Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων & Ψηφιακων
Συστηματων

Μεθοδολογία Παραμετροποίησης Εφαρμογών βασισμένων σε Ταυτόχρονες Δομές Δεδομένων για Ενσωματωμένα Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΛΑΖΑΡΟΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Αθήνα, Ιούλιος 2016

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων

Τομεας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων & Ψηφιακων
Συστηματων

Μεθοδολογία Παραμετροποίησης Εφαρμογών βασισμένων σε Ταυτόχρονες Δομές Δεδομένων για Ενσωματωμένα Συστήματα

Διδακτορική Διατριβή
του

Λάζαρου Παπαδόπουλου
Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

Επταμελής Εξεταστική Επιτροπή

Δημήτριος Σούντρης Αν. Καθηγητής Ε.Μ.Π.	Κιαμάλ Πεκμεστζή Καθηγητής Ε.Μ.Π.	Γεώργιος Οικονομάκος Επ. Καθηγητής Ε.Μ.Π.
Γεώργιος Γκούμας Λέκτορας Ε.Μ.Π.	Κωνσταντίνος Κοντογιάννης Αν. Καθηγητής Ε.Μ.Π.	
Αλέξανδρος Χατζηγεωργίου Αν. Καθηγητής Π. Μακεδονίας	Philippas Tsigas Καθ. Chalmers University	

Περίληψη

Στις σύγχρονες πολυπύρηνες αρχιτεκτονικές ενσωματωμένων συστημάτων, εκτελούνται συχνά εφαρμογές που βασίζονται σε ταυτόχρονες δομές δεδομένων, όπως βάσεις δεδομένων, αλγόριθμοι work-stealing κ.α.. Η επιλογή κατάλληλης δομής δεδομένων είναι ένα πολύπλοκο πρόβλημα, που αφενός η εύρεση λύσης από τον προγραμματιστή είναι χρονοβόρα διαδικασία και αφετέρου η επιλογή μη κατάλληλης υλοποίησης μπορεί να έχει αρνητική επίπτωση σε διάφορους σχεδιαστικούς περιορισμούς, όπως στην απόδοση και στην κατανάλωση ενέργειας. Για την αντιμετώπιση αυτού του προβλήματος, στην παρούσα διατριβή παρουσιάζουμε μία συστηματική μεθοδολογία για την επιλογή κατάλληλων ταυτόχρονων δομών δεδομένων στις εφαρμογές που εκτελούνται σε ενσωματωμένα συστήματα, συμβάλλοντας έτσι στη βελτιστοποίηση των εφαρμογών που τις χρησιμοποιούν. Η μεθοδολογία είναι ημιαυτόματη και βασίζεται στην εξερεύνηση του χώρου λύσεων των ταυτόχρονων δομών δεδομένων. Υποστηρίζεται από μία ροή εργαλείων που αυτοματοποιεί πολλά βήματα της μεθοδολογίας και παρέχει δυνατότητες για αποτελεσματική εξερεύνηση και επεκτασιμότητα. Δίνει την δυνατότητα στους προγραμματιστές να αξιολογήσουν με συστηματικό και αποτελεσματικό τρόπο υλοποιήσεις από τον χώρο λύσεων των ταυτόχρονων δομών δεδομένων και να επιλέξουν αυτήν που παρέχει τα καλύτερα αποτελέσματα σύμφωνα με τους σχεδιαστικούς περιορισμούς. Η μεθοδολογία εφαρμόστηκε σε μια σειρά από benchmarks υλοποιημένα σε δύο ενσωματωμένα συστήματα με διαφορετικές αρχιτεκτονικές. Τα αποτελέσματα έδειξαν ότι με κατάλληλη επιλογή υλοποίησης ταυτόχρονων δομών δεδομένων επιτυγχάνονται σε πολλές περιπτώσεις ανταλλάγματα ανάμεσα σε μετρικές όπως στην απόδοση, στην κατανάλωση ενέργειας του συστήματος και στο fairness της ταυτόχρονης δομής. Επιπλέον, αντίστοιχη μεθοδολογία παραμετροποίησης εφαρμόστηκε σε υλοποιήσεις operators επεξεργασίας ροών δεδομένων εκτελεσμένες σε ενσωματωμένες αρχιτεκτονικές. Μέσω των υλοποιήσεων που προέκυψαν, μπορούν να επιτευχθούν ανταλλάγματα μεταξύ των throughput, latency, μεγέθους απαιτούμενης μνήμης και κατανάλωσης ενέργειας.

Λέξεις κλειδιά

ταυτόχρονες δομές δεδομένων, απόδοση, κατανάλωση ενέργειας, ενσωματωμένα συστήματα, πολυπύρηνες αρχιτεκτονικές, ανταλλαγή μηνυμάτων.

Abstract

Modern embedded system architectures integrate multiple cores and they often execute applications that rely on concurrent data structures. The selection of an efficient concurrent data structure implementation is a difficult and time consuming task. Selecting a non-efficient implementation usually has negative impact on various embedded system metrics such as the performance and the energy consumption. In the present thesis, we propose a systematic methodology for the selection of effective concurrent data structures in modern embedded system applications. The methodology is semi-automatic and it is based on the exploration of the design space of concurrent data structures. It is supported by a tool flow which automates a number of steps of the methodology. The methodology allows the exploration of a number of concurrent data structures implementations by developers and the selection of the most efficient one in each context. It is applied in a number of benchmarks implemented in two modern embedded devices with different architectural specifications. The results show that by changing concurrent data structure implementations, trade-offs can be identified in many cases between metrics such as performance, energy consumption and fairness of the data structure. A similar customization methodology is proposed for the implementation of the streaming aggregation operator in embedded devices. Trade-offs between throughput, latency, required memory size and energy consumption can be identified, by selecting different customized implementations.

Key words

concurrent data structures, performance, energy consumption, embedded systems, multicore architectures, message-passing.

Περιεχόμενα

Πίνακες

Σχήματα

Κεφάλαιο 1

Εισαγωγή

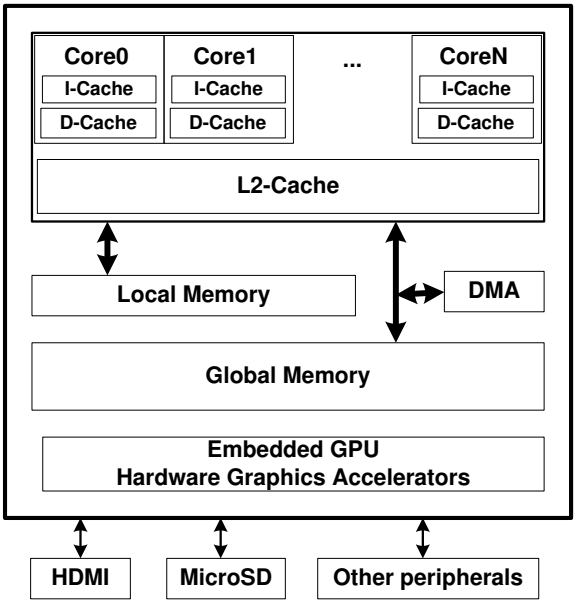
Η διατριβή ''Μεθοδολογία Παραμετροποίησης Εφαρμογών βασισμένων σε Ταυτόχρονες Δομές Δεδομένων για Ενσωματωμένα Συστήματα'' βασίζεται σε δύο κεντρικούς άξονες. Ο πρώτος άξονας αφορά τα ενσωματωμένα συστήματα (όπως αυτά περιγράφονται στην ενότητα 1.1), τα οποία ορίζουν την πλατφόρμα υλικού (hardware platform). Ο δεύτερος άξονας, αφορά τις ταυτόχρονες δομές δεδομένων (όπως περιγράφονται στην ενότητα 1.2), οι οποίες αποτελούν το βασικό πεδίο έρευνας της παρούσας διατριβής.

1.1 Ενσωματωμένα συστήματα

Ένα ενσωματωμένο σύστημα (embedded system) είναι ένα σύστημα με εξειδικευμένο σκοπό, το οποίο σε αντίθεση με τους προσωπικούς υπολογιστές (Personal Computers - PC), εκτελεί πολύ συγκεκριμένες και προκαθορισμένες λειτουργίες [1]. Λόγω της εξειδικευμένης φύσης των ενσωματωμένων συστημάτων, είναι λογικό οι σχεδιαστές τους να προσπαθούν να βελτιώσουν τον σχεδιασμό τους, ώστε να ανταποκρίνονται όσο το δυνατόν καλύτερα στις προκαθορισμένες λειτουργίες που επιτελούν.

Υπάρχουν διάφορες αρχιτεκτονικές ενσωματωμένων συστημάτων, με χαρακτηριστικά ανάλογα με το πεδίο εφαρμογών που καλούνται να εκτελέσουν. Η παρούσα διατριβή στοχεύει σε αρχιτεκτονικές παρόμοιες με αυτές που απεικονίζονται στο Σχήμα 1.1. Η συγκεκριμένη αρχιτεκτονική περιλαμβάνει πολυπύρηνο επεξεργαστή, DMA controller και πολυεπίπεδη ιεραρχία μνήμης. Διάφορες σύγχρονες πλατφόρμες, όπως η Freescale I.MX.6 [2], Exynos 4 [3], Rockchip [4], HummingBoard [5] ακολουθούν το συγκεκριμένο γενικό πρότυπο.

Οι συγκεκριμένες πολυπύρηνες αρχιτεκτονικές περιέχουν shared memories (κοινόχρηστες μνήμες), στις οποίες έχουν πρόσβαση περισσότεροι από ένας πυρήνες. Επομένως, είναι απαραίτητος κάποιος μηχανισμός που θα συγχρονίζει τις ταυτόχρονες προσβάσεις στη μνήμη σε κοινά δεδομένα. Ένας τέτοιος μηχανισμός επιτρέπει την δημιουργία π.χ. κλειδωμάτων (locks) ή ατομικών λειτουργιών (atomic operations) σε υψηλότερο επίπεδο, τα οποία θα μπορούν να χρησιμοποιηθούν απευθείας από την εφαρμογή. Για παράδειγμα, ο επεξεργαστής ARM παρέχει τις μικροεντολές *load-link* και *store-conditional (LL/SC)*, στις οποίες βασίζεται η υλοποίηση κλειδωμάτων στο



Σχήμα 1.1: Παράδειγμα αρχιτεκτονικής ενσωματωμένου συστήματος.

επίπεδο της εφαρμογής (π.χ. *posix mutexes*).

Η ιεραρχία μνήμης στο συγκεκριμένο πρότυπο μπορεί να διαχωριστεί σε local (τοπική) και global (καθολική). Η διαφοροποίηση μεταξύ local και global σχετίζεται με την ταχύτητα πρόσβασης, η οποία μπορεί να οφείλεται στον τύπο της μνήμης (π.χ. local - SRAM, global - DDR) ή στην "απόσταση" από τον αντίστοιχο επεξεργαστή. Επιπλέον, μπορεί η local μνήμη να είναι on-chip, ενώ η global off-chip. Υπάρχει η περίπτωση η local μνήμη να είναι χωρισμένη σε τμήματα, όπου το καθένα από αυτά να είναι "τοπικό" σε συγκεκριμένο πυρήνα. Στην περίπτωση αυτή, κάθε πυρήνας έχει ταχύτερη πρόσβαση στο συγκεκριμένο τμήμα σε αντίθεση με τα υπόλοιπα τμήματα της local μνήμης. Για τη συγκεκριμένη ιεραρχία μνήμης είναι προφανώς απαραίτητη η ύπαρξη DMA, ώστε δεδομένα από την local στην global μνήμη να μεταφέρονται με υψηλή απόδοση.

Τέλος η ιεραρχία μπορεί να περιλαμβάνει πολλαπλά επίπεδα cache (κρυφής) μνήμης. Σε πολυπύρηνες αρχιτεκτονικές όπου υπάρχουν κοινά δεδομένα, η συνοχή (coherence) της cache είναι σημαντικό ζήτημα. Αφορά την αξιοπιστία των δεδομένων που υπάρχουν στις caches σε ένα πολυπύρηνο σύστημα. Το coherence συνήθως εξασφαλίζεται με μηχανισμούς που παρέχονται από την ίδια την πλατφόρμα, χωρίς να χρειάζονται ενέργειες από την πλευρά του προγραμματιστή. Υπάρχει όμως η περίπτωση να είναι αποκλειστικά υπεύθυνος για αυτήν ο προγραμματιστής [6]. Επιπλέον, η ύπαρξη cache μνήμης μπορεί να αποτελέσει παράγοντα μείωσης της απόδοσης σε πολυπύρηνα συστήματα όπου οι πυρήνες έχουν πρόσβαση στα ίδια δεδομένα (π.χ. στο ίδιο κλείδωμα ή στις ίδιες μεταβλητές). Η αλλαγή της τιμής μιας κοινής μεταβλητής από έναν πυρήνα, καθιστά αυτόματα άκυρη την τιμή της μεταβλητής αυτής

στις caches των υπόλοιπων πυρήνων. Αυτό με τη σειρά του θα προκαλέσει αυξημένη κίνηση στον δίαυλο δεδομένων (bus) του συστήματος, με συνέπεια τη μείωση της απόδοσης. Το συγκεκριμένο γεγονός, είναι ο λόγος για τον οποίο οι υλοποιήσεις ταυτόχρονων δομών δεδομένων βασισμένες σε κλειδώματα έχουν περιορισμένο scalability (επεκτασιμότητα) καθώς αυξάνει ο αριθμός των πυρήνων.

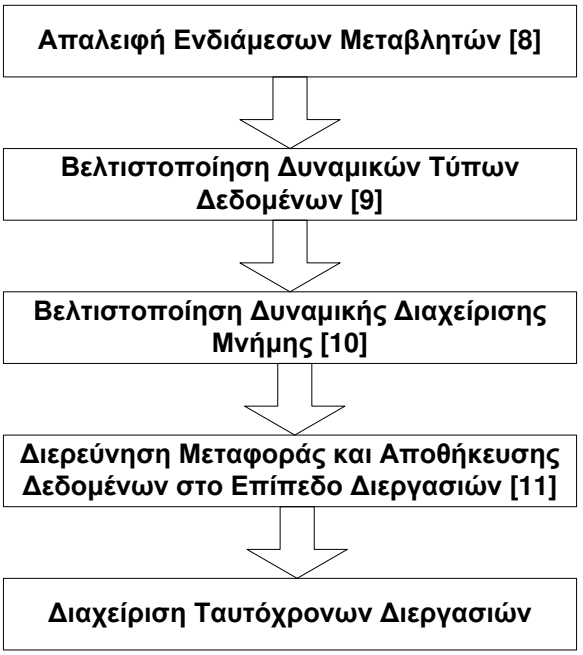
1.2 Δομές δεδομένων

Αντικείμενο της διατριβής αποτελούν οι δομές δεδομένων, οι οποίες μπορεί να είναι στατικές ή δυναμικές. Η διαφορά των δυναμικών με τις στατικές δομές, είναι ότι στην περίπτωση των δυναμικών, το μέγεθος της δομής δεν είναι σταθερό στη διάρκεια εκτέλεσης του προγράμματος. Επιπλέον, η απόδοση της δομής δε μπορεί να καθοριστεί ντετερμινιστικά, καθώς δεν είναι γνωστή η είσοδος του προγράμματος.

Το κυριότερο πλεονέκτημα των δυναμικών δομών έναντι των στατικών είναι η δυναμική εκχώρηση μνήμης ανάλογα με τις ανάγκες τους. Για παράδειγμα, σε μια δυναμική λίστα δεδομένων, η μνήμη εκχωρείται με κάθε στοιχείο που προστίθεται στη δομή και αντίστοιχα απελευθερώνεται με κάθε στοιχείο που αφαιρείται. Έτσι, όταν στο μεγαλύτερο μέρος της εκτέλεσης του προγράμματος η δομή είναι άδεια, δεν καταναλώνεται μνήμη. Το αντίθετο συμβαίνει όταν χρησιμοποιείται κάποια στατική δομή όπως ένας πίνακας. Τότε, η μέγιστη ποσότητα μνήμης εκχωρείται κατά την έναρξη εκτέλεσης του προγράμματος.

Από την άλλη μεριά, το κυριότερο μειονέκτημα των δυναμικών δομών είναι η απρόβλεπτη συμπεριφορά τους. Λόγω της εξάρτησης των δομών από την είσοδο του προγράμματος, ο σχεδιαστής της εφαρμογής καλείται να είναι ιδιαίτερα προσεκτικός, καθώς είναι υπαρκτός ο κίνδυνος της αδυναμίας εκχώρησης μνήμης, σε περίπτωση που μια είσοδος είναι πολύ απαιτητική. Οι εφαρμογές που καλούνται να εκτελέσουν τα σύγχρονα ενσωματωμένα συστήματα, όπως PDA, έξυπνα τηλέφωνα και κονσόλες βιντεοπαιχνιδιών, γίνονται ολοένα πιο πολύπλοκες και δυναμικές. Τέτοιες εφαρμογές μπορεί να είναι η μετάδοση φωνής μέσω διαδικτύου (VoIP), αναπαραγωγή βίντεο, πρόσβαση στο διαδίκτυο, κι αναπαραγωγή τρισδιάστατων εφαρμογών. Σε ένα τέτοιο σύστημα, απαιτούνται δυναμικές δομές δεδομένων, καθώς ο αριθμός των εφαρμογών που εκτελούνται ταυτόχρονα στο σύστημα, αλλά και το είδος τους μεταβάλλεται δυναμικά. Οι παράμετροι που επηρεάζουν τη μεταβολή τους είναι ο χρήστης και το περιβάλλον.

Επιπλέον, οι δομές δεδομένων μπορεί να είναι ταυτόχρονες ή μη-ταυτόχρονες. Στις ταυτόχρονες δεδομένων έχουν πρόσβαση περισσότερα από ένα νήματα, συνήθως σε απρόβλεπτες χρονικές στιγμές κατά τη διάρκεια εκτέλεσης του προγράμματος. Το γεγονός αυτό καθιστά πιο πολύπλοκο τον σχεδιασμό της δομής δεδομένων και αναγκαία την ύπαρξη μηχανισμών που ελέγχουν τις προσβάσεις στη δομή και προστατεύουν από ανεπιθύμητες αλλοιώσεις τόσο τους μηχανισμούς της, όσο και τα



Σχήμα 1.2: Μετα-ροή σχεδιασμού.

δεδομένα. Τα κλειδώματα είναι ο πιο γνωστός από αυτούς τους μηχανισμούς και ιδιαίτερα διαδεδομένος στα ενσωματωμένα συστήματα. Τα τελευταία χρόνια αναζητούνται εναλλακτικοί τρόποι σχεδιασμού των ταυτόχρονων δομών δεδομένων, δίχως την ύπαρξη κλειδωμάτων, ώστε να αποφευχθούν ανεπιθύμητες παρενέργειές τους (blocking, μειωμένο scalability κ.α.).

1.3 Ευρύτερη ροή σχεδιασμού

Τα τελευταία χρόνια, η βιομηχανία δίνει ιδιαίτερη βαρύτητα στην ανάπτυξη εργαλείων αυτοματοποίησης της ροής σχεδιασμού ενσωματωμένων συστημάτων. Για να αντιμετωπιστεί η ολοένα αυξανόμενη πολυπλοκότητα στο σχεδιασμό ενσωματωμένων συστημάτων έχει αναπτυχθεί η Μετα-Ροή Σχεδιασμού Ενσωματωμένων Συστημάτων (Embedded Systems Design Flow) από το Ερευνητικό Ινστιτούτο Interuniversity Microelectronics Center (IMEC) [7] σε συνεργασία με άλλα Ευρωπαϊκά Πανεπιστήμια. Η μετα-ροή σχεδιασμού απεικονίζεται στο Σχήμα 1.2.

Η ροή σχεδιασμού αφορά βήματα/μεθοδολογίες, οι οποίες αναπτύχθηκαν ανεξάρτητα η μια από την άλλη, και παρέχουν τη δυνατότητα στον σχεδιαστή να υλοποιήσει μία εφαρμογή σε ένα ενσωματωμένο σύστημα σε σύντομο χρόνο και με αποδοτικά αποτελέσματα. Υποστηρίζει τόσο στατικές, όσο και δυναμικές εφαρμογές από διάφορα πεδία. Τα βήματα περιγράφονται συνοπτικά παρακάτω:

1. Η **Απαλοιφή Ενδιάμεσων Μεταβλητών** αφορά την απαλοιφή δομών δεδομένων από τον πηγαίο κώδικα της εφαρμογής που δεν είναι απαραίτητες στον

αλγόριθμο γιατί λειτουργούν ως μέσο προσωρινής αποθήκευσης [8].

2. Η **Βελτιστοποίηση Δυναμικών Τύπων Δεδομένων** αφορά την επιλογή κατάλληλων δυναμικών δομών δεδομένων για την εφαρμογή, και βασίζεται σε μετασχηματισμούς του πηγαίου κώδικα της εφαρμογής. Τα κριτήρια είναι η απόδοση, το μέγεθος μνήμης και η κατανάλωση ενέργειας [9].

3. Η **Βελτιστοποίηση Διαχείρισης Δυναμικής Μνήμης** αναφέρεται στην επιλογή κατάλληλου διαχειριστή δυναμικής μνήμης [10].

4. Η **Διερεύνηση Μεταφοράς και Αποθήκευσης Δεδομένων στο Επίπεδο Διεργασιών** αναφερεται στη χρήση μεθόδων χρονοπρογραμματισμού των προσβάσεων των διεργασιών σε κοινά δεδομένα στη μνήμη. Αυτό επιτυγχάνεται με μετασχηματισμούς του πηγαίου κώδικα της εφαρμογής [11].

5. Η **Διαχείριση Ταυτόχρονων Διεργασιών** είναι μία μεθοδολογία που αφορά την αποδοτική, με κριτήριο την ενέργεια, απεικόνιση (mapping) δυναμικών, πραγματικού χρόνου εφαρμογών με παράλληλες διεργασίες σε ετερογενείς πλατφόρμες πολλών επεξεργαστών.

Η παρούσα διατριβή είναι τμήμα του δεύτερου βήματος, δηλαδή της **βελτιστοποίησης δυναμικών τύπων δεδομένων**, το οποίο επεκτείνει στον χώρο των ταυτόχρονων δομών δεδομένων.

Η διατριβή οργανώνεται στα εξής κεφάλαια:

- Στο Κεφάλαιο 2 γίνεται μια σύντομη αναφορά στο αντικείμενο της παρούσας διατριβής και περιγράφονται συνοπτικά τα προβλήματα και οι λύσεις που προτείνει.

- Στο 3ο Κεφάλαιο γίνεται αναλυτική περιγραφή της προτεινόμενης μεθοδολογίας και παρουσιάζεται η εφαρμογή της σε μία σειρά από benchmarks.

- Στο 4ο Κεφάλαιο μελετάται εκτενέστερα η υλοποίηση ταυτόχρονων δομών δεδομένων με χρήση του μοντέλου client-server σε ενσωματωμένα συστήματα, το οποίο αποτελεί τμήμα της μεθοδολογίας που παρουσιάζεται στο προηγούμενο κεφάλαιο.

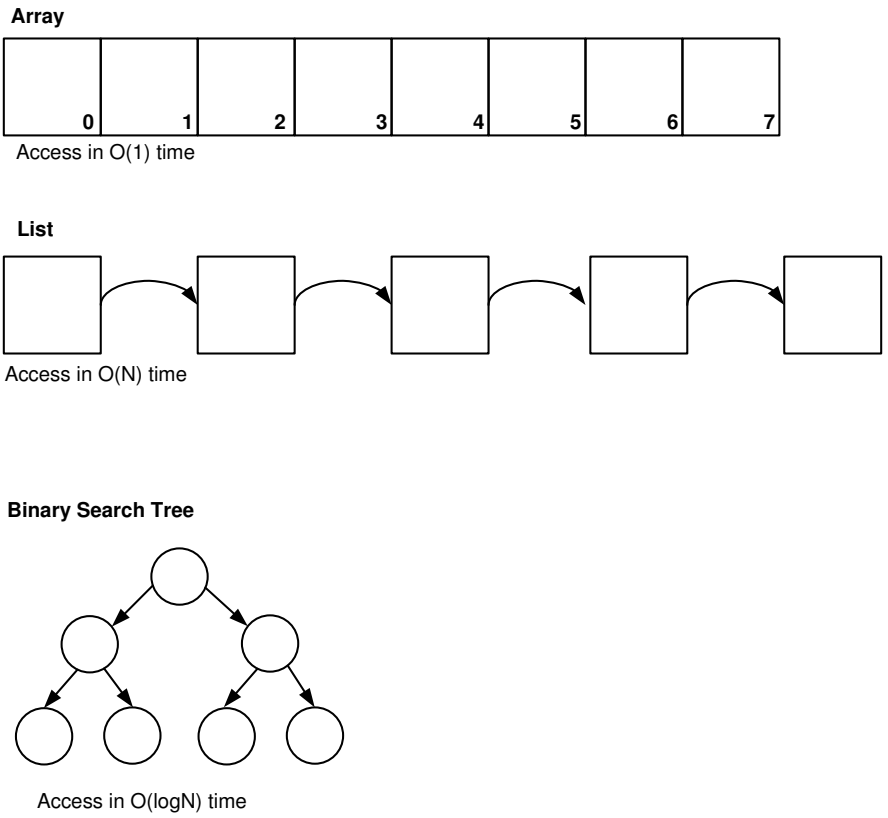
- Στο 5ο Κεφάλαιο δείχνεται η διαδικασία παραμετροποίησης operators από τον χώρο της επεξεργασίας ροών δεδομένων σε ενσωματωμένα συστήματα.

- Τέλος, στο 6ο Κεφάλαιο συνοψίζεται η συνεισφορά της διατριβής και περιγράφονται οι μελλοντικές επεκτάσεις της.

Κεφάλαιο 2

Προσδιορισμός του προβλήματος

Η παρούσα διδακτορική διατριβή καλείται να λύσει το πρόβλημα της βελτιστοποίησης εφαρμογών που κάνουν χρήση ταυτόχρονων δομών δεδομένων. Η βελτιστοποίηση πραγματοποιείται με την εφαρμογή μιας μεθοδολογίας που στοχεύει στην επιλογή κατάλληλης ταυτόχρονης δομής δεδομένων, λαμβάνοντας υπόψη τόσο τα χαρακτηριστικά της εφαρμογής, όσο και της πλατφόρμας στην οποία η εφαρμογή εκτελείται.



Σχήμα 2.1: Δομές δεδομένων.

2.1 Ακολουθιακές και ταυτόχρονες δομές δεδομένων

Ο ρόλος των δομών δεδομένων είναι η οργάνωση των δεδομένων στη μνήμη και η εξυπηρέτηση των αναγκών των εφαρμογών κατά την ώρα εκτέλεσής τους. Ο αλγόριθμος της εφαρμογής πραγματοποιεί λειτουργίες σχετικές με την αποθήκευση, την ανάγνωση και τη διαγραφή δεδομένων, τις οποίες η δομή εξυπηρετεί. Η υλοποίηση αυτών των λειτουργιών εξαρτάται από την εκάστοτε δομή, η οποία μπορεί να ευνοεί συγκεκριμένες ακολουθίες πρόσβασης και αποθήκευσης δεδομένων. Μερικές από τις πιο συχνά χρησιμοποιούμενες δομές δεδομένων απεικονίζονται στο Σχήμα 2.1.

Οι πίνακες είναι οι πιο απλές και ευρέως χρησιμοποιούμενες δομές δεδομένων. Είναι κατάλληλοι όταν απαιτείται στατική διαχείριση μνήμης, δηλαδή όταν ο προγραμματιστής γνωρίζει τον συνολικό χώρο που απαιτεί η εφαρμογή για την αποθήκευση των δεδομένων. Επιπλέον, είναι κατάλληλοι για τυχαίες ακολουθίες προσπελάσεων στα στοιχεία τους και για βελτιστοποίηση του μεγέθους του χώρου αποθήκευσης στη μνήμη. Σε περίπτωση που ο αριθμός των στοιχείων που πρόκειται να αποθηκευτούν στον πίνακα είναι άγνωστος στον προγραμματιστή χρησιμοποιείται δυναμικός πίνακας, όπου το μέγεθός του αυξάνει κατά τη διάρκεια εκτέλεσης της εφαρμογής. Στην περίπτωση αυτή είναι πιθανό μεγάλο μέρος του πίνακα να παραμένει ανεκμετάλλευτο.

Αντίθετα, στις λίστες εκχωρείται και απελευθερώνεται μνήμη δυναμικά, όταν είναι αναγκαίο. Επομένως, συχνά γίνεται καλύτερη διαχείριση της μνήμης σε σχέση με τους πίνακες. Παρόλα αυτά, για την αναζήτηση ενός στοιχείου της λίστας απαιτείται η προσπέλαση όλων των προηγούμενων στοιχείων, γεγονός που συνήθως οδηγεί σε μεγάλο αριθμό προσβάσεων στη μνήμη. Οι δεντρικές δομές είναι κατάλληλες για ταξινόμηση των στοιχείων και προσφέρουν γρήγορο τρόπο αναζήτησής τους. Η διαχείριση της μνήμης είναι δυναμική, παρόμοια με αυτήν των λιστών.

Ακόμη και οι πιο απλές και συνηθισμένες δομές δεδομένων, όπως αυτές που προαναφέρθηκαν, είναι κατάλληλες για διαφορετικά σχήματα πρόσβασης (access patterns). Όπως θα αναλυθεί εκτενέστερα παρακάτω, επιλέγοντας διαφορετικές δομές μπορούμε πετύχουμε διαφορετική απόδοση, κατανάλωση μνήμης και ενέργειας.

Η έννοια του ''ταυτόχρονου'' στα υπολογιστικά συστήματα, αναφέρεται στην εκτέλεση πολλαπλών ανεξάρτητων εργασιών παράλληλα, αντί για ακολουθιακά. Σήμερα, είναι πολύ διαδεδομένα τα ενσωματωμένα συστήματα με πολλαπλούς επεξεργαστές που εκτελούν περισσότερες από μία εργασίες ταυτόχρονα. Συχνά, νήματα (threads) που εκτελούνται ταυτόχρονα χρησιμοποιούν κοινές δομές δεδομένων (shared data structures). Χαρακτηριστικό παράδειγμα είναι οι βάσεις δεδομένων, όπου ένας αριθμός από νήματα αναζητούν, προσθέτουν ή αφαιρούν στοιχεία από μία κοινή βάση. Οι δομές δεδομένων στις οποίες μπορούν να έχουν πρόσβαση περισσότερα από ένα νήματα, ονομάζονται **ταυτόχρονες δομές δεδομένων** (concurrent data structures).

Το *thread safety* είναι μια σημαντική ιδιότητα των ταυτόχρονων δομών δεδομένων

και αφορά το γεγονός ότι σε περίπτωση που περισσότερα από ένα νήματα τροποποιούν στοιχεία της δομής, η δομή θα πρέπει να λειτουργεί όπως έχει προβλεφθεί. Οι λειτουργίες της εφαρμογής μπορεί να είναι *blocking* ή *non-blocking*. Blocking σημαίνει ότι αν ένα νήμα που εκτελεί τη συγκεκριμένη λειτουργία διακόψει τη λειτουργία του, αυτό έχει ως συνέπεια όλα τα υπόλοιπα νήματα που πρόκειται να πραγματοποιήσουν την ίδια λειτουργία να αποτύχουν. Αντίθετα, non-blocking σημαίνει ότι η αποτυχία ενός νήματος στην εκτέλεση μιας λειτουργίας, δεν προκαλεί την αποτυχία των υπόλοιπων νημάτων που πρόκειται να εκτελέσουν την ίδια λειτουργία.

Το thread safety μπορεί να εξασφαλίζεται με διάφορους τρόπους, μεταξύ των οποίων ο πιο διαδεδομένος είναι η χρήση κλειδωμάτων. Πολλαπλά επίπεδα κλειδωμάτων μπορεί να αυξάνουν τον παραλληλισμό, παρέχοντας τη δυνατότητα σε περισσότερα από ένα νήματα να επιτελούν λειτουργίες στη δομή δεδομένων. Παρόλα αυτά, το κόστος της απόκτησης και της απελευθέρωσης μιας σειράς από κλειδώματα μπορεί να είναι υψηλό. Έτσι, σε κάποιες περιπτώσεις η χρήση ενός κλειδώματος μπορεί να παρέχει καλύτερα αποτελέσματα από την χρήση πολυεπίπεδων κλειδωμάτων.

Το μειωμένο scalability που παρέχουν οι δομές δεδομένων βασισμένες σε κλειδώματα, οδήγησε στον σχεδιασμό *lock-free* ταυτόχρονων δομών δεδομένων. Σε μία lock-free δομή δεδομένων περισσότερα από ένα νήματα έχουν συγχρόνως πρόσβαση (δηλαδή, δεν υπάρχει αμοιβαίος αποκλεισμός που επιτυγχάνεται με τα κλειδώματα). Επιπλέον, αν ένα νήμα διακόψει τη λειτουργία του, τα υπόλοιπα νήματα θα πρέπει να είναι σε θέση να ολοκληρώσουν τις δικές τους λειτουργίες (δηλ. δεν υπάρχει mutual exclusion). Οι ατομικές λειτουργίες (π.χ. compare-and-exchange, compare-and-swap) χρησιμοποιούνται στον σχεδιασμό ταυτόχρονων δομών δεδομένων. Οι lock-free δομές δεδομένων περιέχουν συχνά βρόχους που τερματίζουν όταν η ατομική λειτουργία επιτύχει. Αυτό μπορεί να έχει ως συνέπεια νήματα να οδηγηθούν σε starvation. Το συγκεκριμένο πρόβλημα αντιμετωπίζεται με τον σχεδιασμό *wait-free* δομών δεδομένων, όπου είναι εγγυημένο πως κάθε νήμα ολοκληρώνει μία λειτουργία σε συγκεκριμένο αριθμό βημάτων (ή μικρότερο), ανεξάρτητα από τη συμπεριφορά των υπολοίπων νημάτων. Ο σχεδιασμός wait-free δομών δεδομένων είναι ένα ιδιαίτερα δύσκολο πρόβλημα και αποτελεί μία ενεργή ερευνητική περιοχή.

Είναι γεγονός ότι οι lock-free δομές δεδομένων μπορούν να αυξήσουν τις δυνατότητες παραλληλισμού μειώνοντας τον χρόνο αναμονής των νημάτων, γεγονός που μπορεί να οδηγήσει σε αυξημένο scalability. Παρόλα αυτά, συχνά οι επιπλέον έλεγχοι που είναι απαραίτητοι, μπορεί να έχουν ως συνέπεια να απαιτείται μεγάλος αριθμός βημάτων για την πρόσβαση στη δομή, με συνέπεια το αυξημένο overhead. Στην περίπτωση αυτή, η χρήση απλών κλειδωμάτων μπορεί να είναι πιο αποτελεσματική. Αντίστοιχα, όσο αφορά την κατανάλωση ενέργειας, τα spinlocks τείνουν να προκαλούν υψηλή κατανάλωση ενέργειας. Αντίστοιχα όμως, και στις lock-free υλοποιήσεις όλα τα νήματα είναι συνεχώς ενεργά, καθώς κανένα δεν γίνεται suspended από τον kernel του λειτουργικού συστήματος. Έτσι, γίνεται φανερό πως διαφορετικές υλοποιήσεις

των ταυτόχρονων δομών δεδομένων επηρεάζουν με διαφορετικό τρόπο τις μετρικές (metrics) βάση τον οποίων αξιολογείται μία δομή.

2.2 Η σημασία της επιλογής κατάλληλης δομής δεδομένων

Η οργάνωση των δεδομένων στην μνήμη είναι ένα από τα χαρακτηριστικά που καθορίζουν το πόσο αποτελεσματικά αξιοποιούνται οι πόροι του υλικού, όπως η μνήμη και οι δυνατότητες που παρέχει η αρχιτεκτονική για παραλληλισμό. Δεν είναι ασυνήθιστες οι περιπτώσεις όπου μια απλή αλλαγή στη δομή δεδομένων μπορεί να οδηγήσει στη βελτίωση της απόδοσης μιας εφαρμογής κατά τάξεις μεγέθους. Παράδειγμα αποτελούν εφαρμογές από επιστημονικά πεδία κάνουν χρήση αντιστροφής πίνακα [12], πολλαπλασιασμού πινάκων [13], data mining από μεγάλες βάσεις δεδομένων [14] και ανάλυσης γενετικών δεδομένων για εύρεση προτύπων [15]. Σε αυτές τις εφαρμογές έχει τεράστια σημασία η επιλογή της δομής δεδομένων όταν γίνεται προσπάθεια για βελτίωση της απόδοσής τους. Σύμφωνα με το [12], η κατάλληλη επιλογή δομής δεδομένων μπορεί να κάνει την πρόσβαση σε δισδιάστατη υλοποίηση πίνακα έως 20 φορές ταχύτερη.

Σε μία σχετικά πρόσφατη έρευνα, ερευνητές της Google ανέλυσαν τη χρήση της C++ Standard Template Library (STL) [16] σε μεγάλο αριθμό εφαρμογών τους και διαπίστωσαν ότι σε πολλές περιπτώσεις έμπειροι προγραμματιστές επέλεξαν μη-βέλτιστες δομές δεδομένων στις εφαρμογές τους [17]. Στη συγκεκριμένη έρευνα, αναφέρεται πως αλλάζοντας μία δομή δεδομένων σε μία από τις εφαρμογές επιτεύχθηκε μείωση του χρόνου εκτέλεσης κατά 17%. Στο χώρο των ενσωματωμένων συστημάτων, η ακατάλληλη επιλογή δομής δεδομένων έχει αρνητικό αντίκτυπο σε διάφορες μετρικές, όπως η απόδοση και η κατανάλωση ενέργειας [9]. Έτσι, οι κατάλληλες δομές δεδομένων σε εφαρμογές ενσωματωμένων συστημάτων είναι ένα ιδιαίτερα σημαντικό πρόβλημα.

Στην πραγματικότητα, οι περισσότεροι προγραμματιστές αποφεύγουν να σχεδιάζουν δομές δεδομένων. Συνήθως βασίζονται σε μια βιβλιοθήκη, όπως η STL και υποθέτουν πως ο σχεδιαστής της βιβλιοθήκης έχει σχεδιάσει αποτελεσματικά τη συγκεκριμένη δομή. Παρόλο που αυτό ισχύει στις συνηθισμένες περιπτώσεις, υπάρχουν διάφορα σενάρια όπου η επιλεγμένη δομή αποδεικνύεται ακατάλληλη και υπάρχει πολύς χώρος για βελτιστοποίηση.

Όταν οι προγραμματιστές καλούνται να επιλέξουν χειροκίνητα μία δομή δεδομένων για μία εφαρμογή, συνήθως βασίζονται στην ασυμπτωτική ανάλυση. Παρόλο που η ασυμπτωτική ανάλυση είναι ένα πολύ αποτελεσματικό εργαλείο για την κατανόηση των χαρακτηριστικών των δομών δεδομένων, συχνά στα πραγματικά συστήματα οδηγεί σε λανθασμένα συμπεράσματα. Για παράδειγμα, αν συγκριθεί το STL set (που είναι υλοποιημένο ως red-black tree) με το unordered_set που είναι υλοποιημένο ως hash table (πίνακας κατακερματισμού), το set έχει χειρότερη ασυμπτωτική συμπερι-

φορά, αλλά σχεδόν πάντοτε στις σύγχρονες αρχιτεκτονικές έχει ταχύτερη αναζήτηση όταν ο αριθμός των στοιχείων είναι μικρότερος του 200. Υπάρχουν περιπτώσεις όπου δομές δεδομένων με παρόμοια ασυμπτωτική συμπεριφορά παρέχουν πολύ διαφορετικά αποτελέσματα σε πραγματικά συστήματα. Για παράδειγμα, τα splay trees [18] έχουν σχεδόν πάντοτε υψηλότερη απόδοση από τα red-black trees σε πραγματικές εφαρμογές, παρόλο που παρουσιάζουν την ίδια ασυμπτωτική πολυπλοκότητα. Είναι γεγονός πως η ασυμπτωτική πολυπλοκότητα σχεδιάστηκε ως μία ενιαία βάση για την εκτίμηση της απόδοσης αλγορίθμων και όχι δομών δεδομένων. Επιπλέον, πολλές φορές από τη στιγμή που θα επιλεγεί ένας αλγόριθμος για μία εφαρμογή, σπάνια δίνεται βαρύτητα στην κατάλληλη επιλογή δομής δεδομένων. Παρόλα αυτά, η επιλογή της είναι σημαντικό και πολύπλοκο πρόβλημα.

Όσο αφορά τις ταυτόχρονες δομές δεδομένων, είναι γεγονός πως οι υλοποιήσεις δίχως κλειδώματα παρέχουν αυξημένο scalability σε σχέση με αυτές που χρησιμοποιούν κλειδώματα. Παρόλα αυτά, ο βαθμός συμφόρησης πέρα από τον οποίο μία lock-free υλοποίηση ξεκινά να παρέχει υψηλότερη απόδοση σε σχέση με την lock-based δεν είναι γνωστός εκ των προτέρων χωρίς δοκιμές. Επιπλέον, διαφορετικές αρχιτεκτονικές υλοποιούν τα κλειδώματα με διαφορετικό τρόπο. Επομένως, μία υλοποίηση βασισμένη σε κλειδώματα σε μία συγκεκριμένη αρχιτεκτονική δεν είναι βέβαιο πως θα έχει τα ίδια αποτελέσματα αν εκτελεστεί σε μία αρχιτεκτονική με διαφορετικά χαρακτηριστικά. Οι παραπάνω παρατηρήσεις δείχνουν την ανάγκη για προσεκτική επιλογή των ταυτόχρονων δομών και της ανάγκης που υπάρχει για μεθοδολογίες που θα βοηθήσουν τον προγραμματιστή να επιλέξει την κατάλληλη. Ο χώρος σχεδιασμού των ταυτόχρονων δομών δεδομένων, περιέχει μεγάλο αριθμό σχεδιαστικών επιλογών, με συνέπεια να είναι απαραίτητη μία μεθοδολογία επιλογής της κατάλληλης δομής σε κάθε περίπτωση.

2.3 Στόχοι και Συνεισφορά

Στην προηγούμενη ενότητα τονίστηκε η σημασία της επιλογής κατάλληλων δομών δεδομένων στις σύγχρονες εφαρμογές. Εδώ, παρουσιάζονται τα προβλήματα που αντιμετωπίζονται και οι λύσεις που προτείνονται στην παρούσα διατριβή.

Ο στόχος της παρούσας διατριβής είναι να παρέχει μεθοδολογία και εργαλεία μέσω των οποίων οι προγραμματιστές θα μπορούν βελτιστοποιούν εφαρμογές που κάνουν χρήση ταυτόχρονων δομών δεδομένων, επιλέγοντας την καταλληλότερη σε κάθε περίπτωση. Με τον τρόπο αυτό, θα μπορούν να ικανοποιηθούν σχεδιαστικοί περιορισμοί όπως η απόδοση και η κατανάλωση ενέργειας του συστήματος.

Η κύριες συνεισφορές της διατριβής συνοψίζονται ως εξής:

1. Προτείνεται μία μεθοδολογία **εξερεύνησης του χώρου σχεδιασμού των ταυτόχρονων δομών δεδομένων**. Η μεθοδολογία έχει ως στόχο την αξιολόγηση ενός

αριθμού διαφορετικών υλοποιήσεων ταυτόχρονων δομών δεδομένων με σκοπό την επιλογή της καταλληλότερης υλοποίησης. Ο χώρος σχεδιασμού των ταυτόχρονων δομών δεδομένων είναι μεγάλος, καθώς έχουν προταθεί διαχρονικά διάφορες υλοποιήσεις. Για παράδειγμα, υπάρχουν υλοποιήσεις βασισμένες σε κλειδώματα, άλλες που κάνουν χρήση ατομικών λειτουργιών δίχως χρήση κλειδωμάτων, βασισμένων σε ανταλλαγή μηνυμάτων μεταξύ των πυρήνων κλπ. Οι συγκεκριμένες υλοποιήσεις έχουν διαφορετική επίδραση στη απόδοση, στην κατανάλωση ενέργειας, στο fairness κλπ. Η μεθοδολογία παρέχει ένα πλαίσιο αξιολόγησης των υλοποιήσεων και δίνει τη δυνατότητα επιλογής της καταλληλότερης.

2. Παρέχεται μία **ροή εργαλείων που αυτοματοποιεί μέρος της εφαρμογής της μεθοδολογίας**. Η μεθοδολογία υποστηρίζεται από ένα σύνολο εργαλείων, το οποίο αυτοματοποιεί πολλά βήματα της μεθοδολογίας. Η βελτιστοποίηση βασισμένη σε ad-hoc λύσεις και μη συστηματικές δοκιμές από τους προγραμματιστές είναι πιθανό να οδηγήσει σε μη βέλτιστες λύσεις. Στη συγκεκριμένη διατριβή προτείνεται η χρήση ενός συνόλου εργαλείων και βιβλιοθηκών που αυτοματοποιεί πολλά στάδια, με συνέπεια την αποτελεσματική και γρήγορη διαδικασία επιλογής δομών δεδομένων και βελτιστοποίησης της εφαρμογής.

3. **Αξιολογούνται υλοποιήσεις δομών δεδομένων με βάση σημαντικό αριθμό κριτηρίων**. Το κριτήριο της απόδοσης είναι το βασικό κριτήριο βάση του οποίου έως πρόσφατα βελτιστοποιούνταν οι εφαρμογές που εκτελούνταν σε γενικής χρήσης συστήματα. Οι βιβλιοθήκες δυναμικών δομών δεδομένων (π.χ. STL) είναι σχεδιασμένες με έμφαση στην απόδοση. Παρόλα αυτά, στον χώρο των ενσωματωμένων συστημάτων είναι εξίσου σημαντική παράμετρος η κατανάλωση ενέργειας. Παράλληλα, το fairness και το scalability είναι μετρικές που καθορίζουν σε μεγάλο βαθμό την αποτελεσματικότητα των ταυτόχρονων δομών δεδομένων. Η συγκεκριμένη μεθοδολογία λαμβάνει υπόψη της όλα τα προαναφερθέντα κριτήρια και παρέχει λύσεις που έχουν αξιολογηθεί ως προς αυτά.

Επιπλέον, η μεθοδολογία επιτρέπει την **αναγνώριση των χαρακτηριστικών των εφαρμογών και των αρχιτεκτονικών που επιδρούν στην καταλληλότητα των υλοποιήσεων των ταυτόχρονων δομών δεδομένων**. Χαρακτηριστικά του αλγορίθμου της εφαρμογής που βελτιστοποιείται, όπως ο ρυθμός με τον οποίο γίνονται οι προσβάσεις στην δομή από κάθε νήμα ή ο αριθμός των νημάτων (και άρα ο βαθμός της συμφόρησης (contention)), επηρεάζουν σε μεγάλο βαθμό την καταλληλότητα κάθε υλοποίησης. Από την άλλη πλευρά, τα διαθέσιμα χαρακτηριστικά της πλατφόρμας που μπορούν να χρησιμοποιηθούν για τον συγχρονισμό των προσβάσεων στην δομή (π.χ. κλειδώματα) ή η ιεραρχία μνήμης επιδρούν και αυτά στην καταλληλότητα των υλοποιήσεων. Στην παρούσα διατριβή αναγνωρίζονται τα παραπάνω χαρακτηριστικά και γίνεται εκτενής αναφορά σε αυτά.

Υιοθετώντας την μεθοδολογία που προτείνεται στο πλαίσιο της παρούσας διατριβής πετυχαίνεται η προσαρμογή της δομής δεδομένων τόσο στα χαρακτηριστικά της εφαρμογής, όσο και στα χαρακτηριστικά της αρχιτεκτονικής. Με τον τρόπο αυτό, μπορούν να ικανοποιηθούν οι σχεδιαστικοί περιορισμοί όπως η απόδοση και η κατανάλωση ενέργειας του συστήματος.

Τέλος, δείχνεται πως η συγκεκριμένη μεθοδολογία δεν περιορίζεται αποκλειστικά στο χώρο των ταυτόχρονων δομών δεδομένων. Αντίθετα, μπορεί να χρησιμοποιηθεί για την παραμετροποίηση υλοποιήσεων operators από τον χώρο της επεξεργασίας ροών δεδομένων. Η μεθοδολογία απλοποιεί τη διαδικασία εντοπισμού trade-offs μεταξύ διαφόρων μετρικών (throughput, latency, κατανάλωση ενέργειας και μέγεθος απαιτούμενης μνήμης) επιλέγοντας διαφορετικές παραμετροποιημένες υλοποιήσεις operators εκτελεσμένων σε ενσωματωμένες αρχιτεκτονικές.

2.4 Σχετική βιβλιογραφία

Έχουν προταθεί στο παρελθόν διάφορες λύσεις για τη βελτιστοποίηση της αποθήκευσης και της προσπέλασης δεδομένων [19] [20] [21] [22], οι οποίες αφορούν διάφορα επίπεδα σχεδιασμού των ενσωματωμένων συστημάτων. Μία τέτοια προσέγγιση είναι η μεθοδολογία για την Ανάλυση της Μεταφοράς και Αποθήκευσης Δεδομένων (Data Transfer and Storage Exploration – DTSE) [11]. Επιπλέον, υπάρχουν διαθέσιμες βιβλιοθήκες δομών δεδομένων, όπως οι Standared Template Library (STL) για τη γλώσσα C++ [16] και η Generic Data Structures Library (GDSL) για τη γλώσσα C [23]. Σημαντική βιβλιογραφία υπάρχει στον χώρο των μετασχηματισμών βρόχων (loop transformations), με στόχο την αποτελεσματικότερη αξιοποίηση της ιεραρχίας μνήμης [24].

2.4.1 Εργασίες σε εργαλεία επιλογής δομών δεδομένων

Η επιλογή της καταλληλότερης δομής δεδομένων είναι ένα πρόβλημα που συχνά παραβλέπεται από τους προγραμματιστές. Συνήθως βασίζονται στους σχεδιαστές των βιβλιοθηκών για την επιλογή της καλύτερης δομής για τις πιο συνηθισμένες περιπτώσεις και αποδέχονται τα αποτελέσματα. Αυτή η προσέγγιση αφήνει μεγάλα περιθώρια για βελτιστοποίηση. Όταν οι προγραμματιστές επιλέγουν μία συγκεκριμένη υλοποίηση βασίζονται στην ασυμπτωτική ανάλυση, η οποία συχνά οδηγεί σε ακατάλληλες επιλογές σε πραγματικές εφαρμογές. Η ασυμπτωτική ανάλυση σχεδιάστηκε και ανάλυση αλγορίθμων και όχι για επιλογή και παραμετροποίηση δομών δεδομένων.

Το ζήτημα της επιλογής δομών δεδομένων σε διάφορα πλαίσια έχει ερευνηθεί αρκετά στο παρελθόν [25, 26, 27, 28, 29]. Οι Jung και Clark προτείνουν μεθόδους δυναμικής ανάλυσης για τον εντοπισμό των δομών δεδομένων και του interface των συναρτήσεων τους [25]. Δείχνουν ότι ο τρόπος με τον οποίο οι συναρτήσεις επιδρούν

με τις δομές δεδομένων παρέχει σημαντικές πληροφορίες που είναι χρήσιμες στην επιλογή κατάλληλων υλοποιήσεων. Έχει επίσης προταθεί υποστήριξη επιλογής κατάλληλων δομών δεδομένων σε επίπεδο γλωσσών προγραμματισμού. Για παράδειγμα σε υψηλού επιπέδου γλώσσες προγραμματισμού, όπως η SETL είναι αδύνατο να επιλεγούν συγκεκριμένες υλοποιήσεις. Όλες οι δομές δεδομένων ορίζονται ως αφηρημένοι τύποι δεδομένων (abstract data types) και ο compiler αναλαμβάνει την επιλογή συγκεκριμένων υλοποιήσεων [27]. Στη συγκεκριμένη περιοχή έχει γίνει έρευνα βασισμένη κυρίως σε στατική ανάλυση των προγραμμάτων [28].

Τα εργαλεία Chameleon [29] και Perflint [26] χρησιμοποιούνται για τη συλλογή πληροφοριών κατά τη διάρκεια εκτέλεσης των προγραμμάτων, όπως για παράδειγμα, ο αριθμός κλήσεων συγκεκριμένων λειτουργιών στις δομές δεδομένων. Το Chameleon αφορά κώδικα Java και συλλέγει στατιστικά από τον garbage collector, ενώ το Perflint αφορά κώδικες C++. Βάση των στατιστικών που έχουν συλλεχθεί αποφασίζεται αν κάποια συγκεκριμένη υλοποίηση δομής δεδομένων χρειάζεται να αλλάξει. Το εργαλείο Brainy [30] χρησιμοποιεί μεθόδους machine learning για την κατασκευή μοντέλων συμπεριφοράς των δομών δεδομένων στις εφαρμογές που πρόκειται να βελτιστοποιηθούν.

Επιπλέον, υπάρχουν αρκετές εργασίες που αφορούν τον εντοπισμό μη αποτελεσματικής αξιοποίησης των δομών δεδομένων [31, 32, 33, 34]. Για παράδειγμα, στην [31] προτείνεται μία συστηματική μεθοδολογία για τον εντοπισμό δομών δεδομένων που καταναλώνουν περισσότερη μνήμη από όση πραγματικά χρειάζονται. Εισάγουν τη μετρική *health* που δείχνει τον τρόπο με τον οποίο η δομή δεδομένων είναι οργανωμένη στο χώρο μνήμης που καταλαμβάνει και αξιολογούν την αποτελεσματικότητά της οργάνωσής της [33]. Οι Xu και Rountev παρουσιάζουν εργαλεία βασισμένα σε στατική και δυναμική ανάλυση που εντοπίζουν μη αποτελεσματικές υλοποιήσεις δομών δεδομένων Αρχικά, εντοπίζουν τις συναρτήσεις του interface της δομής δεδομένων (π.χ. ADD/GET) χρησιμοποιώντας στατική ανάλυση. Έπειτα, με συνδυασμό στατικής και δυναμικής ανάλυσης εντοπίζουν τον τρόπο με τον οποίο καλούνται κατά τη διάρκεια εκτέλεσης του προγράμματος.

Το ζήτημα της εξερεύνησης του χώρου λύσεων δομών δεδομένων προτάθηκε αρχικά εδω: [9]. Η συγκεκριμένη εργασία παρουσιάζει τη μεθοδολογία Βελτιστοποίησης Δυναμικών Δομών Δεδομένων (Dynamic Data Type Refinement – DDTR). Αφορά τον εντοπισμό trade-offs (ανταλλαγμάτων) μεταξύ υλοποιήσεων μη-ταυτόχρονων δομών δεδομένων, χρησιμοποιώντας ως μετρικές τον χρόνο εκτέλεσης, το μέγεθος μνήμης και την κατανάλωση ενέργειας. Υπάρχουν σημαντικές διαφορές ανάμεσα στη μεθοδολογία DDTR και σε αυτήν που προτείνεται στην παρούσα εργασία: Η μεθοδολογία DDTR είναι ουσιαστικά μία βιβλιοθήκη δομών δεδομένων. Δεν παρέχει συστηματικό τρόπο για την αποφυγή της εκτέλεσης και της αξιολόγησης υλοποιήσεων οι οποίες είτε δεν υποστηρίζονται από την πλατφόρμα στην οποία η εφαρμογή εκτελείται, είτε αναμένεται τα αποτελέσματά τους να είναι μη ικανοποιητικά. Για παράδειγμα, στη

Πίνακας 2.1: Διαφορές μεταξύ της υπάρχουσας μεθοδολογίας DDTR και της προτεινόμενης μεθοδολογίας

	Υπάρχουσα Μεθοδολογία [9] (DDTR)	Προτεινόμενη Μεθοδολογία
Χώρος σχεδιαστικών επιλογών	Ουρά, Σωρός, Deque, Λίστα	Ουρά, Σωρός, Deque, associative arrays δέντρα...
Ταυτόχρονες δομές δεδομένων	Όχι	Ναι
Υλοποιήσεις που αξιολογούνται	Λίστες Πίνακες	Λίστες Πίνακες Hash tables Δέντρα
Μη αξιολόγηση υλοποιήσεων που δεν υποστηρίζονται	Όχι	Ναι
Μεθοδολογία	Βασισμένη σε ad-hoc υλοποιήσεις	Βασισμένη στον ορισμό του χώρου σχεδιαστικών επιλογών
Αξιολόγηση υλοποιήσεων ανεξάρτητων των εφαρμογών	Ναι	Ναι
Αξιολόγηση υλοποιήσεων εξαρτημένων από τις εφαρμογές	Όχι	Ναι
Αξιολόγηση υλοποιήσεων ανεξάρτητων της αρχιτεκτονικής	Ναι	Ναι
Αξιολόγηση υλοποιήσεων εξαρτημένων από την αρχιτεκτονική	Όχι	Ναι
Μετρικές	χρόνος εκτέλεσης μέγεθος μνήμης κατανάλωση ενέργειας	χρόνος εκτέλεσης μέγεθος μνήμης κατανάλωση ενέργειας throughput, operations per thread fairness
Scalability	Ad-hoc προσθήκη νέων υλοποιήσεων	Συστηματική προσθήκη νέων υλοποιήσεων

μεθοδολογία DDTR δεν υπάρχει τρόπος να μην εξεταστεί ένα δέντρο, ενώ η εφαρμογή χρειάζεται μία δομή δεδομένων με FIFO σχήμα πρόσβασης. Αυτό έχει ως αποτέλεσμα να εκτελείται και να αξιολογείται μεγαλύτερος αριθμός υλοποιήσεων κατά τη διαδικασία της εξερεύνησης του χώρου λύσεων από αυτόν που πραγματικά απαιτείται. Για της αντιμετώπιση αυτού του προβλήματος, η μεθοδολογία DDTR περιορίζεται ουσιαστικά σε απλές δομές δεδομένων, όπως λίστες και πίνακες. Δεν έχει εφαρμοστεί ως τώρα σε εφαρμογές που κάνουν χρήση πιο πολύπλοκων δομών. Για παράδειγμα, στην μεθοδολογία DDTR δεν μπορούν να ενσωματωθούν associative arrays, γιατί θα αξιολογούνταν μαζί με λίστες και πίνακες, γεγονός που θα αλλοίωνε την λειτουργικότητα της εφαρμογής. Επιπλέον, δεν είναι δυνατό να επεκταθεί σε ταυτόχρονες δομές δεδομένων: Δεν παρέχει υποστήριξη για την μη εξέταση δομών που κάνουν χρήση στοιχείων συγχρονισμού (synchonization primitives) σε πλατφόρμες που δεν υποστηρίζουν αντίστοιχα στοιχεία. Επομένως, η μεθοδολογία DDTR είναι ανεξάρτητη της εφαρμογής και της πλατφόρμας και είναι υψηλού επιπέδου αφαίρεσης. Λόγω του ότι στερείται τη δυνατότητα να επεκταθεί σε πολύπλοκες δομές δεδομένων που εξαρτώνται από τα χαρακτηριστικά της πλατφόρμας περιορίζεται σε απλές μη-ταυτόχρονες δομές δεδομένων. Η ποιοτική σύγκριση ανάμεσα στις δύο μεθοδολογίες απεικονίζεται στον Πίνακα 2.1.

Η προσέγγιση που προτείνεται στην παρούσα εργασία είναι πολύ διαφορετική: Ορίζουμε τον χώρο σχεδιαστικών επιλογών (design space) των δομών δεδομένων και εντοπίζουμε τα χαρακτηριστικά της πλατφόρμας και της εφαρμογής που καθιστούν ακατάλληλες συγκεκριμένες σχεδιαστικές επιλογές. Οι επιλογές αυτές αφαιρούνται από τον χώρο σχεδιασμού. Με τον τρόπο αυτό, ο χώρος λύσεων ''μετατρέπεται'' από ανεξάρτητος της πλατφόρμας και της εφαρμογής σε εξαρτημένος από το συγκεκριμένο πλαίσιο στο οποίο χρησιμοποιείται. Έτσι, πέρα από το γεγονός ότι ο χώρος λύσεων μειώνεται, η μεθοδολογία μπορεί να εφαρμοστεί σε πολύ μεγαλύτερο εύρος εφαρμογών και ενσωματωμένων συστημάτων, που υποστηρίζουν διαφορετικά στοιχεία συγχρονισμού. Σε αντίθεση με τη μεθοδολογία DDTR, η μεθοδολογία που προτείνεται στη συγκεκριμένη εργασία:

- Υποστηρίζει την αξιολόγηση μόνο δυναμικών δομών δεδομένων που έχουν νόημα στο συγκεκριμένο πλαίσιο βελτιστοποίησης και υπάρχει πιθανότητα να παράξουν αξιόλογα αποτελέσματα, αντί για όλες τις διαθέσιμες υλοποιήσεις.

- Υποστηρίζει την αξιολόγηση υλοποιήσεων εξαρτημένων από την πλατφόρμα (platform-specific) στην οποία η εφαρμογή εκτελείται, αντί μόνο υλοποιήσεις υψηλού επιπέδου αφαίρεσης, μη εξαρτημένων από την εφαρμογή (plaform-independent).

- Ενσωματώνει μεγάλο αριθμό ταυτόχρονων και μη-ταυτόχρονων δομών δεδομένων, όπως hash-tables, skip-lists, και δέντρων.

Τέλος, η μεθοδολογία DDTR αποτελεί ουσιαστικά ένα μικρό κομμάτι της μεθο-

δολογίας που περιγράφεται στην παρούσα εργασία. Ένας αντίστοιχος χώρος σχεδιαστικών επιλογών παρουσιάζεται στο [10]. Αφορά τη βελτιστοποίηση δυναμικών διαχειριστών μνήμης για ενσωματωμένα συστήματα και είναι συμπληρωματικός στην προτεινόμενη μεθοδολογία.

2.4.2 Εργασίες σε ταυτόχρονες δομές δεδομένων

Υπάρχει μεγάλος αριθμός εργασιών που προτείνει ταυτόχρονες δομές δεδομένων, βασισμένες ή μη σε κλειδώματα, για συστήματα γενικού σκοπού, όπως ουρές (queues), δέντρα και hash tables [35]. Στη συγκεκριμένη μεθοδολογία, εντοπίζουμε τις σχεδιαστικές επιλογές των συγκεκριμένων δομών δεδομένων τις οποίες ενσωματώνουμε στον χώρο σχεδιαστικών επιλογών. Έτσι, θεωρούμε τις προτεινόμενες υλοποιήσεις δομών δεδομένων ως κάποιες από τις λύσεις οι οποίες σχηματίζουν το χώρο που θα πρέπει να εξερευνηθεί από τους προγραμματιστές, ώστε να βρεθεί η καταλληλότερη υλοποίηση. Επιπλέον, οι προγραμματιστές, αντί να επιλέξουν αυθαίρετα κάποια από τις προτεινόμενες υλοποιήσεις ή να υλοποιήσουν μεγάλο αριθμό λύσεων προς αξιολόγηση, μπορούν να χρησιμοποιήσουν την προτεινόμενη μεθοδολογία ώστε να επιλέξουν την καταλληλότερη ταυτόχρονη δομή δεδομένων για την εφαρμογή που πρόκειται να βελτιστοποιηθεί.

Είναι διαθέσιμος σημαντικός αριθμός από βιβλιοθήκες ταυτόχρονων δομών δεδομένων, όπως η NOBLE [36] και η Practical Lock-Free Data Structures [37]. Οι βιβλιοθήκες αυτές παρέχουν υλοποιήσεις ταυτόχρονων δομών δεδομένων που είναι σχεδιασμένες για να ενσωματώνονται απευθείας στις εφαρμογές. Στην παρούσα εργασία δεν προτείνουμε μία βιβλιοθήκη, αλλά μία μεθοδολογία επιλογής της καταλληλότερης δομής δεδομένων για συγκεκριμένη εφαρμογή που εκτελείται σε συγκεκριμένο ενσωματωμένο σύστημα. Επιπλέον, η μεθοδολογία υποστηρίζεται όχι μόνο από μία βιβλιοθήκη ταυτόχρονων δομών δεδομένων, αλλά και από ένα σύνολο άλλων εργαλείων που υποστηρίζουν την αυτοματοποίηση διαφόρων βημάτων της μεθοδολογίας: Εργαλεία για την απενεργοποίηση συγκεκριμένων σχεδιαστικών επιλογών, αυτοματοποίηση της εξερεύνησης του χώρου λύσεων, της συλλογής αποτελεσμάτων και του εντοπισμού των σημείων Pareto.

Διάφορες εργασίες παρουσιάζουν τον εντοπισμό trade-offs κατά τη χρήση διαφόρων αλγορίθμων και μεθόδων συγχρονισμού των προσβάσεων σε δεδομένα από περισσότερα από ένα νήματα. Για παράδειγμα, στο [38] συγκρίνεται η απόδοση και η κατανάλωση ενέργειας των κλειδωμάτων σε σχέση με την transactional memory. Αντίστοιχη σύγκριση σε αρχιτεκτονική Haswell παρουσιάζεται στο [39]. Οι Moreshet et. al. προτείνουν τη χρήση χαμηλής κατανάλωσης ενέργειας δομών δεδομένων χωρίς κλειδώματα με χρήση transactional memory υλοποιημένης σε υλικό [40]. Άλλες εργασίες αφορούν την εξέταση διαφόρων επιλογών συγχρονισμού σε αρχιτεκτονικές NUMA. Για παράδειγμα, μία σύγκριση μεταξύ υλοποιήσεων βασισμένων σε message-passing

(ανταλλαγή μηνυμάτων) και shared memory σε αρχιτεκτονικές Xeon και SPARK παρουσιάζεται στο [41], ενώ οι Dice et al. προτείνουν αποδοτικές λύσεις βασισμένες σε κλειδώματα [42]. Τέλος, η αξιολόγηση της απόδοσης και της κατανάλωσης ενέργειας υλοποιήσεων δίχως κλειδώματα σε αρχιτεκτονικές Nehalem παρουσιάζεται στο [43]. Οι παραπάνω τεχνικές αφορούν συστήματα γενικού σκοπού. Η συγκεκριμένη μεθοδολογία αφορά ενσωματωμένα συστήματα με μικρότερο αριθμό πυρήνων και ισχυρούς σχεδιαστικούς περιορισμούς.

2.4.3 Διαφοροποιήσεις της προτεινόμενης μεθοδολογίας σε σχέση με τις υπάρχουσες λύσεις

Οι υπάρχουσες λύσεις παρουσιάζουν διάφορους περιορισμούς, οι οποίοι αντιμετωπίζονται στην παρούσα διατριβή και παρουσιάζονται στον Πίνακα 2.2. Συνοψίζονται στα παρακάτω κύρια σημεία:

1. **Οι υπάρχουσες λύσεις δεν αφορούν ταυτόχρονες δομές δεδομένων.** Οι λύσεις που έχουν προταθεί ως τώρα στη βιβλιογραφία για βελτιστοποίηση δομών δεδομένων αφορούν αποκλειστικά μη ταυτόχρονες δομές δεδομένων. Παρόλα αυτά, σήμερα όπου τα ενσωματωμένα συστήματα που στοχεύουν σε υψηλές επιδόσεις είναι συνήθως πολυπύρηνα, εκτελούν συχνά εφαρμογές που βασίζονται σε ταυτόχρονες δομές δεδομένων. Η διδακτορική διατριβή καλύπτει αυτό το κενό προτείνοντας μια μεθοδολογία βελτιστοποίησης που αφορά τη βελτιστοποίηση εφαρμογών που κάνουν χρήση ταυτόχρονων δομών δεδομένων.

2. **Ελλιπής εξερεύνηση του χώρου λύσεων.** Οι υπάρχουσες μεθοδολογίες εξερευνούν μόνο ένα μέρος του συνόλου των διαθέσιμων λύσεων, όπως, για παράδειγμα, οι δομές δεδομένων που περιλαμβάνονται στην STL. Στην περίπτωση αυτή είναι δυνατό να βρεθεί μια καλή λύση, η οποία όμως θα αποτελεί τοπικό βέλτιστο στο σύνολο των λύσεων. Η συγκεκριμένη διατριβή αντιμετωπίζει το πρόβλημα της εξερεύνησης στο σύνολο του χώρου λύσεων.

3. **Βελτιστοποίηση ως προς τις επιδόσεις.** Οι προτεινόμενες λύσεις αφορούν την επιλογή κατάλληλων δομών δεδομένων με αποκλειστικό κριτήριο τις επιδόσεις. Ο λόγος είναι ότι οι λύσεις αυτές προέρχονται από συστήματα γενικού σκοπού, όπου πραγματικά, το βασικό κριτήριο σχεδιασμού είναι οι υψηλές επιδόσεις. Αυτό δεν ισχύει για τα ενσωματωμένα συστήματα όπου πέρα από την επίδοση, η κατανάλωση ενέργειας, το μέγεθος μνήμης είναι εξίσου σημαντικά κριτήρια. Η παρούσα διατριβή καλύπτει αυτό το κενό παρέχοντας στον σχεδιαστή λύσεις που έχουν εξεταστεί ως προς όλα τα προαναφερθέντα κριτήρια σχεδιασμού, πετυχαίνοντας την βελτιστοποίηση πολλών παραμέτρων ταυτόχρονα.

Πίνακας 2.2: Ποιοτική σύγκριση ανάμεσα στην προτεινόμενη μεθοδολογία και στις υπάρχουσες

	Προτεινόμενη Μεθοδολογία	DDTR [9]	Υπάρχουσες βιβλιοθήκες ταυτ. δομών [36][37]	Υπάρχουσες υλοποιήσεις ταυτ. δομών [35]
Εισαγωγή στον πηγαίο κώδικα της εφαρμογής	χειροκίνητα	χειροκίνητα	χειροκίνητα	χειροκίνητα
Απενεργοποίηση μη υποστηριζόμενων υλοποιήσεων	ναι	όχι	όχι	όχι
Ταυτόχρονες δομές δεδομένων	ναι	όχι	ναι	ναι
Εξερεύνηση χώρου λύσεων	ναι	ναι	όχι	όχι
Αποτελέσματα	απόδοση μέγεθος μνήμης fairness	απόδοση μέγεθος μνήμης	όχι	όχι

Κεφάλαιο 3

Μεθοδολογία επιλογής κατάλληλων ταυτόχρονων δομών δεδομένων

Τα σύγχρονα πολυπύρηνα ενσωματωμένα συστήματα εκτελούν συχνά εφαρμογές που βασίζονται σε ταυτόχρονες δομές δεδομένων. Η επιλογή κατάλληλων δομών δεδομένων για μια συγκεκριμένη εφαρμογή είναι συνήθως πολύπλοκη και χρονοβόρα διαδικασία. Κάθε σχεδιαστική επιλογή επηρεάζει την απόδοση και την κατανάλωση ενέργειας του ενσωματωμένου συστήματος στο οποίο η εφαρμογή εκτελείται με συχνά απρόβλεπτο τρόπο. Η σχεδιαστική πολυπλοκότητα αντιμετωπίζεται από τους προγραμματιστές ενσωματωμένων συστημάτων με την υιοθέτηση αυθαίρετων (ad-hoc) λύσεων, που συχνά οδηγούν σε μη βέλτιστα αποτελέσματα. Ως λύση σε αυτό το πρόβλημα προτείνουμε μία ημιαυτόματη μεθοδολογία για την βελτιστοποίηση των εφαρμογών που κάνουν χρήση ταυτόχρονων δομών δεδομένων, η οποία βασίζεται στην εξερεύνηση του χώρου σχεδιασμού (design space exploration).

3.1 Εισαγωγή

Τα πολυπύρηνα ενσωματωμένα συστήματα είναι σήμερα ευρέως διαθέσιμα στην αγορά και εκτελούν μεγάλο εύρος εφαρμογών, όπως επεξεργασία εικόνας/βίντεο, βάσεις δεδομένων κ.α.. Αυτές οι εφαρμογές συχνά βασίζονται σε ταυτόχρονες δομές δεδομένων, όπου πολλαπλά νήματα αποθηκεύουν και επεξεργάζονται τα δεδομένα τους. Η υλοποίηση αυτών των δομών δεδομένων είναι ιδιαίτερα σημαντική όσο αφορά την απόδοση της εφαρμογής και την Ποιότητα Παροχής Υπηρεσιών (QoS) του συστήματος.

Ο σχεδιασμός αποτελεσματικών ταυτόχρονων δομών δεδομένων είναι πολύπλοκη και απαιτητική διαδικασία για πολλούς λόγους. Καταρχάς, είναι δύσκολο για τους προγραμματιστές εφαρμογών να διαπιστώσουν δίχως δοκιμές και πειράματα ποια ταυτόχρονη δομή δεδομένων είναι κατάλληλη σε κάθε περίπτωση, ιδιαίτερα όταν χρησιμοποιούνται πολύπλοκοι αλγόριθμοι για τον συγχρονισμό των προσβάσεων στη δομή. Ο λόγος είναι ότι η επίδραση ενός αλγόριθμου συγχρονισμού στις μετρικές καθορίζεται από μεγάλο αριθμό παραμέτρων, όπως τον βαθμό συμφόρησης, τον αριθμό των νημάτων, τον χρόνο που απαιτείται για την απόκτηση / απελευθέρωση των κλει-

δωμάτων, τον βαθμό παραλληλίας που παρέχει μια δομή δεδομένων, κ.α.. Το γεγονός αυτό δημιουργεί την ανάγκη του ελέγχου και της αξιολόγησης από τους προγραμματιστές ενός μεγάλου αριθμού διαφορετικών υλοποιήσεων ταυτόχρονων δομών δεδομένων με την ίδια λειτουργικότητα, αλλά διαφορετική υλοποίηση, ώστε να βρεθεί αυτή που ικανοποιεί τους σχεδιαστικούς περιορισμούς.

Το παραπάνω πρόβλημα γίνεται ακόμη πιο περίπλοκο, αν ληφθεί υπόψη το θέμα της φορητότητας (portability). Οι ενσωματωμένες πλατφόρμες υποστηρίζουν διάφορα στοιχεία συγχρονισμού. Συχνά, τα ίδια στοιχεία είναι υλοποιημένα με διαφορετικό τρόπο σε χαμηλό επίπεδο, σε διαφορετικές πλατφόρμες. Μία ταυτόχρονη δομή δεδομένων υλοποιημένη σε μία πλατφόρμα με χρήση ενός συγκεκριμένου αλγορίθμου συγχρονισμού μπορεί να παρέχει πολύ διαφορετικά αποτελέσματα από άποψη απόδοσης ή κατανάλωσης ενέργειας αν υλοποιηθεί σε άλλη πλατφόρμα. Επομένως, η φορητότητα από ένα σύστημα σε ένα άλλο δεν μπορεί να είναι άμεση. Δημιουργείται η ανάγκη έρευνας των στοιχείων συγχρονισμού που παρέχει η συγκεκριμένη πλατφόρμα, τα οποία μπορεί να αξιοποιήσει η ταυτόχρονη δομή δεδομένων, ώστε να υλοποιηθεί με αποτελεσματικό τρόπο σε αυτήν.

Οι αλγόριθμοι συγχρονισμού που έχουν προταθεί για συστήματα γενικού σκοπού (general purpose systems), συχνά δεν μπορούν να υλοποιηθούν στα ενσωματωμένα συστήματα. Τα συστήματα γενικού σκοπού παρέχουν διάφορες επιλογές συγχρονισμού, συνήθως πολύ διαφορετικές σε σχέση με αυτές που παρέχουν τα ενσωματωμένα συστήματα [40][44]. Επιπλέον, αυτοί οι αλγόριθμοι δεν υλοποιήθηκαν αρχικά ώστε να είναι συμβατοί με τους σχεδιαστικούς περιορισμούς που υπάρχουν στα ενσωματωμένα συστήματα, όπως π.χ. η κατανάλωση ενέργειας.

Οι προγραμματιστές αντιμετωπίζουν συνήθως τα παραπάνω προβλήματα με την υιοθέτηση αυθαίρετων λύσεων, χωρίς να λαμβάνουν υπόψη τους στον βαθμό που απαιτείται τους περιορισμούς και τα χαρακτηριστικά του ενσωματωμένου συστήματος τα οποία επηρεάζουν την απόδοση και την κατανάλωση ενέργειας του αλγορίθμου συγχρονισμού. Επιπλέον, δεδομένου ότι η χρήση των μεθόδων συγχρονισμού βασισμένων στα κλειδώματα είναι ιδιαίτερα διαδεδομένες στα ενσωματωμένα συστήματα λόγω της απλότητά τους, υιοθετούνται σε μεγάλο βαθμό από τους προγραμματιστές. Παρόλα αυτά, οι παραπάνω προσεγγίσεις οδηγούν συνήθως σε μη βέλτιστα αποτελέσματα. Για παράδειγμα, κλειδώματα υλοποιημένα ως spinlocks, προκαλούν υψηλή κατανάλωση ενέργειας, η οποία είναι σημαντικός σχεδιαστικός περιορισμός στα ενσωματωμένα συστήματα.

3.1.1 Εξερεύνηση του χώρου λύσεων

Για την επιλογή κατάλληλης ταυτόχρονης δομής δεδομένων για μία εφαρμογή που εκτελείται σε ένα ενσωματωμένο σύστημα είναι απαραίτητη η εξερεύνηση του διαθέσιμου χώρου λύσεων που αποτελείται από τις σχεδιαστικές επιλογές. Σε αυτό το

κεφάλαιο παρουσιάζουμε μία συστηματική μεθοδολογία για τη βελτιστοποίηση των εφαρμογών που χρησιμοποιούν ταυτόχρονες δομές δεδομένων. Η μεθοδολογία είναι ημιαυτόματη και βασίζεται στην εξερεύνηση του χώρου λύσεων των ταυτόχρονων δομών δεδομένων. Υποστηρίζεται από μία ροή εργαλείων που αυτοματοποιεί πολλά βήματα της μεθοδολογίας και παρέχει δυνατότητες για αποτελεσματική εξερεύνηση.

Η ροή της μεθοδολογίας μπορεί να συνοψιστεί ως εξής: Ο προγραμματιστής, αφού παράσχει τους περιορισμούς της εφαρμογής και της αρχιτεκτονικής που απενεργοποιούν συγκεκριμένες σχεδιαστικές επιλογές, εισάγει το interface του εργαλείου στον πηγαίο κώδικα της εφαρμογής που θα βελτιστοποιηθεί. Στη συνέχεια, ακολουθεί η διαδικασία της εξερεύνησης, όπου δημιουργούνται υλοποιήσεις της ταυτόχρονης δομής δεδομένων με διαφορετικές σχεδιαστικές επιλογές η κάθε μία. Η εφαρμογή εκτελείται για κάθε μία από τις υλοποιήσεις. Τα αποτελέσματα διαφόρων μετρικών για κάθε υλοποίηση παρέχονται στον προγραμματιστή, ώστε να είναι σε θέση να επιλέξει την καταλληλότερη από αυτές, λαμβάνοντας υπόψη τους σχεδιαστικούς περιορισμούς. Με τον τρόπο αυτό, αποφεύγεται η υιοθέτηση αυθαίρετων λύσεων, καθώς και η χρονοβόρα διαδικασία της υλοποίησης και εξέτασης μεγάλου αριθμού διαφορετικών ταυτόχρονων δομών δεδομένων από τον προγραμματιστή.

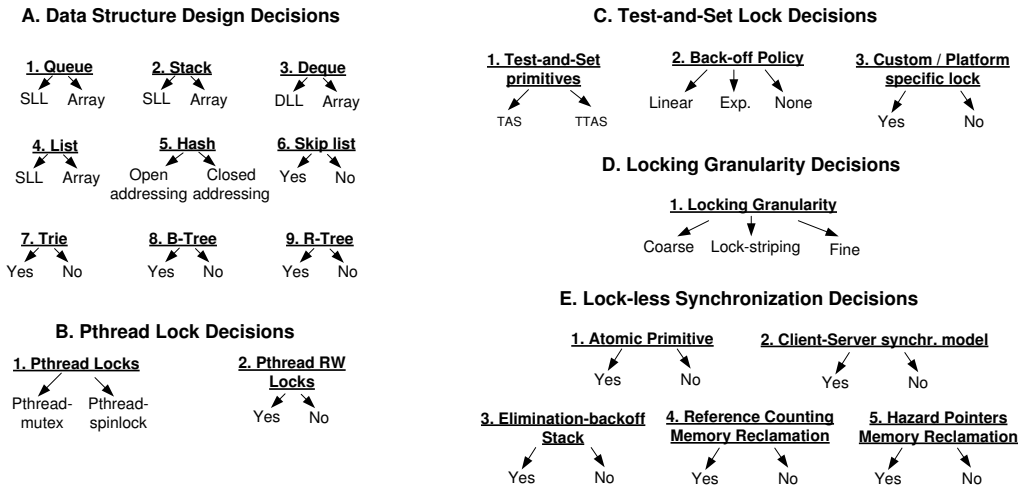
Οι πρωτοτυπίες της μεθοδολογίας μπορούν να συνοψιστούν ως εξής:

- Παρουσιάζουμε μία συστηματική ταξινόμηση των σχεδιαστικών επιλογών των ταυτόχρονων δομών δεδομένων που χρησιμοποιούνται για τη δημιουργία του χώρου λύσεων, τις εξαρτήσεις μεταξύ των σχεδιαστικών επιλογών, καθώς και τους περιορισμούς που τις επηρεάζουν.

- Προτείνουμε μία ημιαυτόματη μεθοδολογία, καθώς και μία ροή εργαλείων, για την αξιολόγηση διαφορετικών υλοποιήσεων ταυτόχρονων δομών δεδομένων, βασισμένη στην εξερεύνηση του χώρου λύσεων. Η μεθοδολογία παρέχει τον εντοπισμό των trade-offs μεταξύ διαφόρων μετρικών, για διαφορετικές υλοποιήσεις.

3.2 Αποφάσεις σχεδιασμού δυναμικών δομών δεδομένων

Ο προτεινόμενος χώρος σχεδιαστικών επιλογών για ταυτόχρονες δομές δεδομένων απεικονίζεται στο Σχήμα 3.1. Οι αποφάσεις που πρέπει να πάρει ο προγραμματιστής που σχεδιάζει μία ταυτόχρονη δυναμική δομή δεδομένων, μοντελοποιούνται σε ένα σετ δέντρων αποφάσεων (decision trees). Ο χώρος σχεδιαστικών επιλογών καλύπτει τις πιο συνηθισμένες σχεδιαστικές επιλογές που έχουν προταθεί στη βιβλιογραφία. Είναι ανεξάρτητος από την εφαρμογή ή από συγκεκριμένη πλατφόρμα, γιατί περιέχει επιλογές που αφορούν πλατφόρμες και αρχιτεκτονικές με διαφορετικά χαρακτηριστικά. Οι σχεδιαστικές επιλογές ομαδοποιούνται σε πέντε κατηγορίες. Κάθε κατηγορία αποτελείται από ένα ή περισσότερα δέντρα αποφάσεων:



Σχήμα 3.1: Χώρος σχεδιαστικών επιλογών ταυτόχρονων δομών δεδομένων.

Η κατηγορία **Data Structure Design Decisions** αναφέρεται στον τρόπο με τον οποίο είναι σχεδιασμένη η δομή δεδομένων, δίχως να λαμβάνει υπόψη του τον αλγόριθμο συγχρονισμού που ρυθμίζει τις ταυτόχρονες προσβάσεις στη δομή. Για παράδειγμα, μια Queue μπορεί να υλοποιηθεί ως απλά συνδεδεμένη λίστα (Single Linked List - SLL) ή ως πίνακας (array). Η Deque ως διπλά συνδεδεμένη λίστα (Doubly Linked List - DLL) ή πίνακας.

Η κατηγορία **Pthread Lock Decisions** ομαδοποιεί τα απλά και τα readers/writer Pthread κλειδώματα. Τα POSIX thread mutexes (Pthread-mutex) λειτουργούν ως κλειδώματα για την προστασία των κοινόχρηστων δεδομένων. Ένα μόνο νήμα μπορεί να είναι κάτοχος του mutex κάθε χρονική στιγμή. Η διαφορά των POSIX mutexes από τα spinlocks (Pthread-spinlock) είναι η εξής: Όταν ένα νήμα βρίσκει το lock κατειλημένο, στην περίπτωση του mutex το νήμα εισέρχεται σε κατάσταση αναμονής και επανέρχεται (runnable) μόλις το κλείδωμα ελευθερωθεί. Αντίθετα, στην περίπτωση του spinlock, το νήμα προσπαθεί διαρκώς να αποκτήσει το lock και είναι συνεχώς runnable. Στην περίπτωση του spinlock, αποφεύγεται το overhead της μετάβασης των νημάτων μεταξύ των δύο καταστάσεων και είναι αποδοτικά σε περίπτωση που ο χρόνος για τον οποίο τα νήματα κρατούν το κλείδωμα είναι σχετικά μικρός. Από την άλλη πλευρά, οδηγούν συνήθως σε υψηλή κατανάλωση ενέργειας, καθώς οι πυρήνες στους οποίους εκτελούνται τα νήματα είναι διαρκώς ενεργοί. Οι λεπτομέρειες υλοποίησης των POSIX κλειδωμάτων εξαρτώνται από την αρχιτεκτονική. Τα POSIX RW locks επιτρέπουν την πρόσβαση στα κοινόχρηστα δεδομένα είτε πολλών νημάτων που έχουν τον ρόλο των readers (δηλ. δεν μεταβάλλουν την δομή), είτε ενός μόνο νήματος που έχει τον ρόλο του writer (δηλαδή μεταβάλει τη δομή) κάθε χρονική στιγμή.

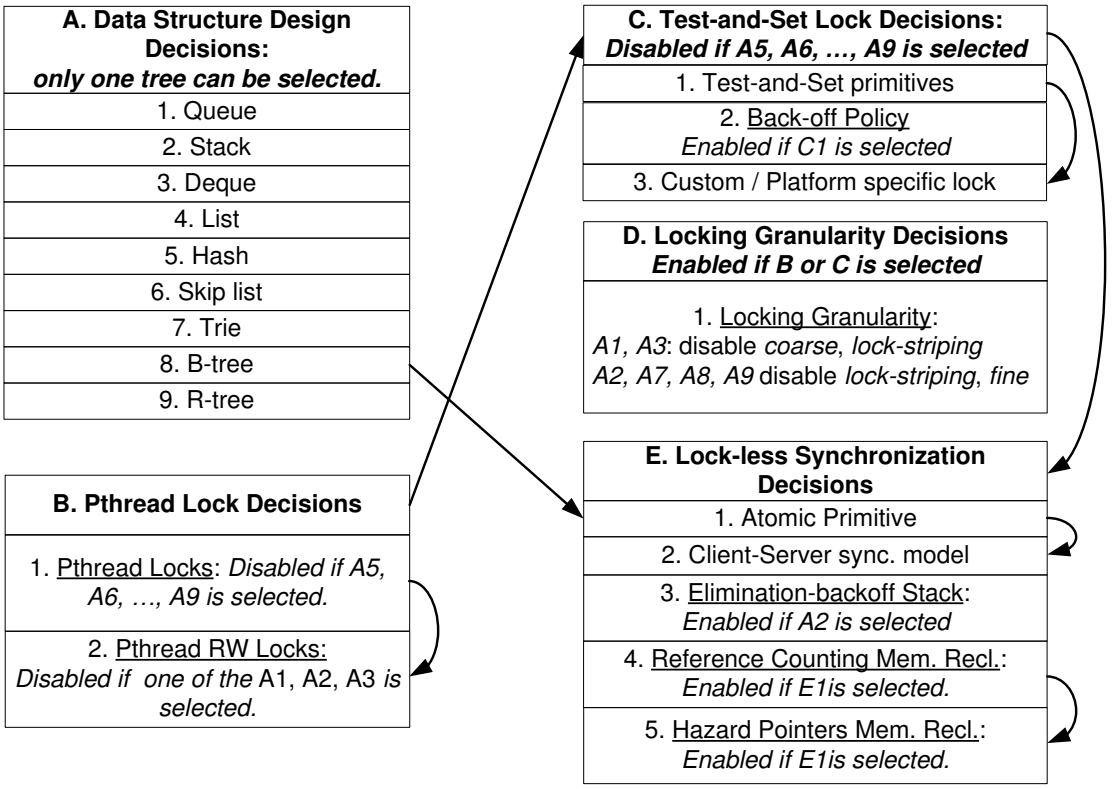
Η κατηγορία **Test-and-Set Lock Decisions** ομαδοποιεί τις επιλογές που αφορούν την παραμετροποίηση των Test-and-Set (TAS) και (Test-and-Test-and-Set) TTAS κλειδωμάτων. Οι συγκεκριμένες υλοποιήσεις έγιναν με χρήση των αντίστοιχων *gcc atomic*

builtins. Το κλείδωμα TAS λειτουργεί ουσιαστικά σαν spinlock, όπου κάθε πυρήνας προσπαθεί με ένα atomic operation να αποκτήσει το κλείδωμα. Σε περίπτωση που κάθε πυρήνας διαθέτει ιδιωτική cache, τότε κάθε φορά που ένας πυρήνας αποκτά ή ελευθερώνει το κλείδωμα, η τιμή του κλειδώματος καθίσταται άκυρη στις caches των πυρήνων, με συνέπεια να προκαλείται συμφόρηση στο bus και η απόδοση να μειώνεται. Η λύση σε αυτό είναι η χρήση TTAS κλειδώματος, όπου κάθε πυρήνας ελέγχει αρχικά την τιμή του κλειδώματος στην ιδιωτική του cache, πριν προσπαθήσει να το αποκτήσει με atomic operation στην κυρίως μνήμη. Η σχεδιαστική επιλογή *Back-off policy* καθορίζει το αν ένα νήμα θα αποχωρήσει για κάποιο χρονικό διάστημα από την διεκδίκηση ενός κλειδώματος για να μειώσει την συμφόρηση του διαύλου δεδομένων. Ο χρόνος αυτός μπορεί να αυξάνει γραμμικά ή εκθετικά.

Η κατηγορία **Locking Granularity Decisions** αναφέρεται στο αν ο καταμερισμός των κλειδωμάτων θα είναι coarse-grain ή fine-grain. Το *lock-striping* είναι μία τεχνική που χρησιμοποιείται σε hash-tables, στην οποία κάθε κλείδωμα προστατεύει έναν συγκεκριμένο αριθμό στοιχείων της δομής.

Η τελευταία κατηγορία είναι η **Lock-less Synchronization Decisions** που ομαδοποιεί σχεδιαστικές επιλογές για δομές που βασίζονται σε οποιοδήποτε αλγόριθμο συγχρονισμού δεν χρησιμοποιεί κλειδώματα. Το μοντέλο client-server (*Client-Server synchronization model*) αναφέρεται στη μέθοδο συγχρονισμού που προτείνεται στο [45] και αναλύεται εκτενέστερα στο επόμενο κεφάλαιο της παρούσας διατριβής: Ένας πυρήνας έχει τον ρόλο του server και είναι ο μοναδικός που έχει άμεση πρόσβαση στη δομή. Οι υπόλοιποι πυρήνες είναι οι clients και όταν θέλουν να αποκτήσουν πρόσβαση στη δομή στέλνουν αντίστοιχο μήνυμα στον server, ο οποίος εκτελεί ο ίδιος την λειτουργία αντί του client. Η επικοινωνία μεταξύ των clients και του server γίνεται αποκλειστικά με ανταλλαγή μηνυμάτων. Το interface της επικοινωνίας μεταξύ τους περιγράφεται στο [45]. Ο αλγόριθμος *Elimination-backoff Stack* αφορά στοίβες και έχει προταθεί στο [46]. Τέλος, οι επιλογές *memory reclamation* αφορούν αποφάσεις σχετικές με την διαχείριση μνήμης των ταυτόχρονων δομών δεδομένων δίχως κλειδώματα. Ο αλγόριθμος Hazard pointers περιγράφεται στο [47], ενώ ο Reference Counting στο [48]

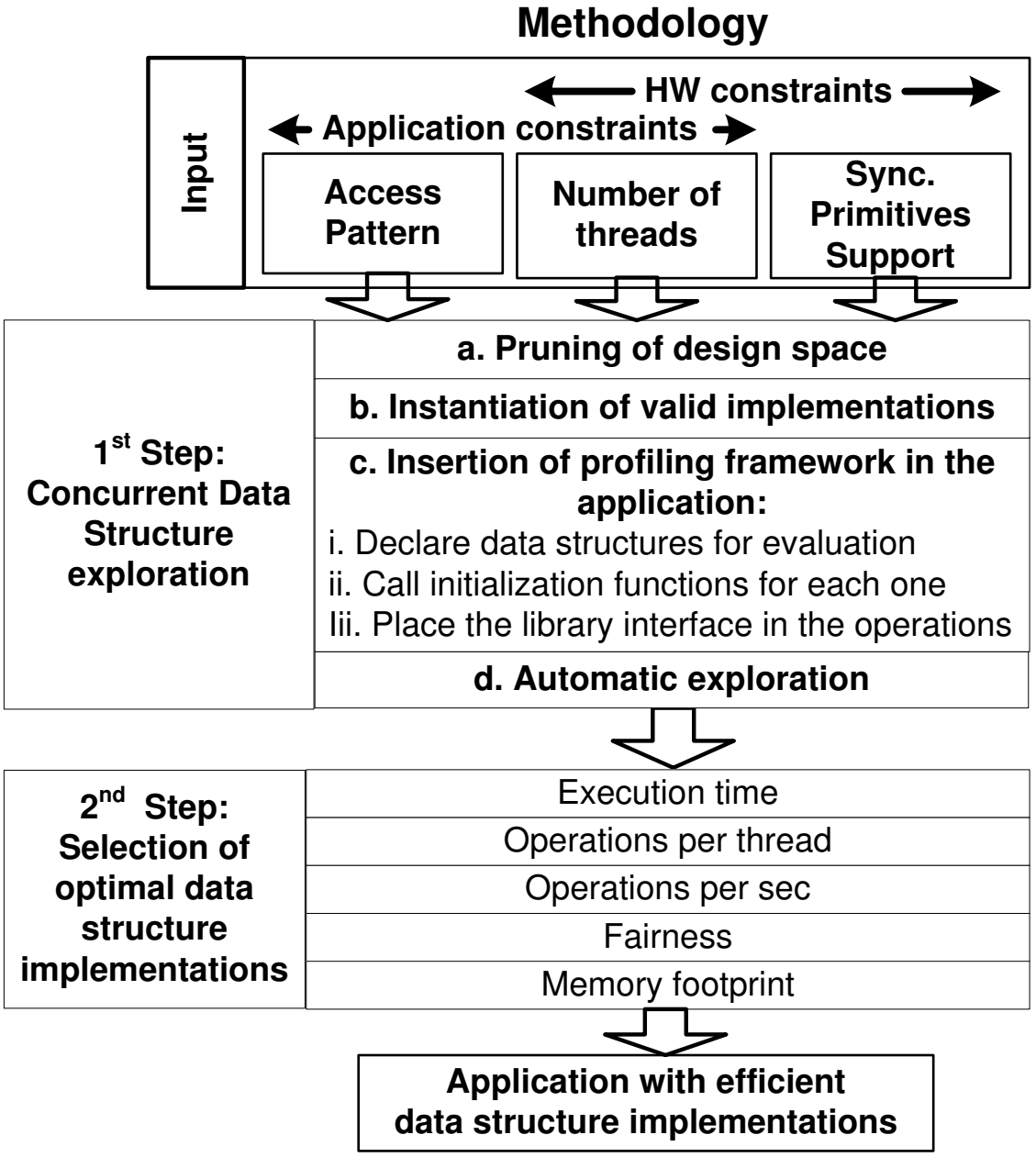
Οι σχεδιαστικές επιλογές που παρουσιάζονται στο Σχήμα 3.1 δεν είναι ανεξάρτητες μεταξύ τους, δεδομένου ότι συγκεκριμένες επιλογές μπορεί να έχουν επίδραση σε άλλες. Μία *αλληλεξάρτηση* (interdependency) υπάρχει όταν μια συγκεκριμένη σχεδιαστική επιλογή εμποδίζει τη χρήση ενός δέντρου αποφάσεων, φύλλου ή κατηγορίας. Οι αλληλεξαρτήσεις είναι ένα σύνολο κανόνων για το πώς μια συγκεκριμένη ταυτόχρονη δομή δεδομένων μπορεί να υλοποιηθεί μέσα από τον χώρο των σχεδιαστικών επιλογών. Οι κανόνες αυτοί είναι τμήμα της ροής εργαλείων που υποστηρίζει τη μεθοδολογία και χρησιμοποιούνται για την υλοποίηση των δομών δεδομένων που προορίζονται για αξιολόγηση. Απεικονίζονται στο Σχήμα 3.2. Η επιλογή της κατηγορίας ή του δέντρου απόφασης στην αρχή του βέλους καθιστά άκυρη (απενεργοποιεί) την



Σχήμα 3.2: Αλληλεξαρτήσεις των σχεδιαστικών επιλογών των ταυτόχρονων δομών δεδομένων.

σχεδιαστική επιλογή που βρίσκεται στο τέλος του βέλους. Κάποιες αλληλεξαρτήσεις απεικονίζονται με βέλη στο Σχήμα 3.2, ενώ άλλες αναγράφονται επάνω στο σχήμα. Οι αλληλεξαρτήσεις είναι τμήμα της ροής εργαλείων και ο προγραμματιστής δεν έχει άμεση πρόσβαση σε αυτές.

Ο χώρος σχεδιαστικών επιλογών σε συνδυασμό με της αλληλεξαρτήσεις είναι ένας αποτελεσματικός τρόπος για να απεικονιστούν οι σχεδιαστικές επιλογές των δομών δεδομένων: Με τον τρόπο αυτό διαχωρίζονται τα δέντρα αποφάσεων που αντιστοιχούν σε *χαρακτηριστικά* και τα φύλλα των δέντρων που αντιστοιχούν σε *τιμές* που μπορεί να έχει το αντίστοιχο χαρακτηριστικό. Αυτή η ιεραρχική απεικόνιση διευκολύνει την κατασκευή μιας ροής εργαλείων και απλοποιεί τη διαδικασία εξερεύνησης, ως εξής: Όλα τα φύλλα και οι σχεδιαστικές επιλογές που δεν είναι απενεργοποιημένα αξιολογούνται κατά τη διαδικασία εξερεύνησης. Με άλλα λόγια, οι υλοποιήσεις που τελικά θα αξιολογηθούν είναι αυτές που μπορούν να σχηματιστούν από τα φύλλα των δέντρων αποφάσεων που δεν έχουν απενεργοποιηθεί, βάση των κανόνων που ορίζονται από τις αλληλεξαρτήσεις. Επιπλέον, η απεικόνιση των επιλογών σχεδιασμού με τον συστηματικό τρόπο των δέντρων αποφάσεων διευκολύνει την επέκταση του χώρου σχεδιαστικών επιλογών, καθώς και την ανάπτυξη της ροής εργαλείων. Βασισμένοι στον ορισμό του χώρου σχεδιαστικών επιλογών, προτείνουμε το συστηματική



Σχήμα 3.3: Μεθοδολογία επιλογής κατάλληλων ταυτόχρονων δομών δεδομένων.

μεθοδολογία εξερεύνησης του χώρου λύσεων για ταυτόχρονες δομές δεδομένων.

3.3 Μεθοδολογία επιλογής κατάλληλων ταυτόχρονων δομών δεδομένων

Η μεθοδολογία επιλογής κατάλληλων ταυτόχρονων δυναμικών δεδομένων αποτελείται από δύο βήματα και παρουσιάζεται στο Σχήμα 3.3. Το πρώτο βήμα είναι η διαδικασία εξερεύνησης του χώρου λύσεων των ταυτόχρονων δομών δεδομένων

Πίνακας 3.1: Αντιστοίχιση σχημάτων πρόσβασης και δέντρων αποφάσεων του χώρου σχεδιαστικών επιλογών που ενεργοποιούνται για κάθε σχήμα

Access Pattern	Ενεργοποιημένα δέντρα αποφάσεων της κατηγορίας 'A'
FIFO	A1
LIFO	A2
Deque	A3
Simple storage	A4
key-value pairs storage	A5, A6, A8
key-value pairs sorted storage	A6, A8
String storage	A7
Spatial access	A9

(Concurrent Data Struture exploration), ενώ το δεύτερο είναι η αξιολόγηση και η επιλογή κατάλληλης υλοποίησης (Selection of optimal data structure implementations). Ο χώρος σχεδιαστικών επιλογών είναι ανεξάρτητος τόσο της εφαρμογής, όσο και της πλατφόρμας στην οποία η εφαρμογή εκτελείται. Επομένως, δεν είναι όλες οι σχεδιαστικές επιλογές κατάλληλες για κάθε εφαρμογή, ούτε υποστηρίζονται από κάθε πλατφόρμα. Πριν την διαδικασία εξερεύνησης, είναι απαραίτητο να απενεργοποιηθούν από το χώρο σχεδιαστικών επιλογών, οι επιλογές που δεν υποστηρίζονται ή δεν έχουν νόημα για τη δεδομένη εφαρμογή και τη δεδομένη πλατφόρμα στην οποία η εφαρμογή εκτελείται.

Εντοπίσαμε τους περιορισμούς της εφαρμογής και της πλατφόρμας οι οποίες ενεργοποιούν ή απενεργοποιούν συγκεκριμένες κατηγορίες ή δέντρα αποφάσεων του χώρου σχεδιαστικών επιλογών: Το *σχήμα πρόσβασης (access pattern)* είναι περιορισμός της εφαρμογής. Ο *αριθμός των νημάτων (number of threads)* είναι περιορισμός της εφαρμογής ή/και της πλατφόρμας, ενώ η *υποστήριξη στοιχείων συγχρονισμού (synchronization primitives support)* είναι περιορισμός της πλατφόρμας. Οι περιορισμοί αυτοί αποτελούν είσοδο της μεθοδολογίας που παρέχεται από τον χρήστη και οδηγούν στην απενεργοποίηση συγκεκριμένων κατηγοριών ή δέντρων αποφάσεων. Δεν είναι σε θέση να απενεργοποιήσουν συγκεκριμένα φύλλα των δέντρων αποφάσεων. Αυτό είναι σημαντικό, ώστε να διατηρείται η συνοχή της μεθοδολογίας και των εργαλείων που την υποστηρίζουν.

Το **σχήμα πρόσβασης** ορίζεται ως η ακολουθία με την οποία γίνεται η πρόσβαση στα δεδομένα από τον αλγόριθμο της εφαρμογής [9]. Βρίσκεται σε επίπεδο αφαίρεσης υψηλότερο από την υλοποίηση της δομής δεδομένων. Επηρεάζει τις σχεδιαστικές αποφάσεις της κατηγορίας A του χώρου σχεδιαστικών επιλογών. Σχήματα πρόσβασης και τα δέντρα αποφάσεων της κατηγορίας A που ενεργοποιούνται για το καθένα παρουσιάζονται στον Πίνακα 3.1.

Ο **αριθμός των νημάτων** σχετίζεται με το κατά πόσο μία δομή δεδομένων είναι

Πίνακας 3.2: Αντιστοίχιση αριθμού νημάτων και κατηγοριών του χώρου σχεδιαστικών επιλογών που απενεργοποιούνται

Number of threads	Απενεργοποιημένες κατηγορίες
One	B, C, D, E
Many	-

Πίνακας 3.3: Αντιστοίχιση υποστήριξης στοιχείων συγχρονισμού και δέντρων αποφάσεων του χώρου σχεδιαστικών επιλογών που ενεργοποιούνται για κάθε στοιχείο

Synchronization primitives support	Ενεργοποιημένα δέντρα αποφάσεων
Pthreads	B, D
Test-and-Set	C1, C2, D
Custom/Platf. Spec. locks	C3, D
Atomic Primitives with CAS	E1, E5
Atomic Primitives with DCAS	E1, E4
Message passing communication	E2

ταυτόχρονη και μπορεί να έχει δύο τιμές: *one* και *many*. Καθορίζει το αν τα στοιχεία της δομής δεδομένων μπορούν να τροποποιούνται από περισσότερα από ένα νήματα ταυτόχρονα. Αν η τιμή είναι *ένα*, τότε οι κατηγορίες *B*, *C*, *D* και *E* απενεργοποιούνται, όπως φαίνεται στον Πίνακα 3.2. Στην περίπτωση αυτή, η δομή δεδομένων είναι ακολουθιακή και η εξερεύνηση περιορίζεται στα δέντρα αποφάσεων της κατηγορίας *A*. Η μεθοδολογία για αριθμό νημάτων *ένα*, περιορισμένη στα τέσσερα πρώτα δέντρα της κατηγορίας *A*, παρουσιάζεται στην εργασία [9].

Ο τελευταίος περιορισμός είναι η *υποστήριξη στοιχείων συγχρονισμού*, που προέρχεται από το υλικό και απεικονίζεται στον Πίνακα 3.3. Τα χαρακτηριστικά της πλατφόρμας στα οποία η εφαρμογή εκτελείται ενεργοποιούν τα δέντρα αποφάσεων των κατηγοριών *B*, *C*, *D* και *E*.

Είναι σημαντικό να τονιστεί η διαφοροποίηση που υπάρχει ανάμεσα στην απενεργοποίηση των δέντρων αποφάσεων και των κατηγοριών που συμβαίνει εξαιτίας των περιορισμών της εφαρμογής ή της πλατφόρμας και στις αλληλεξαρτήσεις. Ο ρόλος των περιορισμών είναι το να απενεργοποιήσουν δέντρα αποφάσεων και κατηγορίες που δεν υποστηρίζονται λόγω των χαρακτηριστικών και της πλατφόρμας και της εφαρμογής. Με αυτόν τον τρόπο, ο χώρος των λύσεων γίνεται από ανεξάρτητος της πλατφόρμας και της εφαρμογής, προσαρμοσμένος σε αυτές. Αντίθετα, οι αλληλεξαρτήσεις αφορούν τον τρόπο με τον οποίο σχηματίζεται κάθε συγκεκριμένη υλοποίηση δομής δεδομένων από το χώρο σχεδιαστικών επιλογών. Με άλλα λόγια, αποτελούν τον σύνολο των κανόνων της μετάβασης από τον χώρο των σχεδιαστικών επιλογών στο χώρο των λύσεων. Δείχνει τον τρόπο με τον οποίο τα δέντρα αποφάσεων που δεν απενεργοποιήθηκαν εξαιτίας των περιορισμών θα χρησιμοποιηθούν ώστε να πα-

ραχθούν οι υλοποιήσεις δομών δεδομένων που θα αξιολογηθούν κατά την εξερεύνηση του χώρου λύσεων.

Πριν την εκτέλεση του πρώτου βήματος της μεθοδολογίας, ο προγραμματιστής εισάγει τους περιορισμούς της εφαρμογής και της πλατφόρμας, ώστε να μην αξιολογηθούν υλοποιήσεις που δεν υποστηρίζονται. Ο χώρος με τις σχεδιαστικές επιλογές που δεν έχουν απενεργοποιηθεί παρέχεται σε ένα script το οποίο χειρίζεται την υλοποίηση των δομών δεδομένων που θα αξιολογηθούν. Μέσω των κανόνων αλληλεξαρτήσεων, σχηματίζονται όλες οι δυνατές υλοποιήσεις δυναμικών δομών δεδομένων, από τις σχεδιαστικές αποφάσεις που δεν έχουν απενεργοποιηθεί λόγω των περιορισμών.

Μετά τη δημιουργία των υλοποιήσεων, ο προγραμματιστής εισάγει το interface της βιβλιοθήκης των ταυτόχρονων δομών δεδομένων στην εφαρμογή χειροκίνητα, αντικαθιστώντας τις δομές δεδομένων της εφαρμογής με αυτές της βιβλιοθήκης. Έτσι, όλες οι λειτουργίες (π.χ. insert, remove, get) πραγματοποιούνται μέσω των δομών δεδομένων της βιβλιοθήκης αντί για αυτές της εφαρμογής. Δεδομένου ότι όλες οι υλοποιήσεις δομών δεδομένων με το ίδιο σχήμα πρόσβασης έχουν το ίδιο interface (Πίνακας 3.1), η διαδικασία αυτή γίνεται μόνο μία φορά.

Στη συνέχεια, ακολουθεί η διαδικασία της εξερεύνησης: Η εφαρμογή εκτελείται για κάθε υλοποίηση δομής δεδομένων που δημιουργήθηκε νωρίτερα. Ο προγραμματιστής θα πρέπει να παράσχει workload στην εφαρμογή που να αντιστοιχεί σε πραγματικές συνθήκες. Για εφαρμογές όπου το workload μπορεί να μεταβάλλεται κατά τη διάρκεια του χρόνου εκτέλεσης, ο προγραμματιστής μπορεί να πραγματοποιήσει τη διαδικασία της εξερεύνησης για έναν αριθμό από αντιπροσωπευτικά workloads. Σε αυτή την περίπτωση θα πρέπει να επαναλάβει τα βήματα 1.c και 2 της μεθοδολογίας, μία φορά για κάθε workload.

Ένα profiling tool είναι ενσωματωμένο στη βιβλιοθήκη και συλλέγει πληροφορίες σχετικά με τον χρόνο εκτέλεσης, τον αριθμό των λειτουργιών ανά νήμα, το throughput, το fainess και το αποτύπωμα μνήμης για κάθε εκτέλεση. Αποτελέσματα άλλων μετρικών, όπως η κατανάλωση ενέργειας συλλέγονται χειροκίνητα. Για να βελτιωθεί η αξιοπιστία των αποτελεσμάτων, ο προγραμματιστής μπορεί να ρυθμίσει χειροκίνητα τον αριθμό επαναλήψεων για κάθε εκτέλεση. Σε περίπτωση που κάθε εκτέλεση πραγματοποιηθεί περισσότερες από μία φορές, το τελικό αποτέλεσμα είναι ο μέσος όρος των τιμών που συλλέχθηκαν. Το μέτωπο Pareto δύο ή περισσότερων optimization objectives, παρέχεται αυτόματα. Για αποτελέσματα που έχουν συλλεχθεί χειροκίνητα, το Pareto front παρέχεται με μικρές τροποποιήσεις του αντίστοιχου script.

Στο δεύτερο βήμα της μεθοδολογίας, τα αποτελέσματα του profiling που συλλέχθηκαν κατά το προηγούμενο βήμα για κάθε υλοποίηση παρέχονται στον προγραμματιστή. Λαμβάνοντας υπόψη τους σχεδιαστικούς περιορισμούς (π.χ. μέγεθος μνήμης, latency/throughput), ο προγραμματιστής επιλέγει την καταλληλότερη δομή δεδομένων για την εφαρμογή που βελτιστοποιείται.

Αν η εφαρμογή περιέχει περισσότερες από μία δομές δεδομένων, απαιτούνται για

Πίνακας 3.4: Αυτόματα και μη-αυτόματα βήματα της μεθοδολογίας

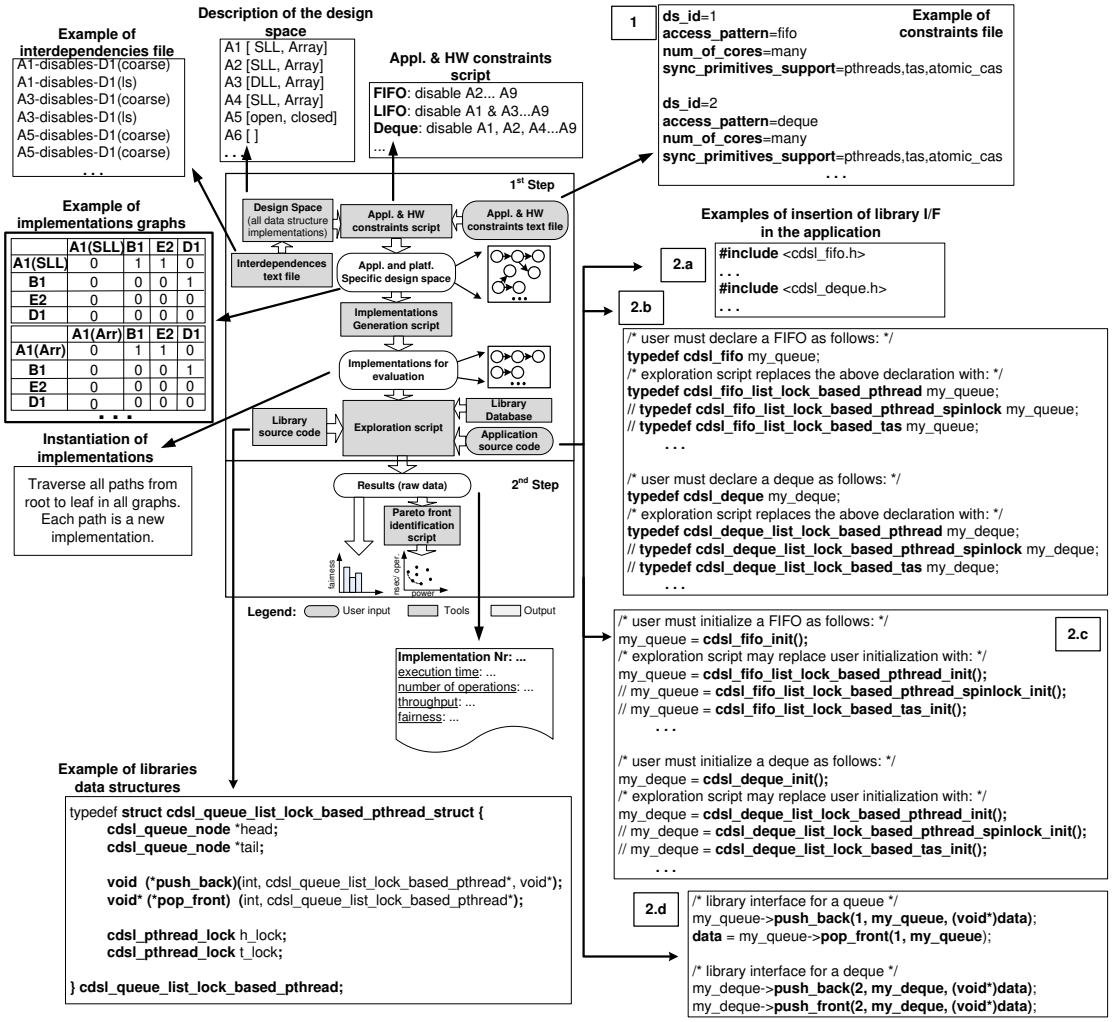
Μη-αυτόματα βήματα	Αυτόματα βήματα
Είσοδος: Δήλωση των περιορισμών	**1a**: Περιορισμός του χώρου σχεδ. επιλογών
1c: εισαγωγή i/f βιβλιοθήκης. a) δήλωση των δομών δεδομένων b) αρχικοποίηση τους c) i/f στις κλήσεις των λειτουργιών	**1b**: δημιουργία έγκυρων υλοποιήσεων
μετρικές 2ου βήματος: κατανάλωση ενέργειας μετρήσεις για κάθε εκτέλεση	**1d**: Ρύθμιση των συναρτήσεων αρχικοποίησης πριν την εκτέλεση. Εκτέλεση έγκυρων υλοποιήσεων **μετρικές 2ου βήματος**: χρόνος εκτέλεσης oper./sec, oper./thread fairness, memory footprint εντοπισμός του Pareto front

κάθε μία το *σχήμα πρόσβασης* και ο *αριθμός των νημάτων*. Στη συνέχεια, το interface της βιβλιοθήκης θα εισαχθεί σε κάθε μία και όλοι οι δυνατοί συνδυασμοί θα αξιολογηθούν με εξερεύνηση brute-force. Στο δεύτερο βήμα της μεθοδολογίας, ο προγραμματιστής επιλέγει τον πιο αποδοτικό συνδυασμό υλοποιήσεων δομών δεδομένων. Τέλος, στον Πίνακα 3.4 απεικονίζονται τα βήματα της μεθοδολογίας που πραγματοποιούνται αυτόματα από την ροή των εργαλείων, καθώς και αυτά που πραγματοποιούνται χειροκίνητα από τον προγραμματιστή.

Ο χρόνος που απαιτείται για να εφαρμοστεί η μεθοδολογία εξαρτάται από δύο βασικές παραμέτρους: Τον αριθμό των δομών δεδομένων για τις οποίες θα εξεταστούν διαφορετικές υλοποιήσεις, καθώς και το μέγεθος του χώρου σχεδιαστικών επιλογών για κάθε μία. Ένας σχετικά μεγάλος αριθμός επιλογών θα έχει ως συνέπεια την εξέταση μεγάλου αριθμού υλοποιήσεων. Παρόλα αυτά, ο περιορισμός του χώρου σχεδιαστικών επιλογών που πραγματοποιείται κατά το πρώτο βήμα της μεθοδολογίας μειώνει δραστικά τον αριθμό των υλοποιήσεων που τελικά εξετάζονται.

3.4 Εργαλεία Επιλογής κατάλληλων ταυτόχρονων δυναμικών δομών δεδομένων

Σε αυτή την ενότητα παρουσιάζεται η ροή των εργαλείων που χρησιμοποιείται στην εφαρμογή των βημάτων της μεθοδολογίας. Απεικονίζεται στο Σχήμα 3.4. Η περιγραφή του χώρου σχεδιαστικών αποφάσεων και τα interdependencies βρίσκονται σε αρχεία κειμένου, στη μορφή που φαίνονται στο Σχήμα. Ο χρήστης της μεθοδολογίας

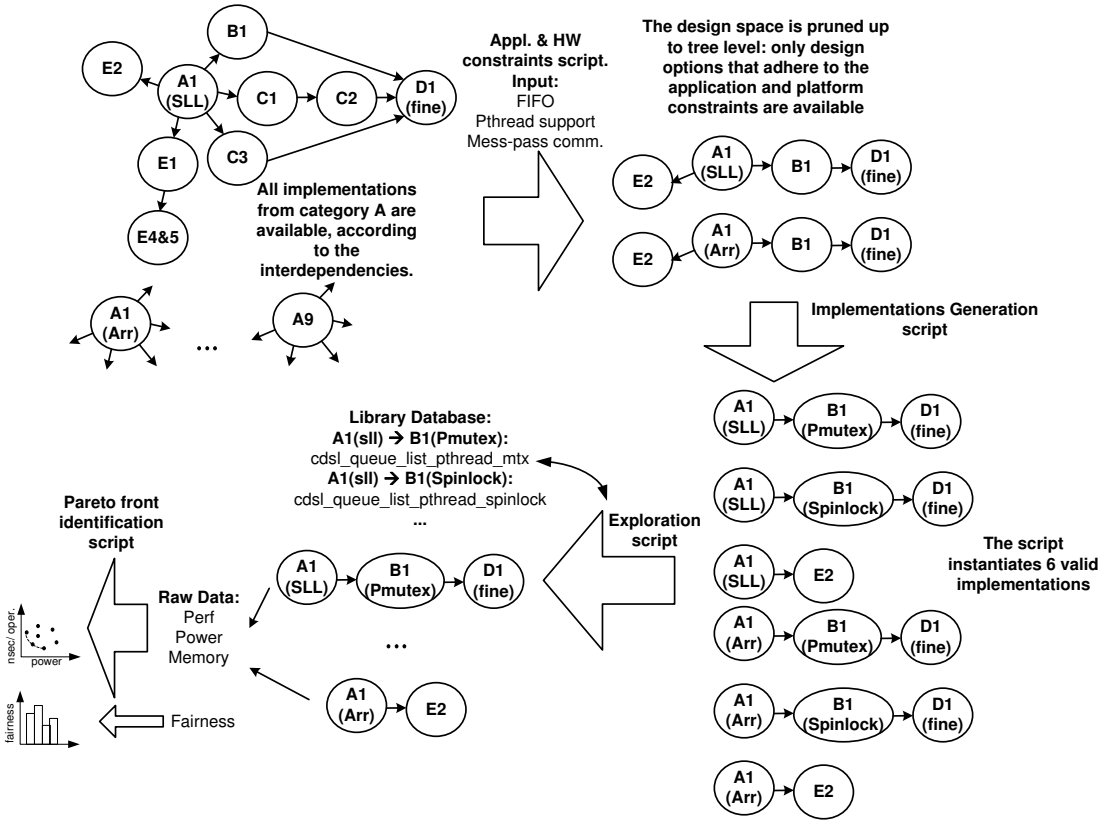


Σχήμα 3.4: Ροή εργαλείων της μεθοδολογίας.

θα πρέπει να ορίσει τους περιορισμούς σε ένα αρχείο κειμένου, όπως φαίνεται στο Σχήμα 3.4. Το *application and hardware constraints script* παίρνει ως είσοδο την περιγραφή του design space, τα interdependencies και τους περιορισμούς και παράγει ως έξοδο το pruned design space, σε μορφή κατευθυνόμενων άκυκλων γράφων (direct acyclic graphs), τα οποία απεικονίζονται σε μορφή adjacency matrices.

Η περιγραφή του pruned design space παρέχεται ως είσοδος στο *implementations generation script*, το οποίο δημιουργεί τις υλοποιήσεις που πρόκειται να αξιολογηθούν, διασχίζοντας όλες τις διαδρομές στους γράφους του pruned design space από τη ρίζα ως τα φύλλα κάθε γράφου. Κάθε διαδρομή είναι μία διαφορετική υλοποίηση, η οποία θα εξεταστεί.

Ο ρόλος του *exploration script* είναι το να αντιστοιχίσει τους κόμβους των γράφων στον πηγαίο κώδικα της βιβλιοθήκης, χρησιμοποιώντας τη *Library Database*, η οποία είναι ουσιαστικά ένας hash table στον οποίο αντιστοιχίζονται οι κόμβοι με τον πηγαίο



Σχήμα 3.5: Παράδειγμα δημιουργίας των υλοποιήσεων προς αξιολόγηση από τα εργαλεία της μεθοδολογίας.

κώδικα της βιβλιοθήκης και τις συναρτήσεις της. Το script εξετάζει τον πηγαίο κώδικα της εφαρμογής και πραγματοποιεί τις απαραίτητες αλλαγές στο interface της βιβλιοθήκης, πριν από κάθε νέα εκτέλεση. Τέλος, συλλέγει τα αποτελέσματα για τον χρόνο εκτέλεσης, τον αριθμό των λειτουργιών ανά νήμα και ανά δευτερόλεπτο, το fairness, και το μέγεθος μνήμης για κάθε δομή δεδομένων. Ένα παράδειγμα δημιουργίας των υλοποιήσεων προς αξιολόγηση από τα εργαλεία της μεθοδολογίας απεικονίζεται στο Σχήμα 3.5.

Τα αποτελέσματα της εκτέλεσης κάθε υλοποίησης παρέχονται σε μορφή αρχείου κειμένου, όπως φαίνονται στο Σχήμα 3.4. Τα αρχεία αυτά παρέχονται στο *Pareto front identification script*, το οποίο εντοπίζει τις βέλτιστες κατά Pareto υλοποιήσεις. Στη συγκεκριμένη εργασία, μια υλοποίηση θεωρείται βέλτιστη κατά Pareto σε ένα συγκεκριμένο πείραμα για 2 συγκεκριμένες μετρικές, όταν δεν υπάρχει άλλη υλοποίηση που να είναι καλύτερη ως προς και τις δύο μετρικές. Ο αλγόριθμος είναι υλοποιημένος σύμφωνα με την περιγραφή στην εργασία [49].

Οι λειτουργίες που πρέπει να πραγματοποιήσει ο χρήστης ώστε να εφαρμόσει τη μεθοδολογία είναι οι εξής:

1. Αναγράφονται τα constraints στο αντίστοιχο αρχείο, όπως φαίνονται στο Σχήμα

3.4.

2. Τροποποιείται ο κώδικας της εφαρμογής συμπεριλαμβάνοντας απαραίτητα αρχεία της βιβλιοθήκης (2.a στο Σχήμα 3.4).

3. Δηλώνονται (2.b) και αρχικοποιούνται οι δομές δεδομένων (2.c).

4. Αντικαθίστανται τα operations της εφαρμογής με αυτά της βιβλιοθήκης (2.d).

Όπως φαίνεται στους κώδικες 2.b και 2.c του Σχήματος 3.4, οι δηλώσεις και οι συναρτήσεις των δομών δεδομένων τροποποιούνται αυτόματα από το *exploration script*, ώστε να δημιουργηθούν διαφορετικές υλοποιήσεις κατά τη διαδικασία εξερεύνησης. Έτσι, δεν απαιτούνται περισσότερες ενέργειες από την πλευρά του προγραμματιστή, πέρα από τα 4 παραπάνω βήματα.

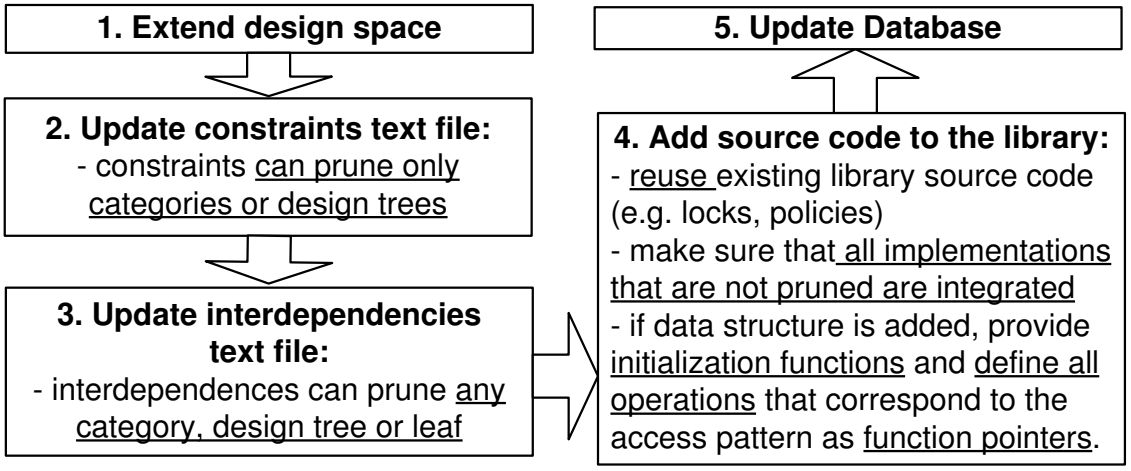
Η βιβλιοθήκη είναι υλοποιημένη σε C. Αναπτύχθηκε με στόχο να παράσχει λειτουργίες και interface αντίστοιχο με αυτές που παρέχει η C++ Standard Template Library (STL). Οι διαθέσιμες λειτουργίες για τα σχήματα πρόσβασης απεικονίζονται στον Πίνακα 3.5. Το interface των λειτουργιών για κάθε υλοποίηση δομής δεδομένων βασίζεται σε function pointers και είναι ακριβώς η ίδια για κάθε υλοποίηση που ανήκει στο ίδιο σχήμα πρόσβασης. Για παράδειγμα, όλες οι υλοποιήσεις FIFO που μπορούν να δημιουργηθούν από τον χώρο σχεδιαστικών επιλογών έχουν ακριβώς το ίδιο interface όσο αφορά τις λειτουργίες τους. Επομένως, δεν απαιτείται καμία αλλαγή στο interface των συναρτήσεων των λειτουργιών της FIFO κατά τη διάρκεια της εξερεύνησης του χώρου λύσεων, ούτε από τον προγραμματιστή, ούτε από κάποιο script. Το μόνο που απαιτείται είναι το exploration script να τροποποιεί τις συναρτήσεις αρχικοποίησης (initialization functions), πριν από κάθε εκτέλεση. Έτσι, παρότι το interface παραμένει το ίδιο, η υλοποίηση μεταβάλλεται. Όλες οι δομές δεδομένων του Πίνακα 3.1 που έχουν το ίδιο σχήμα πρόσβασης, μπορούν να αντικαταστήσουν η μία την άλλη.

Η διαχείριση μνήμης των δομών δεδομένων στη βιβλιοθήκη γίνεται με δυναμικό τρόπο. Οι λίστες είναι υλοποιημένες με απλή συνδεσιμότητα (single linked lists) για τη σχεδιαστική επιλογή *queue* και διπλή συνδεσιμότητας (doubly linked list) για τις επιλογές *deque*, *Stack* και *List*. Ο πίνακας (array) είναι υλοποιημένος βάση του STL vector: όταν δεν μπορούν να αποθηκευτούν περισσότερα στοιχεία η χωρητικότητά του διπλασιάζεται αυτόματα.

Προκειμένου να αυξηθεί η φορητότητα της βιβλιοθήκης, τα *atomic primitives* είναι υλοποιημένα με χρήση των gcc intrinsics. Το μοντέλο client-server υλοποιήθηκε ως ένα υψηλού επιπέδου interface που κάνει χρήση των message buffers της πλατφόρμας. Σε περίπτωση που χρησιμοποιείται κάποια άλλη μέθοδος επικοινωνίας μεταξύ των clients και του server, θα πρέπει να εισαχθεί χειροκίνητα από τον προγραμματιστή στη βιβλιοθήκη.

Πίνακας 3.5: Υποστηριζόμενες λειτουργίες των δομών δεδομένων της βιβλιοθήκης

FIFO	LIFO	Deque	Simple storage		Key-value pairs storage		Key-val. pairs sorted String Storage Spatial Access	
empty	empty	empty	empty	insert	empty	iter_begin	empty	get_first
size	size	size	size	remove	size	iter_end	size	get_last
front	back	push_back	push_back	iter_begin	insert	iter_next	insert	iter_begin
back	push_back	pop_back	pop_back	iter_end	find	iter_deref	find	iter_end
push_back	pop_back	push_front	push_front	iter_next	remove		remove	iter_next
pop_front		pop_front	pop_front	iter_deref	count		count	iter_deref



Σχήμα 3.6: Βήματα και προϋποθέσεις για την επέκταση της μεθοδολογίας.

Παρόλο που τα εργαλεία που πλαισιώνουν τη μεθοδολογία παρέχουν αποτελέσματα για το memory footprint, τα αποτελέσματα αυτά δεν έχουν νόημα σε ταυτόχρονες δομές δεδομένων, καθότι το μέγεθος της μνήμης που εξαρτάται από μη ντετερμινιστικούς παράγοντες, όπως η σειρά με την οποία τα νήματα αποκτούν ένα κλείδωμα. Παρόλα αυτά, τα συγκεκριμένα αποτελέσματα μπορούν να χρησιμοποιηθούν για τον προσδιορισμό της τάξης μεγέθους της απαιτούμενης μνήμης από τις δομές δεδομένων.

3.5 Επεκτασιμότητα της Μεθοδολογίας

Ο χώρος σχεδιαστικών αποφάσεων καλύπτει τις πιο συνηθισμένες δομές δεδομένων που συναντώνται στη βιβλιογραφία. Παρόλα αυτά, δεδομένου ότι δεν καλύπτει κάθε περίπτωση, παρουσιάζουμε τα βήματα που θα πρέπει να ακολουθηθούν από τον προγραμματιστή ώστε να τον επεκτείνει με περισσότερες σχεδιαστικές επιλογές (Σχήμα 3.6).

Το πρώτο βήμα είναι η επέκταση του χώρου σχεδιαστικών επιλογών με την προσθήκη νέων επιλογών. Μπορεί να αφορά είτε την προσθήκη νέων φύλλων σε υπάρχον δέντρο αποφάσεων, είτε την προσθήκη νέων δέντρων αποφάσεων ή νέας κατηγορίας. Το δεύτερο βήμα είναι η επέκταση των περιορισμών. Ο χρήστης θα πρέπει να προσδιορίσει αν η νέα σχεδιαστική επιλογή απενεργοποιείται για κάποιο συγκεκριμένο περιορισμό που προέρχεται είτε από την εφαρμογή είτε από την πλατφόρμα και να ανανεώσει το αντίστοιχο αρχείο κειμένου. Είναι σημαντικό να τονιστεί ότι οι περιορισμοί απενεργοποιούν αποκλειστικά δέντρα αποφάσεων ή κατηγορίες και όχι φύλλα των δέντρων αποφάσεων. Αυτό εξασφαλίζει την ομαλή λειτουργία της ροής εργαλείων.

Το τρίτο βήμα είναι ο προσδιορισμός των αλληλεξαρτήσεων μεταξύ των υπαρχό-

ντων και των νέων σχεδιαστικών επιλογών και η ανανέωση του αντίστοιχου αρχείου κειμένου. Στη συνέχεια, ο πηγαίος κώδικας για την νέα σχεδιαστική επιλογή προστίθεται στη βιβλιοθήκη. Θα πρέπει να περιλαμβάνει όλες τις υλοποιήσεις δομών δεδομένων που δεν απενεργοποιούνται από τις αλληλεξαρτήσεις και τους περιορισμούς. Αν η επέκταση αφορά μια νέα δομή δεδομένων οι λειτουργίες που παρέχονται θα πρέπει να είναι οι ίδιες με τις δομές δεδομένων που έχουν το ίδιο σχήμα πρόσβασης και να είναι ορισμένες ως function pointers. Επιπλέον, θα πρέπει να παρέχεται μια συνάρτηση που θα αρχικοποιεί την δομή.

Είναι σημαντικό ο προγραμματιστής να επαναχρησιμοποιεί τμήματα της βιβλιοθήκης. Για παράδειγμα, αν η βιβλιοθήκη επεκτείνεται με μία νέα δομή δεδομένων, ο πηγαίος κώδικας της υλοποίησης των κλειδωμάτων που ήδη υπάρχει στη βιβλιοθήκη μπορεί να χρησιμοποιηθεί από την νέα δομή δεδομένων απευθείας. Παρόλα αυτά, αν η νέα δομή χρησιμοποιεί νέο είδος κλειδωμάτων, τα νέα κλειδώματα θα πρέπει να εισαχθούν ως νέες σχεδιαστικές επιλογές στη βιβλιοθήκη, ακολουθώντας από την αρχή τα βήματα του Σχήματος 3.6. Τέλος, ο προγραμματιστής ανανεώνει τη βάση δεδομένων της βιβλιοθήκης, αντιστοιχίζοντας τα κατάλληλα αρχεία πηγαίου κώδικα, τις συναρτήσεις και τις δηλώσεις με τις αντίστοιχες σχεδιαστικές επιλογές.

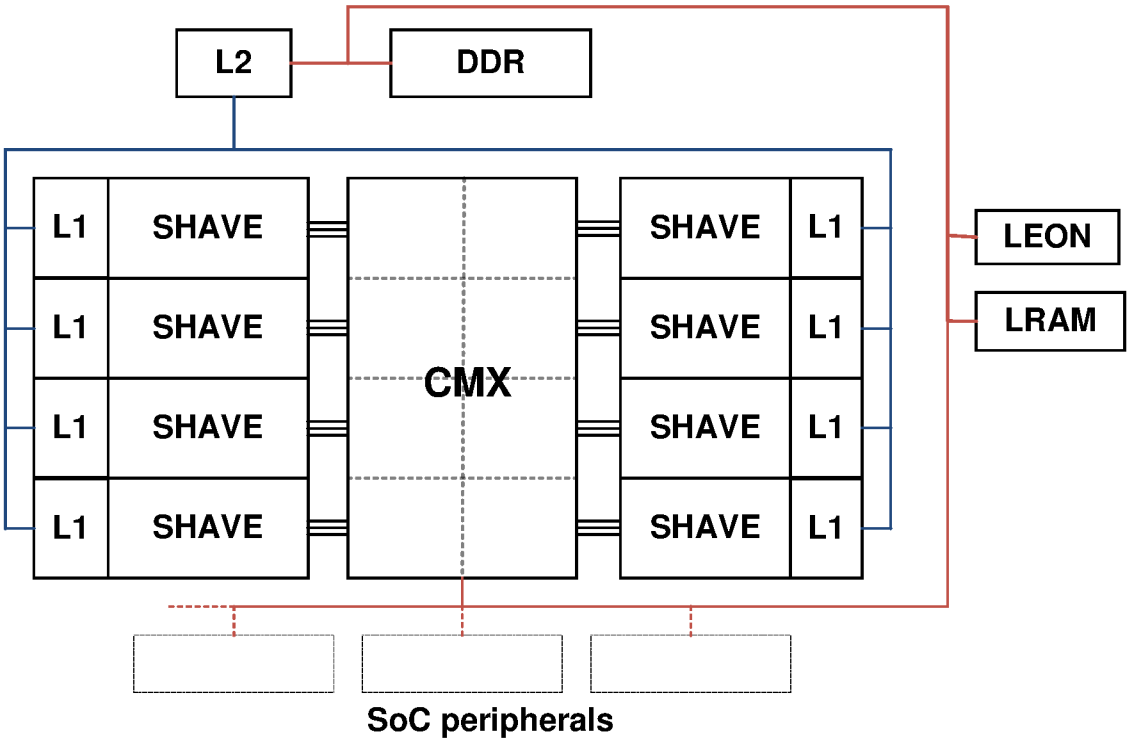
3.6 Εφαρμογή της μεθοδολογίας και πειραματικά αποτελέσματα

Στην ενότητα αυτή παρουσιάζονται πειραματικά αποτελέσματα που προέκυψαν από την εφαρμογή της μεθοδολογίας σε 5 ρεαλιστικά benchmarks που χρησιμοποιούν ταυτόχρονες δομές δεδομένων και εκτελούνται σε δύο αντιπροσωπευτικές ενσωματωμένες πλατφόρμες με διαφορετικά αρχιτεκτονικά χαρακτηριστικά. Ο στόχος της εφαρμογής της μεθοδολογίας είναι να εντοπιστούν trade-offs μεταξύ διαφόρων μετρικών, εξετάζοντας διαφορετικές υλοποιήσεις δομών δεδομένων. Τα αποτελέσματα έδειξαν πως επιλέγοντας διαφορετικές υλοποιήσεις πραγματοποιούνται trade-offs που μπορούν να οδηγήσουν σε βελτιστοποίηση των εφαρμογών που βασίζονται σε αυτές.

3.6.1 Περιγραφή των αρχιτεκτονικών

Οι πλατφόρμες που επιλέχθηκαν για την εφαρμογή της μεθοδολογίας είναι η Freescale I.MX.6 Quad και η Movidius Myriad. Είναι αντιπροσωπευτικές δύο κατηγοριών σύγχρονων πολυπύρηνων ενσωματωμένων αρχιτεκτονικών.

Η Freescale I.MX. 6 Quad ανήκει σε μία οικογένεια πολυπύρηνων single-board υπολογιστών βασισμένες σε ARM επεξεργαστές [2]. Περιέχει τέσσερις πυρήνες ARM Cortex A9 που λειτουργούν σε συχνότητα 1GHz και περιέχει δύο επίπεδα cache μνήμης. Όσο αφορά τα χαρακτηριστικά της που μπορούν να χρησιμοποιηθούν για συγχρονισμό των προσβάσεων στη μνήμη, παρέχει υποστήριξη *pthread* μέσω λειτουργικού



Σχήμα 3.7: Η αρχιτεκτονική της Myriad1.

Linux, *Test-and-Set* και *Atomic Primitives with CAS* (Πίνακας 3.3).

Η Myriad είναι ένα ετερογενές MPSoC στα 65nm σχεδιασμένο από την Movidius Ltd. και στοχεύει σε φορητές συσκευές. Πραγματοποιεί επεξεργασία βίντεο και ροών δεδομένων με χαμηλή κατανάλωση ενέργειας. Το διάγραμμα της αρχιτεκτονικής της απεικονίζεται στο Σχήμα 3.7. Περιλαμβάνει έναν επεξεργαστή LEON3 και 8 VLIW πυρήνες, που ονομάζονται SHAVEs. Σχετικά με τα χαρακτηριστικά της μνήμης, η Myriad περιέχει 1MB SRAM μνήμης, διαθέσιμη σε όλους τους SHAVEs, η οποία δεν είναι cached, και 64MB DDR. Περισσότερες πληροφορίες περιέχονται στο [6]

Σχετικά με τα χαρακτηριστικά της Myriad που μπορούν να χρησιμοποιηθούν για συγχρονισμό, περιέχει 8 spinlocks, υλοποιημένα ως fair κλειδώματα με διατησία (arbitration) round-robin. Επιπλέον, περιέχει ένα σετ καταχωρητών (buffers) που μπορούν να χρησιμοποιηθούν για αποτελεσματική επικοινωνία μεταξύ των SHAVEs. Κάθε SHAVE έχει το δικό του αντίγραφο αυτών των καταχωρητών και το μέγεθος καθενός είναι 4x64 bits. Μπορούν να χρησιμοποιηθούν για την υλοποίηση του μοντέλου client-server, όπως παρουσιάζεται στο [45] και περιγράφεται αναλυτικά στο επόμενο κεφάλαιο. Επομένως, όσο αφορά τα χαρακτηριστικά που χρησιμοποιούνται για συγχρονισμό, η Myriad υποστηρίζει τα *Custom/Platform-specific locks* και *Message Passing communication* (Πίνακας 3.3).

Πίνακας 3.6: Συνοπτική περιγραφή των benchmarks

Test case	Distrib. of operations or dataset	Access Pattern	# conc. d.s.
Deque	50% push 50% pop	deque	1
Database	20% write, 80% read	key-value pairs storage	1
Patricia	40% unique keys [50]	string storage	1
Dedup	[51]	key-value pairs storage	1
Streaming Aggregation	[52]	key-value pairs sorted storage	2

3.6.2 Περιγραφή των benchmarks

Οι πλατφόρμες Myriad και I.MX. 6 Quad, παρόλο που στοχεύουν σε διαφορετικά πεδία εκτελούν εφαρμογές που περιέχουν ταυτόχρονες δομές δεδομένων, όπως βάσεις δεδομένων και επεξεργασίας ροής δεδομένων. Εφαρμόσαμε την μεθοδολογία σε 5 benchmarks από διαφορετικούς χώρους, τα οποία εκτελέστηκαν στις δύο πλατφόρμες. Είναι αντιπροσωπευτικά εφαρμογών που εκτελούνται σε σύγχρονα ενσωματωμένα συστήματα και παρουσιάζονται στον Πίνακα 3.6.

Το πρώτο benchmark είναι μια ταυτόχρονη deque. Τα χαρακτηριστικά του είναι παρόμοια με αυτά εργασιών που αξιολογούν υλοποιήσεις deque [53]. Εκτελέστηκαν τρία πειράματα για το συγκεκριμένο benchmark: Στο πρώτο, η deque είναι αρχικοποιημένη με 100.000 στοιχεία. Στο δεύτερο, είναι αρχικοποιημένη όπως και στο προηγούμενο πείραμα, αλλά κάθε νήμα εκτελεί ένα συνθετικό workload μετά από κάθε λειτουργία. Έτσι, εξετάζουμε την περίπτωση όπου η συμφόρηση στη δομή δεδομένων είναι σχετικά μειωμένη σε σχέση με το προηγούμενο πείραμα. Τέλος, στο τελευταίο πείραμα η deque δεν είναι αρχικοποιημένη με στοιχεία και η συμφόρηση είναι αυξημένη, όπως και στο πρώτο πείραμα. Το δεύτερο benchmark είναι μία in-memory non-relational database. Αποθηκεύει key-value ζεύγη και υποστηρίζει λειτουργίες εύρεσης και αποθήκευσης. Είναι αρχικοποιημένη με 20.000 32-bit κλειδιά. Η εφαρμογή Patricia ανήκει στη σουίτα benchmarks Mibench [54]. Η πολυπύρηνη έκδοση της συγκεκριμένης εφαρμογής βασίζεται στη χρήση κλειδωμάτων για προστασία και κάθε πυρήνας εκτελεί την ίδια ποσότητα workload [50]. Το dedup είναι ένας αλγόριθμος data deduplication από τη σουίτα εφαρμογών Parsec [51]. Τέλος, η τελευταία εφαρμογή πραγματοποιεί multiway streaming aggregation [52]. Χρησιμοποιήθηκε για να παρουσιαστεί η χρήση της μεθοδολογίας σε εφαρμογές με περισσότερες από μία ταυ-

τόχρονες δομές δεδομένων. Η συγκεκριμένη εφαρμογή εκτελεί tuple aggregation και χρησιμοποιεί δύο ταυτόχρονες δομές δεδομένων.

Σε όλα τα πειράματα ο αριθμός των νημάτων που χρησιμοποιούνται, δεν ξεπερνά τον αριθμό των διαθέσιμων πυρήνων. Με άλλα λόγια, όλες οι εφαρμογές που εκτελούνται στην 4-πύρηνη I.MX. 6 Quad, χρησιμοποιούν 4 νήματα, ενώ στην 8-πύρηνη Myriad, έως 8 νήματα με κάθε πυρήνα να αναλαμβάνει ένα συγκεκριμένο νήμα σε όλη τη διάρκεια της εκτέλεσης. Στη Myriad εκτελέστηκαν πειράματα για 2, 4, 6 και 8 πυρήνες. Αυτό ισχύει για όλες τις εφαρμογές εκτός του streaming aggregation, στο οποίο εκτελέστηκαν πειράματα μόνο για 8 πυρήνες. Η διάρκεια κάθε πειράματος είναι τουλάχιστον ένα λεπτό. Τέλος, όλες οι τιμές στα πειράματα που εκτελέστηκαν είναι ο μέσος όρος 10 εκτελέσεων, με αφαίρεση των ακραίων τιμών.

Ο χρόνος εκτέλεσης στο I.MX. 6 Quad μετρήθηκε με χρήση της συνάρτησης *gettimeofday*, ενώ στη Myriad με χρήση αντίστοιχης συνάρτησης που παρέχεται από Myriad SDK. Η κατανάλωση ενέργειας μετρήθηκε με χρήση του Watts Up PRO power meter, ακολουθώντας την διαδικασία που περιγράφεται στο [43]. Η συσκευή συνδέθηκε απευθείας στην παροχή ρεύματος της πλατφόρμας I.MX. 6, ενώ στην Myriad μετρήθηκε με χρήση ενός shunt resistor συνδεδεμένου στο καλώδιο παροχής ρεύματος. Το fairness μετρήθηκε με τον τρόπο που περιγράφεται στο [55]. Προϋπόθεση είναι όλα τα νήματα να εκτελούν την ίδια ποσότητα workload, για το ίδιο χρονικό διάστημα. Τιμές κοντά στο 1 δείχνουν αυξημένο fairness, ενώ μικρότερες τιμές δείχνουν starvation των νημάτων.

3.6.3 Εφαρμογή της μεθοδολογίας στη Freescale I.MX 6 Quad

Ο Πίνακας 3.8 απεικονίζει συνοπτικά τους περιορισμούς, τα ενεργοποιημένα δέντρα αποφάσεων και τον αριθμό των υλοποιήσεων που εξετάστηκαν για κάθε benchmark στην I.MX.6 Quad. Τα Σχήματα 3.8, 3.9, 3.10 δείχνουν την απόδοση σε σχέση με την μέση ισχύ για τα τρία σενάρια του deque benchmark στην I.MX. 6 Quad. Αξιολογήθηκαν 18 διαφορετικές υλοποιήσεις και κάθε σημείο των διαγραμμάτων αντιστοιχεί σε μία διαφορετική υλοποίηση της deque. Οι κατά Pareto βέλτιστες υλοποιήσεις απεικονίζονται στον Πίνακα 3.7. Παρατηρούμε ότι οι πιο αποδοτικές υλοποιήσεις είναι αυτές που βασίζονται σε πίνακα. Το υψηλό locality των πινάκων οδηγεί σε καλύτερη εκμετάλλευση της ιεραρχίας μνήμης του I.MX.6 (που περιλαμβάνει cache μνήμη). Πέρα από αυτό, απαιτείται μικρότερος αριθμός προσβάσεων στη μνήμη για την εισαγωγή και την εξαγωγή στοιχείων σε πίνακα (μετακίνηση ενός δείκτη), σε σχέση με την διπλά συνδεδεμένη λίστα. Στο σενάριο υψηλής συμφόρησης που παρουσιάζεται στο Σχήμα 3.8 ευνοούνται οι υλοποιήσεις που κάνουν χρήση της τεχνικής back-off *DQH1*, *DQH2*. Παρόλα αυτά, καθώς το επίπεδο συμφόρησης μειώνεται, οι υλοποιήσεις back-off παρέχουν χαμηλή απόδοση εξαιτίας της μη αποτελεσματικής χρήσης των κλειδωμάτων. Η τεχνική back-off αποδίδει σε κατάσταση υψηλής συμφόρησης. Αντίθετα, σε καταστάσεις χαμηλής συμφόρησης, τα spinlocks και TTAS κλειδώματα

Πίνακας 3.7: Περιγραφή των βέλτιστων κατά Pareto υλοποιήσεων στην I.MX 6 Quad

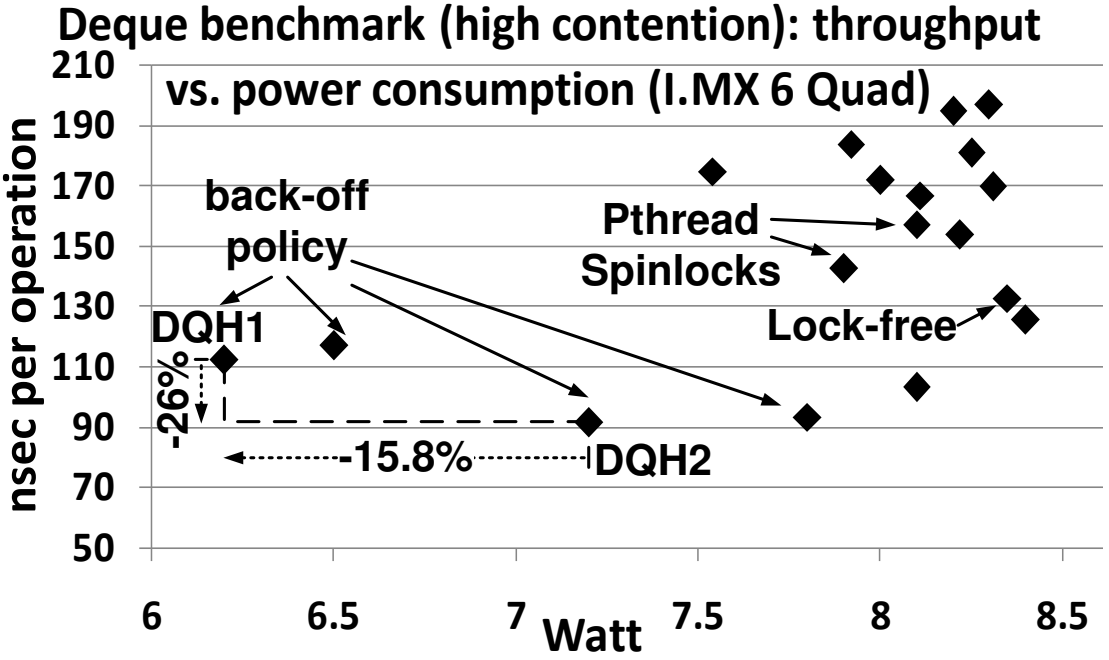
Pareto Point	Data Structure Implementation	Pareto Point	Data Structure Implementation
DQH1	A3(DLL), C1(TTAS), C2(exp), D1(fine)	P2	A7(trie), E1(atomic), E5(hazard)
DQH2	A3(array), C1(TTAS), C2(exp), D1(fine)	DD1	A8(hash-closed), B2(RW locks), D1(coarse)
DQL1	A3(array), C1(TTAS), C2(linear), D1(fine)	DD2	A5(hash-closed), E1(atomic), E5(hazard)
DQL2	A3(array), C1(TTAS), C2(none), D1(fine)	DD3	A5(hash-closed), B2(RW locks), D1(lock-str.)
DQN1	A3(array), C1(TTAS), C2(linear), D1(fine)	S1	A6(skip list), E1(atomic), E5(hazard)/
DQN2	A3(array), C1(TAS), C2(none), D1(fine)		A6(skip-list), B2(RW-locks), D1(lock-str.)
DB1	A8(b-Tree), B2(RW locks), D1(coarse)	S2	A6(skip list), B2(RW locks), D1(lock-str.) /
DB2	A5(open), B2(RW locks), D2(lock-str.)		A6(skip list), E1(atomic), E5(hazard)
DB3	A5(closed), B2(RW locks), D2(lock-str.)	S3	A6(skip list), E1(atomic), E5(hazard) /
P1	A7(trie), B2(RW locks), D1(coarse)		A6(skip list), E1(atomic), E5(hazard)

Πίνακας 3.8: Συνοπτική περιγραφή των περιορισμών και των ενεργοποιημένων δέντρων αποφάσεων για κάθε benchmark στην I.MX.6 Quad

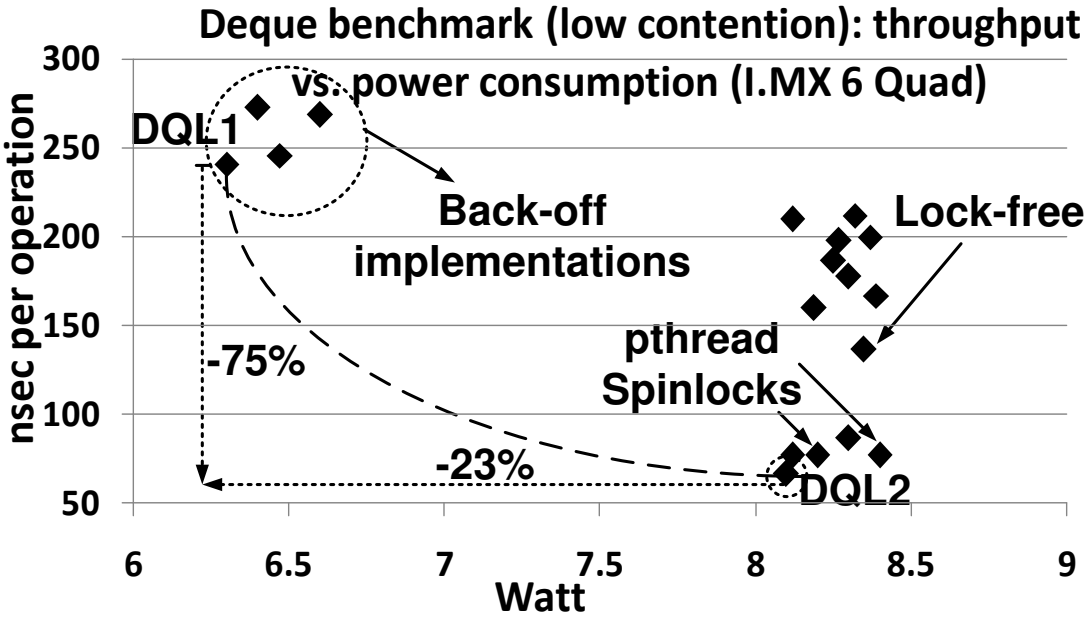
I.MX.6			
Application	Contraints	Trees enabled	Num. implem. generated
Deque	Acc. pattern: Deque Num cores: many Sync. Support: Pthreads, Test-and-Set, Atomic prim. CAS	A3 B1 C1, C2 D E1, E5	18
Database	Acc. pattern: key-value pairs Num cores: many Sync. Support: (same with deque)	A5, A6, A8 B2 D1 E1, E5	7
Patricia	Acc. pattern: string storage Num cores: many Sync. Support: (same with deque)	A7 B2 D1 E1, E5	2
Dedup	Acc. pattern: key-value pairs Num cores: many Sync. Support: (same with deque)	A5, A6, A8 B2 D1 E1, E5	7
Streaming	Acc. pattern: key-value pairs sorted Num cores: many Sync. Support: (same with deque)	A6, A8 B2 D1 E1, E5	3^2

χωρίς back-off παρέχουν υψηλότερη απόδοση *DQL2*. Τέλος, στην περίπτωση της μη αρχικοποιημένης deque η απόδοση και η κατανάλωση ενέργειας εξαρτώνται κυρίως από τη σειρά με την οποία πραγματοποιούνται οι λειτουργίες: Όσο μεγαλύτερος ο αριθμός των περιπτώσεων που ένα νήμα προσπαθεί να αφαιρέσει ένα στοιχείο από την deque και αυτή είναι άδεια, τόσο μικρότερη είναι και η απόδοση.

Τα αποτελέσματα για το database benchmark στο I.MX.6 Quad απεικονίζονται στο Σχήμα 3.11. Συνολικά, αξιολογήθηκαν 7 υλοποιήσεις και εντοπίστηκαν τρεις βέλτιστες κατά Pareto: Είναι το lock-based b-tree (*DB1*) και δύο lock-based hash tables (*DB2* και *DB3*). Η υλοποίηση *DB3* είναι αυτή που παρέχει το υψηλότερο throughput, εξαιτίας της χρήσης lock-striping, η οποία παρέχει υψηλό παραλληλισμό. Μειωμένη ισχύς απαιτείται με τη χρήση της υλοποίησης *DB1*, δεδομένου ότι σε αυτή την περίπτωση είναι μικρότερος αριθμός νημάτων ταυτόχρονα ενεργός (runnable) σε σχέση με την υλοποίηση b-tree. Σε αυτήν, τα νήματα είναι συχνά suspended (δηλ. σε κατάσταση



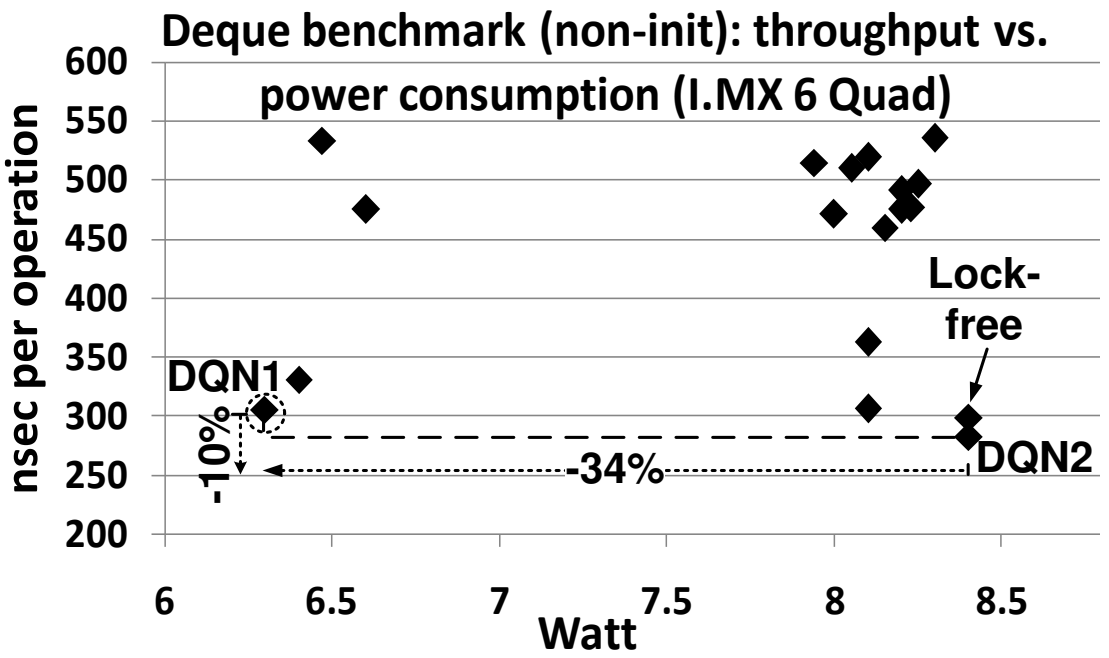
Σχήμα 3.8: Deque σε υψηλή συμφόρηση στην I.MX.6 Quad.



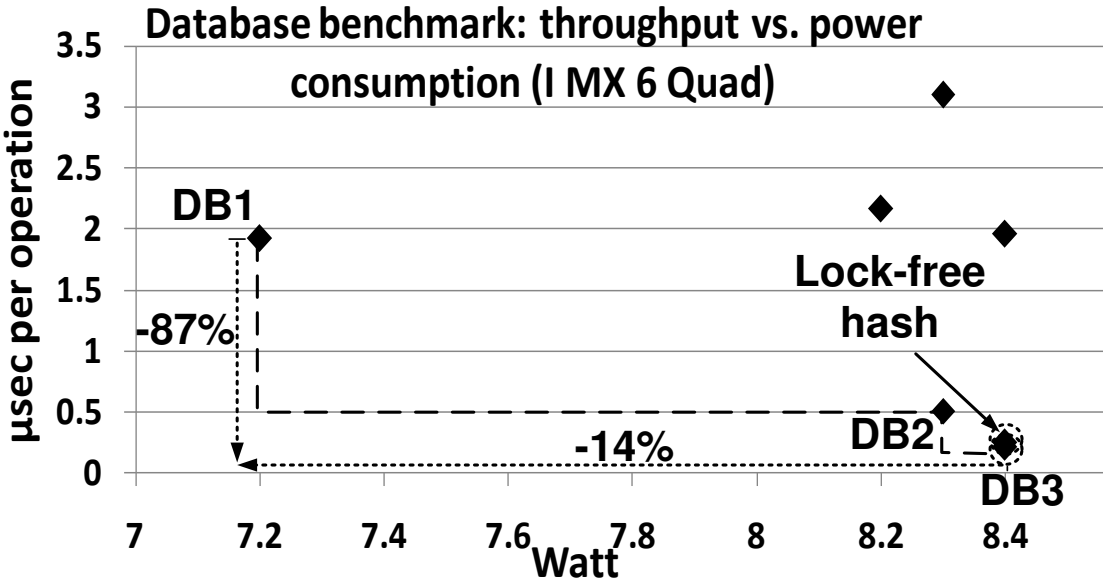
Σχήμα 3.9: Deque σε χαμηλή συμφόρηση στην I.MX.6 Quad.

αναμονής), περιμένοντας το κλείδωμα να γίνει διαθέσιμο.

Υπάρχουν δύο διαθέσιμες υλοποιήσεις για το Patricia benchmark στο I.MX.6 Quad (Σχήμα 3.12). Το σχήμα πρόσβασης string storage ενεργοποιεί το δέντρο αποφάσεων *A7* και επομένως οι διαθέσιμες υλοποιήσεις για τη δομή δεδομένων trie είναι οι εξής:

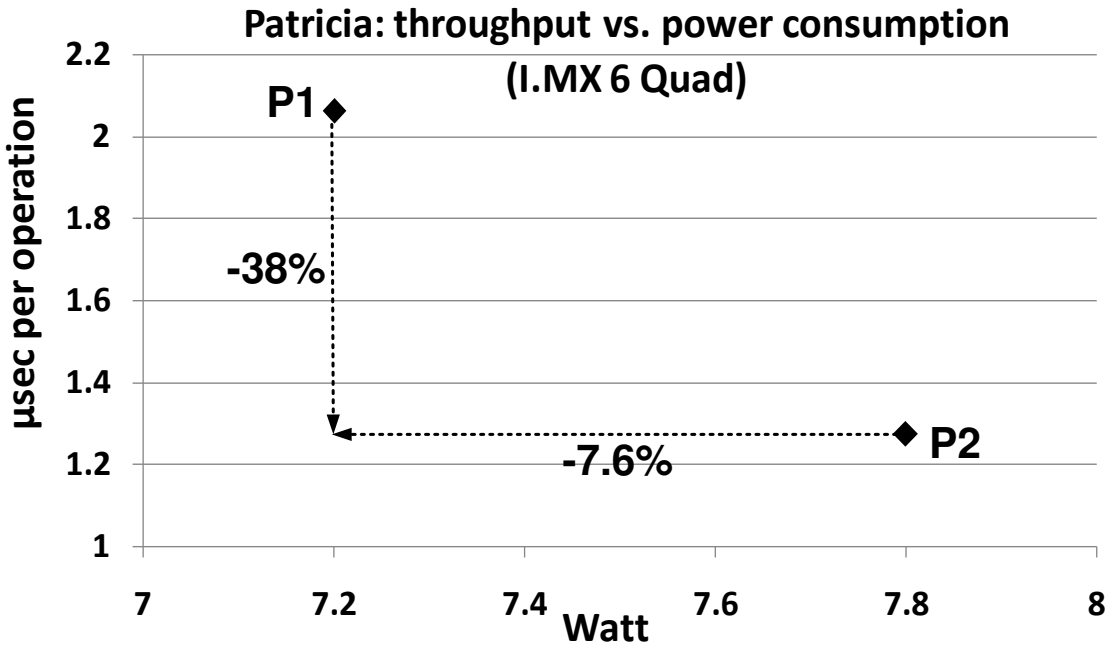


Σχήμα 3.10: Μη αρχικοποιημένη Deque σε υψηλή συμφόρηση στην I.MX.6 Quad.

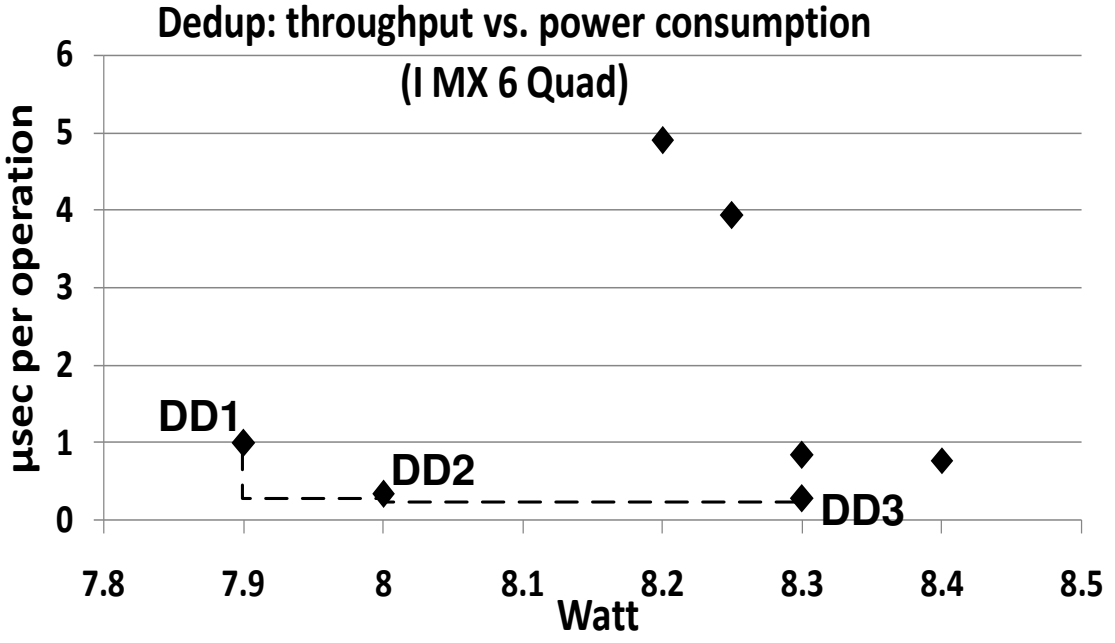


Σχήμα 3.11: Database benchmark στην I.MX.6 Quad.

ένα lock-based radix tree που κάνει χρήση *pthread-rw locks* (*P1*) και η υλοποίηση Ctrie που κάνει χρήση atomic primitives (*P2*). [56]. Το Ctrie έχει σχεδιαστεί για αποτελεσματική χρήση της cache και επομένως παρέχει υψηλή απόδοση στο I.MX.6 Quad. Παρόλα αυτά, το γεγονός ότι στις lock-free υλοποιήσεις τα νήματα είναι συνεχώς ενεργά (π.χ. πραγματοποιώντας συνεχώς προσπάθειες να ολοκληρώσουν επιτυχώς



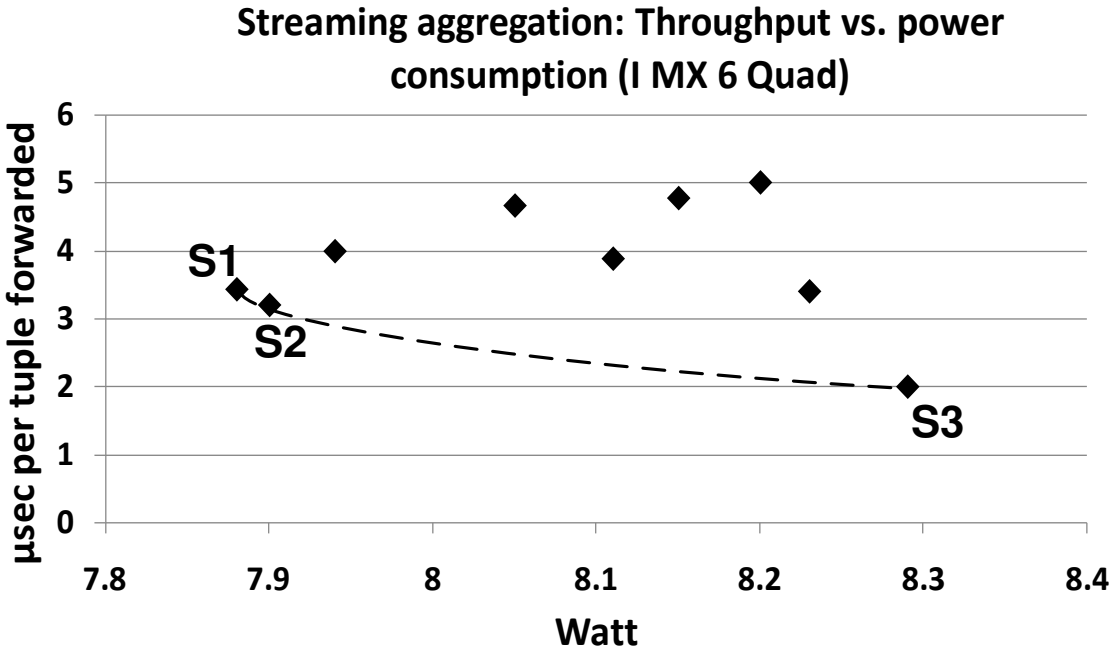
Σχήμα 3.12: Patricia benchmark στην I.MX.6 Quad.



Σχήμα 3.13: Dedup benchmark στην I.MX.6 Quad.

atomic operations που αποτυγχάνουν) εξηγεί την αυξημένη απαιτούμενη ισχύ της υλοποίησης Ctrie.

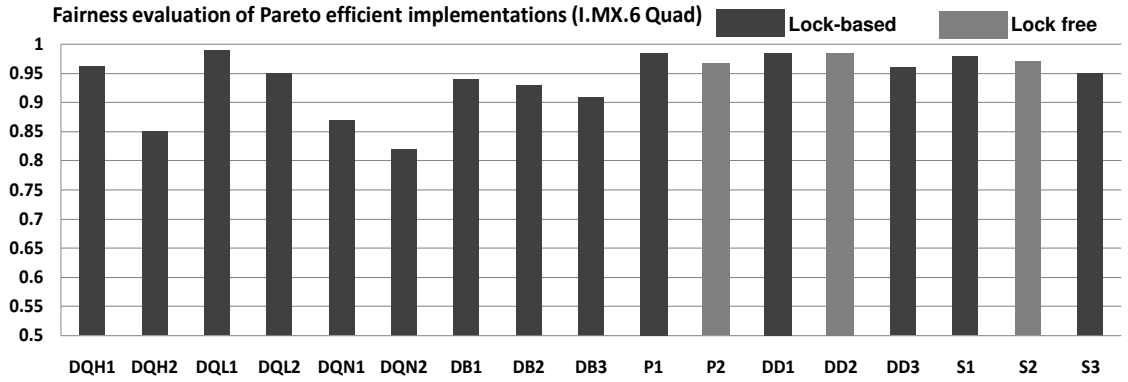
7 διαφορετικές υλοποιήσεις αξιολογήθηκαν για την εφαρμογή Dedup στο I.MX.6 Quad. Εντοπίστηκαν ανταλλάγματα μεταξύ throughput και ισχύος που παρουσιάζο-



Σχήμα 3.14: Streaming aggregation benchmark στην I.MX.6 Quad.

νται στο Σχήμα 3.13. Η υλοποίηση *DD1* απαιτεί 4,8% λιγότερη ισχύ σε σχέση με την υλοποίηση *DD3* που παρέχει 71% υψηλότερο throughput. Όπως φαίνεται στον Πίνακα 3.13, η *DD1* είναι μία lock-based hash table υλοποίηση της ταυτόχρονης δομής δεδομένων του Dedup με coarse-grained locking, ενώ η *DD3* είναι ο lock-based hash table με fine-grained locking. Η συγκεκριμένη υλοποίηση παρέχει υψηλότερο παραλληλισμό σε σχέση την υλοποίηση *DD1*, λόγω της χρήσης lock-striping. Παρόλα αυτά, στην περίπτωση αυτή είναι περισσότεροι πυρήνες ενεργοί ταυτόχρονα, με συνέπεια την αύξηση της απαιτούμενης ισχύος. Η υλοποίηση *DD2* ειναι ένας lock-free hash table. Παρέχει ελαφρώς χαμηλότερη απόδοση σε σχέση με την *DD3*, αλλά απαιτεί περισσότερη ισχύ εξαιτίας του γεγονότος ότι οι πυρήνες είναι συνεχώς ενεργοί. Αντίστοιχα συμπεράσματα για την κατανάλωση ενέργειας των lock-free δομών δεδομένων μπορούν να βρεθούν στη βιβλιογραφία [43]. Η υλοποίηση *DD1* με coarse-grained locking απαιτεί μικρότερη ισχύ, δεδομένου ότι τα νήματα γίνονται συχνά suspended, αναμένοντας το κλείδωμα να ελευθερωθεί.

Στην εφαρμογή streaming aggregation αξιολογήθηκαν 9 διαφορετικές υλοποιήσεις: 3 για κάθε δομή δεδομένων της εφαρμογής (Σχήμα 3.14). Εντοπίστηκαν τρεις βέλτιστες κατά Pareto υλοποιήσεις. Η υλοποίηση *S1* αντιστοιχεί σε μία lock-free skip-list για την πρώτη δομή δεδομένων (στην οποία εισέρχονται τα tuples) και σε lock-based skip-list στη δομή δεδομένων στην οποία υπολογίζεται η aggregated τιμή. Η υλοποίση *S2* είναι ο ακριβώς αντίστροφος συνδυασμός δομών δεδομένων, ενώ ο συνδυασμός *S3* αντιστοιχεί σε lock-free υλοποιήσεις και για τις δύο δομές δεδομένων. Παρατηρούμε ότι η τελευταία υλοποίηση παρέχει το υψηλότερο throughput, σε συνδυασμό με υψη-



Σχήμα 3.15: Fairness για τις βέλτιστες κατά Pareto υλοποιήσεις στην I.MX.6 Quad.

λότερη κατανάλωση ενέργειας (42% και 5.1% αντιστοίχως αυξημένα σε σχέση με την *S1*). Αντίστοιχα αποτελέσματα σχετικά με την απόδοση των lock-free δομών δεδομένων σε streaming aggregation υλοποιήσεις μπορούν να βρεθούν στη βιβλιογραφία [52].

Στο Σχήμα 3.15 παρουσιάζονται οι τιμές fairness για τις υλοποιήσεις στο I.MX. 6 Quad που είναι βέλτιστες κατά Pareto. Γενικά, οι περισσότερες υλοποιήσεις παρέχουν υψηλό fairness σχετικά κοντά στο 0,9. Αυτό ισχύει τόσο για τις lock-based, όσο και για τις lock-free υλοποιήσεις. Μια ενδιαφέρουσα παρατήρηση είναι το γεγονός ότι στο deque και στο database benchmark, οι υλοποιήσεις που παρέχουν την υψηλότερη απόδοση (π.χ. *DQH2*, *DB3*) παρέχουν το χαμηλότερο fairness. Αυτό είναι αποτέλεσμα της συμφόρησης στα κλειδώματα. Πραγματικά, το φαινόμενο αυτό είναι λιγότερο έντονο στα πειράματα του deque, όπου η συμφόρηση είναι μικρότερη (*DQL1* και *DQL2*).

3.6.4 Εφαρμογή της μεθοδολογίας στη Myriad

Ο Πίνακας 3.10 απεικονίζει συνοπτικά τους περιορισμούς, τα ενεργοποιημένα δέντρα αποφάσεων και τον αριθμό των υλοποιήσεων που εξετάστηκαν για κάθε benchmark στη Myriad. Τα αποτελέσματα της εφαρμογής της μεθοδολογίας για το deque benchmark παρουσιάζονται στα Σχήματα 3.16, 3.17, 3.10, ενώ οι σχεδιαστικές αποφάσεις των βέλτιστων κατά Pareto υλοποιήσεων απεικονίζονται στον Πίνακα 3.9. Τέσσερις υλοποιήσεις αξιολογήθηκαν για 2, 4, 6 και 8 νήματα. Παρατηρούμε ότι σε αντίθεση με τα αντίστοιχα πειράματα στην I.MX. 6 Quad, πολλά βέλτιστα κατά Pareto σημεία που παρέχουν υψηλή απόδοση είναι υλοποιήσεις λιστών, αντί για πινάκων (π.χ. *DQH(4)2*, *DQH(6)2*, *DQL(2)2*). Σε αντίθεση με την I.MX. 6 Quad, η Myriad έχει διαφορετική ιεραρχία μνήμης, δίχως cache. Οι υλοποιήσεις που βασίζονται στο μοντέλο client-server όχι μονο απαιτούν γενικά χαμηλή ισχύ, αλλά σε κάποιες περιπτώσεις παρέχουν και υψηλή απόδοση, ιδιαίτερα όταν ο αριθμός των νημάτων είναι σχετικά μικρός (π.χ. *DQH(4)2*, *DQL(4)2*, *DQN(2)*). Επιπλέον, καθώς το επίπεδο συμφόρησης μειώνεται,

Πίνακας 3.9: Περιγραφή των βέλτιστων κατά Pareto υλοποιήσεων στη Myriad

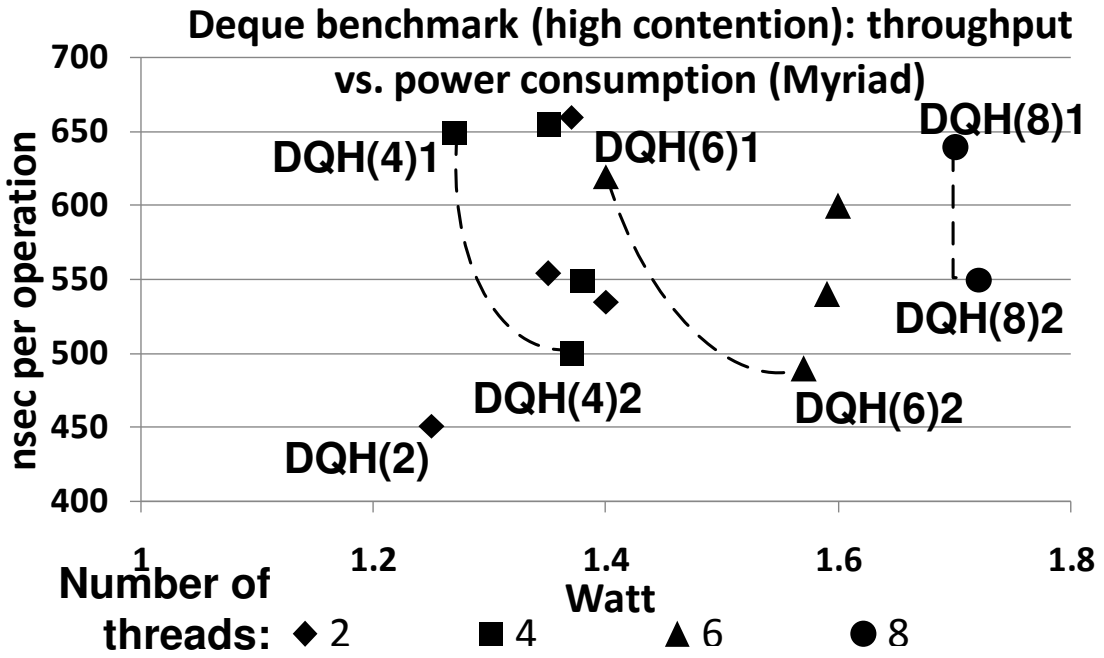
Pareto Point	Data Structure Implementation	Pareto Point	Data Structure Implementation
DQH(2), DQH(8)1	A3(array), C3(custom), D1(fine)	DB(6)3, DB(8)2	A5(hash-closed), C3(custom), D1(lock-str.)
DQH(4)1	A3(array), E2(client-server)	DB(8)1	A8(b-tree), C3(custom), D1(coarse)
DQH(4)2, DQH(6)1	A3(DLL), E2(client-server)	P(2)1, P(4)1, P(6)1	A7(trie), E2(client-server)
DQH(6)2, DQH(8)2	A3(DLL), C3(custom), D1(fine)	P(2)2, P(4)2, P(6)2, P(8)	A7(trie), C3(custom), D1(coarse)
DQL(2)1	A3(array), C3(custom), D1(fine)	DD(2)1, DD(4)3, DD(6)1	A8(hash-open), E2(client-server)
DQL(2)2, DQL(8)	A3(DLL), C3(custom), D1(fine)	DD(2)2, DD(4)2, DD(6)2	A5(hash-closed), E2(client-server)
DQL(4)1, DQL(6)	A3(DLL), E2(client-server)	DD(4)1	A5(hash-open), C3(custom), D1(lock-str.)
DQL(4)2	A3(array), E2(client-server)	DD(6)3, DD(8)2	A5(hash-closed), C3(custom), D1(lock-str.)
DQN(2)	A3(DLL), E2(client-server)	DD(8)1	A8(hash-closed), C3(custom), D1(coarse)
DQN(4), DQN(6)2	A3(array), E2(client-server)	S1	A6(skip-list), E2(client-server) /
DQN(6)1, DQN(8)	A3(DLL), C3(custom), D1(fine)		A6(skip-list), E2(client-server)
DB(2)1, DB(4)1, DB(6)1	A8(b-tree), E2(client-server)	S2	A6(skip-list), C3(custom), D1(lock-str.) /
DB(2)2, DB(4)2, DB(6)2	A5(hash-closed), E2(client-server)		A6(skip-list), C3(custom), D1(lock-str.)

Πίνακας 3.10: Συνοπτική περιγραφή των περιορισμών και των ενεργοποιημένων δέντρων αποφάσεων για κάθε benchmark στην Myriad

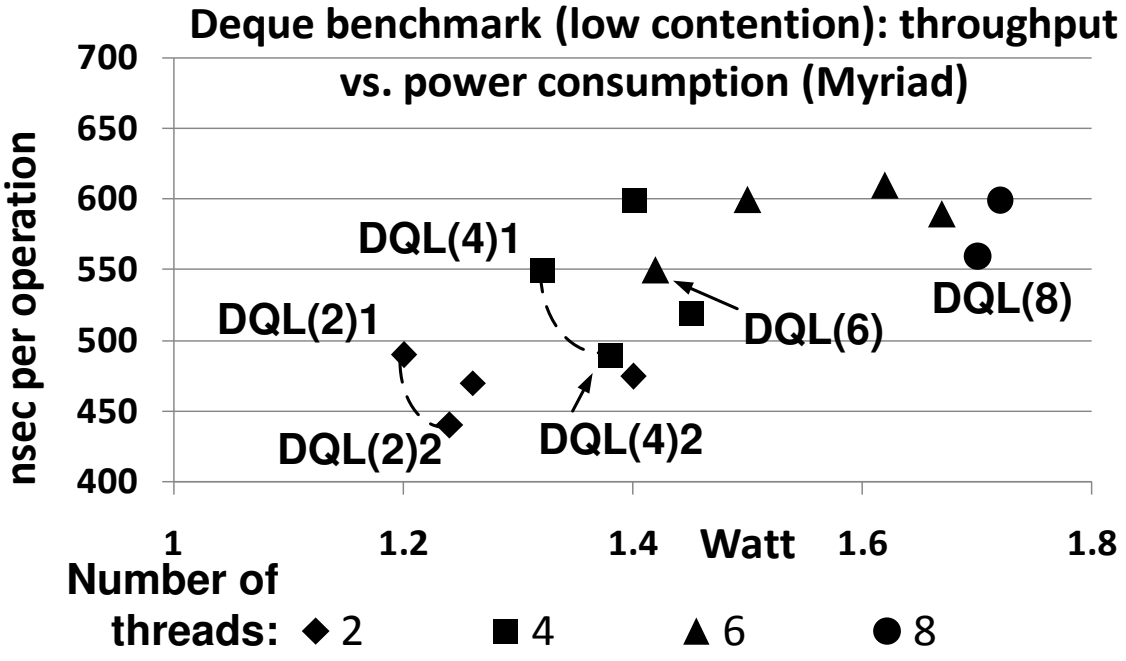
Myriad			
Application	Contraints	Trees enabled	Num. implem. generated
Deque	Acc. pattern: Deque Num cores: many Sync. Support: custom/platf.spec., message based comm.	A3 C3 D1 E2	4
Database	Acc. pattern: key-value pairs Num cores: many Sync. Support: (same with deque)	A5, A6, A8 C3 D1 E2	8
Patricia	Acc. pattern: string storage Num cores: many Sync. Support: (same with deque)	A7 C3 D1 E2	2
Dedup	Acc. pattern: key-value pairs Num cores: many Sync. Support: (same with deque)	A5, A6, A8 C3 D1 E2	8
Streaming	Acc. pattern: key-value pairs sorted Num cores: many Sync. Support: (same with deque)	A6, A8 C3 D1 E2	4^2

παρατηρούμε στο Σχήμα 3.17 ότι η διαφορά ανάμεσα στην απόδοση των υλοποιήσεων client-server και των αντίστοιχων lock-based γίνεται μικρότερη. Για παράδειγμα, η υλοποίηση *DQH(4)2* παρέχει 31% υψηλότερο throughput σε σχέση με την *DQH(4)1*, ενώ η διαφορά ανάμεσα στο *DQL(4)2* και στο *DQL(4)1* είναι 11%. Τέλος, το πείραμα με την μη αρχικοποιημένη deque παρουσιάζεται στο Σχήμα 3.18. Όπως και στο αντίστοιχο πείραμα στην I.MX. 6 Quad, η απόδοση κάθε υλοποίησης εξαρτάται κυρίως από το πόσο συχνά επιχειρείται αφαίρεση δεδομένων από άδεια deque.

Οχτώ διαφορετικές υλοποιήσεις εξετάστηκαν για το Database benchmark στη Myriad. Οι καμπύλες Pareto απεικονίζονται στο Σχήμα 3.19. Στο πείραμα με οχτώ threads παρατηρούμε ότι οι βέλτιστες υλοποιήσεις είναι ίδιες με αυτές του αντίστοιχου πειράματος στο I.MX. 6 Quad (lock-based hash table και lock-based b-tree). Η υλοποίηση *DB(6)3* είναι ένας lock-based hash table, που παρέχει 32,6% υψηλότερο throughput σε σχέση με την *DB(6)1* και 4% υψηλότερη ισχύ. Η υλοποίηση *DB(6)2* είναι αντίστοιχη με την *DB(6)3*, αλλά βασίζεται στο μοντέλο client-server. Παρατη-



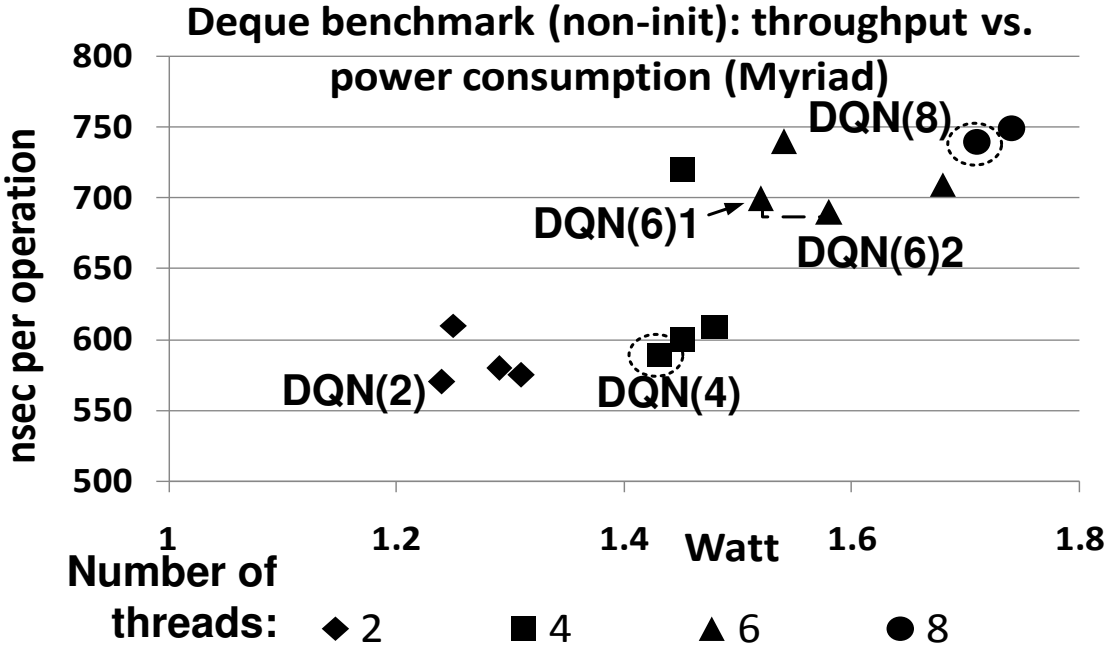
Σχήμα 3.16: Deque σε υψηλή συμφόρηση στη Myriad.



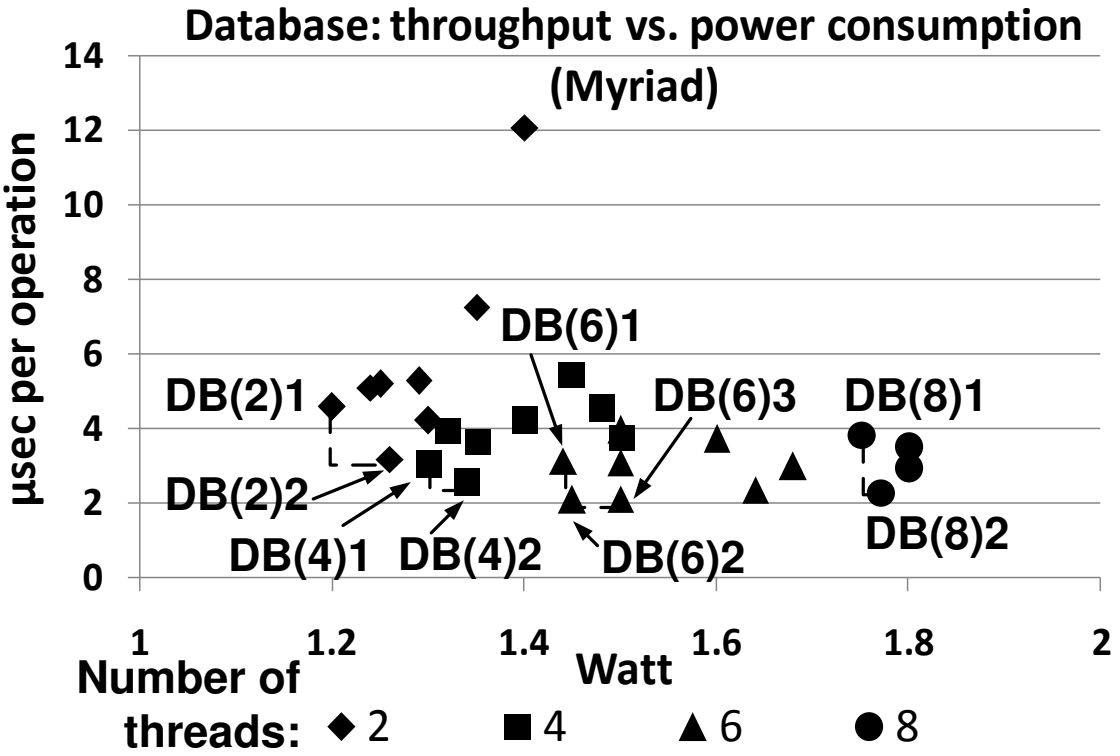
Σχήμα 3.17: Deque σε χαμηλή συμφόρηση στη Myriad.

ρούμε ότι καθώς ο αριθμός των νημάτων αυξάνει, ο lock-based hash table σταδιακά ξεπερνά σε απόδοση την αντίστοιχη υλοποίηση βασισμένη στο client-server, καθότι παρέχει υψηλότερο βαθμό παραλληλισμού.

Τα αποτελέσματα της εφαρμογής της μεθοδολογίας στο Patricia benchmark πα-

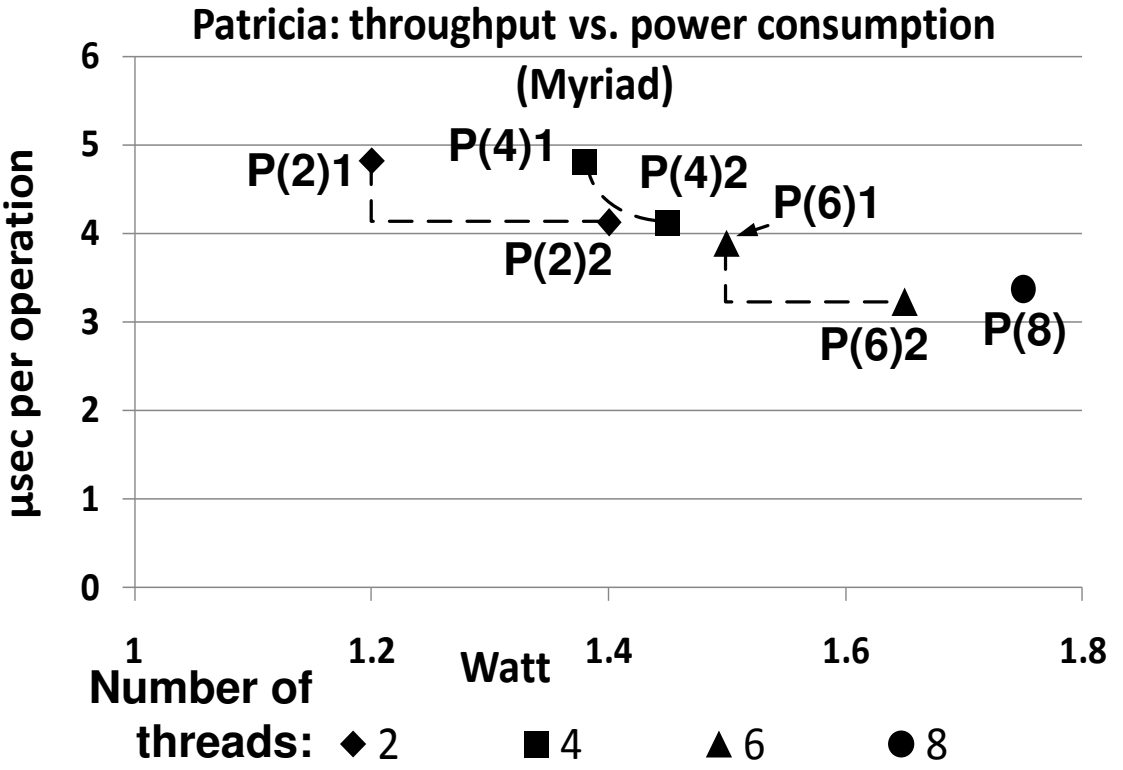


Σχήμα 3.18: Μη αρχικοποιημένη Deque σε υψηλή συμφόρηση στη Myriad.



Σχήμα 3.19: Database benchmark στη Myriad.

ρουσιάζονται στο Σχήμα 3.20. Υπάρχουν δύο διαθέσιμες υλοποιήσεις για το σχήμα πρόσβασης string storage: Η lock-based (*P2(2)*, *P(4)2*, *P(6)2*, *P(8)*) και client-server

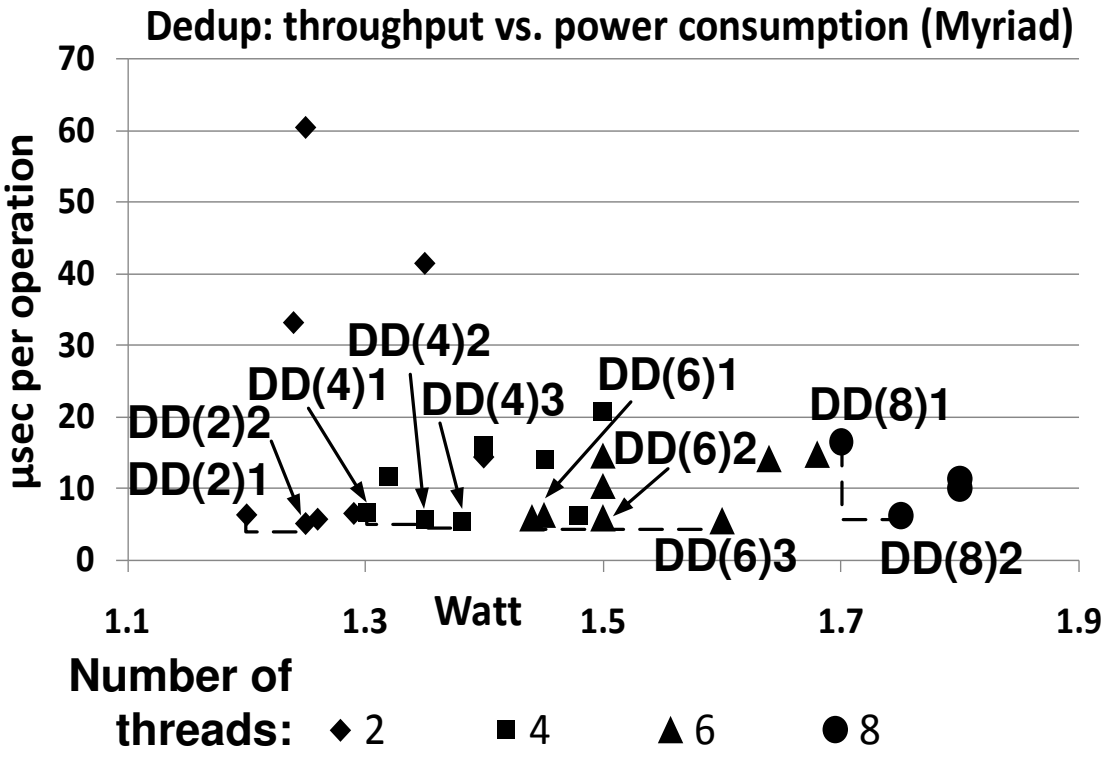


Σχήμα 3.20: Patricia benchmark στη Myriad.

(*P(2)1*, *P(4)1*, *P(6)1*). Ανταλλάγματα μεταξύ απόδοσης και ισχύος εντοπίστηκαν στα πειράματα για 2, 4 και 6 νήματα. Για παράδειγμα, στο πείραμα με 6 νήματα η lock-based trie υλοποίηση (*P(6)2*) παρέχει 17,2% υψηλότερο throughput, ενώ η αντίστοιχη client-server (*P(6)1*) απαιτεί 10,7% χαμηλότερη ισχύ σε σχέση με το *P(6)2*.

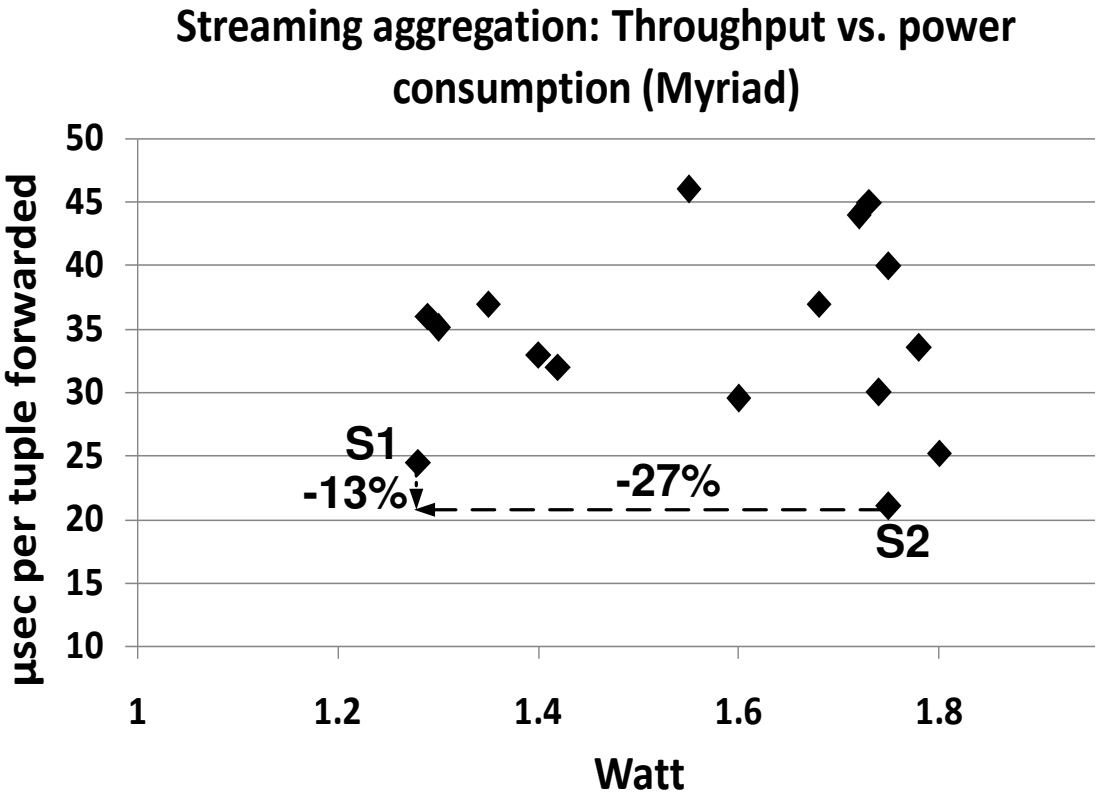
Για την εφαρμογή Dedup, τα αποτελέσματα throughput vs. power παρουσιάζονται στο Σχήμα 3.21. Εντοπίστηκαν ανταλλάγματα σε όλα τα πειράματα. Στο πείραμα με 8 νήματα, με χρήση του hash-table με coarse-grained locking (*DD(8)1*) η απαιτούμενη ισχύς μειώνεται κατά 1,2% σε σχέση με την αντίστοιχη υλοποίηση με fine-grained locking (*DD(8)2*). Η τελευταία υλοποίηση παρέχει αυξημένο throughput κατά 40% λόγω υψηλού παραλληλισμού, χάρη στην τεχνική lock-striping. Παρατηρούμε πως για μικρο αριθμό νημάτων, όπου η συμφόρηση είναι σχετικά μικρή οι υλοποιήσεις που βασίζονται στο client-server μοντέλο είναι αποδοτικότερες, τόσο όσο αφορά το throughput, όσο και την απαιτούμενη ισχύ. Για μεγαλύτερο αριθμό νημάτων (6 και 8), οι hash lock-based υλοποιήσεις είναι αποδοτικότερες.

Η εφαρμογή streaming aggregation υλοποιήθηκε με οχτώ νήματα (Σχήμα 3.22). Ο συνολικό αριθμός των υλοποιήσεων που εξετάστηκαν είναι δεκαέξι (τέσσερις για κάθε δομή δεδομένων). Η *S1* αντιστοιχεί σε client-server υλοποιήσεις και για τις δύο δομές δεδομένων. Δεδομένου ότι μόνο τρία νήματα έχουν πρόσβαση σε κάθε δομή δεδομένων, οι υλοποιήσεις client-server παρέχουν απόδοση συγκρίσιμη με τις αντίστοιχες lock-based.

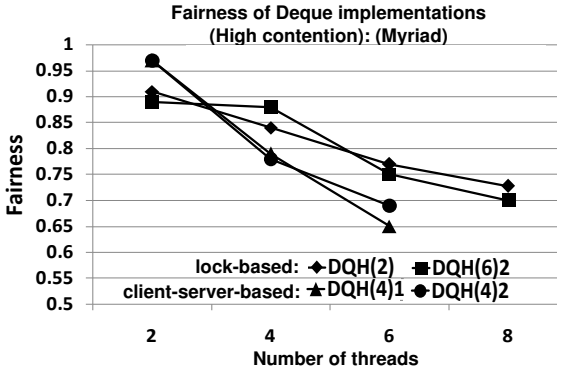


Σχήμα 3.21: Dedup benchmark στη Myriad.

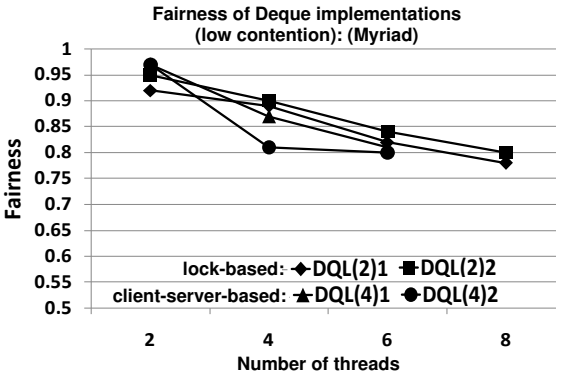
Στα Σχήματα 3.23α΄ – 3.23ζ΄ παρουσιάζονται αποτελέσματα σχετικά με το fairness των βέλτιστων κατά Pareto υλοποιήσεων για κάθε benchmark στη Myriad. Παρατηρούμε γενικά ότι παρόλο που οι υλοποιήσεις client-server παρέχουν fairness παρόμοιο με αυτό των lock-based ως 4 νήματα, στα πειράματα με περισσότερα νήματα το fairness είναι αρκετά μικρότερο. Για παράδειγμα, στα Patricia και Dedup (Σχήματα 3.23ε΄ και 3.23ζ΄) είναι κοντά στο 0,7. Αντίθετα, οι lock-based υλοποιήσεις παρέχουν fairness υψηλότερο από 0,75 σε όλες τις περιπτώσεις στα ίδια πειράματα. Πραγματικά, σε αντίθεση με κλειδώματα, στην επικοινωνία βασισμένη σε μηνύματα δεν υπάρχει κάποιος μηχανισμός που να εξασφαλίζει fairness. Έτσι, όσο η συμφόρηση είναι σχετικά μικρή το fairness είναι σχετικά ικανοποιητικό. Καθώς όμως ο αριθμός των νημάτων, και άρα και η συμφόρηση αυξάνουν, το fairness των client-server υλοποιήσεων μειώνεται.



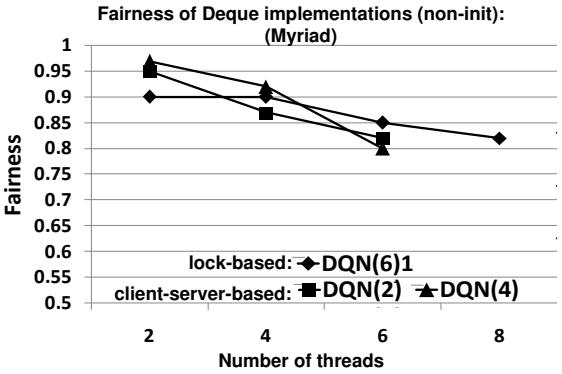
Σχήμα 3.22: Streaming aggregation benchmark στη Myriad.



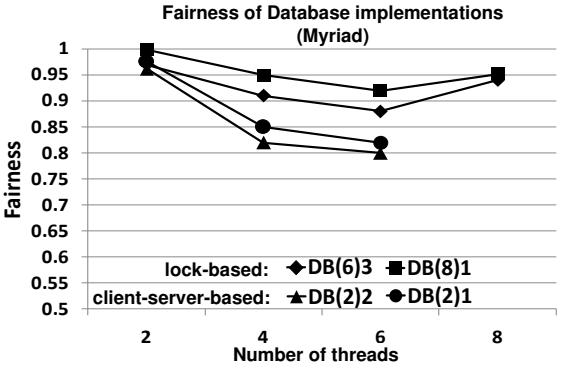
(α΄) deque σε υψηλή συμφόρηση



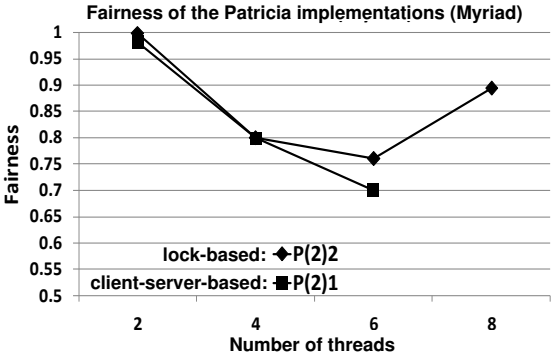
(β΄) deque σε χαμηλή συμφόρηση



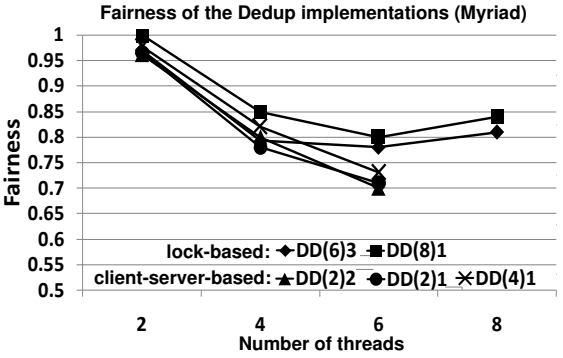
(γ΄) μη αρχικοποιημένη deque



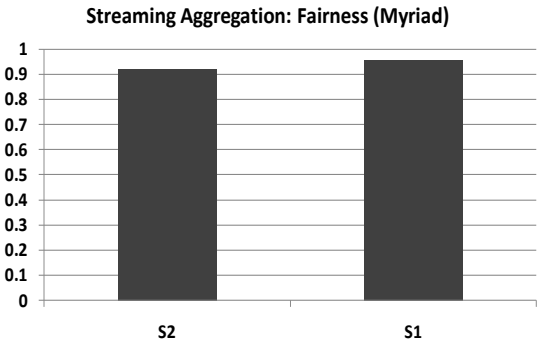
(δ΄) database benchmark



(ε΄) Patricia benchmark



(ς΄) Dedup benchmark



(ζ΄) Streaming aggregation benchmark

Σχήμα 3.23: Fairness για τις βέλτιστες κατά Pareto υλοποιήσεις στη Myriad.

3.6.5 Σχολιασμός των αποτελεσμάτων

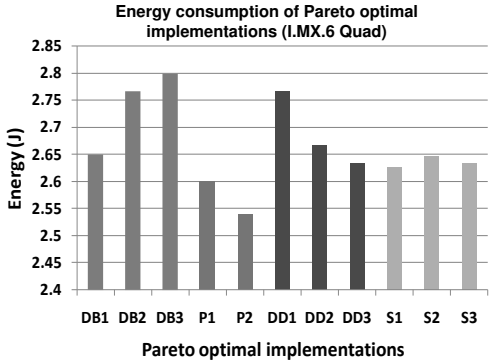
Μελετώντας τα πειραματικά αποτελέσματα μπορούμε να εξάγουμε συμπεράσματα για τα χαρακτηριστικά των αρχιτεκτονικών και των εφαρμογών τα οποία δεν περιορίζουν τον χώρο σχεδιασμού, αλλά επιδρούν στην κατανάλωση ενέργειας, στην απόδοση και στο fairness των σχεδιαστικών επιλογών.

Ο *ρυθμός* με τον οποίο αιτούνται τα νήματα λειτουργίες από τις κοινές δομές δεδομένων μπορεί να έχει επίδραση στη συμφόρηση των κλειδωμάτων και επομένως στην απόδοση των δέντρων αποφάσεων των κατηγοριών *B*, *C* και *D*. Υψηλός ρυθμός αιτήσεων μπορεί να προκαλέσει αυξημένη συμφόρηση, γεγονός που ευνοεί πολιτικές back-off, όπως φαίνεται στο πείραμα της deque με υψηλή συμφόρηση στο Σχήμα 3.8 στην I.MX.6 Quad. Επιπλέον, ο ρυθμός με τον οποίο πραγματοποιούνται λειτουργίες επιδρά και στο fairness. Όπως φαίνεται στο Σχήμα 3.15 για την deque, το fairness όταν η συμφόρηση είναι αυξημένη είναι μικρότερο σε σχέση με το πείραμα στο οποίο η συμφόρηση είναι μειωμένη. Αντίστοιχες παρατηρήσεις έχουν γίνει και σε άλλες εργασίες στη βιβλιογραφία [55].

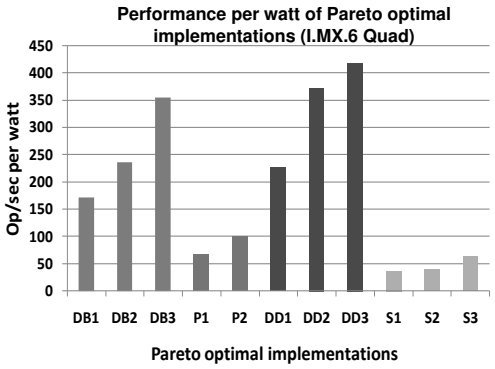
Η ιεραρχία μνήμης, και πιο συγκεκριμένα η ύπαρξη cache και μηχανισμών prefetching, δείχνει να επηρεάζει δέντρα αποφάσεων της κατηγορίας *A*, ευνοώντας υλοποιήσεις που έχουν υψηλό locality (π.χ. στο deque benchmark στην I.MX. 6 στο Σχήμα 3.8). Επιπλέον, η ύπαρξη cache επηρεάζει την απόδοση των υλοποιήσεων που κάνουν χρήση των δέντρων αποφάσεων των κατηγοριών *B*, *C* και *D*. Παρόλο που τα spinlocks παρέχουν υψηλή απόδοση σε περίπτωση χαμηλού ρυθμού αιτήσεων (π.χ. deque benchmark στο Σχήμα 3.9), ο συνδυασμός υψηλού αριθμού αιτήσεων και η ύπαρξη cache, τείνει να μειώνει την απόδοση. Η απόδοση μειώνεται ακόμη περισσότερο σε περίπτωση ύπαρξης fine-grain μηχανισμών κλειδωμάτων (δέντρο απόφασης *D1*), σε περίπτωση υψηλής συμφόρησης [55]. Το φαινόμενο αυτό δεν παρατηρείται στη Myriad, όπως φαίνεται στο πείραμα υψηλής συμφόρησης με 6 νήματα στο Σχήμα 3.16, καθότι η Myriad δεν περιέχει cache.

Ο αριθμός των νημάτων που έχουν πρόσβαση στην ταυτόχρονη δομή δεδομένων επηρεάζει το βαθμό συμφόρησης και κατά συνέπεια έναν μεγάλο αριθμό από δέντρα αποφάσεων του χώρου σχεδιαστικών επιλογών. Όπως φαίνεται στα πειράματα στη Myriad στα Σχήματα 3.23ε΄ και 3.23ζ΄, καθώς ο αριθμός των πυρήνων αυξάνει, το fairness μειώνεται [55]. Υψηλότερο fairness συναντάμε στα πειράματα με 2 πυρήνες. Τέλος, οι υλοποιήσεις client-server (δέντρο αποφάσεων *E2*) παρέχουν υψηλότερη απόδοση για μικρό αριθμό πυρήνων, όπως φαίνεται στα πειράματα των database και dedup benchmarks (Σχήμα 3.19 και 3.21).

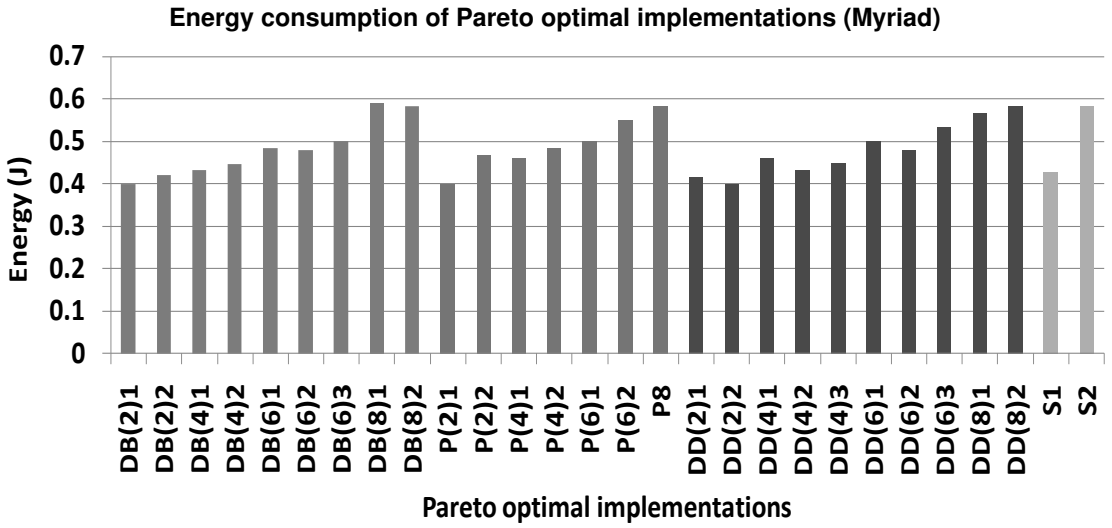
Τέλος, αξίζει να αναφερθεί ότι τα αποτελέσματα που προέκυψαν από την εφαρμογή της μεθοδολογίας, μπορούν να αξιοποιηθούν για την εξαγωγή άλλων αποτελεσμάτων. Για παράδειγμα, στο Σχήμα 3.24 απεικονίζονται αποτελέσματα κατανάλωσης ενέργειας και απόδοσης ανά watt για τα μη-συνθετικά benchmarks.



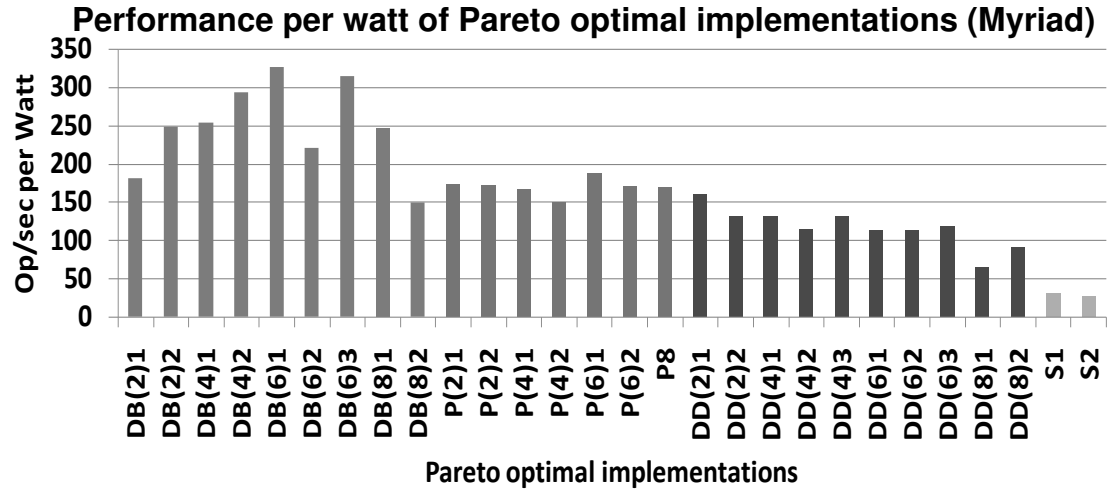
(α′) Κατανάλωση ενέργειας για τις βέλτιστες κατά Pareto υλοποιήσεις στην I.MX.6 Quad.



(β′) Απόδοση ανά watt για τις βέλτιστες κατά Pareto υλοποιήσεις στην I.MX.6 Quad.



(γ′) Κατανάλωση ενέργειας για τις βέλτιστες κατά Pareto υλοποιήσεις στην Myriad.



(δ′) Απόδοση ανά watt για τις βέλτιστες κατά Pareto υλοποιήσεις στην Myriad.

Σχήμα 3.24: Κατανάλωση ενέργειας και απόδοση ανά watt για τις βέλτιστες κατά Pareto υλοποιήσεις στις I.MX.6 και Myriad.

Κεφάλαιο 4

Αναλυτική περιγραφή και αξιολόγηση του μοντέλου client-server

Ο σχεδιασμός ταυτόχρονων δομών δεδομένων σε αρχιτεκτονικές που παρέχουν περιορισμένη υποστήριξη σε στοιχεία συγχρονισμού είναι ενδιαφέρουσα πρόκληση. Οι συνήθεις υλοποιήσεις που βασίζονται σε κλειδώματα παρουσιάζουν γνωστά προβλήματα, όπως περιορισμένο scalability και υψηλή κατανάλωση ενέργειας. Στο παρόν κεφάλαιο παρουσιάζουμε ένα απλό και πρακτικό μοντέλο συγχρονισμού των προσβάσεων σε ταυτόχρονες δομές δεδομένων βασισμένο σε ανταλλαγή μηνυμάτων μεταξύ των πυρήνων, ο οποίος είναι αποτελεσματικός σε αρχιτεκτονικές με περιορισμένη υποστήριξη σε στοιχεία συγχρονισμού και σε ταυτόχρονες δομές δεδομένων που παρέχουν χαμηλό επίπεδο παραλληλισμού (π.χ. ουρές, στοίβες). Το συγκεκριμένο μοντέλο εφαρμόστηκε σε μία ενσωματωμένη πλατφόρμα χαμηλής κατανάλωσης ενέργειας και παρατηρήθηκε πως ξεπερνά σε αρκετές περιπτώσεις σε απόδοση και κατανάλωση ενέργειας τις αντίστοιχες υλοποιήσεις που είναι βασισμένες σε κλειδώματα.

4.1 Εισαγωγή

Η υλοποίηση ταυτόχρονων δεδομένων είναι ένα δυσεπίλυτο πρόβλημα, ιδιαίτερα σε συστήματα με περιορισμένους πόρους και ισχυρούς σχεδιαστικούς περιορισμούς. Η απόδοση των δομών δεδομένων στις οποίες έχουν πρόσβαση περισσότεροι από ένας πυρήνες συγχρόνως, συχνά έχει σημαντική επίδραση στην απόδοση ολόκληρου του συστήματος. Η κατανάλωση ενέργειας είναι ένας ακόμη σημαντικός σχεδιαστικός περιορισμός που σε πολλές περιπτώσεις εξαρτάται σε σημαντικό βαθμό από τον σχεδιασμό των ταυτόχρονων δομών δεδομένων που χρησιμοποιεί η εφαρμογή. Παραδείγματα εφαρμογών που βασίζονται σε ταυτόχρονες δομές δεδομένων είναι οι βάσεις δεδομένων, οι αλγόριθμοι work-stealing, επεξεργασίας ροών δεδομένων, κ.α.. Εκτενής έρευνα στην συγκεκριμένη περιοχή δείχνει πως οι τυπικές μέθοδοι υλοποίησης ταυτόχρονων δομών δεδομένων βασισμένων σε κλειδώματα παρουσιάζουν χαμηλή απόδοση όσο αφορά το scalability [55]και την κατανάλωση ενέργειας [43].

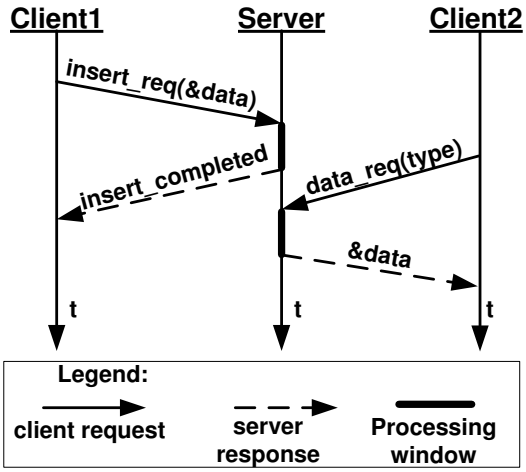
Στη συγκεκριμένη εργασία παρουσιάζουμε ένα μοντέλο συγχρονισμού το οποίο

μπορεί να εφαρμοστεί σε ταυτόχρονες δομές δεδομένων με χαμηλό παραλληλισμό (π.χ. στοίβες, ουρές) που εκτελούνται σε συστήματα που παρέχουν δυνατότητες ανταλλαγής μηνυμάτων μεταξύ των πυρήνων. Η κεντρική ιδέα είναι ότι ένα απλά σχεδιασμένο πρωτόκολλο που θα συγχρονίζει τις προσβάσεις στην δομή δεδομένων από τους πυρήνες και θα βασίζεται σε ανταλλαγή μηνυμάτων μεταξύ τους, μπορεί να παράσχει την ίδια ή υψηλότερη απόδοση σε σχέση με τον τυπικό συγχρονισμό βασισμένο σε κλειδώματα. Στο συγκεκριμένο μοντέλο η ταυτόχρονη δομή δεδομένων διαχειρίζεται από έναν πυρήνα ο οποίος εκτελεί όλες τις λειτουργίες. Κάθε άλλος πυρήνας που χρειάζεται να αποκτήσει πρόσβαση στη δομή για να πραγματοποιήσει μια λειτουργία (π.χ. insert, remove) στέλνει μία αίτηση στον συγκεκριμένο πυρήνα, μαζί με τις απαραίτητες πληροφορίες για την ολοκλήρωση της λειτουργίας. Το μοντέλο αυτό εφαρμόστηκε στην ενσωματωμένη πλατφόρμα Myriad, η οποία παρέχει τα απαραίτητα χαρακτηριστικά για την εφαρμογή του [6]

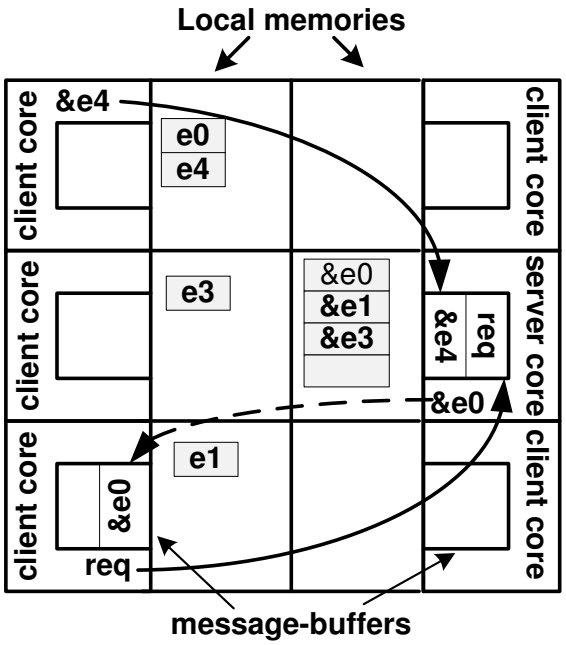
Η ιδέα του μοντέλου client-server για τον συγχρονισμό των προσβάσεων σε ταυτόχρονες δομές δεδομένων προέρχεται από τον αλγόριθμο Remote Core Locking (RCL) [57]. Ο αλγόριθμος RCL έχει σχεδιαστεί για αρχιτεκτονικές HPC, με λειτουργικό σύστημα βασισμένο σε Linux και υποστήριξη POSIX. Σχετικά με την υλοποίηση ταυτόχρονων δομών δεδομένων σε αρχιτεκτονικές με περιορισμένους πόρους έχουν προταθεί διάφορες λύσεις στο παρελθόν. Το C-Lock είναι μία προσπάθεια για τον συνδυασμό των πλεονεκτημάτων των κλειδωμάτων και της transactional memory [58]. Επίσης, απενεργοποιώντας το ρολόι των πυρήνων που είναι μπλοκαρισμένοι μειώνεται η κατανάλωση ενέργειας. Το Synchronization Operation Buffer είναι μία αρχιτεκτονική που στοχεύει στον περιορισμό του polling [59]. Η τεχνική Speculative Lock Elision επιτρέπει την ταυτόχρονη εκτέλεση critical sections της εφαρμογής στα οποία δεν υπάρχουν συγκρούσεις (conflicts) [60]. Τέλος, η Embedded Transactional Memory (Embedded TM) προσπαθεί να συμβιβάσει την απλότητα του σχεδιασμού με την χαμηλή κατανάλωση που απαιτείται στα ενσωματωμένη συστήματα [61]. Η συγκεκριμένη εργασία στοχεύει σε αρχιτεκτονικές με διαφορετικά χαρακτηριστικά από τα παραπάνω. Πιο συγκεκριμένα, αφορά αρχιτεκτονικές χωρίς υποστήριξη atomic primitives, POSIX ή transactional memory, ενώ παρέχει το κατάλληλο interface για επικοινωνία μεταξύ των πυρήνων.

4.2 Περιγραφή του μοντέλου client-server

Η ανάγκη για υλοποίηση ταυτόχρονων δομών δεδομένων σε συστήματα με περιορισμένους πόρους και υποστήριξη σε στοιχεία συγχρονισμού οδηγεί στην αναζήτηση εναλλακτικών μεθόδων σχεδιασμού. Η αποτελεσματική κατανομή των δεδομένων στη μνήμη, η αξιοποίηση του παραλληλισμού που παρέχει η δομή δεδομένων, το υψηλό throughput και η χαμηλή κατανάλωση ενέργειας, είναι χαρακτηριστικά που καθορίζουν την αποτελεσματικότητα μίας ταυτόχρονης δομής δεδομένων.



Σχήμα 4.1: Πρωτόκολλο του μοντέλου client-server.



Σχήμα 4.2: Υλοποίηση του μοντέλου client-server στη Myriad.

Το προτεινόμενο μοντέλο παρουσιάζεται στο Σχήμα 4.1. Βασίζεται στην ιδέα ότι ένας πυρήνας έχει τον ρόλο του server και είναι ο μοναδικός που έχει απευθείας πρόσβαση στη δομή δεδομένων. Οι υπόλοιποι πυρήνες που χρειάζεται να αποκτήσουν πρόσβαση στη δομή για να ολοκληρώσουν κάποια λειτουργία ονομάζονται clients. Αντί για απευθείας πρόσβαση στο δομή, στέλνουν αίτημα στον server και περιμένουν, αν είναι απαραίτητο, την απάντησή του. Ο server, μόλις λάβει κάποιο αίτημα που αντιστοιχεί σε κάποια λειτουργία (π.χ. push, pop), πραγματοποιεί τη λειτουργία για λογαριασμό του client και στέλνει σε αυτόν μία απάντηση που δηλώνει την ολοκλήρωση της λειτουργίας. Η δομή περιέχει διευθύνσεις στοιχείων (αντί για τα ίδια

τα στοιχεία) και είναι αποθηκευμένη στην τοπική μνήμη του server, ώστε ο ίδιος να έχει αποδοτική πρόσβαση σε αυτήν. Οι clients και ο server επικοινωνούν μέσω καταχωρητών που χρησιμοποιούνται για ανταλλαγή μηνυμάτων. Για παράδειγμα, όπως φαίνεται στο Σχήμα 4.2, για να πραγματοποιηθεί ένα *insert*, ο client, αφού πρώτα δημιουργήσει το αντίστοιχο στοιχείο στην τοπική του μνήμη, γράφει τη διεύθυνση του στοιχείου στο message-buffer του server. Για την πραγματοποίηση της λειτουργίας *data_req*, ο client γράφει ένα συγκεκριμένο μήνυμα στο message-buffer του server. Στη συνέχεια, ο client περιμένει τον server να ολοκληρώσει τη λειτουργία και να γράψει τη διεύθυνση του κατάλληλου στοιχείου στο message-buffer του client.

Είναι προφανές ότι ο server σειριοποιεί όλες τις λειτουργίες, γεγονός που αναμένεται να έχει αρνητικό αντίκτυπο στην απόδοση της δομής. Παρόλα αυτά, το προτεινόμενο μοντέλο έχει έναν αριθμό από πλεονεκτήματα, τα οποία αντισταθμίζουν τον μειωμένο παραλληλισμό που παρέχει. Στο Σχήμα 4.2 παρατηρούμε ότι η ταυτόχρονη δομή δεδομένων είναι αποθηκευμένη στην τοπική μνήμη του server. Επίσης, κάθε πυρήνας αποθηκεύει τα στοιχεία του στην τοπική του μνήμη, ενώ στη δομή αποθηκεύονται μόνο οι διευθύνσεις των στοιχείων. Με τον τρόπο αυτό ελαχιστοποιούνται οι προσβάσεις των πυρήνων σε μη τοπικές μνήμες, τόσο όσο αφορά τον server, όσο και τους clients. Το γεγονός αυτό οδηγεί σε υψηλή απόδοση και χαμηλή κατανάλωση ενέργειας. Στην αντίστοιχη τυπική προσέγγιση βασισμένη σε κλειδιά, η δομή δεδομένων θα ήταν αποθηκευμένη στην τοπική μνήμη ενός από τους πυρήνες, με συνέπεια υψηλό αριθμό προσβάσεων σε μη τοπικές μνήμες.

Πέρα από το θέμα της διαχείρισης της μνήμης, η αυξημένη απόδοση του μοντέλου client-server επιτυγχάνεται και με τον αυξημένο παραλληλισμό των λειτουργιών που εισάγουν νέα στοιχεία στην ταυτόχρονη δομή. Αφού ένας πυρήνας δημιουργήσει ένα στοιχείο στην τοπική του μνήμη, γράφει τη διεύθυνση του στοιχείου στο message-buffer του server. Από το σημείο αυτό κι έπειτα, ο client δεν χρειάζεται να περιμένει κάποια απάντηση ή επιβεβαίωση από τον server (δηλ. το μήνυμα *insert_compete* στο Σχήμα 4.1), αλλά συνεχίζει την εκτέλεση του αλγορίθμου της εφαρμογής. Έτσι, σε αντίθεση με τις τυπικές υλοποιήσεις που είναι βασισμένες σε κλειδώματα, μία λειτουργία *insert* μπλοκάρεται από μία άλλη λειτουργία *insert*, μόνο αν το message-buffer του server είναι γεμάτο. Σε αντίθετη περίπτωση, κάθε client γράφει την διεύθυνση του στοιχείου στο message-buffer του server και συνεχίζει.

Τέλος, μία τρίτη παράμετρος που επηρεάζει την απόδοση του μοντέλου είναι η αποτελεσματικότητα της επικοινωνίας μεταξύ clients και server. Για παράδειγμα, η Myriad παρέχει hardware buffers για την υποστήριξη της επικοινωνίας βασισμένης σε μηνύματα μεταξύ των πυρήνων. Κάθε πυρήνας περιέχει ένα buffer με χωρητικότητα 4 θέσεων, όπου η κάθε μία είναι 64bits. Η πρόσβαση στα συγκεκριμένα buffers γίνεται με FIFO τρόπο. Δηλαδή ο πυρήνας στον οποίο ανήκει το buffer διαβάζει από το πρώτο στοιχείο του, ενώ οι υπόλοιποι πυρήνες γράφουν στην ουρά του.

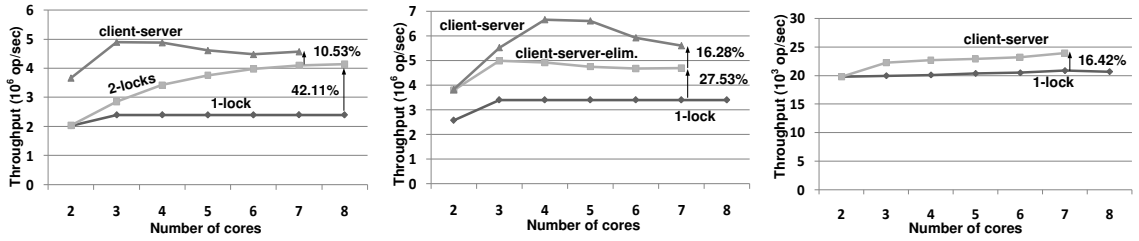
Η μειωμένη κατανάλωση ενέργειας του συγκεκριμένου μοντέλου, μπορεί να επι-

τευχθεί με τη χρήση χαρακτηριστικών που παρέχει η συγκεκριμένη αρχιτεκτονική. Ένα ενδιαφέρον χαρακτηριστικό των message-buffers της Myriad είναι το γεγονός ότι όταν ένας πυρήνας επιχειρεί να γράψει σε ένα γεμάτο buffer, τότε μπλοκάρει έως να ελευθερωθεί μία θέση. Στο διάστημα αυτό μεταβαίνει αυτόματα σε κατάσταση χαμηλής ισχύος. Τη στιγμή που θα ελευθερωθεί μία θέση, αυτόματα επανέρχεται σε κανονική κατάσταση για γράφει τα δεδομένα στην ουρά του buffer. Αντίστοιχα, όταν ένας πυρήνας προσπαθεί να διαβάσει από το buffer του και αυτό είναι άδειο, μπλοκάρει έως ότου γραφτεί τουλάχιστον ένα μήνυμα σε αυτό. Στη διάρκεια αυτή είναι σε κατάσταση χαμηλής ισχύος. Μόλις γραφτούν κάποια δεδομένα, ο πυρήνας επανέρχεται σε κανονική κατάσταση και διαβάζει τα δεδομένα. Το χαρακτηριστικό αυτό αυξάνει την αποτελεσματικότητα του μοντέλου στη συγκεκριμένη πλατφόρμα, όσο αφορά την κατανάλωση ενέργειας, ιδιαίτερα σε περιπτώσεις μεγάλης ή πολύ μικρής συμφόρησης. Σε περίπτωση μεγάλης συμφόρησης, οι πυρήνες βρίσκουν συχνά το message-buffer του server να είναι γεμάτο και επομένως βρίσκονται συχνά σε κατάσταση χαμηλής ισχύος. Αντίστοιχα, σε περίπτωση πολύ μικρής συμφόρησης, ο server προσπαθεί συχνά να διαβάσει από το message-buffer του, αλλά το βρίσκει άδειο, οπότε εισέρχεται σε κατάσταση χαμηλής κατανάλωσης.

Πέρα από το παραπάνω χαρακτηριστικό, η χαμηλή κατανάλωση ενέργειας επιτυγχάνεται και με την αποτελεσματική διαχείριση της μνήμης, όπως αναφέρθηκε νωρίτερα. Ο αυξημένος αριθμός προσβάσεων από κάθε πυρήνα στην τοπική του μνήμη, ευνοεί την χαμηλή κατανάλωση. Αντίθετα, οι τυπικές ταυτόχρονες δομών δεδομένων, που είναι βασισμένες σε κλειδώματα είναι γενικά μη αποτελεσματικές όσο αφορά την κατανάλωση ενέργειας, ειδικά αν τα κλειδώματα είναι υλοποιημένα ως spinlocks.

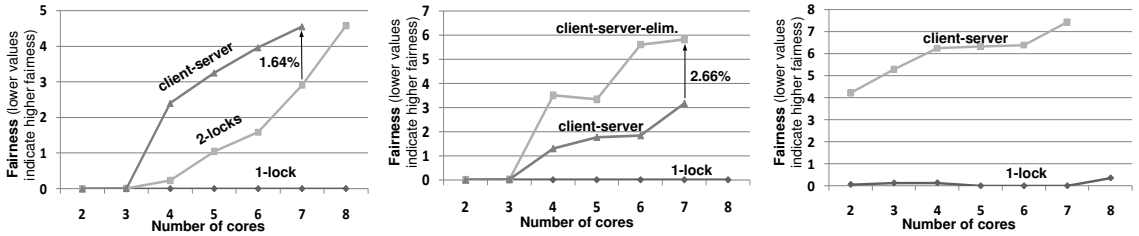
4.3 Πειραματικά αποτελέσματα

Το συγκεκριμένο μοντέλο εφαρμόστηκε σε μερικές από τις πιο γνωστές ταυτόχρονες δομές δεδομένων: ούρά (queue), στοίβα (stack) και ταξινομημένο σωρό (heap). Η αξιολόγησή του έγινε με χρήση synthetic benchmarks. Στην αξιολόγηση της στοίβας εξετάστηκε μία υλοποίηση client-server βασισμένη στην τεχνική "elimination" [46]. Μία λειτουργία push που ακολουθείται από μία λειτουργία pop "εξουδετερώνουν" η μία την άλλη. Με άλλα λόγια, αν αμέσως μετά τη διεύθυνση ενός στοιχείου που διάβασε ο server από το buffer του, ακολουθεί ένα αίτημα pop, τότε αντί να αποθηκευτεί η διεύθυνση στη στοίβα, γράφεται απευθείας στο buffer του client που ζήτησε το pop. Έτσι, αποφεύγεται το overhead της αποθήκευσης του συγκεκριμένου στοιχείου στη στοίβα και της εξαγωγής του αμέσως μετά. Πέρα από το throughput και την κατανάλωση ενέργειας, εξετάστηκε το fairness, ακολουθώντας μια προσέγγιση αντίστοιχη με το [43]. Τιμές κοντά στο μηδέν δείχνουν υψηλό fairness. Μεγαλύτερες τιμές δείχνουν πιθανό starvation. Η κατανάλωση ενέργειας μετρήθηκε με μεθόδους hardware instrumentation.



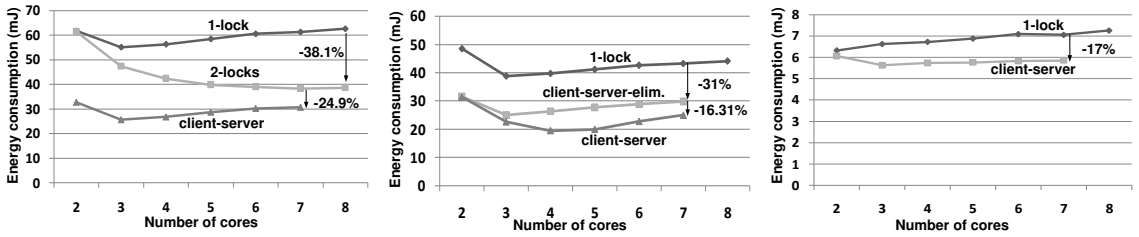
(α΄) Queue: Scalability **(β΄)** Stack: Scalability **(γ΄)** Heap: Scalability

Σχήμα 4.3: Αποτελέσματα Scalability



(α΄) Queue: Fairness **(β΄)** Stack: Fairness **(γ΄)** Heap: Fairness

Σχήμα 4.4: Αποτελέσματα fairness.

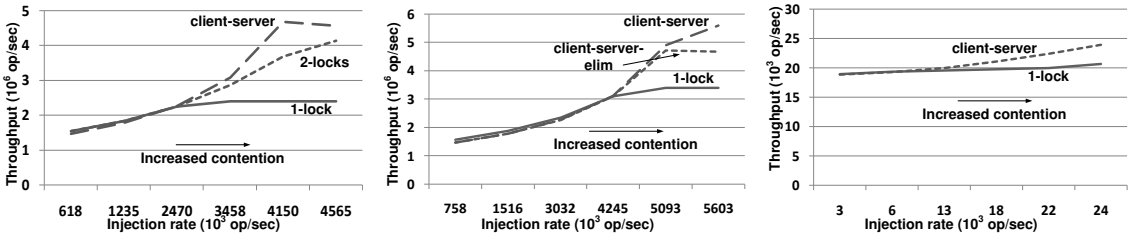


(α΄) Queue: Energy **(β΄)** Stack: Energy **(γ΄)** Heap: Energy

Σχήμα 4.5: Αποτελέσματα κατανάλωσης ενέργειας.

Σχετικά με τα χαρακτηριστικά της πλατφόρμας που χρησιμοποιήθηκε για την εφαρμογή του μοντέλου, η Myriad έχει 8 πυρήνες που λειτουργούν στα 180MHz. Η κοινή μνήμη είναι 1MB και χωρίζεται σε 8 τμήματα (”slices”), με πυρήνα να έχει 128KB τοπική μνήμη. Η συγκεκριμένη μνήμη δεν είναι cached. Κάθε υλοποίηση έχει αξιολογηθεί με βάση το throughput, την κατανάλωση ενέργειας και το fairness. Η κατανομή των λειτουργιών είναι 80% εισαγωγή δεδομένων (insert, push etc.) και 20% εξαγωγή (π.χ. delete, pop).

Τα διαγράμματα του Σχήματος 4.3 απεικονίζουν τα αποτελέσματα του scalability για την ουρά, τη στοίβα και τον σωρό. Παρατηρούμε ότι οι υλοποιήσεις που βασίζονται στο μοντέλο client-server παρέχουν υψηλότερο throughput σε όλες τις δομές. Στην περίπτωση του σωρού, η υλοποίηση client-server παρέχει 43.81% υψηλότερο throughput



(α΄) Queue: Perf. vs. Req. op./s. **(β΄)** Stack: Perf. vs. Req. op./s. **(γ΄)** Heap: Perf. vs. Req. op./s.

Σχήμα 4.6: Αποτελέσματα απόδοσης vs. operations requested per second.

σε σχέση με την υλοποίηση βασισμένη σε κλειδώματα για 7 πυρήνες (Σχήμα 4.3β΄). Είναι αξιοσημείωτο ότι παρόλο που στο μοντέλο client-server σειριοποιούνται όλες οι λειτουργίες, στο πείραμα της ουράς αποδίδει καλύτερα σε σχέση με την υλοποίηση με 2 κλειδώματα, που είναι αυτή που παρέχει τον υψηλότερο δυνατό παραλληλισμό. Όπως φαίνεται στο Σχήμα 4.3α΄, στο πείραμα με 7 πυρήνες, είναι αυξημένο κατά 10,53%. Μια δεύτερη παρατήρηση είναι το γεγονός ότι στο μοντέλο client-server, στα πειράματα της στοίβας και του σωρού, το throughput αυξάνει ως τους 4 πελάτες, ενώ στη συνέχεια σταδιακά μειώνεται. Τέλος, η υλοποίηση στοίβας με χρήση της τεχνικής ''elimination'', παρουσιάζει χαμηλότερο throughput σε σχέση με την απλή υλοποίηση client-server, αλλά υψηλότερο σε σχέση με την βασισμένη σε κλειδώματα, όπως φαίνεται στο Σχήμα 4.3γ΄. Η απλή υλοποίηση client-server παρουσιάζει 16,28% υψηλότερο throughput πείραμα με 7 πυρήνες.

Τα αποτελέσματα fairness απεικονίζονται στα διαγράμματα του Σχήματος 4.4. Η πρώτη παρατήρηση είναι ότι οι υλοποιήσεις βασισμένες σε κλειδώματα παρέχουν πολύ υψηλότερο fairness σε σχέση με τις υλοποιήσεις client-server. Αυτό είναι αναμενόμενο, δεδομένου ότι τα κλειδώματα στη Myriad είναι υλοποιημένα ως fair-locks με round robin scheduling. Έτσι, πετυχαίνεται πολύ υψηλό fairness, ακόμα και στα πειράματα με 8 πυρήνες. Από την άλλη πλευρά, υλοποιήσεις client-server παρέχουν 4,5% ως 7,5% χαμηλότερο fairness. Είναι αξιοσημείωτο ότι στο πείραμα της ουράς (Σχήμα 4.4α΄) το μοντέλο client-server παρουσιάζει μόλις 1,64% υψηλότερο unfairness σε σχέση με την υλοποίηση με 2 κλειδώματα. Τέλος, αξίζει να τονιστεί ότι τα συγκεκριμένα πειράματα είναι σε κατάσταση υψηλής συμφόρησης (μέγιστο δυνατό injection rate). Μικρότερη συμφόρηση θα είχε ως αποτέλεσμα υψηλότερο fairness για όλες τις υλοποιήσεις.

Τα διαγράμματα του Σχήματος 4.5 παρουσιάζουν την κατανάλωση ενέργειας για κάθε μία από τις υλοποιήσεις στις τρεις διαφορετικές δομές δεδομένων που εξετάζονται. Σε όλες τις περιπτώσεις οι υλοποιήσεις που βασίζονται στο προτεινόμενο μοντέλο καταναλώνουν λιγότερη ενέργεια. Στην ουρά, η υλοποίηση client-server απαιτεί 24,9% χαμηλότερη ενέργεια σε σχέση με την υλοποίηση βασισμένη σε 2 κλειδώματα. Η βασική αιτία είναι το γεγονός ότι καθώς η συμφόρηση αυξάνεται, αυξάνει ο χρόνος

κατά τον οποίο οι πυρήνες που έχουν τον ρόλο του client είναι σε κατάσταση χαμηλής ισχύος, λόγω του ότι το message-buffer του server είναι γεμάτο. Τέλος, γενικά παρατηρούμε ότι όσο περισσότερους πυρήνες χρησιμοποιούμε, τόσο αυξάνει η κατανάλωση ενέργειας. Μοναδική εξαίρεση είναι η υλοποίηση ουράς με 2 κλειδώματα (Σχήμα 4.5α΄), όπου η σταδιακή αύξηση των πυρήνων οδηγεί σε μεγάλη μείωση του χρόνου εκτέλεσης του πειράματος, με συνέπεια την μικρότερη κατανάλωση ενέργειας καθώς ο αριθμός των clients αυξάνει.

Στην τελευταία στήλη του Σχήματος 4.6 βρίσκονται τα διαγράμματα που αφορούν τη σχέση του throughput με το injection rate. Με τον όρο injection rate αναφερόμαστε στον ρυθμό με τον οποίο οι πυρήνες ζητούν πρόσβαση στη δομή δεδομένων. Παρατηρούμε ότι σε όλες τις δομές δεδομένων, καθώς η συμφόρηση αυξάνει, στις υλοποιήσεις με ένα κλείδωμα υπάρχει κορεσμός σε χαμηλότερο injection rate σε σχέση με τις υπόλοιπες υλοποιήσεις. Αντίθετα, οι υλοποιήσεις βασισμένες στο μοντέλο client-server παρέχουν υψηλό throughput σε κατάσταση υψηλότερης συμφόρηση.

Συνοψίζοντας, παρατηρούμε πως το μοντέλο client-server παρέχει υψηλή απόδοση σε ενσωματωμένα συστήματα που περιέχουν υποστήριξη επικοινωνίας μέσω μηνυμάτων μεταξύ των πυρήνων. Σε περίπτωση που το πρωτόκολλο επικοινωνίας έχει χαρακτηριστικά που ευνοούν την χαμηλή κατανάλωση, τότε οι υλοποιήσεις βασισμένες στο προτεινόμενο μοντέλο μπορούν να παρέχουν χαμηλή κατανάλωση ενέργειας. Τα πειραματικά αποτελέσματα δείχνουν ότι οι υλοποιήσεις client-server θα πρέπει να αξιολογούνται παράλληλα με τις παραδοσιακές υλοποιήσεις βασισμένες σε κλειδώματα, σε αρχιτεκτονικές που παρέχουν επικοινωνία μέσω μηνυμάτων και όταν σχεδιάζονται δομές δεδομένων που παρέχουν περιορισμένο παραλληλισμό.

Κεφάλαιο 5

Παραμετροποίηση της υλοποίησης του streaming aggregation operator σε ενσωματωμένα συστήματα

Το streaming aggregation είναι μια θεμελιώδης λειτουργία στο χώρο της επεξεργασίας ροών δεδομένων και η υλοποίησή της έχει διάφορες προκλήσεις. Παραδοσιακά, η επεξεργασία δεδομένων πραγματοποιείται από συστήματα υψηλής απόδοσης. Παρόλα αυτά, σήμερα, υπάρχει η τάση της υλοποίησης operators ροών δεδομένων σε συσκευές χαμηλής ισχύος, εξαιτίας του γεγονότος ότι συχνά παρέχουν αυξημένη απόδοση / ισχύ (performance per watt) σε σχέση με τα συστήματα υψηλής απόδοσης. Στη συγκεκριμένη εργασία προτείνεται μία μεθοδολογία για την παραμετροποίηση του streaming aggregation που υλοποιείται σε σύγχρονα ενσωματωμένα συστήματα χαμηλής ισχύος. Η μεθοδολογία βασίζεται στην εξερεύνηση του χώρου σχεδιασμού και παρέχει ένα σύνολο από υλοποιήσεις του συγκεκριμένου operator που μπορούν να χρησιμοποιηθούν από τους προγραμματιστές για την πραγματοποίηση trade-offs μεταξύ του throughput, του latency και της κατανάλωσης μνήμης και ενέργειας. Επιπλέον, στο πλαίσιο της παρούσας διατριβής, συγκρίνουμε τις υλοποιήσεις σε ενσωματωμένα συστήματα με τις αντίστοιχες υλοποιήσεις σε HPC και GPGPU με κριτήριο την απόδοση ανά watt. Τα αποτελέσματα δείχνουν ότι οι υλοποιήσεις σε ενσωματωμένα συστήματα χαμηλής ισχύος παρέχουν έως 54 και 14 φορές υψηλότερη απόδοση ανά watt σε σχέση με τις αντίστοιχες υλοποιήσεις σε Intel Xeon και Radeon HD 6450, αντίστοιχα.

5.1 Εισαγωγή

Η αποτελεσματική διαχείριση και επεξεργασία σε πραγματικό χρόνο ροών δεδομένων που παράγονται από σύγχρονα διασυνδεδεμένα συστήματα είναι ενδιαφέρουσα πρόκληση. Στο παρελθόν, η επεξεργασία ροών δεδομένων με χαμηλό latency αφορούσε κυρίως συστήματα σε χρηματιστηριακά και οικονομικά ιδρύματα. Σήμερα, η επεξεργασία εκατομμυρίων γεγονότων (events) όπως τηλεφωνικές κλήσεις, γραπτά μηνύματα, μετακινήσεων δεδομένων σε ένα δίκτυο και η παραγωγή χρήσιμης πληροφορίας από αυτά είναι σημαντική για να υπάρξει εγγύηση υψηλής Ποιότητας Υπηρεσιών (Quality of Service). Οι τυπικές εφαρμογές επεξεργασίας ροών δεδομένων βασίζονται σε Stream Processing Engines (SPEs) που υλοποιούνται σε υπολογιστικά

συστήματα υψηλής απόδοσης.

Σήμερα, τα ψηφιακά δεδομένα προέρχονται από πολλές πηγές, π.χ. αισθητήρες, κάμερες φορητών συσκευών και wearable συσκευές. Στον κόσμο του Internet of Things υπάρχει ανάγκη σε πολλές περιπτώσεις για επεξεργασία δεδομένων on-the-fly, ώστε να εντοπιστούν γεγονότα σε πολύ μικρό χρονικό διάστημα από τη στιγμή που παράγονται τα δεδομένα. Η συγκεκριμένη προσέγγιση έρχεται σε αντίθεση με την πρακτική όπου τα δεδομένα πρώτα αποθηκεύονται και έπειτα ακολουθεί η επεξεργασία τους (π.χ. σε συστήματα βάσεων δεδομένων). Τα δεδομένα αυτά μπορεί να παράγονται ως ροές από διάφορες πηγές σε πραγματικό χρόνο. Στη συνέχεια, θα πρέπει να συλλεχθούν και μετά από σύντομη επεξεργασία να αναλυθούν όσο ταχύτερα γίνεται, καθώς νέα δεδομένα παράγονται συνεχώς.

Η υλοποίηση των συστημάτων αυτών σε ενσωματωμένα συστήματα έχει δύο βασικά πλεονεκτήματα. Αφενός τα σύγχρονα ενσωματωμένα συστήματα παρέχουν ολοένα και μεγαλύτερη υπολογιστική ισχύ. Έτσι, αποδοτικές υλοποιήσεις εφαρμογών επεξεργασίας ροών δεδομένων μπορούν να παράσχουν χαμηλό latency και υψηλό throughput. Αφ' ετέρου, πολλές σύγχρονες αρχιτεκτονικές λειτουργούν σε εξαιρετικά χαμηλή ισχύ. Έτσι, όταν βασικό κριτήριο μίας υλοποίησης είναι η απόδοση ανά watt, τότε οι αρχιτεκτονικές των ενσωματωμένων συστημάτων είναι σε πολλές περιπτώσεις προτιμητέες [62][63][64][65]. Ως τώρα, διάφορα σενάρια streaming aggregation operators έχουν υλοποιηθεί και αξιολογηθεί σε διάφορες αρχιτεκτονικές, όπως GPUs, Nehalem και Cell processor [66].

Το streaming aggregation είναι είναι ένας operator που χρησιμοποιείται ευρύτατα στο χώρο της επεξεργασίας δεδομένων. Χρησιμοποιείται για να παραχθούν πληροφορίες από ροές δεδομένων μέσω της άθροισης των τιμών συγκεκριμένων ιδιοτήτων ενός υποσυνόλου πακέτων των δεδομένων που απαρτίζουν μία ή περισσότερες ροές. Τα "πακέτα" στον χώρο της εξεργασίας ροών δεδομένων είναι ουσιαστικά σε μορφή tuples που περιέχουν κάποια ταυτότητα (id) και δεδομένα. Τα δεδομένα αποτελούνται από έναν αριθμό από πεδία, που αντιστοιχούν σε ιδιότητες (attributes) των δεδομένων. Η διαδικασία του streaming aggregation γίνεται με την ομαδοποίηση και τον υπολογισμό του αθροίσματος σε πραγματικό χρόνο, των τιμών κάποιων χαρακτηριστικών των tuples. Παραδείγματα χρήσης του operator streaming aggregation συναντώνται σε υπολογιστικά συστήματα χρηματιστηριακών ιδρυμάτων (π.χ. ο υπολογισμός της μέσης τιμής κάθε μετοχής σε συγκεκριμένου μεγέθους χρονικά διαστήματα σε πραγματικό χρόνο), ο συνεχής έλεγχος δικτυακών συστημάτων (π.χ. η μέση κίνηση στο δίκτυο σε πραγματικό χρόνο) κ.ο.κ. Στα παραδείγματα αυτά, ο aggregation operator χρησιμοποιείται σε συνδυασμό με άλλους operators για την εξαγωγή χρήσιμων αποτελεσμάτων από τις ροές δεδομένων, συνεχώς και σε πραγματικό χρόνο.

Τα σύγχρονα ενσωματωμένα συστήματα παρέχουν διαφορετικά χαρακτηριστικά (π.χ. ιεραρχίες μνήμης, διαφορετικές επιλογές στην μεταφορά δεδομένων), ανάλογα

με το είδος των εφαρμογών στο οποίο απευθύνονται. Η βαρύτητα καθενός από αυτά τα χαρακτηριστικά στην απόδοση και στην κατανάλωση ενέργειας του συστήματος, όταν εκτελείται κάποια συγκεκριμένη εφαρμογή είναι συχνά δύσκολο να προβλεφθεί κατά τη διάρκεια του σχεδιασμού. Ακόμη κι αν είναι ασφαλές σε κάποιες περιπτώσεις να υποθέσουμε πως η χρησιμοποίηση ενός συγκεκριμένου χαρακτηριστικού της αρχιτεκτονικής θα βελτιώσει ή θα χειροτερεύσει την τιμή μιας συγκεκριμένης μετρικής σε μια ορισμένη περίπτωση, είναι δύσκολο να ποσοτικοποιηθεί αυτή η επίδραση χωρίς εκτέλεση και αξιολόγηση. Το πρόβλημα γίνεται ακόμη πιο πολύπλοκο σε περίπτωση που ο στόχος είναι η βελτιστοποίηση περισσότερων από μία μετρικές συγχρόνως. Ένα παρόμοιο πρόβλημα είναι το porting μίας εφαρμογής από μία αρχιτεκτονική με συγκεκριμένα χαρακτηριστικά, σε μία άλλη με διαφορετικά χαρακτηριστικά. Στην περίπτωση αυτή, υπάρχει η ανάγκη παραμετροποίησης της εφαρμογής στο νέο σύστημα με διαφορετικό τρόπο, ώστε να επιτευχθεί υψηλή απόδοση ταυτόχρονα με χαμηλή κατανάλωση ενέργειας. Η τυπική λύση που ακολουθείται από τους προγραμματιστές είναι η προσπάθεια βελτιστοποίησης της εφαρμογής χειροκίνητα, με αυθαίρετο τρόπο, η οποία είναι συνήθως χρονοβόρα διαδικασία που οδηγεί συχνά σε μη βέλτιστα αποτελέσματα. Επομένως, υπάρχει ανάγκη για μία συστηματική διαδικασία βελτιστοποίησης: Ο εντοπισμός των διαφόρων σχεδιαστικών επιλογών της εφαρμογής και της αρχιτεκτονικής και η εξερεύνηση του χώρου σχεδιασμού με συστηματικό τρόπο μπορεί να παράσχει υλοποιήσεις που ικανοποιούν τα κριτήρια σχεδιασμού.

Στο κεφάλαιο αυτό περιγράφεται μία ημιαυτόματη βήμα-προς-βήμα μεθοδολογία για την παραμετροποίηση υλοποιήσεων του streaming aggregation operator που εκτελείται σε ενσωματωμένα συστήματα. Η μεθοδολογία βασίζεται i) στην εντοπισμό των σχεδιαστικών παραμέτρων του streaming aggregation operator που επηρεάζουν τις μετρικές που χρησιμοποιούνται για την αξιολόγηση των υλοποιήσεων και ii) στον εντοπισμό των σχεδιαστικών παραμέτρων της ενσωματωμένης πλατφόρμας που επηρεάζουν τις μετρικές κατά την εκτέλεση του streaming aggregator. Το σύνολο αυτών των παραμέτρων σχηματίζει τον χώρο των σχεδιαστικών επιλογών. Η μεθοδολογία παρέχει ένα σύνολο από υλοποιήσεις του streaming aggregation στην συγκεκριμένη αρχιτεκτονική. Για κάθε υλοποίηση, μία ή περισσότερες σχεδιαστικές παράμετροι έχουν διαφορετικές τιμές σε σχέση με τις υπόλοιπες υλοποιήσεις. Με άλλα λόγια, κάθε υλοποίηση του streaming aggregator είναι διαφορετικά παραμετροποιημένη σε σχέση με τις υπόλοιπες, με συνέπεια να παρέχει διαφορετικά αποτελέσματα, όσο αφορά τις μετρικές που χρησιμοποιούνται για την αξιολόγησή τους. Οι προγραμματιστές μπορούν να πραγματοποιήσουν trade-offs μεταξύ διαφορετικών μετρικών, επιλέγοντας διαφορετικές παραμετροποιημένες υλοποιήσεις. Έτσι, αντί για βελτιστοποίηση με αυθαίρετο τρόπο που πιθανόν να παράσχει μη-βέλτιστες υλοποιήσεις, η προτεινόμενη προσέγγιση παρέχει ένα συστηματικό τρόπο εξερεύνησης του χώρου σχεδιασμού.

Συνοψίζοντας, στις ακόλουθες ενότητες θα παρουσιαστεί η προτεινόμενη μεθο-

δολογία παραμετροποίησης του streaming aggregator σε ενσωματωμένα συστήματα. Θα δειχθεί ότι ο aggregation operator υλοποιημένος σε ενσωματωμένα συστήματα παρέχει πολύ υψηλότερη απόδοση ανά watt σε σχέση με τις αντίστοιχες υλοποιήσεις σε HPC και GPU (GPGPU). Τέλος, βασισμένοι στα πειραματικά αποτελέσματα εξάγουμε ενδιαφέροντα συμπεράσματα σχετικά με πώς κάθε μία από τις σχεδιαστικές παραμέτρους της εφαρμογής και της αρχιτεκτονικής επιδρούν σε κάθε μία από τις μετρικές. Η μεθοδολογία υλοποιήθηκε σε δύο streaming aggregation σενάρια που υλοποιήθηκαν σε 4 διαφορετικές αρχιτεκτονικές: Myriad1, Myriad2, Freecale I.MX.6 Quad και Exynos 5 octa. Οι μετρικές βάση των οποίων αξιολογήθηκαν οι υλοποιήσεις είναι οι throughput, μέγεθος απαιτούμενης μνήμης, latency, κατανάλωση ενέργειας και scalability.

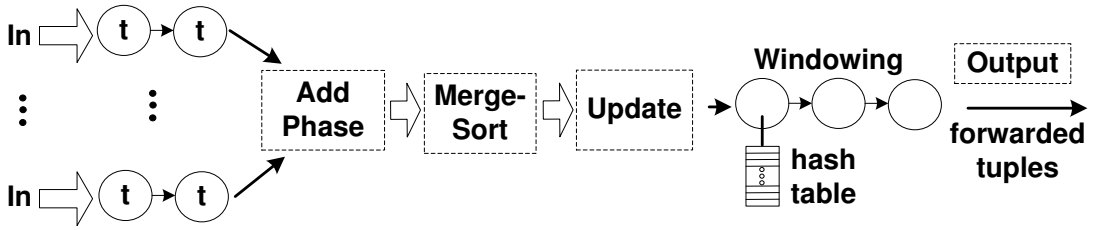
5.2 Επεξεργασία ροών δεδομένων στη βιβλιογραφία

Υπάρχει εκτενής βιβλιογραφία σχετικά με την επεξεργασία ροών δεδομένων σε συστήματα υψηλής απόδοσης. Κάποιες εργασίες αφορούν την παραλληλοποίηση υλοποιήσεων [67], [68], [69]. Περιγράφουν τον τρόπο με τον οποίο κάποιοι operators ανατίθενται σε διαφορετικά partitions της εφαρμογής με στόχο τον αυξημένο παραλληλισμό. Στην εργασία [70] περιγράφεται ένας άλλος τρόπος για την αύξηση του παραλληλισμού: Προτείνεται η χρήση lock-free δομών δεδομένων για την υλοποίηση του streaming aggregation operator σε πολυπύρηνες αρχιτεκτονικές. Η υλοποίηση σε 6-πύρηνο Xeon επεξεργαστή έδειξε αυξημένο scalability.

Κάποιες από τις πιο γνωστές Stream Processing Engines (SPE) είναι οι Aurora και Borealis [71]. Πέρα από την υλοποίηση των operators επεξεργασίας ροών δεδομένων σε συστήματα υψηλής απόδοσης, υπάρχουν εργασίες που αφορούν την υλοποίησή τους σε άλλα συστήματα. Για παράδειγμα, η υλοποίηση σε ετερογενείς αρχιτεκτονικές (CPU + GPU) περιγράφεται στο [69]. Στο [66] αξιολογείται ο streaming aggregation operator σε Core 2 Quad, Nvidia GTX GPU και στην Cell Broadband Engine. Στο κεφάλαιο αυτό θα παρουσιάσουμε υλοποιήσεις του streaming aggregation operator σε διαφορετικές αρχιτεκτονικές, καθώς και ένα πιο πολύπλοκα σενάριο, όπου η επεξεργασία των tuples βασίζεται σε timestamps.

Η χρήση ενσωματωμένων επεξεργαστών σε servers με στόχο την χαμηλή ισχύ έχει προταθεί στο παρελθόν σε διάφορες εργασίες, ενώ υπάρχουν εμπορικές υλοποιήσεις [62]. Πιο συγκεκριμένα, πολλές εργασίες αφορούν την χρήση ARM επεξεργαστών χαμηλής ισχύος σε servers [63] ή την υλοποίηση clusters χαμηλής ισχύος βασισμένων σε επεξεργαστές προορισμένους για φορητές συσκευές [64].

Στο χώρο της επεξεργασίας ροών δεδομένων σε ενσωματωμένα συστήματα, πολλές εργασίες αφορούν τον σχεδιασμό μεταγλωττιστών με στόχο την αύξηση του παραλληλισμού και την αποδοτική αξιοποίηση των πόρων του συστήματος [72]. Μία γλώσσα προγραμματισμού για επεξεργασία ροών δεδομένων που υλοποιούνται σε



Σχήμα 5.1: Οι τέσσερις φάσεις του time-based streaming aggregation.

ενσωματωμένα συστήματα έχει προταθεί στο [73]. Οι συγκεκριμένες εργασίες είναι συμπληρωματικές στην παρούσα προσέγγιση: Τα συμπεράσματα της συγκεκριμένης μελέτης μπορούν να βοηθήσουν στην κατεύθυνση σχεδιασμού αποδοτικών μεταγλωττιστών ή frameworks για υλοποιήσεις συστημάτων επεξεργασίας ροών δεδομένων.

Η εξερεύνηση των σχεδιαστικών επιλογών είναι ένας ακόμη χώρος που σχετίζεται με τη συγκεκριμένη προσέγγιση. Έχουν προταθεί μεθοδολογίες συστηματικής εξερεύνησης του χώρου σχεδιασμού για την παραμετροποίηση αρχιτεκτονικών [49], για την παραμετροποίηση δυναμικών δομών δεδομένων εφαρμογών [9] και δυναμικών διαχειριστών μνήμης [10]. Οι μεθοδολογίες αυτές είναι συμπληρωματικές της προτεινόμενης. Με την εφαρμογή μεθοδολογιών που βελτιστοποιούν τις δομές δεδομένων ή των διαχειριστή μνήμης του συστήματος, η απόδοση ενός συστήματος που εκτελεί streaming operators μπορεί να αυξηθεί και αντίστοιχα, η κατανάλωση ενέργειας να μειωθεί.

5.3 Περιγραφή του operator Streaming Aggregation

Στην ενότητα αυτή περιγράφεται ο streaming aggregation operator και αναλύονται τα σχεδιαστικά ζητήματα που αφορούν την υλοποίηση του operator σε ενσωματωμένα συστήματα.

Το aggregation είναι ένας operator που χρησιμοποιείται ευρύτατα στον χώρο της επεξεργασίας ροών δεδομένων. Χρησιμοποιείται για να ομαδοποιήσει έναν αριθμό από εισερχόμενα tuples (inbound tuples) και να υπολογίσει το άθροισμα κάποιων ιδιοτήτων τους. Λειτουργεί παρόμοια με την εντολή *group-by* της SQL. Στα πλαίσια της συγκεκριμένης μελέτης, περιγράφονται 2 σενάρια aggregation: *Multiway time-based με sliding windows* και *count-based with με tumbling windows*.

5.3.1 Myltiway time-based streaming aggregation

Στο multiway aggregation, πολλαπλές ροές δεδομένων από εισερχόμενα tuples, τα οποία είναι αποθηκευμένα σε ουρές, μετατρέπονται σε μία ροή μέσω του operator *merge*. Τα tuples ταξινομούνται με βάση της τιμής της ιδιότητας timestamp. Αποτελείται από τις εξής 4 φάσεις που σχηματίζουν ένα pipeline και απεικονίζονται στο

Σχήμα 5.1:

1. **Add**: Εισερχόμενα tuples αποσπώνται από την ουρά κάθε ροής δεδομένων.
2. **Merge-Sort**: Τα tuples ταξινομούνται με χρήση του merge operator.
3. **Update**: Κάθε tuple τοποθετείται στο/στα παράθυρα στα οποία ανήκει.
4. **Output**: Υπολογίζεται η aggregated τιμή για κάθε έτοιμο παράθυρο και προωθείται.

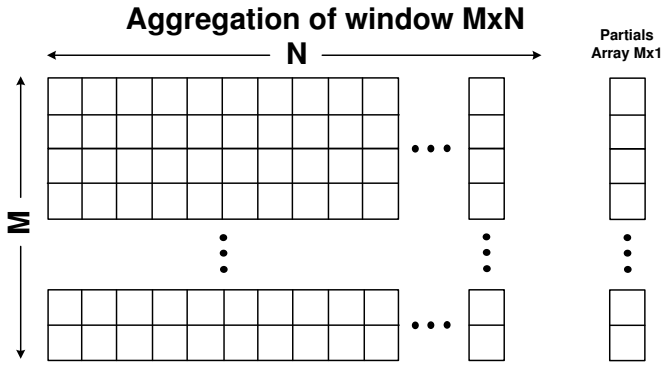
Κατά τη διάρκεια της φάσης *Add*, tuples από κάθε εισερχόμενη ροή προωθούνται στη φάση *Merge-Sort*. Δεδομένου ότι είναι αποθηκευμένα σε ουρές, προωθούνται με first-in-first-out (FIFO).

Κατά τη φάση *Merge-Sort* συνδυάζονται ροές δεδομένων, των οποίων τα tuples είναι ταξινομημένα ως προς μια συγκεκριμένη ιδιότητα, σε μία μοναδική ροή στην οποία τα tuples είναι επίσης ταξινομημένα ως προς την ίδια ιδιότητα. Στο πλαίσιο της συγκεκριμένης μελέτης τα tuples ταξινομούνται βάση του timestamp. Οι λειτουργίες *Merge* και *Sort* γίνονται ταυτόχρονα και μοιράζονται τους ίδιους πόρους, δηλαδή τα εισερχόμενα tuples που έχουν προωθηθεί από την φάση *Add* και μπορούν να θεωρηθούν ως μία θεμελιώδης λειτουργία. Ένα tuple είναι έτοιμο προς επεξεργασία και μπορεί να αφαιρεθεί από την αντίστοιχη ουρά, αν έχει ληφθεί τουλάχιστον ένα tuple με ίσο ή μεγαλύτερο timestamp από όλες τις υπόλοιπες ουρές. Ο κανόνας αυτός εξασφαλίζει την ντετερμινιστική επεξεργασία των εισερχόμενων tuples.

Στη φάση του *Update* κάθε tuple τοποθετείται στα κατάλληλα παράθυρα, κάθε ένα από τα οποία αντιστοιχεί σε κάποιο συγκεκριμένο χρονικό διάστημα. Στα πλαίσια αυτής της μελέτης γίνεται χρήση sliding windows, τα οποία έχουν δύο ιδιότητες: *size* και *advance*. Για παράδειγμα, ένα παράθυρο με size 5 και *advance* 2 χρονικές μονάδες καλύπτει τα χρονικά διαστήματα [0, 5), [2, 7), [4, 9), κλπ.. Ένα tuple με timestamp 3 θα τοποθετηθεί στα παράθυρα [0, 5) και [2, 7).

Στη φάση του *Output* η aggregated τιμή υπολογίζεται για όλα τα παράθυρα στα οποία δεν αναμένεται να τοποθετηθούν νέα tuples (ολοκληρωμένα παράθυρα). Ο ντετερμινιστικός τρόπος επεξεργασίας των tuples που πραγματοποιείται στις προηγούμενες φάσεις εξασφαλίζει ότι η aggregated τιμή θα υπολογιστεί μόνο για τα ολοκληρωμένα παράθυρα. Μετά τον υπολογισμό της τιμής, δημιουργείται ένα νέο tuple στο οποίο η τιμή αποθηκεύεται και το tuple προωθείται στην έξοδο του pipeline.

Το multiway time-based streaming aggregation παρέχει παραλληλισμό pipeline, τον οποίο μπορούμε να τον εκμεταλλευτούμε αναθέτοντας κάθε φάση σε ένα διαφορετικό επεξεργαστικό στοιχείο (processing element - PE). Η απόδοση όμως δεν εξαρτάται μόνο από την αξιοποίηση του παραλληλισμού ή την παρεχόμενη επεξεργαστική ισχύ, αλλά και από από την αποδοτικότητα της μεταφοράς δεδομένων μεταξύ των διαφορετικών φάσεων. Τα ταξινομημένα tuples της φάσης *Merge-Sort* χρησιμοποιούνται από



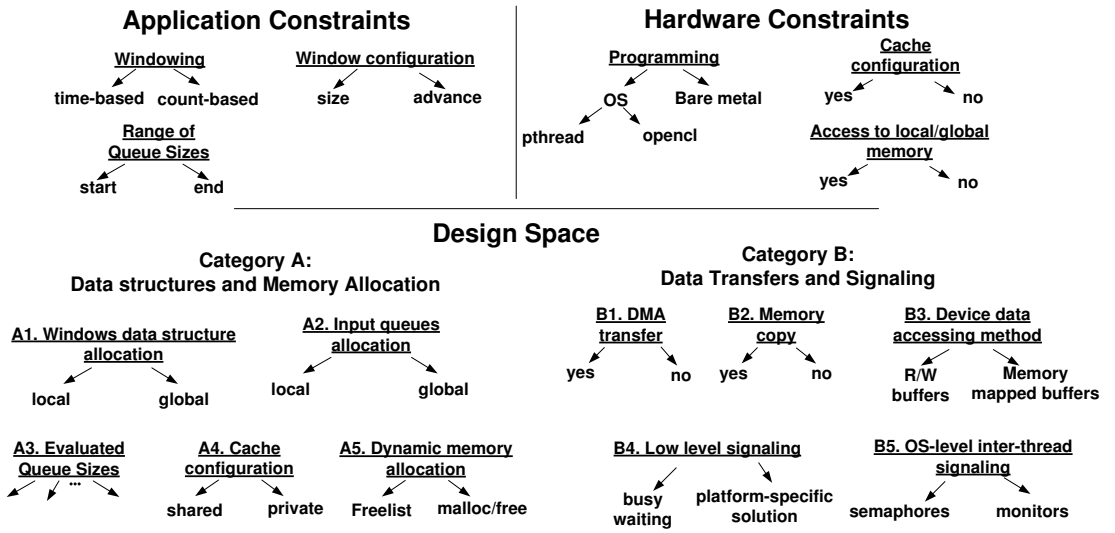
Σχήμα 5.2: Οι δομές δεδομένων του παραθύρου και του partials array που χρησιμοποιούνται στο count-based streaming aggregation.

την φάση *Update* για να τοποθετηθούν στα παράθυρα στα οποία ανήκουν. Η φάση *Update* παρέχει στην φάση *Output* πληροφορίες σχετικά με τα παράθυρα στα οποία τοποθετήθηκε το τελευταίο tuple, ώστε να εντοπιστούν τα ολοκληρωμένα παράθυρα, δηλαδή αυτά για τα οποία μπορεί να υπολογιστεί η aggregated τιμή. Η χρήση αποτελεσματικών τρόπων μεταφοράς δεδομένων από τη μία φάση στην επόμενη, επηρεάζει τόσο την απόδοση όσο και την κατανάλωση ενέργειας του συστήματος. Το ίδιο ισχύει και για τον τρόπο με τον οποίο συγχρονίζονται οι προσβάσεις στα κοινόχρηστα δεδομένα. Επιπλέον, ένα σημαντικό χαρακτηριστικό της υλοποίησης το οποίο επηρεάζει την απόδοση και την κατανάλωση ενέργειας του συστήματος είναι το μέγεθος των ουρών στις οποίες αποθηκεύονται τα εισερχόμενα tuples.

5.3.2 Count-based streaming aggregation

Στην περίπτωση του count-based aggregation, το μέγεθος των παραθύρων εξαρτάται από τον αριθμό των tuples που αποθηκεύονται σε αυτά, αντί για το χρονικό διάστημα που το παράθυρο είναι διαθέσιμο. Στη συγκεκριμένη μελέτη θεωρούμε παράθυρα συγκεκριμένου μεγέθους και το aggregation γίνεται με περιοδικό τρόπο, δηλαδή όταν έχει ληφθεί ένας συγκεκριμένος αριθμός από tuples από την φάση του *Update*. Κάθε φορά που ολοκληρώνεται ο υπολογισμός μιας aggregated τιμής όλα τα tuples που είναι αποθηκευμένα στο συγκεκριμένο παράθυρο απορρίπτονται και το παράθυρο αδειάζει (tumpling window).

Για την υλοποίηση του σεναρίου count-based aggregation, ακολουθήθηκε η προσέγγιση που περιγράφεται στο [66]. Το χρονικό διάστημα μεταξύ δύο aggregations βασίζεται στον αριθμό των tuples που είναι αποθηκευμένα στο παράθυρο και τα αποτελέσματα των αθροισμάτων ενός παραθύρου μπορεί να εξαρτώνται από τα αποτελέσματα του προηγούμενου παραθύρου. Έτσι, χρειάζεται μια δομή δεδομένων στην οποία θα αποθηκευτούν τα ενδιάμεσα αποτελέσματα του aggregation του προηγούμενου παραθύρου, ώστε να χρησιμοποιηθούν από το επόμενο.



Σχήμα 5.3: Περιορισμοί και χώρος σχεδιαστικών επιλογών για streaming aggregation.

Στο Σχήμα 5.2 απεικονίζονται οι δομές δεδομένων που χρησιμοποιούνται για υλοποίηση του count-based σεναρίου: Ένα παράθυρο *MxN* και το partials array που περιέχει *1xM* στοιχεία. *M* είναι ο μέγιστος αριθμός των των ροών δεδομένων και *N* είναι το πλάτος του παραθύρου. Όταν δεν είναι δυνατό για μία ροή να υπολογιστεί η aggregated τιμή για *N* tuples πριν προωθηθεί το συγκεκριμένο παράθυρο, το ενδιάμεσο αποτελέσματα του aggregation αποθηκεύεται στην κατάλληλη θέση του partials array. Η τιμή αυτή θα χρησιμοποιηθεί από το επόμενο παράθυρο για τον υπολογισμό της aggregated τιμής για *N* tuples για τη συγκεκριμένη ροή. Η έξοδος είναι ένα tuple που παράγεται από ένα ερώτημα (query) που εκτελείται στις *M* aggregated τιμές.

Είναι προφανές ότι το count-based aggregation παρέχει υψηλό παραλληλισμό. Η aggregated τιμή κάθε ροής, δηλαδή κάθε γραμμής του παραθύρου, μπορεί να υπολογιστεί ανεξάρτητα από τις υπόλοιπες. Έτσι, ο υπολογισμός της aggregated τιμής κάθε γραμμής μπορεί να ανατεθεί σε ένα διαφορετικό PE, ώστε οι υπολογισμοί να πραγματοποιηθούν παράλληλα. Όπως και στο time-based σενάριο, η μεταφορά δεδομένων και το μέγεθος του παραθύρου επηρεάζουν την απόδοση και την κατανάλωση ενέργειας της υλοποίησης του operator. Τα ενσωματωμένα συστήματα παρέχουν διάφορες λύσεις και κάθε μία επηρεάζει με διαφορετικό τρόπο τις μετρικές. Οι σχεδιαστικές επιλογές σχηματίζουν έναν χώρο σχεδιαστικών επιλογών που περιγράφεται στην επόμενη ενότητα.

5.4 Μεθοδολογία Παραμετροποίησης

Στην ενότητα αυτή περιγράφεται αρχικά ο χώρος σχεδιαστικών επιλογών και στη συνέχεια η προτεινόμενη μεθοδολογία. Ο χώρος σχεδιαστικών επιλογών της υλοποίησης του streaming aggretation παρουσιάζεται με μορφή δέντρων αποφάσεων, που

Πίνακας 5.1: Τα δέντρα αποφάσεων ή φύλλα που απενεργοποιούνται εξαιτίας application ή hardware constraints.

App./Hw constraint	Decision tree/ leaf disabled
Windowing(tuple-based)	A2, A3, A5, B4
Window configuration	may disable A1(local)
Programming(bare metal)	B3 and B5
Programming(pthread)	B1, B3, B4
Programming(OpenCL)	B1, B2, B4, B5
Cache config.(no)	A4
Access to local/global(no)	A1, A2

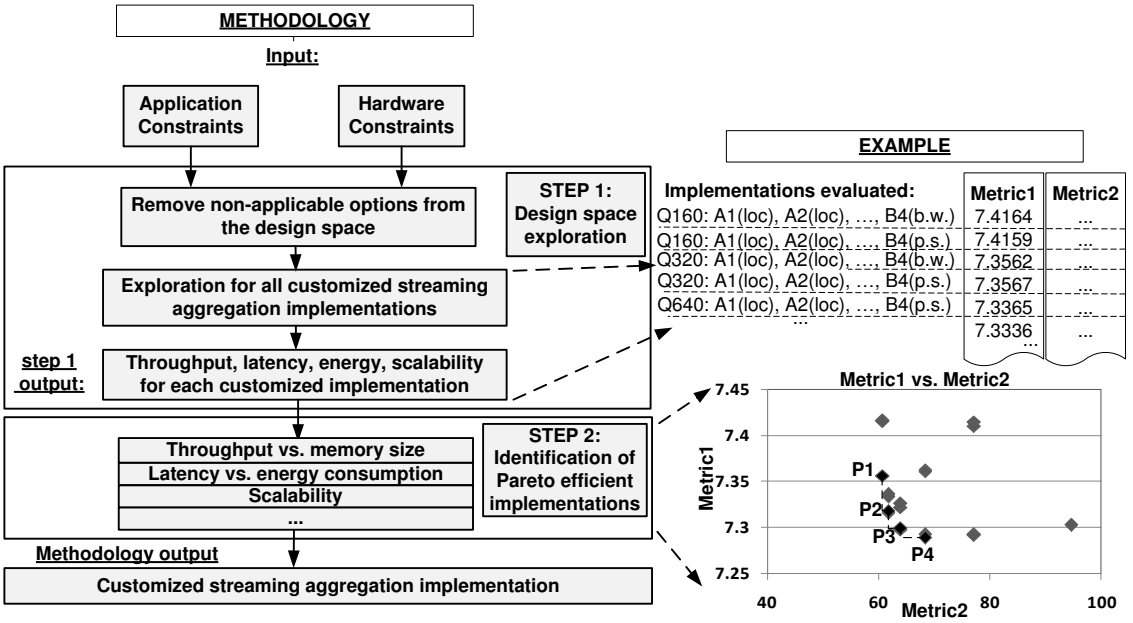
ταξινομούνται σε δύο κατηγορίες (Σχήμα 5.3).

Η Κατηγορία Α αποτελείται από δέντρα αποφάσεων σχετικά με την παραμετροποίηση και τη διαχείριση της μνήμης του συστήματος. Το δέντρο Α4 αφορά ρυθμίσεις σχετικές με την cache: ιδιωτική για κάθε πυρήνα ή κοινόχρηστη. Το δέντρο Α5 σχετίζεται με τη διαχείριση της δυναμικής μνήμης, δηλαδή με την χρήση κλήσεων του συστήματος malloc/free ή τη χρήση freelists.

Τα δέντρα αποφάσεων που σχετίζονται με τη μεταφορά δεδομένων και τον τρόπο με τον οποίο συγχρονίζονται οι προσβάσεις στα κοινά δεδομένα ταξινομούνται στην Κατηγορία Β. Τα τρία πρώτα δέντρα αποφάσεων αφορούν διαφορετικούς τρόπους με τους οποίους τα δεδομένα μπορούν να μεταφερθούν από την global στις local μνήμες, ή μεταξύ δύο local μνημών, ανάλογα με την ιεραρχία μνήμης του ενσωματωμένου συστήματος. Τα δέντρα αποφάσεων Β4 και Β5 αφορούν τον συγχρονισμό μεταξύ των PEs, όταν προσπελαύνουν κοινά δεδομένα. Σε χαμηλό επίπεδο, ο συγχρονισμός μπορεί να επιτευχθεί με spinning σε κοινές μεταβλητές (π.χ. busy waiting) ή με χρήση άλλων platform-specific μεθόδων. Σε αρχιτεκτονικές που υποστηρίζουν posix threads μπορούν να χρησιμοποιηθούν monitors ή semaphores.

Προφανώς, δεν είναι διαθέσιμες όλες οι σχεδιαστικές επιλογές σε κάθε σενάριο streaming aggregation και σε κάθε αρχιτεκτονική. Στο Σχήμα 5.3 απεικονίζονται τα application και platform constraints, δηλαδή οι περιορισμοί οι οποίοι επηρεάζουν το κατά πόσο τα δέντρα αποφάσεων ή τα φύλλα τους είναι εφαρμόσιμα σε ένα συγκεκριμένο σενάριο streaming aggregation που εκτελείται σε μία συγκεκριμένη αρχιτεκτονική. Οι περιορισμοί χρησιμοποιούνται για να απενεργοποιήσουν συγκεκριμένα δέντρα αποφάσεων που αντιστοιχούν σε σχεδιαστικές επιλογές που δεν υποστηρίζονται από την παρούσα αρχιτεκτονική ή το σενάριο.

Ο Πίνακας 5.1 συνοψίζει τα δέντρα αποφάσεων που απενεργοποιούνται εξαιτίας συγκεκριμένων application και hardware constraints. Για παράδειγμα, αν μία πλατφόρμα διαθέτει λειτουργικό σύστημα, τότε συνήθως δεν υπάρχει πρόσβαση από τον προγραμματιστή σε χαμηλού επιπέδου λειτουργίες, (π.χ. μεταφορές δεδομένων), κα-

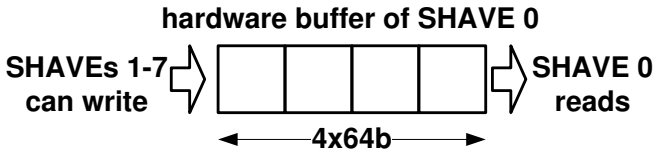


Σχήμα 5.4: Μεθοδολογία παραμετροποίησης.

θώς ο χειρισμός τους γίνεται από το λειτουργικό σύστημα. Ο περιορισμός window configuration μπορεί να αναγκάσει την αποθήκευση του παραθύρου στην global μνήμη, εξαιτίας του μεγέθους του. Οι περιορισμοί, διαγράφοντας τα μη-σχετικά δέντρα αποφάσεων (pruning), μετατρέπουν το platform και application-independent design space σε application και platform dependent. Με άλλα λόγια, το προσαρμόζουν στα χαρακτηριστικά του streaming aggregation σεναρίου και της αρχιτεκτονικής στην οποία αυτό εκτελείται.

Μετά το pruning του χώρου σχεδιαστικών αποφάσεων, δημιουργούνται, από τα εναπομείναντα φύλλα των δέντρα αποφάσεων, οι παραμετροποιημένες υλοποιήσεις του streaming aggregation. Με άλλα λόγια, οι υλοποιήσεις που θα εξεταστούν είναι αυτές που σχηματίζονται από τα φύλλα των δέντρων αποφάσεων που δεν έχουν απενεργοποιηθεί από τους περιορισμούς.

Η προτεινόμενη μεθοδολογία αποτελείται από 2 βήματα τα οποία απεικονίζονται στο Σχήμα 5.4. Η είσοδος της μεθοδολογίας είναι τα applications και τα hardware constraints. Η έξοδος είναι η παραμετροποιημένη υλοποίηση streaming aggregation, βελτιστοποιημένη ως προς μετρικές της απόδοσης και της κατανάλωσης ενέργειας. Το πρώτο βήμα αφορά το pruning του χώρου σχεδιαστικών αποφάσεων από τους περιορισμούς και τον σχηματισμό των συμβατών υλοποιήσεων που θα εξεταστούν. Στη συνέχεια, εκτελείται κάθε μία από τις συγκεκριμένες υλοποιήσεις και συλλέγονται αποτελέσματα ως προς το throughput, το latency, το απαιτούμενο μέγεθος μνήμης και την κατανάλωση ενέργειας. Σε περίπτωση που η αρχιτεκτονική διαθέτει μεγάλο αριθμό από PEs μπορεί να εξεταστεί και το scalability. Στο δεύτερο βήμα εντοπίζονται οι Pareto efficient υλοποιήσεις. Τα trade-offs που έχουν εντοπιστεί μεταξύ διαφο-



Σχήμα 5.5: Hardware buffers της Myriad1.

ρετικών υλοποιήσεων παρουσιάζονται σε μορφή Pareto curves. Οι προγραμματιστές μπορούν να πραγματοποιήσουν trade-offs μεταξύ μετρικών επιλέγοντας διαφορετικές υλοποιήσεις.

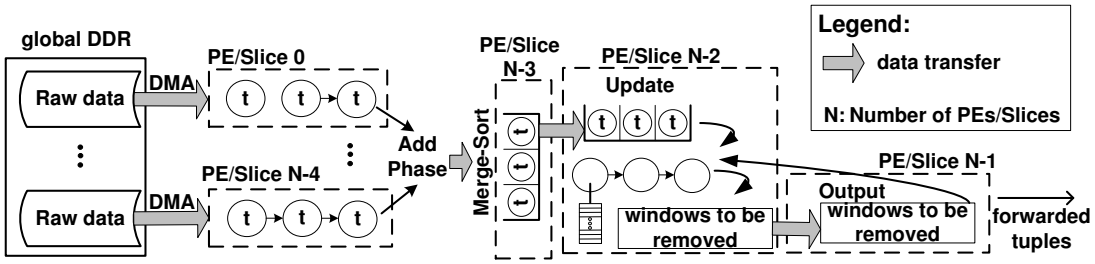
5.5 Εφαρμογή της Μεθοδολογίας

Στην ενότητα αυτή περιγράφονται αρχικά οι ενσωματωμένες αρχιτεκτονικές στις οποίες χρησιμοποιήθηκαν για την αξιολόγηση της μεθοδολογίας. Στη συνέχεια περιγράφονται τα χαρακτηριστικά των σεναρίων streaming aggregation που εκτελέστηκαν στις αρχιτεκτονικές αυτές και τα πειραματικά αποτελέσματα. Τέλος, αναλύονται τα συμπεράσματα που προέκυψαν από τα πειραματικά αποτελέσματα.

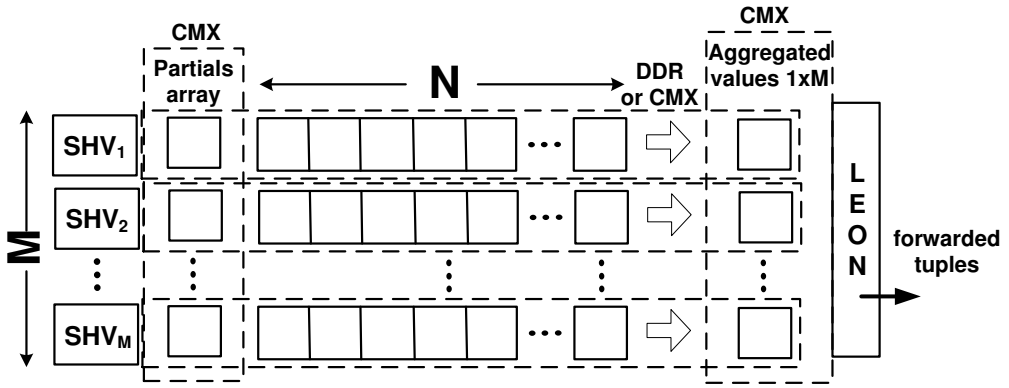
5.5.1 Περιγραφή των Αρχιτεκτονικών

Η επεξεργαστές Myriad σχεδιάζονται από τη Movidius Ltd. [74]. Στοχεύουν σε εφαρμογές ρομποτικής όρασης και επεξεργασίας ροών δεδομένων. Χρησιμοποιούνται στα πλαίσια του Project Tango που στοχεύει στον σχεδιασμό φορητών συσκευών ικανών να δημιουργήσουν το τρισδιάστατο μοντέλο του περιβάλλοντος γύρω τους [75]. Ανήκουν σε μία οικογένεια επεξεργαστών για φορητές συσκευές που παρέχουν υψηλή απόδοση ανά watt [65].

Η Myriad1 είναι σχεδιασμένη στα 65nm. Παρέχει 8 VLIW επεξεργαστικούς πυρήνες που ονομάζονται Streaming Hybrid Architecture Vector Engines (SHAVEs) και λειτουργούν στα 180MHz και έναν RISC επεξεργαστή LEON3, που ελέγχει τη ροή δεδομένων, χειρίζεται τα interrupts κλπ.. Περισσότερες τεχνικές πληροφορίες για την Myriad αναφέρονται στο [6]. Για κάθε SHAVE είναι διαθέσιμη μία τοπική DMA engine. Επίσης, η Myriad παρέχει ένα σετ από hardware buffers για απευθείας επικοινωνία μεταξύ των SHAVEs. Κάθε SHAVE έχει το δικό του hardware buffer στο οποίο έχει FIFO πρόσβαση. Το μέγεθός του είναι 4x64bits. Όπως φαίνεται στο Σχήμα 5.5, κάθε SHAVE μπορεί να αποθηκεύσει δεδομένα στο buffer κάθε άλλου SHAVE, αλλά μπορεί να διαβάσει μόνο από το δικό του. Κάθε SHAVE μπορεί να γράψει στην ουρά κάποιου άλλου buffer, ενώ ο κάτοχος του συγκεκριμένου buffer διαβάζει από το πρώτο στοιχείο. Ένα ενδιαφέρον χαρακτηριστικό της Myriad1 είναι το γεγονός ότι όταν ένας SHAVE επιχειρεί να γράψει σε ένα γεμάτο buffer ή να διαβάσει από το δικό του και αυτό είναι άδειο, τότε εισέρχεται σε κατάσταση χαμηλής ισχύος.



(α΄) Υλοποίηση του time-based aggregation σεναρίου στη Myriad.



(β΄) Υλοποίηση του count-based aggregation σεναρίου στη Myriad.

Σχήμα 5.6: Υλοποίηση των time-based και count-based streaming aggregation στις Myriad.

Εκμεταλλευόμαστε αυτό το γεγονός για να προτείνουμε υλοποιήσεις του streaming aggregation, βελτιστοποιημένες ως προς την κατανάλωση ενέργειας στη Myriad1.

Η Myriad2 έχει σχεδιαστεί στα 28nm [76]. Σε αντίθεση με τη Myriad1, η Myriad2 περιέχει 12 πυρήνες SHAVEs που λειτουργούν στα 504MHz και δύο RISC LEON4: Ο LEON-RT αναλαμβάνει την κατανομή εργασιών στο chip, ενώ ο LEON-OS εκτελεί το λειτουργικό σύστημα: RTEMS/Linux, etc.. Η Myriad2 παρέχει μία ιεραρχική δομή DMA και hardware buffers 16x64bits.

Όσο αφορά τα χαρακτηριστικά της μνήμης, η Myriad1 παρέχει 1MB τοπική μνήμη με unified address space που ονομάζεται Connection Matrix (CMX). Κάθε SHAVE έχει αποδοτικότερη πρόσβαση σε 128KB της συγκεκριμένες μνήμης. Έτσι, η CMX μνήμη μπορεί να θεωρηθεί ότι αποτελείται από έναν αριθμό από "slices", με καθένα από αυτά να είναι συνδεδεμένο σε έναν από τους 8 SHAVEs. Στη Myriad2 η CMX είναι 2MB και κάθε slice 128KB. Επίσης, η Myriad2 παρέχει 1KB L1 και 256KB L2 cache μνήμη. Τέλος και οι 2 αρχιτεκτονικές παρέχουν DDR μνήμη: 64MB η Myriad1 και 128MB η Myriad2.

Στο time-based σενάριο, τα δεδομένα σε raw μορφή (που μπορεί να παράγονται από sensors, κάμερες κλπ.) αποθηκεύονται στη DDR. Κάθε ουρά στην οποία αποθηκεύονται τα εισερχόμενα tuples, ανατίθεται σε έναν SHAVE και τοποθετείται στο

δικό του τοπικό slice. Επομένως, κάθε SHAVE πραγματοποιεί DMA transfers ώστε να μεταφέρει δεδομένα από την DDR στο τοπικό του slice και να δημιουργήσει από τα raw δεδομένα νέα tuples, τα οποία στη συνέχεια αποθηκεύει στην ουρά. Τα παράθυρα αποθηκεύονται σε μία απλά συνδεδεμένη λίστα η οποία είναι αποθηκευμένη στο slice του SHAVE στον οποίο έχει ανατεθεί η φάση του *Update*. Ο τρόπος αποθήκευσης στη μνήμη των δομών δεδομένων, καθώς και άλλες πληροφορίες, απεικονίζονται στο Σχήμα 5.6α΄. Στην περίπτωση του count-based aggregation που χρησιμοποιεί ένα παράθυρο μεγέθους *MxN*, κάθε ένας από τους *M* SHAVEs μεταφέρει συνεχώς δεδομένα από την DDR στο τοπικό του slice, τα οποία αντιστοιχούν σε *N* tuples. Αν η τιμή του *N* είναι πολύ μεγάλη, είναι πιθανό το παράθυρο να μην μπορεί να αποθηκευτεί στη CMX, οπότε αποθηκεύεται στη DDR. Κάθε SHAVE υπολογίζει την aggregated τιμή για *N* tuples και προωθεί την τιμή στον LEON, ο οποίος παράγει το τελικό αποτέλεσμα που αντιστοιχεί στο συγκεκριμένο παράθυρο. Η υλοποίηση απεικονίζεται στο Σχήμα 5.6β΄.

Η Freescale I.MX.6 Quad περιέχει 4 πυρήνες ARM Cortex A9 που λειτουργούν στο 1GHz [2]. Ανήκει σε μία κατηγορία πολυπύρηνων ενσωματωμένων αρχιτεκτονικών που λειτουργούν ως single-board-computers και τρέχουν Linux OS. Παρέχει 1GB RAM μνήμης και 2 επίπεδα cache. Στη συγκεκριμένη αρχιτεκτονική τα raw δεδομένα είναι σε text αρχεία και τμήματα των δεδομένων αποθηκεύονται στην RAM με χρήση της συνάρτησης *fread()*. Στη συνέχεια, από τα δεδομένα σε raw μορφή δημιουργούνται τα tuples και τοποθετούνται στις ουρές. Όλες οι δομές δεδομένων είναι αποθηκευμένες στην RAM.

Η Exynos 5 Octa είναι μία big.LITTLE ARM-based αρχιτεκτονική που αφορά φορητές συσκευές και είναι σχεδιασμένη από τη SAMSUNG στα 28nm [77]. Περιέχει δύο ARM clusters: 4 Cortex-A15 και 4 Cortex-A7. Επιπλέον, διαθέτει μία PowerVR SGX544 embedded GPU που υποστηρίζει OpenCL1.1 και περιέχει τρεις υπολογιστικούς πυρήνες στα 533MHz. Η πλατφόρμα που χρησιμοποιήθηκε είναι η Odroid-XU που παρέχει 2GB DDR3 RAM [78]. Στη συγκεκριμένη εργασία, χρησιμοποιήσαμε την PowerVR GPU για την υλοποίηση του count-based streaming σεναρίου, υλοποιημένο σε OpenCL.

5.5.2 Πειραματική διαδικασία

Το dataset στο οποίο έγινε το aggregation είναι δεδομένα τα οποία έχουν συλλεχθεί από την πλατφόρμα SoundCloud [79]. Αποτελείται από δεδομένα που προήλθαν από περίπου 40.000 χρηστες που έχουν ανταλλάξει περίπου 250.000 μηνύματα μεταξύ 2007 και 2013. Τα εισερχόμενα tuples περιέχουν τα εξής πεδία: *timestamp*, *user_id*, *song_id* και *comment*. Η συνάρτηση που κάνει το aggregation προωθεί το id του χρήστη με τον μεγαλύτερο αριθμό μηνυμάτων σε κάθε παράθυρο. Στο time-based σενάριο, η συνάρτηση εκτελείται για κάθε sliding window, ενώ στο count-based εκτελείται για

Πίνακας 5.2: Hardware constraints για τις πλατφόρμες Myriad1, Myriad2, I.MX.6 Quad και Exynos για τα δύο σενάρια.

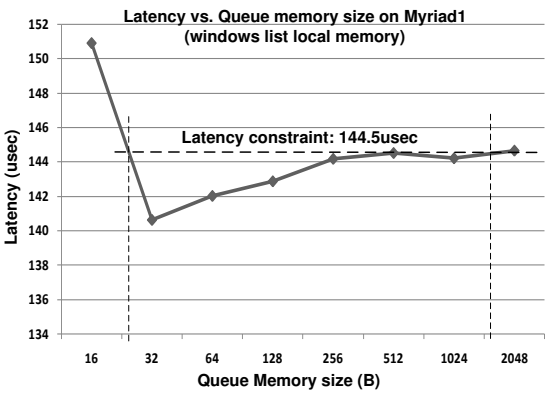
	Time-based aggregation			Count-based aggregation		
	Myriad1	Myriad2	I.MX.6	Myriad1	Myriad2	Exynos
windowing	time	time	time	count	count	count
programming	bare	bare	pthread	bare	bare	OpenCL
cache config.	no	yes	no	no	yes	no
access loc./glob. mem.	yes	yes	no	yes	yes	yes

κάθε tumpling window μεγέθους *MxN* tuples.

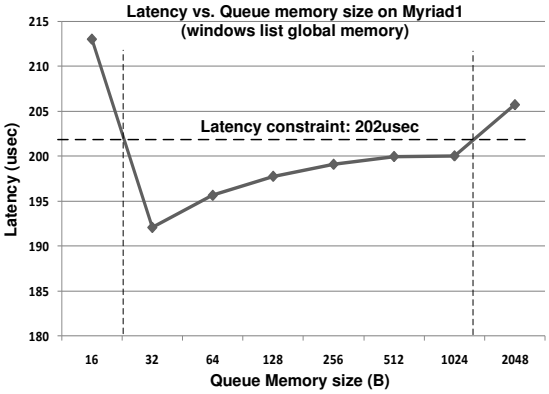
Ο aggregation operator είναι υλοποιημένος σε C. Το throughput μετριέται ως ο αριθμός των tuples που επεξεργάζονται ανά δευτερόλεπτο, ενώ το latency ως η διαφορά των timestamps μεταξύ του tuple που περιέχει την aggregated τιμή στην έξοδο του pipeline και του τελευταίου tuple που αποθηκεύτηκε στο συγκεκριμένο παράθυρο. Στο I.MX.6 η κατανάλωση ενέργειας μετρήθηκε με hardware instrumentation, με χρήση της συσκευής Watts Up PRO και ακολουθώντας μεθόδους που έχουν προταθεί στη βιβλιογραφία [80][81]. Στη Myriad2 μετρήθηκε με χρήση της συσκευής MV198 που είναι ενσωματωμένη στην πλατφόρμα και υπολογίζει το ρεύμα που διοχετεύεται στα διάφορα power islands. Στη Myriad1 υπολογίστηκε με χρήση του simulator moviSim που παρέχεται από το MDK της Myriad. Στο Exynos έγινε χρήση των power sensors που είναι ενσωματωμένη στην συγκεκριμένη πλατφόρμα [78]. Όλες οι τιμές που παρουσιάζονται είναι μέσες τιμές 10 εκτελέσεων. Κάθε πείραμα έχει εκτελεστεί από 30 δευτερόλεπτα ως 1 λεπτό.

Στο time-based σενάριο, το οποίο είναι ουσιαστικά ένα pipeline, εκτελέστηκε στις πλατφόρμες της Myriad και της I.MX.6 Quad, ενώ το count-based, που παρέχει αυξημένο data parallelism, στις Myriad και στην embedded GPU της Exynos. Όπως αναφέρθηκε σε προηγούμενη ενότητα, η Myriad1 παρέχει 8 PEs. Στο time-based aggregation, κάθε μία απο τις *Merge-Sort*, *Update* και *Output* φάσεις έχει ανατεθεί σε ένα συγκεκριμένο PE. Κάθε ένα από τα υπόλοιπα 5 PEs αναλαμβάνει μία ουρά από tuples. Στη Myriad2, που παρέχει 12 PEs, ο αριθμός των ουρών είναι 9. Στο I.MX.6 Quad που παρέχει 4 PEs, ανατέθηκε κάθε μία από τις τρεις φάσεις σε ένα PE, ενώ στο τέταρτο αναλαμβάνει τον χειρισμό 5 ουρών.

Τα hardware constraints απεικονίζονται στον Πίνακα 5.2. Τα πειράματα που πραγματοποιήθηκαν είναι τα ακόλουθα: Η μεθοδολογία υλοποιήθηκε στο I.MX.6 για ένα window configuration. Αντίθετα, στη Myriad1 και στη Myriad2 παρουσιάζονται αποτελέσματα από 2 διαφορετικά σενάρια: Στο πρώτο, το window configuration, δηλαδή οι τιμές *size* and *advance*, τίθενται έτσι ώστε το μέγεθος μνήμης της δομής δεδομένων των παραθύρων να είναι αρκετά μικρό ώστε η δομή να χωρά στην local



(α′) Windows list στη local μνήμη.



(β′) Windows list στη global μνήμη.

Σχήμα 5.7: Latency vs. μέγεθος ουράς στη Myriad1.

μνήμη. Στο δεύτερο πείραμα, η δομή δεδομένων των παραθύρων χωρά μόνο στην global μνήμη. Έτσι, εξετάζουμε τον βαθμό στον οποίο η τοποθέτηση στη μνήμη της συγκεκριμένης δομής επηρεάζει τις διάφορες μετρικές. Στο count-based σενάριο, το aggregation γίνεται παράλληλα, από τον επιταχυντή (accelerator) κάθε πλατφόρμας: Τους SHAVEs της Myriad και την GPU της Exynos.

Η έξοδος της μεθοδολογίας είναι ένα σύνολο από Pareto points για throughput vs. μέγεθος μνήμης και latency vs. κατανάλωση ενέργειας. Στο time-based σενάριο παρουσιάζονται αποτελέσματα για scalability, τόσο για την Myriad1, όσο και για τη Myriad2. Οι υλοποιήσεις για τις οποίες μελετήθηκε το scalability είναι αυτές οι οποίες βρέθηκαν να είναι Pareto optimal στο πείραμα latency vs. κατανάλωση ενέργειας.

5.5.3 Αποτελέσματα για time-based aggregation

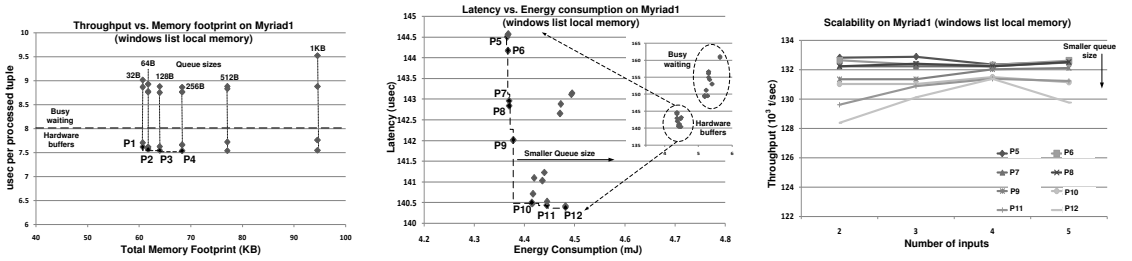
Στο time-based σενάριο, αξιολογήθηκε κάθε υλοποίηση για διαφορετικό αριθμό μεγεθών ουράς. Τα μεγέθη που έχουμε επιλέξει είναι αυτά που παρουσιάζουν latency χαμηλότερο από ένα threshold, που έχουμε ορίσει αυθαίρετα. Έτσι, αρχικά υπολογίζουμε το latency για ένα range από μεγέθη ουράς και επιλέγουμε τα μεγέθη για τα οποία η υλοποίηση παρουσιάζει latency μικρότερο από το threshold. Στη συνέχεια, προχωράμε στην εφαρμογή της μεθοδολογίας. Συνολικά, αξιολογήθηκαν 48 υλοποιήσεις στη Myriad και 4 στην I.MX.6 Quad. Ο αριθμός των υλοποιήσεων εξαρτάται, πέρα από τα χαρακτηριστικά κάθε πλατφόρμας, από τον αριθμό των μεγεθών ουρών που αξιολογήθηκαν.

Εφαρμογή στη Myriad1

Στο πρώτο πείραμα στη Myriad1, οι τιμές *size* και *advance* των παραθύρων έχουν επιλεγεί έτσι ώστε η δομή δεδομένων να χωρά να τοποθετηθεί στην local μνήμη.

Πίνακας 5.3: Περιγραφή των βέλτιστων κατά Pareto σημείων στη Myriad1. Η σχεδιαστική επιλογή B4(p.s.), δηλ. platform specific, αναφέρεται στα hardware buffers της Myriad1.

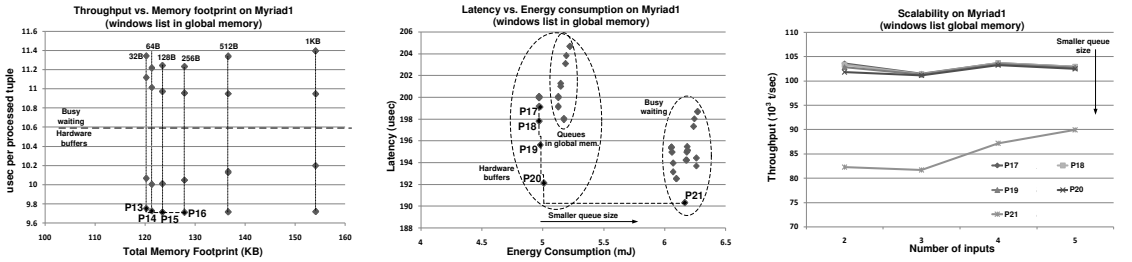
Pareto	Description	Pareto	Description	Pareto	Description
P1	A1(l), A2(l), A3(32B), A5(fl), B2(yes), B4(p.s.)	P8	A1(l), A2(l), A3(128B), A5(fl), B2(yes), B4(b.w.)	P15	A1(l), A2(l), A3(128B), A5(fl), B2(yes), B4(b.w.)
P2	A1(l), A2(l), A3(64B), A5(fl), B2(yes), B4(p.s.)	P9	A1(l), A2(l), A3(64B), A4(fl), B1(yes), B4(p.s.)	P16	A1(on), A2(on), A3(256B), A5(fl), B2(yes), B4(p.s.)
P3	A1(l), A2(l), A3(128B), A5(fl), B2(yes), B4(p.s.)	P10	A1(l), A2(l), A3(64B), A5(fl), B2(yes), B4(p.s.)	P17	A1(l), A2(l), A3(256B), A5(fl), B1(yes), B4(p.s.)
P4	A1(l), A2(l), A3(256B), A5(fl), B2(yes), B4(p.s.)	P11	A1(l), A2(l), A3(64B), A5(fl), B1(yes), B4(p.s.)	P18	A1(l), A2(l), A3(128B), A5(fl), B2(yes), B4(p.s.)
P5	A1(l), A2(l), A3(512B), A5(fl), B2(yes), B4(p.s.)	P12	A1(l), A2(l), A3(32B), A5(fl), B2(yes), B4(p.s.)	P19	A1(l), A2(l), A3(64B), A5(fl), B2(yes), B4(p.s.)
P6	A1(l), A2(l), A3(256B), A5(fl), B2(yes), B4(p.s.)	P13	A1(l), A2(l), A3(32B), A5(fl), B2(yes), B4(p.s.)	P20	A1(l), A2(l), A3(32B), A5(fl), B2(yes), B4(p.s.)
P7	A1(l), A2(l), A3(128B), A5(fl), B1(yes), B4(p.s.)	P14	A1(l), A2(l), A3(64B), A5(fl), B2(yes), B4(p.s.)	P21	A1(l), A2(l), A3(32B), A5(fl), B2(yes), B4(b.w.)



(α΄) Throughput vs. μέγεθος μνήμης (Windows local)

(β΄) Latency vs. κατανάλωση ενέργειας (Windows local)

(γ΄) Scalability (Windows στη local)



(δ΄) Throughput vs. μέγεθος μνήμης (Windows global)

(ε΄) Latency vs. κατανάλωση ενέργειας (Windows global)

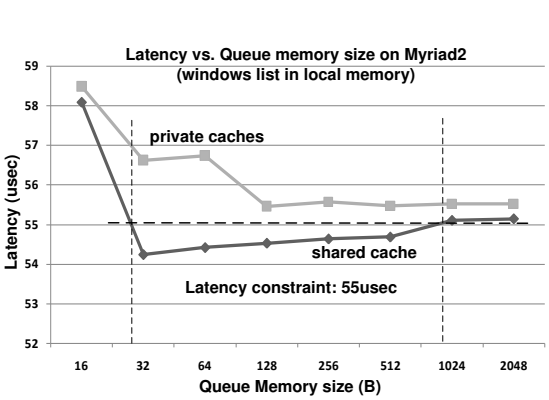
(ς΄) Scalability (Windows στη global)

Σχήμα 5.8: Time-based streaming aggregation υλοποιήσεις στη Myriad1.

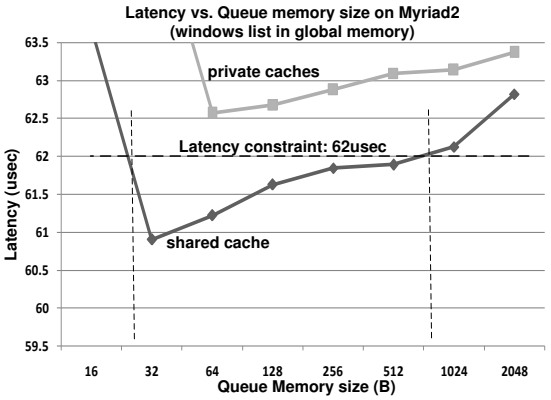
Θεωρώντας latency threshold 144.5usec, το μέγεθος των ουρών που θα εξεταστούν είναι από 32B ως 1KB (Σχήμα 5.7α΄).

Τα αποτελέσματα για throughput vs. μέγεθος μνήμης παρουσιάζονται στο Σχήμα 5.8α΄. Παρατηρούμε ότι τα σημεία Pareto μπορούν να χωριστούν σε δύο κατηγορίες: Σε αυτά με απόδοση χαμηλότερη των 8usec/tuple που αντιστοιχούν σε υλοποιήσεις που κάνουν χρήση του μηχανισμού busy waiting για τον συγχρονισμό των πυρήνων κατά τη μεταφορά των δεδομένων και στις υπόλοιπες που χρησιμοποιούν τα hardware Myriad buffers. Εντοπίστηκαν 4 βέλτιστα σημεία Pareto, τα οποία περιγράφονται στον Πίνακα 5.3. Οι Pareto υλοποιήσεις μπορούν να χρησιμοποιηθούν για την πραγματοποίηση trade-offs μεταξύ απόδοσης και μεγέθους μνήμης. Για παράδειγμα, το throughput μπορεί να αυξηθεί ως 1,02% επιλέγοντας την υλοποίηση *P4*, σε σχέση με την υλοποίηση *P1*. Αντίστοιχα, η *P1* παρέχει 11,2% μικρότερο μέγεθος μνήμης σε σύγκριση με την *P4*.

Τα Pareto σημεία για latency vs. κατανάλωση ενέργειας μπορούν να ομαδοποιηθούν στις ίδιες κατηγορίες: Στην πρώτη κατηγορία ανήκουν οι υλοποιήσεις που κάνουν χρήση busy waiting για τον συγχρονισμό της μεταφοράς δεδομένων μεταξύ των βημάτων του pipeline, ενώ στη δεύτερη οι αυτές που χρησιμοποιούν για τον ίδιο σκοπό τα hardware buffers. Οι υλοποιήσεις της δεύτερης κατηγορίας είναι αποδοτικότερες τόσο όσο αφορά το latency, όσο και την κατανάλωση ενέργειας. Εντοπίστηκαν 8 Pareto ση-



(α′) Windows list στη local μνήμη.



(β′) Windows list in global μνήμη.

Σχήμα 5.9: Latency vs. μέγεθος ουράς στη Myriad2.

μεία, που μπορούν να χρησιμοποιηθούν για την πραγματοποίηση trade-offs μεταξύ απόδοσης και ενέργειας: 2,85% χαμηλότερο latency (*P12*) και 2,6% χαμηλότερη κατανάλωση ενέργειας (*P5*).

Τέλος, τα πειραματικά αποτελέσματα σχετικά με το scalability, παρουσιάζονται στο Σχήμα 5.8γ′. Το throughput παραμένει σχεδόν σταθερό ή αυξάνει οριακά σε όλες τις υλοποιήσεις. Εξαίρεση αποτελεί η υλοποίηση *P12*, στην οποία οι ουρές έχουν πολύ μικρό μέγεθος (32B).

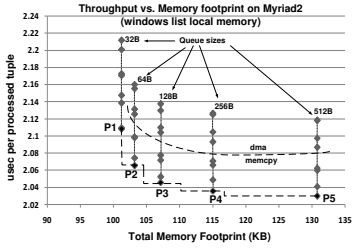
Στο δεύτερο πείραμα, επιλέχθηκε latency threshold 202usec (Σχήμα 5.7β′). Παρατηρούμε ότι τόσο στο Σχήμα 5.8δ′, όσο και στο Σχήμα 5.8ε′ ότι το throughput είναι χαμηλότερο και το latency υψηλότερο σε σχέση με το προηγούμενο πείραμα, καθότι σε αυτό η δομή των παραθύρων βρίσκεται στην global DDR μνήμη. Τα βέλτιστα Pareto σημεία του Σχήματος 5.8δ′ απεικονίζουν trade-offs μεταξύ throughput και μεγέθους μνήμης (κατά 0,5% αυξημένο throughput με επιλογή της υλοποίησης *P16* και 5,9% μειωμένο μέγεθος μνήμης με επιλογή της *P13*). Στο Σχήμα 5.8ε′, παρατηρούμε ότι η υλοποίηση *P21* είναι αυτή που παρουσιάζει το χαμηλότερο latency (4.45% χαμηλότερο σε σχέση με την *P17*), ενώ η *P17* παρέχει χαμηλότερη κατανάλωση ενέργειας (κατά 19,3% σε σχέση με την *P21*). Τα αποτελέσματα του scalability απεικονίζονται στο Σχήμα 5.8ϛ′. Όλες οι υλοποιήσεις παρουσιάζουν σταθερά υψηλό throughput που εξαρτάται σε μικρό βαθμό από τον αριθμό των ουρών. Εξαίρεση αποτελεί η *P21* που κάνει χρήση του μηχανισμού busy-waiting για τον συγχρονισμό των πυρήνων κατά τη μεταφορά δεδομένων και παρουσιάζει πολύ χαμηλότερο throughput σε σχέση με τις υπόλοιπες υλοποιήσεις.

Εφαρμογή στη Myriad2

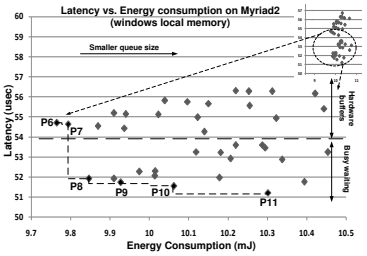
Στα Σχήματα 5.9α′ και 5.9β′ απεικονίζεται το latency για διαφορετικές ρυθμίσεις της cache, κοινόχρηστη και ιδιωτική (δέντρο απόφασης *A4* στο Σχήμα 5.3). Η περι-

Πίνακας 5.4: Περιγραφή των βέλτιστων κατά Pareto σημείων στη Myriad2. Η σχεδιαστική επιλογή B4(p.s.), δηλ. platform specific, αναφέρεται στα hardware buffers της Myriad2.

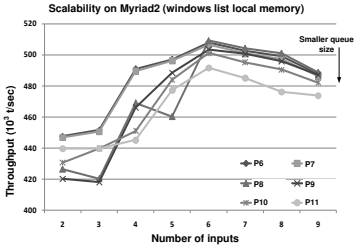
Par.	Description	Par.	Description	Par.	Description
P1	A1(l), A2(l), A3(32B), A4(s), A5(fl), B2(y), B4(b.w.)	P8	A1(l), A2(l), A3(512B), A4(s), A5(fl), B2(y), B4(b.w.)	P15	A1(l), A2(l), A3(256B), A4(s), A5(fl), B2(y), B4(p.s.)
P2	A1(l), A2(l), A3(64B), A4(s), A5(fl), B2(yes), B4(p.s.)	P9	A1(l), A2(l), A3(128B), A4(s), A5(fl), B1(y), B4(b.w.)	P16	A1(l), A2(l), A3(256B), A4(s), A5(fl), B2(y), B4(p.s.)
P3	A1(l), A2(l), A3(128B), A4(s), A5(fl), B2(y), B4(p.s.)	P10	A1(l), A2(l), A3(64B), A4(s), A5(fl), B2(y), B4(b.w.)	P17	A1(l), A2(l), A3(512B), A4(s), A5(fl), B1(y), B4(b.w.)
P4	A1(l), A2(l), A3(256B), A4(s), A5(fl), B2(y), B4(p.s.)	P11	A1(l), A2(l), A3(32B), A4(s), A5(fl), B1(y), B4(b.w.)	P18	A1(l), A2(l), A3(128B), A4(s), A5(fl), B2(y), B4(b.w.)
P5	A1(l), A2(l), A3(512B), A4(s), A5(fl), B2(y), B4(p.s.)	P12	A1(l), A2(l), A3(32B), A4(s), A5(fl), B2(y), B4(b.w.)	P19	A1(l), A2(l), A3(64B), A4(s), A5(fl), B2(y), B4(b.w.)
P6	A1(l), A2(l), A3(512B), A4(s), A5(fl), B2(y), B4(p.s.)	P13	A1(l), A2(l), A3(64B), A4(s), A5(fl), B2(y), B4(p.s.)	P20	A1(l), A2(l), A3(32B), A4(s), A5(fl), B2(y), B4(b.w.)
P7	A1(l), A2(l), A3(256B), A4(s), A5(fl), B1(y), B4(p.s.)	P14	A1(l), A2(l), A3(128B), A4(s), A5(fl), B2(y), B4(p.s.)		



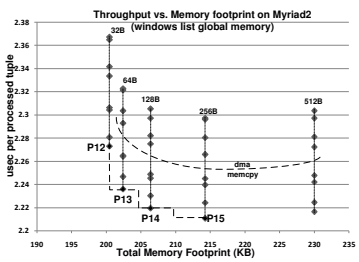
(α΄) Throughput vs. μέγεθος μνήμης (Windows στη local)



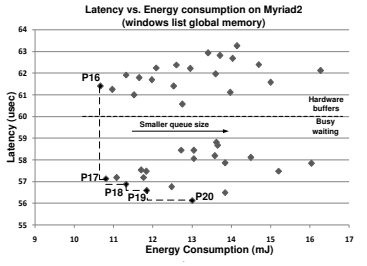
(β΄) Latency vs. κατανάλωση ενέργειας (Windows στη local)



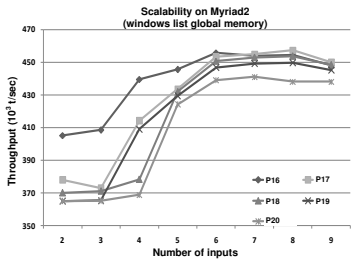
(γ΄) Scalability (Windows στη local memory)



(δ΄) Throughput vs. μέγεθος μνήμης (Windows στη global)



(ε΄) Latency vs. κατανάλωση ενέργειας (Windows στη global)



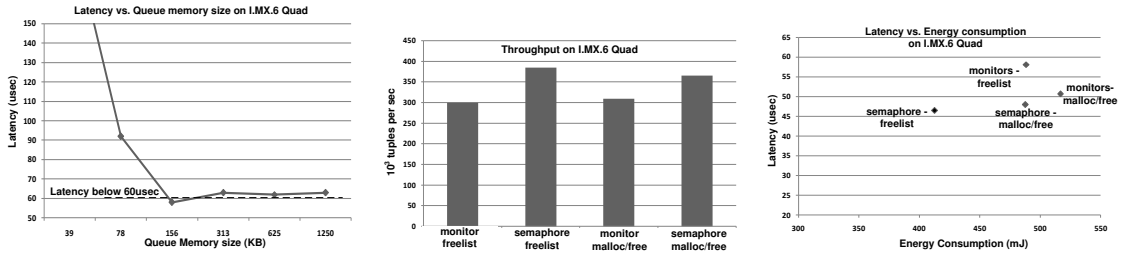
(ς΄) Scalability (Windows στη global)

Σχήμα 5.10: Time-based streaming aggregation υλοποιήσεις στη Myriad2.

γραφή κάθε βέλτιστου κατά Pareto σημείου βρίσκεται στον Πίνακα 5.4. Παρατηρούμε ότι η χρήση κοινόχρηστης cache παρέχει χαμηλότερο latency σε σχέση με την χρήση ιδιωτικής και στα δύο πειράματα (ως 4,2%). Επομένως, όλα τα αποτελέσματα που παρουσιάζονται στη συνέχεια αφορούν υλοποιήσεις που κάνουν χρήση αποκλειστικά κοινόχρηστης cache. Στο πρώτο πείραμα στη Myriad2, στο Σχήμα 5.9α΄, το μέγεθος της δομής δεδομένων των παραθύρων είναι τέτοιο ώστε να μπορεί να τοποθετηθεί στην τοπική μνήμη. Το latency threshold είναι στα 55usec, οπότε εξετάζονται μεγέθη ουράς από 32 ως 512B.

Τα αποτελέσματα throughput vs. μέγεθος μνήμης απεικονίζονται στο Σχήμα 5.10α΄. Υλοποιήσεις που κάνουν χρήση της *memcpy* παρέχουν υψηλότερη απόδοση σε σχέση με αυτές που κάνουν χρήση της dma engine της Myriad2 για τη μεταφορά δεδομένων μεταξύ των CMX slices. Οι 5 βέλτιστες Pareto υλοποιήσεις απεικονίζουν trade-offs, ως 3,7% μειωμένο throughput (*P5*) και 22,5% μειωμένο μέγεθος μνήμης (*P1*).

Τα αποτελέσματα για latency vs. κατανάλωση ενέργειας παρουσιάζονται στο Σχήμα 5.10β΄. Οι υλοποιήσεις μπορούν να ομαδοποιηθούν σε 2 κατηγορίες: Σε αυτές που κάνουν χρήση busy-waiting για τον συγχρονισμό των πυρήνων κατά τη μεταφορά δεδομένων μεταξύ των slices και σε αυτές που πραγματοποιούν μεταφορά μέσω των hardware buffers. Τα 6 Pareto σημεία παρουσιάζουν trade-offs μεταξύ των δύο μετρικών: ως 6,37% χαμηλότερο latency με επιλογή της υλοποίησης *P11* και 5,2%



(α΄) Latency vs. μέγεθος ουράς **(β΄)** Throughput **(γ΄)** Latency vs. κατανάλωση ενέργειας

Σχήμα 5.11: Time-based streaming aggregation υλοποιήσεις στην I.MX.6 Quad.

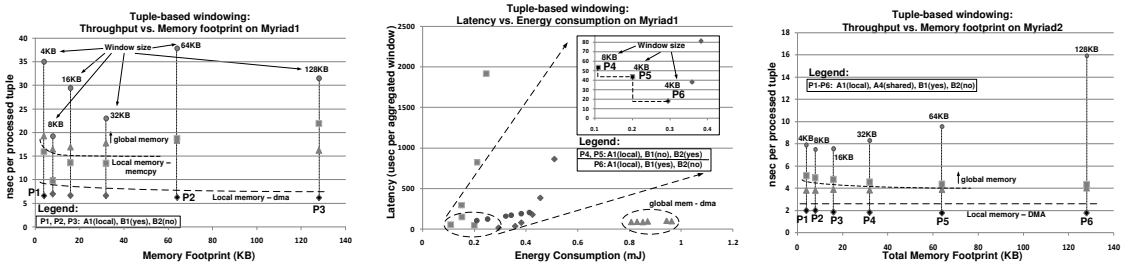
χαμηλότερη κατανάλωση ενέργειας με επιλογή της *P6*.

Σχετικά με το scalability των βέλτιστων Pareto υλοποιήσεων, παρατηρούμε στο Σχήμα 5.10γ΄ ότι το throughput αυξάνει ως τις 6 ουρές και στη συνέχεια παρουσιάζει μικρή μείωση. Όπως και στα πειράματα στη Myria1, οι υλοποιήσεις με χαμηλότερο μέγεθος ουράς τείνουν να παρουσιάζουν χαμηλότερο throughput.

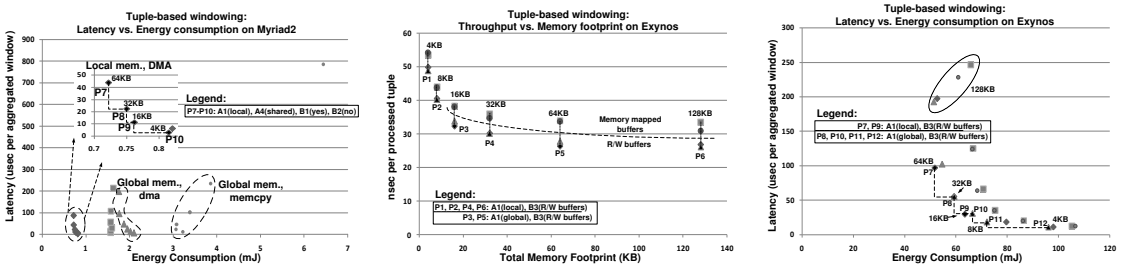
Στο δεύτερο πείραμα, όπου η δομή δεδομένων των παραθύρων τοποθετείται στην global μνήμη, το latency threashold είναι στα 62usec (Σχήμα Fig. 5.9β΄). Τα αποτελέσματα throughput vs. μέγεθος μνήμης απεικονίζονται στο Σχήμα 5.10δ΄, όπου εντοπίστηκαν 4 βέλτιστες Pareto υλοποιήσεις. Στο Σχήμα 5.10ε΄ εντοπίστηκαν 5 υλοποιήσεις που δείχνουν trade-offs μεταξύ latency και κατανάλωσης ενέργειας. Για παράδειγμα, επιλέγοντας την υλοποίηση *P20* το latency μειώνεται κατά 8,59% σε σχέση με την *P16*. Αντίθετα, επιλέγοντας την *P16*, η κατανάλωση ενέργειας είναι μικρότερη κατά 18% σε σχέση με την *P20*. Τα αποτελέσματα του scalability στο Σχήμα 5.10ϛ΄ είναι παρόμοια με αυτά του προηγούμενου πειράματος. Το throughput αυξάνει ως τις 8 ουρές, ενώ στις περισσότερες περιπτώσεις τείνει να μειώνεται ελαφρώς στις 9.

Εφαρμογή στο I.MX.6 Quad

Ο αριθμός των σχεδιαστικών επιλογών για το I.MX.6 είναι σχετικά μικρός, καθότι το λειτουργικό σύστημα χειρίζεται τις χαμηλού επιπέδου λειτουργίες. Από το Σχήμα 5.11α΄ επιλέχθηκε ένα μέγεθος ουράς και εξετάστηκαν τέσσερεις υλοποιήσεις σε κάθε πείραμα. Τα αποτελέσματα throughput vs. μέγεθος μνήμης απεικονίζονται στο Σχήμα 5.11β΄, ενώ latency vs. κατανάλωση ενέργειας στο Σχήμα 5.11γ΄. Παρατηρούμε ότι οι πιο αποδοτικές υλοποιήσεις με κριτήριο την απόδοση και τη μειωμένη κατανάλωση βασίζονται σε semaphores για συγχρονισμό κατά τη μεταφορά δεδομένων και σε freelists για τη διαχείριση μνήμης.



(α′) Throughput στη Myriad1 **(β′)** Latency vs. κατανάλωση ενέργειας στη Myriad1 **(γ′)** Throughput στη Myriad2



(δ′) Latency vs. κατανάλωση ενέργειας στη Myriad2 **(ε′)** Throughput στην Exynos **(ς′)** Latency vs. κατανάλωση ενέργειας στην Exynos

Σχήμα 5.12: Count-based streaming aggregation υλοποιήσεις.

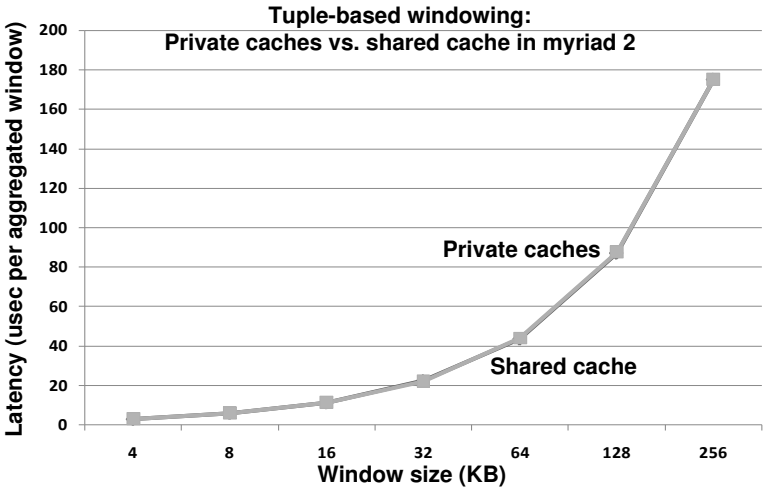
5.5.4 Αποτελέσματα για count-based aggregation

Στο count-based σενάριο εξετάσαμε κάθε παραμετροποιημένη υλοποίηση για διαφορετικά μεγέθη παραθύρων. Σε κάθε πλατφόρμα εξετάστηκαν 24 διαφορετικές υλοποιήσεις.

Εφαρμογή στη Myriad1

Στο Σχήμα 5.12α′ απεικονίζονται τα αποτελέσματα throughput vs. μέγεθος μνήμης στη Myriad1. Οι υλοποιήσεις που μεταφέρουν τα δεδομένα από την global στην local μνήμη και επεξεργάζονται σε αυτήν τα tuples παρέχουν υψηλότερο throughput. Για παράδειγμα, με μέγεθος παραθύρου στα 4KB, η *P1* παρέχει 58% υψηλότερο throughput σε σχέση με την υλοποίηση που χρησιμοποιεί *memcpy* για μεταφορά δεδομένων.

Τα αποτελέσματα latency vs. κατανάλωση ενέργειας παρουσιάζονται στο Σχήμα 5.12β′. Παρατηρούμε ότι μικρά μεγέθη παραθύρων παρέχουν χαμηλό latency, ενώ η μεταφορά tuples στην τοπική μνήμη και η επεξεργασία τους σε αυτήν είναι περισσότερο αποδοτική σε σχέση με την επεξεργασία τους στην global μνήμη. Εντοπίστηκαν τρεις βέλτιστες Pareto υλοποιήσεις που παρέχουν trade-offs μεταξύ των δύο μετρικών.



Σχήμα 5.13: Latency vs. μέγεθος παραθύρου στη Myriad2 για count-based streaming aggregation.

Εφαρμογή στη Myriad2

Στη Myriad2 εξετάστηκε το latency για διαφορετικά μεγέθη παραθύρων και διαφορετικές ρυθμίσεις της cache (ιδιωτικές και κοινόχρηστη). Όπως απεικονίζεται στο Σχήμα 5.13, η χρήση κοινόχρηστης cache παρέχει ελαφρώς χαμηλότερο latency (λιγότερο από 1%) και επομένως παρουσιάζονται παρακάτω αποτελέσματα αποκλειστικά με χρήση κοινόχρηστης cache μνήμης.

Όπως και στη Myriad1, οι υλοποιήσεις που παρέχουν υψηλότερο throughput είναι αυτές στις οποίες τα tuples μεταφέρονται μέσω DMA στην τοπική μνήμη και η επεξεργασία τους πραγματοποιείται σε αυτήν. Στο Σχήμα 5.12γ΄ φαίνεται πως το throughput αυξάνει ως 59% με χρήση των συγκεκριμένων σχεδιαστικών επιλογών σε σχέση με την υλοποίηση όπου η επεξεργασία των tuples γίνεται στην global μνήμη για μέγεθος παραθύρου 64KB. Επιπλέον, παρατηρούμε ότι μεγαλύτερα παράθυρα παρέχουν ελαφρώς υψηλότερο throughput. Για παράδειγμα, αύξηση του μεγέθους των παραθύρων από 4KB σε 128KB έχει ως συνέπεια την αύξηση του throughput κατά περίπου 10% (υλοποιήσεις *P1 – P6*).

Οι υλοποιήσεις που μεταφέρουν τα δεδομένα μέσω DMA στην τοπική μνήμη παρέχουν χαμηλό latency σε συνδυασμό με χαμηλή κατανάλωση ενέργειας, όπως απεικονίζεται στο Σχήμα Fig. 5.12δ΄. Για παράδειγμα, οι συγκεκριμένες σχεδιαστικές επιλογές σε παράθυρο μεγέθους 4KB παρέχουν 31,4% χαμηλότερη κατανάλωση ενέργειας σε σχέση με την αντίστοιχη υλοποίηση όπου η επεξεργασία των tuples γίνεται στην global μνήμη.

Εφαρμογή στην Exynos 5

Τα αποτελέσματα throughput vs. μέγεθος μνήμης απεικονίζονται στο Σχήμα 5.12ε΄. Παρατηρούμε πως μεγαλύτερα μεγέθη παραθύρων οδηγούν σε αυξημένο throughput. Οι υλοποιήσεις που κάνουν χρήση R/W buffers παρέχουν υψηλότερη απόδοση σε σχέση με αυτές που κάνουν χρήση memory mapped buffers (ως 21% για μεγέθος παραθύρου 64KB).

Όσο αφορά τα αποτελέσματα για latency vs. κατανάλωση ενέργειας που παρουσιάζονται στο Σχήμα 5.12ϛ΄, εντοπίστηκαν 6 Pareto υλοποιήσεις. Παρατηρούμε ότι παράθυρα μικρότερου μεγέθους παρέχουν χαμηλότερο latency, αλλά και υψηλότερη κατανάλωση ενέργειας εξαιτίας του αυξημένου ρυθμού μεταφορών δεδομένων. Η χρήση R/W buffers είναι πιο αποδοτική σε σχέση με τα memory mapped buffers, τόσο όσο αφορά το latency, όσο και την κατανάλωση ενέργειας. Εξαιτίας του σχετικά μικρού μεγέθους των buffers, το overhead της αντιγραφής τους από την global στην local μνήμη της GPU είναι σχετικά μικρό.

5.5.5 Αποτελέσματα Απόδοσης ανά watt

Ένας από τους στόχους της συγκεκριμένης μελέτης είναι η σύγκριση της απόδοσης ανά watt των υλοποιήσεων στις ενσωματωμένες αρχιτεκτονικές, σε σχέση με τις αντίστοιχες υλοποιήσεις σε HPC σύστημα και σε GPGPU. Σε αυτή την ενότητα, αρχικά θα δοθούν πληροφορίες σε σχέση με την υλοποίηση του operator στις συγκεκριμένες πλατφόρμες και έπειτα θα παρουσιαστούν τα πειραματικά αποτελέσματα.

Το σενάριο του time-based streaming aggregator υλοποιήθηκε σε έναν 8-πύρηνο Intel Xeon E5 CPU με 8 πυρήνες, οι οποίοι λειτουργούν στα 3.4GHz. Το μέγεθος της RAM είναι 16GB και το λειτουργικό συστημα είναι Ubuntu Linux 12.04. Ο compiler είναι ο gcc v.4.9.2 και η εκτέλεση έγινε με optimization flag *-O3*. Η κατανάλωση ενέργειας μετρήθηκε με συσκευή που εκτελεί δειγματοληψία, προσαρμοσμένη στην CPU και μετρά τη δυναμική ισχύ. Τα throughput και latency μετρήθηκαν με μεθόδους ίδιες με αυτές των ενσωματωμένων αρχιτεκτονικών. Η μεταφορά δεδομένων έγινε με χρήση *memcpy* system calls, ενώ ο συγχρονισμός βασίστηκε σε semaphores.

Τα αποτελέσματα απεικονίζονται στον Πίνακα 5.5. Οι τιμές για τις Myriad1, Myriad2 και I.MX.6 αντιστοιχούν στην υλοποίηση που παρέχει τα καλύτερα αποτελέσματα για τη συγκεκριμένη μετρική. Για να εξασφαλιστεί δίκαιη σύγκριση, όλες οι υλοποιήσεις δέχονται δεδομένα από 5 ουρές. Η απόδοση ανά watt μετράται ως tuples ανά δευτερόλεπτο και ανά watt.

Στα αποτελέσματα του Πίνακα 5.5 παρατηρούμε ότι όσο αφορά την απόδοση, το latency στον Intel Xeon είναι 62.3% χαμηλότερο σε σχέση με την Myriad2, ενώ είναι 3,8 και 9,3 φορές χαμηλότερο σε σχέση με την I.MX.6 και την Myriad1 αντίστοιχα. Όσο αφορά το throughput, ο Xeon παρέχει περισσότερο από 2 φορές υψηλότερο throughput σε σχέση με τη Myriad2, 2,8 περισσότερο σε σχέση με την I.MX.6 και 8,3

Πίνακας 5.5: Time-based streaming aggregation: Σύγκριση μεταξύ των latency, throughput και απόδοσης ανά watt στις ενσωματωμένες αρχιτεκτονικές και στην Intel Xeon.

	Latency (usec)	Throughput (t/sec)	(t/sec)/watt
Myriad1	140.38	132,622	379,041
Myriad2	39.8	497,154	1,004,766
I.MX.6	58	384,952	446,787
Xeon	15	1,105,221	18,412

Πίνακας 5.6: Count-based streaming aggregation: Σύγκριση μεταξύ των latency, throughput και απόδοσης ανά watt στις ενσωματωμένες αρχιτεκτονικές και στην Radeon HD 6450 GPGPU.

	Latency (usec)	Throughput (Mt/sec)	(Mt/sec)/watt
Myriad1	17.98	151.8	593
Myriad2	3.04	505.4	1286
Exynos	7.5	47.4	7,93
GPGPU	1.94	2576.3	85.87

φορές υψηλότερο σε σχέση με τη Myriad1. Η υψηλή απόδοση του Intel Xeon αποδίδεται στην πολύ υψηλότερη επεξεργαστική ισχύ και στο γεγονός ότι λειτουργεί σε πολύ υψηλότερη συχνότητα σε σχέση με τις ενσωματωμένες αρχιτεκτονικές. Παρόλα αυτά, όσο αφορά την απόδοση ανά watt, οι ενσωματωμένες πλατφόρμες παρέχουν καλύτερα αποτελέσματα. Εξαιτίας του γεγονότος ότι απαιτούν πολύ μικρή ισχύ, επιτυγχάνουν υψηλότερη απόδοση ανά watt: 54 φορές υψηλότερη η Myriad2 και 20 φορές η Myriad1. Τέλος, η I.MX6 παρέχει 24 φορές υψηλότερη απόδοση ανά watt σε σχεση με τον Intel Xeon.

Το count-based aggregation σενάριο υλοποιήθηκε σε OpenCL1.1 και εκτελέστηκε σε AMD Radeon HD 6450 GPGPU [82]. Ο host εκτελεί Ubuntu 12.04 με gcc v.4.9.2 και optimization flag *-O3*. Τα throughput και latency μετρήθηκαν με μεθόδους ίδιες με αυτές των ενσωματωμένων αρχιτεκτονικών, ενώ το power εκτιμήθηκε βάση των χαρακτηριστικών της συγκεκριμένες GPU. Η πρόσβαση των δεδομένων του host από τη GPU έγινε με χρήση R/W buffers.

Τα αποτελέσματα απεικονίζονται στο Πίνακα 5.6. Οι ενσωματωμένες αρχιτεκτονικές παρέχουν χαμηλότερο throughput και υψηλότερο latency σε σχέση με την Radeon GPGPU. Παρόλα αυτά, οι Myriad παρέχουν υψηλότερη απόδοση ανά watt εξαιτίας της πολύ χαμηλής ισχύος που απαιτούν. Πιο συγκεκριμένα, η Myriad2 παρέχει σχεδόν

14 φορές, ενώ η Myriad1 7 φορές υψηλότερη απόδοση ανά watt.

5.5.6 Ανάλυση των πειραματικών αποτελεσμάτων

Σε αυτή την ενότητα συνοψίζονται τα συμπεράσματα που προκύπτουν από τα πειραματικά αποτελέσματα που παρουσιάστηκαν στις προηγούμενες ενότητες. Τα trade-offs που εντοπίστηκαν από την υλοποίηση των σεναρίων μπορούν να χρησιμοποιηθούν για να εξαχθούν χρήσιμα συμπεράσματα για τις σχέσεις που υπάρχουν μεταξύ των σχεδιαστικών επιλογών και των μετρικών.

Συμπεράσματα από την υλοποίηση του time-based streaming aggregation

Παρατήρηση 1: Το streaming aggregation θα πρέπει να υλοποιηθεί με διαφορετικό τρόπο, όχι μόνο μεταξύ της I.MX.6 Quad και των αρχιτεκτονικών Myriad, αλλά και μεταξύ των Myriad1 και Myriad2.
Για παράδειγμα, στο πρώτο πείραμα στη Myriad1, στην υλοποίηση που παρέχει το μικρότερο latency, η μεταφορά των δεδομένων μεταξύ των φάσεων γίνεται με χρήση hardware buffers. Αντίθετα, στη Myriad2 γίνεται με χρήση busy-waiting μηχανισμού. Στην υλοποίηση που παρέχει το υψηλότερο throughput, το μέγεθος των ουρών είναι 256B στη Myriad1, ενώ είναι 512B στη Myriad2.

Παρατήρηση 2: Υπάρχει ένα όριο στο μέγεθος των ουρών κάτω από το οποίο το latency αυξάνει απότομα. Επιπλέον, πολύ μεγάλα μεγέθη ουράς έχουν επίσης αρνητική επίπτωση στο latency.
Παρατηρούμε ότι τόσο στη Myriad όσο και στο I.MX.6, το latency είναι πολύ υψηλό για μικρά μεγέθη ουρών, γεγονός που οφείλεται στο overhead που υπάρχει κατά τη συνεχή μεταφορά δεδομένων σε raw μορφή από την DDR στη CMX για τη δημιουργία tuples και την αποθήκευσή τους στις ουρές. Σε αυτές τις περιπτώσεις, το νήμα που υλοποιεί την φάση *Merge-Sort* βρίσκει συχνά τις ουρές άδειες. Καθώς το μέγεθος των ουρών αυξάνει, το latency μειώνεται δραστικά. Παρόλα αυτά, στις αρχιτεκτονικές της Myriad βλέπουμε ότι από κάποιο σημείο κι έπειτα, περαιτέρω αύξηση των ουρών προκαλεί αύξηση του latency (Σχήμα 5.7 και Σχήμα 5.9). Ο λόγος είναι το γεγονός ότι όσο μεγαλύτερες είναι οι ουρές, τόσο μεγαλύτερος αριθμός κύκλων απαιτείται για την μεταφορά των δεδομένων σε raw μορφή από την DDR στη CMX, για τη δημιουργία των tuples και την έναρξη τοποθέτησής τους στην ουρά. Έτσι, τα tuples που εισέρχονται στη φάση του *Update* πριν από μία μεταφορά δεδομένων από την DDR στη CMX και εξέρχονται μετά από τη μεταφορά έχουν υψηλότερο latency σε σχέση με τα υπόλοιπα tuples. Σε αντίθεση με τις Myriad, στο I.MX.6 μπορούν να χρησιμοποιηθούν πολύ μεγαλύτερες ουρές, καθότι η διαθέσιμη μνήμη είναι πολύ μεγαλύτερη. Παρόλα αυτά, πάνω από ένα όριο το throughput και το latency δεν δείχνουν να επηρεάζονται ιδιαίτερα (Σχήμα 5.11α').

Παρατήρηση 3: Το throughput επηρεάζεται κατά κύριο λόγο είτε από τον μηχανισμό μεταφοράς δεδομένων από την DDR στη CMX (στη Myriad2) είτε από τον μηχανισμό συγχρονισμού μεταξύ των πυρήνων κατά τη μεταφορά δεδομένων μεταξύ των φάσεων (στη Myriad1).
Γενικά, τόσο στη Myriad1 όσο και στη Myriad2, το throughput μειώνεται όταν το μέγεθος των ουρών μικραίνει, εξαιτίας της αύξησης της συχνότητας του overhead της μεταφοράς δεδομένων από την global στην local μνήμη (Σχήμα 5.8α΄ και Σχήμα 5.10α΄). Παρόλα αυτά, το latency γίνεται μικρότερο σε αυτή την περίπτωση, καθώς ο χρόνος που ένα tuple παραμένει στην queue πριν εξυπηρετηθεί, μειώνεται. Στη Myriad2, το throughput επηρεάζεται κυρίως από τον μηχανισμό μεταφοράς των δεδομένων από την DDR στην CMX (*memcpy* ή DMA). Η σχεδιαστική αυτή απόφαση έχει την μεγαλύτερη επίδραση στο throughput από κάθε άλλη (Σχήμα 5.10α΄ και Σχήμα 5.10δ΄). Αντίθετα, στη Myriad1, ο βασικός παράγοντας που επηρεάζει το throughput είναι ο μηχανισμός συγχρονισμού μεταξύ των πυρήνων, δηλαδή hardware buffers ή busy waiting (Σχήμα 5.8α΄ και Σχήμα 5.8δ΄). Στη Myriad2 η συγκεκριμένη σχεδιαστική απόφαση φαίνεται πως έχει μικρή επίδραση στο throughput, ενώ στη Myriad1 ο μηχανισμός μεταφοράς δεδομένων φαίνεται πως δεν επηρεάζει ουσιαστικά τη μετρική αυτή. (Γενικά, ο μηχανισμός *memcpy* φαίνεται να είναι οριακά πιο αποδοτικός στη Myriad1 σε σχέση με τις μεταφορές DMA). Στην I.MX.6 η χρήση των freelists ως μηχανισμό διαχείρισης της μνήμης για την αποφυγή των συχνών system calls αυξάνει τόσο το throughput, όσο και το latency. Παρόλα αυτά, ο βασικός παράγοντας που αυξάνει την απόδοση είναι η χρήση semaphores αντί για monitors (Σχήμα 5.11β΄).

Παρατήρηση 4: Το latency επηρεάζεται σημαντικά από τον μηχανισμό συγχρονισμού των πυρήνων κατά τη μεταφορά δεδομένων μεταξύ των φάσεων. Διαφορετικές μέθοδοι πρέπει να χρησιμοποιούνται στη Myriad1 σε σχέση με τη Myriad2.
Ο μηχανισμός συγχρονισμού είναι η βασική σχεδιαστική παράμετρος που επηρεάζει το latency και την κατανάλωση ενέργειας στις δύο αρχιτεκτονικές Myriad. Οι μηχανισμοί busy waiting παρέχουν χαμηλότερο latency στη Myriad2 και ελαφρώς χαμηλότερη κατανάλωση ενέργειας. Αντίθετα, η χρήση hardware buffers στη Myriad1 είναι πιο αποδοτική από άποψη latency. Οι μέθοδοι μεταφοράς δεδομένων από την DDR στην CMX έχουν μικρότερη επίδραση στο latency και στην κατανάλωση ενέργειας σε σχέση με τη μέθοδο συγχρονισμού και στις 2 αρχιτεκτονικές, όσο αφορά το latency και την κατανάλωση ενέργειας.

Παρατήρηση 5: Η συχνότητα με την οποία πραγματοποιούνται οι μεταφορές δεδομένων από την global στην local μνήμη επηρεάζει την κατανάλωση ενέργειας στη Myriad.
Παρατηρούμε ότι υλοποιήσεις με μεγαλύτερα μεγέθη ουρών είναι πιο αποδοτικές όσο αφορά την κατανάλωση ενέργειας στη Myriad1 και στη Myriad2, εξαιτίας του μειωμένου ρυθμού με τον οποίο τα δεδομένα μεταφέρονται στην local μνήμη (Σχήμα 5.8β΄ και Σχήμα 5.10β΄). Αντίθετα, στο I.MX.6 Quad εξαρτάται κατά κύριο λόγο από

τον μηχανισμό συγχρονισμού κάθε υλοποίησης.

Τέλος, μία ενδιαφέρουσα παρατήρηση είναι το γεγονός ότι η τοποθέτηση των ουρών στην local ή στη global μνήμη φαίνεται να επηρεάζει σε πολύ μικρό βαθμό την απόδοση και την κατανάλωση ενέργειας στη Myriad. Ο λόγος είναι το γεγονός ότι και οι δύο αρχιτεκτονικές παρέχουν cache μνήμη και ο ρυθμός των cache misses κατά την τοποθέτηση των ουρών στην global μνήμη είναι σχετικά μικρός. Έτσι, η τοποθέτησή τους στην DDR δεν μειώνει σε μεγάλο βαθμό την απόδοση. Αντίθετα, η τοποθέτηση της δομής δεδομένων των παραθύρων στη global μνήμη μειώνει δραστικά την απόδοση, αυξάνοντας παράλληλα την κατανάλωση ενέργειας. Για παράδειγμα, στη Myriad2, η τοποθέτηση της δομής δεδομένων των παραθύρων στην global μνήμη επιφέρει αύξηση του latency κατά 9%, μείωση του throughput κατά 7% και αύξηση της κατανάλωσης ενέργειας κατά 20% σε σχέση με την τοποθέτηση στη local μνήμη.

Οι παραπάνω παρατηρήσεις μπορούν να χρησιμοποιηθούν για να εξαχθούν γενικότερα συμπεράσματα σχετικά με τον τρόπο με τον οποίο το streaming aggregation θα πρέπει να παραμετροποιηθεί στις συγκεκριμένες αρχιτεκτονικές. Όταν ο στόχος είναι η αυξημένη απόδοση, τότε τα παρακάτω συμπεράσματα θα πρέπει να ληφθούν υπόψη:

- Το μέγεθος των ουρών θα πρέπει να είναι αρκετά μεγάλο, ώστε ο ρυθμός με τον οποίο πραγματοποιούνται μεταφορές δεδομένων από την global στη local μνήμη, να είναι σχετικά μικρός. Μικρού μεγέθους μεταφορές δεδομένων με αυξημένη συχνότητα μειώνουν την απόδοση. Παρόλα αυτά, σε υλοποιήσεις που είναι ευαίσθητες ως προς το latency, πρέπει να αποφεύγονται μεγάλα μεγέθη ουράς, γιατί μπορεί να το επηρεάσουν αρνητικά.

- Οι τιμές *window size* και *advance* επηρεάζουν το μέγεθος της δομής δεδομένων των παραθύρων και επομένως το αν θα τοποθετηθεί στη global ή στη local μνήμη. Η σχεδιαστική αυτή επιλογή επηρεάζει σε μεγάλο βαθμό την απόδοση. Πλατφόρμες με πολύ μικρή local μνήμη και σχετικά χαμηλής απόδοσης global μνήμη μπορεί να μην είναι κατάλληλες για υλοποιήσεις streaming aggregation, γιατί θα μείωναν σε μεγάλο βαθμό τις διαθέσιμες τιμές *size* και *advance*.

- Platform-specific σχεδιαστικά χαρακτηριστικά που ευνοούν την ανταλλαγή μηνυμάτων μεταξύ των πυρήνων (όπως τα hardware buffers της Myriad) θα πρέπει να αξιοποιούνται σε υλοποιήσεις που γίνονται σε χαμηλό επίπεδο. Σε κάποιες περιπτώσεις, όπως στη Myriad1, παρέχουν υψηλή απόδοση.

Από την άλλη πλευρά, αν ο βασικός σχεδιαστικός στόχος είναι η χαμηλή κατανάλωση ενέργειας, θα πρέπει να λαμβάνονται υπόψη τα εξής:

- Το μέγεθος των ουρών θα πρέπει να είναι όσο το δυνατόν μεγαλύτερο, ώστε να αποφεύγεται το overhead της κατανάλωσης ενέργειας που προκαλείται από

συχνές μεταφορές μικρής ποσότητας δεδομένων μεταξύ της global και της local μνήμης.

- Τιμές *size* και *advance* που οδηγούν σε μεγάλα μεγέθη της δομής δεδομένων των παραθύρων και αναγκάζουν την τοποθέτησή του στη global μνήμη, προκαλούν υψηλή κατανάλωση ενέργειας.

- Τέλος, οι προγραμματιστές θα πρέπει να αξιοποιούν χαρακτηριστικά των αρχιτεκτονικών που θέτουν τα PEs σε χαμηλή κατανάλωση ενέργειας όταν είναι αδρανή (όπως τα hardware buffers στη Myriad1).

Συμπεράσματα από την υλοποίηση του count-based streaming aggregation

Παρατήρηση 1: Τόσο το throughput, όσο και το latency στις υλοποιήσεις στη Myriad επηρεάζονται κατά κύριο λόγο από αν το παράθυρο είναι αποθηκευμένο στη global ή στη local μνήμη. Στο Exynos εξαρτώνται από τη μέθοδο πρόσβασης στα δεδομένα. Γενικά, το throughput επηρεάζεται από το μέγεθος του παραθύρου. Πέρα από αυτό, οι σχεδιαστικές επιλογές όπως η τοποθέτησή του στη local μνήμη και η χρήση OpenCL R/W buffers, οδηγούν σε αυξημένο throughput. Σε αντίθεση με το throughput, μικρά μεγέθη παραθύρων παρέχουν μειωμένο latency. Όπως και για το throughput, υλοποιήσεις όπου η επεξεργασία των tuples γίνεται στη local μνήμη και χρησιμοποιούν R/W buffers παρέχουν μειωμένο latency.

Παρατήρηση 2: Η κατανάλωση ενέργειας εξαρτάται από τα χαρακτηριστικά της μνήμης στην οποία αποθηκεύεται το παράθυρο και από το μέγεθός του.
Η κατανάλωση ενέργειας στη Myriad εξαρτάται τόσο από τον τύπο της μνήμης στην οποία επεξεργάζονται τα tuples, όσο και από το μέγεθος του παραθύρου (Σχήμα 5.12δ΄). Στο Exynos, το μέγεθος του παραθύρου έχει την μεγαλύτερη επίδραση στην κατανάλωση ενέργειας (Σχήμα 5.12ϛ΄). Ο ρυθμός με τον οποίο γίνονται οι μεταφορές δεδομένων από την global στην local μνήμη είναι αυξημένος για μικρά μεγέθη παραθύρων, οπότε στην περίπτωση αυτή αυξάνει η κατανάλωση ενέργειας.

Συνοψίζοντας, όταν ο σχεδιαστικός στόχος είναι η αυξημένη απόδοση, η χρήση DMA μεταφορών δεδομένων και R/W OpenCL buffers παρέχουν υλοποιήσεις με υψηλότερο throughput σε σχέση με τις υπόλοιπες σχεδιαστικές επιλογές. Μεγάλου μεγέθους παράθυρα οδηγούν σε αυξημένο throughput, ενώ μικρότερου μεγέθους παρέχουν μειωμένο latency. Τέλος, μεγέθη παραθύρων τα οποία μπορούν να αποθηκευτούν στη local μνήμη ευνοούν τόσο την απόδοση, όσο και την κατανάλωση ενέργειας.

Η μεθοδολογία που προτείνεται παρέχει έναν συστηματικό τρόπο με τον οποίο μπορεί να παραμετροποιηθούν οι εφαρμογές streaming aggregation που εκτελούνται σε ενσωματωμένες αρχιτεκτονικές. Αντί για τη βελτιστοποίηση μιας υλοποίησης streaming aggregation με αυθαίρετο τρόπο, η μεθοδολογία παρουσιάζει τις σχεδιαστικές παραμέτρους της εφαρμογής που θα πρέπει να παραμετροποιηθούν κατάλληλα,

ώστε να ικανοποιηθούν οι σχεδιαστικοί περιορισμοί. Έτσι, παρέχει ένα σύνολο από υλοποιήσεις και ο σχεδιαστής, λαμβάνοντας υπόψη τους σχεδιαστικούς περιορισμούς, μπορεί να επιλέξει αυτήν που θεωρεί καταλληλότερη.

Κεφάλαιο 6

Συμπεράσματα και μελλοντικές προοπτικές

Στο κεφάλαιο αυτό παρουσιάζεται μια επισκόπηση των κυριοτέρων συνεισφορών της παρούσας διατριβής συνοψίζοντας τις πρωτοτυπίες της, καθώς και πιθανές μελλοντικές προεκτάσεις.

6.1 Επισκόπηση της διδακτορικής διατριβής

Στην παρούσα διατριβή παρουσιάστηκε μία μεθοδολογία που στοχεύει στην βελτιστοποίηση εφαρμογών που κάνουν χρήση ταυτόχρονων δομών δεδομένων. Αποτελεί μια συνεισφορά στη λύση του προβλήματος της επιλογής κατάλληλων ταυτόχρονων δομών δεδομένων.

Συνοψίζοντας τη συνεισφορά της, αρχικά, ορίστηκε ο χώρος σχεδιαστικών επιλογών των ταυτόχρονων δομών δεδομένων, που αποτελούν τα δομικά στοιχεία βάση των οποίων σχηματίζονται υλοποιήσεις των ταυτόχρονων δομών. Στη συνέχεια εντοπίστηκαν οι περιορισμοί τόσο της εφαρμογής, όσο και της αρχιτεκτονικής οι οποίοι μετατρέπουν τον χώρο σχεδιαστικών επιλογών από ανεξάρτητο της αρχιτεκτονικής και της εφαρμογής σε εξαρτημένο. Περιγράφηκαν τα βήματα της μεθοδολογίας επιλογής ταυτόχρονων δομών δεδομένων. Για την εφαρμογή της αναπτύχθηκε μια βιβλιοθήκη ταυτόχρονων δομών δεδομένων, καθώς και ένα σύνολο εργαλείων που αυτοματοποιούν ένα σημαντικό αριθμό βημάτων της. Η προτεινόμενη μεθοδολογία εφαρμόστηκε σε 5 διαφορετικά benchmarks από όπου συμπεράναμε τη σημασία που έχει η επιλογή δομών δεδομένων σε έναν μεγάλο αριθμό από μετρικές των ενσωματωμένων συστημάτων. Πιο συγκεκριμένα, δείξαμε ότι επιλέγοντας διαφορετικές υλοποιήσεις δομών δεδομένων, υπάρχει η δυνατότητα να γίνουν trade-offs ανάμεσα στην απόδοση, την κατανάλωση ενέργειας του συστήματος και στο fairness της δομής.

Πέρα από τα παραπάνω, εξετάστηκε αναλυτικά ένα μοντέλο σχεδιασμού ταυτόχρονων δομών δεδομένων κατάλληλο για δομές που παρέχουν χαμηλό επίπεδο παραλληλισμού και εκτελούνται σε αρχιτεκτονικές με υποστήριξη message-passing μεταξύ των πυρήνων. Επιπλέον, η απεικόνιση του χώρου σχεδιασμού με μορφή δέντρων αποφάσεων, αξιοποιήθηκε για την παραμετροποίηση της υλοποίησης του aggregation operator σε έναν αριθμό από ενσωματωμένες πλατφόρμες.

6.2 Μελλοντικές προοπτικές

Η προτεινόμενη μεθοδολογία που παρουσιάστηκε στην παρούσα διατριβή, μπορεί να συνεισφέρει σε σημαντικό βαθμό στην επίλυση προβλημάτων που οφείλονται στη ολοένα αυξημένη πολυπλοκότητα που παρουσιάζουν τα σύγχρονα ενσωματωμένα συστήματα.

Η συγκεκριμένη μεθοδολογία αφορά την επιλογή υλοποίησης δομής δεδομένων κατά τον χρόνο σχεδιασμού. Η δομή παραμένει αναλλοίωτη όσο αφορά την υλοποίηση της κατά τον χρόνο εκτέλεσης της εφαρμογής. Ένα σημαντικό ζήτημα είναι η παραμετροποίηση των δομών δεδομένων στον χρόνο εκτέλεσης. Με άλλα λόγια, η τροποποίηση της υλοποίησης κατά το χρόνο εκτέλεσης της εφαρμογής, με στόχο να ανταποκριθεί αποτελεσματικότερα τόσο είτε στις ανάγκες της ίδιας της εφαρμογής (π.χ. μικρότερο latency), είτε του υλικού (π.χ. ανάγκη για εξοικονόμηση ενέργειας), είτε του χρήστη (π.χ. ανάγκη για υψηλότερη απόδοση). Στην περίπτωση που η δομή παρέχει τέτοια δυνατότητα (autotunning) σημαντικός παράγοντας που πρέπει να ληφθεί υπόψη είναι το overhead που μπορεί να υπάρξει από τους επιπλέον ελέγχους που απαιτούνται και την τροποποίηση της δομής κατά τη διάρκεια που αυτή συνεχίζει να εξυπηρετεί την εφαρμογή.

Παρόλο που η παρούσα μελέτη αφορά τον χώρο των ενσωματωμένων συστημάτων, η προτεινόμενη μεθοδολογία μπορεί να εφαρμοστεί και στον χώρο των HPC. Πιο συγκεκριμένα, από την εφαρμογή της μεθοδολογίας σε μεγάλα κέντρα δεδομένων (data centers) και τη βελτιστοποίηση των αντίστοιχων εφαρμογών που εκτελούνται σε αυτά θα μπορούσαν πιθανώς να προκύψουν σημαντικά κέρδη, τόσο όσο αφορά την απόδοση, όσο και την κατανάλωση ενέργειας. Εφαρμογές όπως το memcached ή αλγόριθμοι με μεγάλες υπολογιστικές απαιτήσεις (π.χ. σχετικοί με τη μελέτη γονιδιωμάτων) στις οποίες οι ταυτόχρονες δομές έχουν κεντρικό ρόλο, θα μπορούσαν να βελτιστοποιηθούν με χρήση της προτεινόμενης μεθοδολογίας.

Μια ακόμη ενδιαφέρουσα επέκταση είναι η μελέτη της παραμετροποίησης των δομών δεδομένων ώστε να υποβοηθείται η ασφάλεια των δεδομένων και η προστασία από το hacking. Είναι πιθανό συγκεκριμένοι συνδυασμοί δομών δεδομένων και αρχιτεκτονικών χαρακτηριστικών να ενισχύουν την προστασία των δεδομένων να οδηγούν σε τέτοια οργάνωση των δεδομένων στη μνήμη που να οδηγεί σε μεγαλύτερη ασφάλεια. Σε αυτό θα μπορούσε να βοηθήσει και ένας κατάλληλος διαχειριστή μνήμης προσανατολισμένος σε αυτό το σκοπό. Το ζήτημα της ασφάλειας και της προστασίας των δεδομένων είναι ιδιαίτερα σημαντικό στον χώρο το Internet of Things όπου τα ενσωματωμένα συστήματα έχουν κυρίαρχο ρόλο.

Σύντομο βιογραφικό

Ο Λάζαρος Παπαδόπουλος αποφοίτησε από τη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Δημοκρίτειου Πανεπιστήμιου Θράκης το 2005. Η διπλωματική του εργασία αφορούσε τις μεθοδολογίες βελτιστοποίησης των δυναμικών διαχειριστών μνήμης σε ενσωματωμένα συστήματα. Το 2008 ολοκλήρωσε τις σπουδές για μεταπτυχιακό δίπλωμα ειδίκευσης στο εργαστήριο Μικροηλεκτρονικής και Υπολογιστών στην ίδια σχολή, όπου ασχολήθηκε με μεθοδολογίες βελτιστοποίησης δυναμικών δομών δεδομένων και αρχιτεκτονικές Network-on-Chip. Από το 2010 είναι υποψήφιος διδάκτορας στη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών και ασχολείται με την υλοποίηση ταυτόχρονων δομών δεδομένων σε ενσωματωμένα συστήματα. Το 2013 επισκέφθηκε για 6 μήνες εργαστήριο του τμήματος Computer Science and Engineering του πανεπιστημίου Chalmers University of Technology όπου εργάστηκε στο ευρωπαικό project EXCESS. Τον Οκτώβριο του 2015, μέσω της υποτροφίας HiPEAC Collaboration Grant επισκέφθηκε το ίδιο εργαστήριο. Τα ερευνητικά του ενδιαφέροντα περιλαμβάνουν δυναμικές δομές δεδομένων, πολυπύρηνες αρχιτεκτονικές και υλοποίηση εφαρμογών σε ενσωματωμένα συστήματα με κριτήριο τη χαμηλή κατανάλωση ενέργειας.

Δημοσιεύσεις

περιοδικά

1. Lazaros Papadopoulos, Ivan Walulya, Philippas Tsigas, Dimitrios Soudris: A Systematic Methodology for Optimization of Applications Utilizing Concurrent Data Structures. IEEE Transactions on Computers (accepted for publication)

2. Lazaros Papadopoulos, Dimitrios Soudris, Ivan Walulya, Philippas Tsigas: Customization methodology for implementation of streaming aggregation in embedded systems. Journal of Systems Architecture 66 (2016): 48-60.

3. Lazaros Papadopoulos, Ivan Walulya, Paul Renaud-Goud, Philippas Tsigas, Dimitrios Soudris, Brendan Barry: Performance and power consumption evaluation of concurrent queue implementations in embedded systems. Computer Science-Research and Development 30.2 (2015): 165-175.

4. Alexandros Bartzas, Lazaros Papadopoulos, Dimitrios Soudris: A system-level design methodology for application-specific networks-on-chip. J. Embedded Computing 3(3): 167-177 (2009)

5. Christos Baloukas, José Luis Risco-Martín, David Atienza, Christophe Poucet, Lazaros Papadopoulos, Stylianos Mamagkakis, Dimitrios Soudris, José Ignacio Hidalgo, Francky Catthoor, Juan Lanchares: Optimization methodology of dynamic data structures based on genetic algorithms for multimedia embedded systems. Journal of Systems and Software 82(4): 590-602 (2009)

6. Lazaros Papadopoulos, Christos Baloukas, Dimitrios Soudris: Exploration methodology of dynamic data structures in multimedia and network applications for embedded platforms. Journal of Systems Architecture - Embedded Systems Design 54(11): 1030-1038 (2008)

συνέδρια

1. Thomas Papastergiou, Lazaros Papadopoulos, Dimitrios Soudris: Platform-aware dynamic data type refinement methodology for radix tree Data Structures. SAMOS 2015: 78-85

2. Lazaros Papadopoulos, Dimitrios Soudris: An Energy Efficient Message Passing Synchronization Algorithm for Concurrent Data Structures in Embedded Systems. SCOPES 2015: 113-116

3. Lazaros Papadopoulos, Ivan Walulya, Philippas Tsigas, Dimitrios Soudris, Brendan Barry: Evaluation of message passing synchronization algorithms in embedded systems. ICSAMOS 2014: 282-289

4. Lazaros Papadopoulos, Alexandros Bartzas, Dimitrios Soudris: Run-Time Dynamic Data Type Transformations. ARCS Workshops 2012: 351-362

5. Christos Baloukas, Lazaros Papadopoulos, Dimitrios Soudris, Sander Stuijk, Olivera Jovanovic, Florian Schmoll, Daniel Cordes, Robert Pyka, Arindam Mallik, Stylianos Mamagkakis, François Capman, Séverin Collet, Nikolaos Mitas, Dimitrios Kritharidis: Mapping Embedded Applications on MPSoCs: The MNEMEE Approach. ISVLSI 2010: 512-517

6. Christos Baloukas, Lazaros Papadopoulos, Robert Pyka, Dimitrios Soudris, Peter Marwedel: An automatic framework for dynamic data structures optimization in C. VLSI-SoC 2010: 155-160

7. Lazaros Papadopoulos, Christos Baloukas, Dimitrios Soudris, Konstantinos Potamianos, Nikolaos S. Voros: Data Structure Exploration of Dynamic Applications. PACT 2007: 421

8. Christos Baloukas, Lazaros Papadopoulos, Stylianos Mamagkakis, Dimitrios Soudris: Component Based Library Implementation of Abstract Data Types for Resource Management Customization of Embedded Systems. ESTImedia 2007: 99-104

9. Lazaros Papadopoulos, Stylianos Mamagkakis, Francky Catthoor, Dimitrios Soudris: Application - specific NoC platform design based on System Level Optimization. ISVLSI 2007: 311-316

10. Lazaros Papadopoulos, Dimitrios Soudris: System-Level Application-Specific NoC Design for Network and Multimedia Applications. PATMOS 2007: 1-9

11. Lazaros Papadopoulos, Christos Baloukas, Nikolaos Zompakis, Dimitrios Soudris: Systematic Data Structure Exploration of Multimedia and Network Applications realized Embedded Systems. ICSAMOS 2007: 58-65

12. David Atienza, Christos Baloukas, Lazaros Papadopoulos, Christophe Poucet, Stylianos Mamagkakis, José Ignacio Hidalgo, Francky Catthoor, Dimitrios Soudris, Juan Lanchares: Optimization of dynamic data structures in multimedia embedded systems using evolutionary computation. SCOPES 2007: 31-40

Βιβλιογραφία

[1] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM 2006: Formal Methods*. Springer, 2006, pp. 1–15.

[2] *Freescale I.MX 6 Quad application processors for industrial products data manual*, Freescale Semiconductor Inc., 2015.

[3] *Samsung Exynos 4 Quad (Exynos 4412) RISC Microprocessor User Manual, ver. 1.00*, Samsung Electronics Co. Ltd., 2012.

[4] *Rockchip RK3188 Tech. Reference Manual, ver. 1.2*, Fuzhou Rockchip Electronics Co. Ltd., 2013.

[5] *HummingBoard User Manual, ver. 1.3*, SolidRun Ltd., 2014.

[6] D. Moloney, "1tops/w software programmable media processor," *HotChips HC23*, 2011.

[7] "Imec: http://www2.imec.be/."

[8] C. Poucet, S. Mamagkakis, D. Atienza, and F. Catthoor, "Systematic intermediate sequence removal for reduced memory accesses," in *Proceedingsof the 10th international workshop on Software & compilers for embedded systems*. ACM, 2007, pp. 51–60.

[9] L. Papadopoulos, C. Baloukas, and D. Soudris, "Exploration methodology of dynamic data structures in multimedia and network applications for embedded platforms," *J. Systems Architecture*, vol. 54, no. 11, pp. 1030–1038, 2008.

[10] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, and K. Pekmestzi, "Custom multi-threaded dynamic memory management for multiprocessor system-on-chip platforms," in *SAMOS*. IEEE, 2010, pp. 102–109.

[11] F. Catthoor, S. Wuytack, G. de Greef, F. Banica, L. Nachtergaele, and A. Vandecappelle, *Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design*. Springer Science & Business Media, 2013.

[12] I. Chung *et al.*, "Towards automatic performance tuning," 2004.

[13] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

[14] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn, "Design and implementation of the logicblox system," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1371–1382.

[15] D. Gusfield, *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[16] A. Stepanov and M. Lee, *The standard template library*. Hewlett Packard Laboratories 1501 Page Mill Road, Palo Alto, CA 94304, 1995, vol. 1501.

[17] L. Liu and S. Rus, "Perflint: A context sensitive performance advisor for c++ programs," in *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2009, pp. 265–274.

[18] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[19] L. Benini, A. Macii, E. Macii, and M. Poncino, "Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation," *IEEE Design & Test of Computers*, no. 2, pp. 74–85, 2000.

[20] P. A. Green Jr, "The art of creating reliable software-based systems using off-the-shelf software components," in *srds*. IEEE, 1997, p. 118.

[21] M. Kandemir, J. Ramanujam, and A. Choudhary, "Improving cache locality by a combination of loop and data transformations," *Computers, IEEE Transactions on*, vol. 48, no. 2, pp. 159–167, 1999.

[22] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*. IEEE, 2002, pp. 409–415.

[23] "Generic data structures library - gdsl: http://home.gna.org/gdsl/."

[24] S. S. Muchnick, *Advanced compiler design implementation*. Morgan Kaufmann, 1997.

[25] C. Jung and N. Clark, "Ddt: design and evaluation of a dynamic program analysis for optimizing data structure usage," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 56–66.

[26] L. Liu and S. Rus, "Perflint: A context sensitive performance advisor for c++ programs," in *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2009, pp. 265–274.

[27] E. Schonberg, J. T. Schwartz, and M. Sharir, "An automatic technique for selection of data representations in setl programs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 3, no. 2, pp. 126–143, 1981.

[28] J. T. Schwartz, "Automatic data structure choice in a language of very high level," *Communications of the ACM*, vol. 18, no. 12, pp. 722–728, 1975.

[29] O. Shacham, M. Vechev, and E. Yahav, "Chameleon: adaptive selection of collections," *ACM Sigplan Notices*, vol. 44, no. 6, pp. 408–418, 2009.

[30] C. Jung, S. Rus, B. P. Railing, N. Clark, and S. Pande, "Brainy: effective selection of data structures," in *ACM SIGPLAN Notices*, vol. 46, no. 6. ACM, 2011, pp. 86–97.

[31] N. Mitchell and G. Sevitsky, "The causes of bloat, the limits of health," *ACM SIGPLAN Notices*, vol. 42, no. 10, pp. 245–260, 2007.

[32] N. Mitchell, E. Schonberg, and G. Sevitsky, "Four trends leading to java runtime bloat," *IEEE software*, vol. 27, no. 1, p. 56, 2010.

[33] G. Xu and A. Rountev, "Detecting inefficiently-used containers to avoid bloat," *ACM Sigplan Notices*, vol. 45, no. 6, pp. 160–173, 2010.

[34] G. Xu, N. Mitchell, M. Arnold, A. Rountev, and G. Sevitsky, "Software bloat analysis: finding, removing, and preventing performance problems in modern large-scale object-oriented applications," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 421–426.

[35] N. Shavit, "Data structures in the multicore age," *Communications of the ACM*, vol. 54, no. 3, pp. 76–84, 2011.

[36] H. Sundell and P. Tsigas, "Noble: non-blocking programming support via lock-free shared abstract data types," *ACM SIGARCH*, vol. 36, no. 5, pp. 80–87, 2009.

[37] K. Fraser, "Practical lock-freedom," Ph.D. dissertation, University of Cambridge, 2004.

[38] A. Gautham, K. Korgaonkar, P. Slpsk, S. Balachandran, and K. Veezhinathan, "The implications of shared data synchronization techniques on multi-core energy efficiency," in *Proc. of the 2012 USENIX Conf. on Power-Aware Computing and Systems*. USENIX Association, 2012, pp. 6–6.

[39] A. Skyrme and N. Rodriguez, "From locks to transactional memory: Lessons learned from porting a real-world application," in *the 8th ACM SIGPLAN Workshop on Transactional Computing*, 2013.

[40] T. Moreshet, R. I. Bahar, and M. Herlihy, "Energy-aware microprocessor synchronization: Transactional memory vs. locks," in *4th Annual Boston-Area Architecture Workshop*, 2006, p. 21.

[41] I. Calciu, D. Dice, T. Harris, M. Herlihy, A. Kogan, V. Marathe, and M. Moir, "Message passing or shared memory: Evaluating the delegation abstraction for multicores," in *Principles of Distributed Systems*. Springer, 2013, pp. 83–97.

[42] D. Dice, V. J. Marathe, and N. Shavit, "Lock cohorting: a general technique for designing numa locks," in *ACM SIGPLAN Notices*, vol. 47, no. 8. ACM, 2012, pp. 247–256.

[43] N. Hunt, P. S. Sandhu, and L. Ceze, "Characterizing the performance and energy efficiency of lock-free data structures," in *Interaction between Compilers and Comput. Archit. (INTERACT)*. IEEE, 2011, pp. 63–70.

[44] D. Dice, V. J. Marathe, and N. Shavit, "Lock cohorting: a general technique for designing numa locks," in *ACM SIGPLAN Notices*, vol. 47, no. 8. ACM, 2012, pp. 247–256.

[45] L. Papadopoulos, I. Walulya, P. Renaud-Goud, P. Tsigas, D. Soudris, and B. Barry, "Performance and power consumption evaluation of concurrent queue implementations in embedded systems," *Computer Science-Research and Development*, vol. 30, no. 2, pp. 165–175, 2014.

[46] D. Hendler, N. Shavit, and L. Yerushalmi, "A scalable lock-free stack algorithm," in *Proc. of the 16th annual ACM Symp. on Parallelism in algorithms and architectures*. ACM, 2004.

[47] M. M. Michael, "Hazard pointers: Safe memory reclamation for lock-free objects," *IEEE Trans. Parallel Distrib. Syst."*, vol. 15, no. 6, pp. 491–504, 2004.

[48] D. L. Detlefs, P. A. Martin, M. Moir, and G. L. Steele Jr, "Lock-free reference counting," *J. Distributed Computing*, vol. 15, no. 4, pp. 255–271, 2002.

[49] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Trans. VLSI Syst.*, vol. 10, no. 4, pp. 416–422, Aug. 2002.

[50] C. Ferri, S. Wood, T. Moreshet, R. I. Bahar, and M. Herlihy, "Embedded-tm: Energy and complexity-effective hardware transactional memory for embedded multicore systems," *J. Parallel and Distributed Computing*, vol. 70, no. 10, 2010.

[51] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proc. of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[52] D. Cederman, V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Brief announcement: concurrent data structures for efficient streaming aggregation." in *SPAA '14*. ACM, 2013, pp. 76–78.

[53] H. Sundell and P. Tsigas, "Lock-free and practical doubly linked list-based deques using single-word compare-and-swap," in *Principles of Distributed Systems*. Springer, 2005.

[54] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization*. IEEE, 2001, pp. 3–14.

[55] D. Cederman, B. Chatterjee, N. Nguyen, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "A study of the behavior of synchronization methods in commonly used languages and systems," in *IPDPS*. IEEE, 2013, pp. 1309–1320.

[56] A. Prokopec, N. G. Bronson, P. Bagwell, and M. Odersky, "Concurrent tries with efficient non-blocking snapshots," in *Acm Sigplan Notices*, vol. 47, no. 8. ACM, 2012, pp. 151–160.

[57] J.-P. Lozi, F. David, G. Thomas, J. Lawall, and G. Muller, "Remote core locking: Migrating critical-section execution to improve the performance of multithreaded applications," in *Proc. of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12, 2012, pp. 6–6.

[58] S. H. Kim, S. H. Lee, M. Jun, B. Lee, W. W. Ro, E.-Y. Chung, and J.-L. Gaudiot, "C–lock: Energy efficient synchronization for embedded multicore systems," *Computers, IEEE Transactions on*, vol. 63, no. 8, pp. 1962–1974, 2014.

[59] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Power/performance hardware optimization for synchronization intensive applications in mpsocs," in *Proc. of DATE*. European Design and Automation Association, 2006, pp. 606–611.

[60] R. Rajwar and J. R. Goodman, "Speculative lock elision: Enabling highly concurrent multithreaded execution," in *Proc. of the 34th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 2001, pp. 294–305.

[61] S. Sanyal, S. Roy, A. Cristal, O. S. Unsal, and M. Valero, "Clock gate on abort: Towards energy-efficient hardware transactional memory," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[62] P. Stanley-Marbell and V. C. Cabezas, "Performance, power, and thermal analysis of low-power processors for scale-out systems," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 863–870.

[63] K. Fürlinger, C. Klausecker, and D. Kranzlmüller, "Towards energy efficient parallel computing on consumer electronic devices," in *Information and Communication on Technology for the Fight against Global Warming*. Springer, 2011, pp. 1–9.

[64] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, "Tibidabo: Making the case for an arm-based hpc system," *Future Generation Computer Systems*, vol. 36, pp. 322–334, 2014.

[65] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *Micro, IEEE*, vol. 35, no. 1, pp. 6–14, 2015.

[66] S. Schneidert, H. Andrade, B. Gedik, K.-L. Wu, and D. S. Nikolopoulos, "Evaluation of streaming aggregation on parallel hardware architectures," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. ACM, 2010, pp. 248–257.

[67] C. Balkesen, N. Tatbul, and M. T. Özsu, "Adaptive input admission and management for parallel stream processing," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM, 2013, pp. 15–26.

[68] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 12, pp. 2351–2365, 2012.

[69] U. Verner, A. Schuster, and M. Silberstein, "Processing data streams with hard real-time constraints on heterogeneous systems," in *Proceedings of the international conference on Supercomputing*. ACM, 2011, pp. 120–129.

[70] D. Cederman, V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Brief announcement: Concurrent data structures for efficient streaming aggregation," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 2014, pp. 76–78.

[71] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *CIDR*, vol. 5, 2005, pp. 277–289.

[72] Y. Choi, Y. Lin, N. Chong, S. Mahlke, and T. Mudge, "Stream compilation for real-time embedded multicore systems," in *Code generation and optimization, 2009. CGO 2009. International symposium on*. IEEE, 2009, pp. 210–220.

[73] R. R. Newton, L. D. Girod, M. B. Craig, S. R. Madden, and J. G. Morrisett, "Design and evaluation of a compiler for embedded stream programs," in *ACM Sigplan Notices*, vol. 43, no. 7. ACM, 2008, pp. 131–140.

[74] "Movidius Ltd.: http://www.movidius.com."

[75] "Project Tango: https://www.google.com/atap/project-tango/."

[76] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, no. 2, pp. 56–66, 2015.

[77] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous multi-processing solution of exynos 5 octa with arm® big. little™ technology."

[78] "Hardkernel. Odroid-xu: http://www.hardkernel.com/."

[79] "SoundCloud: https://www.soundcloud.com."

[80] N. Hunt, P. S. Sandhu, and L. Ceze, "Characterizing the performance and energy efficiency of lock-free data structures," in *Interaction between Compilers and Computer Architectures (INTERACT), 2011 15th Workshop on*. IEEE, 2011, pp. 63–70.

[81] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[82] "AMD Radeon HD 6450 GPU: http://www.amd.com/en-us/products/graphics/desktop/6000/6450."

Αντιστοίχιση ελληνικών σε αγγλικούς όρους

Ελληνικός όρος	**Αγγλικός όρος**
αλληλεξάρτηση	interdependency
ατομικές λειτουργίες	atomic operations
δέντρα αποφάσεων	decision trees
δίαυλος	bus
εισερχόμενα tuples	inbound tuples
επεξεργασία ροών δεδομένων	stream processing
επεξεργαστικό στοιχείο	processing element
ερώτημα	query
εξερεύνηση χώρου σχεδιασμού	design space exploration
κατευθυνόμενοι άκυκλοι γράφοι	direct acyclic graphs
κλειδώματα	locks
κοινές δομές δεδομένων	shared data structures
μετρική	metric
νήμα	thread
ουρά	queue
στοιχεία συγχρονισμού	synchronization primitives
συμφόρηση	contention
συστήματα γενικού σκοπού	general purpose systems
σχήμα πρόσβασης	access pattern
σωρός	stack
ταυτόχρονες δομές δεδομένων	concurrent data structures
φορητότητα	portability
χώρος σχεδιαστικών επιλογών	design space