the aforementioned routing and functional dissimilarities

are tackled through (i) a transformative procedure which is named Uniformity Transformation, and (ii) a design procedure for optimized implementation of the UC.

Uniformity Transformation: Enabling Common Interconnections Among Arithmetic Configurations

Uniformity Transformation is applied on the overall architecture (primary inputs' bit-order, UC's input and output ports) and results to a stable routing scheme among the desired arithmetic behaviors. The transformation is completed in three stages shown in Fig. 4.2 and analyzed in the following paragraphs.

Uniformity Transformation should not be confused with the Uniformization methods presented in the field of systolic arrays [124], [125]. Uniformization is imposed on linear recurrence equations (inter-loop dependencies), which describe the system's behavior. It transforms linear into uniform recurrences in order to enable a direct mapping on a locally connected systolic array. The proposed Uniformity Transformation is imposed on the bit-level description of arithmetic datapaths in order to provide an efficient datapath to datapath mapping. However, in an abstract manner, the main goal of both approaches is the generation of constant structures (inter-loop dependencies in Uniformization, interconnection schemes in Uniformity Transformation) between descriptions with non "one-to-one" correspondence.

Step 1. Selection of the Reference Architecture: At this step, a reference architecture is selected. The candidate architectures are actually two, since the chain-adder and the chain-subtractor have the same interconnect structure. The routing scheme of the reference architecture will remain stable till the end of the process. The two routing architectures are equally canonical and enable efficient VLSI implementation. So, the final selection is driven by the utilization degree of each operation mode. For example, if one application requires a large number of multiplication operations, it is preferable to select the multiplication scheme as reference architecture. Equally, if the additions or subtractions occur more often, then it is more serviceable to select the chain-addition architecture. The utilization degree for each operation can be extracted by profiling the dataflow-graphs of the applications which will be mapped on the proposed reconfigurable arithmetic datapath. Thus, without loss of generality, the reference architecture for the undergoing analysis is considered the chain-addition's one.

Step 2. Generation Of The Mirror-Symmetrical Architecture: Given that the reference architecture has the routing scheme of a CS-chained-adder, in the subsequent stage of the transformation we reverse the bit-order of the multiplier's input $a_j, j \in [0, N-$

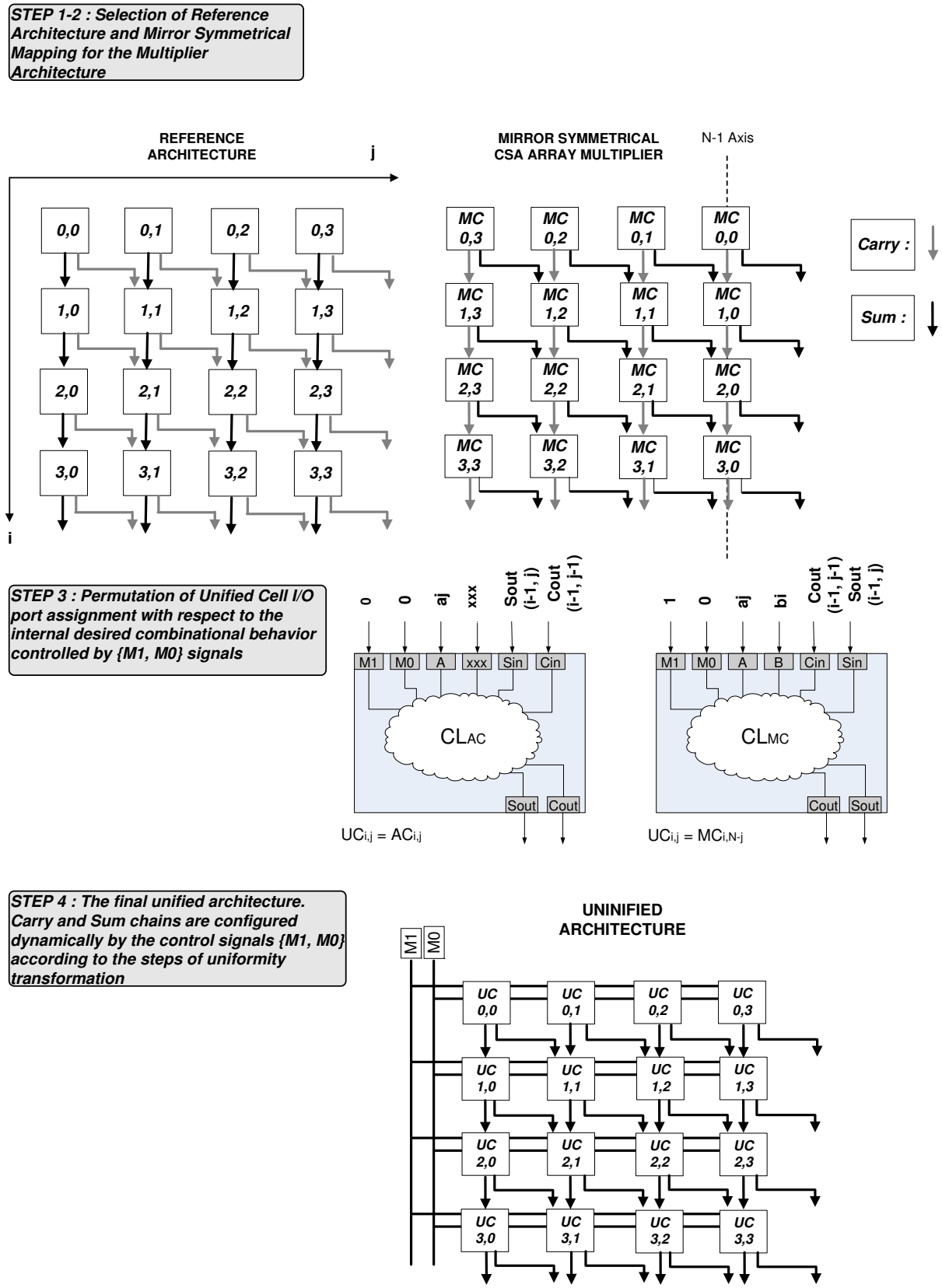


Figure 4.2.: The Uniformity Transformation.

1], and the mapping order of the basic cells to preserve the correct functionality (***Step 2*** in Fig. 4.2). The resultant architecture is mirror-symmetrical to the initial one, over the $j = N - 1$ vertical axis. The new relations describing the mirrored architecture are obtained by substituting $j = (N-1) - j$ and by rewriting the routing equations for the multiplier cell. The reference architecture remains the same. The new form of $MC_{i,N-j}$ is given below:

$$a_{(N-1)-j} \times b_i \times si_{i,(N-1)-j} \times ci_{i,(N-1)-j} \rightarrow so_{i,(N-1)-j}, co_{i,(N-1)-j} \tag{4.4a}$$
$$si_{i,(N-1)-j} = 0, \qquad i = 0 \tag{4.4b}$$
$$si_{i,(N-1)-j} = so_{i-1,(N-1)-j+1}, \quad i \in \{1, N-1\} \tag{4.4c}$$
$$ci_{i,(N-1)-j} = 0, \qquad i = 0 \tag{4.4d}$$
$$ci_{i,(N-1)-j} = co_{i-1,(N-1)-j}, \qquad i \in \{1, N-1\} \tag{4.4e}$$
$$a_{(N-1)-j}, b_i, si_{i,(N-1)-j}, ci_{i,(N-1)-j}, so_{i,(N-1)-j}, co_{i,(N-1)-j} \in \{0, 1\} \tag{4.4f}$$

The routing relations of the new multiplier scheme have changed. By posing $k = N - j$, relation 4.4c becomes $si_{i,k-1} = so_{i-1,k}$, while relation 4.4e becomes $ci_{i,k-1} = co_{i-1,k-1}$. These new relations indicate that the input signals for both chain-adder and mirrored-multiplier cell structures are issued by the same cells of the previous row. Therefore, the routing dissimilarity constraint is partially relaxed, since the input signals now have the same sources. However, each source is still attached to different ports on the sink cell, considering the multiplier's and the adder's basic cell. The different port attachment imposes the need for two 2 to 1 multiplexors under each cell in order to drive the inputs in a proper way. The elimination of the intermediate multiplexors is performed by the third stage of the Uniformity Transformation.

Step 3. Permutation on UC's Internal Port Map: Let us consider a unified basic cell structure. A straightforward port assignment generates the need of the internal multiplex usage. This happens because port si in the chain-adder's case has input from the above cell, but in mirrored-multiplier's case it has inputs from the left above cell. An equivalent condition holds for port ci.

To overcome this limitation, we propose to keep steady the unified cell's external port interface and alter the intra-cell port mapping. The altering can be made as a simple permutation operation on the UC's internal port map. So, let us assume that the port interface template of the unified cell is the binary vector $\{I_1, I_2, I_3, I_4, I_5, I_6, O_1, O_2\}$, where I_i stands for the inputs and O_j for the outputs respectively.

In case of adder/subtractor mode the internal port map is $\{M_1, M_0, a_j, xxx, \underline{si_{i,j}, ci_{i,j}}, co_{i,j}, so_{i,j}\}$. The xxx input port is not used because three input ports are sufficient

for the adder case. In case of multiplication mode the internal port map follows the template $\{M_1, M_0, a_j, b_i, \underline{ci_{i,j}}, \underline{si_{i,j}}, co_{i,j}, so_{i,j}\}$ with respect to, in both cases, the external port declaration of the reference architecture. The b_i input port is bound to the vertical's multiplicand bit.

The permutation over the internal port map eliminates the need of intermediate multiplexors by properly driving the input and output signals of each cell. It is performed at design time and actually pre-routes the input and output ports of each UC to its internal combinational logic (CL). The CL operates on the pre-routed wires and computes the proper values according to the control signals $\{M_1, M_0\}$. The internal semantic values of the two rightmost input and the two output ports of the UC alter when the UC is configured as multiplication or addition/subtraction cell (*STEP 3* in Fig. 4.2). However, the external routing scheme for the whole architecture is preserved stable.

The three stages form the proposed Uniformity Transformation. A stable and canonical routing scheme is exposed for efficient mapping of multiple arithmetic behaviors, on the same architecture, minimizing the routing complexity and circuits' criticality imposed by interconnection dissimilarities (Fig. 4.2, *STEP 4*). The feasibility of multiple behavior mapping generates opportunities of flexible, run time configurable and efficient arithmetic units.

The Unified Cell Structure

The UC design procedure concerns the design of the internal logic of the UCs, in order to maximize hardware sharing among the different configurations. The UC is actually a mapping of the desired bit-level arithmetic behaviors into a single module.

The problem of mapping together multiple behaviors onto the same circuit has been stated in [126], [127]. Given the dataflow graphs of the behaviors, they try to minimize area complexity by efficiently reusing the allocated resources. These techniques operate at a more abstract level and are based on hardware sharing of basic RTL components. In order to design efficiently the unified cell, we have to work on a lower level of abstraction (gate level).

In a straightforward implementation of the UC, Fig. 4.3a, the arithmetic behaviors are instantiated as sub-modules (AC: addition basic cell, SC: subtraction basic cell, MC: multiplication basic cell) into the same top module to form the unified cell. The desired behavior of the UC results from proper control signals $\{M_1, M_0\}$ which drive an output multiplexer. However, this straightforward design methodology introduces a large area overhead with a low utilization degree of the overall circuit of each cell. Large area overheads are introduced because of the un-exploited common gate-level datapaths, which exist inside the module. The main reason that this

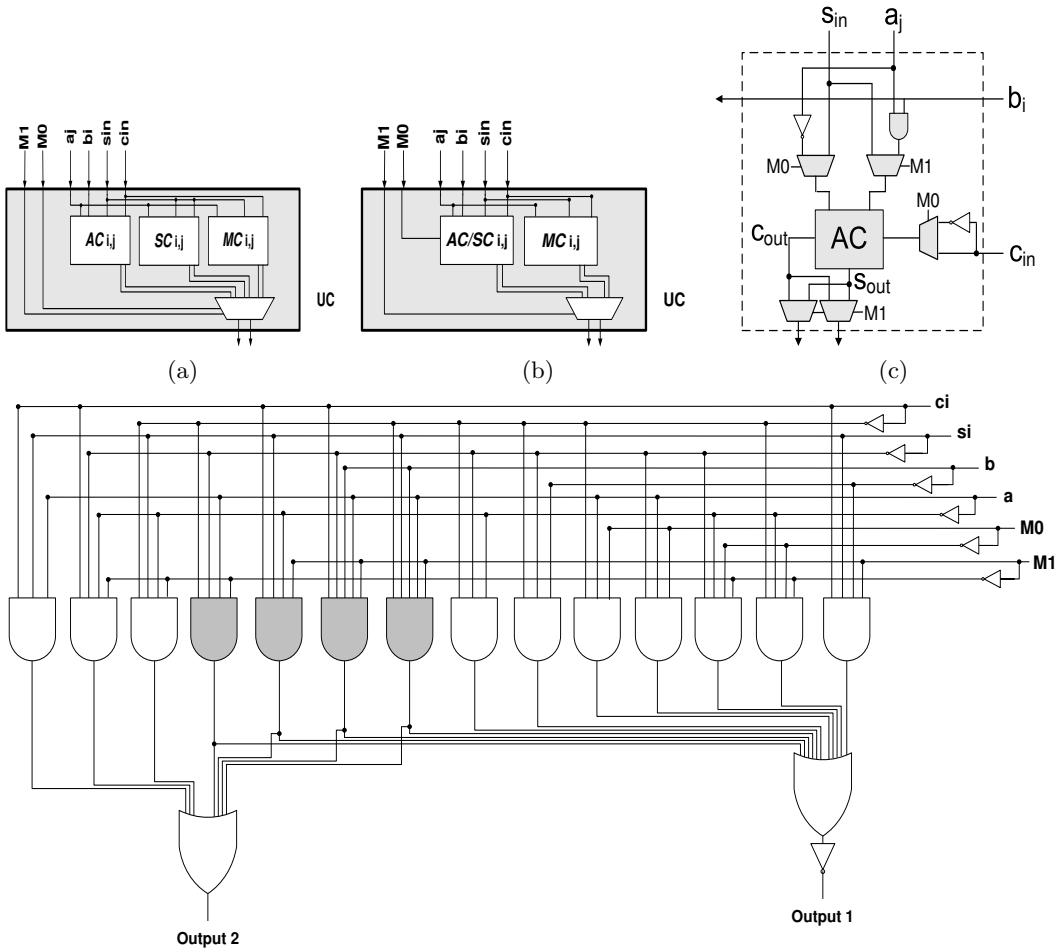


Figure 4.3.: Several implementations of the UC a) Straightforward with no sharing, b) Straightforward with AC/SC sharing, c) Optimized at the RTL, d) Optimized with the proposed technique.

happens is that this implementation does not take into account the inter-cell sharing optimizations at the logic level.

To overcome this inefficiency, we followed a design procedure which optimizes the hardware sharing among the behaviors mapped on a unified cell. The proposed methodology is based on the ESPRESSO logic minimization tool [45] and on the Output Phase Optimization technique presented by Sasao [128].

The Output Phase Optimization technique searches the design space of a multi-output Boolean function in order to find the set of the outputs' phases which minimize the needed product terms. The minimization of the product terms of a Boolean function leads to smaller area circuitry.

The ESPRESSO tool incorporates the utility of performing output phase optimization as a user specified parameter. We performed exhaustive search granted by the fact that the unified cell has a small design space. In all cases, we considered the permutation over the UC internal port map, as imposed by the last step of the Uniformity Transformation.

The design of a UC consists of 4 stages. At first, the truth tables of the desired cell's behaviors/configurations are formed in PLA format. In our case, three truth tables are formed. The first one describes the Boolean function of the multiplier cell, the second one describes the adder's cell function and the third one the subtractor's cell function.

The next step is the concatenation of the truth tables. A proper number of control bits is introduced to form a single unified truth table for all the expected cell's configurations (MC, AC, SC). The unified truth table aggregates all the configurations in a single multi-output function, so that the common gate datapaths between the desired behaviors are revealed to the ESPRESSO tool, maximizing the gate sharing among configurations.

The unified truth table is supplied at the ESPRESSO tool which outputs the overall minimized logic expression. As it was mentioned, the ESPRESSO tool is properly tuned to perform logic minimization with respect to the output phase optimization technique. The minimized output is still in PLA format. An equivalent circuit can be formed in RTL so as the functional correctness is tested.

The synthesis of the cell can be performed using either the minimized PLA or its RTL equivalent. In order to prove the efficiency of our approach in designing the UC internal structure, we synthesized the UC's configurations for the cases of a) a straightforward implementation (Fig. 4.3a) with no sharing, b) a straightforward implementation (Fig. 4.3b) in which the AC and the SC are combined together (AC/SC sharing), c) a sub-optimized UC using custom RTL components (full sharing at RTL level, Fig. 4.3c) and d) a UC based on the described technique (sharing at gate level, Fig. 4.3d), using the Synopsys synthesis tool suite [106]. The synthesis was made according to the standard cell design methodology using the TSMC 0.13 um technology library [107]. All the configurations were synthesized under the timing constraint of 0.4 ns for the UC 's propagation delay. The comparative results are reported in Table 4.1. Our methodology delivers area savings of 71.6% and power savings of 79%, comparing with the first straightforward approach (UC_{STRFW1}). In comparison with the second straightforward approach (UC_{STRFW2}), in which the AC and SC are shared, area and power gains of 27.5% and 68.2% are reported, respectively. Compared with the sub-optimized RTL implementation of the UC, the savings in terms of area are up to 7.8% and in terms of power up to 40.9%. In Table 4.1, we have also included the synthesis results of the simple MC to indicate the overheads imposed of the optimized UC in comparison with the standard multiplier cell. The UC requires approximately double area and dissipates double power, compared with the MC cell. However, the UC can be

Table 4.1.: ***Comparison of the UC's Implementation Strategies.***

Cell	Area (um^2)	Power (uW)	Area Overheads	Power Overheads
UC_{STRFW1}	370.2	271.8	71.6%	79%
UC_{STRFW2}	275.1	255.3	27.5%	68.2%
UC_{SUBOPT}	232.6	213.9	7.8%	40.9%
UC_{OPT}	215.7	151.8	0	0
MC	93.4	63.2	-56.7%	-58.3%

configured either as multiplication cell, or as addition cell or as subtraction cell.

4.4. Reconfigurable Architectural Template: Circuit Level Analysis

Based on the Flexibility Inlining technique, a reconfigurable architecture template was extracted. The micro-architectural details and attributes of the proposed architectural template are provided in Section 4.5.1, since this type of abstractions are used during automated architectural synthesis. In this section, we provide a the detailed circuit level description of the proposed reconfigurable datapath. The analysis is based on a specific instantiation of the architectural template. Alternative instantiations of the architecture are evaluated in Subsection 4.6.4.

Fig. 4.4 shows the architectural instantiation that will serve as guide-specification. The architecture comprises (i) a pipelined 16×12 array of unified cells along with a 16×4 array of multiplier's cells (the Pipelined Reconfigurable Arithmetic Unit), (ii) a multiplexer-based interconnection network, (iii) a small number of registers for intermediate variables' storage (the RegBank), (iv) a CS to conventional binary arithmetic conversion module, (v) a context register and (vi) three reversing-order modules. The unified cells inside each reconfigurable pipeline stage are interconnected according to the canonical scheme of the reference architecture.

The Reconfigurable Arithmetic Unit (RAU) array consists of four Flexible Pipeline Stages (FPSs). Each FPS can operate either as a completely independent flexible arithmetic datapath or in coordination with the above stage. When independent execution is performed, the instantiated kernel emulates a multi-output functional unit. The coordinated execution takes place when the pipeline's datapath operates on the previous stage's outputs. The circuit inside each FPS is fully combinational enabling efficient operation chaining of the mapped behavior. RAU can be clocked by high frequency clocks because of its pipelined structure and its CS internal architecture.

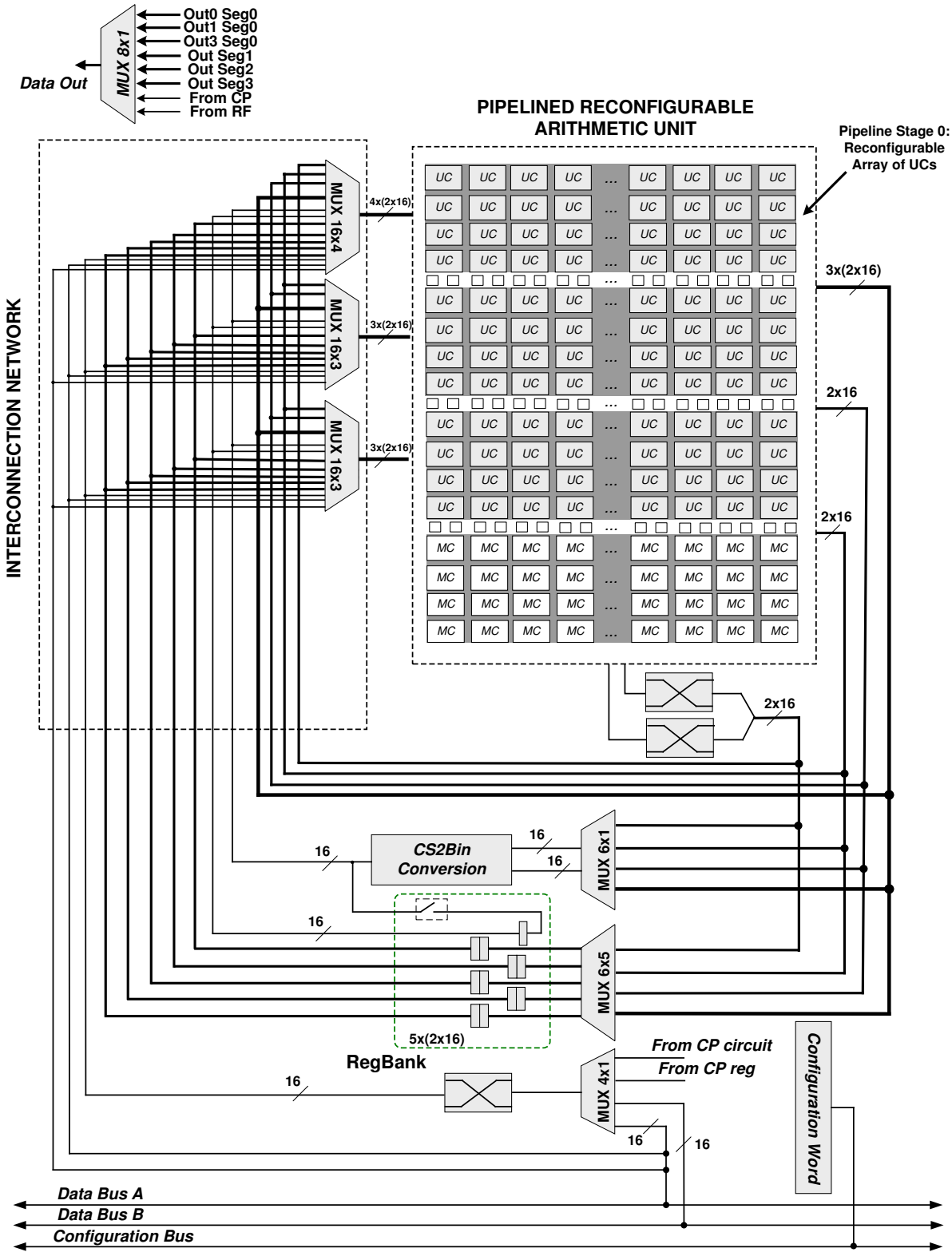


Figure 4.4.: A detailed instance of the Reconfigurable Kernel Architecture.

Fig. 4.5 depicts the internal structure of the first two FPSs for the instantiated RAU. The FPSs have been designed with a degree of heterogeneity. FPS 0 includes an intermediate line of 2-to-1 multiplexors (above the third row), which enable this stage to perform in parallel two independent CS-additions or subtractions of two, three ($O0$) or four operands($O1$, $O3$), as shown in Fig. 4.5). In contrast, FPS 1 contains the multiplexer intermediate line at the beginning of each stage so as to route properly the pipeline's inputs. Also, it includes only one output port (i.e., $O7$). The third reconfigurable pipeline stage, which is not included in Fig. 4.5, has the same internal structure as FPS 1. The three FPSs consist of unified cells and are run-time reconfigurable. The last pipeline stage in Fig. 4.4 is not flexible. It needs no configuration as it consists only of gated adders (Fig. 4.6a), the basic component of array multiplier cells. It is used only as the final stage of a 16×16 multiplication. This design decision reduces the overall hardware area and the required wiring of the RAU. Analyzing the RAU, it is shown the way flexibility inlining technique merged into the datapath of a pipelined array-multiplier the arithmetic behaviors of chain additions and subtractions. Also, through the instantiated heterogeneity, it is shown that flexibility inlining is able to generate and support a large set of circuit level configurations.

The reversing-module (Fig. 4.6b) consists of a simple hardwired 16×16 reversing-order crossbar which assigns correctly the input data to the UCs when the overall kernel is configured to perform a multiplication. The two wire crossbars, which are allocated in the bottom of the last pipeline stage, target the highest order carry and sum bits respectively, after the completion of a 16×16 multiplication operation. Thus, they enable the final carry-propagate addition, for converting CS to conventional binary to be performed, with no extra cycle loss.

The arithmetic conversion module (CS2Bin) implements the conversion of CS to conventional binary format. The circuitry for arithmetic conversion consists of a carry skip adder/subtractor [123]. The two-operand carry-skip adder/subtractor contributes to the result's conversion from the redundant CS arithmetic format to the conventional binary format. The carry-skip adder reduces the requisite time for carry propagation by skipping over groups of 4 consecutive adder bits, without wasting many hardware resources like the Carry-Look-Ahead adders [123]. For further optimization of the carry-skip circuit, the optimized carry-skip adder scheme with low carry-absorb time proposed in [129], can be adopted. The case of overflown results can be handled by the well known techniques referenced in [123], [130].

The CS to binary conversion of the multiple CS formatted RAU's outputs forms a potential bottleneck, since only one arithmetic conversion module is allocated. In order to tackle this potential bottleneck without spending additional hardware resources (i.e., an extra allocated CS2Bin module), the instantiated architecture template enables arithmetic operations over redundant CS formatted operands. RAU's output bits (in CS format) can be used directly to the next RAU computation without passing from the arithmetic conversion module. The efficient manipulation of

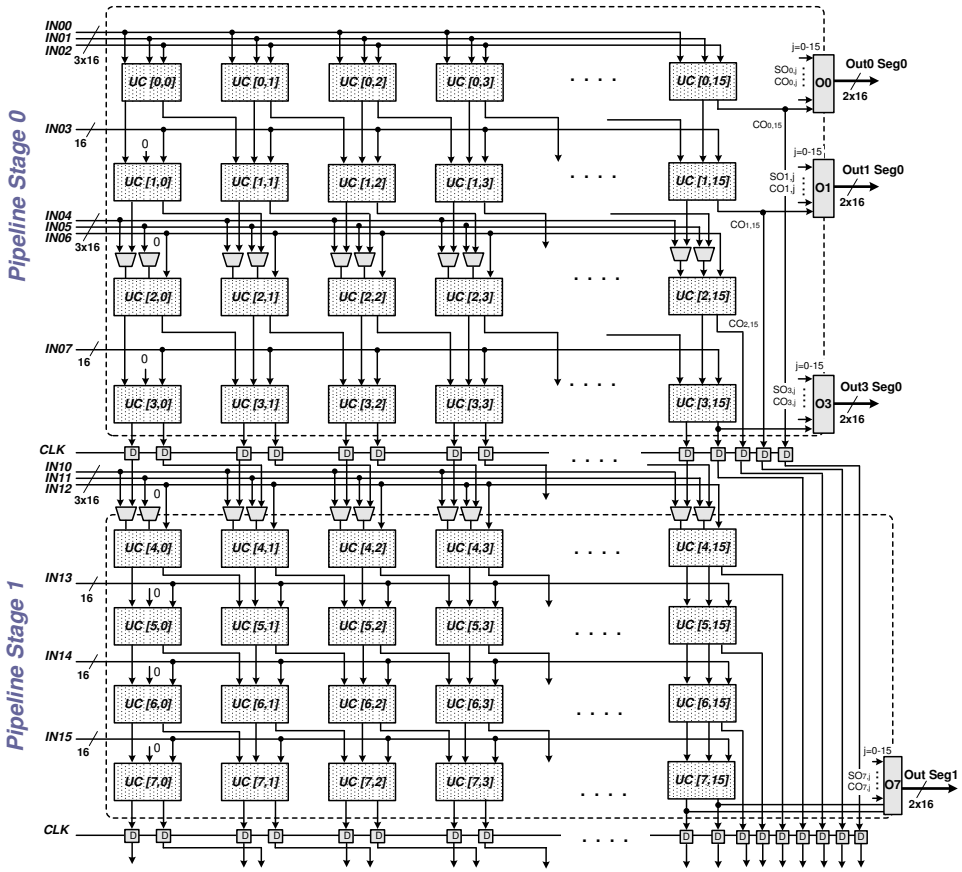


Figure 4.5.: Internal structure of pipeline stages 0-1. Stage 2 is identical to stage 1. Stage 3 is the typical CS array multiplier scheme.

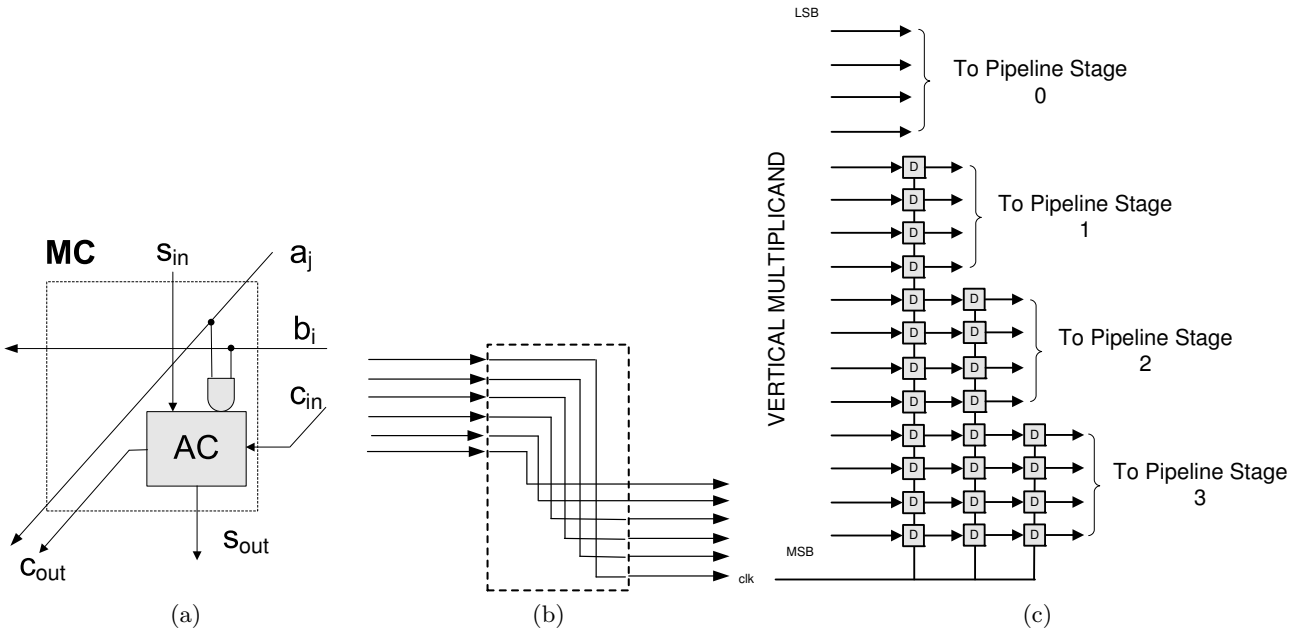


Figure 4.6.: Primitive components a) The gated-adder MC circuit, b) The reversing module, c) he synchronization circuit for the vertical multiplicand.

this potential bottleneck is also supported by the intra-pipeline stage operation chaining which contributes to less intermediate results. With proper operation scheduling and by continuously supplying the carry-skip adder with CS inputs, the bottleneck can be further relaxed. The combination of the last two strategies contributes to early producing the binary formatted values needed in future control steps.

Taking into account the pipelined organization of the RAU, multiplication operations larger than 16×4 are mapped as a pipelined carry-save operation. A 16×16 multiplication is completed in 4 clock cycles, with the full precision of a 32-bit final product. The early generated product bits (product's Least Significant Bits (LSBs)) are propagated through the stages of the RAU in a pipelined way (Fig. 4.5). In case that 16-bit product's precision is considered enough, the extra Flip-Flops can be ignored.

In the general case, multiplication operations can be initialized from any FPS and complete to any FPS. Without loss of generality, in this instantiation of the reconfigurable architecture, we assumed that every multiplication operation has to be initiated FPS 0 and move along the FPSs in the conventional order. The vertical multiplicand (input port b_i of the UC) has to be segmented in four groups of 4-bits and every group has to be issued to its pipeline stage in a synchronized manner. This synchronization mechanism of the vertical multiplicands has not been included in Fig. 4.4, for clarity reasons. It is illustrated separately in Fig.

Table 4.2.: ***Operation Level Reconfigurability of Each FPS In Fig. 4.4.***

Operation Mode	FPS 0	FPS 1	FPS 2	FPS 3
Chain-Add of up to 6 binary or up to 3 CS integers	⋆	⋆	⋆	-
Chain-Sub of up to 6 binary integers	⋆	⋆	⋆	-
Chain-Sub of up to 6 binary or up to 3 CS integers	⋆	⋆	⋆	-
Chain-Add with both CS and binary integers	⋆	⋆	⋆	-
Chain-Sub with both CS and binary integers	⋆	⋆	⋆	-
Two Chain-Adds each of up to 4 binary or up to 2 CS integers	⋆	-	-	-
Two Chain-Subs each of up to 4 binary or up to 2 CS integers	⋆	-	-	-
One Chain-Add and one Chain-Sub each of up to 4 binary or up to 2 CS integers	⋆	-	-	-
Two Add-Sub or Sub-Add operations each of up to 4 binary integers	⋆	-	-	-
16×4 Multiplier	⋆	⋆	⋆	⋆

4.6c. It is composed of 24 D-Flip-Flops interconnected in a way which permits (i) the segmentation of the vertical multiplicand's word in a cycle by cycle basis and (ii) the correct issuing of the segmented words to the proper pipeline stage of the RAU.

4.4.1. Configurability of the Architecture

In order to provide a detailed description concerning the configurability of the proposed architecture, we use the previous instantiated template. The following analysis can be applied for every architecture template in a straightforward manner.

The configurability of the instantiated architecture is presented both at the architecture and at the operation level. At the architecture level the RAU is able to operate (i) as one multi-cycle and pipelined functional unit (i.e. as a 4-way pipelined multiplier) or (ii) as 4 stand-alone and independent single cycle computational units (adders/subtractors). The RAU's pipelined structure permits high throughput applications to be mapped, while independent execution of each pipeline stage enables the exploitation of the application's inherent operation level parallelism.

Operation level reconfigurability is referred to each pipeline stage separately. Table 4.2 reports the basic operation modes of each reconfigurable or non-reconfigurable pipeline stage of the architecture in Fig. 4.4. Table 4.2 shows that each FPS is able to operate as an independent processing element which can be configured either (i) as a 16x4 multiplier or (ii) as a chain adder-subtractor of up to 6 binary formatted

operands or (iii) as a chain adder-subtractor of up to 3 CS formatted operands or (iv) as a chain adder-subtractor of a number of hybrid formatted operands (i.e., 2 CS operands and 2 conventional binary operands).

Additionally, Table 4.2 depicts that configurability is not the same for all the four pipelined stages. FPS 0 of Fig. 4.5, which is enhanced by the inter-row multiplexors, exhibits greater configurability compared with the remaining ones. The non-enhanced stages (i.e., FPS 1 in Fig. 4.5) can be configured either (i) as a chain-adder/subtractor of 6 binary formatted inputs, or (ii) as a chain-adder/subtractor of 3 carry-save formatted inputs, or (iii) as a chain carry-save adder with hybrid formatted inputs, or (iv) as a 16x4 multiplier, (v) or as part of a larger 16x16 multiplication process. Apart from the aforementioned operation modes, FPS 0 can perform also (vi) 2 independent CS-additions/subtractions, or (vii) one independent single addition/subtraction and one 4 operand chain-addition etc.

The proposed architecture can be reconfigured in a cycle by cycle basis by writing configuration words to the configuration context register. The context register drives all the multiplexors allocated inside or outside of the RAU. The format of the configuration word is depicted in Fig. 4.7. It is organized in groups of control bits, starting from the bits that control the configuration of FPS 0, continuing with the bits which control the other two flexible pipeline stages (FPS 3 needs no configuration) and ending with the control bits dedicated to the remaining multiplexors of the kernel architecture. The modular organization of the configuration word enables easy parametrization of the reconfigurable kernel architecture in order to synthesize architectural prototypes with various degrees of configurability (Subsection 4.6.4).

The heterogeneity between the pipeline stages is reflected into the configuration word. Thus, the configuration group of FPS 0, which delivers higher configurability, is larger (25 bits long) in comparison with the configuration group of pipeline stage 1 (15 bits long). The configurability of each FPS affects the configuration group of the non-RAU components too. This happens because the number of outputs ports per pipeline stage, which form the set of inputs of the non-RAU multiplexors, alternate according to the configuration capabilities of each pipeline stage.

We have considered the following naming convention rules, in order to make readable the configuration's word format (Fig. 4.7). Inside each configuration group, the name of the sub-words which control the operation mode, starts with the sequence "M0", "M1" or "M2" according to their functionality, and is followed by the identifier "SEG" for segment and the pipeline's segment number. The remaining subwords inside each configuration group control the multiplexors allocated in the reconfigurable kernel architecture. Their name starts with the identifier "SEL" for selection, followed by the name of the component which each sub-word refers to (i.e. SEL_RF2: sub-word controlling the input selection of the second register in the register file).

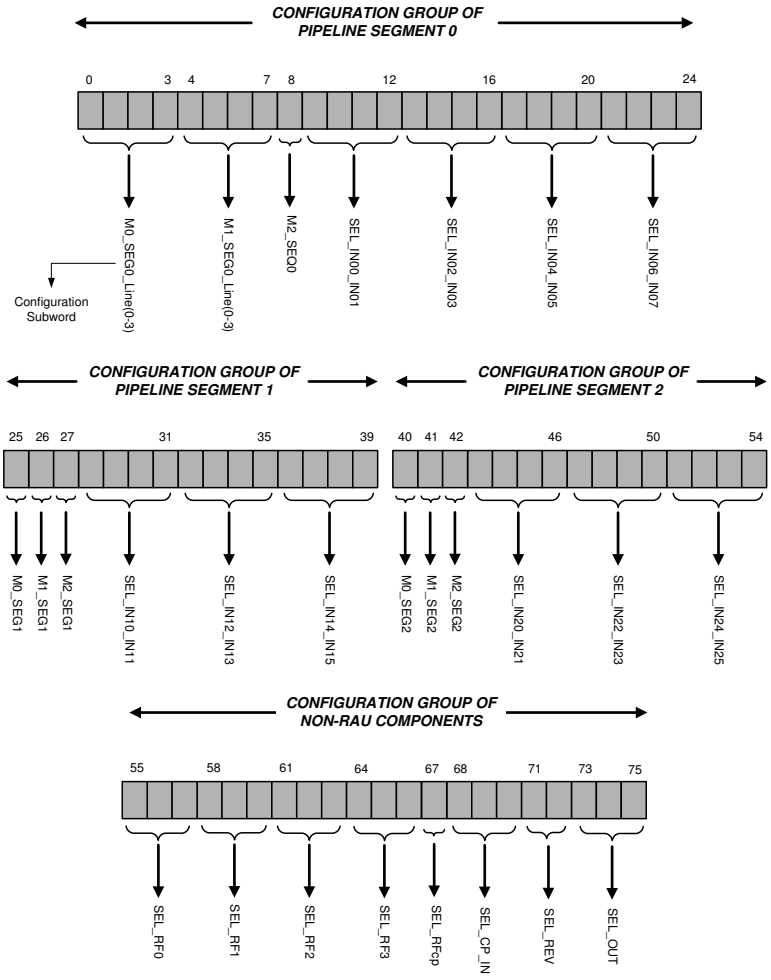


Figure 4.7.: The format of the configuration word.

Table 4.3.: ***Operation Mode of Line i*** $\in \{0,1,2,3\}$ ***of Pipelined Stage 0.***

Operation	$M0\_SEG0\_Line(i)$	$M1\_SEG0\_Line(i)$
Addition	0	0
Subtraction	1	0
Multiplication	0	1
Don't Care	1	1

Each configuration group in Fig. 4.7 for the FPSs, starts with the configuration bits controlling the UCs' functionality. The bits ($M0\_SEG_i$, $M1\_SEG_i$, $i \in \{1,2\}$) control each UC's function, which is either addition, subtraction or multiplication. The configuration bit $M2\_SEG_i$ controls the intra RAU multiplexors which pass either an external input data or the RAU's internal propagated data (Fig. 4.5). For FPS 0, the configuration subwords ($M0\_SEG0\_Line(i)$, $M1\_SEG0\_Line(i)$, $i \in \{0,1,2,3\}$) configure each addition line separately. Thus they enable the FPS to perform complex chained operations (i.e., add-sub-add, two parallel add-sub sub-add etc.). Table 4.3 shows the operation modes of each line in FPS 0, according to the values of $M0\_SEG0\_Line(i)$, $M1\_SEG0\_Line(i)$ control signals. FPS 1 and FPS 2 do not support per addition configurability and they can be configured either as 4 chained adders/subtractors or 16×4 multipliers. This is the reason why the configuration sub-words of FPS 1 and 2 include only 1-bit for the signals ($M0\_SEG_i$, $M1\_SEG_i$).

The remaining sub-words, inside each FPS configuration group, perform the selection the input operands. Given that the input data can be in the CS arithmetic format, the selection multiplexors pass 32-bit wide words (2×16-bit), concatenating together the sum and the carry bits of a CS number. Each CS formatted input refers to two 16-bit input ports of the RAU. In case that a conventional binary formatted number is about to be processed in the RAU, the carry's 16-bits are passed as zeros. In the configuration group of FPS 0, there are four configuration sub-words of selection bits ($SEL\_IN_i\_IN_{i+1}$), since FPS 0 incorporates up to 8 binary inputs(=4 CS inputs). Each of these configuration sub-words consists of 4-bit, in order to select between the sixteen CS formatted inputs. The other two flexible pipeline stages(= FPSs 1,2) incorporates 6 binary inputs. Thus, three configuration sub-words of selection bits exist in their configuration group. The non-configurable pipeline stage does not contribute at all to the configuration word. It accepts the horizontal multiplicand from the previous pipeline stage and the vertical multiplicand from the synchronization mechanism depicted in Fig. 4.6c.

The last configuration group controls all the non-RAU components. Each register in the register file is controlled by three control bits (SEL_RF_i) which select between six inputs. The CS number that is going to be converted is selected by the SEL_CP_IN sub-word of the configuration word. One bit (SEL_RF_{CP}) controls if the converted data are going to be stored at one 16-bit register for future use or if they are going to be fed into the RAU just after the arithmetic conversion. The 16-bit operand

that counts for reversing its bit order is selected by the SEL_REV configuration bits. Finally, the output of the reconfigurable architecture results from the SEL_OUT bits of the configuration word.

The configuration word of the proposed reconfigurable architecture is up to 76-bits wide which resembles the instruction length of current VLIW processors [131], [132]. The configuration word can of course be encoded but this would require a decoder unit to be embedded into the kernel architecture augmenting the critical path and the overall area of the implementation. For these reasons, we preferred to fully expose the control of the kernel architecture delivering the original configuration word to the context register.

4.5. Architectural Synthesis Framework

This section introduces a new synthesis methodology which delivers high performance and area efficient flexible datapaths. Architecturally optimized coarse-grained reconfigurable datapaths demand the close coupling between the applications' behavior and the structure of the underlying hardware. This requirement has made HLS techniques [44] an attractive solution for the design of coarse-grained reconfigurable datapaths. Many architectural HLS optimization concepts like operation level parallelism (horizontal parallelism), pipelined execution (vertical parallelism) or operation chaining [6], have been incorporated in the architecture specification of coarse-grained datapaths, especially for row-based ones. However, there is no any known approach in the literature that exploits the advantages of: (i) horizontal parallelism, (ii) vertical parallelism and (iii) operation chaining, into a single flexible architecture. We show that designing reconfigurable datapaths with inlined flexibility based on the combined exploitation of the three architectural optimization concepts, deliver solutions with higher performance and better area complexity than the existing approaches.

4.5.1. Micro-architectural Abstractions of the Reconfigurable Architectural Template

In order to develop synthesis algorithms for efficient code mapping onto the proposed reconfigurable datapath, the micro-architectural attributes of the underlying hardware structures have to be abstracted at a higher level than the circuit level 4.3. Micro-architectural abstraction enables high level computational models, generated from an algorithmic description, to be mapped onto machine-specific components. Furthermore, micro-architectural abstraction enables the exploitation of hardware-dependent architectural characteristics in a high level context. We perform micro-architectural abstraction of the proposed reconfigurable datapath by (i)

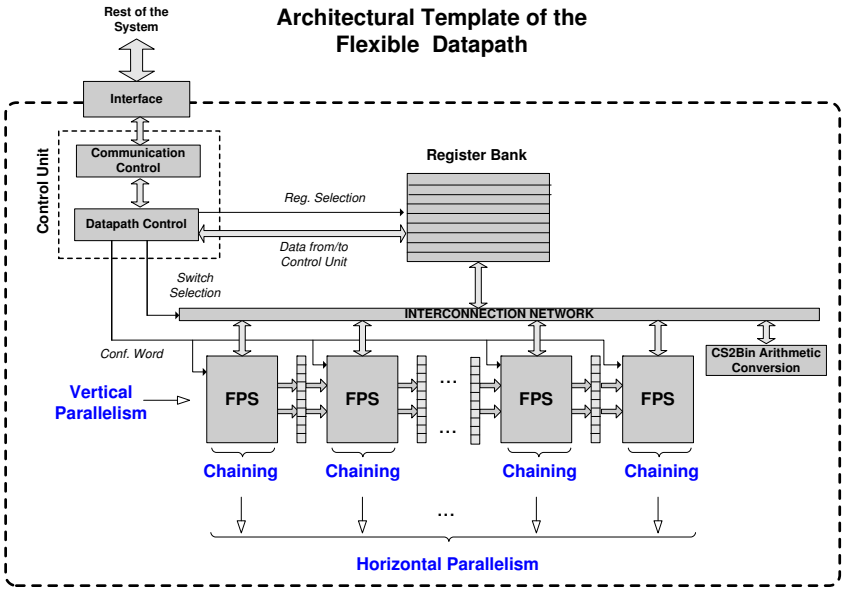


Figure 4.8.: Micro-architectural view of the reconfigurable datapath.

revealing the advanced architectural optimizations integrated in the proposed datapath and (ii) analyzing the architecture specific instruction set (operation template library).

A micro-architectural view of the proposed coarse-grained reconfigurable template is shown in Fig. 4.8. In a macroscopic view, it comprises of six major components: (i) the flexible pipeline stages (FPS), (ii) the pipeline registers between each stage, (iii) a register bank for storage, (iv) the interconnection network, (v) a CS to binary arithmetic conversion module (CS2Bin) and (vi) the control unit which drives the overall architecture (configuration words to the flexible pipeline stages and selection signals). The architecture in this section is considered homogeneous. All the FPSs are considered as multi-input-single-output components, like FPS 1 in Section 4.4). We adopt an execution model in which the configuration words are provided by the control unit and the actual data are loaded before the execution phase. The architecture has been designed to operate on word-level operands of 16-bit, which are considered adequate for the computationally-intensive DSP applications [46]. It can be extended straightforwardly to support operands with bit-width larger than 16-bit. Fig. 4.8 shows the way in which the flexible portion of the architecture (FPS units) integrates the three architectural optimizations in a combined manner.

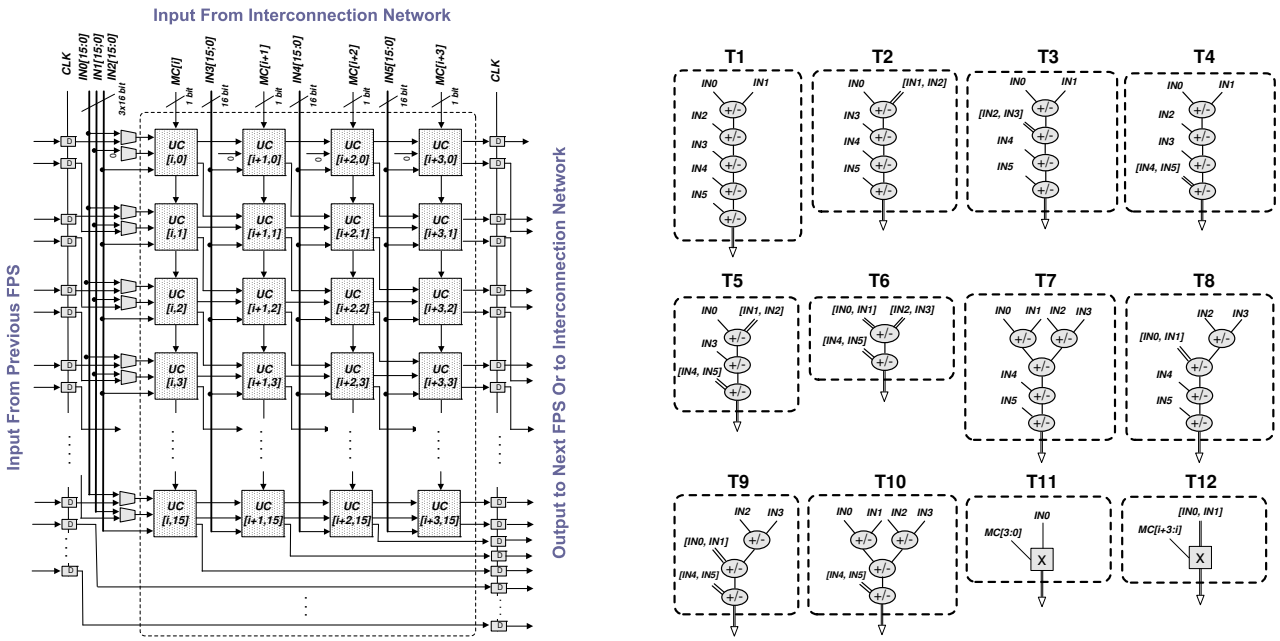
Horizontal parallelism is supported at the architecture's organization level through the number of FPS operations that can be executed simultaneously in the same control step. In architectural synthesis, it is restricted by the imposed resource constraints. In the proposed flexible datapath, each FPS can operate independently, simultaneously with the others and write back its results to the register bank. The number of allocated FPS units forms the resource constraint imposed to the

datapath. Thus, the horizontal parallelism is proportional to the number of FPS units.

Vertical parallelism is also supported at the architecture's organization level. It is provided by the inter-pipeline stage registers. A pipelined computation can be performed with successive flexible pipeline stages. It can be initiated/terminated from/to any FPS. Multiple pipelined computations, constrained by the horizontal's parallelism degree, can be initiated and flowed through the FPS units in a circular manner. By this way, cooperation between horizontal and vertical parallelism is enabled at a finer granularity in comparison to the NISC approach [17], where vertical parallelism strictly follows only the conventional flow inside each pipelined functional unit.

Operation chaining is enabled in an intra-FPS manner. Each FPS is able to perform chaining of data-depend addition or subtraction operations. This architectural feature is supported through the internal structure of the FPS unit. A signal detailed internal structure of a FPS unit is depicted in Fig. 4.9a in order to enable the easy matching between the circuit-level FPS signals with the micro-architectural operation library (Fig. 4.9b). Since CS arithmetic [53] has been considered to eliminate the time consuming carry-propagation, data of both arithmetic representations (i) CS and (ii) conventional binary, are handled by the FPS units. The internal structure of each FPS enables the computation of either a 16×4 bit multiplication or a chained addition/subtraction operation template of up to 6 integer formatted inputs. No generic boolean operations are performed by the FPS. Fig. 4.9b depicts the basic operation templates that can be mapped onto a FPS in one control step. Single input/output lines declare binary formatted data while double input/output lines CS formatted data. Notice that the T1-T10 operation templates implement the arithmetic behavior of a 6:2 compressor circuit of integer operands (or equally a 3:2 compressor of CS formatted operands). The inputs of the first UCs' row inside each reconfigurable pipeline stage (Fig. 4.9a) are selected between the previous stage outputs and the inputs from the external interconnection network. Composite multiplication architectures (larger than 16×4 bit) can be mapped onto the FPS units of the proposed architectural template (Fig. 4.8). For example, a 16×16 multiplication is decomposed to four data-dependent multiplications of 16×4 bit-width, each of which is mapped to a FPS unit in a pipelined manner.

The Control Unit is a Finite State Machine (FSM) which drives the entire flexible architecture through proper control signals. Conceptually, it can be partitioned in the communication control unit and the datapath control unit. The communication control unit contains interface logic used when loading data to the local register bank. In case of data loading, it triggers the datapath control unit to write the received data to the local register bank. Except from data loading, the datapath control unit operates in a cycle by cycle basis and generates (i) the configuration words which guide the operation of each FPS, (ii) the switch selection signals in order to drive the routing paths of the interconnection network and (iii)



(a) (b)

Figure 4.9.: a) Internal structure of a FPS unit. IN0-IN5 forms the additive inputs. In case of multiplication operation: (i) the CS number [IN0, IN1] forms the multiplier. (ii) The MC[i] forms the multiplicand (templates T11, T12 in Fig. 4.9b). (iii) The i^{th} column produces the i^{th} partial product. b) Basic Template Library of a FPS unit.

the read/write selection signals of the register bank.

The register bank is composed of scratch registers that provide the necessary memory bandwidth for each DSP kernel. They perform the storage of loaded data, intermediate results and the sharing of variables among non-adjacent FPS units. Since the set of the hardware accelerated DSP kernels is known during design time, the number of scratch registers is computed separately for each kernel and the minimum number of registers which satisfies all DSP kernels is allocated.

The interconnection network handles the communication between the register bank and the FPS units. A crossbar-based interconnection can be adopted as the one already presented in Section 4.3 or a specialized interconnection can be allocated according to the communication paths defined by the set of kernels. Specialized interconnections among various kernels can be generated using the techniques presented in [121], [122].

Finally, the arithmetic conversion module (CS2Bin) performs the conversion from the CS format to the conventional binary arithmetic representation. There are two cases in which the CS to conventional binary conversion is required. The first one occurs when a computed data in CS format has to be delivered to the rest of the system (primary output), given that the latter operates on conventional

binary arithmetic format. The second occurs in cases that a computed data in CS format forms the operand of a multiplication operation. This feature is an inherent restriction of the UC structure (Section 4.3), which imposes that each multiplication has to be initiated with the two operands in conventional binary format.

4.5.2. Qualitative Analysis: Proposed Flexible Datapath vs Row-Based Coarse-Grained Reconfigurable Architectures

This subsection demonstrates the beneficial features of the flexible datapath. In particular, we evaluate the proposed solution over conventional row-based coarse-grained reconfigurable architectures, considering in a qualitative manner the performance and area utilization design metrics.

The majority of coarse-grained reconfigurable architectures are cell-based [117], [118], [15], [49], [16]. Each coarse-grained reconfigurable cell (CGRC) is comprised of discrete computational components, such as one multiplier and one ALU. A general view of a CGRC is depicted in Fig. 4.10a. A closer look at the CGRC's structure reveals its inherent shortcomings. Specifically, its maximum operating frequency is constrained by the multiplication unit, which forms the critical path of the CGRC. The CGRC of Fig. 4.10a consists of two unbalanced execution paths, with large timing slacks every time it is configured to execute an ALU operation. Additionally, CGRCs suffer from inefficient utilization of their underlying hardware area, since in every control step only one of their allocated components is utilized. Reconfigurable architectures with aggressive chaining of CGRCs, i.e. the FCC (Fig. 4.10b) computational component [16], are greatly affected by the CGRC's inefficiencies. The potential of performance and area utilization gains of the proposed flexible datapath are analyzed based on a simple motivational example shown in Fig. 4.11.

Fig. 4.11a shows a behavioral specification represented using the Data-Flow Graph (DFG) model. We consider two different flexible datapaths in order to map the DFG of Fig. 4.11a. One datapath composed of two CGRCs of Fig. 4.10a and another datapath composed of four FPS units similar to Fig. 4.9a. Both datapaths have been found to allocate almost the same hardware area (rough estimation considering only the area of the functional units), while their critical path has been calculated so that the operation period of the CGRC is $T_{CGRC} = 3.9ns$ and for the FPS $T_{FPS} = 2.0ns$ (detailed synthesis results can be found in the Subsection 4.6.6).

The DFG is scheduled with a mobility based resource-constrained list scheduler [44], and the operations are bound to the flexible cells of each datapath. Fig. 4.11b and Fig. 4.11c illustrate the post scheduling allocation tables in case of the CGRC-based and of the FPS-based solution, respectively. The colors in the DFG have a

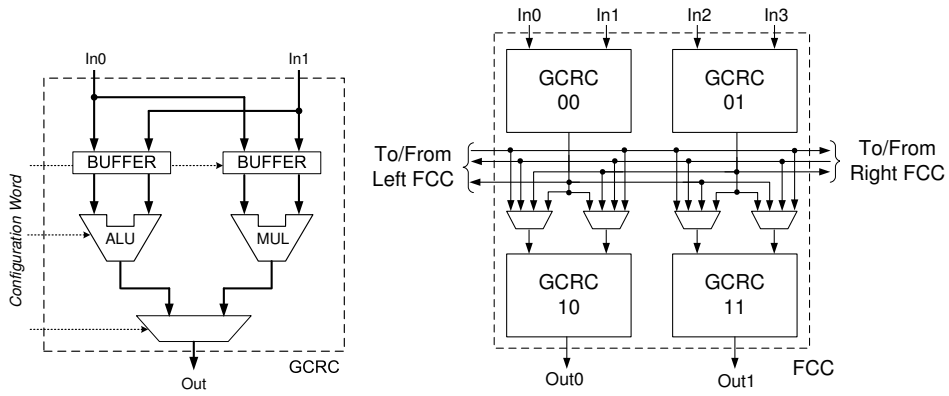


Figure 4.10.: Existing flexible computational cells: (a) Conventional coarse-grained reconfigurable cell (CGRC), (b) FCC reconfigurable cell [16].

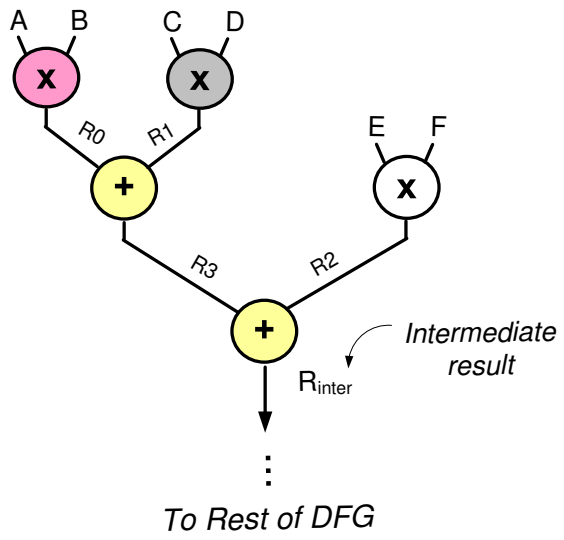
one-to-one correspondence with the colors in the post scheduling allocation tables. Each allocation table shows the number of control steps, the allocated resources, the unutilized area per control step and the execution time at each control step. The comparison of the rightmost columns of each allocation table shows that the proposed solution delivers total execution time nearly 2 ns less than the CGRC-based datapath, even for this small DFG.

In order to evaluate the hardware area utilization, we define the *Area Utilization Ratio (AUR)* metric:

$$AUR = [1 - \frac{\sum Area_{NonUtilized_i}}{Area_{tot} \cdot \#Cycles}] \cdot 100\% \tag{4.5}$$

The $Area_{NonUtilized_i}$ refers to the sum of the non-utilized combinational area of the reconfigurable cells for all control steps needed to complete the execution, while the $Area_{tot}$ is the total hardware area of the datapath.

The CGRC-based datapath delivers an area utilization ratio close to 50% (specifically 51.9% considering the values found in Section 4.6.7). So, half of the CGRC-based datapath is efficiently utilized. Since real world reconfigurable architectures [15], [49] comprise of up to four or more CGRCs (instead of two), the AUR is expected to have much lower values in these cases. On the other hand, the FPS-based solution delivers an area utilization ratio of 65%, which can be considered an efficient value for the specific DFG. Even though the same number of empty slots can be seen in Fig. 4.11b and Fig. 4.11c, the FPS-based datapath presents finer granularity. Thus, the unutilized hardware of a multiplier block in Fig. 4.11b has much greater impact than the unutilized hardware of a FPS block in Fig. 4.11c. Also, in Fig. 4.11c almost the half of the total unutilized hardware is due to control step 5, in which only one operation is available for execution. However, such a situation is rare with larger DFGs.



(a)

	CGRC 0		CGRC 1			
Control Step	**ALU 0**	**MUL 0**	**ALU 1**	**MUL 1**	**Non Utilized HW Area**	**Execution Time**
1	—	R0 ← AxB	—	R1 ← CxD	$2xArea_{ALU}$	3.9 ns
2	—	R2 ← ExF	R3 ← R0+R1	—	$Area_{ALU}+Area_{MUL}$	7.8 ns
3	R_{inter} ← R2+R3	—	—	—	$2xArea_{MUL}+Area_{ALU}$	11.7 ns
				Total	$3xArea_{MUL}+4xArea_{ALU}$	*11.7 ns*

(b)

Control Step	**FPS 0**	**FPS 1**	**FPS 2**	**FPS 3**	**Non Utilized HW Area**	**Execution Time**
1	$R0_{p0} \leftarrow AxB_{0-3}$	$R1_{p0} \leftarrow CxD_{0-3}$	$R2_{p0} \leftarrow ExF_{0-3}$	—	$Area_{FPS}$	2 ns
2	—	$R0_{p1} \leftarrow AxB_{4-7}$	$R1_{p1} \leftarrow CxD_{4-7}$	$R2_{p1} \leftarrow ExF_{4-7}$	$Area_{FPS}$	4 ns
3	$R2_{p2} \leftarrow ExF_{8-11}$	—	$R0_{p2} \leftarrow AxB_{8-11}$	$R1_{p2} \leftarrow CxD_{8-11}$	$Area_{FPS}$	6 ns
4	$R1 \leftarrow CxD_{12-15}$	$R2 \leftarrow ExF_{12-15}$	—	$R0 \leftarrow AxB_{12-15}$	$Area_{FPS}$	8 ns
5	—	$R_{inter} \leftarrow R2+R1+R0$	—	—	$3xArea_{FPS}$	10 ns
				Total	$7xArea_{FPS}$	*10 ns*

(c)

Figure 4.11.: DFG mapping example. (a) Example Data-Flow Graph. (b) Post scheduling allocation table of CGRC-based solution. (c) Post scheduling allocation table of the proposed solution.

4.5.3. Synthesis Methodology

The previous example showed the potential of the proposed solution to deliver high performance datapaths with efficient hardware utilization. Here, we introduce a synthesis methodology targeting FPS-based architectures.

The suggested synthesis methodology operates at a post hardware/software partitioning level. It receives as input the partition of the application which will be mapped onto hardware, specifically onto the proposed architecture. Thus, the set of DSP kernels $K = \{K_1, ..., K_i, ..., K_N\}$ for hardware acceleration is a priori known. A kernel, K_i, is modeled as a separate node in the Control-Flow Graph (CFG) of the input C specification. The synthesis flow schedules and binds individually each kernel onto the set of common datapath resources (FPS units, CS2Bin modules and registers). Thus, all the kernels share the same datapath resources. An optimized instance of the flexible architecture is generated in synthesizable Verilog RTL [47] that implements the desired behaviors. It consists of two main phases:

1. **Phase 1 performs a refinement of the Intermediate Representation (IR) extracted by the input C code (IR lowering procedure) along with some preprocessing optimizations. The lowering procedure derives the corresponding refined IR, having the resource model imposed by the FPS structure.**

2. **Phase 2 performs the mapping (scheduling-binding) of the lowered IR onto the flexible architectural template.**

Phase 1: Lowering and Pre-processing Optimization

Phase 1 of the synthesis methodology is depicted in Fig. 4.12. It accepts the kernels' description in C code and after standard compiler passes [133], creates the corresponding Control Data-Flow Graph. The input behavior is actually partitioned in code segments called Basic Blocks (BBs) which capture the control flow (CFG), and their DFGs which handle the data dependencies inside each BB. We adopt an execution model similar to [43]. It is based on the sequential execution of the BBs found in the extracted CDFG. The execution and synchronization of each BB are driven by appropriate control signals.

After CDFG extraction, phase 1 generates the Lowered-CDFG (L-CDFG). We first define the L-DFG and then the L-CDFG as follows:

Definition 1. A L-DFG is a directed acyclic graph $G_{L-DFG}(V_{L-DFG}, E_{L-DFG})$, where the vertex set $V_{L-DFG} = \{node_i; i = 1, ..., n_{nodes}\}$ is the set of n_{nodes} operations expressed according to the FPS resource model of the flexible architectural template (with primitives found within the FPS), while the edge set $E_{L-DFG} = \{(node_i, node_j, z); i, j = 1, ..., n_{nodes}, z = 1, 2\}$ represents

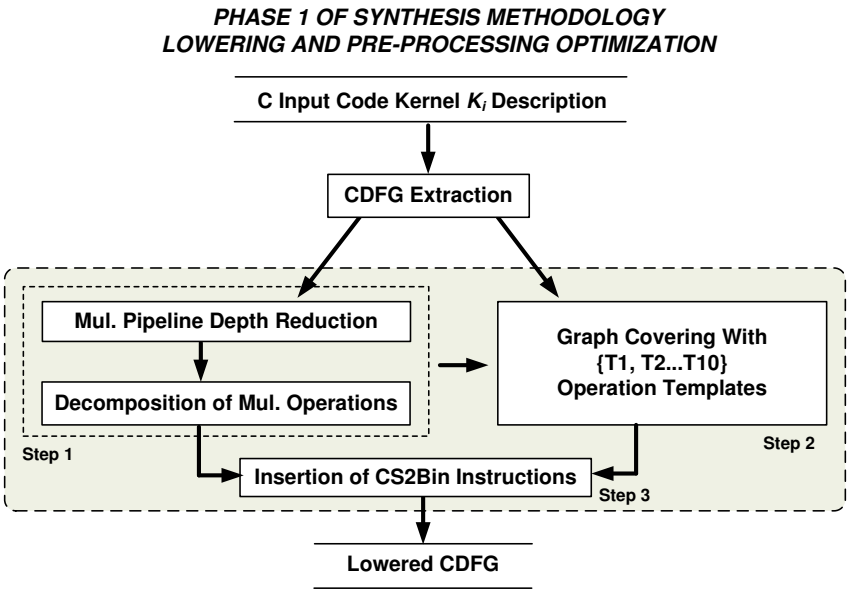


Figure 4.12.: Synthesis flow Phase 1: Lowering and pre-processing optimization.

the flow data dependencies together with their arithmetic format ($z = 1$: for conventional binary formatted integer, $z = 2$: for CS formatted integer).

Definition 2. A L-CDFG is a layered graph with two layers of nodes corresponding to the G_{CFG} and G_{L-DFG} graphs. The edges of the L-CDFG are the mapping between the nodes of the G_{L-DFG} and the Basic Blocks of the G_{CFG}.

The lowering procedure iterates through all the basic blocks of the initial CDFG and transforms the corresponding DFGs according to the FPS resource model found into the flexible architectural template, since conventional DFGs consider only resource models of two-input-single-output functional units. The L-DFGs extend the resource model of simple DFGs in order to integrate chained operation nodes (Fig. 4.9b) annotated with the arithmetic format of each input/output edge. The lowering procedure is completed in three steps.

Step 1 performs the manipulation of multiplication operations found into the initial DFG. Since each FPS performs 16×4 bit multiplication (T11, T12 templates of Fig. 4.9b), a straight-forward approach would consider that an integer multiplication (16×16 bit multiplication) is split into four data dependent 16×4 multiplication operations. However, DSP kernels often comprise the fixed coefficient multiplication operations (i.e, FIR filters, DCT etc.). The multiplication's pipeline depth reduction in Fig. 4.12 performs an optimization pass which exploits the presence of fixed coefficients in multiplication operations in order to reduce the number of pipeline stages. By reducing the number of pipeline stages for certain multiplication operations, the size of L-DFG is also reduced providing opportunities to achieve faster schedules.

The aforementioned optimization pass actually performs an annotation on the multiplication operations of the DFG with the *DepthFactor* variable (Fig. 4.13). *DepthFactor* indicates in how many pipeline stages a multiplication operation has to be decomposed. At first, all the multiplication operations of the DFG are inserted in a list (line 4). Then the values of input edges for each multiplication operation in this list are checked, to determine the *DepthFactor* in each case (lines 5-14). If one of the multiplication's input operands has a value smaller than $Value \leq 15$ then only one pipeline stage is needed for multiplication (lines 6-7). Two or three pipeline stages are allocated in case that the value of the fixed multiplication operand ranges $16 \leq Value \leq 255$ (lines 8-9) or $256 \leq Value \leq 4095$ (lines 10-11), respectively. In all other cases four pipeline stages are allocated (lines 12-13). There is no need to check if both the multiplicative operands are constants, since such cases are handled during the CDFG creation through a constant propagation compiler pass [133]. Now, each multiplication vertex in the initial DFG has been substituted by *k=DepthFactor* data-depended vertex nodes and the output edges of each inserted vertex is annotated with weight *z=2*, since each FPS produces the result in CS format.

Step 2 composes L-DFG's nodes to execute operation templates of chained additions/substractions (templates T1-T10 in Fig. 4.9b with partial covering enabled). By this way the intra-FPS operation chaining is extracted. The graph covering according to the T1-T10 templates is based on template matching/selection techniques [7], [48]. The template matching problem is equivalent to the subgraph isomorphism problem, which is known to be NP-complete [134]. In this work, the template library is a-priori defined. Thus, we follow the approach proposed in [7]. In brief, each node of the input DFG is annotated by a "match list", which indicates the possible node matches of the DFG nodes. Each element of the "match list" is a 3-tuple of the form $(X_{DFG}, Y_{TMPL}, TMPL\_ID)$, meaning that the DFG node X_{DFG} is covered by the Y_{TMPL} node found into the $TMPL\_ID$ template of the library. From node matches the overall template matches are extracted. The partial matching of the templates is guided by the designer. Specifically, the designer explicitly specifies any other partial covers of basic templates (Fig. 4.9b) in the form of extra library elements. The explicit specification of library elements enables a finer tractability of the template matching procedure since the commutativity of each template is bounded by the specific library elements [7]. Template generation is followed by a selection among the candidate template matches. The template selection decisions are manually guided by the designer. The selection criterion followed in this paper is the minimization of the kernel's critical path.

Step 3 performs the insertion of arithmetic conversion instructions in the L-DFG. An arithmetic conversion instruction performs the conversion from CS to binary format and it is mapped to the CS2Bin arithmetic conversion module (Fig. 4.8). A CS2Bin instruction transforms a CS edge (*z=2*) of the L-DFG into a binary edge (*z=1*). There are two cases for the insertion of a CS2Bin instruction: (i) The output nodes of the L-DFG, which are used either from the rest of the system outside the flexible architecture or in a next loop iteration inside the flexible architecture. (ii)

```
1. Input: DFG;
2. Output: Annotated DFG With Depth Factor For Each Mul Operation;
3.
4. Mul_Op_List := Find_Mul_Ops(DFG);
5. for each Opi in Mul_Op_List {
6.      if ((Opi.Input1 <= 15) or (Opi.Input2 <= 15))
7.              Opi.DepthFactor := 1;
8.      else if ((Opi.Input1 <= 255) or (Opi.Input2 <= 255))
9.              Opi.DepthFactor := 2;
10.     else if ((Opi.Input1 <= 4095) or (Opi.Input2 <= 4095))
11.             Opi.DepthFactor := 3;
12.     else
13.             Opi.DepthFactor := 4;
14. } end for
```

Figure 4.13.: Pseudocode of multiplication pipeline depth reduction algorithm.

One or both of the inputs of a multiplication operation are in CS arithmetic format. More specifically, the T12 operation template in Fig. 4.9b cannot handle the case in which both input operands are in CS format (edge weight *z=2*). Additionally, only the T11 template can be used for the initialization of a multiplication operation, which imposes the constraint that both multiplication inputs have to be in binary format in order to be initiated correctly.

Phase 2: Scheduling and Binding of the L-CDFG

After the generation of the L-CDFG, scheduling and binding are performed to enable an optimized mapping onto the RTL structure of the reconfigurable architectural template. Phase 2 accepts the L-CDFG along with the maximum number of available FPS and CS2Bin modules, which form the resource constraints of the underlying flexible architecture (Fig. 4.14).

Scheduling considers the available horizontal parallelism found in the application's behavioral specification. The resource-constrained minimum-latency scheduling problem was considered, since the architectural templates are composed by a fixed number of FPS-based computational elements. We used a mobility aware list-based scheduler [44]. Operations' mobility of the L-DFG (=ALAP-ASAP schedule values of the L-DFG nodes) guides the node selection during scheduling. The nodes that lay onto the L-DFG's critical path (nodes with zero mobility) receive the highest priority. The scheduler iterates through all BBs of the lowered CDFG graph and schedules each BB separately.

During the binding phase, the assignment of operations to flexible pipeline stages is performed. Due to the features of the underlying data-path, an efficient binding algorithm was developed in order to exploit the opportunities for fine-grained verti-

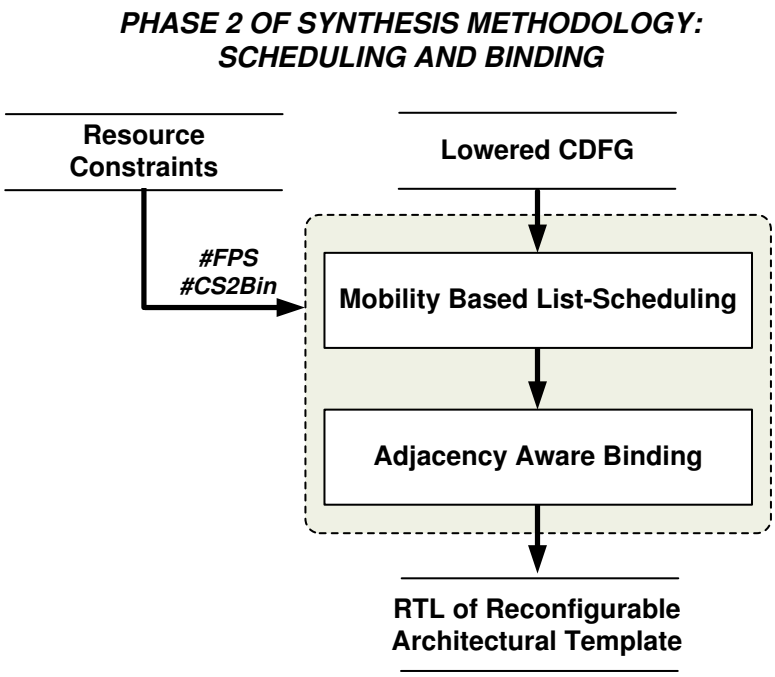


Figure 4.14.: Synthesis flow Phase 2: Scheduling and binding.

cal parallelism. Since the list-based scheduler does not take into account the inter-pipeline registers and the relative positions of the FPS units, suboptimal mappings are produced unless the binding procedure is tuned properly for the underlying architecture.

The proposed binding algorithm maps the nodes of the scheduled L-DFGs onto the FPS units of the architectural template, taking into account their adjacency and the data transfers denoted into the L-DFG by the data dependency edges. Its optimization criterion is to maximally exploit vertical parallelism by mapping data-dependent scheduled operations in adjacent flexible pipeline stages, in order to maximally utilize the inter-pipeline registers.

The pseudo-code of the binding algorithm is shown in Fig. 4.15. The inputs are the scheduled L-DFG and the number of the control steps ($\#sched\_steps$). The algorithm iterates through the control steps of the scheduled L-DFG (line 6). Two list structures are defined. Op_List_{step} stores the operations scheduled in the current control step (line 7), while Op_List_{step-1} stores the operations of the previous bound control step (line 18). The maximum size of these list structures is bounded by the resource constraints of the architecture specification. For each operation, its parent nodes are stored in $ParentOpList$ (line 10). If a parent node has been scheduled in the previous control step, then the $index$ of the FPS unit, where the parent node was bound, is retrieved and the operation in the current control step (op_i) is mapped to the right adjacent pipeline stage through $map(op_i, index{+}1)$ (lines 11-14). In case that none of the parent nodes is scheduled in the previous control step, the operation is mapped to the first available FPS unit (line 16).

```
1. Input: Sched_DFG;
2. Input: #sched_steps;
3. Output: Mapped DFG;
4.
5. Op_List_step-1 := NULL;
6. for (step ≤ #sched_steps) {
7.     Op_List_step := Get_Sched_Ops(step, Sched_DFG);
8.     while(Op_List_step non empty) {
9.         op_i := ExtractFirstOp(Op_List_step);
10.        ParentOpList := GetParentNodes(op_i);
11.        ParentOp_step-1 := GetOpMatch(ParentOpList, Op_List_step-1);
12.        if (ParentOp_step-1 non empty);
13.            index := GetIndex(ParentOp_step-1);
14.            map(op_i , index+1);
15.        else
16.            map (op_i , first available index);
17.    } end while
18. Op_List_step-1 := Op_List_step ;
19. } end for
```

Figure 4.15.: Pseudocode of adjacency aware binding algorithm.

The binding algorithm consists of two loop structures which iterate through all the nodes of the L-DFG graph. Assuming a L-DFG graph of size N, this iterative process is bounded to $O(N)$. In each iteration, the Op_List_{step-1} list is searched according to the number of elements of the $ParentOpList$, in order the $GetOpMatch$ function to return the first match. The size of $ParentOpList$ is constrained by the number of available inputs of every reconfigurable pipeline stage. Hence, in our case, if K denotes the number of available inputs then $K \leq 6$ (Fig. 4.9a). Additionally, Op_List_{step-1} is upper bounded by the number of available reconfigurable pipeline stages found in the architecture specification. Assuming M available reconfigurable pipeline stages, the overall worst case complexity of the algorithm is upper bounded by $O(K \times M \times N) \rightarrow O(M \times N)$, for $K = 6$.

4.6. Experimental Evaluation

In this section, we demonstrate the effectiveness of the proposed reconfigurable architectural template and introduced synthesis methodology.

In order to implement the proposed synthesis methodology, a semi-automatic tool-flow based on C++ and Perl scripts was developed (Fig. 4.16). The tool-flow integrates: (i) the academic C-to-RTL SPARK HLS framework [43], [110], (ii) several custom design passes (dark grey boxes in Fig. 4.16) and (iii) the commercial Synopsys Design Compiler [135] RTL-to-Standard Cell synthesis tool. The C code of the DSP kernel set is inserted into the SPARK HLS tool and the corresponding CDFG is extracted using the Intermediate Representation class of SPARK. During CDFG extraction standard compiler optimizations (i.e. copy propagation,

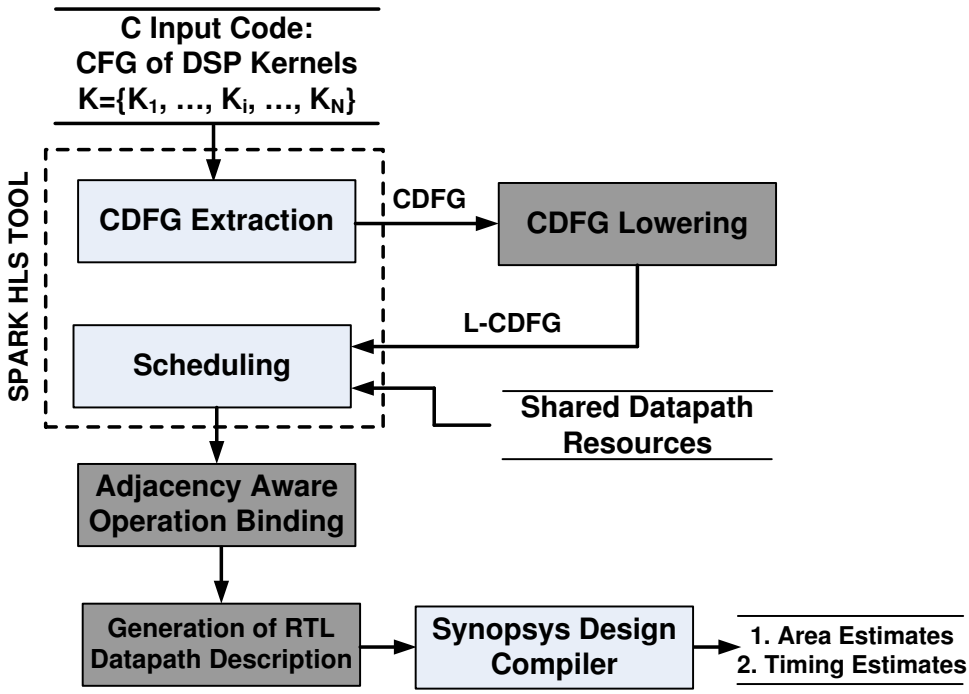


Figure 4.16.: The tool-flow implementing the proposed synthesis methodology.

dead code elimination etc) were considered. Further parallelizing code motions (i.e. loop unrolling, loop shifting/pipelining etc) were excluded to reduce the complexity of control structures. The lowering and pre-processing optimization procedure is implemented as a custom pass on the original CDFG. The L-CDFG is scheduled utilizing the SPARK's scheduling classes. Custom designed back-end passes read the scheduled L-CDFG, performs adjacency aware operation binding and generates a synthesized RTL description of the control-unit and the flexible datapath in Verilog Hardware Description Language (HDL). Functional verification is performed through Modelsim [136]. The RTL description of the flexible datapath is further synthesized using a TSMC 0.13 um standard cell technology library [107] and the Synopsys Design Compiler [135] (version 2006) tool-suite. The final netlist is checked for timing violations and the area and timing reports are generated.

Specifically, we performed the following seven sets of experimental comparisons.

1. **The first set of experiments (Section 4.6.2) qualitatively evaluates the proposed reconfigurable architecture in comparison to state-of-the-art coarse-grained reconfigurable architectures recently presented in the literature.**

2. **The second set of experiments (Section 4.6.3) provides quantitative comparisons of the proposed reconfigurable architecture presented in Section 4.3 in comparison to non-reconfigurable arithmetic circuits and other coarse-grained reconfigurable arithmetic architectures recently presented in the literature.**

For comparison reasons, the proposed architecture was coded in Verilog-HDL and was mapped onto a classical FPGA device. The functional verification of the proposed architecture was made using the ModelSim simulation environment [136]. Note that the proposed coarse-grained architecture does not target the FPGA's fine-grained implementations. It was mapped onto a FPGA device only for straightforward comparison with the other coarse-grained architectures which have been evaluated using the same FPGA device.

3. The next set of quantitative experiments (Section 4.6.4) is took place considering the ASIC design space. Alternative implementation scenarios based on the proposed architecture abstract model were explored in terms of the configurability's degree of each reconfigurable pipelined stage. The scalable behavior of the proposed architecture along with its hardware characteristics, in a quantitative manner, are reported. For this set of quantitative experiments, parameterized reconfigurable kernel templates have been synthesized using the Synopsys Design Compiler [106] and the TSMC 0.13 um technology library [107], following a standard cell design methodology.

4. In the fourth set of experimental data (Section 4.6.5), we evaluate the dominant components of the proposed architecture based on the implementation technology (FPGA or ASIC) and we explore different addition schemes that can be incorporated into the reconfigurable pipeline stages.

5. The fifth set performs a circuit level explorative comparison of the basic flexible units found in coarse-grained reconfigurable architectures (Section 4.6.6). From these comparisons the actual operating points of each architectural solution are extracted.

6. The sixth set of experiments (Section 4.6.7) considers the actual evaluation of the proposed synthesis methodology. A representative set of DSP kernels was formed. Each kernel was mapped and synthesized onto the FPS-based architecture. The synthesized kernels are evaluated in comparison with state-of-the-art reconfigurable datapath solutions.

7. The seventh set of experiments explores the scalability of the proposed solution. Scalability is evaluated considering both various resource allocation scenarios and operands' bit-widths.

4.6.1. Experimental Results: Brief Overview

In order to provide a quick overview of the evaluation process, we briefly review the major findings produced by analyzing the aforementioned experimental data. Specifically, qualitative comparisons show that the proposed architecture is able to handle a large set of mapped arithmetic behaviors without wasting computa-

tional resources. The proposed architecture delivers gains up to 46.9 % in area coverage, 296.5% in clock frequency, 25.2% in power dissipation and 32.75% in MOPS comparing to the reconfigurable MAC unit in [22]. In comparison to dedicated MACs, the proposed architecture delivers gains up to 218.6% in MOPS and 163.33% in energy efficiency (MOPS/mW). In addition, exploration results, altering the reconfigurability/flexibility incorporated into the proposed architecture, show an almost linear scaling behavior with regard to area complexity and power consumption.

Experimentation on a representative set of DSP benchmarks, show that the proposed methodology achieves significant gains in terms of execution time and area complexity compared to previously published coarse-grained reconfigurable datapaths. Specifically, the comparison with the row-based coarse-grained reconfigurable datapath in [15] have shown an average improvement of 33.9% and 23.5% in latency and area, respectively. Compared to reconfigurable architectures with aggressive chaining opportunities [16], gains of 28.4% in execution time and 53.9% in area complexity are reported. Assuming reconfigurable datapaths with horizontal/vertical parallelism and data forwarding features [17], our methodology delivers an average improvement up to 70% and 30.9% in latency and area, respectively. The Area-Delay product and area utilization metrics further prove the efficiency of our approach. Finally, the scalability of the proposed solution has been evaluated showing its advantageous features considering the execution latency and bit-width scaling.

4.6.2. Qualitative Comparisons

The proposed reconfigurable arithmetic unit (RAU) was qualitatively compared with state-of-the-art coarse-grained reconfigurable solutions. For the sake of judicial comparisons, in case of array-based reconfigurable architectures such as [137], [16], we considered only their basic reconfigurable cells and not the whole array of the architecture. The comparative results are reported in Table 4.4. The second column of Table 4.4 reports the operations that can be mapped onto each referenced architecture, while the third column shows the basic allocated resources which form each architecture. RAU based architectures enable the highest degree of different mapped operations among all the others reconfigurable architectures, except for the case of the reconfigurable cell in [16], which presents the highest degree among all. However, the cell in [16] allocates a large number of computational resources. The proposed architecture enables these functions with only one enhanced multiplier resource allocated, with significantly larger clock frequencies than the cell in [16]. The last column of Table 4.4 shows that the proposed architecture is the only one which manipulates both unsigned binary and carry-save formatted data (for fast chained Add/Sub computations) permitting also mixed computations between them.

Table 4.4.: ***Qualitative Comparison Between Coarse-Grain Reconfigurable Cells.***

Kernel	*Functions*	*Allocated Resources*	*Arithmetic Format*
RAU	Chained/Single Addition, Chained/Single Subtraction, Multiplication	Mux-Enhanced Multiplier	Carry Save, Unsigned Binary
Cell in [137]	Single ALU operation or Multiplication	One ALU unit and one Multiplier	Unsigned Binary
Cell in [16]	Chained Mult-ALU or Mult-Mult operations	Four ALUs and and Four Multipliers	Unsigned Binary
[138]	Single Addition or Multiplication	One Mux-Enhanced Multiplier	Unsigned Binary
[139]	Multiplication	One Decomposed Multiplier with input Duplication Network	Unsigned Binary, 2's complement
[22]	Single Addition or Multiplication	One Decomposed Multiplier/Adder with input Duplication Network	Signed Binary, 2's complement
[140]	Multiplication	Mux-Enhanced Multipliers Shifters, Adders	Unsigned Binary
[141]	Multiplication	One Partitioned Mux-Based Multiplier	2's complement

4.6.3. Quantitative Comparisons With Reconfigurable And Dedicated Architectures Mapped Onto FPGA Devices

The proposed architecture was mapped onto a Xilinx Virtex-II xc2v3000 device [142] and it was compared with (i) non-reconfigurable MAC units and multipliers and (ii) other reconfigurable architectures [22], [140], all implemented in the same Virtex-II FPGA device. The experimental procedure followed in all cases is based on the HDL-to-bitstream tool-flow provided by Xilinx [142]. We performed measurements which are provided in Table 4.5 and in Table 4.6. The measurements for our architecture are based on the real implementation data, after the completion of the place and route process onto the Virtex-II xc2v3000 FPGA device.

Table 4.5 reports comparison results among the instantiated reconfigurable kernel architecture (Section 4.3), the coarse-grained reconfigurable MAC unit presented in [22] and two versions of a non-reconfigurable MAC units. The first version of the non-reconfigurable MAC, MAC_{v1}, makes use of a CS array multiplier, which was mapped onto the programmable slices of the device. The second version of the non-reconfigurable MAC, MAC_{v2}, is based on the dedicated embedded multiplier, which is available into the Virtex-II FPGA device as a hard-macro. The addition component for both non-reconfigurable MACs is a 16-bit Carry-Look-Ahead adder [123].

Table 4.5.: ***Quantitative Comparison Between the Proposed Architecture, the Architecture in*** [22] **and Non-Reconfigurable MACs Mapped Onto Virtex-II FPGA**.

Arch.	Area (# slices)	Clock Freq. (MHz)	Power (mW/MHz)	No. of operands	Op. Density (#op./Area)	MOPS	MOPS/mW
Proposed	3187	55.5	7.63	20	6.27	333	0.79
Ref. [22]	6013	13.996	10.2	32	5.32	223.94	1.56
Gains	46.9%	296.5%	25.2%	-	17.8%	48.70%	-
MAC_{v1}	374	52.26	6.58	2	5.3	104.52	0.30
Gains	-	6.2%	-	900%	18.3%	218.6%	163.33%
MAC_{v2}	76	90	3.82	2	N/A	180	0.52
Gains	-	-	-	900%	N/A	85%	51.92%

In comparison with the reconfigurable MAC in [22], our architecture delivers better results in almost all the cases. The hardware complexity of the proposed architecture is 46.9% smaller in terms of number of slices. Also, it is able to operate with a clock frequency of 55.5 MHz, which outperforms the maximum frequency of the architecture presented in [22] in a size of magnitude close to ×4. Considering the Mega-Operations-Per-Second (MOPS) metric, the proposed architecture delivers a gain factor of 32.75%. The value of MOPS is defined by the number of operations multiplied by the clock frequency divided by the operation latency (=number of cycles), as in [22]. The power consumption in terms of mW/MHz is 25.2% lower. However, in terms of MOPS/mW the architecture in [22] has a better value. This is explained by the fact that the actual power of our architecture, in terms of mW, is much higher than the actual power of [22], since the proposed architecture runs at higher clock frequency and it is based in a high pipelined structure. We remind though, that the proposed architecture is coarse-grained reconfigurable and it targets to standard cell based implementations, which does not suffer from the high quiescent power dissipation like the Xilinx Virtex-II FPGA devices. Further information about the power consumption of the proposed architecture implemented in the with standard cell libraries is given in Subsections 4.6.4, 4.6.6, 4.6.7. We have also compared the number of input operands that the two architectures can operate on, for the case that the architecture in [22] is configured to perform 16-bit multiplications. The proposed architecture operates on twenty 16-bit formatted operands while the architecture of [22] is able to operate on thirty-two. However, in terms of operand's density, specified as $\#operands/Area$, the proposed architecture delivers a higher rate of 17.8%.

We also compared the proposed architecture with the two dedicated MACs. As expected, the non-reconfigurable architectures are more efficient in terms of area complexity than the proposed architecture. We have to mention that the 76 slices

Table 4.6.: ***Quantitative Comparison Between Reconfigurable And Dedicated Multiplication Units Mapped Onto Virtex-II FPGA***.

Reconf. Modules	Area (# slices)	Min. Clock Delay (ns)	Power (mW/MHz)	MOPS	MOPS/mW
16×16 RAU	1361	9.98	7.17	250.2	0.35
16×16 in [22]	2552	18.5	N/A	288	N/A
16×17 in [140]	729	28.7	13.4	69.7	0.15
16×16 CS-Mult	315	19	6.45	52.26	0.15

reported in the MAC_{v2} refers only to the logic mapped onto the reconfigurable part and do not include the area of the hard-macro multiplier. This is the reason why we excluded the *#operands/Area* result from Table 4.5, in case of MAC_{v2}. The proposed reconfigurable architecture delivers gains up to 18.3% in operands density, 218.6% in MOPS and 163.33% in MOPS/mW which makes it an energy efficient solution compared with the non-reconfigurable MAC_{v1}. Comparing to the MAC_{v2} implementation, our architecture provides 85% and 51.92% gains in case of MOPS and MOPS/mW metric, respectively.

We have also compared in Table 4.6 the RAU component of the kernel architecture against reconfigurable multiplication units presented in other approaches. The RAU, the 16×16 multiplication modules presented in [22] and in [140] and a 16×16 CS-array multiplier were mapped onto the Virtex-II xc2v3000 FPGA device. Given that RAU supports multiplication operations along with conventional and chained addition/subtraction ones, the MOPS value resulted as the mean value of the MOPS when the RAU is configured to perform multiplication and when the RAU is configured to perform chain addition or subtraction operations. RAU is able to operate approximately ×2 and ×3 faster that the multiplication units in [22] and in [140], respectively. Gains up to 256% in MOPS value and 133% in MOPS/mW are reported comparing to [140]. In general, RAU outperforms the other two reconfigurable solutions except for the area coverage compared with the multiplication unit in [140] and for the MOPS metric compared with [22]. However, the solution in [140] has been specially designed for Xilinx Virtex-II FPGA devices while the solution in [22] supports only bitwidth based reconfigurability. In comparison with the non-reconfigurable CS-array multiplier, RAU is able to operate ×2 faster and delivers gains up to 198% and 133% considering the MOPS and MOPS/mW metrics, respectively.

The results reported in Table 4.6 considered RAU as well as the other multiplication units, as stand-alone components mapped onto the Virtex-II FPGA. When RAU is measured as part of the overall reconfigurable kernel architecture (clock frequency 55.5MHz), the power consumption of RAU is 5.93 mW/MHz while the MOPS value of RAU becomes 138.75**.**

The aforementioned comparative data reports that the proposed architecture forms an efficient solution which combines the advantages of both the array-based re-

configurable architectures and the reconfigurable multiplier/MAC architectures. It offers fast implementation of arithmetic behaviors along with high reconfiguration capabilities at the operation level without consuming large portions of hardware computational resources, like the most of the array-based architectures. Also, it is not strictly application specific in one arithmetic behavior like the bitwidth-based reconfigurable multipliers/MACs. For that reasons, the proposed architecture can be a promising solution for applications which require high performance along with operation level flexibility, like those found in the DSP domain.

4.6.4. Reconfigurability Based Quantitative Exploration of the Proposed Architectural Templates in the ASIC Design Space

This set of experiments concerns the exploration of alterative implementation scenarios for the proposed architecture, considering the configurability degree of each FPS. The scaling behavior of the proposed architecture model among different configurability's degrees is reported and its hardware characteristics are evaluated quantitatively. Synthesis of the reconfigurable architecture according to the standard cell design methodology was performed. Eight different architectural templates of the proposed architecture varying in their dynamical reconfigurable characteristics were modeled, synthesized and evaluated. The architectural templates were modeled using the Verilog hardware description language and were synthesized using the Synopsys Design Compiler [106] with the TSMC 0.13 um ASIC CMOS technology library [107].

We adopted a hierarchical design procedure, starting from the bottom hierarchy level and combining structurally the Verilog modules towards the top level hierarchies. All the templates were synthesized under the timing constraint of 300 MHz clock frequency and mapped onto the standard cells provided by the TSMC 0.13 um library. We consider typical operating conditions and automatic wire load model selection, except for the case of the RAU's input multiplexors (the Interconnection Network in Fig. 4.4) where the aggressive wire model parameter was set to optimize the imposed delays. For further optimization of the global interconnection scheme, the top level design module was flattened during synthesis.

The eight templates of the Reconfigurable Kernel Architecture derived by changing the reconfiguration capability of each pipeline stage found in RAU. Each pipeline stage can be configured either as a complex reconfigurable (CR) like the FPS 0 in Fig. 4.5 or as a single reconfigurable (SR) like the FPS 1 in Fig. 4.5, or as a non-reconfigurable (NR) like the last FPS of the RAU. Given the stable interconnection scheme inside the RAU's pipeline stages, the area overheads of the SR stages over the NR stages are imposed only from the extra allocated hardware of the UC cells, in respect to the conventional multiplier's cells. The area overheads of the CR stages

Table 4.7.: ***Characterization of the Architectural Templates.***

CR: 1 chained add/sub, or 2 smaller chained adds/subs in parallel, or 16×4 mult.
SR: 1 chained add/sub, or 16×4 mult.
NR: 16×4 mult.

Architecture Template	*Pipeline Stage 0*	*Pipeline Stage 1*	*Pipeline Stage 2*	*Pipeline Stage 3*	***Remanence***
ARCH0	CR	CR	CR	NR	1.5
ARCH1	CR	CR	SR	NR	1.25
ARCH2	CR	SR	SR	NR	1.0
ARCH3	CR	CR	NR	NR	1.0
ARCH4	SR	SR	SR	NR	1.0
ARCH5	CR	SR	NR	NR	0.75
ARCH6	CR	NR	NR	NR	0.5
ARCH7	SR	NR	NR	NR	0.25

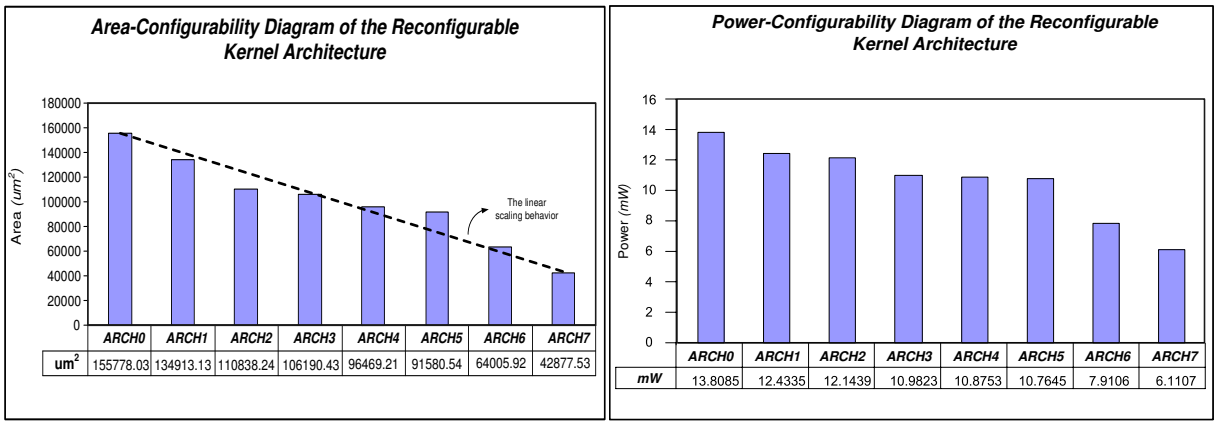
over the SR stages are imposed by the intermediate multiplexer line and the higher wiring due to the extra two output ports allocated.

The overall configurability of the RAU has a great impact on the whole architecture and especially on the number and the width of the allocated steering logic. Table 4.7 shows the eight architectural templates (*ARCH0-ARCH7*) characterized by the RAU's reconfiguration capability. The templates have been sorted in a descending order based on their Remanence metric [143]. The remanence characterizes the dynamical character of the reconfigurable architecture. The higher the remanence's value is, the more configurable the underlying hardware architecture becomes. For a reconfigurable architecture composed of N_{PE} reconfigurable processing elements, running at the clock frequency F_e and configuring N_c processing units at each reconfiguration cycle of frequency F_c, the Remanence is defined by the following formula:

$$Remanence = \frac{N_{PE} \cdot F_e}{N_c \cdot F_c} \tag{4.6}$$

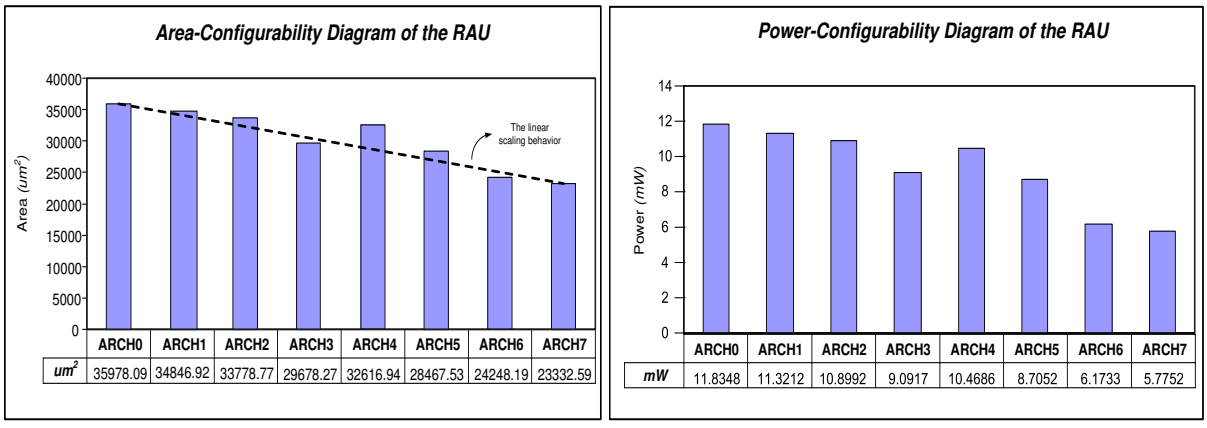
For all the architectural templates $N_c = 4$ and $F_e = F_c = 300MHz$. We include the last non-reconfigurable pipeline stage in the N_c value given that it can be configured either as CR or SR pipeline stage too. The N_{PE} depends on the configuration of each pipeline stage and it is different in each architectural template. *ARCH2* is the instantiated architectural template which we analyzed in Section 4.3.

In Fig. 4.17a the hardware area (um^2) of the synthesized architectural templates is depicted. In Fig. 4.17b the power consumption (mW) for each template is reported. The tables under the diagrams shows the active area/power estimates



(a) (b)

Figure 4.17.: The Reconfigurable Kernel's Architecture a) Area-Configurability Diagram, b) Power-Configurability Diagram.



(a) (b)

Figure 4.18.: The RAU's a) Area-Configurability Diagram, b) The Power-Configurability Diagram.

for each architectural template. As expected, the area scales down moving towards the architectural templates with smaller remanence values. The occupied hardware area is affected mainly by the input and output ports of the RAU which contribute to more feedback wires and to larger input selection multiplexors. Fig. 4.17a depicts that, the area complexity of the reconfigurable architecture scales almost linear with respect to the incorporated reconfigurability. The same scaling behavior is observed in the power consumption diagram as well. This feature of linear scaling gives the advantage to the designer to select among different architectural solutions without worrying about unexpected area/power overheads. However, it has to be noted that the number of FPSs into the RAU has been restricted to four. For RAUs with more than four flexible pipeline stages, the area/power scaling factor is expected to approximate an exponential growth as the number of RAU's available pipelines augments.

Driven by the fact that the RAU is the dominant computational component in the proposed architectural template, we provide the Area-Configurability and the Power-Configurability diagrams for the different RAU implementations (Fig. 4.18). As in the case of the overall reconfigurable architecture, the area coverage of RAU scales linearly among the different RAU templates. The gradient though in Fig. 4.18a is much lower than in Fig. 4.17a, since the measurements concerns only the RAU component and they did not include the area of the input multiplexors.

4.6.5. Evaluation Of Implementation Targets And Design Alternatives

In Subsections 4.6.3 and 4.6.4, the proposed reconfigurable architecture was mapped onto two different implementation technologies, namely FPGA and ASIC (standard cell library), respectively. In this subsection, the performance differences between the two implementations are quantitatively evaluated. We experimentally prove our previous statement that the proposed architecture targets the ASIC technology rather the FPGA's one. Based on this result, we further investigate, in the ASIC domain, alternative chain-addition schemes for the internal structure of the reconfigurable pipeline stages.

In this set of experiments we synthesized separately (i) the single reconfigurable (SR) pipeline stage and (ii) the multiplexer based interconnection network (Fig. 4.4). The two components were coded in technology independent Verilog-HDL using structural RTL coding style. The HDL descriptions were mapped onto (i) the Xilinx Virtex-II xc2v3000 FPGA device [142] (fabrication technology parameters: 0.15 um 8-Layer Metal process with 0.12 um high-speed transistors) and (ii) onto the standard cells provided by the TSMC 0.13 um ASIC library [107].

Tables 4.8 and 4.9 report the comparative results of the two implementation tech-

Table 4.8.: ***Evaluation Of The RAU Mapped Onto FPGA and ASIC Libraries.***

Target Technology	Optimization Criterion	Actual Area	Equiv. Gate Count	Critical Path (ns)	Power (mW/MHz)
2*Virtex-II FPGA	Speed	201 slices	2925	11.1	3.66
	Area	201 slices	2925	11.1	3.66
TSMC 0.13 um	Speed-Area	13486.72 um^2	1134.29	2.0	0.025

nologies. The second column of Tables 4.8 and 4.9 refers to the optimization criterion which guided the synthesis procedure. While Synopsys Design Compiler [106] enables synthesis taking into account both optimization criterions simultaneously, Xilinx tool-flow permits only one optimization criterion to guide the synthesis procedure. We provide FPGA results separately for speed constrained synthesis and area constrained synthesis. The equivalent gate count metric is provided for a straightforward comparison of the area efficiency between the two implementation targets.

At first, some interesting results are reported comparing only the two FPGA implementations. The area, critical path and power dissipation of the SR pipeline stage (Table 4.8) are the same for both optimization criterions. The UCs' in every pipeline stage have been designed at the gate level of abstraction and the optimization criterion does not affect the final netlist. In case of the multiplexer-based interconnection network (Table 4.9) the area optimized FPGA design delivers smaller area, critical path delay and power dissipation than the speed optimized. Although that the actual area (slices) of the FPGA device is smaller in the area optimized netlist of the interconnection network, the equivalent gate count has a larger value than the speed optimized one. The area optimization criterion enables more compact placement and routing onto the FPGA device than the speed optimization criterion. More compact placement and routing deliver smaller routing tracks which are the reason that the critical path delay is smaller in the area optimized than in the speed optimized implementation.

The ASIC implementations outperform the FPGA ones in all cases, as expected. The ASIC implementation of an SR FPS delivers $\times$1.58 gain in area complexity, $\times$4.55 gain in critical delay and $\times$145 gain in power dissipation (Table 4.8), when it is compared with the FPGA one. Additionally, the ASIC implementation of the multiplexer based interconnection network (Table 4.9) delivers $\times$3.75 gain in area coverage, $\times$4.7 gain in critical path's delay and $\times$93 gain in power consumption, in comparison to the best FPGA implementation (area optimized). As stated in Subsection 4.6.3, FPGA implementation was considered only for straightforward comparisons with previously published reconfigurable arithmetic architectures.

Table 4.10 reports exploration results, based on various chain-addition schemes, for the internal structure of a SR pipeline stage. Specifically, we consider the (i)

Table 4.9.: ***Evaluation Of The Multiplexer-Based Interconnection Network Mapped Onto FPGA and ASIC Libraries.***

Target Technology	Optimization Criterion	Actual Area	Equiv. Gate Count	Critical Path (ns)	Power (mW/MHz)
2*Virtex-II FPGA	Speed	1220 slices	17400	3.08	1.02
	Area	1206 slices	17820	2.85	0.94
TSMC 0.13 um	Speed-Area	43474.64 um^2	3657	0.5	0.01

Table 4.10.: ***Comparison Between Different Chain Addition Schemes For The Intra-Pipeline Stage Structure.***

Addition Schemes	Area (um^2)	Critical Path (ns)	Power (mW/MHz)
Chained Carry Save (UC_{OPT})	13486.72	2.0	0.025
Chained Carry Save (UC_{SUBOPT})	13942.72	2.0	0.030
Chained Carry Lookahead	15283.38	2.9	0.049
Chained Manchester	13815.12	4.0	0.056
Chained Ripple Carry	14038.32	4.4	0.066

Carry-Save, (ii) Carry Look-Ahead, (iii) Manchester Carry-Chain and (iv) Carry Ripple single addition schemes [123]. Based on these schemes, chained adders with four single addition units were formed and they were enhanced in order to execute multiplication and chained subtraction as well. Thus, various SR FPSs were generated based on the different chain-addition schemes and were synthesized using the Synopsys Design Compiler [106] and the standard cell ASIC technology library TSMC 0.13 um [107]. Specifically, we iterate the synthesis procedure for each of the aforementioned SR schemes, with a timing constraint interval of 0.1 ns, until no timing violations were reported.

We considered five variants of the RAU's SR pipeline stage, reported in the first column of Table 4.10. The two Carry Save variants refer to the same internal structure altering the UC implementation strategy (Table 4.1). The UC_{OPT} based implementation delivers smaller area and power dissipation than the UC_{SUBOPT} implementation. In general, the proposed UC_{OPT} based Carry Save chained addition delivers the smallest critical path's delay and power consumption among all other solutions. It is up to 0.9 ns faster, it dissipates approximately 50% less power and it occupies 11.75% less area than the chained Carry Look-Ahead scheme. Also, the proposed chained addition scheme occupies less area and is approximately $\times 2$ faster and power efficient than the chained Manchester implementation. Comparing with the Carry Ripple based implementation, the proposed CS chained addition scheme delivers even higher gains in critical delay and power dissipation.

4.6.6. Quantitative Comparison of Flexible Computational Units in State-of-The-Art Coarse-Grained Reconfigurable Architectures

A quantitative area-timing exploration has been conducted between the three flexible computational components: (i) the FPS unit (SR type), (ii) the CGRC of Fig. 4.10a, which forms the basic computational unit of the reconfigurable slice architecture of CRISP [15] and (iii) the FCC unit in [16]. A qualitative comparison between these flexible cells has been presented in Subsection 4.6.2. We have also included results for the case of a conventional 16-bit multiplication unit (CS-Mul), providing straightforward comparison with a non-flexible component. All the components were implemented using a TSMC 0.13 um standard cell technology library [107] and Synopsys Design Compiler [135]. The Carry-Save "CSA" architecture from the Synopsys Design Ware IP library [135] was considered for both the conventional multiplication unit and the multiplication units found in the CGRC [15] and the FCC [16]. The usage of "CSA" provides judicious comparisons since the proposed flexible datapath is based on the structure of CS array multiplier [53].

An iterative synthesis procedure was used to expose the lower limits of the area-timing values for the four aforementioned units. The delay constraint was altered in each iteration considering a time interval of 0.1 ns, with an initial value of 1.0 ns. For each component we collect ten operating points starting from its first violation free version. Fig. 4.19 reports the comparative results. The FPS is able to operate, without timing violations, in a time frame between [1.5ns, 2.5ns]. Accordingly, the operating points of CGRC [15] lay between [3.4ns, 4.4ns], while FCC unit [16] operates without timing violations in a time frame of [7.0ns, 8.0ns]. The CS-Mul unit has a violation free timing range of [3.3ns, 4.3ns] with similar hardware characteristics to the CGRC.

As expected, the FPS unit delivers the smaller area and the highest operating frequency among all. It has a larger operation range of about $\Delta T_{FPSvsGCRC} = 1.9ns$ than the CGRC [15]. In comparison with the FCC unit [16], the FPS has a $\Delta T_{FPSvsFCC} = 5.5ns$ and it occupies about $\times 6.5$ smaller area. The comparison between the CGRC unit with the FCC unit delivers a $\Delta T_{GCRCvsFCC} = 3.6ns$. In [16], the authors report that the delay constraint/relation $T_{FCC} < 2.3 \times T_{CGRC}$ has to be satisfied in order to the FCC-based datapaths to have better performance than the CGRC-based datapaths. Considering the operating ranges of the FCC and the CGRC unit in Fig. 4.19, the aforementioned delay relation is satisfied for each operation point.

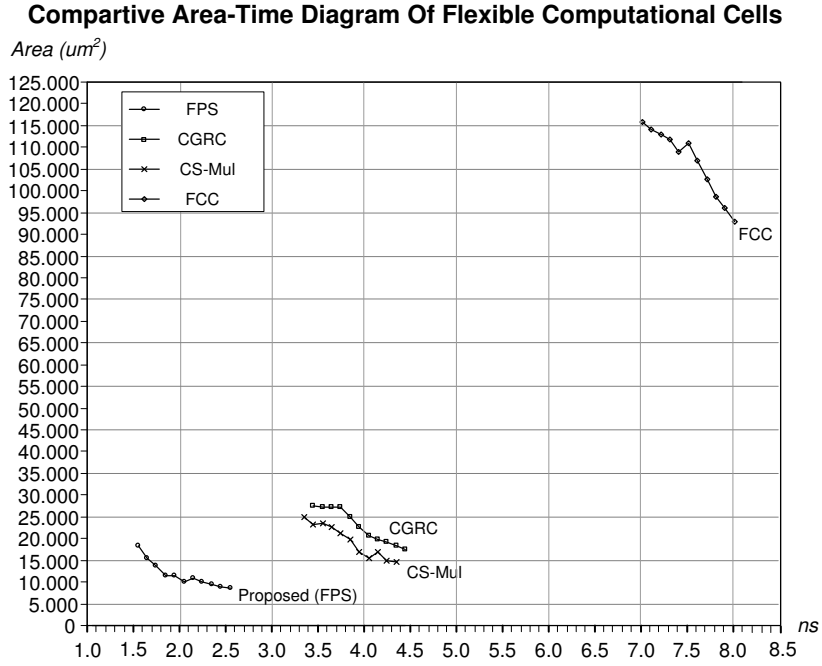


Figure 4.19.: Area-Time exploration diagram for the flexible computational units.

4.6.7. Performance Improvements

We evaluate the efficiency of our synthesis methodology based on a set of benchmarks found into computationally intensive DSP applications. The benchmark suite consists of 13 kernels: (i) 7th-order FIR filter (FIR7) [144], (ii) 7th-order IIR filter (IIR7) [144], (iii) Lattice filter (Lattice) [144], (iv) 5th-order Elliptic filter (Elliptic) [144], (v) Wavelet transformation [144], (vi) 7th-order Wave Digital Filter (WDF7) [144], (vii) 3th-order non-linear IIR Volterra filter (Volterra), (viii) 16th-order FIR filter (FIR16), (ix) a one-dimensional Discrete Cosine Transformation (1D DCT), (x) a 16-point Fast Fourier Transform (FFT16) [145], (xi) Horizontal Gauss Blur image transformation (HGB) found into Cavity Detector algorithms, (xii) 2D DCT kernel of JPEG application (Jpeg DCT) [146] and (xiii) 2D Inverse DCT kernel of MPEG application (Mpeg IDCT) [146]. The aforementioned DSP kernels were described in C. The Volterra and FIR16 kernels are based on custom C implementations. The 1D DCT kernel is an unrolled implementation (unrolling the innermost loop) found in [147], where the input image is inserted in a pipelined manner row-by-row. The HGB kernel has been unrolled for the two innermost loops. The sine and cosine coefficients in FFT16 kernel [145] were up-scaled with a 2^{12} scaling factor, since our architecture handles only integer data.

The DSP kernels were scheduled, mapped and synthesized onto four flexible datapath architectures. All the datapaths have similar characteristics considering the horizontal parallelism degree. Specifically, we considered the following datapath

Table 4.11.: ***Comparative Latency Results.***

		Latency (ns) of Flexible Datapaths				***Latency Gains (%)***		
DSP Kern.	DFG Nodes	CRISP [15]	FCC [16]	NISC [17]	Proposed FPS	Proposed vs CRISP	Proposed vs FCC	Proposed vs NISC
FIR7	14	27.3	30.0	133.0	16.0	41.4	46.6	87.9
IIR7	18	27.3	30.0	174.8	16.0	41.4	46.6	90.8
Lattice	23	39.0	37.5	254.6	32.0	17.9	14.6	87.4
Elliptic	39	54.6	45.0	395.2	24.0	56.0	46.6	93.9
Wavelet	82	58.5	52.5	406.6	30.0	48.7	42.3	92.6
WDF7	57	50.7	45.0	364.8	44.0	13.2	2.2	87.9
Volterra	36	54.6	52.5	197.6	42.0	23.1	20.0	78.7
FIR16	32	62.4	60.0	235.6	32.0	48.7	46.6	86.4
1D DCT	17	1996.8	1440.0	1292.0	1042.0	47.8	27.6	19.3
				(714.4)				*(-45.8)*
FFT16	85	105.3	150.0	524.4	70.0	33.5	53.3	86.6
HGB	49	304.2	270.0	763.8	252.0	17.2	6.6	67.0
				(395.2)				*(36.2)*
Jpeg DCT	199	1287.0	1027.5	1109.6	868.0	32.6	15.5	21.7
				(904.4)				*(4.0)*
Mpeg IDCT	169	1193.4	975.0	1094.4	968.0	18.9	0.7	11.5
				(573.8)				*(-68.7)*
Average	-	-	-	-	-	33.9	28.4	70.1
								(55.2)

Table 4.12.: ***Datapath Area Results: CRISP Slice [15] vs FCC-Based [16] vs NISC-Based [17] vs FPS-Based.***

	Active Area (um^2)				***Area Gains (%)***		
DFG	CRISP [15]	FCC [16]	NISC	Proposed	Proposed vs CRISP	Proposed vs FCC	Proposed vs NISC
FIR7	143259.67	239100.68	101308.22	99858.21	30.3	58.2	1.4
IIR7	138972.23	265092.78	83586.48	78809.96	43.3	70.3	5.7
Lattice	142220.26	234649.81	173597.81	68062.46	52.1	70.9	60.8
Elliptic	156536.96	293476.53	232246.83	87457.89	44.1	70.2	62.3
Wavelet	160993.09	283160.06	364021.54	158973.42	1.3	43.9	56.32
WDF7	156489.87	283434.28	189447.48	135691.93	13.3	51.2	28.4
Volterra	148120.42	278643.28	118066.88	116323.72	21.5	58.3	1.5
FIR16	156475.56	258984.35	221589.56	149685.06	4.3	42.2	32.4
1D DCT	164758.46	272743.09	201209.17	114735.69	30.4	57.9	42.9
			(294178.29)				*(60.9)*
FFT16	225466.76	324843.68	287772.74	166273.71	26.6	48.8	42.2
HGB	234554.9	358189.52	164857.46	196534.15	16.2	45.1	-19.2
			(153408.21)				*(-28.1)*
Jpeg DCT	245573.2	370835.6	327503.83	217593.6	11.4	41.3	33.6
			(321386.95)				*(32.3)*
Mpeg IDCT	234981.3	369906.2	424242.54	210836.4	10.3	43.0	50.3
			(287449.59)				*(26.6)*
Average	-	-	-	-	23.5	53.9	30.9
							(29.4)

solutions: (i) the proposed flexible datapath which comprise four FPS components (SR type) and one CS2Bin module, (ii) a CRISP-like [15] coarse-grained reconfigurable datapath comprising four CGRC units (Fig. 4.10a), (iii) a FCC-based [16] reconfigurable datapath comprising two FCC nodes (Fig. 4.10b) with aggressive operation chaining and (iv) the NISC architecture [17] with 4 ALUs and 1 multiplier unit combining horizontal and vertical parallelism together with data forwarding features. Actually, for the CRISP-like case we consider the reconfigurable datapath of one flexible slice. In case of the FPS-based datapath solution the tool-flow illustrated in Fig. 4.16 was used in order to schedule and map each of the benchmarks' DSP kernel. For both the CRISP-like and the FCC-based datapaths, the SPARK HLS tool [43] with the default optimization passes was used for scheduling the kernels' DFGs. In case of FCC-based datapaths, SPARK was configured to perform intra-template chaining. Inter-template chaining between the FCCs was also considered by manually improving the resulting FCC datapaths. For the NISC datapath solution the online NISC tool-set [148] was considered for the mapping of the DSP kernels along with the corresponding datapath generator [149]. For each kernel mapped in a datapath solution, the maximum memory bandwidth has been allocated, between the local storage and the processing elements. The memory bandwidth requirements depend on the architecture of the processing elements of each datapath. The CRISP, FCC-based and the proposed datapath have been synthesized considering scratch registers and for each kernel the maximum memory bandwidth has been allocated. Specifically: (i) CRISP datapath has 8 read ports and 4 write ports, (ii) FCC-based datapath has 12 read ports and 8 write ports and (iii) the FPS-based datapath has 12 read ports and 5 write ports. The NISC datapath considers the allocation of register file rather than scratch registers. The register file type that delivers the maximum allowable memory bandwidth has been selected for NISC implementations (RF 16×8 register file type [148]).

From Fig. 4.19, we extracted the operating points of each reconfigurable datapath architecture as the middle value between the lower and the upper operation points reported for each flexible computational unit. Thus, $T_{FPS} = 2ns$, $T_{CRISP} = 3.9ns$, $T_{FCC} = 7.5ns$ and $T_{NISC} = T_{CS-MUL} = 3.8ns$ are the clock cycle periods of the FPS-, CRISP-, FCC- and NISC-based solutions, respectively.

Table 4.11 reports the DFG's size of each DSP kernel and the execution latency in *ns* for each DSP kernel mapped onto the four flexible datapaths. In case of the proposed datapath, the latency measurements include also the overheads imposed by the CS2Bin conversion. Execution latency is defined as the product of the total number of cycles multiplied by the clock cycle period of each datapath ($\#Cycles \times T_{DatapathSolution}$). For the image kernels (1D DCT, HGB, Jpeg DCT, Mpeg IDCT) the execution latency was measured for an image processing of 8×8 pixels. The last three columns of Table 4.11 report the latency gains of the FPS-based solution over the CRISP, the FCC-based and the NISC-based datapaths. Columns 4 and 8 in Table 4.11 provide two different latency values for the benchmarks with loop structures (1D DCT, HGB, Jpeg DCT and Mpeg IDCT). The reason behind this

is that NISC compiler uses parallelizing code motions (specifically loop unrolling combined with loop pipelining) during NISC datapath synthesis. Since, our toolflow does not support parallelizing code motions, we provide NISC mappings with both the parallelizing code motion disabled and enabled to provide a holistic view. The values enclosed in brackets refer to the NISC mapping with parallelizing code motions.

The FPS approach delivers lower execution latency (thus higher performance) than the CRISP and FCC-based flexible datapaths in all cases. Specifically, the FPS-based datapath delivers an average reduction in execution latency of 33.9% in comparison to CRISP-like datapaths. The corresponding gain with FCC-based datapaths is 28.4%. For a benchmark dominated by several alu-mul and mul-mul operation templates i.e. Lattice, WDF7 and Mpeg IDCT, the FPS-based approach delivers lower execution latency gains, since such operation templates force the insertion of many CS2Bin arithmetic conversion instructions, which delay the overall execution by augmenting the schedule length of the L-DFG. As expected, the many mul-mul operation templates favor mainly the FCC-based solutions. However, even in these worst application scenarios, the proposed synthesis methodology delivers faster implementations of about 5.8% compared with the FCC datapath. Two different sets of comparison results between FPS-based and NISC solution are included. In case that the NISC parallelizing code motions are disabled, thus the NISC's mapping efficiency depends heavily on the datapath's architecture, the FPS-based solution delivers faster schedules in all cases, with an average latency gain of 70.1%. In case that the code motions are enabled, thus NISC's mapping efficiency depends heavily on the compiler, the FPS-based solution still delivers faster schedules for the majority of the DSP kernels. Specifically, the FPS-based solution delivers slower schedules only for the 1D DCT and Mpeg IDCT kernels in comparison with the NISC datapath. These kernels seem to be extremely benefited from the parallelizing code motions. However, the average latency gain of the FPS-based compared to the fully optimized NISC solutions remains high (up to 55.2%).

Table 4.12 reports the active post-synthesis area estimates (control logic-storage-routing-functional units) for each DSP kernel. Each architecture was synthesized under the timing constraints reported in the legend line of Table 4.12. The RTL code of NISC architectures was manually optimized (i) by allocating only the minimum number of registers for each NISC instance and (ii) by changing the NISC's bitwidth from 32- to 16-bit length. These optimizations guarantee the fair comparisons of the flexible datapaths. Our methodology forms the most efficient area solution in all cases compared with the CRISP and FCC-based datapaths. In comparison to NISC datapaths, the FPS-based solution is more efficient in all cases except the HGB kernel. Specifically, it delivers average area reductions of 23.5%, 53.9%, 30.9% (29.4% when parallelizing code motions are enabled) compared with the CRISP, FCC-based and NISC architectures, respectively.

Additionally, we report the values of the Area-Delay-Product (ADP) as an effi-

ciency's metric of our approach. In Fig. 4.20a the ADP of the FPS-based versus the CRISP-like datapath solutions is illustrated. Fig. 4.20b shows the ADP of the FPS-based over the FCC-based solutions, while Fig. 4.20c depicts the FPS-based versus the NISC-based datapaths (the optimized NISC ADP values were considered). The ADP values have been normalized according to the respective ADP values of each benchmark for the CRISP, the FCC and NISC cases, respectively. The dashed line (ADP = 1) represents the ADP of either the CRISP or the FCC or the NISC case. In comparison with the CRISP and FCC solutions, the FPS-based datapath is more efficient in all cases, since FPS-based solutions delivers both faster (lower execution latency) and smaller (lower area coverage) implementations than the other two approaches. In comparison with the CRISP datapaths (Fig. 4.20a), the ADP ranges between 24.5% (Elliptic) and 82% (HGB). In comparison with the FCC datapaths (Fig. 4.20b), the ADP ranges between 15.8% (IIR7 and Elliptic) and 56.5% (Mpeg IDCT). In comparison with NISC architectures, the proposed solution delivers significant ADP gains for a large class of kernels (FIR7, IIR7, Lattice, Elliptic, Wavelet, WDF7, Volterra, FIR16, FFT16) ranging between 8.6% (IIR7) to 20.9% (Volterra). In kernels with loop structures (1D DCT, HGB, Jpeg DCT), the FPS-based datapath delivers ADP values ranging from 55.9% (1D DCT) to 81.6% (HGB) in comparison with the code motion optimized NISC architectures. For the case of Mpeg IDCT, the FPS-based solution is less efficient than NISC, since the specific kernel is highly affected by the parallelizing code motions enabled into NISC compiler. In case that the parallelizing code motions are disabled for the Mpeg IDCT, the corresponding ADP value becomes much lower (specifically 43.9%).

Fig. 4.20d depicts the area distribution (area of FPS units, storage area and control/steering-logic) inside each FPS-based kernel implementation. The area portion of the CS2Bin module is very low and has not been included in Fig. 4.20d. It can be seen that for kernels with low code complexity the FPS area is the dominant area component, while for kernels with large code complexity the storage area is the dominant one. Additionally, the increasing complexity of DSP kernels has a direct effect on the steering logic of the datapath, since for the larger kernels (Jpeg DCT and Mpeg IDCT) the area portion of the steering logic is almost equal to the area portion occupied by the FPS units (smaller than 30% in all cases though). However, the impact of increasing code complexity on the timing of the datapath is much smaller than the area case, since all kernels were successfully synthesized under the timing constraint of T_{FPS} = 2 ns.

In the proposed approach, the highly pipelined architecture and the CS arithmetic require a larger number of storage elements (registers) to be allocated than the binary arithmetic architectures. Fig. 4.20e illustrates the storage requirements per DSP kernel for the four flexible datapaths. FPS-based solutions allocate larger area for storage than CRISP and FCC-based datapaths. This is a direct effect of CS arithmetic format used inside the FPS-based datapaths. Nevertheless, the storage area overhead is much smaller than the area gain of the whole datapath due to both the much lower combinational logic required by each FPS unit and

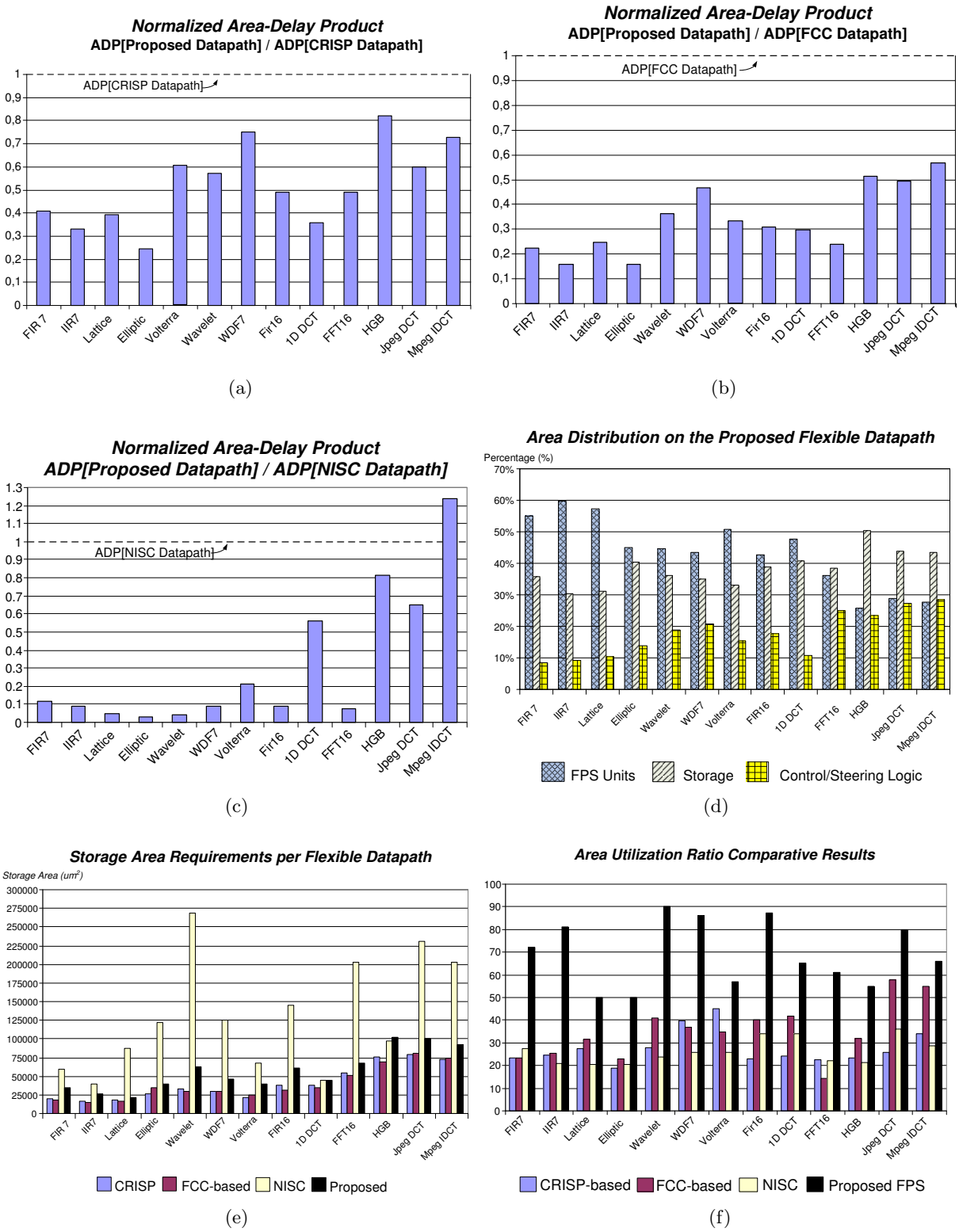


Figure 4.20.: Post synthesis evaluation results. a) Norm. ADP: Proposed vs CRISP [15], b) Norm. ADP: Proposed vs FCC [16], c) Norm. ADP: Proposed vs NISC [17], d) Area distribution of the proposed datapaths, e) Storage requirements per flexible architecture, f) Area Utilization Ratio comparative results.

the high reuse of FPS units which are able to execute a large number of different operation templates. Comparing the FPS-based datapath with NISC, we inferred that the proposed delivers significantly lower storage area requirements. This is a direct effect of using scratch registers rather than register files which are used in NISC datapaths. However, NISC architectures target to the FPGA implementations in which embedded block RAMs are available to map the register file structures.

In order to quantitatively evaluate the reuse of hardware area, the Area Utilization Ratio (AUR) has been estimated (Equation 1 in Subsection 4.5.2), taking into account the area of datapath components (functional units, pipeline registers and steering circuitry). Fig. 4.20f illustrates the AUR values of each DSP kernel for the four examined flexible datapath solutions. The FPS-based solution outperforms the other three, achieving an average AUR value of 69.3%, ranging between 55% - 90%. The average AUR is 27.6%, 35.1% and 26.3% for the CRISP-based, FCC-based and NISC datapaths, respectively. Specifically, the proposed solution delivers average AUR gains of 162.5% over the CRISP, of 119.1% over the FCC datapaths and of 169.7% over NISC, respectively.

4.6.8. Scalability Study of the Proposed Solution

This subsection studies the scalability of the proposed approach. Firstly, we evaluate the performance improvements of each DSP kernel when the number of allocated FPS units is incremented. In this set of experiments, we define scalability as the gradient of the *Latency vs. #Resources* curve. Secondly, we evaluate the hardware area and timing characteristics of the FPS-based datapath for various operand's bit-widths.

Considering the first set of scalability experiments, each DSP kernel was scheduled in an iterative manner by incrementing the available computational resources using the tool-flow of Fig. 4.16. The same iterative procedure was also applied to the CRISP-like datapaths [15]. The serial execution of each kernel was considered as the lower bound of the iterative process, by allocating only one computational resource (FPS unit or CGRC unit). Nine computational resources formed the upper bound of the exploration. The execution latency of each kernel was estimated taking into account the clock cycle periods of Table 4.11. The C input code of the DSP kernels remained the same during the scalability exploration and no further parallelizing code transformations [133] were considered.

Fig. 4.21 shows the *Latency vs. #Resources* diagrams extracted from the aforementioned iterative procedure. The zero gradient of the *Latency vs. #Resources* curve means that the number of available computational resources exceeds the available operation level parallelism found into the DSP kernel. The scalability features of FIR7 and IIR7 kernels are illustrated together because they presented the same

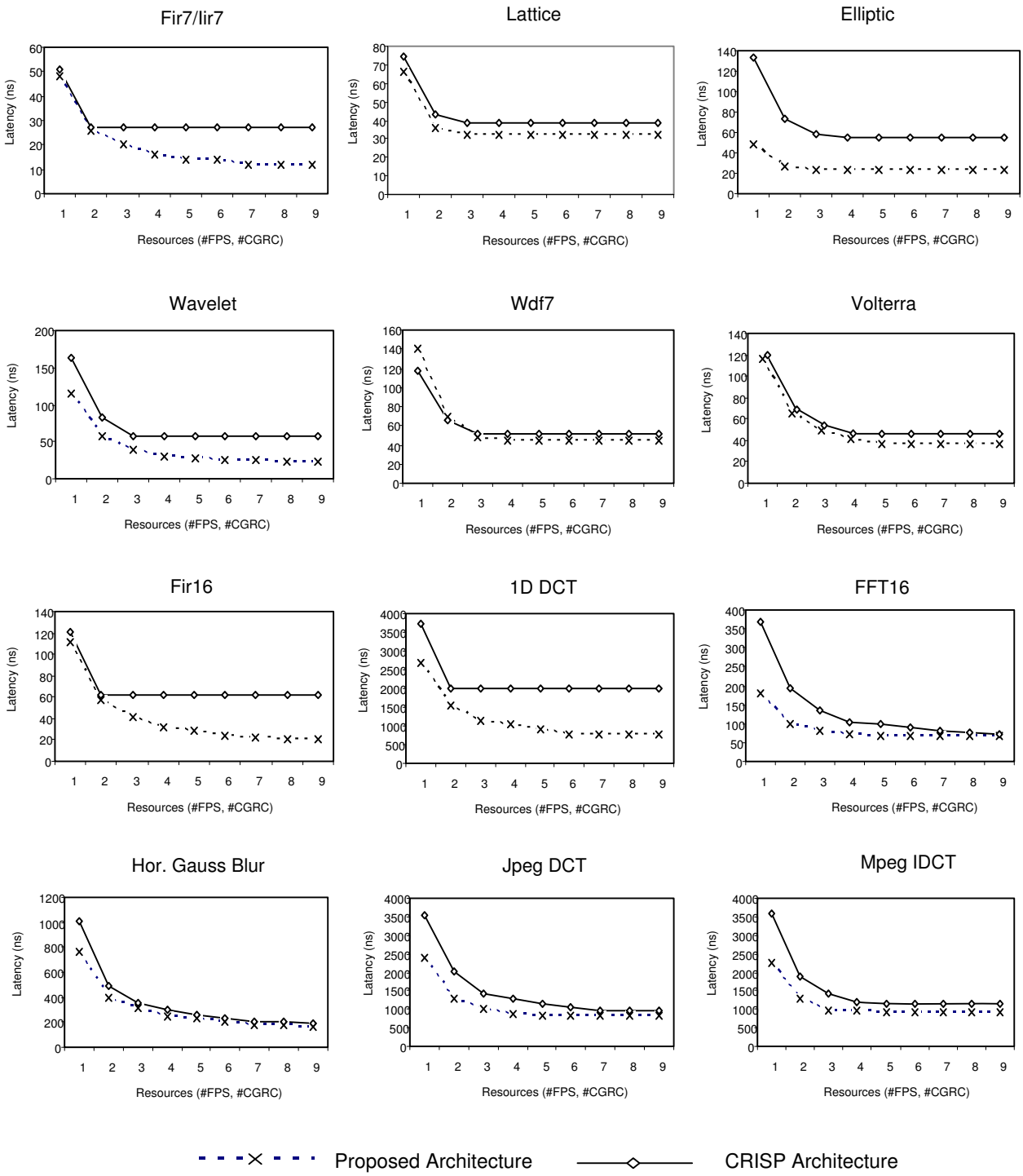


Figure 4.21.: Scalability Analysis of the DSP kernels. Horizontal axis considers the number of either CRISP's or FPS's allocated resources. Vertical axis reports the execution latency for each allocation scenario.

behavior. The proposed solution delivers faster implementations in all allocation scenarios than CRISP-based ones, except from the case of serial execution in the WDF7 kernel. WDF7 kernel is dominated by many alu-mul operation templates which impose the frequent invocation of CS2Bin conversion, thus limiting performance. However, this inefficiency is occurred only for the case of serial execution which does not form a realistic resource allocation scenario. In general, the proposed solution presents either better scalability features (FIR7, IIR7, Wavelet, Volterra, FIR16, 1D DCT kernels) or the same scalability (Lattice, Elliptic, WDF7, HGB, Jpeg DCT, Mpeg IDCT kernels) in comparison with the CRISP-like architectural solutions. Only for the FFT16 benchmark the FPS datapath presents lower scalability than CRISP. This is due to the DFG structure of the FFT16 kernel which includes a large number of additive operations, thus permitting CRISP to exploit the increment in the number of ALU resources. However, even in the FFT16 case, FPS solutions deliver faster implementations than the CRISP ones in all resource allocation scenarios.

Considering the second set of experiments, we generate various FPS units by changing their internal bit-width. We considered FPS units operating on 16-, 24-, 32-, 48- and 64-bit operands. The various bit-width FPS units were synthesized with Synopsys Design Compiler [135] using the TSMC 0.13 um library [107]. The basic reconfigurable units found in [15], [16] were also scaled in terms of operands bit-width and synthesized using the same experimental setup, to provide quantitative area-time comparisons. Fig. 4.22a reports the delay overheads, while Fig. 4.22b depicts the respective area overheads imposed by the bit-width scaling. The FPS unit remains the most efficient solution among all the other alternative flexible units, in both delay and area, for all the different bit-width configurations. Specifically, the FPS unit delivers gains up to 61.5% in delay and 71.4% in area (for the 64-bit case) in comparison with the CGRC units. Compared with the 64-bit implementation of the FCC unit, the FPS unit delivers gains up to 80.4% and 93% in delay and area, respectively.

In order to evaluate the bit-width scaling not only at the component's level but also its impact on the overall datapath architecture, we generate the CRISP-based [15], FCC-based [16] and the FPS-based flexible datapaths for the FIR16 kernel, considering the five different operand bit-widths. Fig. 4.22c and Fig. 4.22d depict the overall latency and area for each one of the aforementioned datapaths implementing the FIR16 kernel. It can be seen that the proposed flexible datapath solution scales better for various operands' bit-widths than its competitors. In each case, the FPS-based solution delivers significant area and latency gains i.e. up to 61% in latency and up to 72% in area compared with the FCC-datapath for 64-bit operands. For the same bit-width, latency gains up to 62% and area gains up to 49% are reported in comparison with the CRISP datapath.

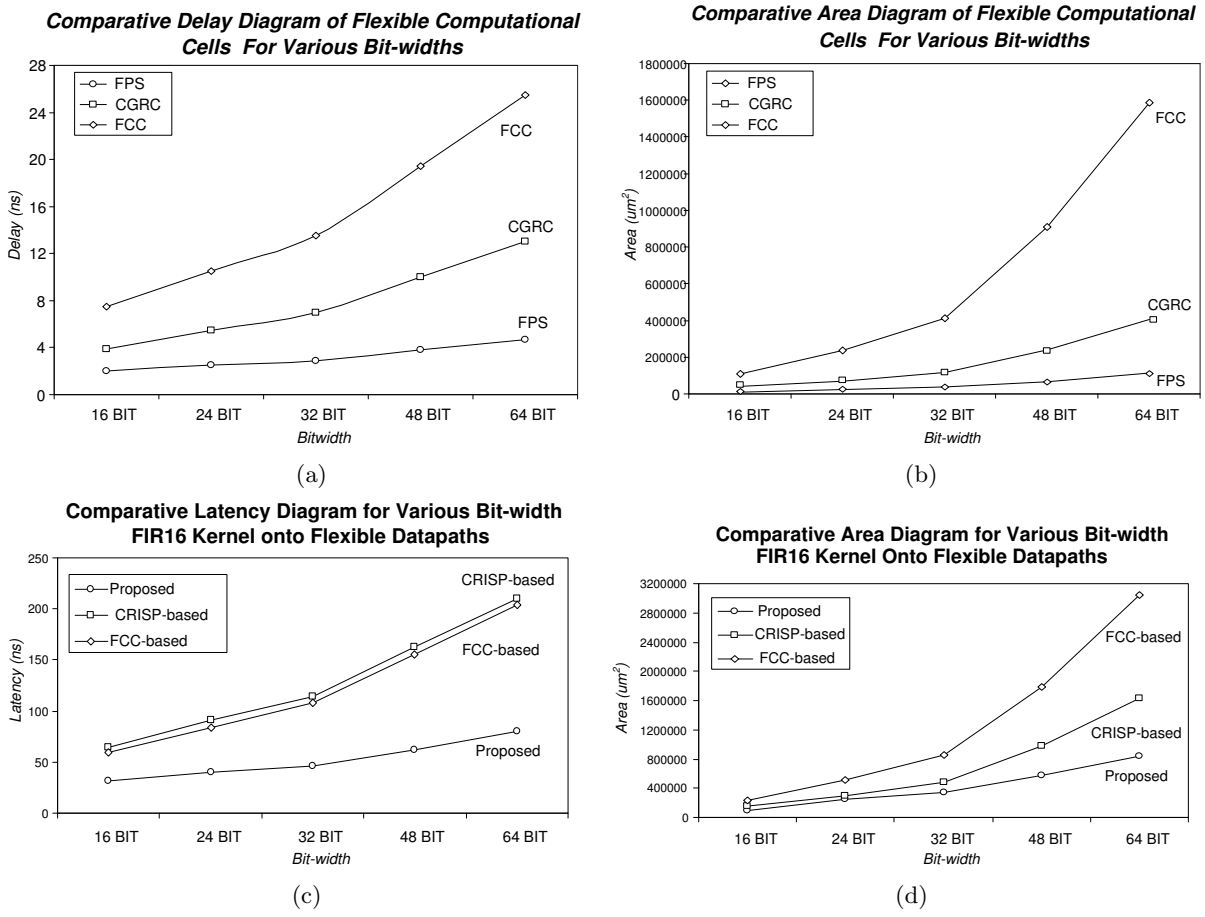


Figure 4.22.: Bit-width aware scalability. a) Delay of basic flexible computational components for various bit-widths. b) Area of basic flexible computational components for various bit-widths. c) Comparative latency results of different flexible architectures for various bit-widths implementing the FIR16 kernel. d) Comparative area results of different flexible architectures for various bit-widths implementing the FIR16 kernel.

4.7. Conclusion

This chapter presented a methodology for design and synthesis of flexible datapath architectures. The synthesis methodology targets a novel flexible architectural template by jointly exploiting the horizontal/vertical parallelism and operation chaining found into DSP kernels. Experimental results on several DSP benchmarks have shown significant performance and area gains compared to previously published high performance reconfigurable datapaths.

5. Architectural Synthesis of Flexible Hardware Accelerators Exploiting Carry-Save Operation Templates

In Chapter 4, we presented a synthesis approach for designing flexible DSP datapaths incorporating array-based multiplication units. However, complex operation templates incorporating tree-based multiplication units have been proven extremely efficient especially for the DSP application domain. In this chapter, we propose an alternative synthesis methodology targeting to template-based flexible datapaths with tree-based multipliers. Motivated by the observation that only architectural level optimizations have been reported for the specification and implementation of the operation templates, we introduce the consideration of arithmetic level optimizations for template based datapath synthesis. A high performance architecture for the implementation of DSP kernels is proposed, based on flexible and arithmetically optimized components able to perform a large set of operation templates. A synthesis methodology for optimized mapping of DSP kernels onto the proposed architecture is also presented. Experimental results showed significant gains in execution time, active chip area and power dissipation in comparison to state-of-the-art template-based flexible datapaths incorporating tree multipliers.

5.1. Introduction

Embedded computing has gain a lot of attention due to its continuously increased share in the ICs market. In contrast with general purpose computing, embedded devices target to specific and usually a-priori known application domain. For example, a digital system within a cellular phone it most likely needs to perform computationally intensive DSP functions i.e. filtering operations, FFT, image processing etc. Architectural customization of the embedded system has been recognized as a promising solution to achieve performance, area and power/energy gains. This customization tends towards the use of ASICs as specialized hardware accelerators tailored to the needs of the targeted application domain [7, 50, 150, 151]. The increasing success of HLS tools (CYBER [152], Catapult C [153], SPARK-HLS [67]) in automatically synthesizing high-level design descriptions makes possible the rapid evaluation of the ASIC accelerators.

Although efficient, ASICs are extremely inflexible. After fabrication the functionality of the device remains unchangeable, thus being unable to handle the migration of new or updated standards. In addition, this inflexibility comes along with increased silicon complexity due to the large number of the instantiated ASIC modules needed for accelerating the various kernels. Many researchers have identify this trade-off for high efficiency and increased flexibility and proposed the use of domain specific coarse-grained reconfigurable accelerators [16], [48], [118], [120], [15], [49] as a balanced solution to bridge these conflicting requirements.

Regarding high performance flexible datapath implementations, the authors in [16], [48], [49] (but also in [50], [7] regarding ASIC synthesis) proposed the instantiation of flexible chained operation templates in order to efficiently map to complex resources primitive operations found into the initial dataflow graph (DFG) of the hardware accelerated kernel. The complex operation templates are either specified in a predefined behavioral template library [16], [7], or extracted directly from the kernel's DFG [48].

Given that the template based HLS methodologies work at the architectural level, most of the aforementioned techniques generate datapaths taking into account only observations on the structure of the DFGs. Thus, little or none awareness of techniques from lower design levels (i.e arithmetic level optimization) are encountered. The exclusion of arithmetic optimizations during architectural synthesis comes along with the inherent limitation that the critical delay of the generated datapaths is determined from the large carry-propagation chains, found in the conventional binary arithmetic designs. In order to eliminate the large carry-propagation chains, some recent research activities, i.e. [18, 19, 51], concerned datapath optimization using CS arithmetic. However, these studies target only to inflexible ASIC implementations without concerning about flexibility of the final architecture.

The objective of this chapter is to deliver a synthesis methodology for flexible hardware accelerators exploiting arithmetically optimized operation templates. Targeting the DSP application domain, we introduce a novel flexible datapath architecture combining concepts and optimization techniques both from the architectural and the arithmetic level of abstraction. The proposed architecture comprises of uniform and flexible computational units which enable the execution of a large set of operation templates found into DSP kernels. The overall architecture operates in CS arithmetic delivering high speed implementations. The main contributions of this work are summarized in the following lines:

- We introduce a flexible accelerator architecture. The proposed flexible datapath is carefully designed to enable the increment of the scope that conventional CS aware arithmetic optimizations are applied.

- We present a systematic template-based synthesis methodology for optimized mapping of DSP kernels onto the proposed architecture.

Experimental comparisons at the circuit level show that the proposed flexible architecture is able to operate on a significantly larger range of operating frequencies and with lower area overheads than the existing ones. Based on a representative set of DSP kernels, we show that the proposed approach delivers an average improvement of 30% in execution latency together with 42% area reduction and 10% power gains, compared with state-of-the-art reconfigurable datapath solutions.

The rest of the chapter is organized as follows. Section 5.2 provides a discussion on recent literature concerning both flexible datapaths and CS optimization techniques. Section 5.3 shows the existing limitations of CS optimization approaches which highly motivated this work. Section 5.4 analyzes in depth the design considerations of proposed flexible architecture. In Section 5.5, we present a synthesis methodology developed for optimized mapping of DSP kernels onto instances of the proposed architecture. Experimental results are reported in Section 5.6, while Section 5.7 concludes the paper.

5.2. Related Work

In this section, we concentrate the discussion on existing work regarding coarse-grained reconfigurable architectures targeting the DSP application domain and CS arithmetic optimization techniques. Since DSP application domain is dominated from word level computationally intensive kernels with strict timing constraints, word level computations make coarse-grained reconfigurable datapaths an efficient architectural solution. In addition, CS arithmetic optimizations has been widely used to improve circuits' timing. Thus, the combination of coarse-grained reconfigurability and CS arithmetic forms a promising approach to move towards higher quality flexible datapath solutions.

Existing work in coarse-grained datapaths is mainly focused on exploiting architectural level optimizations i.e. instruction level parallelism, pipelining etc. Thus, most of the reconfigurable architectures [119], [120], [15], [49], [16] support increased levels of ILP, which is a standard optimization for hardware accelerators. Pipelining is supported in [119] and [120]. In [119], a reconfigurable architecture is presented which maps the computation onto a 1-D linear pipeline of coarse-grained components i.e. ALUs, RAMs etc. In [120], the authors proposed a reconfigurable architecture consisting of identical stripes, organized in a pipelined manner. Each stripe consists of parallel processing elements enabling ILP to be exploited in an intra-stripe manner. Combining ILP together with operation chaining is suggested in [49], [16]. They propose the usage of flexible operation-templates in order to enable efficient intra-template [49], [16] and inter-template [16] operation chaining.

The aforementioned reconfigurable architectures do not consider any type of arithmetic optimization during datapath's specification. Thus, any arithmetic optimization (if any) is limited only to the internal circuit structure of their primitive components (i.e. adders, ALUs, multipliers) during the logic synthesis [106]. However, recent research activities [19, 51, 154] have shown that arithmetic optimization at higher levels of abstraction than the structural circuit level, has significant impact on the quality of the final datapath. Specifically, transformation techniques onto the application's DFG are presented in [19] to maximize the use of CS arithmetic prior actual synthesis of the datapath. In [154], the authors optimizes the accelerator datapath of linear DSP systems using common subexpression elimination in CS computations. In [51], the problem of combined retiming, module selection and CS arithmetic representation selection is formulated, introducing the Mixed representation Flow Graph model to resolve the signal representation mismatch (CS vs 2's complement arithmetic). The final datapath comprises a mixture of CS or binary arithmetic operators. In [18], [54], timing driven post-RTL CS optimization is presented after the whole behavioral specification has been laid out as a single circuit. Although, the aforementioned CS optimization approaches operate at high abstraction level, they target inflexible datapath implementations, thus neither flexibility nor hardware-sharing issues have been considered. In addition, they are not handle the limitation of carry propagation prior to each multiplication operation, discussed in Section 5.3.

In this chapter, we leverage the aforementioned limitations regarding CS arithmetic optimizations during flexible datapath synthesis. We differentiate from previous approaches by proposing a template-based (semi-automated) architectural synthesis methodology which enables efficient implementations of flexible datapath accelerators inherently exploiting the use of CS arithmetic optimizations.

5.3. Carry Save Optimization: Limitations and Motivational Observations

Carry-Save representation [53] has been widely used to build fast arithmetic circuits (usually parallel multipliers [155]) due to its inherent feature of eliminating the large carry-propagation chains (i.e. Fig. 5.1a). CS arithmetic optimization is based on the usage of compressor trees. We call N:2 CS compressor a circuit which takes $N \geq 3$ input words, implements their CS addition (CSA) and produces the sum, S, and carry, C, words according to the following equations:

$$\{S, C\} = F(A_0, A_1, ..., A_{N-1}) \tag{5.1}$$

$$S + C = \sum_{i=0}^{N-1} A_i \tag{5.2}$$

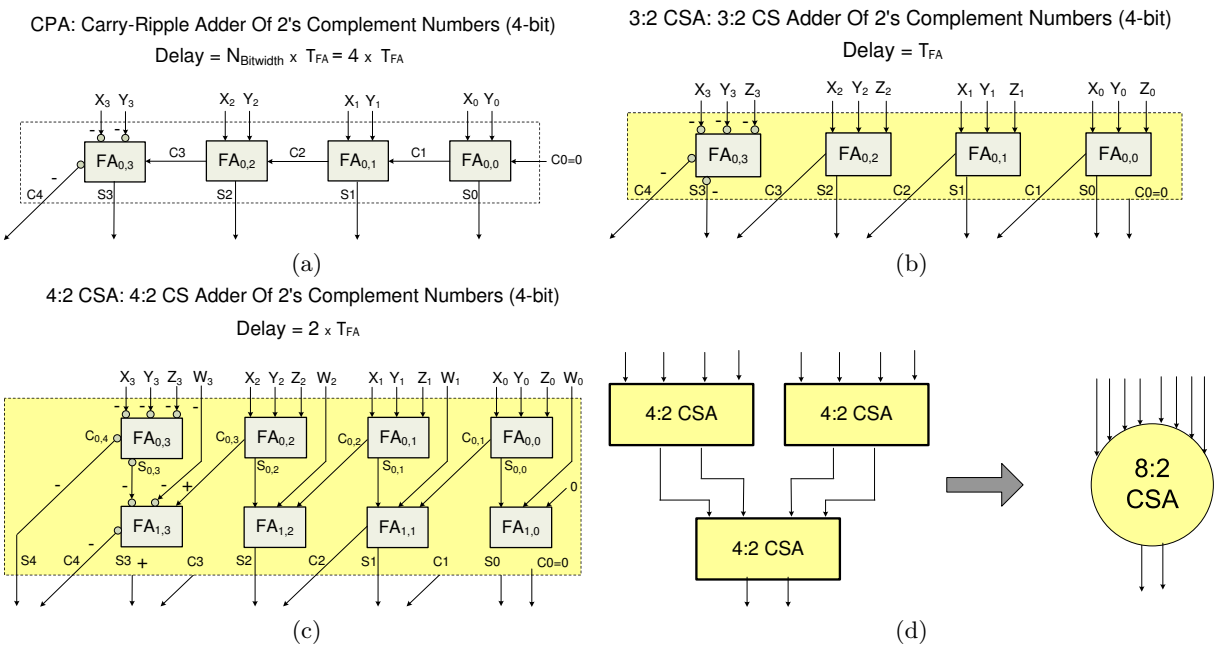


Figure 5.1.: a) A ripple-carry adder (carry propagate). b) A 3:2 CSA adder (4-bit). c) A 4:2 CSA adder (4-bit). d) A 8:2 compressor tree using 4:2 CSA units

As shown in [19], the use of compressors reduces the critical path (in most cases) of the application's DFG. However, the instantiation of large compressor trees in conjunction with late arrival of an addendum can make the final circuit implementation counterproductive. Thus, we limit further discussion on 3:2 (Fig. 5.1b) and 4:2 CS compressors (Fig. 5.1c). This is not restrictive, since larger compressor trees can be build using as primitive components these compressor blocks (Fig. 5.1d).

The main goal in CS arithmetic optimization is to maximize the range that the computation is performed with CS components to reduce the critical path of the DFG. Thus, the common ground in CS arithmetic optimization approaches is the appliance of DFG's transformations in order to rearrange the DFG and expose multi-input addition operations which are further mapped to CS compressors. Whenever the computation cannot be performed in CS representation, i.e. when a logical operator is interleaved, a carry-propagate addition is invoked to re-transform the data in conventional binary representation (i.e. 2's complement arithmetic representation). Recently, Verma et al. [19] have proposed a set of rewriting rules to transform the original DFG to a semantically equivalent one in which an increased number of CS compressor trees can be instantiated. As in [18], they manage the rearrangement of additive operations separated by shifting, selection or logical operators enabling the merging of multiple adders in a common CS compressor. Such type of DFG transformations are applied prior synthesis and can be used in a straightforward manner in our approach.

Table 5.1.: Delay-Area Estimation Model Based on FA modules for 16-bit Wordlength.

Module	Area (FAs)	Delay (T_{FA})
4:2 CSA	32	2
CPA	16	16
PPG	16	1
CSA Tree	128	4
CS-to-MB Recoding	32	2

Although high efficient in DFGs with shifting, logical etc. operations, the aforementioned CS optimization approaches have limited impact in DFGs dominated by multiplication operations, which is the case for the majority of filtering DSP applications. Thus, whenever a multiplication operation node is interleaved, a CS to 2's complement representation conversion is invoked [18] or the DFG is transformed according to the multiplication's distributive property [19]. Fig. 5.2b shows the former case in which a CS to 2's complement representation conversion is performed on the exemplary DFG of Fig. 5.2a. The delay of the DFG is increased since a carry-propagation operation is added prior to multiplication. In the worst case that the compressor tree prior multiplication is balanced, the delay increment is proportional to the word length. Fig. 5.2c illustrates the later case in which the multiplication is distributed over the CS formatted data. Although, the delay of the DFG is not increased, the area complexity is augmented since the number of the partial product generators and the CS compressors for performing a single multiplication is doubled.

Previous analysis indicates that delay-area trade-off exists concerning the CS arithmetic optimization of the DFG. However, the so far proposed solutions limit the resolution of this trade-off considering only the two extreme approaches (Fig. 5.2d) of increasing either the delay or the area of the final circuit implementation. Inspired from [52], we leverage the above limitation by adopting a CS to Modified Booth (MB) [53] recoding approach each time a multiplication operation has to be handled within a CS optimized datapath implementation. The CS-to-MB recoding module exhibits very efficient area and delay characteristics–critical delay of 8 logic gates $\approx$ 2.5$\times$T$_{FA}$ independent from the data word-length, area complexity of one 4:2 CSA approximately, (see Subsection 5.4.2 for implementation details.

Recoding operations enables the computation to be performed in CS representation across multiplications (no need for large carry-propagation chains). It replaces the intermediate Carry Propagate Adder (CPA) module of Fig. 5.2b, eliminating time consuming carry-propagation. In order to quantify the benefits offered from the proposed solutions, we incorporate a delay-area estimation model based on FA characteristics (Table 5.1) Fig. 5.2d depicts the positioning of the proposed ap-

proach in respect to the aforementioned CS optimization techniques. Thus, through the incorporation of recoding operations prior multiplication we manage to generate CS optimized circuit implementations with area complexity similar to Fig. 5.2b and critical delay similar to 5.2c.

Based on the above observation, we propose a novel flexible architecture based on CS chained operation templates for hardware acceleration of computationally intensive DSP kernels. The proposed architecture is able to perform computations in both 2's complement and CS arithmetic representation. Furthermore, the integration of hardware recoding modules enables the CS optimized DFGs to be efficiently mapped onto the proposed datapath.

5.4. Flexible Datapath Architecture

In this section, we introduce the proposed flexible architecture. An abstract architectural model of the proposed data-path is illustrated in Fig. 5.3. It is composed by (i) the Flexible Computational Units (FCUs) described in Subsection 5.4.1 , (ii) the register bank, (iii) the data interconnection network, (iv) a CS to 2's complement conversion module (CS2Bin) and (v) the control unit which drives the overall architecture (configuration words and multiplexor's selection signals) in each control step.

The proposed architecture is able to perform computations either in CS or 2's complement arithmetic representation. Each individual component has been designed to operate on word operands of 16-bit, since such a word length is considered adequate for the majority of DSP datapaths [46]. The number of FCUs can be determined at design time according to the desired ILP and the area constraints imposed by the designer. The register bank is introduced for the storage of intermediate results and the sharing and the communication of variables among the FCUs. It is implemented by scratch registers according to the register allocation of each kernel. The data interconnection network handles the communication between the register bank and the FCUs. Different DSP kernels (different register allocation and data communication patterns per kernel) can be mapped onto the proposed architecture utilizing post RTL datapath interconnection sharing techniques [122]. The arithmetic conversion module (CS2Bin) performs the conversion from the CS format to 2's complement arithmetic representation. It is implemented as a carry-ripple adder [53], since its critical path delay is overlapped with the critical path delay of the FCUs. The arithmetic conversion usually takes place at the end of each kernel execution in order to output the computed result in binary format.

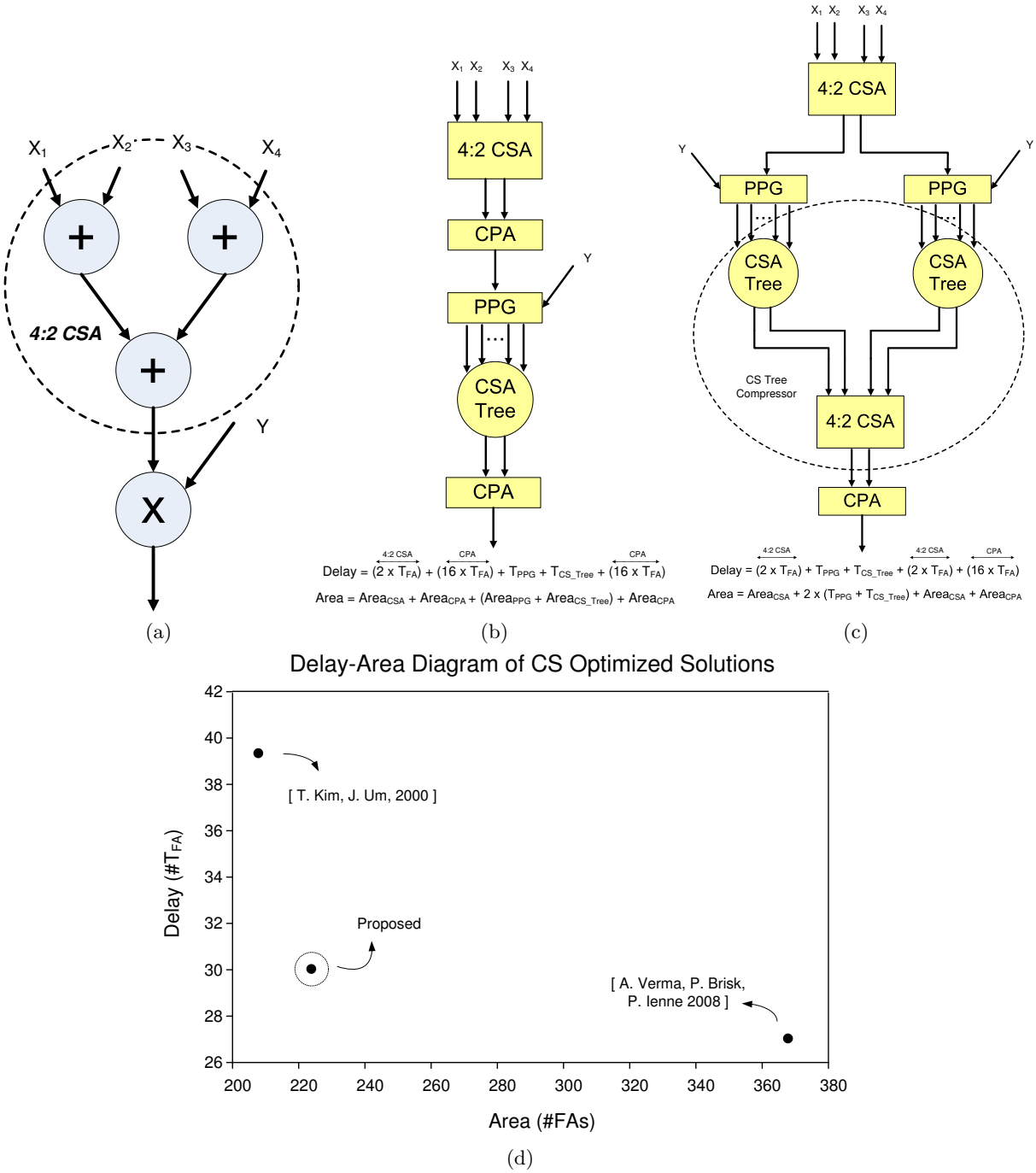


Figure 5.2.: a) Exemplary DFG. b) CS optimization [18]. c) CS optimization based on multiplication distribution [19]. d) Positioning of the proposed approach in respect to CS optimizations in [18], [19].

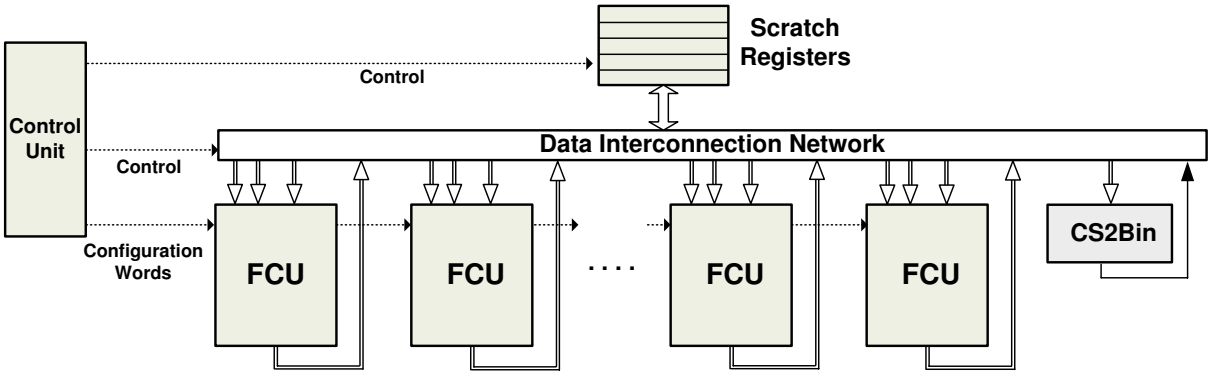


Figure 5.3.: Abstract View of the Flexible Datapath.

5.4.1. Structure of Flexible Computational Unit

The structure of the FCU has been designed to allow high performance template-based and flexible operation chaining. Operation chaining schedules data dependent operations in the same clock cycle, eliminating the registers storing the intermediate results. Thus, improvements in both total delay and in area are delivered. The notion of flexible operation chaining has been recently introduced in [16] to enable efficient template sharing. Fig. 5.4 depicts the abstract model of the proposed FCU. The internal structure of the FCU node (detailed model) and the way the abstract model maps onto the detailed one are also outlined.

The proposed FCU enables intra-template operation chaining by merging together the pre- and the post- multiplication addition. The alternative execution paths in each FCU are determined by properly setting the control signals of the two multiplexors. The FCU defines a configurable operator able to perform any operation subgraph defined in the following two complex operations:

$$Z^* = (X^* + Y^*) \times A + K^* \tag{5.3}$$

$$Z^* = A \times K^* + (X^* + Y^*) \tag{5.4}$$

The superscript, *, denotes a CS arithmetic representation composed of two numbers both in 2's complement form. Thus, the X^*, Y^*, K^*, N^*, Z^* are in CS arithmetic representation. The quantities X^C, X^S, Y^C, Y^S, K^C, K^S, N^C, N^S, Z^C, Z^S and A are all 2's complement conventional binary numbers (Fig. 5.4). In general, the following relation stands for all CS formatted data: $X^* = \{X^C, X^S\} = X^C + X^S$.

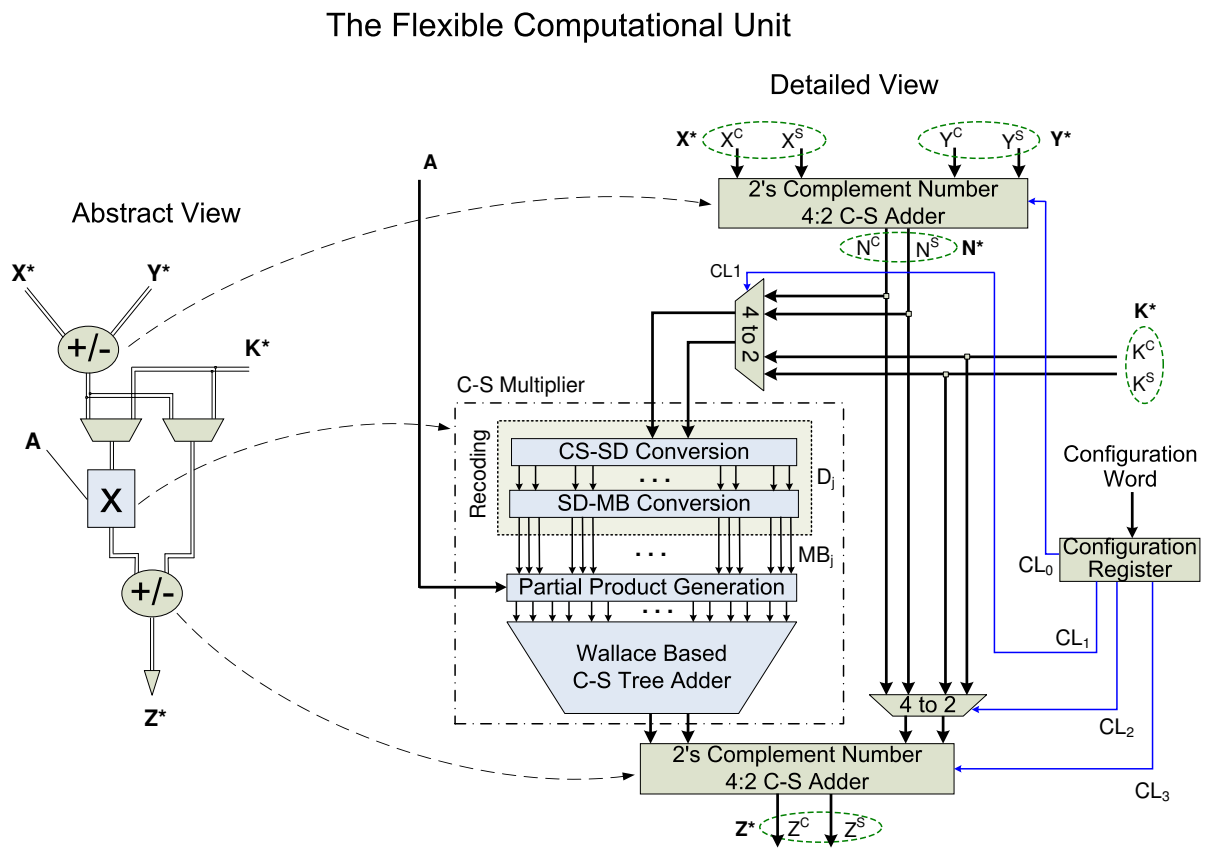


Figure 5.4.: The Flexible Computational Unit.

Fig. 5.5 illustrates the library of template operations that each FCU can be configured. The template operations are depicted in their CS arithmetic representation. Input data in 2's complement arithmetic, can be directly processed by a FCU without any conversion overhead, since every CS formatted number can be viewed as two 2's complement numbers. Two level operation templates (of type Add-Mul, Mul-Add) dominates most of the DSP kernels [7]. However, three level operation templates are also occurred in many DSP kernels i.e. in the Symmetrical FIR filters, DCT algorithms etc. The proposed datapath supports also this type of templates (i.e T1 of Fig. 5.5).

The internal structure of the FCU (Fig. 5.4) consists of (i) a 4:2 CS adder of 2's complement numbers, for the addition of the input data (X^*, Y^*), (ii) a CS to MB recoding scheme (detailed description in Subsection 5.4.2), (iii) a tree based adder for the addition of the partial products, (iv) the final CS accumulation unit implemented also by a 4:2 CS adder and (v) a configuration register. Each FCU operates directly on, and produces data in CS arithmetic representation. This feature enables the direct reusability of the unit's output and eliminates the latency intensive carry-chains found in the conversion from the CS representation to the conventional binary form.

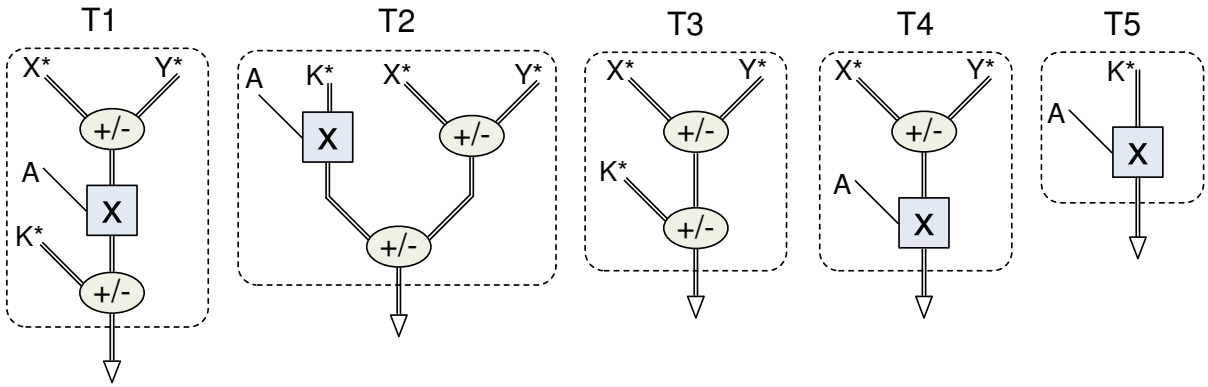


Figure 5.5.: The FCU Template Library.

The upper 4:2 CS adder computes the $N^* = X^* + Y^*$ which can be used either in the pre- or the post- multiplication addition. Input A has to be always in 2's complement binary numeric representation. At next, the 2's complement CS formed data, N^* or K^*, are driven either for multiplication or addition.

We further describe the multiplication path, since it forms the critical path of the FCU. When the FCU is configured to perform multiplication, the recoding unit is activated. The recoding unit enables the multiplication operation to be conducted with one operand (N^* or K^*) in CS representation. It performs the conversion from CS format to an intermediate format of Signed Digits (SDs) [53] and subsequently the conversion of intermediate SDs to the Modified Booth (MB) digit representation [53]. Each MB digit has a value ranging in {-2, -1, 0, +1, +2}, while in conventional Booth encoding each digit ranges in {-1, 0, +1}. This is achieved by grouping together three subsequent bits (with bit overlap) in MB encoding rather than two bits used in conventional Booth. The MB encoding technique is used in high performance multipliers. It reduces the number of generated partial products thus delivering both performance and area improvements. It is worth to be noticed that the CS to SD and the SD to MB recoding modules are carry-propagation-less circuits, contributing only slightly on the critical path delay (Subsection 5.4.2). The MB digits are passed into the partial product generator and the partial products are reduced within a CS Wallace tree addition scheme [53].

The final accumulation unit is a 4:2 CS adder with two fixed inputs (the multipliers CS output) and two configurable inputs. All the inputs of the accumulation unit are in 2's complement CS format. The configured inputs are either the N^* derived from the initial 4:2 CS adder or an independent CS input number (K^*). Finally, the operation mode of the FCU and the signs of the input numbers (determining addition or subtraction operations) are controlled through the configuration register, by driving with proper control logic bits (CL_i) the multiplexors and the sign selection inside each 4:2 CS adder module. The configuration register is loaded in a cycle by cycle basis with configuration words which are generated in the control

unit (Fig. 5.3).

5.4.2. The Recoding Unit

The recoding module transforms a CS data in its equivalent set of MB digits [53]. In conventional MB encoders, the input data is assumed to be in 2's complement arithmetic representation. In order to enable MB encoding directly from a CS representation, alleviating the delay overhead of CS to 2's complement conversion, we adopt the recoding technique proposed in [52]. Let us assume that P^* is a CS formatted data. The following equations stand:

$$P^* = P^S + P^C,\ P^S = \{p^S_{L-1}, p^S_{L-2}, ..., p^S_0\},\ P^C = \{p^C_{L-1}, p^C_{L-2}, ..., p^C_0\} \tag{5.5}$$

$$P^* = (-2^{L-1}) \cdot (p^S_{L-1} + p^C_{L-1}) + \sum_{i=0}^{L-2} (p^S_i + p^C_i) \cdot 2^i = (-2^{L-1}) \cdot z_{L-1} + \sum_{i=0}^{L-2} z_i \cdot 2^i \tag{5.6}$$

Where: L is the number of bits in P^S and P^C, p^S_i and p^C_i the i^{th}-rank bits and the digit $\widehat{z}_i = \{p^S_i = p^S_i + p^C_i, p^C_i\} \in \{0, 1, 2\}$. Equation 5.6 is further analyzed as follows:

$$P^* = 4^{(\frac{L}{2})-1} \cdot ((-2) \cdot p^S_{L-1} + (-2) \cdot p^C_{L-1}) + \sum_{j=0}^{(\frac{L}{2})-2} (2 \cdot p^S_{2j+1} + 2 \cdot p^C_{2j+1} + p^S_{2j} + p^C_{2j}) \cdot 4^j \tag{5.7}$$

$$P^* = 4^{(\frac{L}{2})-1} \cdot \widehat{z}_{(\frac{L}{2})-1} + \sum_{j=0}^{(\frac{L}{2})-2} \widehat{z}_j \cdot 4^j \tag{5.8}$$

Each $\widehat{z}_j \in \{0, 1, 2, 3, 4, 5, 6\}$, except $\widehat{z}_{(\frac{L}{2})-1} \in \{0, 1, 2, 3, 4\}$. The value range of $\widehat{z}_i$ digits is not a MB encoding. Thus, we further transform it by inserting the digits $\widehat{t}_{j+1} = \{t^+_{j+1}, t{-}_{j+1}\}$ and $\widehat{u}_j = \{u^+_j, u^{++}_j\}$ [52]. Table 5.2 depicts the truth table of the above mentioned recoding. Specifically, $\widehat{z}_j$:

$$\widehat{z}_j = \begin{cases} 4 \cdot \widehat{t}_{j+1} + \widehat{u}_j = 4 \cdot t^+_{j+1} + u^+_j + 2 \cdot u^{++}_j & \widehat{z}_j \neq \widehat{z}_{(\frac{L}{2})-1} \\ 4 \cdot \widehat{t}_{j+1} + \widehat{u}_j = 4 \cdot t^-_{j+1} + u^+_j + 2 \cdot u^{++}_j & \widehat{z}_j = \widehat{z}_{(\frac{L}{2})-1} \end{cases} \tag{5.9}$$

Grouping digits with the same rank ($\widehat{t}_j$, $\widehat{u}_j$), the $\widehat{q}_j$ digit is defined ranging in $\{0, 1, 2, 4\}$.

$$\widehat{q}_j = \widehat{t}_j + \widehat{u}_j = t^+_j + u^+_j + 2 \cdot u^{++}_j \tag{5.10}$$

Table 5.2.: Recoding table of $\widehat{z}_i$ according to $\widehat{t}_{j+1}$ and $\widehat{u}_j$ digits. The values inside brackets refers to the case that $\widehat{z}_i = \widehat{z}_{(\frac{L}{2})-1}$.

$\widehat{z}_j$	$\widehat{t}_{j+1}$	t_{+j+1}	t_{-j+1}	$\widehat{u}_j$	u_j^+	u_j^{++}
0(-4)	0(-1)	0	(1)	0	0	0
1(-3)	0(-1)	0	(1)	1	1	0
2(-2)	0(-1)	0	(1)	2	1	1
3(-1)	0(-1)	0	(1)	3	1	1
4(0)	1(0)	1	(0)	0	0	0
5(1)	1(0)	1	(0)	1	1	0
6(2)	1(0)	1	(0)	2	0	1

Table 5.3.: Recoding table of $\widehat{q}_j$ according to $\widehat{h}_{j+1}$ and $\widehat{n}_j$ digits.

$\widehat{q}_j$	$\widehat{h}_{j+1}$	h_{+j+1}	$\widehat{n}_j$	n_j^+	n_j^{--}
0	0	0	0	0	0
1	0	0	1	1	0
2	1	1	-2	0	1
3	1	1	-1	1	1
4	1	1	0	0	0

Since the value range of $\widehat{q}_j$ digits still remains different from the MB encoding one, we further recode the $\widehat{q}_j$ using the $\widehat{h}_{j+1} = h_{j+1}^+$ and $\widehat{n}_j = (n_j^+, n_j^{--})$ digits.

$$\widehat{q}_j = 4 \cdot \widehat{h}_{j+1} + \widehat{n}_j = 4 \cdot h_j^+ + n_j^+ + 2 \cdot n_j^{--} \tag{5.11}$$

Table 5.3 depicts the truth table of the the second recoding.

Grouping again digits with the same rank ($\widehat{h}_j$, $\widehat{n}_j$), the $\widehat{s}_j$ digit is defined ranging in $\{-2, -1, 0, +1, +2\}$, which forms a MB encoding.

$$\widehat{s}_j = 4 \cdot \widehat{h}_j + \widehat{n}_j = (-2) \cdot n_j^{--} + h_j^+ + n_j^+ \tag{5.12}$$

Thus, each $\widehat{s}_j$ is a MB digit that can be directly used for partial product generation during a multiplication operation. Specifically, the CS data P^* can be now expressed as follows:

$$P^* = \sum_{j=0}^{(\frac{L+1}{2})-1} ((-2) \cdot n_j^{--} + h_j^+ + n_j^+) \cdot 4^j = \sum_{j=0}^{(\frac{L+1}{2})-1} (s_j \cdot 4^j) \tag{5.13}$$

Eq. 5.14-5.18 describes the logic expressions of the n_j^{--}, n_j^+, h_j^+, h_{j+1}^+, t_{j+1}^+ signals for the case $\widehat{z}_j \neq \widehat{z}_{(\frac{L}{2})-1}$. The logic expressions have been extracted according to the

Table 5.4.: Modified Booth Encoding table.

n_j^{--}	n_j^{+}	h_{+j}	N_j	SL_j	Z_j
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	0	0	1

recoding Tables 5.2-5.3. Fig. 5.6a shows the corresponding gate-level implementation.

$$n_j^{--} = \overline{p_{2j}^S \cdot p_{2j}^C} \bigoplus p_{2j+1}^S \bigoplus p_{2j+1}^C \bigoplus \overline{(p_{2j}^S \bigoplus p_{2j}^C) \cdot t_j} \tag{5.14}$$

$$n_j^{+} = p_{2j}^S \bigoplus p_{2j}^C \bigoplus t_j \tag{5.15}$$

$$h_j^{+} = h_j \tag{5.16}$$

$$h_{j+1} = (p_{2j}^S \cdot p_{2j}^C) \bigoplus p_{2j+1}^S \bigoplus p_{2j+1}^C \bigoplus (p_{2j}^S \bigoplus p_{2j}^C) \cdot t_j \tag{5.17}$$

$$t_{j+1} = p_{2j}^S \cdot p_{2j}^C \cdot (p_{2j+1}^S + p_{2j+1}^C) + (p_{2j+1}^S \cdot p_{2j+1}^C) \tag{5.18}$$

Eq. 5.19-5.21 describes the logic expressions of the n_j^{--}, n_j^{+}, h_j^{+} signals in case of the most-significant digit is recoded. Since the Most Significant Bit (MSB) of a 2's complement number has a negative value, special care has to be taken during its recoding. Fig. 5.6b shows the corresponding the gate-level implementations.

$$n_j^{--} = \begin{cases} t_j & L \in \{2k, \forall k \in N\}, j = \frac{L}{2} \\ (p_{L-1}^S \cdot p_{L-1}^C) + (p_{L-1}^S + p_{L-1}^C) \cdot \bar{t}_j & L \in \{2k+1, \forall k \in N\}, j = \frac{L-1}{2} \end{cases} \tag{5.19}$$

$$n_j^{+} = \begin{cases} t_j & L \in \{2k, \forall k \in N\}, j = \frac{L}{2} \\ t_j \bigoplus (p_{L-1}^S \bigoplus p_{L-1}^C) & L \in \{2k+1, \forall k \in N\}, j = \frac{L-1}{2} \end{cases} \tag{5.20}$$

$$h_j = h_j \tag{5.21}$$

Although the n_j^{--}, n_j^{+}, h_j^{+} signals form a MB encoded digit and can be used directly for partial product generation, we perform one more recoding step to the N_j, SL_j, Z_j signals which form a more suitable MB representation for partial product generation during multiplication. Table 5.4 shows the MB encoding, Fig. 5.7a its gate-level implementation and Eq.5.22-5.24 the corresponding logic expressions.

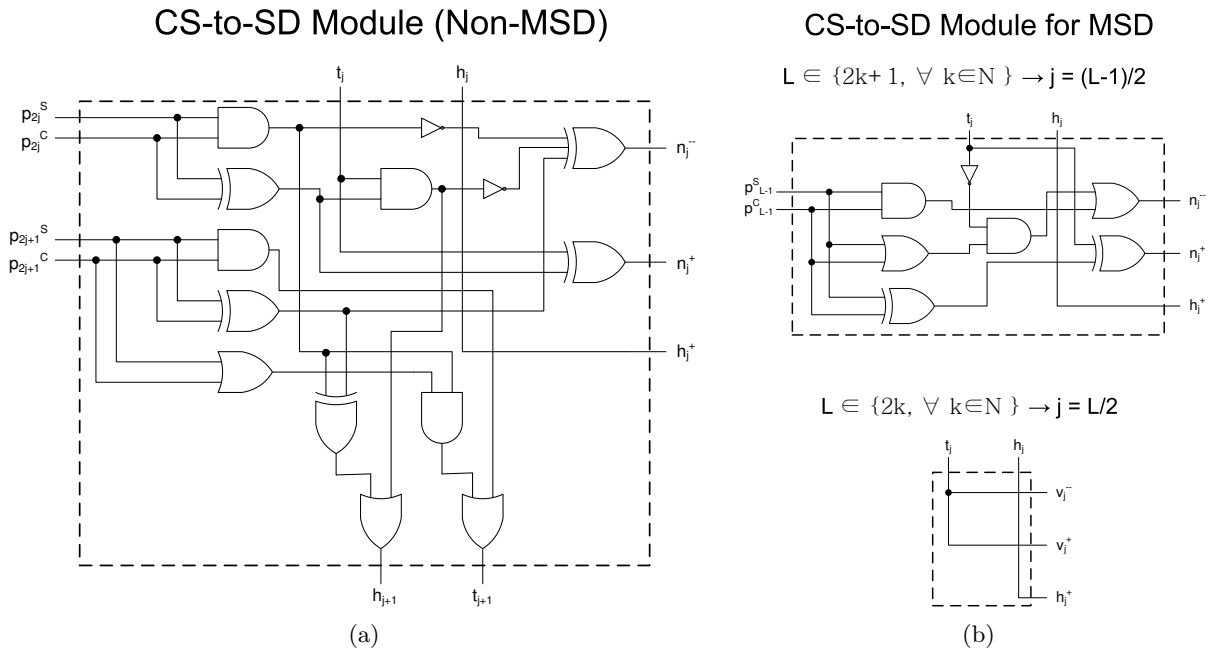


Figure 5.6.: Gate-level implementation of CS-to-SD modules a) Non Most Significant Digit (Non-MSD). b) Significant Digit (Non-MSD).

$$N_j = n_j^{--} \cdot \overline{(n_j^{--} \cdot n_j^{+} \cdot h_j^{+})} \tag{5.22}$$

$$SL_j = (n_j^{--} \cdot \overline{n}_j^{+} \cdot \overline{h}_j^{+}) + (\overline{n}_j^{--} \cdot n_j^{+} \cdot h_j^{+}) \tag{5.23}$$

$$Z_j = (\overline{n}_j^{--} \cdot \overline{n}_j^{+} \cdot \overline{h}_j^{+}) + (n_j^{--} \cdot n_j^{+} \cdot h_j^{+}) \tag{5.24}$$

The proposed flexible datapath operates on 16-bit data regarding the 2's complement arithmetic representation. The corresponding CS representation is formed through 17 CS digits. Thus, the CS to MB recoding module generates 9 MB digits. Fig. 5.7b depicts the structure of the CS to MB recoding unit instantiated inside each FCU. It consists of the CS-to-SD modules performing the first recoding (Eq. 5.19-5.21) and the MBE modules performing the MD encoding (Eq. 5.22-5.24). Regarding the jth rank in the CS2SD module, the h_{j+1} and t_{j+1} signals are generated with a delay of 3 logic gates (Eqs. 5.17-5.18, Fig. 5.6a). However, h_{j+1} depends on the the t_j signal (generated in j-1th rank), thus a propagation delay of 3 logic gates is required. Based on the above analysis, an overall delay of 6 logic gates is required for the generation of h_j and h_j^{++} signal. In addition, each MBE module has a delay of 2 logic gates. Thus, the overall delay can be estimated to the delay of 8 logic gates.

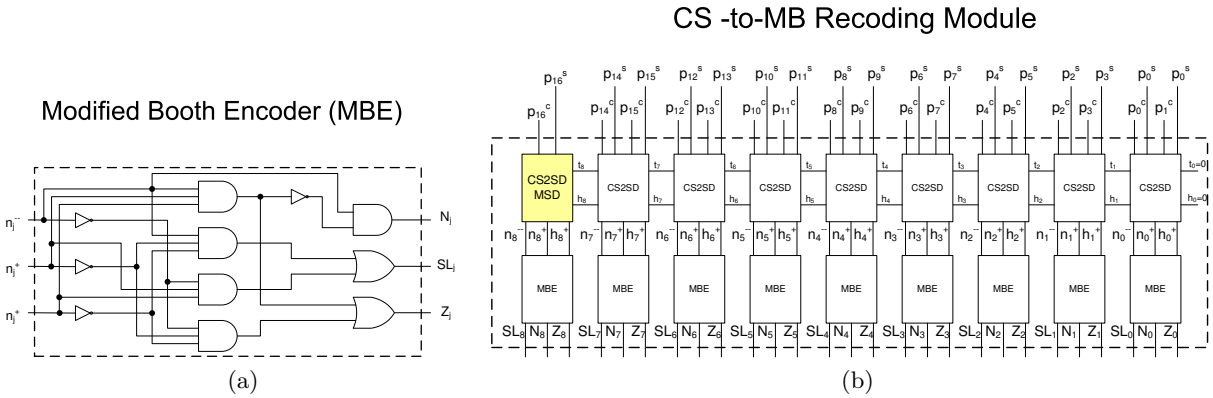


Figure 5.7.: a) Gate-level implementation of Modified Booth Encoder. b) The CS-to-MB recoding module.

5.5. Synthesis Methodology

An HLS synthesis methodology has been developed in order to enable the efficient mapping of DSP kernels onto the FCU-based architecture. The overall flow of the proposed methodology is depicted in Fig. 5.8. It consists of 3 major phases. Each phase is reported with different color in Fig. 5.8, in order to be clearly distinguished.

Phase 1: The DFG of the kernel is extracted from its C code specification,. Arrays' address calculations are statically pre-computed and substituted in the original DFG by register to register transfers. By this way, the FCUs are concentrated on effective computations and not on simple index calculations. The proposed datapath produces data in CS arithmetic representation, which is inadequate format for arrays' indexes. The DFG graph is unrolled according to the number of multiplication operations in the DFG and the allocated FCUs. By this way, the available ILP is balanced to the utilization of the allocated hardware, since the multiplier is the area-dominant component of the FCU.

Phase 2: In this phase the CS aware reduction of the original DFG is performed. It is similar to applying aggressively the CS module selection technique in [51]. The DFG reduction exploits the inherent feature of CS addition/subtraction circuits to merge/compress multiple-additions into a single one [54]. Thus, the size of the DFG shrinks, offering opportunities for faster schedules (number of cycles) than considering primitive resource operations. The proposed flexible architecture is able to handle 3:2 together with 4:2 CS compression, since 4:2 CS compressors are a superset of 3:2 compressors.

The pseudocode of the proposed CS aware DFG reduction is shown in Fig. 5.9a. We used the notion of Boundary Operations (B_Ops) similar to [54]. Boundary Opera-

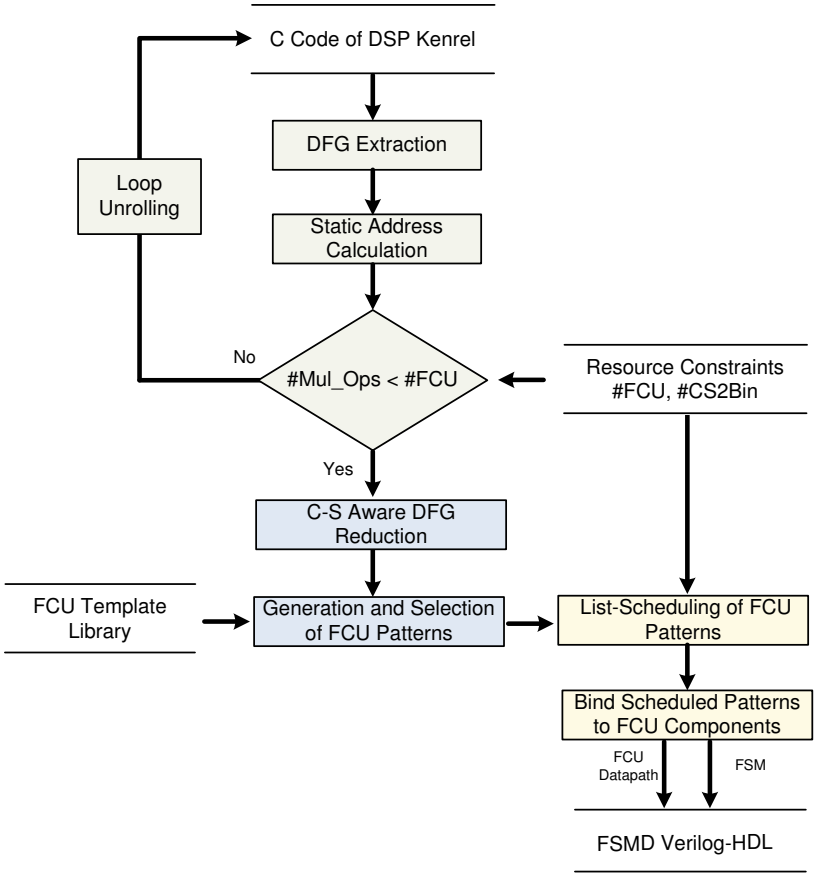


Figure 5.8.: The proposed Synthesis Flow.

tions are the DFG's nodes which set the boundaries of CS aware reductions. Thus, CS reductions are conducted in DFG nodes that lay between B_Ops nodes. In our case, assuming DFG graphs which comprise only addition/subtraction and multiplication operations, three types of B_Ops are encountered:

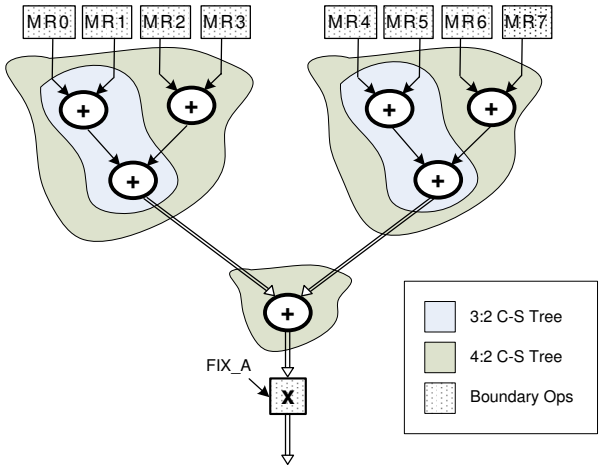
- **The DFG's primary inputs.**
- **The DFG's primary outputs.**
- **The multiplication nodes of the DFG.**

At first, the original DFG graph is ASAP scheduled, in order to topologically order the DFG's nodes according to their timing dependencies. Next, the CS addition trees are formed iteratively, by merging primitive DFG nodes. The B_Ops are excluded of the merging process, while the high fan-out DFG nodes are merged only as roots of the CS trees. At the end of each iteration, the formed CS trees are substituted in the original DFG in order to be candidate nodes for extra merging in the next iteration.

```
C-S Aware DFG Reduction
Input_1: DFG;
/*Set Boundary Operations (B_Op) */
Input_2: B_Op ← {I/O Ops, Mul Ops};
Output: Reduced DFG;

Schedule ASAP the DFG;
for (step=2; step<= #sched_steps; step++){
    for each Opi in step{
        if (Opi != B_Op){
            Bottom-Up Generation of 3:2 C-S Trees;
            Bottom-Up Generation of 4:2 C-S Trees;
            Reconstruct the Scheduled DFG;
        }
        else
        continue;
    }
}
```

(a)



(b)

Figure 5.9.: a) Psheudocode CS Aware DFG Reduction. b) Example of CS Reduction on a sample DFG.

In each iteration, the 3:2 CS reductions are considered first and a second pass forms the 4:2 CS reductions wherever it is possible. The 3:2 compression nodes remain until the end of the CS aware reduction process in order to include the case of a larger merging at the next steps. After, the completion of the CS reduction process, the remained uncompressed addition nodes or 3:2 compression nodes are substituted by 4:2 compressor nodes, by adding one or two zero inputs, respectively, at the unbound ports. Fig. 5.9b illustrates the CS aware reduction on a sample DFG.

The CS aware reduction process transforms the original DFG in an intermediate representation which is compliant with the FCU's resource model. A pattern generation procedure is applied onto the reduced DFG with respect to the FCU template library (Fig. 5.4). The pattern generation is a covering of the reduced DFG according to the FCU's operation templates. It actually clusters the DFG's CS reduced nodes with the multiplication nodes, in order to perform maximal operation chaining. Currently, the covering patterns are generated in an exhaustive manner. The designer is responsible to select those patterns that optimally cover the reduced DFG, as far as it concerns the minimization of DFG's execution time.

Phase 3: The clustered DFG is scheduled in order to assign each FCU operation to a specific control step. Since the datapath is realized by a fixed number of FCUs, a resource-constrained scheduling problem with the goal of latency minimization is considered. The number of available FCUs and CS2Bin modules determines the

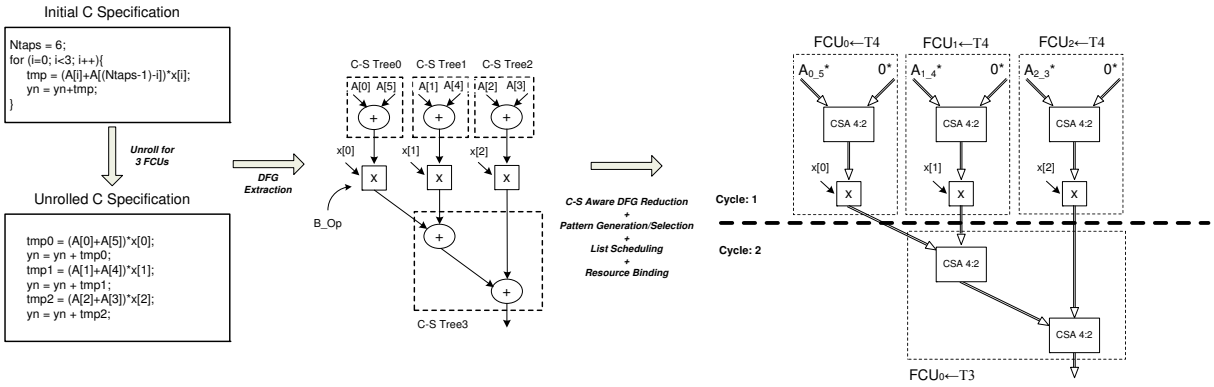


Figure 5.10.: Mapping onto FCU datapaths: An illustrative example.

resource constraint set. A list-based scheduler [44] that takes into account the mobility of FCU operations has been used. The ASAP and ALAP time-stamps of the FCU operations are calculated and the mobility for each FCU operation is extracted as the difference of the corresponding ALAP-ASAP value. The FCU operations are prioritized based on the lower mobility value, since the lower the operation's mobility the most critical the operation. Next, the scheduled FCU operations are bound onto specific FCU instances and the proper configuration bits are generated. After the completion of register allocation, a FSM description is extracted in order to implement the control unit of the overall architecture, and a synthesizable Verilog model of the FCU based architecture is generated. Fig. 5.10 shows an illustrative example of the proposed methodology, considering the basic computational kernel of a 6-tap Symmetrical FIR filter.

5.6. Experimental Results

In this section, we provide experimental results showing the effectiveness of our approach. We have compared the FCU based datapath with the one presented in [16] to show the effectiveness of the proposed CS arithmetic optimization.

A comparative area-timing exploration between the two flexible computational components, the proposed FCU and the FCC in [16] has been conducted. We have also included explorative results for the case of a conventional 16-bit multiplication unit (DW_Mul), in order to provide straightforward comparison with the area dominant component (multiplication unit) of non-flexible implementations. All the components were mapped onto the standard cells of TSMC 0.13um technology library, using the Synopsys Design Compiler version 2006 [135]. Since in this chapter we target to flexible datapaths incorporating tree-based multiplication operators, the arithmetic optimized "pparch" implementations from the Synopsys DesignWare library [135] were considered for both the conventional multiplication unit and the

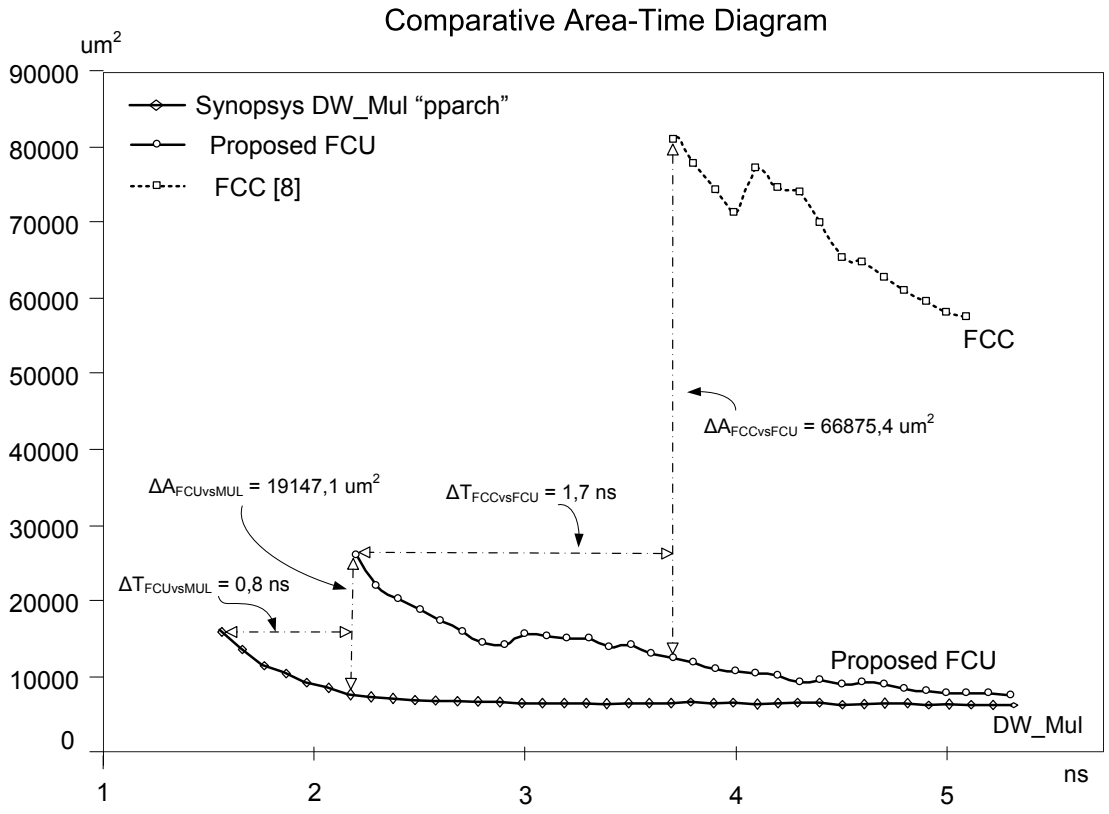


Figure 5.11.: Area-Time Explorative Diagram.

multiplication units found into FCC [16]. "Pparch" implementations of DesignWare library [135] offer the most efficient multiplier solutions tailored to area and timing constraints.

The lower limits of the area-timing values for the three implementations were exposed based on an iterative synthesis procedure which generated different delay-constrained versions of the two datapaths. The delay constraint was altered in each iteration considering a time interval of 0.1 ns, with an initial value of 1.0 ns and final value of 5.3 ns. Fig. 5.11 reports the comparative results. The proposed FCU is able to operate, without timing violations, in a time frame between [2.2ns, 5.3ns]. Respectively, the FCC unit [16] operates without timing violations in a time frame of [3.9ns, 5.3ns]. Thus, the proposed FCU has a larger operative range of about $\Delta T_{FCCvsFCU} = 1.7ns$ than the FCC [16]. As expected, the DW_Mul unit has the largest violation free timing range of [1.6ns, 5.3ns], among the other two flexible components. The rather small $\Delta T_{FCUvsMUL} = 0.8ns$ between the proposed flexible computational unit and the optimized non-flexible DW_Mul unit shows the efficiency of our approach. Additionally, comparing the area of the two flexible components at 3.9 ns (the upper operative point of the FCC unit), the proposed FCU delivers approximately $\times 6$ smaller area than the FCC. The comparison of the proposed FCU and the DW_Mul at the upper operative point of FCU (T = 2.2 ns) shows that

FCU occupies about $\times 3$ larger area than DW_Mul, at that specific point. However, Fig. 5.11 shows that for operative points larger than 4 ns the area of the FCU converge towards the optimized non-flexible DW_Mul.

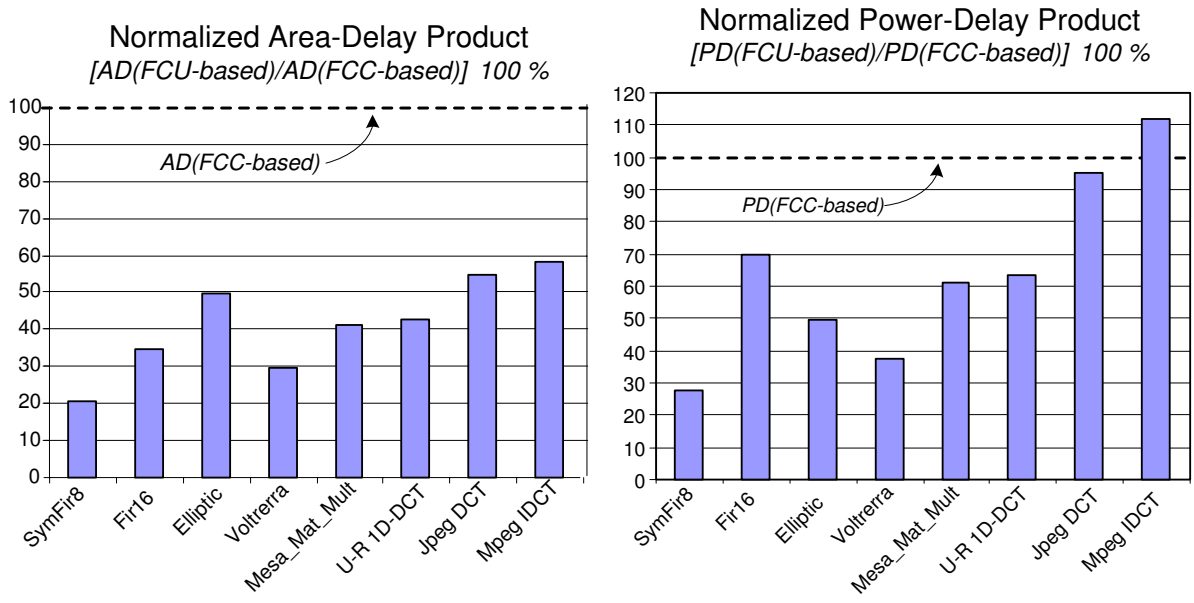
A representative set of computationally intensive DSP kernels was formed in order to demonstrate the efficiency of the proposed solution. The benchmark suite consists of: (i) an 8-taps Symmetrical FIR filter (SymFir8), (ii) a 16-taps FIR filter (Fir16), (iii) a 6th order Elliptic filter (Elliptic), (iv) a Volterra IIR filter, (v) the MESA Matrix Multiplication (Mesa Mat_Mul) kernel [100], (vi) a straightforward 1-D DCT kernel with unrolled the column's loop (U-R 1D-DCT), (vii) the 2-D DCT (Jpeg DCT) used in JPEG [100] and (viii) the 2-D Inverse DCT (Mpeg DCT) used in MPEG [100].

These kernels were mapped onto a FCU-based architecture comprising 4 FCUs and onto a FCC-based datapath with 2 FCCs (= 8 ALUs and 8 Muls) [16]. The SymFir8 and Fir16 have been unrolled 4 times, while each loop of the Volterra filter has been fully unrolled according to the synthesis methodology of Section 5.5. The Mesa Mat_Mul, U-R 1D-DCT and Jpeg DCT kernels were not unrolled since in each iteration more than 4 multiplication operations were available. For the mapping onto the FCC-based architecture, the above benchmarks were scheduled using the SPARK HLS tool [67] with the aggressive operation chaining option enabled and we manually optimized the resultant FCC-based datapath in order to take into consideration the inter-template chaining, which is not supported by SPARK. We omitted comparative results between the proposed approach and datapaths composed only by primitive operators. However, such a comparison has been conducted in [16] for the case of FCC-based datapath. Since we are compared with the approach in [16], some straightforward qualitatively conclusions about the performance efficiency of our approach in comparison with primitive resource datapaths can be safely inferred.

The FCU-based and FCC-based datapaths were synthesized with Synopsys Design Compiler and TSMC 0.13um technology library. For the proposed FCU-based datapath a timing constraint of $T_{clk_{FCU}} = 3.8ns$ was imposed while for the FCC-based datapath the timing-constraint was set to $T_{clk_{FCC}} = 4.8ns$ (the middle values between the upper and lower operative points of Fig. 5.11, for each flexible component). Power analysis of the synthesized netlists performed with Synopsys PrimePower [135]. Worst case power analysis was considered by imposing $ToggleRate = 1$ to all the inputs and the internal nets of the synthesized netlists. Table 5.5 reports (i) the actual latency ($\#cycles \times T_{clk}$) in ns, (ii) the active area in um^2 and (iii) the power dissipation in $mWatt$ for each DSP kernel. The proposed datapath delivers faster implementations with smaller area complexity than the FCC-based datapaths in all cases. Specifically, the proposed datapath delivers average latency and area reductions of 29.5% and 42.1%, respectively. The average power consumption is 10.1% lower for FCU-based datapaths. In some kernels the power consumption of the proposed datapath is larger than the FCC-based datapath. This occurs due to the higher operating frequency of FCU datapath in comparison to the FCC

Table 5.5.: Latency, Area and Power Consumption Results: FCU vs FCC [16] Datapaths.

	Proposed FCU			***FCC [16]***			***Gains (%)***		
DSP Kernel	Latency (ns) (Cycles×$T_{clk_{FCU}}$)	Area (um^2)	Power (mWatt)	Latency(ns) (Cycles×$T_{clk_{FCC}}$)	Area (um^2)	Power (mWatt)	Latency (%)	Area (%)	Power (%)
SymFir8	15.2	75124.5	7.3	28.8	190252.6	13.9	47.2	60.5	47.5
Fir16	22.8	94108.4	10.0	38.4	161715.3	8.5	40.6	41.8	-17.6
Elliptic	22.8	127900.4	11.2	28.8	203526.8	17.8	20.8	37.2	37.1
Voltrerra	19	99462.8	8.8	33.6	189059.9	13.3	43.4	47.4	33.8
Mesa Mat_Mul	79.8	103862.1	13.3	105.6	189404.7	16.4	24.4	45.2	18.9
U-R 1D-DCT	505.4	98705.3	10.6	614.4	189742.7	13.7	17.7	47.9	22.6
Jpeg DCT	497.8	198984.1	47.1	657.6	275579.8	37.4	24.3	27.8	-25.7
Mpeg IDCT	513	199757.3	50.5	624	280835.5	37.1	17.8	28.9	-36.11
Average	-	-	-	-	-	-	29.5	42.1	10.1



(a) (b)

Figure 5.12.: Designs Metrics a) Area-Delay b) Power-Delay.

datapath. However, the small area complexity (small load capacitance) of FCU-based datapaths amortize the power effect of high operating frequency in most DSP kernels.

We considered two design metrics, namely the Area-Delay product (ADP) and the Power-Delay product (PDP), in order to evaluate the efficiency of the synthesized datapaths. The ADP and PDP values (the lower the better) have been normalized according to the respective values of each benchmark for the FCC case [16]. Thus, the top dashed line (value 100%) represents the corresponding ADP or PDP of the FCC datapath. FCU datapath solution outperforms the FCC one, in all cases and for all design metrics, except the PDP value for the MPEG IDCT kernel. The ADP values of FCU-based datapaths lay between 20.8%-58.4%. The PDP and EDP values range between 27.7%-111% and 14.6%-92%, respectively.

5.7. Conclusion

In this chapter, we presented a synthesis methodology for flexible datapaths incorporating tree-based multiplication units. The proposed methodology relies on the instantiation of flexible and arithmetically optimized architectural templates. Experimental results on several DSP kernels have shown average performance, area

and power improvements of about 30%, 42% and 10% respectively, over previously published high performance and flexible datapath solution.

6. Conclusions and Future Extensions

> **There are two possible outcomes: if the result confirms the hypothesis, then you've made a measurement. If the result is contrary to the hypothesis, then you've made a discovery.**
>
> ***Enrico Fermi***

This chapter lists the main conclusions drawn by this dissertation, summarizes the thesis innovations and presents possible future extensions. The introduced innovations enable the design of optimized embedded systems offering synthesis methodologies for customization of the software and the hardware components. The proposed solutions targeted to critical problems faced during system design, thus their adoption makes possible the development of new design methodologies for the next generation of computing devices.

The remainder of this chapter is organized in the following manner. Section 6.1 presents the innovations and the contributions of this PhD thesis in the field of embedded system design. In Section 6.2, we summarize the main conclusions drawn by applying the proposed methodologies onto specific case studies. Possible future extensions are reported in Section 6.3. Finally, Section 6.4 presents the scientific publications emerged during the preparation of the specific dissertation.

6.1. Introduced Innovations

This section summarizes the innovations of this thesis. They are categorized into five classes, Each class refers to the solutions proposed for each of the five targeted design problems discussed in Section 1.6. In the rest of this section, we report, in brief, the targeted problems and we highlight the innovations proposed for each of them in order to be efficiently tackled.

Problem 1: There is lack of efficient approaches regarding application-specific dynamic memory management for multi-threaded applications. The following innova-

tions have been proposed:

- **Definition of a unified design space for multi-threaded dynamic memory management.**
- **Analysis of the design decisions regarding dynamic memory management issues in multi-threaded environments.**
- **Categorization and parametrization of the design decisions into inter-heap and intra-heap decisions regarding their scope of interest.**
- **Dependence, semantic and cost analysis between the design decisions.**
- **Definition of valid design decision orderings according to the targeted optimization objective.**
- **Development of a new software library enabling the modular composition of various dynamic memory managers.**
- **Development of efficient exploration strategy based on constraint-orthogonal design space partitioning to handle the advanced complexity of the design space.**
- **Development of heuristic pruning algorithms in the parameter space according to specific optimization objectives (memory footprint, memory accesses).**
- **Pareto analysis and automatic software synthesis of Pareto customized multi-threaded dynamic memory management solutions.**

Problem 2: Existing architectural synthesis approaches do not take into account trade-offs produced by the combined impact of architecture level optimizations together with behavioral level optimizations. The following innovations have been proposed:

- **Definition of an extended design space for architectural synthesis, unifying design decisions found in the compiler- and architecture-level.**
- **Analysis of the design decisions regarding Delay-Area trade-offs.**
- **Definition of a valid decision ordering for design space decomposition targeting the Delay-Area optimization objectives.**

- **Upper bounding condition definitions for each of the generated decomposed design sub-spaces.**

- **Detailed and parameterized area modeling of datapath architecture templates.**

- **Development of gradient-based heuristic pruning algorithm with amortization mechanisms applied during the searching of the solution space.**

- **Automatic synthesis of Delay-Area Pareto-optimal coprocessor datapaths.**

Problem 3: Existing coarse-grained reconfigurable architectures take into account only sharing at the Functional Unit level, neglecting the optimization opportunities delivered by combine sharing at the Functional Unit level and the bit-level. The following innovations have been proposed:

- **Identification of the bit-level structural symmetries between the circuits implementing addition, subtraction and multiplication.**

- **Introduction of the Uniformity transformation for revealing common interconnection structures among the targeted hardware operators.**

- **Functionality merging through logic synthesis for area-delay optimized bit-level reconfigurable cells.**

- **Introduction of the Flexibility Inlining technique for mapping chain addition/subtraction hardware operators onto the structure of array multipliers (combining Uniformity transformation and bit-level functional sharing).**

- **Specification of a new coarse-grained reconfigurable architectural template utilizing flexible functional units designed using the Flexibility Inlining technique.**

- **Micro-architectural analysis and optimization of the proposed reconfigurable datapath for supporting horizontal, vertical parallelism and operation chaining in a combined manner.**

Problem 4: Existing coarse-grained reconfigurable architectures do not take into account arithmetic level optimizations during architecture specification. The following innovations have been proposed:

- **Introduction of the Carry-Save arithmetic representation in the flexible functional units generated by the Flexibility Inlining technique for delay optimization.**

- **Architecture specification of a new coarse-grained reconfigurable datapath exploiting Carry-Save operation templates with tree-based multipliers.**

- **Utilization of recoding techniques and their hardware implementation for enabling seamless computations in Carry-Save arithmetic representation.**

- **Micro-architectural analysis and optimization of the new reconfigurable datapath for supporting horizontal parallelism and aggressive operation chaining in a combined manner.**

Problem 5: Productivity and strict time-to-market requirements lead to the non adoption of new optimized architectural templates based on complex resource models. The following innovations have been proposed:

- **Introduction of new architectural synthesis frameworks targeting each of the proposed reconfigurable datapaths to make transparent to the designer the complex resource models of the introduced reconfigurable architectures.**

- **Micro-architectural analysis and abstraction of the introduced flexible computational units building the corresponding resource libraries.**

- **Development of pre-synthesis optimization passes for transforming the intermediate representations of the initial behavioral descriptions in equivalent ones regarding the underlying resource models.**

- **Fine-tuning of existing resource constrained scheduling algorithms in order to take into account the new complex resource models exhibited by the introduced reconfigurable datapaths, exploiting horizontal parallelism and operation chaining features.**

- **Development of the adjacency-aware operation binding algorithm for maximal exploitation of vertical parallelism in reconfigurable datapaths based on Flexibility Inlining.**

6.2. Main Conclusions

In this section, we summarize the main conclusions drawn by the appliance of the methodologies, proposed in this thesis, onto the examined use cases. We verified the conclusion reported in many research activities, that there is a strong dependence between the design algorithms and the efficiency of the solution. However, the most basic outcome of this thesis is that the efficiency of the design solution is high correlated not only to the design algorithms but also to both the completeness of the exploration parameters and the interactions between the various abstraction levels. While in the common case, design algorithms propose efficient techniques to guide through a specific abstraction layer, this limits the set of the explored parameters, thus not leading to globally optimized solutions due to the fine-grained fragmentation of the design space. In order to alleviate this inefficiency, design algorithms have to evolve to take into account more complex parameter spaces and optimization opportunities found in more than one abstraction layers. Of course, the development of design algorithms that applies onto a completely de-fragmented design space is not possible yet, regarding the huge parameter and solution spaces and the computational capabilities of current computing devices. Thus, this thesis proposes techniques that enables exploration and optimization algorithms targeting to not a completely de-fragmented but to a design space with coarser fragmentation than the previous approaches.

More specifically, regarding the dynamic memory management in multi-threaded applications, we showed that there is a strong dependence between the architecture of a dynamic memory manager and the cost metrics, i.e. memory footprint, number of memory accesses, code size etc. Thus, when the application sets are a-priori known to the designer, as in the case of embedded systems, the customization of the software structure of the dynamic memory manager leads to optimized solutions. In addition, the structure of the dynamic memory manager can be customized regarding a large set of design decisions, playing the role of exploration parameters. These design decisions are laid across two parameter subspaces, namely the inter-heap and intra-heap design decisions, the one handling decisions in a global manner, while the other one managing customization in a local fashion. We showed that there is a high correlation among these design parameters even between the two parameters spaces and that these correlations have to be taken into account for establishing proper decision orderings according to the targeted optimization objective. Given the huge design space, these correlations can significantly aid the exploration's automation. They can be viewed as semantic constraints that are propagated within the design space and exploited for the development of efficient early pruning of the semantically non-valid solutions, reducing the explored solution space without compromising the solutions quality. Furthermore, the various decision orderings are exploited for the development of efficient heuristics for traversing the design space in affordable runtime. Finally, experimental data showed that the solution space, resulted during exploration for efficient memory footprint versus memory accesses trade-offs, is very vague which make extremely difficult even for

a expert designer to find Pareto optimal solutions without the aid of automation tools.

For the fundamental Delay-Area exploration problem during architectural synthesis of hardware accelerators, we identify the strong correlation existing between the parameter spaces of compiler- and architectural-level decisions. Although compiler-level transformations have been extensively studied in the field of architectural synthesis, they were usually explored either as user-defined parameters or in a stand-alone manner without interacting with the architectural synthesis decisions. We showed that these approaches silently neglecting the trade-offs produced from the combined impact of behavioral-level together with architectural-level parameters, leading to a significant unexplored portion of the solution space that also includes the Pareto-optimal configurations. In order to tackle this inefficiency, we formed an enhanced instance of the design space where both code transformations and architectural optimizations are treated as exploration parameters. In contrast with the design space considered for dynamic memory management, the specific design space does not propagates so strong dependencies among its parameters to enable early identification of semantically non-valid solutions. Due to this feature the exploration strategy has to be applied onto the solution space rather than the parameter one. However, even in the case of loose dependencies at the parameter space, we showed that proper decision ordering can be used in order to decompose the huge solution space in many smaller ones that can be handled more efficiently. Based on this solution space decomposition, we proved that upper bounding conditions exists both for the overall but also for partial solution subspaces. In addition, we showed that the decomposed solution space follows a well defined structure that can be exploited during exploration for prune design points based on a newly introduced gradient-based heuristic. Experimental evaluation based on real-life benchmarks indicated a shift of the approximated Pareto curve towards higher quality solutions in comparison to the existing design exploration methodologies with efficient convergence to the global Pareto-front and with significant speedup gains (up to 8$\times$ faster) in comparison to exhaustive exploration of the newly introduce design space.

Regarding the design of coarse-grained reconfigurable architectures we studied the optimization possibilities exposed during architecture specification when enhancing the designer visibility to both the architectural- and circuit-level of abstraction. We showed that the circuit level view of the reconfigurable architecture enables a combined functional unit and bit-level sharing of the resources leading to more efficient flexible circuits in terms of occupied area and utilization of the underlying hardware. In addition, it enables the appliance of arithmetic level optimizations to be incorporated during architecture design for high performance datapaths. From this study, two new flexible architectural templates have been introduced targeting the Digital Signal Processing application domain. Linking the bit-level arithmetic level optimizations with the operation chaining architectural optimization, the design of flexible computational units with high speed, low area and high operation density characteristics has been enabled. In addition, by identifying the high perfor-

mance characteristics of Carry-Save arithmetic representation, arithmetic recoding techniques have been incorporated during architecture specification to permit the computation to be seamlessly performed in the specific arithmetic representation. Furthermore, we showed that the joint exploitation of multiple architectural optimizations i.e. horizontal parallelism, vertical parallelism, operation chaining, with arithmetic level optimizations is a viable solution for coarse-grained reconfigurable datapaths that leads to efficient implementations. This collaborative view of architectural and circuit abstraction layers, came along with increased complexity in the programming models supported by the introduced datapaths. Although existing architectural synthesis tools are very efficient for datapath architectures builded from functional unit resource models that match the resource model of conventional processors, they fail to lead to optimized solutions for datapath architectures that adopt more complex resource models. We proposed to manage this programming complexity at high abstraction levels in order to be able to reuse well-established algorithms and tools coming from the field of architectural synthesis. The specific problem have been addressed through proper "source-to-source" transformations developed to perform the mapping of the high-level (compiler) intermediate representation onto the micro-architectural abstractions of the introduced datapaths structures. Experimental evaluation at the circuit level show that the proposed flexible architectures are able to operate on a significantly larger range of operating frequencies and with lower area overheads than the existing ones. Based on a representative set of DSP kernels, we show that the proposed approach delivers an significant average improvements both in execution latency and area reduction, compared with state-of-the-art reconfigurable datapath solutions.

6.3. Future Extensions

This PhD thesis proposed a set of novel design methodologies targeting to aid designers to move towards optimized system solutions. The design tools implementing the proposed methodologies and the theoretical and quantitative conclusions drawn within this thesis can be used as a solid basis for addressing new design problems resulted from the new technological challenges. In the rest of this section, we summarize some of the possible research extensions in respect with the alternative research activities performed in our laboratory.

• Combined Design Space Exploration of Design Time Customized and Run-time Adaptive Multi-Threaded Dynamic Memory Management With Platform Architectural Level Parameters.

In this thesis, we efficiently addressed the problem of generating application-specific

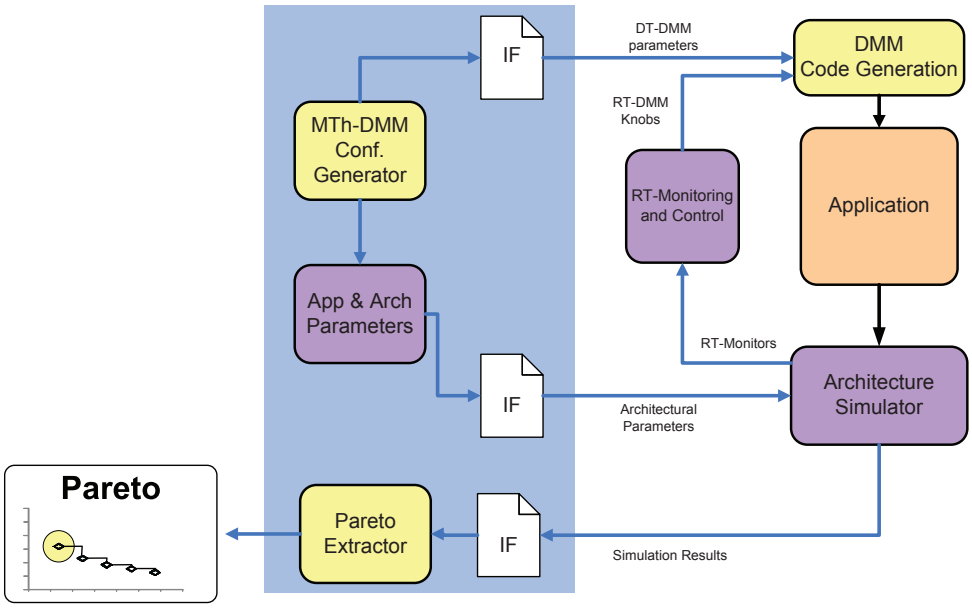


Figure 6.1.: Runtime Adaptive and Platform Customized Dynamic Memory Management.

dynamic memory managers for multi-threaded environments targeting MPSoC platforms. Based on the technology developed so far, we propose its extension towards two high correlated directions.

The first direction refers to the runtime adaptivity of the dynamic memory manager in order to be able to adapt to the user and or platform level variations during runtime. Thus, the dynamic memory manager has not only to be customized to a specific application but also to be enhanced with proper mechanisms that will enable the software module to (i) sense the actual user and platform variations, (ii) to analyze the monitored data, (iii) take an optimal decision regarding the new configuration and (iv) reconfigure the software module implementing the management. From some initial evaluations performed until now, allowing the full run-time switching between the available decisions in the parameter space is rather unrealistic due to un-affordable runtime overheads that the switching will impose. Thus, the parameter space presented so far for customized multi-threaded dynamic memory management, has to be further analyzed in order to evaluate the runtime overheads when a specific decision is reconfigured. Overhead analysis will indicate the set of parameters that exhibit lightweight reconfiguration overheads. The modular software library have to be extended in order to incorporate the reconfiguration mechanisms. Furthermore, new decision strategies have to be incorporated to enable the run-time tuning of a specific configuration.

The second direction refers to the combined evaluation of the dynamic memory design-time parameters with the architectural parameters of the underlying platform in order to move towards more customized system solutions. The exploration

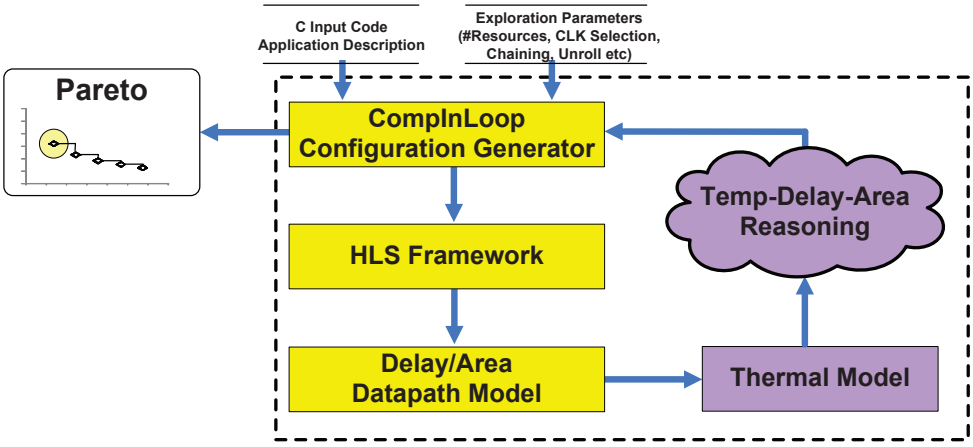


Figure 6.2.: Thermal Aware Compiler-in-Loop Exploration.

approach proposed in this thesis assumed a fixed platform architecture with specific cache sizes, memory bandwidth etc. Combined exploration of the platform dependent architectural parameters with the platform independent ones will reveal the qualitative interaction between the software policies and their impact on the actual hardware, leading to globally customized system solutions regarding not only dynamic memory cost functions but also cost functions regarding the overall platform.

Fig. 6.1 depicts an abstract view of the proposed future extensions, in which both design time customization and runtime tuning of the dynamic memory manager have been integrated.

• Thermal Aware Compiler-in-Loop Exploration Targeting Coprocessor Design in Deep Sub-Micron Technologies.

The proposed Compiler-in-Loop exploration approach targeted to the Delay-Area trade-offs exposed during architectural synthesis. However, the continuous scaling in VLSI process technologies came along not only with increased logic densities but also with increased power budgets (especially for deep sub-micron process technologies < 65 nm). This significant increment of the on-chip power-densities leads to non-uniform temperature profiles across the die due to the power to heat conversion effect. This non-uniform temperature profiles heavily affects both the circuit's performance and reliability i.e. switching speed of transistors decreases, electromigration accelerates etc. Based on this technology projection, we propose the extension of the Compiler-in-Loop exploration approach towards its thermal aware variant.

A possible extension of the already developed exploration framework, is depicted in Fig. 6.2. The current version of the Compiler-in-Loop exploration consists of the three yellow blocks that is the configuration generator, the HLS engine which performs operation scheduling and binding and an abstract delay-area model of the targeted datapath architecture. The goal of the thermal aware variant of the Compiler-in-Loop exploration methodology is to evaluate a extended design space which takes into account also design decisions regarding the lower abstraction levels which are highly correlated with thermal issues in order to provide to the designer information regarding the impact of the compiler- and architectural-level optimization onto the thermal profile of the device. Towards this direction two main additions (the purple boxes in Fig. 6.2) have to been considered in the already proposed exploration flow. The first addition refers to an accurate thermal model of the underlying hardware architecture which will enable thermal analysis to be conducted. A compact thermal model has been proposed in the Hotspot temperature analysis tool [156], which models heat flow based on its RC thermal equivalent. Such type of thermal models can be adopted also in the proposed extended exploration tool. The second and more important addition is the development of a unified delay-area-power-temperature reasoning methodology in order to capture the correlations between the compiler-, architectural- and thermal-parameters and reason on this type of correlations. High level thermal reasoning module will be responsible for guiding the configuration generation module to provide a configurations which optimize the targeted trade-offs. From some early evaluations seemed that gradient-based approaches are not adequate to capture the thermal aware architectural correlations. Thus, analytical models and/or statistical techniques can be employed to tackle the specific reasoning problem. However, recent advances in the field of compiler optimization have reported promising results on using machine-learning techniques for automatic compiler tuning [157]. Such type of techniques can be extended or fine tuned for thermal aware architectural reasoning.

• Exploring the Embedding of Coarse-Grained Reconfigurable Components into FPGA platforms.

The coarse-grained reconfigurable architectures proposed in this thesis targets to configurable ASIC MPSoC systems by providing Delay-Area efficient flexible co-processor modules to accelerate the computationally intensive software tasks. The continuous advances in FPGA technologies made possible the placement of MPSoC circuits onto a single FPGA device. In order to reduce the performance gap between FPGA-based versus ASIC-based MPSoCs, FPGA providers moved towards heterogeneous FPGA architectures. Specifically, ASIC hardware blocks implementing some computationally intensive functions, usually called DSP blocks, have been incorporated inside the FPGA devices. However, these DSP blocks are limited to very specific arithmetic functions i.e. multipliers, Multiplier-Accumulator (MAC) functions etc. Thus, only limited logic is mapped onto these DSP blocks. Taking into consideration this inefficiency of current heterogeneous FPGAs, we propose an alternative to the DSP blocks that are employed today based on the incorporation of coarse-grained reconfigurable blocks in heterogeneous FPGAs as

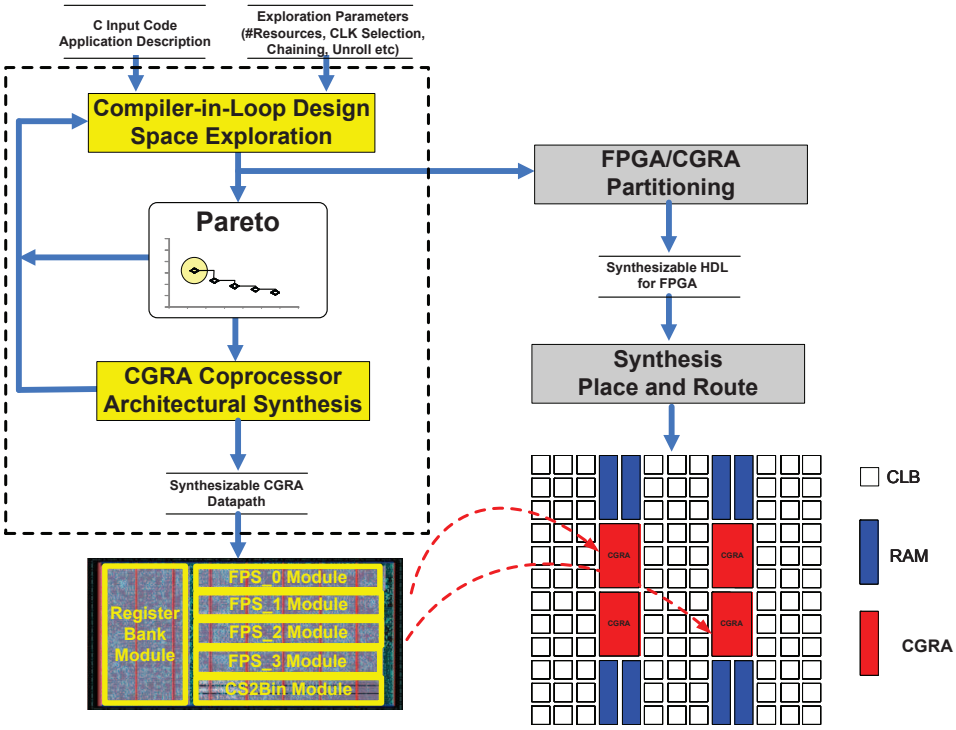


Figure 6.3.: Mapping hardware coefficients on heterogeneous FPGA structures.

a possible future extension of our work in coarse-grained reconfigurable architectures.

The incorporation of the proposed coarse-grained reconfigurable architectures in FPGA fabrics generates some interesting trade-offs between application partitioning, architectural-level and physical level exploration. At the application partitioning level, the existence of coarse-grained together with fine-grained configurable blocks requires customized co-design methodologies since the application has now to be partitioned in three coefficients (software/fine-grained reconfigurable hardware/coarse-grained reconfigurable hardware). At the architectural-level, the coarse-grained reconfigure coefficients can be manipulated by the exploration and synthesis tools proposed within this thesis. However, there exists an open question on where to place the barrier between the fine- and coarse-grained logic. At the physical level, the conventional placement and routing algorithms targeting homogeneous FPGA structures have to enhanced to take into consideration blocks of coarser granularity and the non-uniform distribution of the fine grained configurable cells of the FPGAs on the device. Also, the type and number of routing tracks have to be explored in order to result to a routing architecture that can support the new architectural templates. Fig. 6.3 depicts an abstract schematic visualizing the mapping of the hardware coefficients onto the heterogeneous FPGA fabric. Some exiting work targeting the aforementioned design issue has been proposed in [48], [158], [159], however, they target either specific design subspaces

i.e. the architectural synthesis subspace [48], [158], or the physical-optimizations subspace [159].

6.4. Publications

This section lists the published work delivered during the elaboration of this PhD thesis. Overall, this thesis resulted to thirty publications until now, among them two journal publications, four book chapter publications and twenty-four publications to peer-reviewed international conferences. More specifically:

International Journals:

- J3. Sotirios Xydis, Kiamal Pekmestzi, Dimitrios Soudris, George Economakos, "Compiler-in-the-Loop Exploration During Datapath Synthesis for Higher Quality Delay-Area Trade-offs", under preparation.

- J2. Sotirios Xydis, George Economakos, Dimitrios Soudris, Kiamal Pekmestzi, "High Performance and Area Efficient Flexible DSP Datapath Synthesis", article in press IEEE Transactions on Very Large Scale (VLSI) Systems, doi: 10.1109/TVLSI.2009.2034167 .

- J1. Sotiris Xydis, George Economakos, Kiamal Pekmestzi, "Designing Coarse-Grain Reconfigurable Architectures by Inlining Flexibility into Custom Arithmetic Datapaths", Elsevier Integration the VLSI Journal, vol. 42, no. 4, pp. 486-503, 2009, doi:10.1016/j.vlsi.2008.12.003 .

Chapters in Scientific Books:

- B1. Sotirios Xydis, Kiamal Pekmestzi, Dimitrios Soudris, George Economakos, "A High Level Synthesis Exploration Framework with Iterative Design Space Partitioning," Book Chapter in "Designing Very Large Scale Integration Systems: Emerging Trends Challenges" Editors: N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, M. Huebner, Springer 2011.

- B2. Bernard Candaele, Sylvain Aguirre, Michel Sarlotte, Iraklis Anagnostopoulos, Sotirios Xydis, Alexandros Bartzas, Dimitris Bekiaris, Dimitrios Soudris, Zhonghai Lu, Xiaowen Chen, Jean-Michel Chabloz, Ahmed Hemani, Axel Jantsch, Geert Vanmeerbeeck, Jari Kreku, Kari Tiensyrja, Fragkiskos Ieromnimon, Dimitrios Kritharidis, Andreas Wiefrink, Bart Vanthournout, Philippe Martin, "The MOSART Mapping Optimisation for multi-core ARchiTectures", Book Chapter in "Designing Very Large Scale Integration Systems: Emerging Trends Challenges" Editors: N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, M. Huebner, Springer 2011.

- **B3. C. Silvano, W. Fornaciari, S. Crespi Reghizzi, G. Agosta, G. Palermo, V. Zaccaria, P. Bellasi, F. Castro, S. Corbetta, A. Di Biagio, E. Speziale, M. Tartara, D. Siorpaes, H. Hubert, B. Stabernack, J. Brandenburg, M. Palkovic, P. Raghavan, C. Ykman-Couvreur, A. Bartzas, S. Xydis, D. Soudris, T. Kempfk, G. Ascheidk, R. Leupersk H. Meyrk, J. Ansarik, P. Mahonenk, and B. Vanthournout, "2PARMA: Parallel Paradigms and Run-time Management Techniques for Many-Core Architectures", Book Chapter in "Designing Very Large Scale Integration Systems: Emerging Trends Challenges" Editors: N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, M. Huebner, Springer 2011.**

- **B4. S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, "Application Specific Multi-Threaded Dynamic Memory Management", Book Chapter in "Scalable Multi-core Architectures: Design Methodologies and Tools" Editors: Axel Jantsch and Dimitrios Soudris, Springer 2011.**

International Peer-Reviewed Conferences:

- **C24. S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, K. Pekmestzi, "Custom Mutli-Threaded Dynamic Memory Management for Multiprocessor System-on-Chip Platforms", in Pro-ceedings of the IEEE Embedded Computer Systems: Architectures, Modeling and Simulation (ICSAMOS'2010), Samos, Greece, pages 102-109, 2010.**

- **C23. S. Xydis, C. Skouroumounis, Kiamal Pekmestzi, Dimitrios Soudris, George Economakos, "Efficient High Level Synthesis Exploration Methodology Combining Exhaustive and Gradient-Based Pruned Searching", in Proceedings of the IEEE Annual Symposium on VLSI (ISVLSI'2010), Lixouri-Kefallonia, Greece, pages 104-109, 2010.**

- **C22. S. Xydis, Kiamal Pekmestzi, Dimitrios Soudris, George Economakos, "High-Level Synthesis Methodologies for Delay-Area Optimized Coarse-Grained Reconfigurable Coprocessor Architectures", in Proceedings of the IEEE Annual Symposium on VLSI (ISVLSI'2010), PhD Forum, Lixouri-Kefallonia, Greece, pages 486-487, 2010.**

- **C21. Bernard Candaele, Sylvain Aguirre, Michel Sarlotte, Iraklis Anagnostopoulos, Sotirios Xydis, Alexandros Bartzas, Dimitris Bekiaris, Dimitrios Soudris, Zhonghai Lu, Xiaowen Chen, Jean-Michel Chabloz, Ahmed Hemani, Axel Jantsch, Geert Vanmeerbeeck, Jari Kreku, Kari Tiensyrja, Fragkiskos Ieromnimon, Dimitrios Kritharidis, Andreas Wiefrink, Bart Vanthournout, Philippe Martin, "Mapping Optimisation for Scalable multi-core ARchiTecture: The MOSART approach", in Proceedings of the IEEE Annual Symposium on VLSI (ISVLSI'2010), Lixouri-Kefallonia, Greece, pages 518-523, 2010.**

- **C20. C. Silvano, W. Fornaciari, S. Crespi Reghizzi, G. Agosta, G. Palermo, V. Zaccaria, P. Bellasi, F. Castro, S. Corbetta, A. Di Biagio, E. Speziale, M. Tar-**

tara, D. Siorpaes, H. Hubert, B. Stabernack, J. Brandenburg, M. Palkovic, P. Raghavan, C. Ykman-Couvreur, A. Bartzas, S. Xydis, D. Soudris, T. Kempfk, G. Ascheidk, R. Leupersk H. Meyrk, J. Ansarik, P. Mahonenk, and B. Vanthournout, "2PARMA: Parallel Paradigms and Run-time Management Techniques for Many-Core Architectures", in Proceedings of the IEEE Annual Symposium on VLSI (ISVLSI'2010), Lixouri-Kefallonia, Greece, pages 494-499 2010.

- **C19. S. Xydis, C. Skouroumounis, K. Pekmestzi, D. Soudris, G. Economakos, "Designing Efficient DSP Datapaths Through Compiler-in-the-Loop Exploration Methodology", in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Paris, France, pages 2598-2601, 2010.**

- **C18. G. Economakos and S. Xydis, I. Koutras, D. Soudris, "Construction of Dual Mode Compo-nents for Reconfiguration Aware High-Level Synthesis", in Proceedings of the IEEE/ACM Inter-national Conference Design Automation and Test in Europe (DATE 2010), Dresden, Germany, pages 1357-1360, 2010.**

- **C17. Sotirios Xydis, Iraklis Anagnotopoulos, Alexandros Bartzas, Dimitrios Soudris, "Dynamic Memory Management Customization for Multi-Processor Systems-on-Chip", in Workshop on Designing for Embedded Parallel Computing Platforms: Architectures, Design Tools, and Appli-cations DATE 2010.**

- **C16. Sotirios Xydis, Kiamal Pekmestzi, Dimitrios Soudris, George Economakos, "High Performance Flexible Coprocessor Synthesis", in PhD Forum of DATE 2010.**

- **C15. Dimitris Bekiaris, Sotiris Xydis, George Economakos and Kiamal Pekmestzi, "A Design Methodology for High-Performance and Low-Leakage Fixed-Point FIR Filters", in Proceedings of the 16th International Conference on Electronics, Circuits and Systems (ICECS), Hammamet, Tynisia, December 13-16, 2009.**

- **C14. G. Economakos and S. Xydis, "Optimized Reconfigurable RTL Components for Performance Improvements During High-Level Synthesis", in Proceedings of EUROMICRO 09 Conference, Symposium on Digital System Design (DSD09), Patras, Greece 2009, pp 164-171.**

- **C13. Sotirios Xydis, Ioannis Triantafyllou, George Economakos, Kiamal Pekmestzi, "Flexible Datapath Synthesis Through Arithmetically Optimized Operation Chaining", in Proceedings of of IEEE NASA/ESA International Conference on Adaptive Hardware and Systems (AHS 2009), pages 407 - 414, July 29 - Aug. 1, 2009.**

- **C12. George Economakos, Sotiris Xydis, "High-Level Synthesis with Coarse Grain Reconfigurable Components", in Proceedings of IEEE International**

Symposium on International Parallel Distributed Processing 2009 (IPDPS 2009), pages 1-4, May 23-29, 2009.

- **C11. Sotiris Xydis, George Economakos, Dimitrios Soudris, Kiamal Pekmestzi, "Joint Exploitation of Horizontal/Vertical Parallelism and Operation Chaining for Flexible DSP Synthesis", in Proceedings of International Conference on Very Large Scale Integration (VLSI-SoC 2008), pages 501-504, October 13-15, Rhodes Island, Greece, 2008.**

- **C10. George Economakos, Sotiris Xydis, "A Synthesis Postprocessor for Fully Morphable RTL Datapaths", in Proceedings of International Conference on Very Large Scale Integration (VLSI-SoC 2008), pages 457-462, October 13-15, Rhodes Island, Greece, 2008.**

- **C9. George Economakos, Sotiris Xydis, "A Scheduling Postprocessor to Exploit Morphable RTL Components During High Level Synthesis", in Proceedings of 11th Euromicro Conference on Digital System Design (DSD'2008), pages 494-499, September 3-5, Parma, Italy, 2008.**

- **C8. Sotiris Xydis, Isidoros Sideris, George Economakos, Kiamal Pekmestzi, "A Flexible Architecture for DSP Applications Combining High Performance Arithmetic with Small Scale Configurability", in Proceedings of 16th European Signal Processing Conference (EUSIPCO-2008), August 25-29, Lausanne, Switzerland, 2008.**

- **C7. Sotiris Xydis, George Economakos, Dimitrios Soudris, Kiamal Pekmestzi, "Mapping DSP Applications onto High Performance Architectural Templates with Inlined Flexibility", in Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2008), pages 346-354, 2008.**

- **C6. George Economakos, Sotiris Xydis, "High-Level Synthesis Heuristics for Run-Time Reconfigurable Architectures", in Proceedings of 15th European Signal Processing Conference (EUSIPCO-2007), September 3-7, Poznan, Poland, 2007.**

- **C5. Sotiris Xydis, George Economakos, Kiamal Pekmestzi, "A Regular Interconnection Scheme for Efficient Mapping of DSP Kernels Into Reconfigurable Hardware", in Proceedings of 15th European Signal Processing Conference (EUSIPCO-2007), September 3-7, Poznan, Poland, 2007.**

- **C4. Sotiris Xydis, George Economakos, Kiamal Pekmestzi, "A Reconfigurable Arithmetic Data-path Based On Regular Interconnection", in Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2007), August 5-8, Edinburgh, Scotland-United Kingdom, 2007 (Best paper award in Reconfigurable Systems category).**

- **C3. Sotiris Xydis, George Economakos, Kiamal Pekmestzi, "Flexibility Inlining into Custom Arithmetic Data-paths Exploiting A Regular Interconnection Scheme", in Proceedings of 5th International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS VII) July 16-19, Samos, Greece, 2007.**

- **C2. George Economakos, Christoforos Economakos, Sotiris Xydis, "Run-Time Reconfigurable Solutions for Adaptive Control Applications ", in Proceedings of 5th International Conference on Informatics in Control, Automation and Robotics (ICINCO-2007), May 11-15, Funchal, Madeira-Portugal, 2007.**

- **C1. G. Economakos, S. Xydis, "Behavioral Synthesis of Run-Time Reconfigurable Networking Devices", in Proceedings of 5th International Symposium of Communication Systems, Networks and Digital Signal Processing, 19-21 July, Patras, 2006.**

Awards:

- **1. Best Paper Award at IEEE/NASA/ESA International Conference on Adaptive Hardware and Systems (AHS 2007).**

- **2. PhD Forum presentation at IEEE International Conference on Design Automation and Test in Europe (DATE 2010).**

- **3. University Booth at IEEE International Conference on Design Automation and Test in Europe (DATE 2010).**

- **4. PhD Forum presentation at IEEE International Symposium on VLSI (ISVLSI'2010).**

- **5. ISVLSI'2010 invitation for book chapter contribution of the paper "S. Xydis, C. Skouroumounis, K. Pekmestzi, D. Soudris, G. Economakos, Efficient High Level Synthesis Exploration Methodology Combining Exhaustive and Gradient-Based Pruned Searching"**

Glossary

μP	Micro-processor, 66
1D-DCT	Single Dimensional Discrete Cosine Transform, 119
AC	Adder or Full Adder Cell, 156
ADP	Area Delay Product, 203
ALAP	As Late As Possible, 184
ALU	Arithmetic Logic Unit, 72
AO	Access Oriented, 107
ASAP	As Soon As Possible, 131
ASIC	Application Specific IC, 70
ASIP	Application Specific Instruction Processor, 117
AUR	Area Utilization Ratio, 179
B_Ops	Boundary Operations, 227
BB	Basic Block, 181
CDFG	Control Data Flow Graph, 130
CFG	Control-Flow Graph, 181
CompAssisted	Compiler Assisted DSE, 118
CompAssistedCLK	Compiler Assisted with CLK Selection DSE, 118
CompDirected	Compiler Directed DSE, 118
CompInLoop	Compiler-In-Loop, 116
CR	Complex Reconfigurable, 193
CS	Carry Save, 77
CS2Bin	Carry Save to Binary conversion, 167
DDT	Dynamic Data Type, 99
DFG	Data-Flow Graph, 178
DM	Multi-Threaded Dynamic Memory, 89
DMM	Dynamic Memory Management, 75
DRR	Deficit Round Robin, 109
DSE	Design Space Exploration, 67
DSP	Digital Signal Processing, 71
DSP	Digital Signal Processor, 70
DT	Decision Tree, 90
DTSE	Data Transfer and Storage Exploration, 74

EDA Electronic Design Automation, 65
ExCompInLoop Exhaustive Compiler-In-Loop DSE, 138

FCC Flexible Computational Component, 178
FFT Fast Fourier Transform, 199
FIFO First-In-First-Out, 74
FIR Finite Impulse Response, 136
FO Footptint Oriented, 107
FPGA Field Programmable Gate Array, 71
FPS Flexible Pipeline Stage, 165
FSM Finite State Machine, 176
FU Functional Unit, 77

GCRC Coarse Grained Reconfigurable Cell, 178
GD Generational Distance, 141

HLS High Level Synthesis, 66
HW ACC Hardware Accelerator, 66

IC Integrated Circuit, 65
ILP Instruction Level Parallelism, 89
IR Intermediate Representation, 181

L-CDFG Lowered CDFG, 181
L-DFG Lowered DFG, 181
LD/ST Load/Store, 122
LIFO Last-In-First-Out, 91
LMS Least Mean Square, 136
LSB Least Significant Bit, 169
LUF Loop Unrolling Factor, 126

MAC Multiply-Accumulate, 153
MC Multiplier Cell, 156
MD Modified Booth, 221
MOPS Mega Operations Per Second, 153
MPSoC Multi-Processor System-on-Chip, 76
MTh-DMM Multi-Threaded Dynamic Memory Management, 76

NISC No Instruction Set Computer, 154
NR Non Reconfigurable, 193

OS Operating System, 91

QoS **Quality of Service, 65**

RAU **Reconfigurable Arithmetic Unit, 165**
RC **Reconfigurable Computing, 70**
RTL **Register Transfer Level, 152**

SC **Subtraction Cell, 158**
SD **Signed Digit, 221**
SL **Solution Coverage, 141**
SoC **System-on-Chip, 70**
SR **Single Reconfigurable, 193**
STL **Standard Template Library, 94**

UC **Unified Cell, 158**

VLIW **Very Large Instruction Word, 65**

Bibliography

[1] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of asics," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, 1989.

[2] M. Balakrishnan and P. Marwedel, "Integrated Scheduling and Binding: A Synthesis Approach for Design Space Exploration," in *DAC '89: Proceedings of the 26th ACM/IEEE Design Automation Conference*, 1989, pp. 68–74.

[3] R. Dutta, J. Roy, and R. Vemuri, "Distributed Design-Space Exploration for High-Level Synthesis Systems," in *DAC '92: Proceedings of the 29th ACM/IEEE Design Automation Conference*, 1992, pp. 644–650.

[4] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "A Solution Methodology for Exact Design Space Exploration in a Three-Dimensional Design Space," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 5, no. 1, pp. 69–81, 1997.

[5] G. Wang, W. Gong, B. DeRenzi, and R. Kastner, "Exploring Time/Resource Trade-offs by Solving Dual Scheduling Problems with the Ant Colony Optimization," *ACM Trans. Design Autom. Electr. Syst.*, vol. 12, no. 4, 2007.

[6] P. Marwedel, B. Landwehr, R. Domer, "Built-in Chaining: Introducing Complex Components into Architectural Synthesis," in *Proc. of the ASP-DAC*, 1997, pp. 599–605.

[7] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabaey, "Perfomance Optimization Using Template Mapping for Datapath-Intensive High-Level Synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 15, pp. 877–888, 1996.

[8] S. A. Blythe and R. A. Walker, "Efficiently Searching the Optimal Design Space," in *GLS '99: Proceedings of the Ninth Great Lakes Symposium on VLSI*, 1999, p. 192.

[9] S. Kurra, N. K. Singh, and P. R. Panda, "The impact of Loop Unrolling on Controller Delay in High Level Synthesis," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, 2007, pp. 391–396.

[10] J. Gerlach and W. Rosenstiel, "A Methodology and Tool for Automated Transformational High-Level Design Space Exploration," in *ICCD*, 2000, pp. 545–548.

[11] J. P. B. S. Pedro Diniz, Mary Hall and H. Ziegler, "Bridging the gap between compilation and synthesis in the defacto system," in *In Proc. 14th Workshop Languages and Compilers for Parallel Computing, LNCS*. Springer-Verlag, 2001, pp. 52–70.

[12] O. Dragomir, E. Panainte, K. Bertels, and S. Wong, "Optimal Unroll Factor for Reconfigurable Architectures," in *Proc. of ARC*, 2008, pp. 4–14.

[13] P. Bonzini and L. Pozzi, "Code transformation strategies for extensible embedded processors," in *CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*. New York, NY, USA: ACM, 2006, pp. 242–252.

[14] A. C. Murray, R. V. Bennett, B. Franke, and N. P. Topham, "Code Transformation and Instruction Set Extension," *ACM Trans. Embedded Comput. Syst.*, vol. 8, no. 4, 2009.

[15] F. Barat, M. Jayapala, T. Vander, R. Lauwereins, G. Deconinck, H. Corporaal, "Low Power Coarse-Grained Reconfigurable Instruction Set Processor," in *Proc. of FPL'03*. Springer, 2003, pp. 230–239.

[16] M. Galanis, G. Theodoridis, S. Tragoudas, C. Goutis, "A High Performance Data-Path for Synthesizing DSP Kernels," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, pp. 1154–1163, 2006.

[17] M. Reshadi, B. Gorjiara, D. Gajski, "Utilizing Horizontal and Vertical Parallelism with a No-Instruction-Set Compiler for Custom Datapaths," in *Proc. of ICCD'05*, 2005, pp. 69–76.

[18] Taewhan Kim and Junhyung Um, "A practical approach to the synthesis of arithmetic circuits using carry-save-adders," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 19, no. 5, pp. 615–624, 2000.

[19] A. K. Verma, P. Brisk, and P. Ienne, "Data-flow transformations to maximize the use of carry-save representation in arithmetic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1761–1774, 2008.

[20] M. R. Krishnan, "Heap: Pleasures and pains," *Microsoft Developer Newsletter*, 1999.

[21] E. Berger *et al.*, "Hoard: A scalable memory allocator for multithreaded applications," *SIGPLAN Not*, vol. 35, no. 11, Nov. 2000.

[22] K. Tatas, G. Koutroumpezis, D. Soudris, and A. Thanailakis, "Architecture design of a coarse-grain reconfigurable multiply-accumulate unit for data-intensive applications," *Integration*, vol. 40, no. 2, pp. 74–93, 2007.

[23] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[24] K. Fan, M. Kudlur, G. S. Dasika, and S. A. Mahlke, "Bridging the computation gap between programmable processors and hardwired accelerators," in *15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, Raleigh, North Carolina, USA*, 2009, pp. 313–322.

[25] ARM Corporation, www.arm.com.

[26] Texas Instruments Corporation, www.ti.com.

[27] Intel Corporation, www.intel.com.

[28] V. Pareto, *Manuale di Economia Politica.* Picola Biblioteca Scientifica, Milan, 1906, Translated into English by Ann Schweir (1971), Manual of Political Economy, MacMillan, London, 2008.

[29] M. Gries, "Methods for Evaluating and Covering the Design Space during Early Design Development," *Integr. VLSI J.*, vol. 38, no. 2, pp. 131–183, 2004.

[30] D. Atienza *et al.*, "Systematic Dynamic Memory Management Design Methodology for Reduced Memory Footprint," *ACM TODAES*, vol. 11, pp. 465–489, 2006.

[31] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Comp. Surveys*, vol. 34, pp. 171–210, 2002.

[32] G. Theodoridis, D. Soudris, S. Vassiliadis, "A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools," *Book Chapter in "Fine- and Coarse-Grain Reconfigurable Computing", S. Vassiliadis and D. Soudris Editors, Springer*, pp. pages 3–149, 2007.

[33] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh, "A quick safari through the reconfiguration jungle," in *Proceedings of the 38th annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 172–177. [Online]. Available: http://doi.acm.org/10.1145/378239.378404

[34] P. R. Wilson *et al.*, "Dynamic storage allocation, a survey and critical review," in *Proc. of IWMM*, 1995.

[35] S. Mamagkakis *et al.*, "Energy-efficient dynamic memory allocators at the middleware level of embedded systems," in *Proc. of EMSOFT*, 2006, pp. 215–222.

[36] Kiem-Phong Vo, "Vmalloc: A general and efficient memory allocator," *Softw. Pract. Exper.*, no. 26, pp. 1–18, 1996.

[37] V. Vee and W. Hsu, "A scalable and efficient storage allocator on shared memory multiprocessors," in *Proc. of I-SPAN*, 1999, pp. 230–235.

[38] P. Larson and M. Krishnan, "Memory allocation for long-running server applications," in *Proc. of ISMM*, 1998.

[39] A. Iyengar, "Parallel dynamic storage allocation algorithms," in *Proc. of PDP*, 1993.

[40] E. D. Berger *et al.*, "Composing high performance memory allocators," in *Proc. of PLDI*, 2001.

[41] K. Keutzer *et al.*, "System level design: Orthogonalization of concerns and platform-based design," *IEEE TCAD*, vol. 19, pp. 1523–1543, 2000.

[42] G. Bracha and W. Hook, "Mixin-based inheritance," in *Proc. of OOPSLA*, 1990, pp. 303G•–311.

[43] S. Gupta, N. Savoiu, N. Dutt, R. Gupta, A. Nicolau, "Using Global Code Motions to Improve the Quality of Results for High-Level Synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 23, 2003.

[44] G. De Micheli, *Synthesis and Optimization of Digital Circuits.* McGraw-Hill Higher Education, 1994.

[45] R. Brayton, A. Sangiovanni-Vincentelli, C. McMullen, and G. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[46] G. Constantinides, P. Cheung, W. Luk, *Synthesis And Optimization Of DSP Algorithms.* Kluwer Academic Publishers, 2004.

[47] Verilog HDL, http://www.verilog.org.

[48] R. Kastner, S. Ogrenci-Memik, E. Bozorgzadeh, M. Sarrafzadeh, "Instruc-

tion Generation for Hybrid Reconfigurable Systems," *ACM Trans. on Design Automation of Electronic Systems*, vol. 7, pp. 605–627, 2002.

[49] P. Heysters, G. Smit, E. Molenkamp, "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems," *The Journal of Supercomputing*, vol. 26, pp. 283–308, 2003.

[50] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III: Architecture-Driven High-Level Synthesis for High Throughput DSP Applications," in *Proc. ACM/IEEE DAC*, 1991, pp. 597–602.

[51] Z. Yu, K.Y. Khoo, A. Willson, "The Use of Carry-Save Representation in Joint Module Selection and Retiming," in *Proc. of IEEE/ACM DAC*, 2000, pp. 768–773.

[52] W.C. Yeh, C.W. Jen, "A High Performance Carry-Save to Signed-Digit Recoder for Fused Addition-Multiplication," in *Proc. of IEEE ICASSP*, 2000, pp. 3259–3262.

[53] B. Parhami, "Computer arithmetic: algorithms and hardware designs." Oxford, UK: Oxford University Press, 2000.

[54] T. Kim, W. Jao, S. Tjiang, "Circuit Optimization Using Carry-Save-Adder Cells," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 974–984, 1998.

[55] A. L. Sangiovanni-Vincentelli, "The tides of eda," *IEEE Design & Test of Computers*, vol. 20, no. 6, pp. 59–75, 2003.

[56] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006*, 2006, pp. 1–15.

[57] P. Coussy and A. Morawiec, *High-Level Synthesis: From Algorithm to Digital Circuit.* Springer, Berlin, Germany, 2008.

[58] F. Catthoor, E. d. Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design.* Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[59] A. Olugbon, S. Khawam, T. Arslan, I. Nousias, and I. Lindsay, "An AMBA AHB-based reconfigurable SoC architecture using multiplicity of dedicated flyby dma blocks," in *proccedings of ASP-DAC.*

[60] S. Wallner, "A Configurable System-on-Chip Architecture for Embedded and

Real-Time Applications: Concepts, Design and Realization," *in Elsevier Journal of Systems Architecture*, vol. 51, no. 6-7, p. 350, 2005.

[61] N. Dutt, "Memory-aware noc exploration and design," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 1128–1129. [Online]. Available: http://doi.acm.org/10.1145/1403375.1403650

[62] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou, "An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures," in *Proc. of 8th Euromicro Workshop on Parallel and Distributed Processing, (PDP2000), IEEE.* Press, 2000, pp. 406–413.

[63] R. Baert, E. Brockmeyer, S. Wuytack, and T. Ashby, "Exploring parallelizations of applications for mpsoc platforms using mpa," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 1148 – 1153.

[64] Y. Bouchebaba, B. Girodias, G. Nicolescu, E. M. Aboulhamid, B. Lavigueur, and P. G. Paulin, "Mpsoc memory optimization using program transformation," *ACM Trans. Design Autom. Electr. Syst.*, vol. 12, no. 4, 2007.

[65] G. De Micheli, R. Ernst, and W. Wolf, Eds., *Readings in hardware/software co-design.* Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[66] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, vol. 33, pp. 1023–1029, 1984.

[67] S. Gupta, R. Gupta, N. Dutt, A. Nicolau, *SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits.* Springer.

[68] J. Mignolet and R. Wuyts, "Embedded multiprocessor systems-on-chip programming," *IEEE Software*, vol. 26, no. 3, pp. 34–41, 2009.

[69] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip," in *ICCAD*, 2001, pp. 25–30.

[70] G. Palermo, C. Silvano, and V. Zaccaria, "An efficient design space exploration methodology for on-chip multiprocessors subject to application-specific constraints," in *SASP*, 2008, pp. 75–82.

[71] H. Wang, L.-S. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *DATE*, 2005, pp. 1238–1243.

[72] P. Grun, A. Nicolau, and N. Dutt, *Memory Architecture Exploration for Programmable Embedded Systems*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[73] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[74] K. Hirata and J. Goodacre, "ARM MPCore; The streamlined and scalable ARM11 processor core," in *Proc. of ASP-DAC*, 2007, pp. 747–748.

[75] S. Agarwala *et al.*, "A 65nm C64x+ Multi-Core DSP Platform for Communications Infrastructure," in *Proc. of ISSCC*, 2007, pp. 262–601.

[76] T. Johnson, "A concurrent fast-fits memory manager," *Technical Report TR91-009, University of Florida, Department of CIS*, vol. , no. , 1991.

[77] C. Schlatter Ellis and T. J. Olson, "Algorithms for parallel memory allocation," *International Journal of Parallel Programming*, vol. 17, no. 4, pp. 303–345, 1988.

[78] B. Bigler *et al.*, "Parallel dynamic storage allocation," in *Proc. of ICPP*, 1985, pp. 272–275.

[79] M. S. Johnstone, "Non-Compacting Memory Allocation and Real-Time Garbage Collection," *PhD thesis, University of Texas at Austin*, vol. , no. , Dec. 1997.

[80] F. Garcia and J. Fernandez, "POSIX thread libraries," *Linux Journal*, p. 36.

[81] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," in *Proc. of FOCS*, Nov. 1994, pp. 356G●–368.

[82] The Standard Template Library C++: Allocators, "http://www.sgi.com/tech/stl/Allocators.html."

[83] Solaris 9 Reference Manual man pages for mtmalloc, "http://docs.sun.com/."

[84] W. Gloger, "Dynamic memory allocator implementations in Linux system libraries," *http://www.dent.med.uni-muenchen.de/ wmglo/malloc-slides.html*, 2002.

[85] P. E. McKenney and J. Slingwine, "Efficient kernel memory allocation on shared memory multiprocessor," in *Proc. of USENIX*, 1993, pp. 295–305.

[86] A. Gidenstam *et al.*, "Allocating Memory in Lock-Free Manner," in *Proc. of ESA*, 2005, pp. 329–342.

[87] M. M. Michael, "Scalable lock-free dynamic memory allocation," *SIGPLAN Not*, vol. 39, no. 6, pp. 35–46, 2004.

[88] A. Silberschatz and P. B. Galvin, "Operating System Concepts," *Addison-Wesley*, 1994.

[89] R. Hudson *et al.*, "Mcrt-malloc: a scalable transactional memory allocator," in *Proc. of ISMM*, 2006, pp. 74–83.

[90] M. Herlihy *et al.*, "Nonblocking memory management support for dynamic-sized data structures," *ACM Trans. Comput. Syst.*, vol. 23, no. 2, pp. 146–196, 2005.

[91] M. Greenwald and D. Cheriton, "The synergy between non-blocking synchronization and operating system structure," in *Proc. of OSDI*, 1996, pp. 123–136.

[92] E. G. Daylight *et al.*, "Incorporating energy efficient data structures into modular software implementations for internet-based embedded systems," in *Proc. of WSP*, July 2002.

[93] M. Leeman *et al.*, "Efficient data type identification on image processing applications with dynamic memory behaviour," in *Proc. of PACT*, vol. , no. , 2002.

[94] A. Bartzas *et al.*, "Software metadata: Systematic characterization of the memory behaviour of dynamic applications," *Journal of Systems and Software*, vol. In Press, Corrected Proof, pp. –, 2010.

[95] Y.-L. Lin, "Recent developments in High-Level Synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 2, no. 1, pp. 2–21, 1997.

[96] S. A. Blythe and R. A. Walker, "Efficient optimal design space characterization methodologies," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, no. 3, pp. 322–336, 2000.

[97] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Coordinated Parallelizing Compiler Optimizations and High-Level Synthesis," *ACM Trans. Des. Autom. Electron. Syst*, vol. 9, p. 2004, 2002.

[98] A. Shrivastava, I. Issenin, N. Dutt, S. Park, and Y. Paek, "Compiler-in-the-loop design space exploration framework for energy reduction in horizontally partitioned cache architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 3, pp. 461–465, 2009.

[99] B. C. Schäfer and K. Wakabayashi, "Design space exploration acceleration

through operation clustering," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 1, pp. 153–157, 2010.

[100] C. Lee, M. Potkonjak, W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems," in *Proc. of the MICRO-30*, 1997, pp. 330–335.

[101] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 1998, pp. 545–550.

[102] J. A. Davis and J. D. Meindl, *Interconnect Technology and Design for Gigascale Integration*. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[103] T. Kim and X. Liu, "Compatibility path based binding algorithm for interconnect reduction in high level synthesis," in *ICCAD '07: Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 435–441.

[104] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *ASP-DAC '04: Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2004, pp. 68–73.

[105] S. O. Memik, N. Bellas, and S. Mondal, "Presynthesis area estimation of reconfigurable streaming accelerators," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 11, pp. 2027–2038, 2008.

[106] www.synopsys.com/products/logic/design_compiler.

[107] Artisan Components, TSMC 0.13 Library Databook.

[108] D. Atienza, J. M. Mendias, S. Mamagkakis, D. Soudris, and F. Catthoor, "Systematic dynamic memory management design methodology for reduced memory footprint," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 2, pp. 465–489, 2006.

[109] K. Kennedy and J. R. Allen, *Optimizing compilers for modern architectures: a dependence-based approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[110] SPARK HLS Tool, http://mesl.ucsd.edu/spark/methodology.

[111] SNU-RT Benchmarks, "http://archi.snu.ac.kr/realtime/benchmark/."

[112] C to Verilog, "Circuit Design Automation, http://c-to-verilog.com/."

[113] Berkeley MPEG-1 Video Tools, "http://www.bmrc.berkeley.edu/mpeg."

[114] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indices for multi-objective optimisation," in *Proc. of 2003 Congress on Evolutionary Computation*. IEEE Press, 2003, pp. 878–885.

[115] A. Peymandoust, L. Pozzi, P. Ienne, G. De Micheli, "Automatic Instruction Set Extension and Utilization for Embedded Processors," in *Proc. of IEEE ASAP Conference*, 2003, pp. 103–114.

[116] F. Sun, S. Ravi, A. Raghunathan, N. Jha, "Synthesis of Custom Processors Based on Extensible Platforms," in *Proc. IEEE/ACM ICCAD*, 2002, pp. 641–648.

[117] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. Filho, "Morphosys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Trans. on Computers*, vol. 49, p. 465, 2000.

[118] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *Proc. of FPL'03*, 2003, pp. 61–70.

[119] C. Ebeling, D. Green, P. Franklin, "RaPiD: Reconfigurable Pipelined Datapath," in *Proc. of FPL'06*. Springer, 1996, pp. 126–135.

[120] S. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, R. Taylor, R. Reed, "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33, pp. 70–77, 2000.

[121] Z. Huang, S. Malik, N.Moreano, G. Araujo, "The design of dynamically reconfigurable datapath coprocessors," *ACM Trans. Embedded Comput. Syst.*, vol. 3, pp. 361–384, 2004.

[122] N. Moreano, E. Borin, C. de Souza, and G. Araujo, "Efficient datapath merging for partially reconfigurable architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, pp. 969–980, 2005.

[123] K. Hwang, "Computer Arithmetic," in *John Wiley and Sons*, 1979.

[124] V. Van-Dongen and P. Quinton, "Uniformization of linear recurrence equations: A step towards the automatic synthesis of systolic arrays," in *Proc. of the IEEE International Conference on Systolic Arrays*, 1998, pp. 473–482.

[125] M. Manjunathaiah, G. Megson, S. Rajopadhy, and T. Risset, "Uniformization

of affine dependence programs for parallel embedded system design," in *Proc. IEEE International Conference on Parallel Processing (ICPP '01)*, 2001, pp. 205–213.

[126] L. Chiou, S. Bhunia, and K. Roy, "Synthesis of Application-specific Highly Efficient Multi-mode Cores for Embedded Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 1, pp. 168–188, 2005.

[127] W. Zhao and C. Papachristou, "Synthesis of reusable DSP cores based on multiple behaviours."

[128] T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's," *IEEE Trans. Computers*, vol. 33, no. 10, pp. 879–894, 1984.

[129] S. Perri, P. Corsonello, G. Cocorullo, "A High-Speed Energy-Efficient 64-bit Reconfigurable Binary Adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 939–943, 2003.

[130] F. Elguibaly, "Overflow Handling in Inner-Product Processors," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 0, 2000.

[131] Texas Instruments Inc., TMS320C6000 Family: High Performance DSP Platform, available at: http://www.ti.com.

[132] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, F. Homewood, "Lx: A technology platform for customizable vliw embedded processing," in *Proc. of the 27th Annual international Symposium on Computer Architecture, ISCA '00*, 2000, pp. 203–213.

[133] A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1988.

[134] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[135] www.synopsys.com/products.

[136] ModelSim, http://www.model.com.

[137] H. Singh, M. Lee, F. K. G. Lu, N. Bagherzadeh, and E. Filho, "Design and implementation of the morphosys reconfigurable computing processor," *in Journal of VLSI Signal Processing Systems, Kluwer*, vol. 24, pp. 147–164, 2000.

[138] Chiricescu, Schuette, Glinton, and Schmit, "Morphable multipliers," in *Proc.*

of the International Conference on Field Programmable Logic and Applications, 2002, pp. 647–656.

[139] R. Lin, "Reconfigurable parallel inner product processor architectures," *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 261–272, 2001.

[140] P. Corsonello, S. Perri, M. Iachino, G. Cocorullo, "Variable precision multipliers for fpga-based reconfigurable computing systems," in *Proc. of the FPL 2003*, 2003, pp. 661–669.

[141] O. Al-Khaleel, C. Papachristou, F. Wolff, K. Pekmestzi, "A large scale adaptable multiplier for cryptographic applications," in *Proc. of the IEEE/NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006*, 2006, pp. 477–484.

[142] w. Xilinx Corporation, Xilinx Virtex Series.

[143] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, and G. Cambon, "Metrics for reconfigurable architectures characterization: Remanence and scalability," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 176a, 2003.

[144] CDFG toolset, http://poppy.snu.ac.kr/CDFG/cdfg.html.

[145] 16-point FFT source code, http://www.mit.edu/ emin/source_code/fft.

[146] The ExPRESS group, http://express.ece.ucsb.edu.

[147] G. Aggarwal, D. Gajski, "Exploring DCT Implementations," *UC Irvine, Tech. Rep. ICS-TR-98-10*, vol. available at: http://www.cecs.uci.edu/ cad/publications/tech-reports/1998/.

[148] Online NISC Toolset, http://cecs02.cecs.uci.edu/nisc/demo.v10.

[149] B. Gorjiara, D. Gajski, "Automatic Architecture Refinement Techniques for Customizing Processing Elements," in *Proc. of the IEEE/ACM 45th DAC*, 2008, pp. 379–384.

[150] S. A. Mahlke, R. A. Ravindran, M. S. Schlansker, R. Schreiber, and T. Sherwood, "Bitwidth cognizant architecture synthesis of custom hardware accelerators," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1355–1371, 2001.

[151] S. Ahuja, S. T. Gurumani, C. Spackman, and S. K. Shukla, "Hardware coprocessor synthesis from an ansi c specification," *IEEE Des. Test*, vol. 26, no. 4, pp. 58–67, 2009.

[152] K. Wakabayashi, "C-based synthesis experiences with a behavior synthesizer, "cyber"," in *DATE '99: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 1999, p. 83.

[153] Catapult C Synthesis, Mentor Graphics [Online]. Available: http://www.mentor.com.

[154] A. Hosangadi, F. Fallah, R. Kastner, "Optimizing High Speed Arithmetic Circuits Using Three-Term Extraction," in *Proc. of IEEE/ACM DATE*, 2006, pp. 1294–1299.

[155] P. F. Stelling, C. U. Martel, V. G. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Trans. Computers*, vol. 47, no. 3, pp. 273–285, 1998.

[156] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, pp. 94–125, March 2004.

[157] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams, "Using machine learning to focus iterative optimization," in *Proceedings of the International Symposium on Code Generation and Optimization*, ser. CGO '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 295–305.

[158] A. Cevrero, P. Athanasopoulos, H. Parandeh-Afshar, A. K. Verma, S. H. A. Niaki, C. Nicopoulos, F. K. Gürkaynak, P. Brisk, Y. Leblebici, and P. Ienne, "Field programmable compressor trees: Acceleration of multi-input addition on fpgas," *TRETS*, vol. 2, no. 2, 2009.

[159] A. Kahoul, G. A. Constantinides, A. M. Smith, and P. Y. Cheung, "Heterogeneous architecture exploration: Analysis vs. parameter sweep," in *Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, ser. ARC '09, 2009, pp. 133–144.