Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Αλγόριθμοι δρομολόγησης εφαρμογών σε επίπεδο υλικού για πολυνηματικές αρχιτεκτονικές.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΜΠΟΤΣΑΡΗΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Αλγόριθμοι δρομολόγησης εφαρμογών σε επίπεδο υλικού για πολυνηματικές αρχιτεκτονικές.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΜΠΟΤΣΑΡΗΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Ιουλίου 2009.

.....	.....	.....
Νεκτάριος Κοζύρης Αν. Καθηγητής Ε.Μ.Π.	Δημήτριος Σούντρης Επ. Καθηγητής Ε.Μ.Π.	Γιώργος Οικονομάκος Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

Κωνσταντίνος Μπότσαρης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι σύγχρονες αρχιτεκτονικές επεξεργαστών παρουσιάζουν πια ένα μεγάλο αριθμό νημάτων (threads) και μάλιστα σε περισσότερους από ένα πυρήνες (cores). Κάθε πυρήνας παρουσιάζεται με δύο ή και περισσότερα νήματα αυξάνοντας έτσι το βαθμό παραλληλοποίησης. Μέχρι πρόσφατα η πιο σημαντική χρήση αυτών των επεξεργαστών ήταν ως εξυπηρετητές εργασίας (job servers) που έτρεχαν τα διάφορα νήματα στους διάφορους SMT πυρήνες του επεξεργαστή. Με την εμφάνιση όμως του Intel i7 ο οποίος αποτελείται από 4 πυρήνες των 2 νημάτων παρατηρούμε ότι η βιομηχανία άρχισε να προωθεί αυτού του είδους επεξεργαστές για χρήση και στο σπίτι.

Η συνύπαρξη φυσικά περισσότερων του ενός νημάτων σε ένα πυρήνα συνεπάγεται και χρήση κοινών πόρων (μονάδων επεξεργασίας, κρυφής μνήμης) με αποτέλεσμα την εμφάνιση συγκρούσεων κατά την εκτέλεση εφαρμογών και τη μείωση της απόδοσης. Όλα αυτά λοιπόν καθιστούν αναγκαία τη δρομολόγηση των εφαρμογών(νημάτων) και την σωστή επιλογή για συνύπαρξή τους με βάση κάποια κριτήρια. Σκοπός αυτής της δπλωματικής εργασίας είναι η μελέτη και αξιολόγηση των διάφορων πολιτικών - αλγορίθμων δρομόλογησης των εφαρμογών στις πολυνηματικές αρχιτεκτονικές σε επίπεδο υλικού με ζητούμενο την βελτιστοποίηση της ρυθμαπόδοσης και γενικότερα της απόδοσης του συστήματος.

Η όλη έρευνά μας πραγματοποιήθηκε σε ένα διαμορφωμένο εξομοιωτή GEMS (General Execution-driven Multiprocessor Simulator) για υποστήριξη μίας CMT αρχιτεκτονικής με δυνατότητα μετανάστευσης νημάτων. Μελετήθηκαν τρεις αλγόριθμοι δρομολόγησης, ο DCCS (Data Cache Conflict Scheduler), IPCS (Instruction Per Cycle Scheduler) και μία παραλλαγή του DCCS ικανή να παίρνει αποφάσεις δρομολόγησης με περισσότερη ακρίβεια από τον DCCS.

Λέξεις κλειδιά

Δρομολόγηση νημάτων, SMT, CMP, CMT, General Execution-driven Multiprocessor Simulator (GEMS), Data Cache Conflict Scheduling (DCCS), Instruction Per Cycle Scheduling (IPCS).

Abstract

Modern processor architectures include from now and on a large number of threads in more than one cores. More specifically each core includes 2 or more threads (SMT core) increasing that way the thread level parallelism (TLP). Until recently the most prominent use of such processors, was as job servers running multiple independent threads on the different contexts of the various SMT cores. With the apperance of Intel i7 which is constituted from a CMP of 4 2-thread SMT cores we can see that processor industry begins to proporse this kind of processors for home use too.

Of course coexistence of more than one threads in a core involves also however common use of resources (processing units, cache memories) resulting the appearance of conflicts during the execution of applications and reduction of overal throughput.All these impose the scheduling of the threads and the right decisions for their pairing base on some criteria. The main goal of this diploma thesis is the study and evaluation of different policies - algorithms of thread scheduling in hardware level for multiprocessor architectures with asked the improvement of throughput and system performance more generally.

All of our research was realised using a modified GEMS (General Execution-driven Multiprocessor Simulator) simulator for the support of a CMT processor with thread migration capability. We studied three scheduling algorithms, DCCS (Data Cache Conflict Scheduling), IPCS (Instruction Per Cycle Scheduling) and a modified version of DCCS with the ability to take more accurate scheduling decisions than DCCS.

Key words

Thread scheduling, SMT, CMP, CMT, General Execution-driven Multiprocessor Simulator (GEMS), Data Cache Conflict Scheduling (DCCS), IPC Scheduling (IPCS).

Ευχαριστίες

Αυτή η διπλωματική εργασία πραγματοποιήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών(CSLAB) του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη του Αναπληρωτή Καθηγητή Νεκτάριου Κοζύρη. Θα ήθελα λοιπόν να ευχαριστήσω ιδιαίτερα τον καθηγήτη μου κ.Νεκτάριο Κοζύρη για την εποπτεία και την άριστη συνεργασία κατά την διάρκεια εκπόνησης της διπλωματικής μου εργασίας, καθώς και για την συμβολή του στην ολοκλήρωση μου ως μηχανικού μέσα από τις διδασκαλίες του. Ακολούθως θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Υπολογιστικών Συστημάτων. Πρωτίστως δε τον Μεταδιδακτορικό Ερευνητή Δρ. Κωνσταντίνο Νίκα για την συνεχή και αμέριστη καθοδήγηση καθώς επίσης και τον υποψήφιο Διδάκτωρα Νίκο Αναστόπουλο. Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη συμπαράσταση καθόλη τη διάρκεια των σπουδών μου στο ΕΜΠ.

Κωνσταντίνος Μπότσαρης,

Αθήνα, Ιουλίος 2009.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά ....., Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Υπολογιστικών Συστημάτων, Ιούλιος 2009.

Περιεχόμενα

Σχήματα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά

Στην εποχή μας παρατηρείται μια αλματώδης πρόοδος στον τομέα των υπολογιστικών συστημάτων, μια πρόοδος που παρουσιάζεται τα τελευταία είκοσι χρόνια, από τότε δηλαδή που εμφανίστηκε ο πρώτος ηλεκτρονικός υπολογιστής γενικής χρήσης. Ισχυρό χαρακτηριστικό αυτής της εξέλιξης είναι ο διαχρονικός νόμος του Moore, ο οποίος αναφέρει ότι από την ανακάλυψη του πρώτου ολοκληρωμένου κυκλώματος ο αριθμός των τρανζίστορς που μπορούν να τοποθετηθούν σε ένα ολοκληρωμένο κύκλωμα αυξάνεται εκθετικά σε σχέση με τον χρόνο και πιο συγκεκριμένα διπλασιάζεται περίπου κάθε δύο χρόνια. Σημαντικό παράγοντα στην εξέλιξη αυτή των υπολογιστών αποτελεί η ανακάλυψη της τεχνικής VLSI (Very-large-integration) με την οποία δημιουργούνται ολοκληρωμένα κυκλώματα με τον συνδυασμό εκατομμυρίων κυκλωμάτων τρανζίστορς σε ένα και μοναδικό τσιπ. Πρόσφατα μπορούμε να μιλάμε πλέον για δισεκατομμύρια τρανζίστορς με την εμφάνιση του Intel Montecito [21]. Φυσικό επακόλουθο αυτής της ανάπτυξης σε μια μάλλον αμφίδρομη σχέση είναι η εξέλιξη των επιστημών γύρω από αυτό το αντικείμενο, επιστήμες όπως η ψηφιακή σχεδίαση , η σχεδίαση - ανάλυση κυκλωμάτων VLSI, η σχεδίαση μικροεπεξεργαστών και φυσικά η αρχιτεκτονική υπολογιστών, που αποτελούν κομμάτια μιας μεγαλύτερης αλυσίδας. Ξεχωριστό κομμάτι αποτελεί η αρχιτεκτονική υπολογιστών που ο ρόλος της φαίνεται ξεκάθαρα μέσω του επόμενου ορισμού[1]:

> Ρόλος του αρχιτέκτονα υπολογιστών είναι να σχεδιάζει και να κατασκευάζει τα διάφορα επίπεδα ενός υπολογιστικού συστήματος με σκοπό την μεγιστοποίηση της απόδοσης και προγραμματιστικότητας στα όρια της τεχνολογίας και του κόστους.

Σύμφωνα με αυτά φαίνεται ο σημαντικότατος ρόλος του συγκεκριμένου κλάδου στη διαμόρφωση της επιστήμης των υπολογιστών, αφού ο αρχιτέκτονας όχι μόνο σχεδιάζει ένα σύστημα αλλά με την χρήση μετρικών απόδοσης και με τη βοήθεια μετροπρογραμμάτων ουσιαστικά αξιολογεί την επίδοση ενός επεξεργαστή και σχεδιάζει τη διαμόρφωσή του με περιορισμούς τεχνολογίας και κόστους. Αξίζει να σημειωθεί ότι τα μετροπρογράμματα επιλέγονται σε σχέση με τη χρήση για την οποία προορίζεται κάποιος επεξεργαστής (διακομιστής εργασίας, οικιακή χρήση, διακομιστής ιστού κτλ.). Στην επόμενη ενότητα θα δούμε κάποια στοιχεία για την ανάπτυξη των πολυπύρηνων αρχιτεκτονικών και τους λόγους για τους οποίους η βιομηχανία επεξεργαστών στράφηκε πλέον σε αυτή την τεχνολογία.

1.2 Πολυπύρηνες-Σύγχρονες αρχιτεκτονικές

Στις αρχιτεκτονικές ενός επεξεργαστή, οι αρχιτέκτονες υπολογιστών επικεντρώνονταν κυρίως σε τρεις σημαντικούς παράγοντες για βελτίωση της επίδοσης. Ο πρώτος ήταν η αύξηση την συχνότητας του επεξεργαστή, μια αύξηση που ήταν εφικτή με την χρήση όλο και μικρότερων κλιμάκων ολοκλήρωσης. Ο δεύτερος παράγοντας που εξερευνήθηκε ήταν η παραλληλοποίηση σε επίπεδο εντολών (Instruction Level Parallelism). Η χαρακτηριστική βελτίωση σε αυτό το κομμάτι ξεκίνησε με την διοχέτευση (pipelining) και στην συνέχεια επεκτάθηκε με την δυνατότητα ταυτόχρονου issueing εντολών

[1] Ο ορισμός πάρθηκε από τις διαλέξεις του μαθήματος CS258-Parallel Computer Architecture - University Of California,Berkeley

ανά κύκλο με την εμφάνιση των wide-issue superscalar επεξεργαστών. Ο τελευταίος παράγοντας ήταν η εξερεύνηση της ιεραρχίας μνήμης, κυρίως των κρυφών μνημών. Αποτέλεσμα αυτού ήταν η εμφάνιση αρκετών οργανώσεων κρυφής μνήμης, με κάθε μια να εμφανίζει τα δικά της πλεονεκτήματα και μειονεκτήματα.

Οι βελτιώσεις και τα ποσοστά παραλληλισμού που μπορούσαν να προσφέρουν πλέον αυτές οι αρχιτεκτονικές έχουν πλέον φτάσει στα όρια τους λόγω κυρίως των περιορισμών σε επίπεδο κυκλώματος με την δημιουργία πολύπλοκων pipelines. Η ύπαρξη ενός επεξεργαστή επιτρέπει μόνο την παραλληλοποίηση σε επίπεδο εντολών (ILP) που όμως δεν μπορεί να προσφέρει ικανοποιητικά ποσοστά παραλληλίας λόγω των εξαρτήσεων στον κώδικα. Σε συνδυασμό με τα ψηλά ποσοστά κατανάλωσης και κόστους που παρουσιάζουν η βιομηχανία αναγκάστηκε να στραφεί πλέον στην χρήση απλούστερων παράλληλων επεξεργαστών (πυρήνων) πάνω σε ένα τσιπ. Η τεχνολογία ολοκληρωμένων κυκλωμάτων έχει φτάσει σε τέτοιο σημείο που μπορεί να υποστηρίξει πλήρως τις πολυπύρηνες αρχιτεκτονικές και πλέον ενός μεγάλου αριθμού πυρήνων πάνω σε ένα τσιπ. Στο μέλλον μάλιστα γίνεται λόγος για δεκάδες πυρήνες στο ίδιο chip.

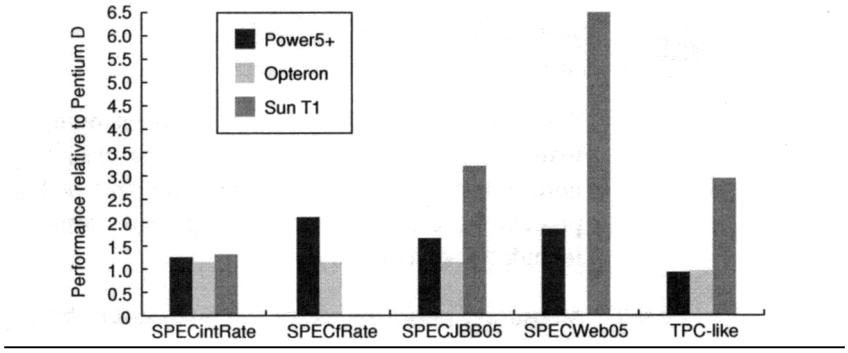
Παρ'όλα αυτά η βελτίωση στην επίδοση των πολυπύρηνων αρχιτεκτονικών είναι πλήρως εξαρτώμενη από τους αλγόριθμους του λογισμικού που χρησιμοποιούνται και από την υλοποίησή τους. Πιο συγκεκριμένα τα πιθανά κέρδη από αυτές τις αρχιτεκτονικές περιορίζονται από τα ποσοστά των χρησιμοποιημένων προγραμμάτων που παραλληλοποιούνται. Θεωρητικά κάποιος θα περίμενε επιταχύνσεις της τάξης του αριθμού των πυρήνων όμως λόγω της περιορισμένης παραλληλοποίησης σε κάποια προγράμματα αυτό στην πράξη δεν συμβαίνει.

Οι τελευταίες προτάσεις της βιομηχανίας αφορούν πολυπύρηνους επεξεργαστές στους οποίους πλέον όμως ο κάθε πυρήνας εμφανίζει περισσότερα του ενός νήματα. Έτσι πλέον μιλάμε για CMT (Chip Multithreaded) αρχιτεκτονικές, προσπαθώντας ακόμη περισσότερο να εκμεταλλευτούμε την παραλληλία. Συνάμα όμως εισάγοντας νέες τεχνολογίες, αυξάνοντας την παραλληλία σε επίπεδο πυρήνα, μαζί με την βελτίωση εισάγουμε και νέες προκλήσεις.

Characteristic	SUN T1	AMD Opteron	Intel Pentium D	IBM Power5
Cores	8	2	2	2
Instruction issues per clock per core	1	3	3	4
Multithreading	Fine-grained	No	SMT	SMT
Caches L1 I/D in KB per core L2 per core/shared L3 (off-chip)	16/8 3 MB shared	64/64 1 MB/core	12K uops/16 1 MB/core	64/32 L2: 1.9 MB shared L3: 36 MB
Peak memory bandwidth (DDR2 DRAMs)	34.4 GB/sec	8.6 GB/sec	4.3 GB/sec	17.2 GB/sec
Peak MIPS FLOPS	9600 1200	7200 4800 (w. SSE)	9600 6400 (w. SSE)	7600 7600
Clock rate (GHz)	1.2	2.4	3.2	1.9
Transistor count (M)	300	233	230	276
Die size (mm^2)	379	199	206	389
Power (W)	79	110	130	125

Σχήμα 1.1: Πίνακας με τέσσερις σύγχρονους πολυπύρηνους επεξεργαστές.

Στο Σχήμα 1.1 παραθέτουμε ένα πίνακα με τις περιγραφές τεσσάρων σύγχρονων πολυπύρηνων αρχιτεκτονικών. Οι τρεις από τους τέσσερις επεξεργαστές, πιο συγκεκριμένα οι SUN T1, Intel Pentium D και ο IBM Power 5 παρουσιάζουν πολυνηματικούς πυρήνες σε αντίθεση με τον AMD Opteron στον οποίο κάθε πυρήνας μπορεί να τρέχει ένα και μοναδικό νήμα. Ο Sun T1 περιλαμβάνει τον μεγαλύτερο αριθμό πυρήνων on-chip αποτελούμενος από οκτώ πυρήνες ενώ οι υπόλοιποι δύο.



Σχήμα 1.2: Η επίδοση των αρχιτεκτονικών Sun T1, AMD Opteron και IBM Power5+ σε σχέση με την Intel Pentium D στα διάφορα μετροπρογράμματα.

Το Σχήμα 1.2 εμφανίζει την επίδοση των Sun T1, AMD Opteron και IBM Power5+ σε σχέση με την αντίστοιχη στον Intel Pentium D στα μετροπρογράμματα SPECRate CPU, SPECJBB2005 Java Business , SPECWeb05 Web server και TPC-C. Το προαναφερθέν σχήμα μας επιτρέπει να κάνουμε μια απλή αλλά πολύ σημαντική παρατήρηση. Στις πολυνηματικές εφαρμογές με υψηλό ποσοστό παραλληλοποίησης (μετροπρογράμματα με χρήση πολυνηματικού προγραμματισμού ή την παράλληλη εκτέλεση νημάτων στους πυρήνες) όπως είναι τα SPECJBB05, SPECWeb05 και TPC-C ο Sun T1 ο οποίος διαθέτει την μεγαλύτερη δυνατότητα για παραλληλία σε επίπεδο νημάτων λόγω του μεγάλου αριθμού πυρήνων παρουσιάζει τεράστιες βελτιώσεις σε σχέση με τις υπόλοιπες αρχιτεκτονικές. Αντίθετα στα μετροπρογράμματα SPECintRate και SPECfRate όπου το ποσοστό παραλληλοποίησης είναι χαμηλό λόγω της εκτέλεση ενός και μοναδικού νήματος ο Sun T1 στην μία περίπτωση εμφάνισε σχεδόν μηδενική βελτίωση σε σχέση με τους υπόλοιπους επεξεργαστές ενώ στην άλλη ίση επίδοση με τον Intel Pentium D και μικρότερη σε σχέση με τους AMD Opteron και IBM Power5.

Εδώ ουσιαστικά αποδεικνύεται η πρόταση που αναφέραμε προηγουμένως: Για να μπορούμε να εκμεταλλευτούμε πλήρως τις δυνατότητες της παραλληλίας σε επίπεδο νημάτων που μας δίνει μια πολυπύρηνη αρχιτεκτονική πρέπει να στραφούμε στον πολυνηματικό προγραμματισμό ή την παράλληλη εκτέλεση διεργασιών-νημάτων στους πυρήνες. Αυτό είναι σημαντικό για να μπορέσουμε να πετύχουμε βελτιώσεις καλύτερες από αυτές των αρχιτεκτονικών ενός επεξεργαστή. Η εγκατάλειψη της σειριακής λογικής επιβάλλεται και η στροφή στην παραλληλία είναι το κλειδί για την εκμετάλλευση των δυνατοτήτων των σύγχρονων πολυπύρηνων αρχιτεκτονικών. Στο προηγούμενο παράδειγμα φάνηκε το πόσο εύκολα εκμηδενίστηκαν τα οφέλη που εισάγονται με χρήση κώδικα χαμηλού ποσοστού παραλληλοποίησης αφού σε κάποιες περιπτώσεις ένα οκταπύρηνο σύστημα δεν κατάφερε να εισάγει την παραμικρή βελτίωση σε αντίθεση με την χρήση εφαρμογών που εκμεταλλεύονταν την παραλληλία σε επίπεδο κώδικα όπου η βελτίωση ήταν χαρακτηριστική.

1.3 Σκοπός

Μια από τις σημαντικότερες προκλήσεις στους CMT επεξεργαστές αφορά εκτός από την δρομολόγηση των νημάτων σε επίπεδο πυρήνα(δηλαδή εφόσον ανατεθούν στον πυρήνα πως θα εκτελούνται χρο-

νικά), την επιλογή των νημάτων για συνδρομολόγηση στον ίδιο πυρήνα σε κάποια περίοδο εκτέλεσης σύμφωνα με κάποιο αλγόριθμο. Σκοπός της συνδρομολόγησης των νημάτων είναι η εύρεση των καταλληλότερων συνδυασμών με σκοπό την βελτίωση επίδοσης σύμφωνα με κάποιες μετρικές.

Μέχρι τώρα στους περισσότερους πολυνηματικούς πολυεπεξεργαστές ενός τσιπ η συνδρομολόγηση των νημάτων γίνεται από το λειτουργικό η οποία συνήθως γίνεται ανά περιόδους παρατηρώντας κυρίως το load-balancing μεταξύ των πυρήνων του συστήματος όπως π.χ στην περίπτωση του λειτουργικού Linux [1]. Παρόλο που αυτό είναι μια λογική προσέγγιση πολλές φορές τα αποτελέσματα που επιτυγχάνονται δεν είναι ικανοποιητικά κυρίως όσον αφορά την μεμονωμένη επίδοση των νημάτων δημιουργώντας την ανάγκη για περαιτέρω έρευνα όσον αφορά το αντικείμενο αυτό.

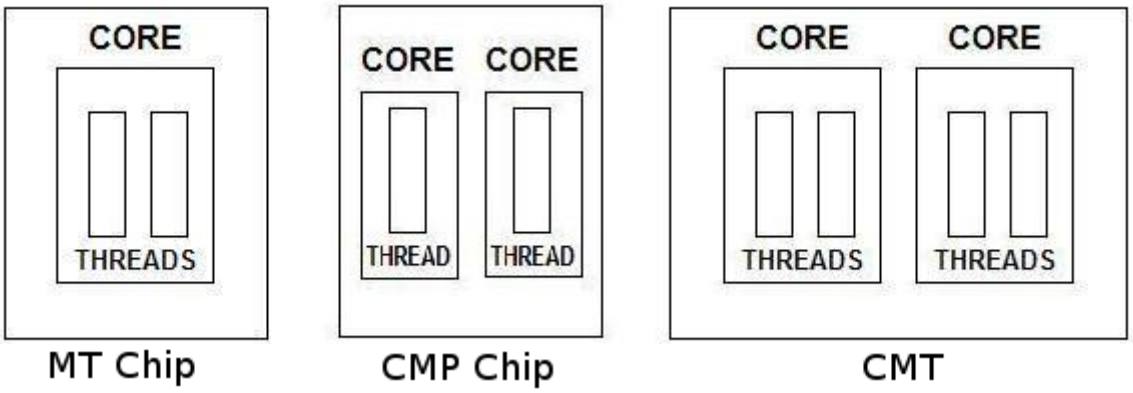
Ένα μεγάλο κομμάτι της έρευνας στους CMTs είναι ακριβώς αυτό με προτάσεις τόσο σε επίπεδο υλικού όσο και σε επίπεδο λειτουργικού. Εμείς σε αυτή την διπλωματική εργασία θα μελετήσουμε αλγόριθμους για την συνδρομολόγηση των νημάτων στους πυρήνες σε επίπεδο υλικού και θα δούμε την βελτίωση που εισάγει ο καθένας τόσο στην απόδοση συστήματος όσο και κάθε νήματος.

Κεφάλαιο 2

Πολυνηματικές Αρχιτεκτονικές

2.1 Γενικά

Σε αυτή την ενότητα θα μελετήσουμε τις σύγχρονες πολυνηματικές αρχιτεκτονικές. Πιο συγκεκριμένα θα μελετήσουμε τον πολυνηματισμό στην περίπτωση του ενός επεξεργαστή, τον πολυεπεξεργαστή ενός τσιπ και τέλος τον πολυεπεξεργαστή ενός τσιπ με πολυνηματικούς πυρήνες(CMT). Αξίζει να σημειωθεί ότι η αρχιτεκτονική CMT είναι ένας συνδυασμός των δύο τεχνολογιών πολυνηματισμού και CMP.



Σχήμα 2.1: Πολυνηματικές αρχιτεκτονικές.

Στο Σχήμα 2.1 απεικονίζονται οι προαναφερθείσες αρχιτεκτονικές οι οποίες θα αναλυθούν στην συνέχεια. Στο τέλος της ενότητας θα γίνει αναφορά στην υπάρχουσα έρευνα για συνδρομολόγηση στους CMT που είναι και το κύριο θέμα στην διπλωματική μας εργασία.

2.2 Πολυνηματισμός σε ένα επεξεργαστή

2.2.1 Γενικά

Ένα νήμα είναι μία ξεχωριστή διεργασία με τις δικές της εντολές και δεδομένα. Ένα νήμα επίσης μπορεί να αντιπροσωπεύει μια διεργασία,που είναι ένα κομμάτι ενός παράλληλου προγράμματος, αποτελούμενο από πολλές τέτοιες διεργασίες ή μπορεί να είναι ένα ανεξάρτητο πρόγραμμα.

Στις πολυνηματικές αρχιτεκτονικές ενός επεξεργαστή έχουμε την παρουσία πολλών νημάτων να τρέχουν παράλληλα σε ένα επεξεργαστή και έτσι μιλούμε πια για παραλληλία σε επίπεδο νημάτων σε αντίθεση με παραλληλία επιπέδου εντολών όπου εκεί έχουμε εκμετάλλευση της παραλληλοποίησης σε επίπεδο κώδικα από ένα μόνο νήμα. Τα νήματα σε αυτή την αρχιτεκτονική μοιράζονται τα δομικά στοιχεία ενός πυρήνα δηλαδή τις υπολογιστικές μονάδες,τις κρυφές μνήμες και το translation lookaside buffer

(TLB). Σε αντίθεση με τα πολυεπεξεργαστικά συστήματα που περιλαμβάνουν πολλαπλές μονάδες υπολογισμού, ο πολυνηματισμός εδώ στοχεύει στην βελτίωση της χρήσης ενός πυρήνα με τη βοήθεια του παραλληλισμού σε επίπεδο νημάτων.

2.2.2 Πολυνηματισμός σε επίπεδο υλικού

Οι πολυνηματικές αρχιτεκτονικές σχεδιάστηκαν για την βελτίωση της επίδοσης ενός επεξεργαστή παρ' όλα αυτά όμως όπως συμβαίνει και στις περισσότερες περιπτώσεις αρχιτεκτονικών μαζί με τα πλεονεκτήματα που εμφανίζονται,συνυπάρχουν και κάποια μειονεκτήματα.

Μια σημαντική βελτίωση που επιτεύχθηκε ήταν η εξής: Αν ένα νήμα αναγκαστεί να καθυστερήσει για οποιοδήποτε λόγο (όπως π.χ όταν συναντήσει αρκετές αστοχίες κρυφής μνήμης που εισάγουν καθυστέρηση), τα άλλα νήματα μπορούν να συνεχίσουν εκμεταλλευόμενα τους ελεύθερους πόρους του επεξεργαστή, με αποτέλεσμα την γρηγορότερη εκτέλεση και την αύξηση απόδοσης. Παρόμοια συμπεριφορά μπορεί να εμφανισθεί στην περίπτωση που ο κώδικας εμφανίζει μεγάλο ποσοστό εξαρτήσεων μεταξύ των εντολών αφήνοντας κάποιους πόρους ανεκμετάλλευτους στην διοχέτευση (pipeline). Έτσι το να τρέξουμε ένα άλλο νήμα που μπορεί να εκμεταλλευτεί αυτούς τους πόρους, βοηθά στην αύξηση της απόδοσης. Μια επιπλέον βελτίωση είναι το γεγονός ότι τα νήματα μπορούν πλέον να μοιράζονται κάποια δεδομένα με αποτέλεσμα την καλύτερη χρήση της κρυφής μνήμης (εξασφαλίζοντας παράλληλα και την συνάφεια της) και συγχρονισμό δεδομένων.

Από την άλλη όμως κοινή χρήση των επεξεργαστικών πόρων μεταξύ των νημάτων και παράλληλη παρουσία τους στην διοχέτευσης μπορεί να οδηγήσει στον συνωστισμό της. Συγκρούσεις και ανταγωνισμός μεταξύ των νημάτων κάνουν πλέον την παρουσία τους αισθητή αφού υπάρχει περίπτωση νήματα να περιμένουν την απελευθέρωση πόρων από κάποια άλλα. Τέλος παράλληλα με την σχεδίαση σε επίπεδο υλικού μιας τέτοιας αρχιτεκτονικής, εμφανίζεται η ανάγκη για διαμόρφωση του υπάρχοντος λογισμικού ώστε να την υποστηρίζει και να την εκμεταλλεύεται σωστά. Επιπλέον η δημιουργία ενός προγραμματιστικού μοντέλου της υποστήριξης πολυνηματισμού είναι αναγκαία για μελλοντική σχεδίαση προγραμμάτων.

Οι κατηγορίες του πολυνηματισμού σε ένα επεξεργαστή

Το hardware multithreading μπορεί να χωριστεί σε τρεις κύριες κατηγορίες: το coarse grained , το fined grained και το simultaneous multithreading. Η διαφορά ανάμεσα στις τρεις αυτές αρχιτεκτονικές έγκειται στον τρόπο που αυτές εναλλάσσουν τα διάφορα νήματα. Ένας coarse-grained επεξεργαστής ενεργοποιεί ένα άλλο νήμα όταν το νήμα που τρέχει καθυστερήσει λόγω μιας αίτησης μνήμης ή εξαιτίας κάποιου άλλου γεγονότος που εισάγει καθυστέρηση. Το μειονέκτημα αυτής της τεχνικής είναι το ψηλό κόστος σε απόδοση που έχει αυτή η εναλλαγή κι αυτό γιατί ο επεξεργαστής αποφασίζει για την ενεργοποίηση του καινούριου νήματος όταν το άλλο βρίσκεται ήδη σε προχωρημένο στάδιο στο pipeline, αφού το νήμα έχει ήδη κάποιες εντολές in flight στον επεξεργαστή. Έτσι αυτές οι εντολές πρέπει κάπως να ακυρωθούν από τον επεξεργαστή, με το κόστος σε κύκλους όμως για την έκδοσή τους να παραμένει.

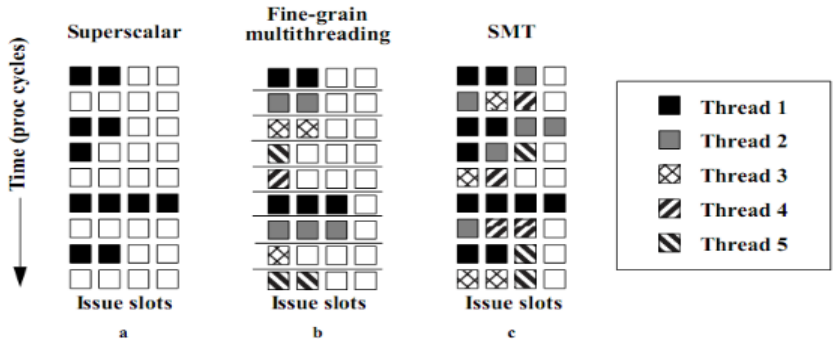
Αντίθετα στην fine-grained πολυνηματική αρχιτεκτονική τα νήματα εναλλάσσονται σε κάθε κύκλο υλοποιώντας έτσι μια Round Robin πολιτική. Με αυτό τον τρόπο οι πιο πάνω σπατάλες σε κύκλους που εμφανίζονταν στον coarse-grained πολυνηματισμό παύουν πλέον να υπάρχουν αφού τα νήματα που παρουσιάζουν καθυστέρηση σε κάποια στιγμή μπορούν να προσπεραστούν. Με αυτό τον τρόπο ο fine-grained πολυνηματισμός επιτρέπει την απόκρυψη απωλειών σε ρυθμαπόδοση που εμφανίζονται λόγω μικρών ή μεγάλων καθυστερήσεων. Το βασικό μειονέκτημα αυτής της πολιτικής είναι ότι μπορεί να προκύψει παρεμπόδιση των νημάτων που μπορούν να συνεχίσουν χωρίς καθυστερήσεις με την εκτέλεση εντολών από άλλα νήματα.

Μια πιο ευρέως διαδεδομένη προσέγγιση στο hardware multithreading είναι το simultaneous multithreading που αναλύεται στην επόμενη υποενότητα.

Αρχιτεκτονική Simultaneous Multithreading (SMT)

Το simultaneous multithreading συνδυάζει χαρακτηριστικά δύο άλλων τύπων επεξεργαστών: των wide-issue superscalar και των multithreaded επεξεργαστών. Από τους μεν πρώτους παίρνει την ικανότητα να εκδίδει πολλές εντολές ανά κύκλο και από τους δεύτερους την ικανότητα να εκτελεί διάφορα προγράμματα (ή νήματα) ταυτόχρονα. Το αποτέλεσμα είναι ένα επεξεργαστής που μπορεί να εκδώσει πολλές εντολές από διάφορα νήματα σε κάθε κύκλο.

Η διαφορά μεταξύ υπερβαθμωτού, πολυνηματικού και simultaneous multithreading φαίνεται στο



Σχήμα 2.2: Κατανομή θυρίδων εντολών σε διαφορετικές αρχιτεκτονικές

Σχήμα 2.2, όπου παρουσιάζεται ένα παράδειγμα ακολουθίας εκτέλεσης για τις τρεις αρχιτεκτονικές. Στα σχήματα αυτά, κάθε γραμμή αναπαριστά ένα κύκλο εκτέλεσης και τα τέσσερα κουτάκια δείχνουν τις τέσσερις πιθανές εντολές που μπορεί ο επεξεργαστής να εκδώσει ανά κύκλο. Ένα γεμάτο κουτί δείχνει ότι ο επεξεργαστής μπόρεσε να βρει μια εντολή για να εκδώσει σε αυτή τη θυρίδα, ενώ ένα άδειο κουτί δείχνει αχρησιμοποίητη ή σπατάλη (wasted) θυρίδας. Διαχωρίζουμε τις θυρίδες εντολών που σπαταλώνται σε δύο τύπους. Οριζόντια σπατάλη (horizontal waste) παρατηρείται όταν κάποια, αλλά όχι όλες, οι θυρίδες μπορούν να χρησιμοποιηθούν σε κάθε κύκλο. Αυτό είναι τυπικό δείγμα έλλειψης ILP σε ένα πρόγραμμα. Κατακόρυφη σπατάλη (vertical waste) παρατηρείται όταν ένας κύκλος μένει πλήρως αχρησιμοποίητος. Αυτό μπορεί να προκληθεί από μία εντολή με μεγάλη καθυστέρηση (όπως μία ανάγνωση μνήμης) που κατακρατά την έκδοση περαιτέρω εντολών.

Ένας τυπικός superscalar, όπως ο DEC Alpha 21164, ο HP PA-8000, ο Intel Pentium Pro,ο MIPS R10000 και ο PowerPC 604 φαίνεται στο Σχήμα 2.2a. Σύμφωνα με τη λειτουργία των υπερβαθμωτών, εκτελεί ένα και μόνο πρόγραμμα ή νήμα, από το οποίο προσπαθεί να βρει πολλαπλές εντολές για να εκδώσει σε κάθε κύκλο. Σε περίπτωση που αποτύχει, οι θυρίδες έκδοσης εντολών μένουν αχρησιμοποίητες, κάτι που προκαλεί τόσο οριζόντια όσο και κάθετη σπατάλη. Οι αρχιτεκτονικές multithreaded (π.χ. ο Tera), από την άλλη, περιέχουν κατάσταση υλικού – μετρητή προγράμματος και καταχωρητές – για αρκετά νήματα. Σε κάθε δεδομένο κύκλο εκτέλεσης ο επεξεργαστής εκτελεί εντολές από ένα εκ των νημάτων. Στον επόμενο κύκλο, εκτελεί context switching σε context άλλου νήματος, έτσι ώστε να μπορεί να εκτελέσει εντολές από το νέο νήμα. Το context switching μπορεί να γίνει σε ένα μόνο κύκλο, επειδή οι πολλαπλές καταστάσεις υλικού(hardware contexts) που περιέχουν αντικαθιστούν την ανάγκη για αποθήκευση και επαναφορά της κατάστασης του επεξεργαστή. Το context switching νημάτων και η επακόλουθη ακολουθία εκτέλεσης φαίνονται στο Σχήμα 2.2b . Τα διάφορα σχεδιαστικά μορφήματα αναπαριστούν διαφορετικά νήματα, τα οποία εκδίδουν εντολές σε διαφορετικούς κύκλους. Στο σχήμα διαφαίνεται το κύριο πλεονέκτημα του πολυνηματισμού, και συγκεκριμένα η καλύτερη ανοχή σε εντολές με μεγάλες καθυστερήσεις, γιατί μπορεί να δρομολογήσει κάποιο άλλο νήμα

κατά τη διάρκεια που ένα νήμα έχει μπλοκάρει. Αξίζει να σημειωθεί, όμως, ότι παρόλο που στην αρχιτεκτονική αυτή δεν παρουσιάζεται κατακόρυφη σπατάλη, έχει αυξηθεί η οριζόντια σπατάλη, αφού υπάρχει μετατροπή κάποιας από την πρότινος κατακόρυφη σπατάλη σε οριζόντια. Συνεπώς, καθώς το παράθυρο εκτέλεσης πλαταίνει οι πολυνηματικές αρχιτεκτονικές θα έχουν τελικά την ίδια μοίρα με τους υπερβαθμωτούς, δηλαδή δε θα έχουν τη δυνατότητα να βρουν αρκετό ILP σε ένα νήμα που εκτελείται μόνο του για να αξιοποιήσουν επαρκώς τον επεξεργαστή.

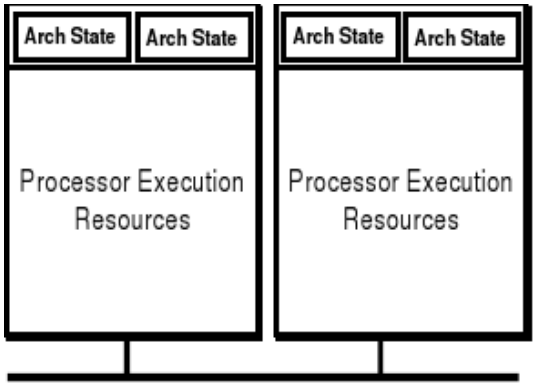
Η αρχιτεκτονική simultaneous multithreading, όπως φαίνεται στο Σχήμα 2.2c , συνδυάζει τα καλύτερα στοιχεία των αρχιτεκτονικών multithreaded και superscalar. Όπως τους superscalar, ένας SMT μπορεί εκμεταλλευτεί την επαλληλία σε επίπεδο εντολών(ILP) ενός νήματος εκδίδοντας πολλαπλές εντολές ανά κύκλο όμοια με τους multithreaded, μπορεί να καλύψει διεργασίες με μεγάλη καθυστέρηση εκτελώντας εντολές από άλλα νήματα. Η διαφορά του SMT είναι ότι μπορεί να κάνει και τα δύο ταυτόχρονα, δηλαδή στον ίδιο κύκλο. Σε κάθε κύκλο ένας SMT επεξεργαστής επιλέγει εντολές για εκτέλεση από όλα τα ενεργά νήματα. Ο επεξεργαστής χρονοδρομολογεί δυναμικά τα νήματα σε όλους τους υλικούς πόρους του, παρέχοντας έτσι την καλύτερη πιθανότητα για την υψηλότερη αξιοποίηση του υλικού. Αν ένα νήμα έχει υψηλό επίπεδο επαλληλίας εντολών, τότε αυτός ο τύπος επαλληλίας ικανοποιείται εάν τρέχουν πολλά νήματα, με το καθένα να έχει χαμηλό επίπεδο επαλληλίας εντολών, μπορούν να εκτελούνται μαζί, εξισορροπώντας έτσι το χαμηλό επίπεδο ILP του κάθε νήματος. Συνεπώς, η αρχιτεκτονική simultaneous multithreading έχει τη δυνατότητα να επαναφέρει θυρίδες εκτέλεσης που χάθηκαν λόγω οριζόντιας και κατακόρυφης σπατάλης.

Το αποτέλεσμα που προκύπτει από την SMT αρχιτεκτονική είναι η καλύτερη απόδοση για μια ποικιλία από εφαρμογές. Εάν πρόκειται για ένα σύνολο από ανεξάρτητα προγράμματα, υπάρχει βελτίωση του συνολικού throughput του μηχανήματος. Όταν ένα πρόγραμμα δεν έχει εντολές έτοιμες για εκτέλεση, εντολές μπορούν να βρεθούν σε ένα από τα άλλα προγράμματα. Αντίστοιχα, προγράμματα που είναι παραλληλοποιήσημα, είτε από τον compiler είτε από τον προγραμματιστή, έχουν τα ίδια κέρδη σε throughput αλλά εδώ το αποτέλεσμα είναι μείωση του χρόνου εκτέλεσης για την εφαρμογή. Τέλος προγράμματα που πρέπει να εκτελεστούν σαν ένα και μόνο νήμα, έχουν όλους τους πόρους μηχανής στη διάθεσή τους και πετυχαίνουν περίπου τα ίδια επίπεδα απόδοσης όπως όταν εκτελούνται σε ένα μονονηματικό επεξεργαστή.

Τεχνολογία Intel Hyper-Threading

Η τεχνολογία Hyper-Threading [19] είναι η SMT υλοποίηση της Intel που χρησιμοποιείται στους επεξεργαστές της Pentium 4, Atom και τελευταία στους Core i7. Η τεχνολογία Hyper-Threading κάνει ένα επεξεργαστή να φαίνεται σαν πολλαπλοί λογικοί επεξεργαστές. Για να επιτευχθεί αυτό διατηρείται ένα αντίγραφο της αρχιτεκτονικής κατάστασης σε κάθε λογικό επεξεργαστή και οι λογικοί επεξεργαστές μοιράζονται ένα κοινό σύνολο από φυσικές πηγές εκτέλεσης. Από προγραμματιστικής πλευράς καθώς και από την πλευρά του λειτουργικού οι διεργασίες δρομολογούνται στους λογικούς επεξεργαστές δηλαδή το λειτουργικό αντιλαμβάνεται μόνο τους λογικούς επεξεργαστές και τους θεωρεί σαν φυσικούς. Από αρχιτεκτονικής πλευράς όμως οι εντολές από κάθε λογικό επεξεργαστή εκτελούνται ταυτόχρονα σε κοινές μονάδες εκτέλεσης.

Όπως βλέπουμε στο Σχήμα 2.3 έχουμε δύο φυσικούς επεξεργαστές που να υποστηρίζουν τεχνολογία Hyper-Threading με δύο αντίγραφα της αρχιτεκτονικής κατάστασης σε κάθε φυσικό επεξεργαστή άρα συνολικά τέσσερις λογικούς επεξεργαστές. Η πρώτη υλοποίηση της τεχνολογίας Hyper-threading έγινε στους Intel Xeon με διπλούς επεξεργαστές ή ακόμα και πολυεπεξεργαστές όπου ο καθένας υποστήριζε δύο λογικούς επεξεργαστές. Με την οργανωμένη χρήση των επεξεργαστικών πόρων αυτή η οικογένεια επεξεργαστών αύξησε σημαντικά την απόδοση κρατώντας σταθερό το κόστος συστήματος. Η τεχνολογία Hyper-Threading αύξησε μεν το μέγεθος του chip (λιγότερο από 5%) για την υλοποίησή της και την ανάγκη για περισσότερη ισχύ, όμως τα πλεονεκτήματα που προσφέρει ξεπερνούν αυτές τις μικρές αδυναμίες.



Σχήμα 2.3: Επεξεργαστές με τεχνολογία Hyper-Threading

2.2.3 Υλοποίηση στους επεξεργαστές Intel Xeon

Η υλοποίηση Hyper-Threading στην οικογένεια επεξεργαστών Intel Xeon ήταν η πρώτη που τέθηκε σε κυκλοφορία στο εμπόριο. Τρεις ήταν οι βασικοί στόχοι της σχεδίασης αυτού του επεξεργαστή:

1. Η ελαχιστοποίηση του κόστους σε μέγεθος ψηφίδας που θα προέκυπτε από την υλοποίηση της τεχνολογίας Hyper-Threading, η οποία περιοχή τελικά καταλάμβανε λιγότερο του 5% της ψηφίδας του επεξεργαστή.

2. Να εξασφαλιστεί ότι σε περίπτωση που ένας λογικός επεξεργαστής αναγκαστεί να καθυστερήσει για οποιοδήποτε λόγο(αστοχία κρυφής μνήμης, λανθασμένη πρόβλεψη διακλάδωσης, χαμηλό ILP), ο άλλος εκμεταλλευόμενος θα μπορεί να συνεχίσει και να προχωρήσει την εργασία του. Αυτό επιτεύχθηκε με την χρήση ουρών buffering όπου δεν μπορεί ένας επεξεργαστής να καταλάβει όλη την ουρά σε περίπτωση που και τα δύο νήματα είναι ενεργά[1]. Πρακτικά αυτό υλοποιήθηκε με διαχωρισμό των εγγραφών στο buffer ανάμεσα στους δύο λογικούς επεξεργαστές.

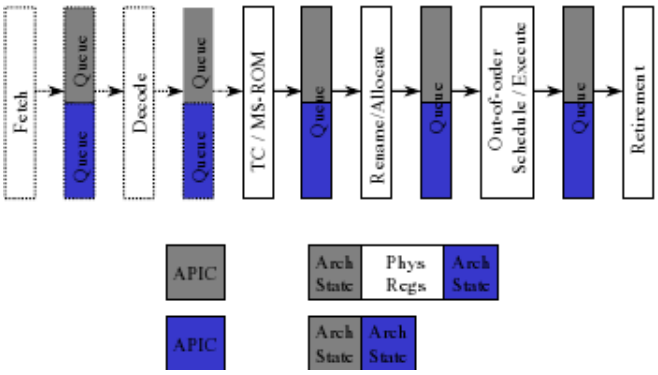
3. Τέλος, το να επιτρέπεται σε ένα επεξεργαστή που υποστηρίζει την τεχνολογία Hyper-Threading και στον οποίο τρέχει μόνο ένα νήμα να τρέχει στην ίδια ταχύτητα με ένα επεξεργαστή χωρίς αυτή, δηλαδή να μην υπάρχει απώλεια σε επίδοση στην περίπτωση ενός τρέχοντος νήματος. Αυτό σημαίνει κατάργηση του διαμερισμού των πόρων και στους δύο λογικούς επεξεργαστές στην περίπτωση που μόνο ένα νήμα τρέχει.

Η μορφή του pipeline στους επεξεργαστές Intel Xeon έχει την μορφή του Σχήματος 2.4.
Στο εν λόγω σχήμα παρατηρούμε την παρουσία ουρών μεταξύ των κύριων κομματιών του pipeline. Οι ουρές buffering είτε διαμερίζονται είτε αντιγράφονται για την εξασφάλιση ανεξάρτητης προόδου εργασίας σε κάθε λογικό μπλοκ.

2.3 Chip Multiprocessor - Πολυπύρηνες Αρχιτεκτονικές

Ένας πολυπύρηνος επεξεργαστής είναι ένα σύστημα επεξεργασίας που αποτελείται από δύο ή περισσότερους ανεξάρτητους πυρήνες. Οι πυρήνες είναι ενσωματωμένοι επάνω σε ένα ενιαίο chip ολοκληρωμένων κυκλωμάτων. Ένας manycore επεξεργαστής είναι ο επεξεργαστής στον οποίο ο αριθμός πυρήνων είναι αρκετά μεγάλος (με κατώτατο όριο μερικές δεκάδες πυρήνων) και στον οποίο οι κλασικές τεχνικές πολυεπεξεργαστών δεν είναι πλέον αποδοτικές.

[1] Ουσιαστικά σε κάποιο νήμα τρέχει ένα ανενεργό loop από μεριάς του λειτουργικού το οποίο ελέγχει της ουρές εργασίας, ένα loop που δυστυχώς κοστίζει σε μονάδες εκτέλεσης.



Σχήμα 2.4: Το pipeline του επεξεργαστή Intel Xeon

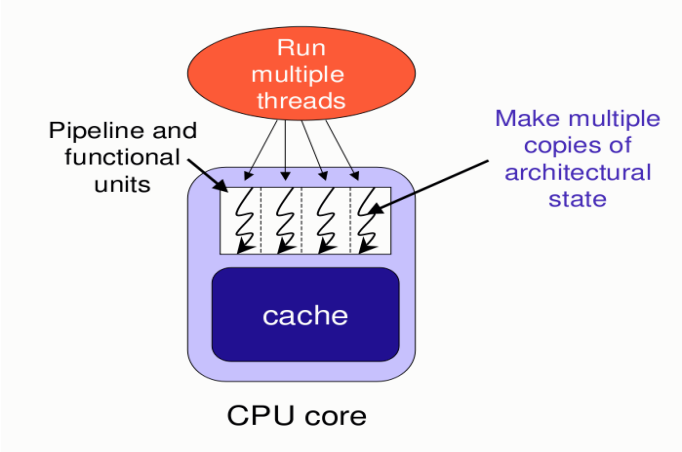
Ο πολυπύρηνος επεξεργαστής εφαρμόζει την πολυεπεξεργασία σε μια ενιαία φυσική συσκευασία. Οι πυρήνες μπορεί να μοιράζονται ή να μην μοιράζονται τις κρυφές μνήμες και επικοινωνούν μέσω διαμοιραζόμενης μνήμης ή στέλνοντας και λαμβάνοντας μηνύματα Κάποιες συχνά χρησιμοποιημένες τοπολογίες δικτύων διασύνδεσης πυρήνων είναι οι εξής: bus, ring, 2-dimensional mesh και crossbar. Όλοι οι πυρήνες είναι ίδιοι στα ομοιογενή (homogeneous) πολυπύρηνα συστήματα ενώ διαφέρουν στα ετερογενή (heterogeneous) πολυπύρηνα συστήματα. Ακριβώς όπως με τα συστήματα ενιαίων επεξεργαστών, οι πυρήνες στα πολυπύρηνα συστήματα μπορούν να εφαρμόσουν τις αρχιτεκτονικές όπως superscalar, VLIW, διανυσματικής επεξεργασίας ή SIMD. Οι πολυπύρηνοι επεξεργαστές χρησιμοποιούνται ευρέως σε πολλές περιοχές συμπεριλαμβανομένων: γενικής χρήσης, ενσωματωμένα συστήματα, δίκτυα, επεξεργασία ψηφιακού σήματος και γραφικών.

2.4 CMT - Chip Multithreaded επεξεργαστής

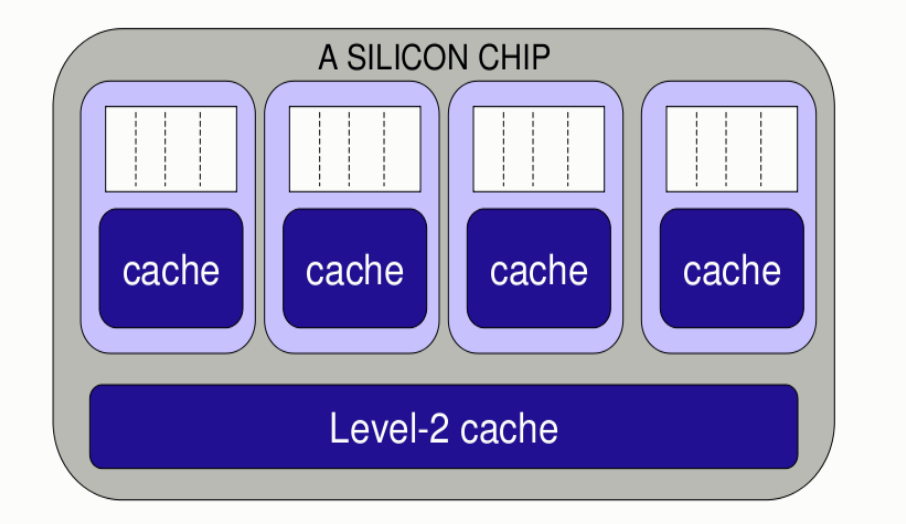
2.4.1 Γενικά

Οι CMT επεξεργαστές όπως βλέπουμε στα Σχήματα 2.1 και 2.6 τρέχουν πολλαπλά νήματα σε επεξεργαστές που βρίσκονται στο ίδιο chip. Όπως αναφέρθηκε και στην περίπτωση του SMT το CMT προσπαθεί να εκμεταλλευτεί την παραλληλία σε επίπεδο νημάτων σε μεγαλύτερο βαθμό βέβαια λόγω της ύπαρξης παράλληλων πυρήνων.
Η δυνατότητα εξαγωγής παραλληλισμού από μια ακολουθία εντολών (instruction stream) γενικότερα είναι δύσκολη γιατί πρέπει να προβλέψουμε τα μελλοντικά εκτελεστικά μοτίβα της ακολουθίας εντολών. Με το CMT όμως μπορούμε να χρησιμοποιήσουμε τον παραλληλισμό σε πολλαπλά streams εντολών. Το CMT ουσιαστικά είναι ένας συνδυασμός του Hardware Multithreading και του Chip Multiprocessing. Οι επεξεργαστικοί πόροι μοιράζονται ανάμεσα στους πυρήνες όπως φαίνεται στο Σχήμα 2.5.

Οι κύρια χρήση για την οποία προορίζονται οι CMT είναι σε server μηχανήματα ενώ πλέον με την εμφάνιση του Intel i7 μπορούμε να μιλούμε και για χρήση τους στους επιτραπέζιους υπολογιστές στο σπίτι. Ένα χαρακτηριστικό παράδειγμα CMT αρχιτεκτονικής είναι ο Sun Niagara ο οποίος θα αναλυθεί στην επόμενη υποενότητα.



Σχήμα 2.5: Ένας από τους πυρήνες της CMT αρχιτεκτονικής.



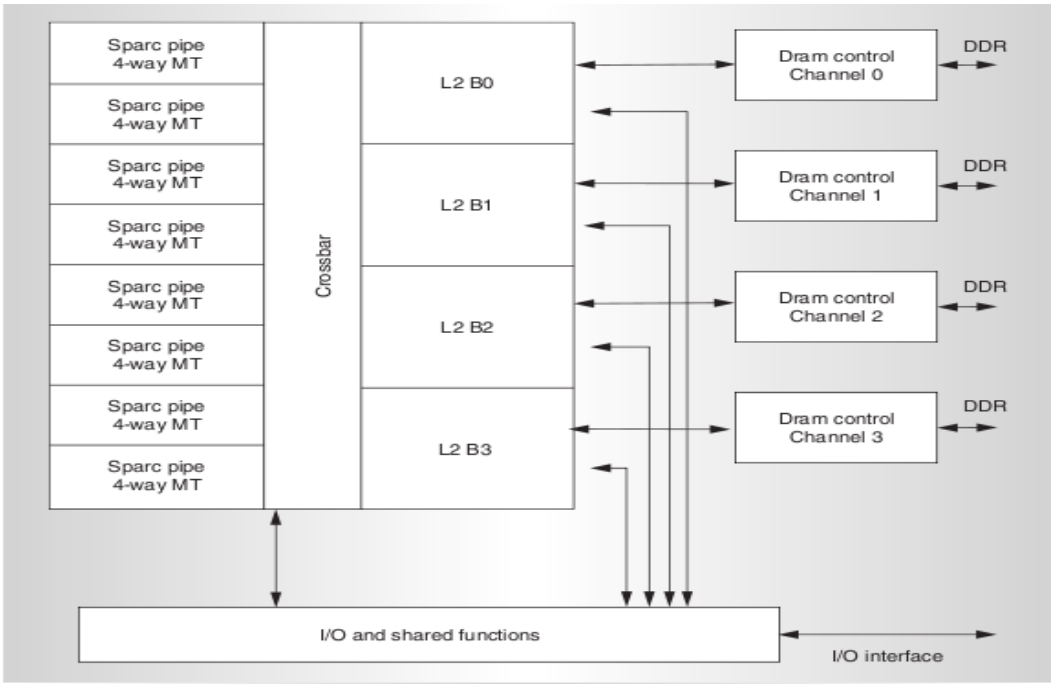
Σχήμα 2.6: Ο CMT επεξεργαστής.

2.4.2 Επεξεργαστής Sun UltraSPARC T1(Niagara)

Γενικά

Ο επεξεργαστής Sun Niagara [15] που φαίνεται στο Σχήμα 2.7 στηρίζεται σε μια νέα υλοποίηση της αρχιτεκτονικής Sparc V9 η οποία εκμεταλλεύεται μεγάλα ποσοστά παραλληλισμού μέσα στο chip για να προσφέρει υψηλή ρυθμαπόδοση. Ο Niagara περιέχει οκτώ επεξεργαστές και ο καθένας τέσσερα νήματα. Συνολικά έχουμε τριάντα δύο λογικούς επεξεργαστές δηλαδή συνδυάζει τις ιδέες του CMP και του πολυνηματισμού (fine-grained). Η παράλληλη εκτέλεση πολλών νημάτων, ως γνωστόν, κρύβει τις μεγάλες καθυστερήσεις. Παρ'όλα αυτά τα τριάντα δύο νήματα είναι μια μεγάλη απαίτηση για το υποσύστημα μνήμης έτσι ώστε αυτό να μπορέσει να υποστηρίξει υψηλό bandwidth. Για την παροχή αυτού του bandwidth ένα crossbar δίκτυο διασύνδεσης δρομολογεί τις αναφορές μνήμης από τα νήματα στην μοναδική κρυφή μνήμη L2 που μοιράζονται όλα. Τέσσερις ανεξάρτητοι ελεγκτές μνήμης παρέχουν bandwidth της τάξης του 20Gbytes/s. Η προσέγγιση που έγινε στην περίπτωση του Niagara για αύξηση του throughput σε εμπορικές εφαρμογές για server έχει να κάνει με την αύξηση των νημάτων στον επεξεργαστή και με το υποσύστημα μνήμης με τέτοια κλιμάκωση ώστε να υποστηρίζει το μεγάλο bandwidth.

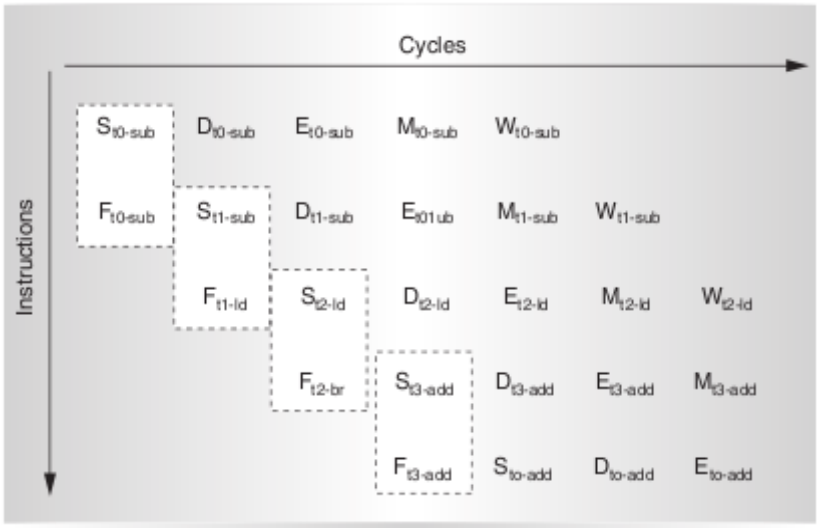
Όπως αναφέρθηκε οι επεξεργαστικοί πόροι του Niagara υποστηρίζουν τριάντα δύο νήματα. Η αρχιτεκτονική του επεξεργαστή χωρίζει τα νήματα αυτά ανά τέσσερα σε ένα group και το οποίο group μοιράζεται ένα κοινό pipeline γνωστός και ως sparc pipe. Έτσι προκύπτουν οκτώ τέτοιες ομάδες από νήματα. Κάθε sparc pipe έχει τις δικές του κρυφές μνήμες L1 εντολών και δεδομένων. Το hardware κρύβει τις καθυστερήσεις σε μνήμη και σε pipeline για κάποιο νήμα με το να δρομολογεί τα υπόλοιπα νήματα του συγκεκριμένου group με ποινή εναλλαγής δύο νημάτων ίση με μηδέν κύκλους. Τα τριάντα δύο νήματα μοιράζονται μια κρυφή μνήμη L2 μεγέθους 3-MByte. Η μνήμη αυτή είναι χωρισμένη σε τέσσερα μέρη και pipelined για περισσότερο bandwidth. Είναι 12-way associative έτσι ώστε να ελαχιστοποιήσει τα conflict misses μεταξύ του μεγάλου αριθμού των νημάτων. Ο εμπορικός server κώδικας υποστηρίζει κοινή διαχείριση δεδομένων, η οποία μπορεί να οδηγήσει στα υψηλά ποσοστά coherence misses. Στα συμβατικά SMP συστήματα που χρησιμοποιούν ξεχωριστούς επεξεργαστές με coherent δίκτυα διασύνδεσης, τα coherence misses βγαίνουν πέρα από τα χαμηλής συχνότητας off-chip buses ή συνδέσεις και μπορούν να εισαγάγουν στο σύστημα υψηλές καθυστερήσεις. Η σχεδίαση του Niagara με την κοινή on-chip κρυφή μνήμη απαλείφει αυτές τις αστοχίες αντικαθιστώντας τα με μικρής καθυστέρησης επικοινωνία στην κρυφή μνήμη. Το δίκτυο διασύνδεσης crossbar παρέχει την επικοινωνία μεταξύ των sparc pipes, τα μέρη της κρυφής μνήμης L2 και τους άλλους κοινούς πόρους στον επεξεργαστή. Μια ουρά δύο εισόδων για κάθε ζεύγος προέλευση - προορισμός είναι διαθέσιμη και μπορεί να φιλοξενήσει μέχρι και ενενήντα έξι δοσοληψίες προς οποιαδήποτε κατεύθυνση στο δίκτυο διασύνδεσης. Το crossbar δίκτυο διασύνδεσης παρέχει επίσης μια θύρα για την επικοινωνία με το I/O υποσύστημα. Ο ανταγωνισμός για θύρες προορισμού χρησιμοποιεί προτεραιότητα με βάση την ηλικία μιας αίτησης και έτσι εξασφαλίζει δίκαια δρομολόγηση μεταξύ των αιτούντων. Το crossbar είναι επίσης το σημείο που γίνεται το memory ordering για το μηχάνημα. Το memory interface είναι τέσσερα κανάλια από dual data rate 2 (DDR2) DRAM, για την υποστήριξη μέγιστου bandwidth παραπάνω από 20Gbytes/s και χωρητικότητας μέχρι και 128Gbytes.



Σχήμα 2.7: Σχεδιάγραμμα του επεξεργαστή Sun Niagara.

Η πολιτική επιλογής νημάτων σε επίπεδο πυρήνα

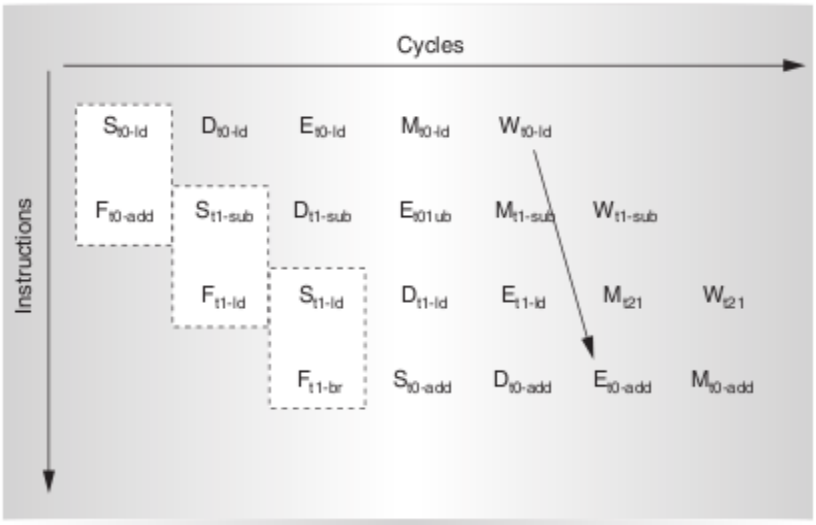
Η πολιτική επιλογής νημάτων εναλλάσσει τα διαθέσιμα νήματα σε κάθε κύκλο, δίνοντας προτεραιότητα στο λιγότερο πρόσφατα χρησιμοποιημένο νήμα. Τα νήματα μπορούν να γίνουν μη διαθέσιμα λόγω εντολών μεγάλης καθυστέρησης όπως οι loads, τα branches, οι πολλαπλασιασμοί και οι διαιρέσεις. Μπορούν επίσης να γίνουν μη διαθέσιμα λόγω εισαγωγής καθυστερήσεων στο pipeline όπως αστοχίες της κρυφής μνήμης, traps και συγκρούσεις πόρων. Ο δρομολογητής νημάτων θεωρεί ότι τα load θα είναι ευστοχίες κρυφής μνήμης και γι' αυτό μπορεί να εκδώσει μια εξαρτημένη εντολή από το ίδιο νήμα νωρίτερα. Παρ' όλ'αυτα σε ένα τέτοιο νήμα ορίζεται μικρότερη προτεραιότητα για έκδοση εντολής σε σύγκριση με ένα νήμα που μπορεί να εκδώσει μια κανονική εντολή.



Σχήμα 2.8: Επιλογή νημάτων. Όλα τα νήματα είναι διαθέσιμα.

Το Σχήμα 2.8 μας δείχνει τη λειτουργία στο pipeline όταν όλα τα νήματα είναι διαθέσιμα. Ο οριζόντιος άξονας δείχνει την πορεία μιας εντολής μέσα στο pipeline, ενώ ο κάθετος υποδεικνύει τις νέες εντολές που γίνονται fetch από την κρυφή μνήμη εντολών. Ο συμβολισμός S_{t0-sub} αναφέρεται σε μια εντολή αφαίρεσης από το thread0 που βρίσκεται στο στάδιο S (Thread-select) του pipeline. Σε αυτό το παράδειγμα το t0-sub προχωρά μέσα στο pipe. Καθώς τα άλλα τρία νήματα γίνονται διαθέσιμα, η μηχανή κατάστασης νημάτων επιλέγει το thread1 και αποδεσμεύει το thread0. Στον δεύτερο κύκλο παρόμοια το pipeline εκτελεί την t1-sub (στάδιο M) και επιλέγει για εκτέλεση την t2-ld (στάδιο S) για έκδοση στον επόμενο κύκλο. Όταν η t3-add είναι στο στάδιο S όλα τα νήματα έχουν εκτελεστεί και για τον επόμενο κύκλο επιλέγεται το λιγότερο πρόσφατα χρησιμοποιημένο το οποίο σε αυτή την περίπτωση είναι το thread0. Όταν το στάδιο thread-select επιλέγει ένα νήμα για εκτέλεση, το στάδιο fetch επιλέγει το ίδιο νήμα για πρόσβαση στην κρυφή μνήμη εντολών.

Στο Σχήμα 2.9 παρουσιάζεται η περίπτωση που μόνο δύο νήματα είναι διαθέσιμα. Εδώ το thread0 και το thread1 είναι διαθέσιμα ενώ τα thread2 και thread3 όχι. Το t0-ld στο στάδιο thread-select του παραδείγματος είναι μία πράξη μεγάλης καθυστέρησης. Γι' αυτό και προκαλεί αποδέσμευση του thread0. Παρ'όλα αυτά η t0-ld προχωρεί μέσα στο pipe. Στον επόμενο κύκλο αφού το άλλο διαθέσιμο είναι το thread1, ο δρομολογητής νημάτων το επιλέγει. Αυτή τη στιγμή δεν υπάρχουν άλλα νήματα διαθέσιμα και αφού η t1-sub είναι πράξη διάρκειας ενός κύκλου, το thread1 συνεχίζει να είναι το επιλεγόμενο νήμα και για τον επόμενο κύκλο. Η επόμενη εντολή είναι η t1-ld η οποία προκαλεί την αποδέσμευση του thread1 στον τέταρτο κύκλο. Έτσι τώρα μόνο το thread0 μπορεί να είναι διαθέσιμο και μόνο με την έκδοση εξαρτημένης εντολής αφού η t0-ld ακόμα εκκρεμεί. Έτσι αν η t0-ld είναι μία ευστοχία



Σχήμα 2.9: Επιλογή νημάτων. Μόνο δύο νήματα είναι διαθέσιμα.

κρυφής μνήμης, τότε το αποτέλεσμα της μπορεί να προωθηθεί στην εξαρτημένη t0-add στο στάδιο execute. Αν όμως η t0-ld προκαλέσει μια αστοχία κρυφής μνήμης, τότε έχουμε flushing της t0-add στο στάδιο thread-select και επανέκδοσης της όταν το αποτέλεσμα της t0-load επιστρέψει από την κρυφή μνήμη L2.

2.5 Προτάσεις / Ερευνητικές Προσπάθειες

Η βιομηχανία έχει υιοθετήσει τους Chip Multiprocessors αποτελούμενων από SMT πυρήνες για τα συστήματα γενικού σκοπού. Η πιο σημαντική χρήση τέτοιων επεξεργαστών, τουλάχιστον βραχυπρόθεσμα, θα είναι ως κεντρικοί υπολογιστές εργασίας που τρέχουν πολλαπλά νήματα στους διάφορους πυρήνες. Σε ένα τέτοιο περιβάλλον, η δρομολόγηση των φάσεων[2] από τα διαφορετικά νήματα διαδραματίζει έναν σημαντικό ρόλο στην ολική ρυθμαπόδοση. Η χαμηλότερη απόδοση επιτυγχάνεται όταν δρομολογούνται μαζί οι φάσεις από τα διαφορετικά νήματα που συγκρούονται για κάποιους συγκεκριμένους πόρους υλικού, σε αντίθεση με την περίπτωση όπου συμβατές φάσεις νημάτων δρομολογούνται μαζί στον ίδιο πυρήνα SMT. Η επίτευξη της δεύτερης περίπτωσης απαιτεί ακριβείς στατιστικές σε επίπεδο υλικού ανά φάση, που ο δρομολογητής λειτουργικού μπορεί να χρησιμοποιήσει για να προσδιορίσει γρήγορα πιθανή ασυμβατότητα των φάσεων διαφορετικών νημάτων, αποφεύγοντας με αυτό τον τρόπο το ενδεχόμενο κόστος σε επίδοση λόγω του συνωστισμού μεταξύ νημάτων.[7] Σε αυτή την ενότητα θα μελετήσουμε μερικές ερευνητικές προσπάθειες για την μελέτη αποδοτικών μεθόδων-αλγορίθμων για την επιλογή της συνδρομολόγησης νημάτων στους CMT επεξεργαστές. Σε γενικές γραμμές όλες οι μέθοδοι που περιγράφονται προσπαθούν να περιορίσουν τις επιπτώσεις της συνδρομολόγησης νημάτων στις περιοχές εισαγωγής καθυστέρησης όπως είναι οι κρυφές μνήμες,το pipeline κτλ.

2.5.1 Σημαντικότεροι Αλγόριθμοι Δρομολόγηση νημάτων

Το πρόβλημα της δρομολόγησης και συνύπαρξης νημάτων στον ίδιο πυρήνα σε ένα πολυεπεξεργαστικό, πολυνηματικό σύστημα αποτελεί αντικείμενο έρευνας, τόσο σε επίπεδο υλικού όσο και λειτουργικού συστήματος. Όσον αφορά το επίπεδο υλικού οι El-Moursy,Garg,Albonesi και Dwarkadas

[2]Με τον όρο φάσεις εννοούμε της περιόδους εκτέλεσης ενός προγράμματος που ποικίλλουν και έχουν διαφορετικά χαρακτηριστικά

[7] εξετάζοντας κάποιους αλγόριθμους σε επίπεδο υλικού προτείνουν τη χρήση του RIR (ready to in-flight instruction ratio) αλγόριθμου για τη λήψη αποδοτικών αποφάσεων δρομολόγησης σε έναν CMT επεξεργαστή. Ο συγκεκριμένος αλγόριθμος χρησιμοποιεί τον λόγο των έτοιμων εντολών στις ουρές εντολών προς τον αριθμό των εντολών που βρίσκονται μέσα στο pipeline από το στάδιο issue μέχρι το στάδιο write back, ως μέτρο δρομολόγησης, δρομολογώντας μαζί στον ίδιο πυρήνα τα νήματα με το μεγαλύτερο RIR πηλίκο. Στην ίδια έρευνα μελετώνται οι DCCS (Data Cache Conflict Scheduler) και IPCS (Instruction Per Cycle Scheduler) οι οποίοι θα αναπτυχθούν περαιτέρω σε επόμενη ενότητα, αφού είναι αυτοί που χρησιμοποιήθηκαν ουσιαστικά στη διπλωματική μας εργασία.

Οι Tam,Azimi και Stumm μελετούν την περίπτωση των SMP-CMP-SMT συστημάτων δηλαδή πολλαπλών CMT επεξεργαστών [29]. Περιγράφουν την σχεδίαση και υλοποίηση ενός σχήματος για την δρομολόγηση νημάτων στηριζόμενοι στην εύρεση κοινών μοτίβων σε πραγματικό χρόνο μεταξύ των νημάτων με την βοήθεια των μονάδες καταγραφής απόδοσης(PMUs) τα οποία υπάρχουν στους σύγχρονους επεξεργαστές. Πιο συγκεκριμένα προσπαθούν να δρομολογήσουν μαζί νήματα, δημιουργώντας ομάδες νημάτων στην οποία κάθε μια τοποθετούνται μαζί τα νήματα τα οποία είχαν παρόμοιο pattern προσβάσεων στην κρυφή μνήμη L2 του chip τους. Στην συνέχεια κάθε ομάδα τοποθετείται σε ένα chip. Με αυτό τον τρόπο προσπαθούν να γλιτώσουν τις απόμακρες προσβάσεις σε κρυφή μνήμη(μεταξύ διαφορετικών chip) οι οποίες εισάγουν την περισσότερη καθυστέρηση στο σύστημα.

Από την άλλη οι Fedorova, Seltzer και Smith μελετούν την δρομολόγηση νημάτων στους πολυπύρηνους επεξεργαστές σε επίπεδο λειτουργικού [10]. Προτείνουν την χρήση ενός αλγόριθμου που μειώνει τις επιπτώσεις από την άνιση χρήση της κρυφής μνήμης μεταξύ των νημάτων η οποία τελικά προκαλεί άνιση χρήση του επεξεργαστή, αντιστροφή της προτεραιότητας μεταξύ των νημάτων καθώς και μείωση της συνολικής απόδοσης του επεξεργαστή στο χρόνο. Ο αλγόριθμος ουσιαστικά υπολογίζει της τιμές των αστοχιών L2, του CPI rate, των εντολών και των κύκλων για ένα νήμα σε μια περίοδο, για την ιδανική περίπτωση που υπάρχει δίκαιη χρήση της κρυφής μνήμης μεταξύ των νημάτων. Στην συνέχεια συγκρίνει τις τιμές αυτές με τις πραγματικές και αναλόγως καθορίζει το κβάντο χρόνου για κάθε νήμα στην επόμενη περίοδο.

Οι De Vuyst, Kumar και Tullsen στο [6] μελετούν την μη ισοζυγισμένη δρομολόγηση νημάτων στους CMTs σε αντίθεση με τις συμβατικές τεχνικές δρομολόγησης που προσπαθούν να ισοκατανείμουν το νηματικό φορτίο στους πυρήνες. Υποστηρίζουν ότι αυτές οι συμβατικές τεχνικές μηδενίζουν ένα από τα σημαντικότερα πλεονεκτήματα των αρχιτεκτονικών αυτών - την ικανότητα της χρήσης μη ισοζυγισμένων δρομολογήσεων για την δέσμευση του κατάλληλου ποσοστού πόρων για κάθε νήμα. Παρ' όλα αυτά συμπεραίνουν ότι η χρήση τέτοιων μη ισοζυγισμένων μεθόδων εισάγει κάποιες δυσκολίες με την μεγαλύτερη να είναι ότι το διάστημα αναζήτησης για όλες τις δρομολογήσεις (ισοζυγισμένων και μη) είναι μεγαλύτερο από το αντίστοιχο για τις ισοζυγισμένες. Πιο συγκεκριμένα προτείνουν τεχνικές δρομολόγησης σε επίπεδο λειτουργικού που οδηγούν το σύστημα σε καλύτερες δρομολογήσεις νημάτων. Οι τεχνικές που προτείνονται είναι:

- **Η ισοζυγισμένη συμβίωση.** Τα νήματα κατανέμονται τυχαία στους πυρήνες με την προϋπόθεση ότι στους πυρήνες ανατίθεται ίσος αριθμός νημάτων και πάντα μαζί συνδρομολογούνται φιλικά νήματα, δηλαδή νήματα που ανήκουν σε ομάδες εφαρμογών που τρέχουν καλά μαζί.

- **Η συμβίωση.** Παρόμοια με την ισοζυγισμένη χωρίς τον περιορισμό της ισοκατανομής των νημάτων στους πυρήνες.

- **Ισοζυγισμένη τυχαία.**Εντελώς τυχαία ισοζυγισμένη κατανομή νημάτων στους πυρήνες.

- **Prefer Last - Numbers.** Μια νέα δρομολόγηση είναι ίδια με την προηγούμενη αν ο αριθμός των νημάτων στους πυρήνες είναι ο ίδιος με πριν. Με αυτό τον τρόπο προσπαθεί να διατηρήσει το ίδιο ποσοστό φορτίου σε κάθε πυρήνα.

- **Prefer Last - Swap.** Η νέα δρομολόγηση προκύπτει με την ανταλλαγή δύο νημάτων που βρίσκονται σε δύο διαφορετικούς πυρήνες.

- **Prefer Last - Move.** Η νέα δρομολόγηση προκύπτει με την μετακίνηση ενός τυχαίου νήματος από ένα πυρήνα σε ένα πυρήνα που δεν τρέχει κάποιο νήμα.

- **Electron - Performance.** Το συνολικό IPC κάθε πυρήνα υπολογίζεται σε κάθε περίοδο. Στην επόμενη περίοδο ένα νήμα από τον πυρήνα με το ψηλότερο IPC μεταναστεύει στον πυρήνα με το χαμηλότερο εφόσον υπάρχει ελεύθερο thread context στον δεύτερο. Σε αντίθετη περίπτωση η δρομολόγηση παραμένει η ίδια.

- **Electron - Power.** Αντίστοιχα με την Electron - Performance μόνο που αντί το IPC για κάθε πυρήνα υπολογίζεται η κατανάλωση ενέργειας και ο πυρήνας στον οποίο θα μεταναστεύσει το νήμα δεν ήταν ανενεργός την προηγούμενη περίοδο.

- **Electron - EDP.** Όπως στις δύο προηγούμενες περιπτώσεις μόνο που εδώ το μέγεθος που ρυθμίζει την δρομολόγηση είναι το γινόμενο καθυστέρησης και κατανάλωσης ισχύς σε κάθε πυρήνα (EDP) . Πάλι ένα τυχαίο νήμα μεταναστεύει από τον πυρήνα με την μικρότερη τιμή EDP σε αυτό με την μεγαλύτερη. Ένας ανενεργός πυρήνας σε μια περίοδο δρομολόγησης παρουσιάζει την μεγαλύτερη τιμή EDP. Έτσι η τεχνική αυτή ενθαρρύνει την μη ύπαρξη ανενεργού πυρήνα σε κάθε περίοδο δρομολόγησης.

Το συμπέρασμα στο οποίο οδηγήθηκαν μετά την πιο πάνω έρευνα είναι ότι δεν φαίνεται ξεκάθαρα πια προσέγγιση (ισοζυγισμένη ή μη ισοζυγισμένη) όσον αφορά το νηματικό φορτίο στον επεξεργαστή είναι η καλύτερη. Αυτό καθορίζεται σε κάθε περίπτωση από το εκτελούμενο workload και έτσι μόνο μια δυναμική συμπεριφορά του αλγόριθμου σε σχέση με την συμπεριφορά των εκάστοτε προγραμμάτων μπορεί να έχει τις επιθυμητές βελτιώσεις.

Οι Chen και συνεργάτες στο [5] προτείνουν την χρήση του δρομολογητή PDF(Parallel Depth First) στους CMPs, ο οποίος σχεδιάστηκε για να ενθαρρύνει τα συνεργαζόμενα νήματα στο τσιπ να μοιράζονται εποικοδομητικά την L2 on-chip κρυφή μνήμη. Ουσιαστικά γίνεται σύγκριση του εν λόγω δρομολογητή με τον Work Stealing (WS) δρομολογητή. Ο Work Stealing (WS) είναι ένας δημοφιλής άπληστος (greedy) αλγόριθμος δρομολόγησης νημάτων για τα πολυνηματικά προγράμματα με καλές συμπεριφορές στη χρήση κρυφής και κύριας μνήμης. Ο αλγόριθμος διατηρεί μια ουρά εργασίας για κάθε επεξεργαστή. Με την δημιουργία(fork) ενός νέου νήματος,το δημιουργημένο νήμα τοποθετείται στην κορυφή της τοπικής ουράς. Όταν μια εργασία ενός επεξεργαστή ολοκληρωθεί, ο επεξεργαστής ψάχνει για μία εργασία έτοιμη για εκτέλεση ψάχνοντας πρώτα την κορυφή της τοπικής ουράς. Εάν μια τέτοια εργασία βρεθεί, την αφαιρεί από την ουρά και την εκτελεί. Εάν η τοπική ουρά είναι κενή ελέγχει τις ουρές εργασίας των άλλων επεξεργαστών και “κλέβει” μια εργασία από το τέλος της πρώτης μη κενής ουράς που βρίσκει. Ο WS είναι ένας ελκυστικός αλγόριθμος επειδή όταν υπάρχει αρκετός παραλληλισμός, η ”κλοπή” είναι αρκετά σπάνια και, επειδή οι εργασίες σε μια ουρά αναμονής συσχετίζονται, υπάρχει καλή συγγένεια μεταξύ των εργασιών που εκτελούνται από οποιοδήποτε επεξεργαστή. Εντούτοις, ο WS δεν σχεδιάστηκε για την εποικοδομητική διανομή της κρυφής μνήμης, επειδή οι επεξεργαστές τείνουν να έχουν εντελώς διαφορετικά σύνολα εργασίας.
Από την άλλη ο PDF (Parallel Depth First) είναι ένας άλλος άπληστος αλγόριθμος βασισμένος στην πιο κάτω ιδιότητα. Τα σημαντικά σειριακά προγράμματα έχουν ήδη βελτιστοποιηθεί σε μεγάλο βαθμό για να έχουν καλή επίδοση κρυφής μνήμης σε ένα πυρήνα, με την χρήση μικρών σετ εργασίας, με καλή χρονική και τοπική της χρήση κτλ. Στον PDF όταν ένας πυρήνας ολοκληρώσει μια εργασία στην συνέχεια του ανατίθεται μια εργασία που είναι έτοιμη για εκτέλεση και που το σειριακό πρόγραμμα θα είχε ολοκληρώσει το νωρίτερο δυνατό. Έτσι κατά κάποιο τρόπο ο PDF συνδρομολογεί εργασίες οι οποίες προκύπτουν ακολουθώντας την πορεία εκτέλεσης τους στο αντίστοιχο σειριακό πρόγραμμα. Έτσι για τα σειριακά προγράμματα που εμφανίζουν καλή επίδοση κρυφής μνήμης, ο PDF προσφέρει την αντίστοιχη παράλληλη καλή επίδοση κρυφής μνήμης

Σε αρκετές διαμορφώσεις του CMP ο PDF δρομολογητής έφτασε αλλά και ξεπέρασε τον WS για περιπτώσεις fine-grained παράλληλων προγραμμάτων. Με την πιο αποτελεσματική χρήση των πόρων κρυφής μνήμης ο δρομολογητής PDF επιτρέπει την χρήση περισσότερων πυρήνων με κάποιο κόστος σε πόρους κρυφής μνήμης που δεν είναι τόσο σημαντικό όσο το όφελος που εισάγεται με την χρήση του PDF. Τέλος συμπέραναν ότι το διάστημα δρομολόγησης των εργασιών στους πυρήνες είναι σημαντικός παράγοντας για την επίδοση της κρυφής μνήμης και πρότειναν μια αυτόματη προσέγγιση για επιλογή αυτού του διαστήματος κατά την χρήση του PDF.

Τέλος οι Rajan, Sondag και Krishnamurthy στο [26] περιγράφουν μια τεχνική για την ανάθεση νήματος σε πυρήνα στις ετερογενείς πολυπύρηνες αρχιτεκτονικές. Το πρώτο βήμα της τεχνικής αυτής είναι η προσέγγιση της συμπεριφοράς φάσης για κάποιο πρόγραμμα. Ακολούθως αυτή η συμπεριφορά καθώς και τα χαρακτηριστικά εκτέλεσης για ένα σετ αντιπροσωπευτικών φάσεων του προγράμματος επεξεργάζονται κατά την εκτέλεση για να αποφασιστούν αναθέσεις νημάτων στους πυρήνες που πιθανόν να εισάγουν βελτιώσεις στις μετέπειτα φάσεις του προγράμματος. Με την παρακολούθηση της εκτέλεσης ενός μικρού αντιπροσωπευτικού σετ σε αντίθεση με την συνολική εκτέλεση μειώνεται η καθυστέρηση της εξομοίωσης. Έτσι εφαρμόζοντας αυτή την τεχνική στους πυρήνες των λειτουργικών συστημάτων και στους μεταγλωττιστές η χρήση της μπορεί να γίνεται αυτόματα χωρίς την παρέμβαση του χρήστη και ως αποτέλεσμα προγράμματα γραμμένα από τον χρήστη να μπορούν να εκμεταλλευτούν πλήρως την δυνατότητα της ετερογενούς πολυπύρηνης αρχιτεκτονικής.

Κεφάλαιο 3

Εξομοιωτές-Εργαλεία

3.1 Γενικά

Στην αρχιτεκτονική υπολογιστών ένας εξομοιωτής είναι ένα πακέτο λογισμικού που μοντελοποιεί συσκευές ή συστατικά (components) ενός υπολογιστικού συστήματος, προκειμένου να προβλέψει την έξοδο ή διάφορες μετρικές απόδοσης για μια δεδομένη είσοδο. Ένας τέτοιος εξομοιωτής ενδέχεται να ενδέχεται να μοντελοποιεί από μόνο ένα επεξεργαστή έως ένα ολόκληρο σύστημα που τυπικά περιλαμβάνει ένα σύνολο από επεξεργαστές, ένα σύστημα μνήμης και κάποιες συσκευές εισόδου/εξόδου.

Η έντονη ερευνητική δραστηριότητα στον τομέα της αρχιτεκτονικής υπολογιστών και τα προβλήματα που παρέχουν οι εξομοιωτές, συντέλεσαν στην ανάπτυξη πολλών διαφορετικών υλοποιήσεων. Γενικά οι εξομοιωτές αρχιτεκτονικής υπολογιστών ταξινομούνται σε πολλές διαφορετικές κατηγορίες ανάλογα με το χαρακτηριστικό που ενδιαφέρει. Πιο συγκεκριμένα:

- **Ακρίβεια:** Οι λειτουργικού (functional) εξομοιωτές δίνουν μεγαλύτερη προτεραιότητα στην ταχύτητα της εξομοίωσης, επιτυγχάνοντας την ίδια λειτουργικότητα με τα συστατικά του συστήματος που μοντελοποιούν. Ένας εξομοιωτής συνόλου εντολών υπάγεται στην κατηγορία των functional simulators και αποτελεί ένα μοντέλο εξομοίωσης που μιμείται την συμπεριφορά μιας συσκευής ή ενός συστατικού (component) κάποιου συστήματος παρέχοντας την ίδια λειτουργικότητα και τα ίδια αποτελέσματα, χωρίς ωστόσο να δίνεται ιδιαίτερη έμφαση στην ακρίβεια με την οποία λειτουργεί το μοντέλο εσωτερικά. Για παράδειγμα ένας επεξεργαστής μπορεί να υλοποιηθεί σε έναν functional simulator, διαβάζοντας εντολές και διατηρώντας εσωτερικές μεταβλητές που αντιπροσωπεύουν τους καταχωρητές του επεξεργαστή. Από την άλλη, οι εξομοιωτές χρονισμού (timing simulators) δίνουν έμφαση στην ακριβή αναπαραγωγή των χαρακτηριστικών απόδοσης/χρονισμού των συστημάτων που μοντελοποιούν.

- **Εύρος:** Η εξομοίωση μπορεί να περιλαμβάνει από μόνο έναν επεξεργαστή (micro-architecture simulator) έως ένα ολόκληρο σύστημα (full-system simulator). Συχνά, ένας εξομοιωτής πλήρους συστήματος περιλαμβάνει ορισμένους εξομοιωτές συνόλου εντολών για να εξομοιώσει συσκευές όπως οι επεξεργαστές, που αποτελούν τμήματα του συστήματος.

- **Είσοδος:** Οι trace-driven (ή event-driven) εξομοιωτές χρησιμοποιούν κατά τη λειτουργία τους ίχνη ή γεγονότα (traces/events). Τα ίχνη ή γεγονότα είναι εκ των προτέρων καταγεγραμμένες ροές εντολών που αντιστοιχούν σε κάποια συγκεκριμένη είσοδο, επομένως δε χρειάζεται η μοντελοποίηση και εξομοίωση των επεξεργαστικών στοιχείων. Τέτοιοι εξομοιωτές στηρίζονται σε υπάρχοντα εργαλεία για να συλλέξουν τα ίχνη διευθύνσεων μνήμης και να τα καταγράψουν σε κάποιο αρχείο για μετέπειτα χρήση. Αντίθετα οι καθοδηγούμενοι από την εκτέλεση εξομοιωτές (execution-driven simulators), στηρίζονται σε λειτουργικά μοντέλα για να εκτελέσουν τον δυαδικό κώδικα μιας εφαρμογής. Οι διευθύνσεις μνήμης που παράγονται από το λειτουργικό μοντέλο τροφοδοτούνται σε πραγματικό χρόνο σε ένα εξομοιωτή συστημάτων μνήμης που διαμορφώνεται μέσα στο λειτουργικό μοντέλο. Γενικά, η trace-driven εξομοίωση είναι πιο γρήγορη και έχει γίνει δημοφιλής για την πραγματοποίηση μελετών σε υπο σχεδίαση συστήματα. Από την άλλη μεριά, απαιτεί συχνά περίπλοκα εργαλεία για την απόκτηση του trace από την

προς μελέτη αρχιτεκτονική συνόλου εντολών, καθώς και τη μοντελοποίηση της τελευταίας με ακρίβεια.

Οι εξομοιωτές αρχιτεκτονικής έχουν γίνει αναγκαία εργαλεία της έρευνας στον τομέα της αρχιτεκτονικής υπολογιστών κυρίως λόγω των παρακάτω πλεονεκτημάτων τους:

- Προσφέρουν την δυνατότητα αξιολόγησης διαφορετικών σχεδίου υλικού χωρίς να απαιτείται η κατασκευή ακριβών πραγματικών συστημάτων, καθώς και την δυνατότητα προσέγγισης μη υπαρχόντων πραγματικών υπολογιστικών συστημάτων και συστατικών.
- Επιτρέπουν την συλλογή λεπτομερών μετρικών απόδοσης. Συχνά μία και μόνο εκτέλεση σε έναν εξομοιωτή δύναται να δημιουργήσει ένα μεγάλο σύνολο δεδομένων απόδοσης.
- Καθιστά πολύ ευκολότερη τη διόρθωση και απομάκρυνση λαθών. Σε πραγματικά συστήματα αυτό συνήθως απαιτεί επανεκκίνηση και επανεκτέλεση του κώδικα προκειμένου να αναπαραχθούν τα ίδια λάθη-προβλήματα. Αντίθετα, πολλοί εξομοιωτές έχουν ένα πλήρως ελεγχόμενο περιβάλλον και επιτρέπουν στους μηχανικούς λογισμικού να εκτελούν τον κώδικα προς τα πίσω, όταν ανιχνευθεί κάποιο σφάλμα.

Όσον αφορά το θέμα της απόδοσης που παρουσιάζουν οι εξομοιωτές στην μοντελοποίηση των αρχιτεκτονικών, αυτό υπογραμμίστηκε από την αύξηση του μεγέθους και της πολυπλοκότητας των φόρτων εργασίας που χρησιμοποιούνται ως είσοδοι. Η απόδοση ενός εξομοιωτή μπορεί να ποσοτικοποιηθεί με χρήση του παράγοντα επιβράδυνσης (slowdown factor). Για τυπικές υλοποιήσεις εξομοιωτών έχουν καταγραφεί παράγοντες επιβράδυνσης από 10 εώς 100, ανάλογα με τον τύπο του, την εφαρμογή εισόδου και διαμόρφωση της αρχιτεκτονικής που εξομοιώνεται. Στις χειρότερες περιπτώσεις, ένας εξομοιωτής που πραγματοποιεί λεπτομερή εξομοίωση ανά κύκλο ενδέχεται να παρουσιάζει παράγοντα επιβράδυνσης της τάξης του 2000. Η απόδοση ενός εξομοιωτή και η καθυστέρηση που επιφέρει στην εκτέλεση, αποτελούν σήμερα έναν από τους καθοριστικότερους παράγοντες για την επιλογή ή την απόρριψη του.

3.2 Ο εξομοιωτής συστήματος Simics

3.2.1 Γενικά

Ο Simics [8] αποτελεί μια πλατφόρμα για πλήρη εξομοίωση συστήματος που χρησιμοποιείται για να επιτρέψει την εικονική ανάπτυξη λογισμικού. Ο Simics παρέχει το εικονικό hardware, που τρέχει τα ίδια εκτελέσιμα αρχεία με το φυσικό υλικό. Ο στόχος της εικονικής ανάπτυξης λογισμικού είναι να ενισχυθεί η διαδικασία ανάπτυξης λογισμικού με τη μείωση της εξάρτησης από το φυσικό υλικό.

Στην εξομοίωση πλήρους συστήματος συνδυάζεται ένας γρήγορος instruction specific εξομοιωτής ενός αρχιτεκτονικού στόχου με τα πρότυπα όλων των συσκευών του φυσικού υλικού. Κατά συνέπεια, το λογισμικό δοκιμάζεται σε ένα εικονικό σύστημα που είναι λειτουργικά ίδιο με το φυσικό σύστημα, ικανό να τρέχει τα ίδια εκτελέσιμα, συμπεριλαμβανομένων των οδηγών συσκευών, των λειτουργικών συστημάτων, και των εφαρμογών.

Ο Simics σχεδιάστηκε για τη γρήγορη εκτέλεση του κώδικα στο εικονικό σύστημα, με εκτέλεση αρκετών εκατομμυρίων των εξομοιουμένων εντολών ανά δευτερόλεπτο, επιτρέποντας έτσι την εκτέλεση πλήρων φορτίων σε ρεαλιστικό χρόνο. Η βασική τεχνική που διευκολύνει την εξομοίωση μεγάλων εκτελέσεων είναι ότι ο Simics χρησιμοποιεί μικρότερη κλίμακα χρονισμού από τον χρονισμό του φυσικό υλικό. Κατά συνέπεια, ο Simics είναι μια κατάλληλη πλατφόρμα για τη δοκιμή των λειτουργικών πτυχών του συνδεόμενου λογισμικού, που οποίο αποτελεί το 80-90% της ανάπτυξης, της επικύρωσης, και της δοκιμής των ενσωματωμένων συστημάτων πραγματικού χρόνου.
Ο Simics προσφέρει ένα βολικό περιβάλλον εξομοίωσης και debugging λογισμικού, με χαρακτηριστικά γνωρίσματα όπως η στιγμιαία στάση της εκτέλεσης, τα προκαθορισμένα test cases, ο πλήρης

ντετερμινισμός, τα checkpoints, η επανεκκίνηση, το reverse debugging και το reverse execution. Ο Simics σαν εξομοιωτής προσφέρει ολοκληρωτικό έλεγχο σε θέματα των υπολογιστικών συστημάτων προς εξομοίωση, επιτρέποντας την ακριβή εμφάνιση των ελαττωμάτων και το scripting των δοκιμών. Τα ελαττώματα μπορούν να είναι στις εσωτερικές συσκευές, μνήμες, επεξεργαστές, και στους διαδρόμους και δίκτυα που τα συνδέουν. Ο Simics περιλαμβάνει έναν πλήρη γλωσσικό Python διερμηνέα, επιτρέποντας το ισχυρό scripting.

Η Virtutech περιλαμβάνει μια μεγάλη βιβλιοθήκη συσκευών προς εξομοίωση, διαθέσιμων στους χρήστες για τη δημιουργία ενός ολοκληρωμένου συστήματος, που μειώνει το χρόνο για να χτιστεί ένα πρότυπο ενός συγκεκριμένου συστήματος. Παραδείγματα εξομοιούμενων αρχιτεκτονικών περιλαμβάνουν τους SPARC V8/Leon2, PowerPC 750, x86 και άλλους επεξεργαστές. Υπάρχουν επίσης μοντέλα των κοινών στοιχείων ενός συστήματος όπως οι σειριακές θύρες, οι μνήμες SDRAM, οι μνήμες Flash και οι ελεγκτές συστήματος. Οι διάδρομοι και τα δίκτυα προς εξομοίωση που είναι διαθέσιμα περιλαμβάνουν τα MIL-STD 1553, ARINC 429, και Ethernet. Οι διάδρομοι και τα δίκτυα προς εξομοίωση μπορούν να συνδεθούν με τα πραγματικά δίκτυα για εξομοιώσεις που περιλαμβάνουν μαζί εικονικούς και φυσικούς κόμβους.

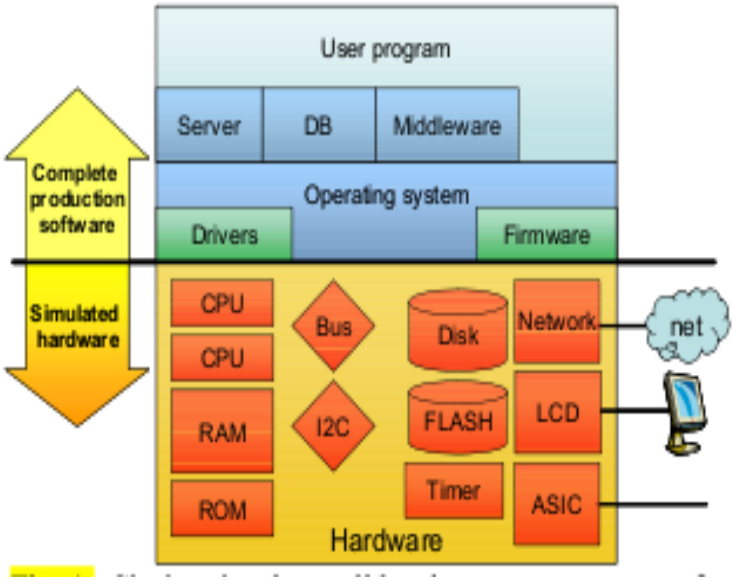
Τα μοντέλα των νέων στοιχείων υλικού αναπτύσσονται από τους χρήστες του Simics ή από την Virtutech σε μια πολύ αποδοτική γλώσσα προγραμματισμού αποκαλούμενη DML, (Device Modelling Language - Γλώσσα Μοντελοποίησης Συσκευών). Οι χρήστες μπορούν να επεκτείνουν τον Simics, χρησιμοποιώντας ένα καλά δοκιμασμένο και ισχυρό API(Application Programming Interface - Διεπαφή προγραμματισμού εφαρμογών). Οι επεκτάσεις χρηστών μπορούν να είναι νέα πρότυπα υλικού είτε νέες ικανότητες συστημάτων εξομοίωσης όπως τα inspection and fault injection modules. Χάρη στην DML και τη μεγάλη βιβλιοθήκη των συστατικών διαθέσιμων από την Virtutech, ο Simics προσφέρει τη δυνατότητα να δημιουργηθούν γρήγορα τα στοιχεία του υλικού. Τα μοντέλα μπορούν ακόμη και να χτιστούν πριν την φυσική τους διαθεσιμότητα, επιτρέποντας στην ανάπτυξη λογισμικού να αρχίσει, προτού το μοντέλο είναι φυσικά διαθέσιμο. Χαρακτηριστικό παράδειγμα είναι η μεταφορά του λειτουργικού NetBSD στην αρχιτεκτονική AMD64 που έγινε στο περιβάλλον Simics πριν την κυκλοφορία του συγκεκριμένου chip [1].

3.2.2 Χρονισμός Εξομοίωσης

Σε ένα μοντέλο του Simics που χρησιμοποιείται για τη εικονική ανάπτυξη λογισμικού, τα λειτουργικά αποτελέσματα των εντολών και οι προσβάσεις στο υλικό είναι ίδια με εκείνα ενός πραγματικού μηχανήματος (απαραίτητες προϋποθέσεις προκειμένου να μπορούν να τρέξουν τα πραγματικά εκτελέσιμα) κάτι όμως που δεν ισχύει για τον χρονισμό, που διαφέρει. Πιο συγκεκριμένα η εξομοίωση δεν έχει ακρίβεια κύκλου. Αυτή η απλοποίηση του χρονισμού είναι απαραίτητη προκειμένου να επιτευχθεί μια αρκετά γρήγορη εκτέλεση στο εικονικό περιβάλλον.

Αναφορικά με τους επεξεργαστές, ο Simics δεν προσπαθεί να διαμορφώσει τον ακριβή συγχρονισμό κύκλων της εκτέλεσης ενός κώδικα όπως υπαγορεύεται από τη δομή των pipelines στους επεξεργαστές, τις ιεραρχίες κρυφής μνήμης τους και την κίνηση στου διαδρόμους μνήμης. Τέτοια λεπτομερή πρότυπα επεξεργαστών είναι και δύσκολο να σχεδιαστούν αλλά και αργά στην εκτέλεσή τους, αφού περιέχουν πολλή λεπτομέρεια. Η ταχύτητα ενός επεξεργαστή κατά την εκτέλεση εντολών στο περιβάλλον του Simics ορίζεται από τη συχνότητα ρολογιού του και το CPI του. Τυπικά η ρύθμιση του CPI στο ένα, δηλ. μια εντολή ανά κύκλο λειτουργεί καλά. Η ρύθμιση του CPI σε άλλες τιμές όπως 2 ή και 3/2 μπορεί να βοηθήσει τον Simics περισσότερο,για την προσέγγιση στη μέση ταχύτητα εκτέλεσης εντολής ενός φυσικού επεξεργαστή. Η εμπειρία έδειξε ότι αυτή η διαδικασία λειτουργεί καλά για σχεδόν όλους τους φόρτους εργασίας και σχεδόν όλες τις περιπτώσεις.

[1] http://www.netbsd.org/ports/amd64/



Σχήμα 3.1: Σχεδιάγραμμα λειτουργίας Simics.

Στους διαδρόμους διασύνδεσης (interconnection buses), ο λεπτομερής μηχανισμός διαιτησίας και μεταφοράς δεδομένων των διαδρόμων μνήμης (memory buses) δεν υλοποιείται αυστηρά στον Simics. Η αφαιρετική διαμόρφωση που μοντελοποιείται είναι μάλλον μια χαρτογράφηση της μνήμης, όπου αναγνώσεις και εγγραφές σε μια διεύθυνση οδηγούνται άμεσα στις περιοχές μνήμης ή τις συσκευές που αντιστοιχούν σε εκείνη τη διεύθυνση. Αυτό μπορεί να γίνει κατά τρόπο πολύ αποδοτικό, που εξασφαλίζει έτσι υψηλή απόδοση στην εξομοίωση. Όλες οι διαδικασίες μνήμης ολοκληρώνονται αμέσως, γεγονός που είναι φυσικό σε ένα πρότυπο συναλλαγών (transaction-based) όπως αυτό που υιοθετείται στον Simics. Επιπρόσθετα στον Simics μπορούν να καταγραφούν και να συλλέγουν όλα τα ίχνη πρόσβασης στη μνήμη και στις συσκευές, για την μετέπειτα ανάλυση των απαιτήσεων σε εύρος ζώνης, χωρίς ο Simics να χρειάζεται να περιορίζει το εύρος ζώνης των διαδρόμων κατά τη διάρκεια της εξομοίωσης.

3.2.3 Επεκτασιμότητα στον Simics

Η δυνατότητα του Simics να τρέχει μέχρι και σύνθετες ετερογενείς εξομοιώσεις συστημάτων συμπληρώνεται από APIs που έχουν σκοπό να διευκολύνουν το στόχο της ένταξης του υπάρχοντα κώδικα και συσκευών εξομοίωσης στο περιβάλλον εξομοίωσης Simics. Ο πυρήνας των APIs του Simics επιτρέπει στην εξομοίωση να επεκταθεί από modules λειτουργίας χρηστών. Αυτός ο τύπος λειτουργίας είναι απεριόριστος αλλά συχνά περιλαμβάνει προσαρμογή εντολών, έλεγχο/αυτοματοποίηση ή εξέταση του περιβάλλοντος εξομοίωσης. Οι βασικές ιδιότητες του πυρήνα των Simics APIs είναι:

- Είναι προσβάσιμος από την C,την C++,την Python,την εντολή γραμμών του Simics και την DML.
- Προγραμματιστική και σεναριοποιημένη πρόσβαση που αφορά στην κατάσταση όλων των προτύπων.
- Πρόσβαση και ενημέρωση όσον αφορά στη λειτουργία του OS, των διακοπών και των breakpoints αλλά και για οποιοδήποτε άλλο γεγονός στο σύστημα.
- Είναι σχεδιασμένος για να είναι αυθαίρετα επεκτάσιμος από modules χρηστών, συμπεριλαμβανομένης και της προσθήκης των νέων διεπαφών και γεγονότων που άλλα modules μπορούν

να χρησιμοποιήσουν.

- Πλήρως ετερογενής και ανεξάρτητος από το υλικό και το λογισμικό των στόχων προς εξομοίωση.
- Αντιμετωπίζει τα ζητήματα word-length και endianness καθιστώντας το σύστημα των στόχων εξομοίωσης πλήρως εικονικό χωρίς τις εξαρτήσεις που προκύπτουν από την αρχιτεκτονική του μηχανήματος που φιλοξενεί την εξομοίωση.
- Αποδεδειγμένος σε χρήση και πλήρως αναπτυσσόμενος από το 1991.
- Υποστηρίζει πολυνηματική και κατανεμημένη εξομοίωση.

3.3 GEMS - General Execution-driven Multiprocessor Simulator

3.3.1 Γενικά

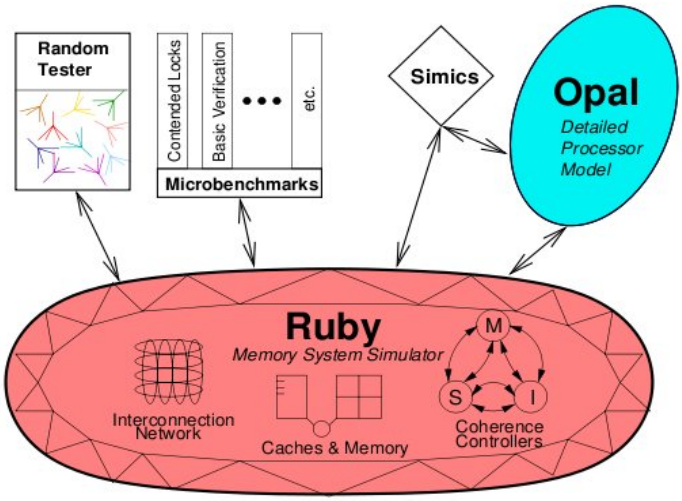
Ο GEMS (General Execution-driven Multiprocessor Simulator)[20] αποτελεί ένα εργαλείο εξομοίωσης το οποίο αναπτύχθηκε στο Πανεπιστήμιο του Wisconsin. Δημιουργήθηκε για τη μελέτη της απόδοσης των πολυεπεξεργαστικών συστημάτων που χρησιμοποιούνται σε εξυπηρετητές βάσεων δεδομένων και διαδικτύου. Αποτελεί ένα σετ από modules για τον πλήρη εξομοιωτή συστήματος Simics για την μοντελοποίηση του χρονισμού του υποσυστήματος μνήμης και των πολυεπεξεργαστών. Αυτή η διαμόρφωση του εξομοιωτή επιτρέπει την εκτέλεση πειραμάτων με την χρήση εμπορικών workloads σε πιο περιορισμένο μέγεθος.

3.3.2 Διαχωρισμός λειτουργικότητας και χρονισμού εξομοίωσης

Ο GEMS διαχωρίζει το λειτουργικό κομμάτι της εξομοίωσης από αυτό του χρονισμού. Όπως αναφέρθηκε πιο πριν ο GEMS αποτελεί ένα σετ από modules για τον Simics στον οποίο διάφορα modules χρονισμού μπορούν να φορτωθούν δυναμικά. Αυτός ο διαχωρισμός διευκολύνει την αποτελεσματικότητα και τη σταθερότητα ενός λειτουργικού εξομοιωτή. Μάλιστα ένας εξομοιωτής σε μορφή modules όπως ο GEMS παρέχει περισσότερη ευελιξία στην εξομοίωση διάφορων συσκευών ενός συστήματος σε διαφορετικά επίπεδα λεπτομέρειας. Αφού ο Simics είναι ένας σταθερός λειτουργικός εξομοιωτής που μπορεί να φορτώσει ένα λειτουργικό σύστημα,χρησιμοποιήθηκε σαν το λειτουργικό κομμάτι της εξομοίωσης, δηλαδή το αποτέλεσμα της εκτέλεσης μιας εντολής είναι απόλυτα εξαρτημένο από τον Simics. Από την άλλη τα modules χρονισμού του GEMS είναι αυτά που σε συνεργασία με τον Simics ελέγχουν το πότε πρέπει να εκτελέσει μια εντολή. Εδώ φαίνεται ξεκάθαρα η λειτουργική αποδοτικότητα και ευκολία ενός εξομοιωτή που διαχωρίζει αυτά τα δύο κομμάτια, από ένα εξομοιωτή που μοντελοποιεί και τον χρονισμό αλλά και την λειτουργική ορθότητα κάθε συνάρτησης που περιλαμβάνεται στην πλήρη εξομοίωση συστήματος. Έτσι π.χ. ένα μικρό λάθος στον χειρισμό ενός I/O αιτήματος ή στην μοντελοποίηση μιας ειδικής περίπτωσης αριθμητικής κινούμενης υποδιαστολής μάλλον δεν θα επηρεάσει το timing κομμάτι της εξομοίωσης σε αντίθεση φυσικά με το λειτουργικό κομμάτι της εξομοίωσης το οποίο επηρεάζεται, πράγμα το οποίο μπορεί να εμποδίσει την εξομοίωση να συνεχίσει. Επιτρέποντας λοιπόν πάντοτε στον Simics να αποφασίζει για το αποτέλεσμα μιας εκτέλεσης, το προς εξομοίωση πρόγραμμα πάντα θα τρέχει σωστά.

Ο GEMS είναι ένας execution-driven εξομοιωτής γεγονός που δηλώνει ότι παρόλο που γίνεται διαχωρισμός σε δύο μέρη, λειτουργικό και χρονισμού, εντούτοις το πρώτο επηρεάζεται από το δεύτερο επιτρέποντας στο σύστημα να παίρνει αποτελέσματα εξαρτημένα από το χρονισμό. Για παράδειγμα, το μοντέλο χρονισμού θα καθορίσει το νικητή μεταξύ δύο επεξεργαστών που προσπαθούν να έχουν πρόσβαση στο ίδιο software lock στην μνήμη. Έτσι αφού η διεπαφή χρονισμού αποφασίζει το πότε θα προχωρήσει το λειτουργικό κομμάτι της εξομοίωσης, έχουμε εξάρτηση της εξομοίωσης σε αυτήν. Αντίθετα μια trace-driven εξομοίωση αποτυγχάνει να καταγράψει τέτοια σημαντικά γεγονότα.

3.3.3 Γενική περιγραφή του GEMS



Σχήμα 3.2: Τα κύρια μέρη του GEMS

Η καρδιά του GEMS είναι το σύστημα μνήμης με το όνομα Ruby το οποίο θα αναπτυχθεί περισσότερο αργότερα αφού αποτέλεσε το κύριο κομμάτι επέκτασης για υποστήριξη του ζητούμενου σε αυτή την διπλωματική εργασία. Στο σχεδιάγραμμα βλέπουμε ότι υπάρχουν διάφορες πηγές λειτουργιών μνήμης στη Ruby. Πιο συγκεκριμένα:

- Το Random tester module. Είναι ο απλούστερος οδηγός και χρησιμοποιείται για τον έλεγχο αλλαγών/προσθηκών στην Ruby χρησιμοποιώντας ακραίες περιπτώσεις του συστήματος μνήμης.

- Το Micro-benchmark module που παρέχει διάφορα micro-benchmarks και χρησιμοποιείται για επαλήθευση του μοντέλου χρονισμού και την ανάλυση κάποιον ειδικών συνθηκών.

- Ο Simics. Αυτό είναι το κομμάτι που χρησιμοποιήθηκε σε αυτή την διπλωματική εργασία. Ο οδηγός για υποστήριξη του Simics εξομοιώνει ένα απλό in-order επεξεργαστή χωρίς καθυστερήσεις που οφείλονται στο pipeline. Όλες οι load, store και instruction fetch αιτήσεις από τον Simics περνούν στη Ruby, η οποία εκτελεί προσβάσεις στην κρυφή μνήμη πρώτου επιπέδου και καθορίζει, αν το αίτημα ευστοχεί ή αστοχεί στην κρυφή μνήμη. Στην περίπτωση ευστοχίας ο Simics συνεχίζει την εκτέλεση εντολών, εναλλάσοντας μεταξύ των διαφόρων επεξεργαστών σε ένα πολυεπεξεργαστικό περιβάλλον. Σε μια αστοχία η Ruby αναλαμβάνει τις καθυστερήσεις των αιτημάτων του Simics από τον αντίστοιχο Simics επεξεργαστή και εξομοιώνει την αστοχία κρυφής μνήμης. Κάθε επεξεργαστής μπορεί να έχει μόνο μία εκκρεμή αστοχία ανά πάσα στιγμή στην εξομοίωση, αλλά ο συνωστισμός και άλλες παράμετροι που αφορούν τον χρονισμό θα καθορίσουν πότε θα ολοκληρωθεί το συγκεκριμένο αίτημα. Με το χρονικό έλεγχο του πότε ο Simics προχωρεί, η Ruby καθορίζει το χρονικά εξαρτώμενο κομμάτι της λειτουργικής εξομοίωσης του Simics.

- Ο Opal. Αυτός ο οδηγός μοντελοποιεί ένα δυναμικά δρομολογούμενο SPARC v9 επεξεργαστή σε αντίθεση με τον Simics ο οποίος μοντελοποιεί επεξεργαστές in-order εκτέλεσης. Παρ'όλα αυτά χρησιμοποιεί τον Simics για την επαλήθευση της λειτουργικότητάς του.
  Σε αυτή την διπλωματική εργασία όπως αναφέρθηκε πιο πριν η μελέτη έγινε με την βοήθεια του Simics και της Ruby (μετά από αρκετές αλλαγές για την υποστήριξη πολυνηματικών αρχιτεκτονικών και του thread scheduling-migration).

3.3.4 Η ιεραρχία μνήμης του GEMS (Ruby)

Γενικά

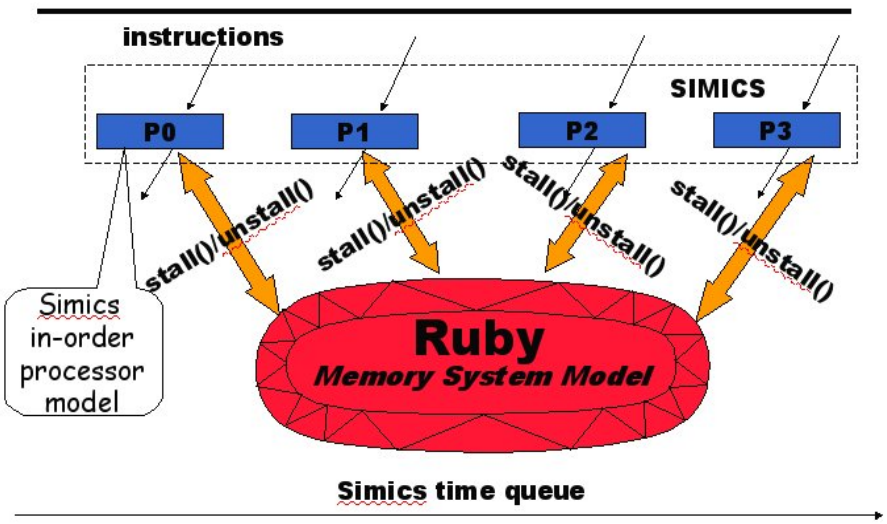
Η Ruby αποτελεί ένα εξομοιωτή χρονισμού ενός συστήματος μνήμης πολυεπεξεργαστών ο οποίος μοντελοποιεί τις κρυφές μνήμες, τους ελεγκτές κρυφής μνήμης, το δίκτυο διασύνδεσης του συστήματος, τους ελεγκτές μνήμης και τα τμήματα της κύριας μνήμης. Η Ruby συνδυάζει hardcoded χρονική εξομοίωση για συσκευές που είναι ανεξάρτητες από το πρωτόκολλο συνάφειας κρυφής μνήμης σε μεγάλο βαθμό με την ικανότητα να καθορίζει και τις εξαρτημένες από το πρωτόκολλο (π.χ. οι ελεγκτές κρυφής μνήμης) σε μια domain-specific γλώσσα που λέγεται SLICC (Specification Language for Implementing Cache Coherence).

Υλοποίηση της Ruby

Η Ruby είναι υλοποιημένη στην αντικειμενοστρεφή γλώσσα προγραμματισμού C++ και χρησιμοποιεί ένα queue-driven μοντέλο χρονισμού για να εξομοιώσει τον χρονισμό. Τα διάφορα στοιχεία του συστήματος επικοινωνούν με buffers μηνυμάτων ποικίλης καθυστέρησης και εύρους ζώνης και το στοιχείο που βρίσκεται στο άκρο λήψης του buffer είναι προγραμματισμένο να ξυπνήσει όταν το επόμενο μήνυμα θα είναι διαθέσιμο προς ανάγνωση στον buffer. Παρόλο που κάποιοι buffers είναι σχεδιασμένοι αυστηρά με first-in-first-out (FIFO) λογική δεν περιορίζονται μόνο σε αυτή την FIFO συμπεριφορά. Η εξομοίωση προχωρά με την κλήση της wakeup μεθόδου για το επόμενο προγραμματιζόμενο γεγονός στην ουρά γεγονότων (event queue). Σημασιολογικά η εξομοίωση θα ήταν η ίδια αν όλα τα στοιχεία του συστήματος ξυπνούσαν σε κάθε κύκλο όμως η προσθήκη της ουράς γεγονότων μπορεί να θεωρηθεί σαν μια βελτιστοποίηση στο να αποφύγουμε αχρείαστη επεξεργασία σε κάθε κύκλο.

3.4 Υλοποίηση CMT simulator με υποστήριξη μετανάστευσης νημάτων

3.4.1 Γενικά



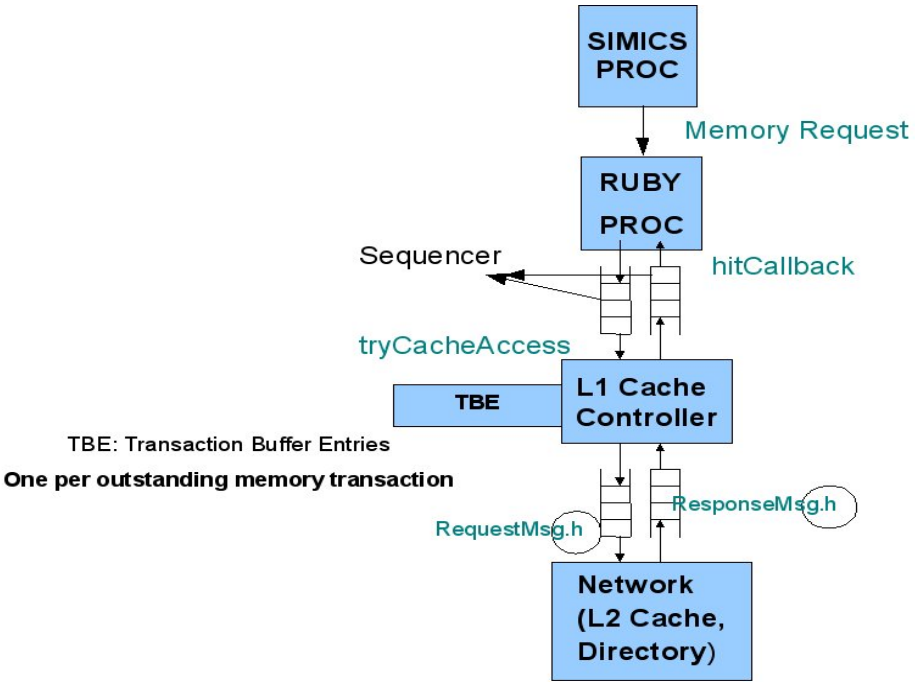
Σχήμα 3.3: Επικοινωνία-λειτουργία των Simics-Ruby.

Όπως αναφέρθηκε προηγουμένως ο Simics Driver είναι ο οδηγός ο οποίος περνά όλες τις αιτήσεις στη μνήμη από τον Simics στη Ruby. Έπειτα η Ruby αναλαμβάνει τις προσβάσεις στην κρυφή μνήμη L1 και μετά στα ψηλότερα επίπεδα μνήμης. Αξίζει να σημειωθεί ότι καθ' όλη την διάρκεια της

εξομοίωσης μόνο μία εκκρεμής αίτηση μπορεί να υπάρχει από κάθε επεξεργαστή Simics.

Όπως βλέπουμε στο Σχήμα 3.3 ο κάθε Simics επεξεργαστής για τον οποίο εκκρεμεί ένα αίτημα απενεργοποιείται μέσω της Ruby. Στην περίπτωση που μια πρόσβαση μνήμης για κάποιον επεξεργαστή έχει εξομοιωθεί στη Ruby τότε ο επεξεργαστής για τον οποίο εκτελέστηκε η συγκεκριμένη πρόσβαση επανενεργοποιείται και συνεχίζει.

Η πορεία ενός αιτήματος από τον Simics στην Ruby



Σχήμα 3.4: Η πορεία ενός αιτήματος από τον Simics στην Ruby.

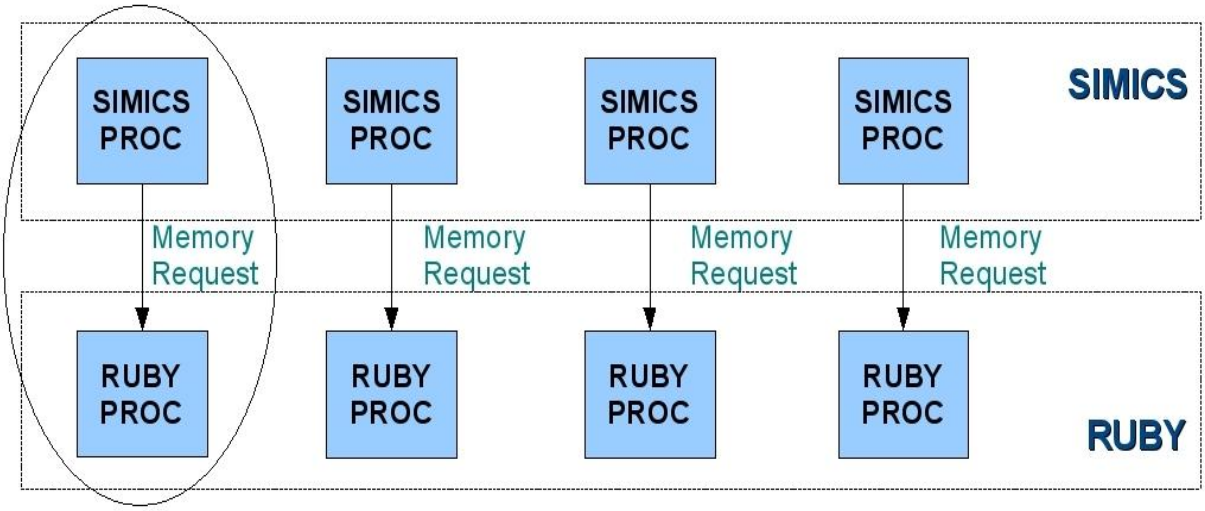
Στο Σχήμα 3.4 παρουσιάζεται διαγραμματικά η πορεία ενός αιτήματος μνήμης από τον επεξεργαστή Simics στον αντίστοιχο επεξεργαστή Ruby. Η αρχική παρατήρηση στην οποία μπορούμε να προβούμε είναι η αντιστοιχία ένα προς ένα μεταξύ Simics και Ruby επεξεργαστών.Η Ruby και γενικότερα ο GEMS προορίζεται για εξομοίωση πολυεπεξεργαστών χωρίς υποστήριξη πολλαπλών αιτημάτων από διάφορους Simics επεξεργαστές σε ένα Ruby επεξεργαστή δηλαδή χωρίς κοινή χρήση πόρων μεταξύ των Simics επεξεργαστών - νημάτων όπως προϋποθέτει ο πολυνηματισμός. Οι αιτήσεις από κάθε επεξεργαστή Simics περνούν στον αντίστοιχο Ruby. Όταν το αίτημα απαιτεί την εξομοίωση της Ruby τότε ο αντίστοιχος επεξεργαστής Simics απενεργοποιείται μέχρι την εξομοίωση του αιτήματος στην Ruby. Ο κάθε επεξεργαστής Ruby περιλαμβάνει μια δομή Sequencer, ο οποίος δεν είναι τίποτα άλλο από μία ουρά στην οποία εισέρχονται τελικά τα αιτήματα που προορίζονται για τον συγκεκριμένο Ruby επεξεργαστή. Κάθε καινούριο αίτημα στον Sequencer είναι σε κατάσταση Serving.

Ο Sequencer ακολούθως προωθεί το αίτημα στην ουρά του ελεγκτή κρυφής μνήμης L1. Αντίστοιχα αν το αίτημα δεν ικανοποιηθεί στην κρυφή μνήμη L1 τότε ο ελεγκτής κρυφής μνήμης L1 καλεί μέσω του πρωτοκόλλου συνάφειας τον αντίστοιχο ελεγκτή κρυφής μνήμης L2 για ικανοποίηση του αιτήματος από την κρυφή μνήμη L2. Αφού εξομοιωθεί η ευστοχία ή αστοχία, ένα μήνυμα που αφορά στην ικανοποίηση του αιτήματος καταλήγει πίσω στον Sequencer μέσω του ελεγκτή κρυφής μνήμης L1. Το αίτημά μας έχει εξομοιωθεί και είναι πλέον σε κατάσταση Done.

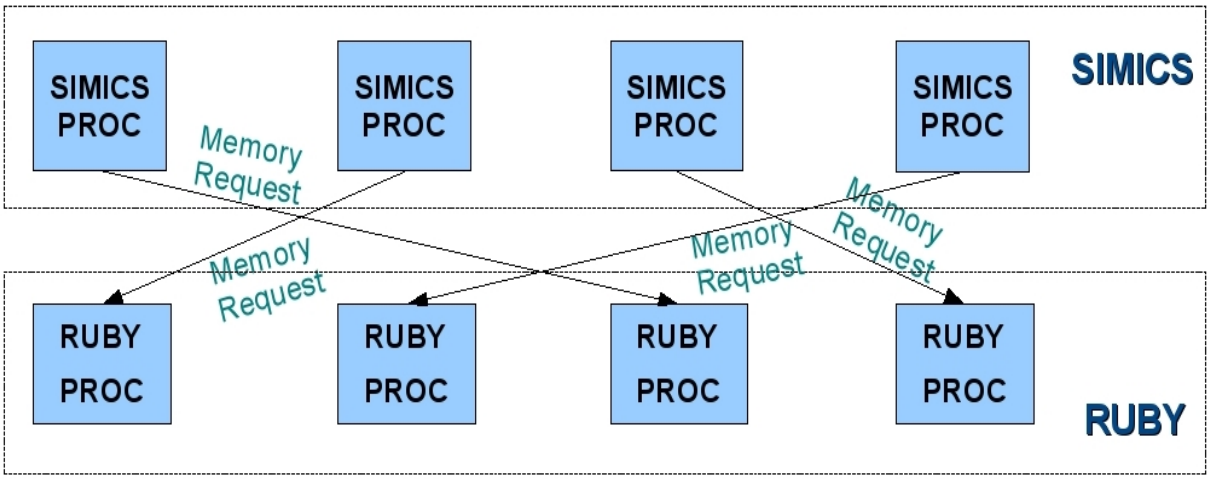
Ακολούθως ο Simics επεξεργαστής για τον οποίο το αίτημα έχει εξομοιωθεί ενεργοποιείται. Στο σημείο αυτό έχουμε επανέκδοση του ίδιου αιτήματος το οποίο εξομοιώθηκε πριν για σκοπούς ορθής λειτουργίας. Με το πέρασμα του ίδιου αιτήματος στον αντίστοιχο Ruby εξομοιωτή το οποίο όμως τώρα βρίσκεται σε κατάσταση Done,η Ruby επιστρέφει στον Simics μέσω του μοντέλου χρονισμού μηδενικό αριθμό κύκλων καθυστέρησης, λέγοντας ουσιαστικά στον Simics να προχωρήσει στο επόμενο αίτημα. Η όλη διαδικασία συνεχίζεται μέχρι το πέρας της εξομοίωσης.

3.4.2 Υποστήριξη Μετανάστευσης νημάτων

Η πρώτη υλοποίηση στην οποία προβήκαμε ήταν η μετανάστευση νήματος (σε πρώτο στάδιο) από ένα επεξεργαστή Ruby σε ένα άλλο.



Σχήμα 3.5: Κανονική λειτουργία Simics-Ruby σε ένα σύστημα 4 επεξεργαστών.



Σχήμα 3.6: Λειτουργία Simics-Ruby με υποστήριξη μετανάστευσης νημάτων σε ένα σύστημα 4 επεξεργαστών.

Στο Σχήμα 3.5 παρατηρούμε την προκαθορισμένη συμπεριφορά Simics και Ruby στην οποία κάθε

Simics επεξεργαστής αντιστοιχεί στο Ruby επεξεργαστή με το ίδιο id, δηλαδή ο Simics 0 περνά τα αιτήματά του στο Ruby 0, ο Simics 1 στο Ruby 1 κτλ. Στο Σχήμα 3.6 μετά από τις κατάλληλες τροποποιήσεις στις οποίες προβήκαμε στον πηγαίο κώδικα της Ruby βλέπουμε ότι η καινούρια της διαμόρφωση μπορεί να υποστηρίξει το πέρασμα αιτημάτων από οποιονδήποτε επεξεργαστή Simics σε οποιονδήποτε επεξεργαστή Ruby. Ουσιαστικά αυτό επιτυγχάνεται με την χρήση ενός ενδιάμεσου διανύσματος ακεραίων map_proc στο οποίο κάθε θέση του αντιστοιχεί στον συγκεκριμένο Simics επεξεργαστή με id ίσο με το δείκτη της θέσης. Η τιμή του διανύσματος τέλος σε κάθε θέση ισούται με το id του συγκεκριμένου Ruby επεξεργαστή στον οποίο περνά τα αιτήματα ο Simics επεξεργαστής που αντιστοιχεί σε εκείνη την θέση. Έτσι για την προκαθορισμένη συμπεριφορά της Ruby σε ένα σύστημα τετραπύρηνο σύστημα έχουμε ένα διάνυσμα map_proc με τιμές [0,1,2,3]. Ενώ π.χ. στην αντίστοιχη περίπτωση όπου ο Simics 0 περνά τα αιτήματά του στον Ruby 0, ο Simics 1 στον Ruby 3, ο Simics 2 στον Ruby 1 και ο Simics 3 στον Ruby 2 τότε η τιμή του map_proc θα είναι [0,3,1,2]. Η τιμή του διανύσματος μπορεί να αλλάζει δυναμικά υλοποιώντας έτσι την μετανάστευση νημάτων σε διαφορετικό επεξεργαστή. Η διαμόρφωση της Ruby στην οποία προβήκαμε μπορεί να υποστηρίξει μεγαλύτερο αριθμό επεξεργαστών. Το πιο πάνω παράδειγμα για τέσσερις είναι ενδεικτικό.

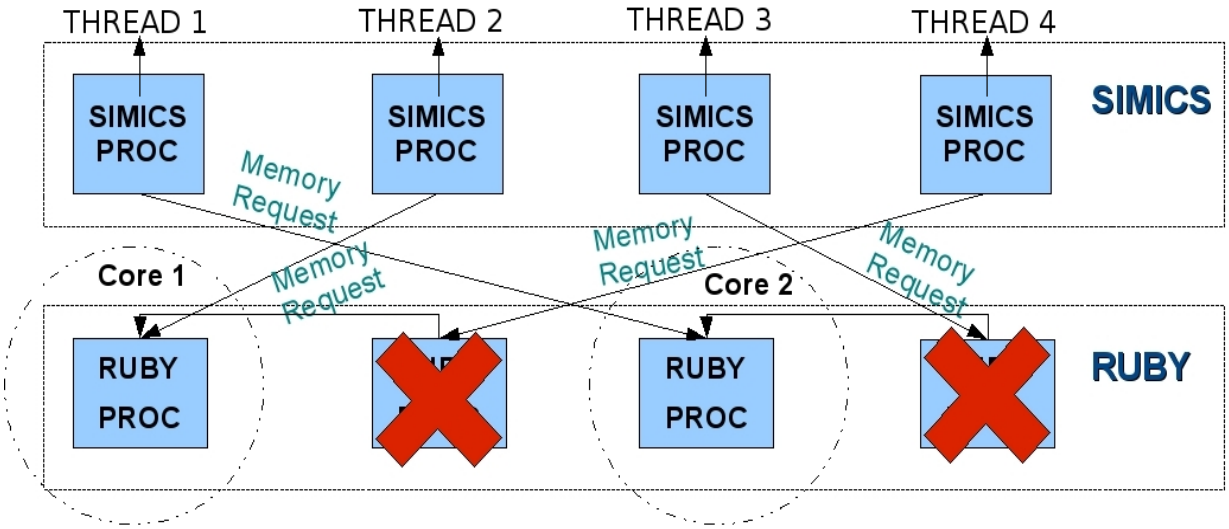
Εδώ πρέπει να προσθέσουμε ότι έπρεπε να γίνουν αλλαγές και στο σύστημα στατιστικών για κάθε επεξεργαστή Ruby επειδή η αντιστοιχία Simics Proc ID = Ruby Proc ID παύει να υπάρχει. Στην default έκδοση της Ruby πολλές πληροφορίες ήταν δυνατό να ληφθούν από τον αντίστοιχο Simics επεξεργαστή πράγμα που στην περίπτωση της μετανάστευσης νημάτων και μάλιστα η οποία θα γίνεται δυναμικά κατά περιόδους δεν ισχύει, εφόσον σε κάθε περίοδο ο Ruby επεξεργαστής μπορεί να αντιστοιχεί σε διαφορετικό Simics επεξεργαστή.

3.5 Η υλοποίηση των πολυνηματικών πυρήνων στην Ruby

Με την πιο πάνω μέθοδο επιτυγχάνεται η μετανάστευση νημάτων στους διάφορους επεξεργαστές. Όμως σε ένα CMT επεξεργαστή πρέπει σε κάθε πυρήνα να έχουμε κοινή χρήση πόρων από περισσότερα του ενός νήματα. Το επόμενο βήμα ήταν η επιπλέον διαμόρφωση της Ruby ώστε να έχουμε κοινή χρήση ενός επεξεργαστή Ruby από περισσότερους του ενός επεξεργαστές Simics ούτως ώστε να γίνεται κοινή χρήση της μνήμης από αυτούς και ουσιαστικά να έχουμε πλέον Simics επεξεργαστές που να συμπεριφέρονται σαν νήματα και Ruby επεξεργαστές που να συμπεριφέρονται σαν SMT πυρήνες. Ο σχεδιασμός που κάναμε εμφανίζεται στο Σχήμα 3.7.

Όπως φαίνεται στο Σχήμα 3.7 δημιουργούμε ομάδες-πυρήνες Ruby επεξεργαστών που υπεύθυνος για την εκτέλεση των αιτημάτων από τους Simics επεξεργαστές - νήματα κάθε ομάδας είναι ο πρώτος Ruby επεξεργαστής. Στο παράδειγμά μας έτσι έχουμε δύο πυρήνες οι οποίοι δέχονται αιτήματα από δύο Simics επεξεργαστες-νήματα. Ο ένας πυρήνας εκτελεί αιτήματα που αφορούν τους Ruby επεξεργαστές 0 και 1 και ο δεύτερος τους 2 και 3. Για να επιτευχθεί αυτό τα αιτήματα των επεξεργαστών Simics που αφορούν τους Ruby 1 και 3 (βάση του διανύσματος map_proc) περνούν αντίστοιχα στους Ruby επεξεργαστές 0 και 1. Έτσι οι 0 και 1 αλλά και οι 2 και 3 μοιράζονται πλέον Sequencer, κρυφή μνήμη L1,ελεγκτή κρυφής μνήμης L1 και ελεγκτή κρυφής μνήμης L2, άρα πλέον πρόκειται για ένα διπύρηνο σύστημα πολυνηματικών πυρήνων με δύο νήματα ανά πυρήνα που μάλιστα υποστηρίζει μετανάστευση νημάτων. Η L2 παραμένει κοινή για όλους τους πυρήνες στο chip, όπως προνοεί η σχεδίαση των CMTs.

Αξίζει να σημειωθεί ότι η διαμόρφωση στην οποία προβήκαμε υποστηρίζει οποιοδήποτε αριθμό πυρήνων και νημάτων μέσω παραμέτρων που εισάγει ο χρήστης είτε απευθείας στην κονσόλα του Simics, είτε μέσω Simics script. Για την υποστήριξη πολλαπλών νημάτων σε ένα Ruby επεξεργαστή, χρειάστηκε πρώτα ανάλογος πολλαπλασιασμός δομών δεδομένων ανάλογα πάντα με τις παραμέτρους συστήματος που εισάγει ο χρήστης και που αναφέρθηκαν προηγουμένως. Στη συνέχεια χρειάστηκε διαμόρφωση κάποιων υφιστάμενων μεθόδων όπως η hitCallback που είναι η μέθοδος σε κάθε Ruby



Σχήμα 3.7: Λειτουργία Simics-Ruby με υποστήριξη μετανάστευσης νημάτων και SMT σε ένα σύστημα 4 επεξεργαστών.

επεξεργαστή μέσω της οποία καταλήγει ένα μήνυμα ικανοποίησης αιτήματος από τις κρυφές μνήμες όπως επίσης και διαμόρφωση του συστήματος καταγραφής στατιστικών. Η διαμόρφωση του συστήματος καταγραφής στατιστικών επιβάλλεται αφού πλέον πρέπει να κρατάμε στατιστικά σε κάθε κύριο Ruby επεξεργαστή του πυρήνα για τους Simics επεξεργαστές - νήματα που αντιπροσωπεύει. Επίσης έπρεπε να δημιουργήσουμε δομές δεδομένων για την εξακρίβωση της ταυτότητας του κάθε νήματος στους κύριους Ruby επεξεργαστές για την ορθή καταγραφή των στατιστικών τους π.χ. των αστοχιών τους και των εντολών που έχουν συμπληρώσει.

3.6 Στατιστικές μέθοδοι SimPoints

3.6.1 Γενικά

Τα μοντέρνα πακέτα μέτρησης επιδόσεων (π.χ. SPEC CPU2006) χρειάζονται συνολικά μήνες για να εξομοιωθούν. Οι ερευνητές και οι επαγγελματίες χρησιμοποιούν έτσι τεχνικές μερικής εξομοίωσης για την μείωση του απαιτούμενου χρόνου, ελπίζοντας να μην θυσιάσουν την ακρίβεια. Το SimPoint είναι μια δημοφιλής μέθοδος επιλογής των αντιπροσωπευτικών μερών που προσεγγίζουν την συνολική συμπεριφορά μιας εφαρμογής. Η προσέγγιση χωρίζει μια εφαρμογή σε διαστήματα, παράγει ένα Basic Block Vector (BBV) που αντιπροσωπεύει τις οδηγίες που εκτελούνται σε κάθε διάστημα, ομαδοποιεί τα BBVs σύμφωνα με την ομοιότητα και επιλέγει ένα αντιπροσωπευτικό διάστημα από τις σημαντικότερες ομάδες. Δυστυχώς αποτελεσματικά εργαλεία για την παραγωγή BBVs είναι μέχρι τώρα ανύπαρκτα για πολλές αρχιτεκτονικές, ειδικά για αυτές των ενσωματωμένων συστημάτων.[33].

Σ' αυτή την διπλωματική εργασία χρησιμοποιήσαμε την αρχιτεκτονική SPARC που υποστηρίζει ο GEMS. Έγινε χρήση του πακέτου μέτρησης επιδόσεων SPEC2000 στο οποίο για καθένα από τα χρησιμοποιούμενα μετροπρογράμματα (τα οποία μεταγλωττίστηκαν σε SPARC αρχιτεκτονική) έγινε εξαγωγή ενός BBV, του αντιπροσωπευτικότερου, ούτως ώστε να πάρουμε τα καλύτερα δυνατά αποτελέσματα σε περιορισμένο χρόνο. Στη συνέχεια θα εξηγήσουμε τι είναι ακριβώς το BBV και θα παρουσιάσουμε τα εργαλεία που χρησιμοποιήθηκαν.

3.6.2 BBV - Χρήση του qemu_bbv

BBV

Ένα βασικό block είναι ένα τμήμα του κώδικα που εκτελείται από την αρχή μέχρι το τέλος με μια είσοδο και μια έξοδο σε αυτό το τμήμα[24]. Χρησιμοποιούμε τις συχνότητες με τις οποίες τα βασικά block εκτελούνται σαν μέτρο σύγκρισης των διαφορετικών τμημάτων της εκτέλεσης της εφαρμογής. Το σημαντικό σε αυτό είναι ότι η συμπεριφορά του προγράμματος σε μια δεδομένη στιγμή συσχετίζεται άμεσα με τον κώδικα που εκτελεί εκείνη τη στιγμή, και τα βασικά blocks παρέχουν αυτές τις πληροφορίες. Ένα πρόγραμμα αφού τρέξει για οποιοδήποτε χρονικό διάστημα θα εκτελέσει κάθε βασικό block στο πρόγραμμα για κάποιο αριθμό φορών. Η γνώση αυτής της συχνότητας για κάθε βασικό block παρέχει την πληροφορία σε ποια σημεία - βασικά blocks κώδικα η εφαρμογή ξοδεύει το χρόνο της. Η βασική ιδέα είναι να βρεθεί ένα λογικό σε μέγεθος χρονικό διάστημα στην εκτέλεση του προγράμματος που έχει μοτίβο προσβάσεων στα βασικά blocks παρόμοιο με αυτό στην πλήρη εκτέλεση του προγράμματος. Εάν μπορούμε να το βρούμε αυτό, ξέρουμε ότι και η πλήρης εκτέλεση του προγράμματος και το διάστημα που επιλέξαμε ξοδεύουν αναλογικά τον ίδιο χρόνο για τον ίδιο κώδικα. Το BBV είναι ένας μονοδιάστατος πίνακας που κάθε στοιχείο του αντιστοιχεί σε κάθε βασικό block στο πρόγραμμα. Η τιμή κάθε στοιχείου στον πιο πάνω πίνακα αποτελεί τον αριθμό φορών που εισήλθε το πρόγραμμά μας στο αντίστοιχο βασικό block για κάποιο διάστημα εκτέλεσης.

Χρήση του qemu_bbv

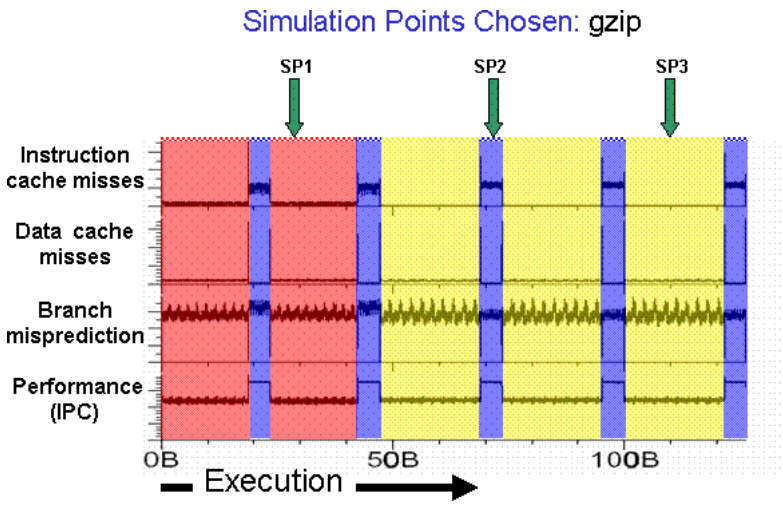
Το qemu_bbv [33] αποτελεί το μοναδικό εργαλείο το οποίο που μπορεί να εξάγει ένα αρχείο BBV για binaries αρχιτεκτονικής SPARC για την μετέπειτα επεξεργασία του από το SimPoints[25]. Είναι μια προσθήκη στον προσομοιωτή επεξεργαστών Qemu η οποία επιτρέπει εξαγωγή αρχείων BBV σε αρκετές αρχιτεκτονικές, περιλαμβανομένης και αυτής που μας ενδιέφερε(SPARC). Ο Qemu επίσης αποτελεί ένα φορητό, δυναμικό translator. Χρησιμοποιείται ευρέως για το τρέξιμο λειτουργικών συστημάτων μέσω hardware emulation, περιλαμβάνοντας όμως ένα Linux user-space emulator αποκτά την δυνατότητα να τρέχει Linux-binaries από μόνα τους με την χρήση του system-call translation. Οι υποστηριζόμενες αρχιτεκτονικές περιλαμβάνουν τις Alpha, SPARC, PowerPC, sh4, IA32, AMD64, MIPS, m68k και ARM.

3.6.3 Η μεθοδολογία του SimPoint

Ο αυξημένος χρόνος πλήρους εκτέλεσης των μετροπρογραμμάτων, ως αποτέλεσμα των μεγαλύτερων ιχνών δεδομένων και των αυξημένων δυναμικών πληθών εντολών. καθιστούν ασύμφορη την λεπτομερή εξομοίωση τους σε ακριβή μοντέλα (εξομοίωσης). Προκειμένου να επιλυθεί αυτό, οι αρχιτέκτονες επεξεργαστών επιλέγουν μόνο ένα μικρό τμήμα της εκτέλεσης κάθε μετροπρογράμματος για την πραγματοποίηση των εξομοιώσεων. Για τον λόγο αυτό έχουν προταθεί στατιστικές μέθοδοι οι οποίες χρησιμοποιώντας ένα μικρό κομμάτι κάθε εφαρμογής, επιτρέπουν την εξαγωγή ασφαλών συμπερασμάτων για την συνολική συμπεριφορά τους εισάγοντας μόνο ένα μικρό σφάλμα. Ωστόσο και η επιλογή αντιπροσωπευτικών τμημάτων των εφαρμογών είναι δύσκολη, διότι η συμπεριφορά του προγράμματος μεταβάλλεται σε συνάρτηση με τον χρόνο.

Η μεθοδολογία του SimPoint [25] προτείνει ένα τρόπο αναγνώρισης επαναλαμβανόμενων μοτίβων στην εκτέλεση των εφαρμογών και επιλογής των αντιπροσωπευτικών τμημάτων αυτών. Ο αλγόριθμος του SimPoint διαιρεί την εκτέλεση του προγράμματος σε διαστήματα με ίσο πλήθος εντολών και τα κατηγοριοποιεί ως προς την συμπεριφορά που παρουσιάζουν, βάσει της ταχύτητας με την οποία εκτελούνται. Μία *φάση* (phase) της εκτέλεσης είναι μία συλλογή από διαστήματα με παρόμοια συμπεριφορά. Από κάθε φάση επιλέγεται ένα αντιπροσωπευτικό τμήμα το οποίο ονομάζεται *σημείο εξομοίωσης* (simulation point). Στο Σχήμα 3.8 παρουσιάζεται η πλήρης εκτέλεση του gzip από το πακέτο μετροπρογραμμάτων SPEC2000 [13] και οι τρεις φάσεις εκτέλεσης που προσδιόρισε ο αλγόριθμος ταξινόμησης φάσεων του SimPoint. Το τελευταίο βήμα του αλγόριθμου είναι να επιλέγει

από κάθε φάση εκτέλεσης το αντιπροσωπευτικότερο διάστημα και το οποίο θα θεωρηθεί ως σημείο εξομοίωσης για τη φάση. Στο Σχήμα 3.8 τα SP1, SP2 και SP3 αντιπροσωπεύουν τα τρία σημεία που επιλέγονται.



Σχήμα 3.8: Η επιλογή σημείων εξομοίωσης στην περίπτωση του μετροπρογράμματος gzip

Για την ανάλυση των εξαχθέντων BBV αρχείων χρησιμοποιήθηκε το εργαλείο SimPoint [12] που αναπτύχθηκε στο Πανεπιστήμιο της California στο San Diego.

Κεφάλαιο 4

Μετροπρογράμματα

Benchmarks ή μετροπρογράμματα ονομάζονται τα προγράμματα που χρησιμοποιούνται για την μέτρηση της απόδοσης στα υπολογιστικά συστήματα. Η καλύτερη επιλογή για μετροπρογράμματα είναι συνήθως οι πραγματικές εφαρμογές, όπως ένας compiler. Προσπάθειες για την χρήση προγραμμάτων αρκετά απλούστερων από μια πραγματική εφαρμογή έχουν οδηγήσει σε παγίδες με χαρακτηριστικά παραδείγματα τα εξής:

- *kernels* που αποτελούν μικρά και σημαντικά κομμάτια πραγματικών εφαρμογών.

- *toy programs* που είναι προγράμματα μήκους 100 γραμμών κώδικα από απλές προγραμματιστικές ασκήσεις όπως η μέθοδος ταξινόμησης quicksort.

- *synthetic benchmarks* τα οποία είναι <<ψεύτικα>> προγράμματα δημιουργημένα για να συμπεριφέρονται όπως τις πραγματικές εφαρμογές π.χ. το Dhrystone.

Αντίθετα σήμερα χρησιμοποιούνται πακέτα μετροπρογραμμάτων (benchmark suites) τα οποία χρησιμοποιούν μια ποικιλία εφαρμογών με σκοπό την μέτρηση της απόδοσης ενός επεξεργαστή. Το πλεονέκτημα αυτών των πακέτων είναι ότι με τον συνδυασμό όλων των μετροπρογραμμάτων σε ένα πακέτο τελικά καταφέρνουμε να μελετήσουμε όλες τις πτυχές της απόδοσης ενός επεξεργαστή. Μια από τις πιο πετυχημένες προσπάθειες για τη δημιουργία ενός τέτοιου πακέτου ήταν η SPEC (Standard Performance Evaluation Corporation). Τα πακέτα αυτά εξελίσσονταν σε σχέση πάντα με την εξέλιξη της βιομηχανίας των υπολογιστών και έτσι σήμερα έχουμε SPEC benchmarks για την κάλυψη όλων των ομάδων εφαρμογών.

Στη διπλωματική εργασία αυτή, έγινε χρήση του πακέτου μετροπρογραμμάτων SPEC CPU2000. Το πακέτο SPEC CPU2000 αποτελεί ένα δημοφιλές πακέτο μετροπρογραμμάτων δημιουργημένο από την SPEC (Standards Performance Evaluation Corporation Evaluation) και χρησιμοποιείται εκτενώς τόσο σε ερευνητικό επίπεδο όσο και στην βιομηχανία υπολογιστών για την αξιολόγηση των επεξεργαστικών αρχιτεκτονικών. Ο έλεγχος της απόδοσης των υπολογιστικών συστημάτων με το προαναφερθέν πακέτο γίνεται με την εκτέλεση έντονων επεξεργαστικών υπολογισμών. Επιπρόσθετα, τα αποτελέσματα εξαρτώνται μόνο από τον επεξεργαστή και το μεγαλύτερο κομμάτι των λειτουργιών εκτελείται στην RAM. Έτσι συσκευές όπως η κάρτα γραφικών,ο σκληρός δίσκος ή οποιοδήποτε οπτικό μέσο δεν συμμετέχουν στην διαμόρφωση της επίδοσης.

Όλα τα μετροπρογράμματα του πακέτου χωρίζονται σε δύο ομάδες. Η πρώτη ομάδα αποτελείται από το πακέτο CINT2000 το οποίο περιλαμβάνει δώδεκα μετροπρογράμματα που κάνουν χρήση δεδομένων ακέραιων αριθμών (και λογικών τελεστών), έντεκα από τα οποία είναι γραμμένα σε γλώσσα προγραμματισμού C και ένα σε C++. Η δεύτερη ομάδα αποτελείται από το πακέτο CFP2000,το οποίο περιέχει δεκατέσσερα μετροπρογράμματα που περιλαμβάνουν έντονες λειτουργίες αριθμητικής κινητής υποδιαστολής(έξι γραμμένα σε Fortran-77, τέσσερα σε Fortran-90 και τέσσερα σε C).

4.1 Σύντομη Περιγραφή μετροπρογραμμάτων

Σε αυτή την ενότητα θα περιγράψουμε τα μετροπρογράμματα του πακέτου SPEC CPU2000 που χρησιμοποιήθηκαν στις διάφορες εξομοιώσεις. Η περιγραφή των μετροπρογραμμάτων χωρίζεται σε δύο υποενότητες, όπως αυτά χωρίζονται στο πακέτο SPEC CPU2000, δηλαδή στην ομάδα των μετροπρογραμμάτων ακέραιας αριθμητικής και στην ομάδα των μετροπρογραμμάτων αριθμητικής κινητής υποδιαστολής.

4.1.1 SPEC CINT2000

256.bzip2

Το μετροπρόγραμμα 256.bzip2 είναι ένα δημοφιλές πρόγραμμα συμπίεσης δεδομένων. Η έκδοση που χρησιμοποιείται στο πακέτο είναι έκδοση 0.1 γραμμένη από τον Julian Seward. Η μόνη διαφορά μεταξύ του bzip2 0.1 και του 256.bzip2 είναι ότι η έκδοση που χρησιμοποιείται στο πακέτο SPEC CPU2000 δεν εκτελεί οποιαδήποτε λειτουργία εισόδου/εξόδου, εκτός από το να διαβάζει την είσοδο Όλη η συμπίεση και αποσυμπίεση των δεδομένων γίνεται εξ' ολοκλήρου στην μνήμη. Με αυτό τον τρόπο όλη η λειτουργία γίνεται στην κεντρική μονάδα επεξεργασίας και στο υποσύστημα μνήμης.

254.gap

Το μετροπρόγραμμα 254.gap αποτελείται από ένα διερμηνέα (interpreter) που υλοποιεί μια γλώσσα προγραμματισμού και από μια βιβλιοθήκη σχεδιασμένα κυρίως για τις υπολογιστικές πράξεις σε ομάδες (computing in groups). Το GAP είναι συντομογραφία για το Groups,Algorithms and Programming.

164.gzip

Αξίζει να σημειωθεί ότι το μετροπρόγραμμα gzip (GNU zip) είναι ένα δημοφιλές πρόγραμμα συμπίεσης δεδομένων γραμμένο από τον Jean-Loup Gailly <gzip@gnu.org> για το GNU project. Το gzip χρησιμοποιεί την κωδικοποίηση Lempel-Ziv ως αλγόριθμο συμπίεσης. Η SPEC έκδοση του gzip δεν εκτελεί λειτουργίες εισόδου/εξόδου εκτός από την την ανάγνωση της εισόδου. Επίσης όλη η συμπίεση και αποσυμπίεση των δεδομένων γίνεται εξ' ολοκλήρου στην μνήμη. Με αυτό τον τρόπο όλη η λειτουργία γίνεται στην κεντρική μονάδα επεξεργασίας και στο υποσύστημα μνήμης.

181.mcf

Το 181.mcf είναι εμπνευσμένο από το πρόγραμμα MCF που χρησιμοποιείται για την επίλυση του προβλήματος δρομολόγησης οχημάτων με την εύρεση διαδρομών με το μικρότερο χρονικό κόστος. Το μετροπρόγραμμα είναι γραμμένο σε C και χρησιμοποιεί αποκλειστικά ακέραια αριθμητική. Ο αλγόριθμος δικτύων simplex που χρησιμοποιείται στο μετροπρόγραμμα είναι μια ειδική έκδοση του γνωστού αλγόριθμου simplex που χρησιμοποιείται στα προβλήματα ελάχιστης ροής. Είναι άξιο λόγου, ότι ο χρόνος χειρότερης εκτέλεσης για τον εν λόγω αλγόριθμο είναι ψευδοπολυωνυμικός και άρα το πρόβλημα είναι weakly NP-complete.

197.parser

Το 197.parser αποτελεί υλοποίηση του Link Grammar Parser που είναι ένας συντακτικός αναλυτής της αγγλικής γλώσσας. Βασίζεται στην link grammar που αποτελεί θεωρία του συντακτικού της αγγλικής γλώσσας. Δοθείσας μιας πρότασης, το σύστημα καθορίζει μια συντακτική δομή γι' αυτήν που περιλαμβάνει ομάδες αποτελούμενες από συνδέσμους που συνδέουν ζευγάρια λέξεων. Ο parser περιλαμβάνει ένα λεξικό από 60000 μορφές λέξεων. Η είσοδος που χρησιμοποιείται είναι προτάσεις, μία για κάθε γραμμή.

175.vpr

Το 175.vpr είναι πρόγραμμα που βοηθά στο σχεδιασμό ολοκληρωμένων κυκλωμάτων. Πιο συγκεκριμένα υλοποιεί τοποθετήσεις και δρομολογήσεις στα FPGAs. Με τον όρο τοποθέτηση εννοούμε την επιλογή του λογικού block και του I/O pad μέσα στο FPGA που θα υλοποιήσει την κάθε μια συνάρτηση του κυκλώματος. Σκοπός είναι η τοποθέτηση των στοιχείων που είναι συνδεδεμένα κοντά, για την μείωση των καλωδιώσεων στο κύκλωμα και την μεγιστοποίηση της ταχύτητας του κυκλώματος. Ο αλγόριθμος που χρησιμοποιείται για την τοποθέτηση είναι ο simulated annealing. Όσον αφορά στην δρομολόγηση, εννοούμε τον καθορισμό των προγραμματιζόμενων διακοπτών που πρέπει να τεθούν σε λειτουργία για να επιτευχθεί σύνδεση των καλωδίων στο FPGA που συνδέουν τις εισόδους και εξόδους του. Σκοπός είναι η δημιουργία όλων των συνδέσμων που απαιτούνται από το κύκλωμα και η μεγιστοποίηση της ταχύτητας του κυκλώματος. Ο αλγόριθμος που χρησιμοποιείται σε αυτή την περίπτωση είναι μια εξειδίκευση του αλγόριθμου του Djikstra.

4.1.2 SPEC CFP2000

179.art

Το μετροπρόγραμμα art χρησιμοποιείται για την αναγνώριση αντικειμένων σε μια θερμική εικόνα. Τα αντικείμενα που χρησιμοποιούνται στην συγκεκριμένη περίπτωση είναι ένα ελικόπτερο και ένα αεροπλάνο. Ουσιαστικά δημιουργείται ένα νευρωνικό δίκτυο με βάση τα δύο αυτά αντικείμενα. Αφού ολοκληρωθεί το εν λόγω δίκτυο τα αντικείμενα ανιχνεύονται στην θερμική εικόνα. Ένα παράθυρο με μέγεθος αντίστοιχο των αντικειμένων σαρώνει την θερμική εικόνα και αποτελεί την είσοδο για το δημιουργημένο νευρωνικό δίκτυο. Στην συνέχεια το νευρωνικό δίκτυο προσπαθεί να ταυτίσει την εικόνα του παραθύρου με μια από τις εικόνες βάσει των οποίων έχει δημιουργηθεί.

183.equake

Το εν λόγω πρόγραμμα μιμείται τη διάδοση των ελαστικών κυμάτων στις μεγάλες, ιδιαίτερα ετερογενείς κοιλάδες, όπως το San Fernando Valley της Καλιφόρνιας, ή την Greater Basin του Λος Άντζελες. Ο στόχος είναι να ανακτηθεί η χρονική ιστορία της κίνησης του εδάφους παντού μέσα στην κοιλάδα λόγω ενός συγκεκριμένου σεισμικού γεγονότος. Οι υπολογισμοί εκτελούνται σε ένα μη δομημένο πλέγμα που επιλύει τοπικά τα μήκη κύματος, χρησιμοποιώντας μια πεπερασμένη μέθοδο στοιχείων.

189.lucas

Το lucas εκτελεί τη δοκιμή Lucas-Lehmer για να ελέγξει το primality των Mersenne αριθμών (της μορφής δηλαδή 2^p-1), χρησιμοποιώντας την αριθμητική αυθαίρετης ακρίβειας (ακέραιων πινάκων). Πετυχαίνει τον τετραγωνισμό του Mersenne-mod μέσω της τεχνικής διακριτής μετατροπής Crandall και Fagin. Παράλληλα χρησιμοποιεί ένα φιλικό στην κρυφή μνήμη, γρήγορο μετασχηματισμό Fourier για να εκτελέσει αποτελεσματικά την τετραγωνοποίηση των επαναλήψεων Lucas-Lehmer που έχει ως αποτέλεσμα την δημιουργία μεγάλων ακεραίων.

177.mesa

Η mesa είναι μια βιβλιοθήκη γραφικών παρόμοια με την OpenGL. Υποστηρίζει ένα γενικό buffer πλαισίων ούτως ώστε να μην υπάρχει καμιά εξάρτηση σε σχέση με το λειτουργικό ή το σύστημα παραθύρων που χρησιμοποιείται Μπορούν να γραφτούν πάρα πολλά client προγράμματα με σκοπό να στρεσάρουν την απόδοση σε FP λειτουργίες,κλιμάκωση και μνήμη (ακόμα και συνδυασμό τους). Η έξοδος των αποτελεσμάτων της εκτέλεσης μπορεί να γραφτεί σε αρχεία εικόνων για επαλήθευση.

4.2 Μετρικές Απόδοσης στις Πολυνηματικές Αρχιτεκτονικές

Οι μετρικές τις κλασικής αρχιτεκτονικής ενός επεξεργαστή δεν μπορούν να εκφράσουν την απόδοση στις σύγχρονες πολυνηματικές αρχιτεκτονικές αφού στις δεύτερες εισάγονται θέματα όπως η δικαιοσύνη μεταξύ νημάτων. Πιο συγκεκριμένα οι μετρικές που χρησιμοποιούνται πλέον σε αυτές τις αρχιτεκτονικές είναι οι εξής:

- Το άθροισμα του IPC κάθε νήματος σε μια πολυνηματική αρχιτεκτονική. Πιο συγκεκριμένα στην περίπτωση που ο επεξεργαστής εμφανίζει n νήματα έχουμε:

$$IPC_{sum} = \sum_{k=1}^{n} IPC_k \text{ όπου } k \text{ ο δείκτης του κάθε νήματος.} \tag{4.1}$$

  Το πιο πάνω άθροισμα φανερώνει την συνολική απόδοση του συστήματος σε IPC (throughput) και μπορεί να μην είναι δίκαιος δείκτης για την βελτίωση στα νήματα χαμηλού IPC.

- Το Weighted Speedup ή WS το οποίο αποτελεί το άθροισμα των πηλίκων του IPC ενός νήματος σε μια περίοδο, όταν τρέχει παράλληλα με άλλα νήματα προς το IPC του όταν αυτό τρέχει μόνο του, δηλαδή το single threaded IPC του. Στην περίπτωση των n νημάτων έχουμε:

$$WS_{algorithm} = \sum_{k=1}^{n} \frac{IPC_k concurrent}{IPC_k single} \text{ όπου } k \text{ ο δείκτης του κάθε νήματος.} \tag{4.2}$$

  Το IPC_{single} δείχνει το μέγιστο IPC ή το θεωρητικό όριο. Άρα το πηλίκο δείχνει το πόσο αποδοτικά τρέχει σε κάθε περίπτωση το κάθε νήμα. Έτσι στο WS συμπεράσματα μπορούν να εξαχθούν για την μεμονωμένη βελτίωση του IPC για κάθε νήμα. Επιπλέον λόγω της ίσης αντιμετώπισης νημάτων αποτελεί πιο δίκαια μετρική για όλα τα νήματα και την μεμονωμένη απόδοση νήματος.

- Ο αρμονικός μέσων των κανονικοποιημένων IPCs των νημάτων. Δηλαδή για n νήματα έχουμε:

$$IPC_{harmonic\_mean} = \frac{n}{\sum_{k=1}^{n} \frac{IPC_k concurrent}{IPC_k single}} \text{ όπου } k \text{ ο δείκτης του κάθε νήματος.} \tag{4.3}$$

  Ο $\text{IPC}_{harmonic\_mean}$ συνδυάζει βελτίωση σε επίδοση συστήματος και δικαιοσύνη μεταξύ των νημάτων.

Κεφάλαιο 5

Υλοποιήσεις αλγόριθμων δρομολόγησης νημάτων - Πειραματικά Αποτελέσματα - Ανάλυση Αποτελεσμάτων

Στο κεφάλαιο αυτό θα μελετήσουμε συγκεκριμένες πολιτικές δρομολόγησης οι οποίες υλοποιήθηκαν στον CMT εξομοιωτή που δημιουργήσαμε. Πιο συγκεκριμένα υλοποιήθηκαν οι πολιτικές DCCS(Data Cache Conflict Scheduler) και IPCS (Instruction Per Cycle Scheduler) η οποίες παρουσιάζονται στις επόμενες υποενότητες. Αφού αναλυθούν πρώτα τα αποτελέσματα για τις δύο αυτές πολιτικές στην συνέχεια θα παρουσιάσουμε μια τρίτη η οποία ουσιαστικά αποτελεί μια διαφοροποίηση του DCCS και η οποία σε κάποιες περιπτώσεις παίρνει καλύτερες αποφάσεις δρομολόγησης. Στο τέλος αυτού του κεφαλαίου θα παρουσιαστούν τα πειραματικά αποτελέσματα αυτής της υλοποίησης.

5.1 Υλοποιήσεις αλγόριθμων δρομολόγησης νημάτων

5.1.1 $1^{η}$ υλοποίηση - Στατική δρομολόγηση

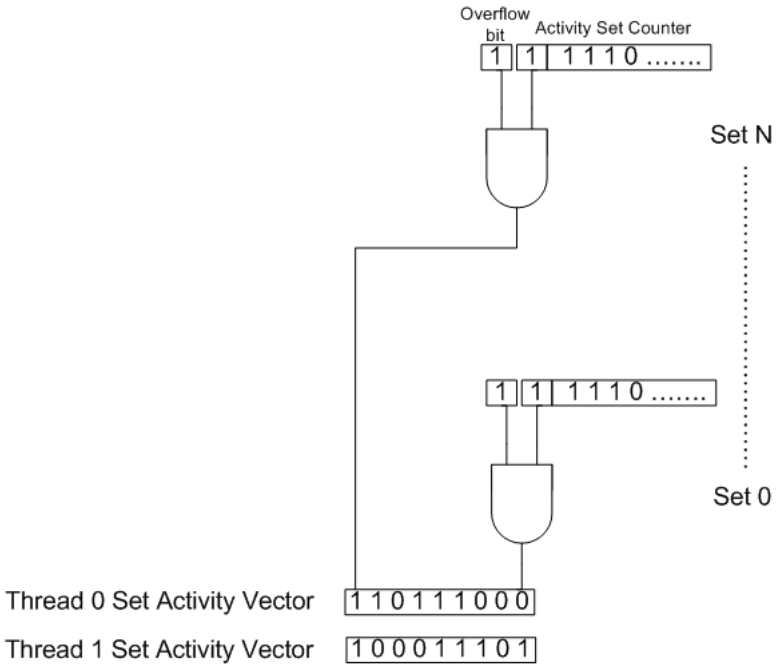
Στην συγκεκριμένη υλοποίηση τα νήματα παραμένουν από την αρχή μέχρι το τέλος της εκτέλεσης στον ίδιο πυρήνα. Σε αυτή την περίπτωση δηλαδή δεν υποστηρίζεται η μετανάστευση νημάτων. Αποτελεί το μέτρο σύγκρισης για τους υπόλοιπους αλγόριθμους που υλοποιήθηκαν. Αξίζει να σημειωθεί ότι αρκετοί CMT επεξεργαστές σήμερα στηρίζονται αποκλειστικά στη δρομολόγηση του λειτουργικού χωρίς την υποστήριξη κάποιου αλγόριθμου σε επίπεδο υλικού.

5.1.2 $2^{η}$ υλοποίηση - DCCS (Data Cache Conflict Scheduling)

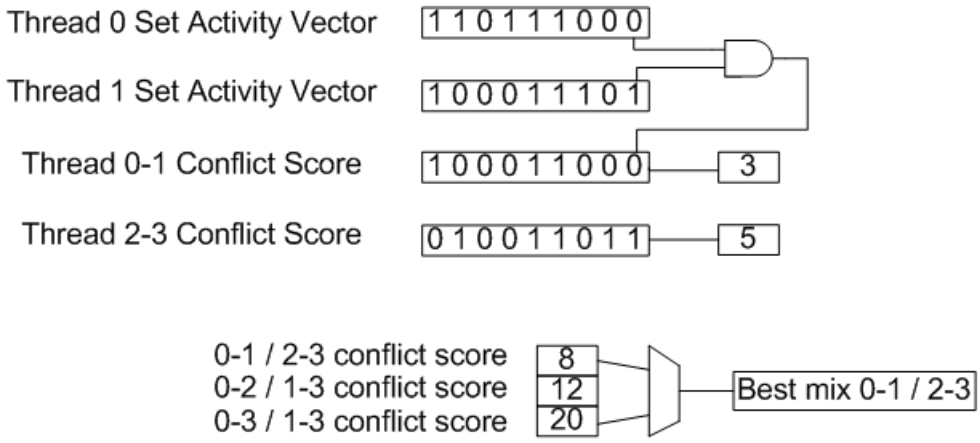
Η υλοποίηση αυτή στηρίζεται πάλι στην περιγραφή των El-Moursy,Garg,Albonesi και Dwarkadas [7]. Όπως φαίνεται στο Σχήμα 5.1 ένας μετρητής (έστω n ψηφίων) σε κάθε νήμα για κάθε σετ κρυφής μνήμης δεδομένων L1, χρησιμοποιείται για να μετρά τις προσβάσεις ενός νήματος στο συγκεκριμένο σετ για μια περίοδο. Επίσης για κάθε νήμα δημιουργείται ένα διάνυσμα δραστηριότητας το οποίο έχει μήκος ίσο με το πλήθος των σετ της κρυφής μνήμης δεδομένων και του οποίου ο ρόλος θα εξηγηθεί στη συνέχεια. Αν ο αριθμός των προσβάσεων ενός νήματος σε κάποιο σετ είναι μεγαλύτερος του 2^{n}+2^{n-1} τότε θεωρούμε ότι το νήμα κάνει μεγάλη χρήση του σετ και θέτουμε μονάδα στο αντίστοιχο ψηφίο του διανύσματος δραστηριότητας του νήματος. Για να υλοποιηθεί αυτό, το πιο σημαντικό ψηφίο και το ψηφίο υπερχείλισης του μετρητή γίνονται λογικά ΚΑΙ,δίνοντας το αντίστοιχο ψηφίο στο διάνυσμα δραστηριότητας.

Τα αντίστοιχα ψηφία στο ζευγάρι διανυσμάτων δραστηριότητας νημάτων γίνονται λογικά ΚΑΙ και το παραγόμενο διάνυσμα ανά ζεύγος νημάτων ονομάζεται διάνυσμα συγκρούσεων ζευγαριού νημάτων(Σχήμα 5.2). Αυτό το διάνυσμα δείχνει τα σετ της κρυφής μνήμης που παρουσιάζουν τον περισσότερο συνωστισμό για αυτά τα δύο (ή περισσότερα στην υλοποίηση μας) νήματα. Αξίζει να σημειωθεί ότι στην περίπτωση των τεσσάρων νημάτων και για δύο πυρήνες υπάρχουν έξι πιθανοί συνδυασμοί.

Στο παράδειγμα του Σχήματος 5.2 φαίνονται τα διανύσματα δραστηριότητας για το νήμα 0 και 1.



Σχήμα 5.1: Δημιουργία των διανυσμάτων δραστηριότητας για τα σετ της κρυφής μνήμης L1D.



Σχήμα 5.2: Ο συνδυασμός των διανυσμάτων δραστηριότητας για την δημιουργία του δείκτη σύγκρουσης και τον καθορισμό του καταλληλότερου συνδυασμού για τα τέσσερα νήματα στην επόμενη περίοδο.

Το προκύπτον διάνυσμα συγκρούσεων των νημάτων 0 και 1 παρουσιάζει τρεις μονάδες, οπότε με το άθροισμά τους ο δείκτης σύγκρουσης γίνεται ίσος με 3. Η αντίστοιχη διαδικασία έχει γίνει και για τα νήματα 2 και 3 υπολογίζοντας δείκτη σύγκρουσης 5. Αφού βρεθεί ο δείκτης σύγκρουσης για όλα τα ζεύγη νημάτων τότε παίρνουμε τον μικρότερο συνολικό δείκτη σύγκρουσης όλων των δυνατών συνδυασμών (στο παράδειγμα τρεις). Σε αυτή την περίπτωση είναι ίσο με 8 όποτε και επιλέγεται η συνδρομολόγηση για την επόμενη περίοδο των νημάτων 0-1 και 2-3 αντίστοιχα.

5.1.3 3η υλοποίηση - IPCS (IPC Scheduling)

Σ' αυτή την υλοποίηση ακολουθήσαμε την περιγραφή του IPCS των El-Moursy,Garg,Albonesi και Dwarkadas [7] η οποία, όπως αναφέρουν, είναι εμπνευσμένη από τη δουλειά του Parekh et al. [22] και στην οποία γίνεται χρήση των τιμών IPC των νημάτων για να οδηγήσουν την δρομολόγηση των νημάτων. Αυτό επιτυγχάνεται με τη χρήση hardware μετρητών για τη μέτρηση του IPC από περίοδο σε περίοδο. Παρατηρήθηκε ότι η καλύτερη επίδοση επιτυγχάνεται με τη δρομολόγηση μαζί των νημάτων με τις μεγαλύτερες τιμές IPC. Η λογική πίσω από αυτό είναι η εξής: η δρομολόγηση μαζί νημάτων που μπορούν να χρησιμοποιήσουν τους πόρους αποδοτικότερα είναι η καλύτερη προσέγγιση, αφού έτσι εμποδίζονται τα αργά νήματα που καταλαμβάνουν πολλούς και για περισσότερη ώρα πόρους λόγω των εντολών τους, που εισάγουν καθυστέρηση (κυρίως αστοχίες κρυφής μνήμης) από το να παρεμποδίσουν τα γρήγορα νήματα. Στην υλοποίηση μας προσθέσαμε μετρητές για κάθε νήμα που μετρούσαν των αριθμό των εντολών που ολοκληρώθηκαν και των κύκλων που χρειάστηκαν να ολοκληρωθούν. Στη συνέχεια υπολογίζοντας το πηλίκο του αριθμού των ολοκληρωμένων εντολών προς διάρκεια σε κύκλους βρίσκουμε το IPC.

5.2 Πειραματικά αποτελέσματα - Ανάλυση αποτελεσμάτων

Η μελέτη της επίδοσης των δύο αλγορίθμων (DCCS , IPCS) έγινε για διάφορους συνδυασμούς μετροπρογραμμάτων και για δύο τιμές διαστήματος δρομολόγησης, 100K κύκλους (fine-grain) και για 10M κύκλους (coarse grain) στον διαμορφωμένο GEMS(Ruby) και Simics. Η εκκίνηση της πραγματικής εξομοίωσης γίνεται όταν όλα τα μετροπρογράμματα του συνδυασμού φτάσουν στην αρχή του διαστήματος που βρέθηκε όπως αναφέρθηκε σε προηγούμενη ενότητα με την βοήθεια του BBV και του Simpoints.

Συγκεκριμένα για να μπορέσουμε να συγχρονίσουμε τα μετροπρογράμματα με αυτό τον τρόπο προβήκαμε στην σχεδίαση ενός module στον Simics το οποίο όταν ένας Simics επεξεργαστής φτάσει στο σημείο που πρέπει να ξεκινήσει την εξομοίωση, απενεργοποιείται. Όταν φτάσει και ο τελευταίος από τους τέσσερις επεξεργαστές στο σημείο αυτό, δηλαδή όλοι πλέον βρίσκονται στο σωστό σημείο έτοιμοι να ξεκινήσουν την εξομοίωση, ενεργοποιεί και πάλι τους υπόλοιπους. Ακριβώς αυτό είναι το χρονικό σημείο όπου όλοι οι επεξεργαστές οι οποίοι τρέχουν κάποιο benchmark είναι συγχρονισμένοι στο αρχικό σημείο του τμήματος, που καθορίστηκε με το BBV και το SimPoint και με αυτό τον τρόπο να παίρνουμε όσο το δυνατόν αντιπροσωπευτικότερα αποτελέσματα για ολόκληρο το μετροπρόγραμμα.

5.2.1 Παράμετροι Εξομοίωσης

Στόχος μας ήταν να εξομοιώσουμε ένα 2 X 2 σύστημα δηλαδή διπύρηνο με κάθε πυρήνα να είναι SMT δύο νημάτων. Ο κάθε πυρήνας έχει τοπική κρυφή μνήμη L1 και όλοι μοιράζονται μια την κρυφή μνήμη L2. Οι παράμετροι κρυφής μνήμης που χρησιμοποιήθηκαν φαίνονται στον πίνακα 5.1.

L1 ICache	32Kb,2-way associative ανά πυρήνα
L1 ICache latency	1 cycle
L1 DCache	32Kb,2-way associative ανά πυρήνα
L1 DCache latency	1 cycle
L2 Cache	2MB,8-way associative
L2 Cache latency	20 cycles

Πίνακας 5.1: Παράμετροι εξομοίωσης

Στη υλοποίηση μας χρησιμοποιήθηκαν τέσσερις πυρήνες. Οι πρώτοι δύο πυρήνες χρησιμοποιήθηκαν για εκτέλεση τυχόν daemons του λειτουργικού. Η χρήση τους δηλαδή επιβαλλόταν για λόγους ακρίβειας και αποφυγής συγκρούσεων του λειτουργικού με τους υπόλοιπους δύο πυρήνες οι οποίοι είναι αυτοί που ουσιαστικά μας ενδιαφέρουν. Στους δύο τελευταίους σε κάθε νήμα τους, τρέχει και ένα μετροπρόγραμμα και με βάση των στατιστικών στα νήματα τους λαμβάνονται αποφάσεις για τη δρομολόγηση. Στο τέλος κάθε περιόδου δρομολόγησης τα στατιστικά που καταγράφονταν, αφού χρησιμοποιηθούν για τον αντίστοιχο αλγόριθμο και την απόφαση δρομολόγησης για την επόμενη περίοδο, μηδενίζονται ώστε να μην επηρεάζουν μελλοντικές αποφάσεις.

5.2.2 Benchmarks - Εκτέλεση εξομοίωσης

Οι συνδυασμοί των μετροπρογραμμάτων που χρησιμοποιήθηκαν (ένα μετροπρόγραμμα ανά νήμα) φαίνονται στον Πίνακα 5.2. Η σειρά που εμφανίζονται στον εν λόγω πίνακα αντιπροσωπεύει την

Σ1	art-gzip-mcf-gap
Σ2	equake-mcf-mesa-parser
Σ3	gap-bzip-parser-gzip
Σ4	gzip-mesa-equake-art
Σ5	mcf-lucas-art-mesa
Σ6	parser-mesa-art-bzip
Σ7	vpr-bzip-mesa-equake
Σ8	art-mesa-lucas-equake
Σ9	lucas-equake-vpr-art
Σ10	mcf-vpr-mesa-lucas

Πίνακας 5.2: Οι συνδυασμοί των μετροπρογραμμάτων που χρησιμοποιήθηκαν στις εξομοιώσεις

σειρά που μοιράζονται τους πυρήνες στην περίπτωση του baseline σεναρίου (static mapping) και φυσικά σε όλους τους αλγόριθμους πριν να γίνει ανακατανομή νημάτων για πρώτη φορά. Όλα τα πιο πάνω μετροπρογράμματα έγιναν compile σε μηχάνημα αρχιτεκτονικής SPARC (Niagara) με όλα τα απαραίτητα optimization flags και έγινε χρήση τους με τις εισόδους αναφοράς. Τελικά καταγράψαμε τα αποτελέσματα για όλους τους αλγόριθμους σε fine-grain και coarse-grain διαστήματα σταματώντας την εξομοίωση, όταν ένα από τα μετροπρογράμματα του συνδυασμού είχε εκτελέσει 200M εντολές.

5.2.3 Μεγέθη υπολογισμού επίδοσης

Για τον υπολογισμό της επίδοσης χρησιμοποιήθηκαν τα πιο κάτω μεγέθη:

- Το άθροισμα του IPC κάθε νήματος του συνδυασμού για κάποια περίπτωση και στην συνέχεια συγκρίνοντάς το με το αντίστοιχο IPC άθροισμα στην baseline περίπτωση. Δηλαδή έχουμε :

$$IPC_{sum} = \sum_{k=1}^{4} IPC_k \text{ όπου } k \text{ ο δείκτης του κάθε νήματος.} \tag{5.1}$$

  Ακολούθως για την εύρεση του ποσοστού βελτίωσης έχουμε

$$\%\ Improvement = (\frac{IPC_{sum}algorithm}{IPC_{sum}baseline} - 1) \times 100 \tag{5.2}$$

  Το πιο πάνω ποσοστό φανερώνει την αύξηση/μείωση της **συνολικής** απόδοσης του συστήματος και μπορεί να μην είναι δίκαιος δείκτης για τη βελτίωση στα νήματα χαμηλού IPC.

- Το Weighted Speedup ή WS το οποίο αποτελεί το άθροισμα των πηλίκων του IPC ενός νήματος σε κάποια περίπτωση προς το IPC στην περίπτωση που αυτό τρέχει μόνο του, δηλαδή το single threaded IPC του.

$$WS_{algorithm} = \sum_{k=1}^{4} \frac{IPC_k algorithm}{IPC_k single} \text{ όπου } k \text{ ο δείκτης του κάθε νήματος.} \quad (5.3)$$

Έπειτα με τον ίδιο τρόπο όπως και πριν βρίσκουμε το ποσοστό βελτίωσης:

$$\% \, Improvement = (\frac{WS_{algorithm}}{WS_{baseline}} - 1) \times 100 \quad (5.4)$$

Στην περίπτωση του WS για τον υπολογισμό της απόδοσης λαμβάνεται υπόψη κυρίως η βελτίωση που υπάρχει ανά νήμα απομονωμένα, οπότε είναι πιο δίκαια μετρική από την προηγούμενη για νήματα που εμφανίζουν χαμηλό IPC.

5.2.4 Αποτελέσματα

Αρχικά παραθέτουμε στον πίνακα 5.3 τις τιμές του IPC για κάθε μετροπρόγραμμα που χρησιμοποιήθηκε στην περίπτωση που αυτό τρέχει μόνο του σε ένα από τους πυρήνες(3 ή 4) του συστήματός μας. Για την εύρεση αυτού του IPC τρέξαμε το μετροπρόγραμμα για 200M εντολές από την αρχή του τμήματος που προσδιόριζαν BBV και SimPoints. Στη συνέχεια θα συγκρίνουμε τα αποτελέσματα για

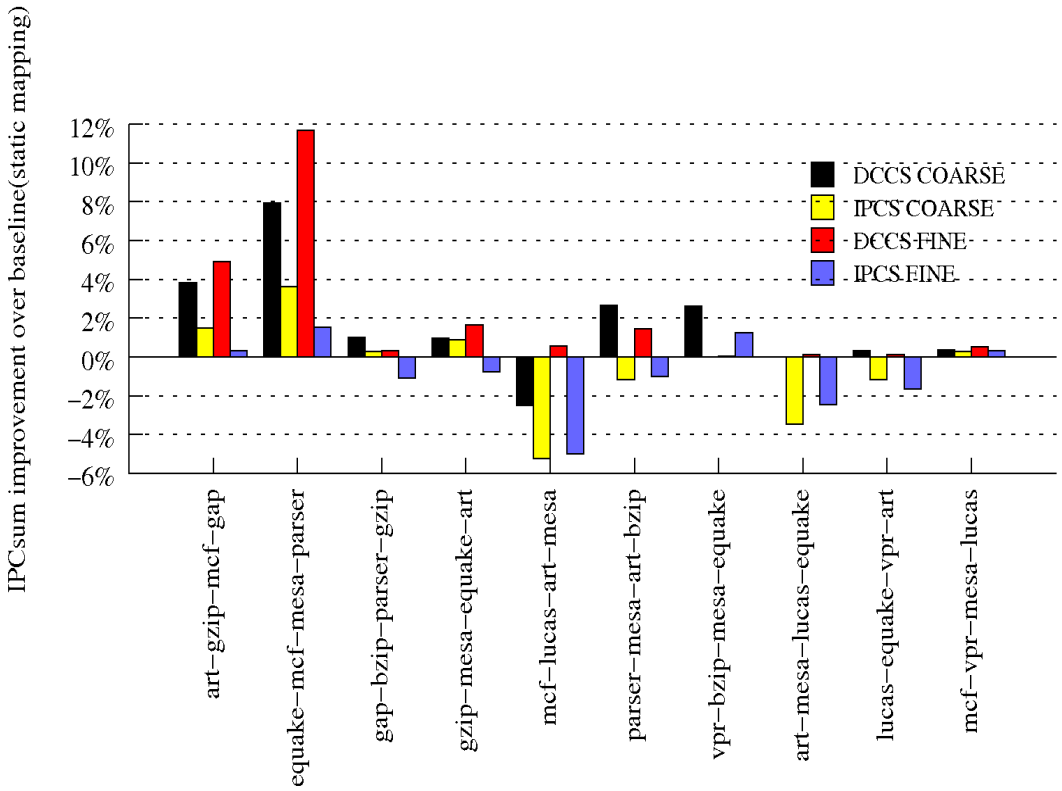
Benchmark	**IPC**$_{single}$
art	0.11779
bzip	0.6439
gap	0.64254
gzip	0.38689
equake	0.16931
lucas	0.2068
mesa	0.83031
mcf	0.09124
parser	0.57775
vpr	0.49062

Πίνακας 5.3: Τα μετροπρογράμματα που χρησιμοποιήθηκαν και τα IPC τους όταν τρέχουν μόνα τους στο σύστημα.

τους DCCS και IPCS σε coarse grain και fine-grain περίοδο δηλαδή κάθε 10M και 100K κύκλους αντίστοιχα.

Βελτίωση στο IPC$_{sum}$ - Βελτίωση επίδοσης συστήματος

Στο Σχήμα 5.3 φαίνεται η βελτίωση του IPC$_{sum}$ σε σχέση με την baseline στις διάφορες περιπτώσεις. Στους περισσότερους συνδυασμούς οι αλγόριθμοί μας παρουσιάζουν βελτίωση εκτός από κάποιες μεμονωμένες περιπτώσεις. Στις περιπτώσεις αυτές κάποιοι αλγόριθμοι, αλλά όχι όλοι, μειώνουν την απόδοση λόγω του ότι σε κάποιο ή κάποια από τα μετροπρογράμματα οι προσβάσεις στα σετ των κρυφών μνημών και το IPC δεν είναι προβλέψιμα από περίοδο σε περίοδο. Δηλαδή, παρουσιάζονται διαφοροποιημένα μοτίβα προσβάσεων στα σετ κρυφής μνήμης από τα νήματα ή αυξομειώσεις στην τιμή του IPC κάποιων νημάτων από περίοδο σε περίοδο(συνδυασμοί 5,8,9). Βελτίωση κυρίως παρατηρείται στις περιπτώσεις όπου τα single threaded IPC των μετροπρογραμμάτων που παρουσιάζονται στο συνδυασμό (συνδυασμοί 1,2) εμφανίζουν μεγάλες διαφορές μεταξύ τους. Έτσι οι επιλογές των



Σχήμα 5.3: Βελτίωση στην μετρική IPCsum σε σχέση με την baseline περίπτωση.

DCCS και IPCS στις περιόδους δρομολόγησης είναι σίγουρα σωστές και αντίστοιχα οι συμπεριφορές των μετροπρογραμμάτων προβλέψιμες και για τις υπόλοιπες περιόδους. Ο DCCS παρουσιάζει καλύτερη επίδοση συστήματος από τον IPCS γιατί ουσιαστικά στο σύστημά μας ο κύριος παράγοντας αύξησης της καθυστέρησης και κατά συνέπεια μείωσης της επίδοσης είναι ο συνωστισμός στις κρυφές μνήμες και οι αστοχίες. Έτσι είναι λογικό ένας αλγόριθμος όπως ο DCCS που κτυπά αυτό το πρόβλημα στην ρίζα του να παρουσιάζει καλύτερη επίδοση.

Παρ'όλα αυτά και ο αλγόριθμος IPCS παρουσιάζει βελτίωση πιο μικρή στις πλείστες περιπτώσεις, αφού υπολογίζοντας μόνο τα IPCs σε κάθε περίοδο χωρίς να παρακολουθεί τις προσβάσεις στα σετ της L1 είναι δυνατό να δρομολογεί μαζί δύο νήματα που ναι μεν έχουν υψηλό IPC αλλά μπορεί να έχουν το μεγαλύτερο σύνολο προσβάσεων στα ίδια σετ κρυφής μνήμης. Έτσι αυτόματα αυξάνει το συνωστισμό και τις αστοχίες σε εκείνο τον πυρήνα την επόμενη περίοδο με επιπτώσεις και στο IPC. Στις περιπτώσεις που το single threaded IPC κάποιων μετροπρογραμμάτων του συνδυασμού είναι παραπλήσιο τότε η επιλογή της συνδρομολόγησης για την επόμενη περίοδο γίνεται ακόμη δυσκολότερη και μπορεί να επιφέρει ακόμη και χειρότερη επίδοση από την baseline (Συνδυασμοί 5,8,9).

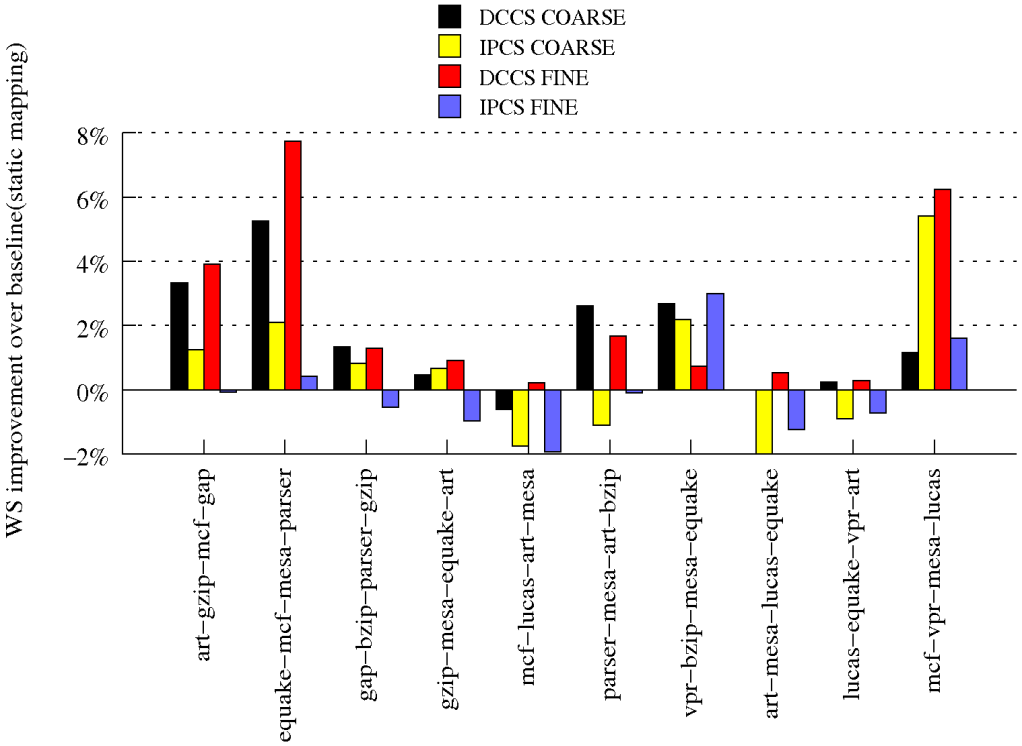
Όσον αφορά στα διαστήματα δρομολόγησης, καλύτερα αποτελέσματα φαίνεται να έχουμε στην fine-grained προσέγγιση (κυρίως στον DCCS) όπου τα στατιστικά συλλέγονται πιο συχνά από ότι στην coarse grain περίπτωση και άρα μπορούν να εξαχθούν καλύτερα συμπεράσματα (συνδυασμοί 1,2,4,5,10). Αυτό φυσικά μπορεί να επιφέρει και το αντίθετο αποτελέσμα σε κάποιες συγκεκριμένες περιπτώσεις, που κάποια μετροπρογράμματα μπορεί να μην έχουν ομοιόμορφη πρόσβαση στα σετ της κρυφής μνήμης ή να παρουσιάζουν IPC με αυξομειώσεις από περίοδο σε περίοδο, όπως αναφέρθηκε προηγουμένως. Οι συχνές εναλλαγές στην κατανομή των νημάτων σε μια τέτοια περίπτωση οδηγούν σε κάποιες λανθασμένες επιλογές δρομολόγησης που στοιχίζουν σε επίδοση (βελτιώνουν μεν αλλά όχι

στον βαθμό που πρέπει) σε αντίθεση με την επιλογή μιας κατανομής για περισσότερο διάστημα (coarse-grained) το οποίο θα επηρεαζόταν από αυτές τις ανομοιόμορφες προσβάσεις σε μια μεγαλύτερη περίοδο αλλά όχι στον βαθμό του fine-grained (Συνδυασμοί 3,6).

Η καλύτερη επίδοση γενικότερα εμφανίζεται στον Συνδυασμό 2 στην περίπτωση του DCCS fine αλγόριθμου που εμφανίζεται μια βελτίωση γύρω στο 11.7% σε ένα συνδυασμό όπου όλοι οι αλγόριθμοί μας παρουσιάζουν βελτίωση στην επίδοση του συστήματος. Η χειρότερη επίδοση αντίστοιχα εμφανίζεται στον Συνδυασμό 5 στον IPC coarse-grained με ένα ποσοστό μείωσης επίδοσης της τάξης του 5%. Σε πολλές περιπτώσεις τέλος που άλλες μέθοδοι δρομολόγησης παρουσιάζουν χειρότερη επίδοση από την baseline, ο αλγόριθμος DCCS παρουσιάζει αν όχι βελτίωση στην επίδοση, την λιγότερο χειρότερη όπως π.χ. στον Συνδυασμό 5.

Weighted Speedup - Βελτίωση στην μεμονωμένη επίδοση κάθε νήματος

Το Σχήμα 5.4 παρουσιάζει τη βελτίωση στην επίδοση σύμφωνα με την WS μετρική.



Σχήμα 5.4: Βελτίωση στην μετρική WS σε σχέση με την baseline περίπτωση.

Τα αποτελέσματα που παρουσιάζονται στο Σχήμα 5.4 είναι λίγο πολύ αναμενόμενα. Ο αλγόριθμος DCCS παρουσιάζει τις καλύτερες επιδόσεις σε όλους τους συνδυασμούς, αφού αντιμετωπίζει ως ίσα όλα τα νήματα, ανεξαρτήτως του IPC που εμφανίζουν σε κάθε περίοδο, ελέγχοντας μόνο σε ποια σετ κρυφής μνήμης έχουν πρόσβαση. Έτσι παρατηρείται μια γενική βελτίωση επίδοσης σε όλα τα νήματα του συνδυασμού. Αντίθετα στην περίπτωση του IPCS γίνεται διαχωρισμός των νημάτων, τα δύο με τα μεγαλύτερα IPC σε μια περίοδο δρομολογούνται μαζί και αντίστοιχα τα δύο με το μικρότερο IPC. Αυτό έχει σαν αποτέλεσμα την αύξηση του μεμονωμένου κόστους σε επίδοση στα δύο τελευταία, αφού, τοποθετώντας μαζί το IPC τους γίνεται ακόμη πιο μικρό σε αντίθεση με τα δύο πρώτα που ευνοούνται περισσότερο. Αυτό σε κάποιες περιπτώσεις μπορεί να αποβεί μοιραίο, αφού ενώ ο DCCS μπορεί να παρουσιάσει βελτίωση, ο IPCS μπορεί να παρουσιάσει χειρότερη απόδοση από την baseline (Συνδυασμοί 5,8,9). Αυτή η κατάσταση γίνεται ακόμη χειρότερη είτε όταν τα single threaded

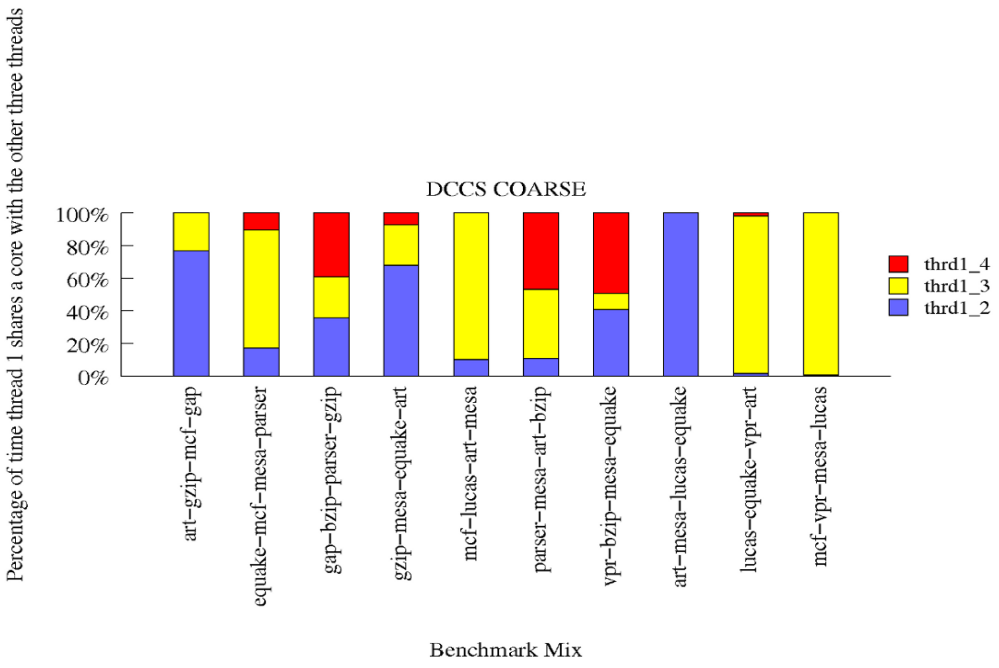
IPCs των γρήγορων νημάτων έχουν μεγάλη διαφορά τουλάχιστον του ενός από τα πιο αργά (Συνδυασμοί 4,6) είτε στην περίπτωση που έχουμε ένα ουσιαστικά γρήγορο μετροπρόγραμμα που λόγω των τριών πιο αργών ζημιώνει σε επίδοση(Συνδυασμοί 5,8) . Υπάρχουν φυσικά περιπτώσεις όπου παρόλο που ισχύει αυτό παρουσιάζεται κάποια βελτίωση, όχι σε ικανοποιητικά επίπεδα σε σχέση με τον DCCS(Συνδυασμός 2).

Όσον αφορά στην περίοδο δρομολόγησης σε αυτή την μετρική η fine-grained προσέγγιση έχει καλύτερα αποτελέσματα στον DCCS (συνδυασμοί 1,2,3,4,8,9,10) σε σχέση με την coarse-grained ενώ στον IPCS συμβαίνει το αντίθετο. Για να αποφασίσουμε ποια περίοδος δρομολόγησης είναι καλύτερη στον IPCS μεγάλο ρόλο παίζει η συμπεριφορά των νημάτων του συνδυασμού από περίοδο σε περίοδο και πιο συγκεκριμένα οι απότομες ή ομαλότερες μεταβολές του IPC τους.

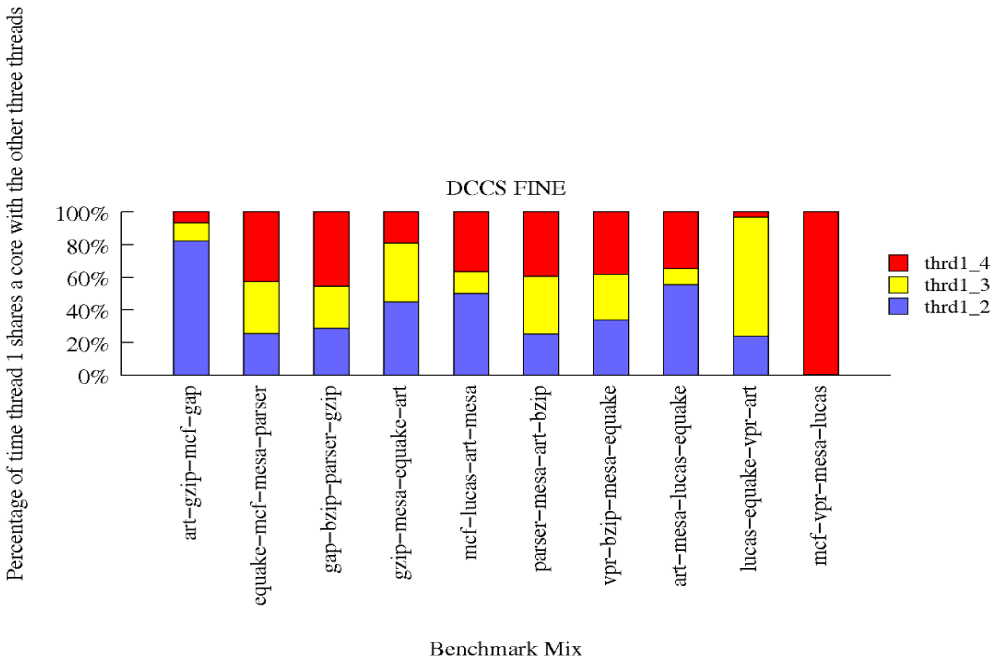
Την καλύτερη επίδοση σε αυτή την μετρική έχει ο DCCS fine-grained στο Συνδυασμό 2 με ένα ποσοστό βελτίωσης κοντά στο 8% ενώ η χαμηλότερη σημειώνεται για τον IPCS fine-grained στον Συνδυασμό 8 με μείωση επίδοσης της τάξης του 2%.

Ποσοστά συνδρομολόγησης νημάτων

Στην συνέχεια θα παρουσιαστούν τα ποσοστά της συνδρομολόγησης των νημάτων με το νήμα 1 για τους διάφορους συνδυασμούς στις coarse-grained και fine-grained προσεγγίσεις στους DCCS και IPCS. Αξίζει να σημειωθεί ότι τα σχεδιαγράμματα που θα ακολουθήσουν παρουσιάζουν απλώς το ποσοστό που κατέχει ένας συνδυασμός σε σχέση με την συνολική εκτέλεση και όχι την χρονική σειρά με την οποία εμφανίζονται οι συνδυασμοί των νημάτων στους πυρήνες.
Το Σχήμα 5.5 παρουσιάζει την περίπτωση του DCCS coarse grained ενώ το Σχήμα 5.6 την αντίστοιχη fine-grained περίπτωση.



Σχήμα 5.5: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του DCCS coarse grained.



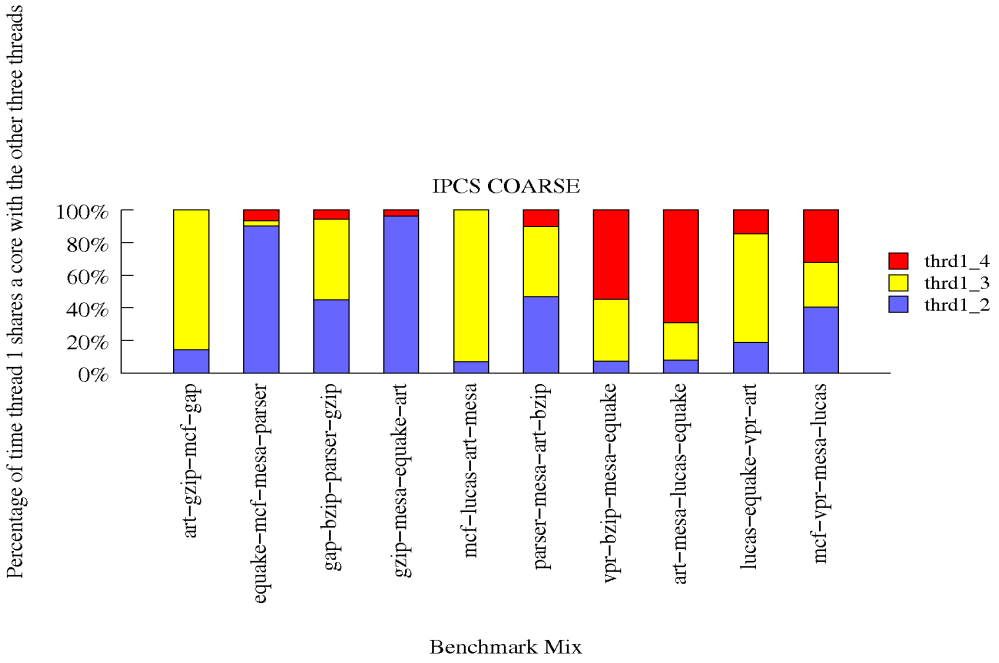
Σχήμα 5.6: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του DCCS fine grained.

Με τη σύγκριση των δύο σχημάτων βλέπουμε ότι fine-grained και coarse-grained πολιτικές παρουσιάζουν σε αρκετές περιπτώσεις παρόμοια ποσοστά συνδρομολόγησης μεταξύ των νημάτων (Συνδυασμοί 1,3,4,6,7) σε κάποιες άλλες μικρές διαφορές στα ποσοστά (Συνδυασμοί 2,5,8,9) και στην περίπτωση 10 εντελώς διαφορετική συμπεριφορά. Στην πρώτη ομάδα μπορεί να εμφανιστούν πολύ μικρά ποσοστά κάποιου καινούργιου συνδυασμού στην περίπτωση του fine-grained (Συνδυασμός 1) που όμως μπορεί να επιφέρει βελτίωση 1-1.5% περισσότερη από ότι στην coarse-grained. Αντίθετα στην περίπτωση που τα παρόμοια ποσοστά παρουσιάζουν σχεδόν μηδενική διαφορά τότε η coarse-grained τακτική είναι καλύτερη γιατί αποφεύγουμε ενδιάμεσες ανακατανομές που μπορεί να δημιουργήσουν περισσότερες αστοχίες στην κρυφή μνήμη λόγω των συνεχών εναλλαγών ενός νήματος σε κάποιο πυρήνα.

Για τη δεύτερη ομάδα συνδυασμών η βελτίωση που εισάγει η fine-grained πολιτική είναι ξεκάθαρη σε όλες τις περιπτώσεις εκτός του Συνδυασμού 9 αφού η συχνότητα επιλογής του καταλληλότερου συνδυασμού είναι μεγαλύτερη και έτσι επιλέγονται ορθότερες συνδρομολογήσεις νημάτων σε πυρήνες για την επόμενη περίοδο. Τέλος στον τελευταίο συνδυασμό ο DCCS coarse-grained επιλέγει ένα εντελώς διαφορετικό συνδυασμό νημάτων σε σχέση με την fine-grained υλοποίηση του. Τα στατιστικά που συλλέγονται ανά περίοδο στην πρώτη περίπτωση οδηγούν τον DCCS να συνδρομολογήσει τα νήματα 1 και 3 από την αρχή μέχρι το τέλος της εκτέλεσης ενώ αντίθετα στην δεύτερη συνδρομολογούνται τα νήματα 1 και 4. Ο fine-grained DCCS καταφέρνει τελικά να εισάγει ελαφρώς καλύτερη βελτίωση στην επίδοση σε σχέση με τον coarse-grained στην IPC$_sum$ μετρική αλλά στην WS μετρική η διαφορά στην βελτίωση που προκύπτει είναι χαρακτηριστική.

Τα Σχήματα 5.7 και 5.8 είναι τα αντίστοιχα για τα ποσοστά συνδρομολόγησης στο IPCS.

Σχεδόν σε όλες τις περιπτώσεις τα νήματα που είχαν τα μεγαλύτερα single threaded IPCs, όπως ήταν λογικό, δρομολογούνται μαζί με το μεγαλύτερο ποσοστό συνδρομολόγησης όπως επιβάλλεται από

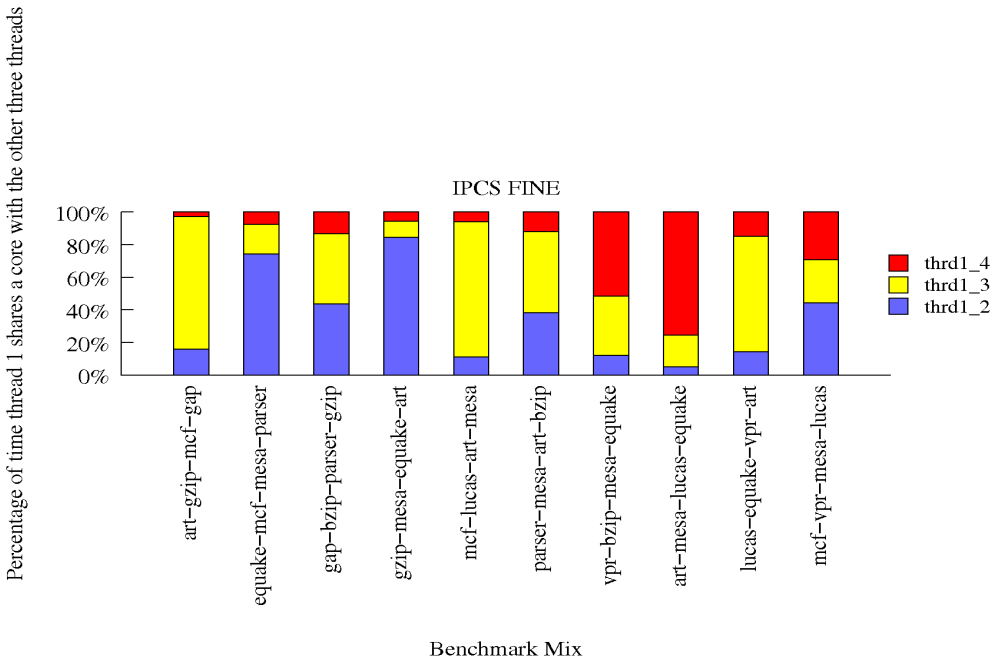


Σχήμα 5.7: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του IPCS coarse grained.

τον αλγόριθμο IPCS. Τα ποσοστά που παρατηρούνται για κάθε συνδυασμό νημάτων είναι παρόμοια. Όμως στην περίπτωση όμως του fine-grained οι ανακατανομές είναι συχνότερες γεγονός που μπορεί να προκαλέσει περισσότερες αστοχίες και κατά συνέπεια την μείωση της απόδοσης σε αυτή την πολιτική σε σχέση με την coarse-grained. Δρομολόγηση δύο αργών νημάτων μαζί στον ίδιο πυρήνα επιφέρει ακόμη περισσότερο συνωστισμό στον συγκεκριμένο και κατά συνέπεια επιπτώσεις στην επίδοση του συστήματος καθώς και στα αργά νήματα όσον αφορά στην μεμονωμένη βελτίωση επίδοσης.

Τα ποσοστά συνδρομολόγησης σχεδόν σε όλους σχεδόν τους συνδυασμούς παρουσιάζουν ελαφρά διαφορετικές τιμές. Σε κάποιες περιπτώσεις στην fine-grained εμφανίζονται και κάποιοι καινούργιοι συνδυασμοί. Μερικές φορές η fine-grained φαίνεται να υστερεί της coarse-grained πολιτικής (Συνδυασμοί 1,2,3,4,5) ενώ στις υπόλοιπες παρουσιάζει καλύτερα αποτελέσματα (συνδυασμοί 6,7,8,9,10). Εδώ φαίνεται ότι η σημαντική διαφορά που παρατηρήθηκε από τον DCCS είναι ότι για τον IPCS η fine-grained περίπτωση δεν φαίνεται ξεκάθαρα να είναι καλύτερη από την coarse-grained και στις δύο μετρικές. Αυτό εξηγείται ως εξής: Στις περιπτώσεις όπου τα IPCs των μετροπρογραμμάτων του συνδυασμού διαφέρουν χαρακτηριστικά και τα τέσσερα μεταξύ τους (Συνδυασμοί 1,2,3,4,5) τότε χρήση της fine-grained πολιτικής δεν έχει πολύ νόημα αφού φαίνεται ξεκάθαρα το πως πρέπει να συνδρομολογηθούν τα νήματα. Αντίθετα εισάγει μικρά ποσοστά κάποιου διαφορετικού συνδυασμού ως αποτέλεσμα της διαφοροποίησης της συμπεριφοράς ενός νήματος σε κάποιες περιόδους. Οδηγεί έτσι σε λανθασμένες συνδρομολογήσεις στις περιόδους που τις ακολουθούν, επηρεάζοντας τελικά τη μεμονωμένη απόδοση των νημάτων και τη συνολική απόδοση του συστήματος.

Για παράδειγμα στον πρώτο συνδυασμό έχουμε τα art,gzip,mcf,gap. Τα αντίστοιχα single threaded IPCs τους είναι 0.11779, 0.38669, 0.09124 και 0.64254. Παρατηρώντας κάποιος αυτά τα IPC σίγουρα θα περιμένει με βάση τον IPCS τα νήματα 2 και 4 να είναι μαζί τις περισσότερες περιόδους και αντίστοιχα τα 1 και 3, πράγμα που ισχύει τόσο σε coarse-grained όσο και σε fine-grained. Στην πε-



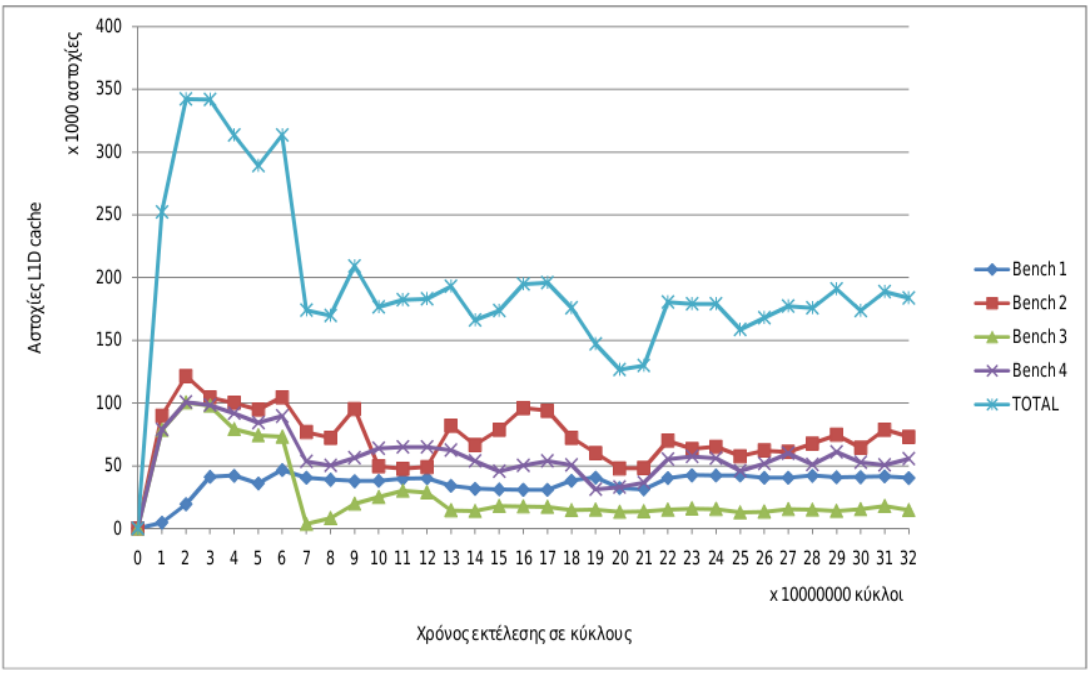
Σχήμα 5.8: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του IPCS fine grained.

ρίπτωση του fine-grained παρουσιάζεται και ένα μικρό ποσοστό του συνδυασμού νημάτων 1 και 4 που στην coarse-grained δεν εμφανίστηκε. Αυτό το ποσοστό εμφανίζεται όταν σε κάποια fine-grained περίοδο το art εμφανίσει μεγαλύτερο IPC του gzip και έτσι ο αλγόριθμος μας αποφασίζει την συνδρομολόγησή του στην επόμενη περίοδο μαζί με το νήμα 4 που είναι αδιαμφισβήτητα το γρηγορότερο. Το πρόβλημα όμως που εμφανίζεται έγκειται στο γεγονός ότι σε αυτή την περίοδο που 1 και 4 τρέχουν στον ίδιο πυρήνα, το νήμα 2 μπορεί να εμφανίσει μεγαλύτερο IPC από το 1. Άρα η απόφαση που πάρθηκε στην προηγούμενη περίοδο, παρόλο που ήταν φαινομενικά σωστή, τελικά να αποδειχτεί λανθασμένη. Αυτό φυσικά δεν συμβαίνει στην coarse-grained περίπτωση όπου τα διαστήματα στα οποία λαμβάνονται μετρήσεις και αποφάσεις είναι μεγαλύτερα και τέτοιες απότομες αυξομειώσεις μεταξύ των IPCs των νημάτων είναι εξαιρετικά δύσκολο να εμφανιστούν. Υπάρχουν βέβαια και περιπτώσεις όπου τα ενδιάμεσα IPCs των νημάτων δεν συμπεριφέρονται τόσο απρόβλεπτα, οπότε και η περίπτωση fine-grained εισάγει βελτίωση σε σχέση με την coarse-grained. Στους συνδυασμούς όπου κάποια μετροπρογράμματα παρουσιάζουν παραπλήσια single-threaded IPCs (Συνδυασμοί 6,7,8,9), η χρήση της fine-grained μεθόδου συνιστάται για καλύτερη παρακολούθηση των IPCs σε μικρότερες περιόδους, δίνοντας καλύτερες βελτιώσεις από την coarse-grained σε μεμονωμένη και συνολική απόδοση.

Το βασικό συμπέρασμα που πρέπει να κρατήσουμε είναι ότι απαραίτητη προϋπόθεση για να μπορεί να εισάγει βελτίωση η fine-grained περίπτωση στον IPCS, είναι να μην υπάρχουν απότομες μεταβολές στα IPCs των νημάτων μεταξύ δύο συνεχόμενων περιόδων, οδηγώντας έτσι σε λανθασμένες αποφάσεις δρομολόγησης. Αυτό δεν ισχύει στην περίπτωση του IPCS coarse-grained ο οποίος δεν είναι τόσο ευαίσθητος σε τέτοιες περιπτώσεις αφού το χρονικό διάστημα δρομολόγησης είναι αρκετά μεγαλύτερο.

Μελέτη αστοχιών - συνδρομολογήσεων στην περίπτωση του equake-mcf-mesa-parser

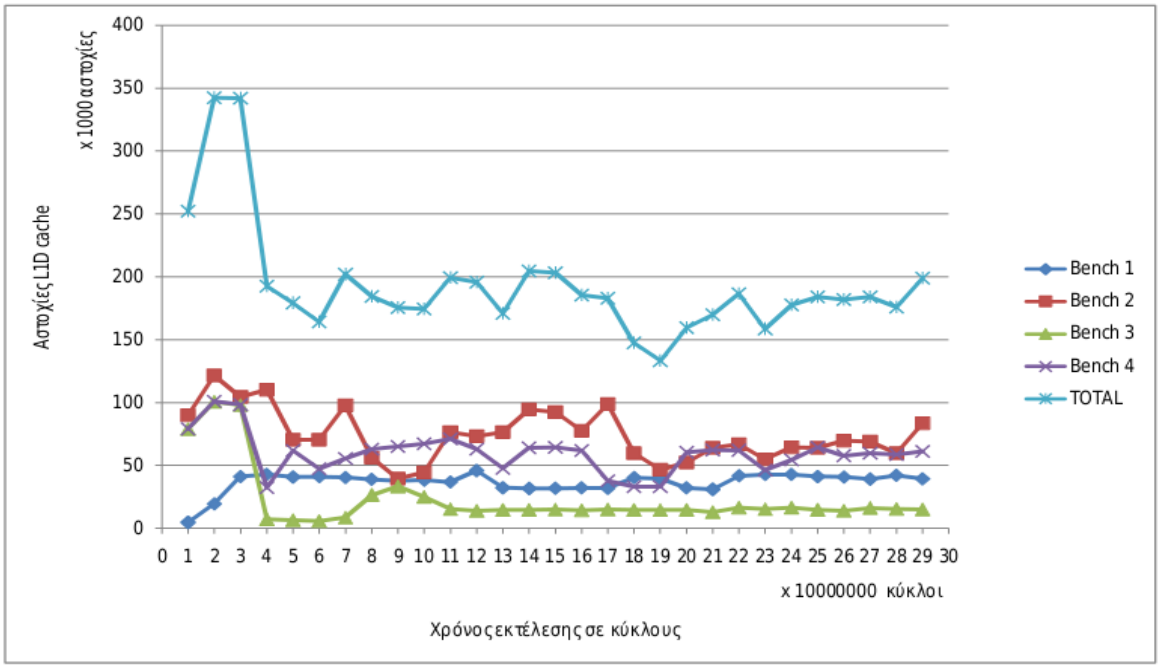
Σε αυτή την υποενότητα θα μελετήσουμε την μεταβολή στις αστοχίες κρυφής μνήμης L1D συνολικές και νημάτων σε κάθε περίοδο δρομολόγησης καθώς επίσης και τον εκάστοτε συνδυασμό νημάτων στους πυρήνες. Πιο συγκεκριμένα θα μελετήσουμε την coarse grained πολιτική των DCCS και IPCS. Αρχικά παραθέτουμε το Σχήμα 5.9 που παρουσιάζει τις αστοχίες L1D κρυφής μνήμης κατά περίοδο στην baseline περίπτωση και το οποίο θα χρησιμοποιηθεί για σκοπούς σύγκρισης.



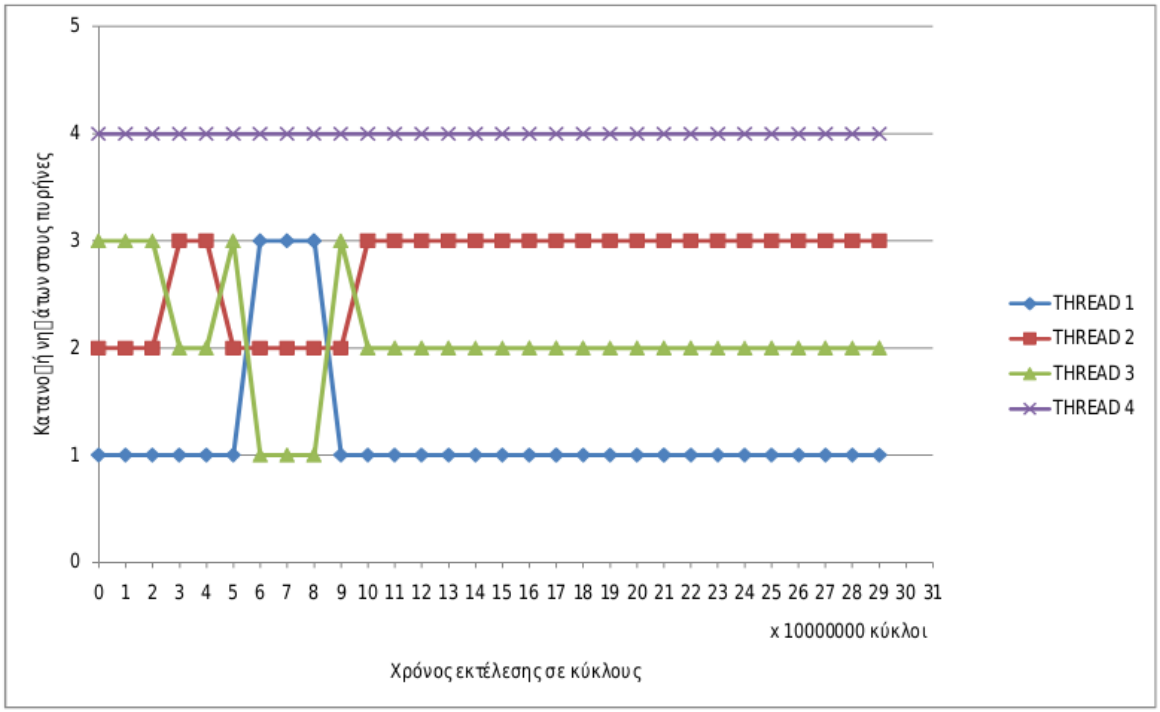
Σχήμα 5.9: Οι αστοχίες κρυφής μνήμης L1D κάθε νήματος και οι συνολικές σε κάθε περίοδο στην baseline.

Το Σχήμα 5.10 παρουσιάζει τις αστοχίες κρυφής μνήμης L1D που γνωρίζει το νήμα κάθε μετρο-πρόγραμμα σε μία περίοδο ενώ αντίστοιχα το Σχήμα 5.11 παρουσιάζει την κατανομή των νημάτων στους πυρήνες σε κάθε περίοδο για την περίπτωση του DCCS.

Η βασικότερη μείωση στις αστοχίες σε σχέση με την baseline παρατηρείται στις περιόδους 30 M - 60 M κύκλους. Πιο συγκεκριμένα οι συνολικές αστοχίες για αυτό το διάστημα είναι μικρότερες των 200000 ανά περίοδο σε αντίθεση με την baseline που κυμαίνονται γύρω στις 300000. Για τα νήματα 3 και 4 παρατηρείται επίσης στην ίδια περίοδο πτώση στον αριθμό των αστοχιών σε σχέση με την baseline. Στις επόμενες περιόδους τα ποσοστά των αστοχιών είναι σε γενικές γραμμές ελαφρώς καλύτερα από ότι στην baseline. Παρατηρώντας το Σχήμα 5.10 θα δούμε ότι για το μεγαλύτερο ποσοστό της εκτέλεσης ο αλγόριθμος καταφέρνει να κρατήσει τις συνολικές αστοχίες σε χαμηλότερα επίπεδα απ'ότι στις αρχικές περιόδους εφαρμογής του. Σε τρία από τα τέσσερα νήματα, οι αστοχίες για τις περιόδους που βρίσκονται μετά τους 50M κύκλους εκτέλεσης είναι σε χαμηλότερα επίπεδα από ότι γι'αυτές που βρίσκονται πριν το χρονικό σημείο των 50 M κύκλων. Εξαίρεση αποτελεί το νήμα 1 (equake) για το οποίο συμβαίνει ακριβώς το αντίθετο. Παρ'όλα αυτά οι αστοχίες που σημειώνονται δεν παρουσιάζουν συνεχή αύξηση αλλά παραμένουν σταθερές χωρίς να αυξάνονται γύρω από την τιμή των 45000. Σε γενικότερες γραμμές η συνολική βελτίωση που παρατηρήθηκε σε σχέση με την baseline οφείλεται στην μείωση των ενδιάμεσων αστοχιών ανά περίοδο και η διατήρηση τους σε ένα σταθερό ποσοστό. Η μείωση φυσικά που επιτυγχάνεται δεν μπορεί να ξεπεράσει κάποιο όριο εξαιτίας της φύσεως των μετροπρογραμμάτων μας (παρουσιάζουν αρκετές αστοχίες από μόνα τους).



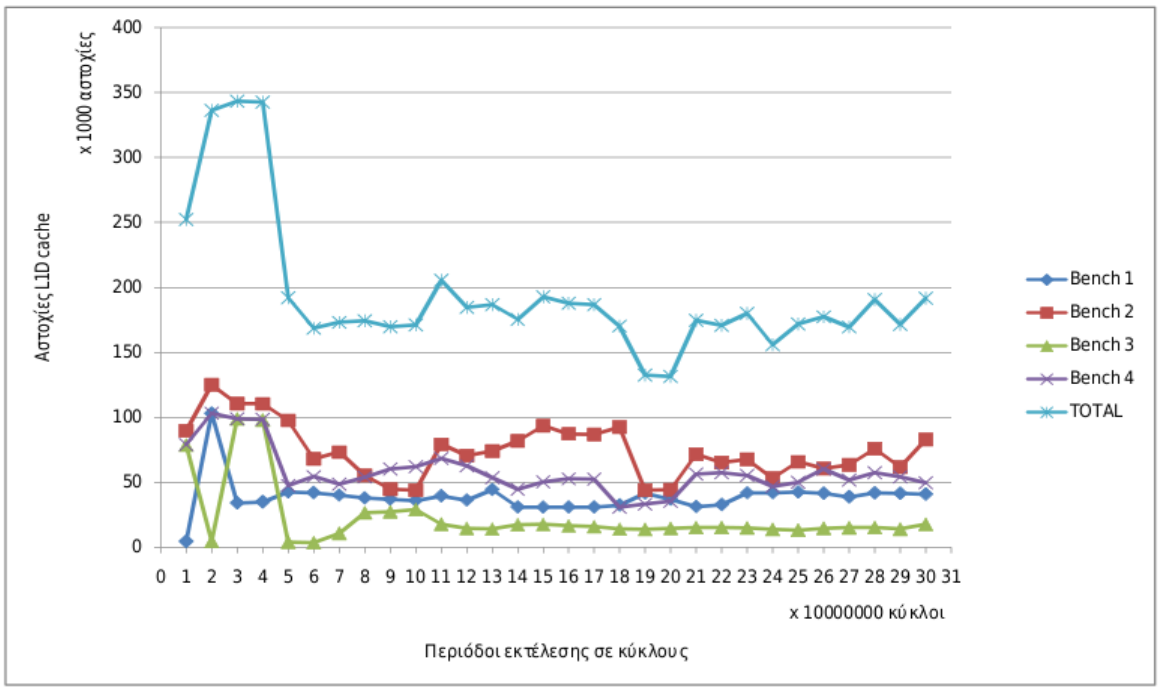
Σχήμα 5.10: Οι αστοχίες κρυφής μνήμης L1D κάθε νήματος και οι συνολικές σε κάθε περίοδο στον DCCS.



Σχήμα 5.11: Η κατανομή των νημάτων στους πυρήνες σε κάθε περίοδο στον DCCS.

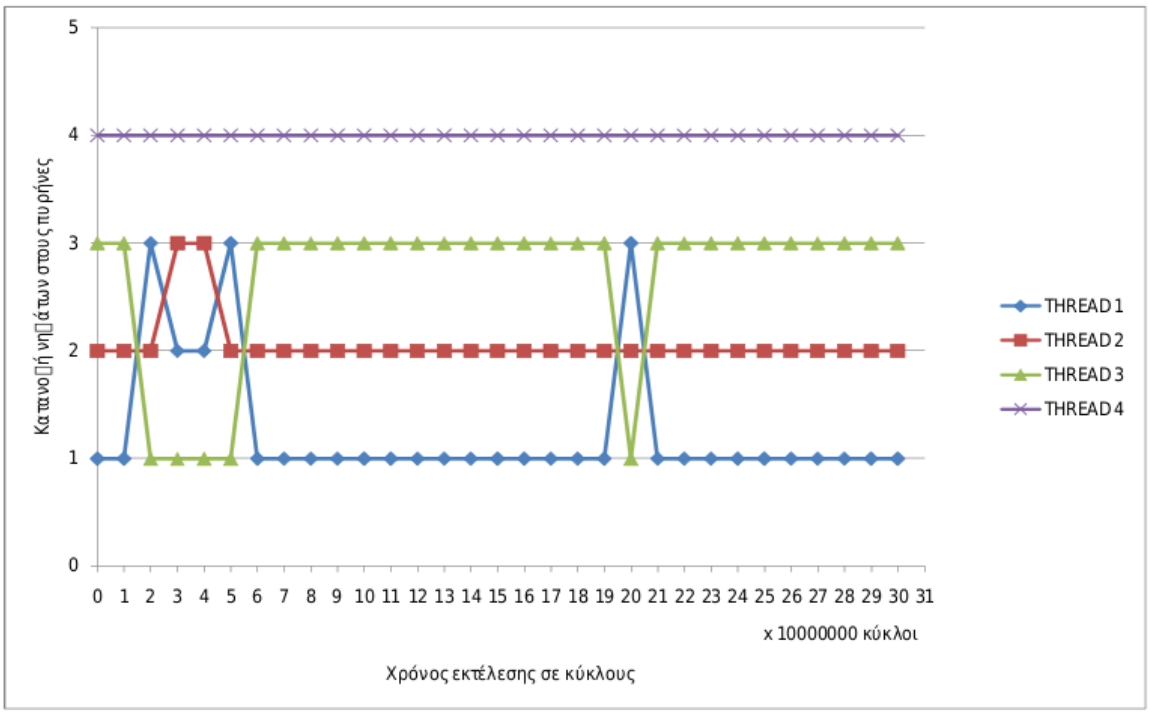
Το Σχήμα 5.11 παρουσιάζει τη συνδρομολόγηση των νημάτων στους πυρήνες. Βλέπουμε ότι στο μεγαλύτερο διάστημα ο αλγόριθμός μας δρομολογεί μαζί νήματα με τέτοιο τρόπο, ώστε να παρατηρείται μείωση στις αστοχίες τους σε σχέση με τις αρχικές περιόδους πριν καλά καλά να εφαρμοστεί ο αλγόριθμος. Πιο συγκεκριμένα στις περιόδους μεγαλύτερες των 110 M κύκλων τα νήματα 1 και 3 συνδρομολογούνται στον ίδιο πυρήνα. Αντίστοιχα οι αστοχίες που παρουσιάζουν σε αυτές τις περιόδους είναι και οι λιγότερες από όλα τα νήματα γεγονός που αποδεικνύει την ορθότητα του DCCS. Ο DCCS μελετώντας τις συγκρούσεις μεταξύ των νημάτων κατάφερε να δρομολογήσει μαζί τα νήματα τα οποία θα εμφάνιζαν όσο το δυνατό λιγότερο συνωστισμό - αστοχίες μαζί. Ουσιαστικά αποφεύγει τις αλλεπάλληλες αστοχίες μεταξύ των νημάτων που μοιράζονται τον ίδιο πυρήνα. Μπορεί η χρήση του DCCS να παρουσιάζει μειωμένα ποσοστά αστοχιών και να εισάγει κάποια βελτίωση, αλλά λόγω της φύσης των μετροπρογραμμάτων που χρησιμοποιούμε, τα οποία παρουσιάζουν από μόνα τους αρκετές αστοχίες σε ένα σύστημα, είναι δύσκολο να περιορίσουμε τις αστοχίες κάτω από ένα όριο.

Στην περίπτωση του IPC αλγόριθμου τα αποτελέσματα φαίνονται στα Σχήματα 5.12 και 5.13.



Σχήμα 5.12: Οι αστοχίες κρυφής μνήμης L1D κάθε νήματος και οι συνολικές σε κάθε περίοδο στον IPCS.

Όπως και στον DCCS εδώ στην περίοδο 40 M - 60 M παρατηρήθηκε η αντίστοιχη πτώση των συνολικών αστοχιών L1D κρυφής μνήμης σε σχέση με την baseline. Αντίστοιχα παρουσιάστηκε μείωση και στις αστοχίες των νημάτων 3 και 4. Στις περιόδους που ακολουθούν τα ποσοστά αστοχιών βρίσκονται, αν όχι σε καλύτερα επίπεδα στα ίδια με την baseline. Στην περίπτωση του IPCS αρχικά παρατηρούμε ότι σύμφωνα με το Σχήμα 5.13 στο μεγαλύτερο διάστημα της εκτέλεσης, συνδρομολογούνται τα νήματα που επέδειξαν το μεγαλύτερο IPC στην single threaded εκτέλεσή τους και αντίστοιχα για τα πιο άργα, γεγονός που αποδεικνύει την ορθότητα του αλγόριθμου μας. Οι συνολικές αστοχίες ανά περίοδο κινούνται σε επίπεδα μεγαλύτερα από τον DCCS (π.χ διαστήματα 30 M -40 M κύκλων,100 M -110 M κύκλων,120 M -130 M κύκλων) , απόλυτα αναμενόμενο γιατί η εφαρμογή του IPCS επέφερε μικρότερη βελτίωση μεμονωμένη και συστήματος σε σχέση με τον DCCS. Μία ακόμη ερμηνεία έγκειται στο γεγονός ότι ο IPCS παρακολουθεί μόνο το IPC για να αποφασίσει την συνδρομολόγηση των νημάτων στην επόμενη περίοδο. Η αδυναμία του αλγόριθμου έγκειται στο γεγονός ότι τα δύο νήματα



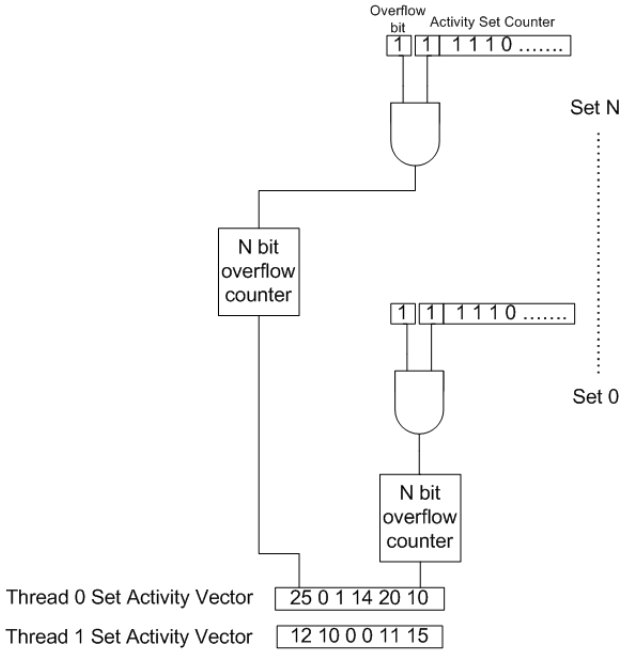
Σχήμα 5.13: Η κατανομή των νημάτων στους πυρήνες σε κάθε περίοδο στον IPCS.

με το μεγαλύτερο IPC σε μια περίοδο, μπορεί να παρουσιάζουν τις περισσότερες κοινές προσβάσεις στα ίδια σετ κρυφής μνήμης. Έτσι, η συνδρομολόγησή τους μπορεί να εμφανίσει μεγάλες ποινές στα IPC τους και κατά συνέπεια μείωση της συνολικής επίδοσης του συστήματος. Επίσης παρατηρούνται αυξημένες αστοχίες κατά περιόδους στα πιο αργά νήματα, γεγονός που εξηγεί και την μειωμένη βελτίωση στην WS μετρική (μεμονωμένη βελτίωση στο IPC των νημάτων). Φυσικά επιπτώσεις στην βελτίωση μπορεί να εμφανιστούν και λόγω των αυξημένων αστοχιών κρυφής μνήμης εντολών αλλά και των αστοχιών στην κρυφή μνήμη L2. Σε ένα σύστημα όπως το δικό μας όπου οι συγκρούσεις στην κρυφή μνήμη είναι ο κυρίαρχος παράγοντας εισαγωγής καθυστέρησης, ένας τέτοιος αλγόριθμος ήταν αναμενόμενο να οδηγήσει σε χαμηλότερες βελτιώσεις τόσο σε επίπεδο συστήματος όσο και νήματος.

5.3 4η υλοποίηση - DCCS(Data Cache Conflict Scheduling) modified με χρήση μετρητών για το overflow bit

Ο αλγόριθμος αυτός προσπαθεί να βελτιώσει κάποιες ειδικές περιπτώσεις του απλού DCCS αλγόριθμου. Κατά την διάρκεια των δοκιμών του απλού αλγόριθμου παρατηρήθηκε το εξής: Πολλές φορές οι μετρητές για τις προσβάσεις στα σετ κρυφής μνήμης για ένα νήμα υπερχείλιζαν περισσότερες από μία φορές. Όπως φαίνεται, ένας μετρητής σταθερού μήκους bit μπορεί σε κάποιες περιπτώσεις να υπερχειλίζει μία φόρα όπως περιγράφεται στον απλό αλγόριθμο, αλλά στην περίπτωση που οι περισσότερες προσβάσεις γίνονται σε κάποια συγκεκριμένα σετ τότε είναι πιθανό οι μετρητές για εκείνα τα σετ να υπερχειλίζουν πολλές φορές. Ο απλός DCCS αλγόριθμος δεν λαμβάνει υπόψη το πόσες φορές έχει συμβεί υπερχείλιση παρά μόνο αν έχει γίνει έστω μία φορά. Έτσι ένα νήμα του οποίου ο μετρητής για κάποιο σετ έχει υπερχειλίσει είτε μια φορά είτε περισσότερες λαμβάνει μονάδα στην εγγραφή του συγκεκριμένου σετ στο διάνυσμα συγκρούσεων. Αυτό ίσως να επηρεάζει την ακρίβεια του υπολογισμού του βαθμού συγκρούσεων για δύο νήματα και κατά συνέπεια την επιλογή της δρομολόγησης για την επόμενη περίοδο.

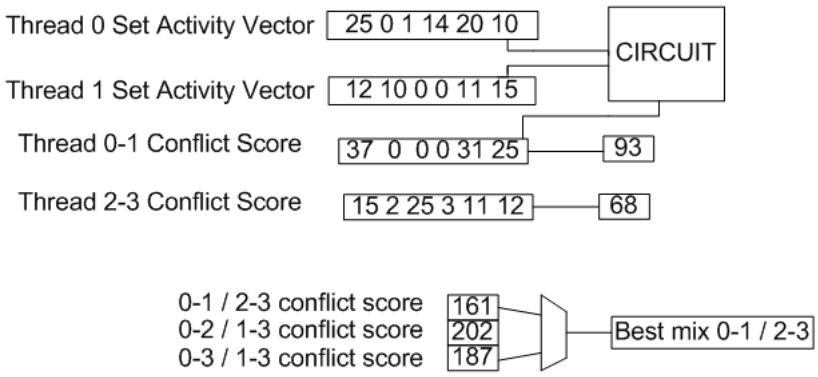
Για την αντιμετώπιση του πιο πάνω προβλήματος σχεδιάσαμε τον DCCS modified που κάνει χρήση μετρητών για το overflow bit. Τα αντίστοιχα διαγράμματα του DCCS modified για τα διανύσματα δραστηριότητας νημάτων και καθορισμού του καλύτερου συνδυασμού σύμφωνα με τον τελικό δείκτη σύγκρουσης του φαίνονται στα Σχήματα 5.14 και 5.15 αντίστοιχα.



Σχήμα 5.14: Ο συνδυασμός των διανυσμάτων δραστηριότητας για την δημιουργία του δείκτη σύγκρουσης και τον καθορισμό του καταλληλότερου συνδυασμού για τα τέσσερα νήματα στην επόμενη περίοδο, για την περίπτωση του DCCS modified.

Όπως φαίνεται στο Σχήμα 5.14 οι τιμές πλέον στο διάνυσμα δραστηριότητας δεν είναι 0 ή 1 αλλά ο αριθμός των υπερχειλίσεων που συνέβηκαν στον αντίστοιχο μετρητή του σετ κρυφής μνήμης. Έτσι έχουμε μια καλύτερη αίσθηση του ποσοστού χρήσης των σετ κρυφής μνήμης από τα νήματα από ότι στην περίπτωση του DCCS.
Πιο συγκεκριμένα στο Σχήμα 5.15 οι τιμές στο διάνυσμα συγκρούσεων δύο νημάτων πλέον προκύπτουν με χρήση ενδιάμεσου κυκλώματος το οποίο υλοποιεί την ακόλουθη λογική: Αν τουλάχιστον μια από τις δύο τιμές στις αντίστοιχες θέσεις των διανυσμάτων δραστηριότητας των δύο νημάτων είναι ίση με μηδέν, τότε η τιμή που προκύπτει για τον δείκτη σύγκρουσης των δύο νημάτων για εκείνο το σετ είναι μηδενική. Αντίθετα, αν και οι δύο τιμές είναι διάφορες του μηδενός, ο δείκτης σύγκρουσης των δύο νημάτων για το σετ είναι ίσος με το άθροισμά τους. Με παρόμοιο τρόπο όπως στον DCCS δηλαδή προσθέτοντας όλες τις εγγραφές στο διάνυσμα σύγκρουσης, παίρνουμε τον τελικό δείκτη σύγκρουσης του κάθε ζεύγους νημάτων. Στην συνέχεια επιλέγεται ο συνδυασμός των νημάτων στον οποίο ο συνολικός τελικός δείκτης είναι ο μικρότερος.

Στο παράδειγμα του Σχήματος 5.15 φαίνονται τα διανύσματα δραστηριότητας για το νήμα 0 και 1. Το προκύπτον διάνυσμα συγκρούσεων των νημάτων 0 και 1 παρουσιάζει άθροισμα εγγραφών δηλαδή δείκτη σύγκρουσης ίσο με 93. Η αντίστοιχη διαδικασία έχει γίνει και για τα νήματα 2 και 3 υπολογίζοντας δείκτη σύγκρουσης 68. Αφού βρεθεί ο δείκτης σύγκρουσης για όλα τα ζεύγη νημάτων τότε παίρνουμε το μικρότερο συνολικό δείκτη σύγκρουσης όλων των δυνατών συνδυασμών (στο παράδειγμα τρεις). Σε αυτή την περίπτωση είναι ίσο με 161 όποτε και επιλέγεται η συνδρομολόγηση για

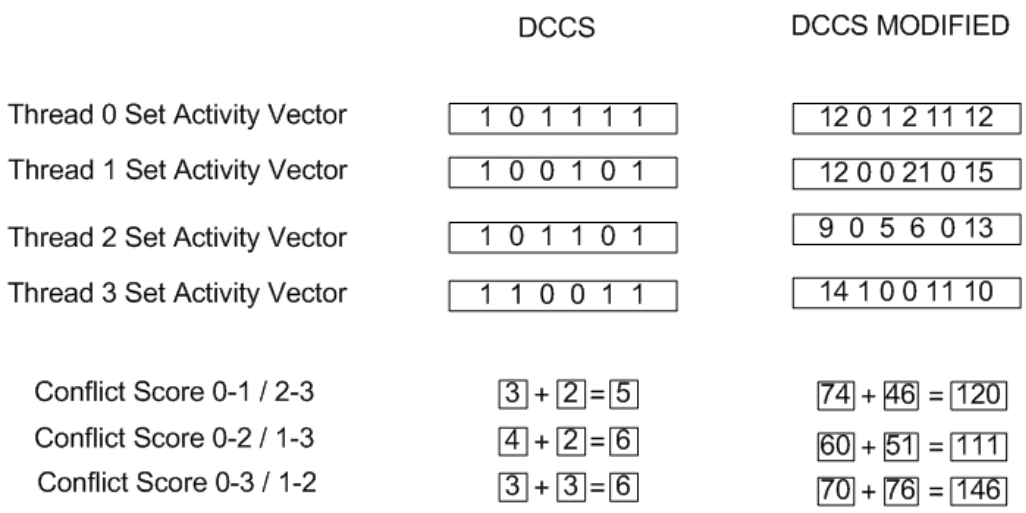


Σχήμα 5.15: Ο συνδυασμός των διανυσμάτων δραστηριότητας για την δημιουργία του δείκτη σύγκρουσης και τον καθορισμό του καταλληλότερου συνδυασμού για τα τέσσερα νήματα στην επόμενη περίοδο, για την περίπτωση του DCCS modified.

την επόμενη περίοδο των νημάτων 0-1 και 2-3 αντίστοιχα.

5.3.1 Παράδειγμα σύγκρισης DCCS και DCCS modified

Στο Σχήμα 5.16 παρουσιάζεται ένα παράδειγμα που φαίνεται μία περίπτωση που η υλοποίηση του DCCS υστερεί έναντι αυτής του DCCS modified.



Σχήμα 5.16: Παράδειγμα σύγκρισης DCCS και DCCS modified.

Στο σχήμα φαίνονται τα διανύσματα δραστηριότητας και για τα τέσσερα νήματα του συστήματος μας, όπως επίσης και οι τελικοί δείκτες σύγκρουσης για κάθε ζεύγος αλλά και οι προκύπτοντες τελικοί συνδυασμοί των νημάτων. Στο παράδειγμα παρατηρούμε ότι σύμφωνα με τον DCCS το ζεύγος νημάτων 0-1 παρουσιάζει δείκτη σύγκρουσης ίσο με 3 σε αντίθεση με το 0-2 του οποίου ο δείκτης σύγκρουσης είναι μεγαλύτερος και ίσος με 4. Ο DCCS modified ο οποίος εφόσον έχει ως δείκτη του ποσοστού πρόσβασης των νημάτων στα σετ κρυφής μνήμης την συχνότητα των υπερχειλίσεων στους αντίστοιχους μετρητές, υπολογίζει ακριβέστερα τον δείκτη συγκρούσεων. Σε αυτή την περίπτωση ο

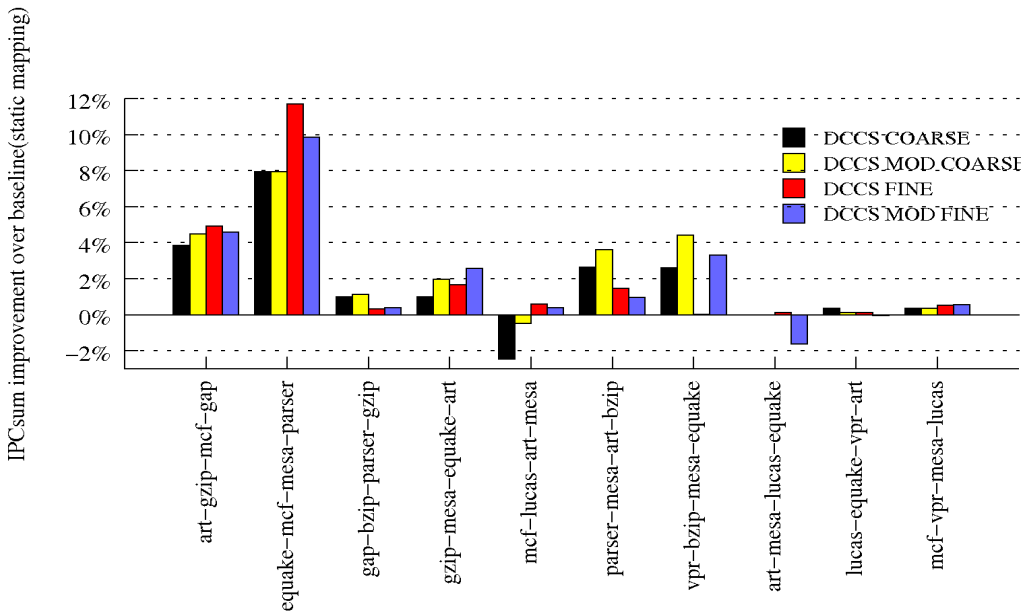
πραγματικός δείκτης σύγκρουσης (βάσει τον μετρητών) για το ζεύγος 0-1 είναι ίσος με 74 ενώ για το ζεύγος 0-2 ίσος με 60. Αυτή η συμπεριφορά του DCCS όπου ο προκύπτον τελικός δείκτης του ζεύγους 0-1 είναι μικρότερος από αυτόν του 0-2, οδήγησε τελικά στην επιλογή του συνδυασμού 0-1/2-3 που στην πραγματικότητα είναι λανθασμένη. Αντίθετα ο DCCS modified υπολογίζοντας τους ακριβείς δείκτες σύγκρουσης ζεύγους και τελικούς θα επιλέξει τον 0-2/1-3 ο οποίος είναι σύμφωνα με τους μετρητές ο καλύτερος.

5.3.2 Αποτελέσματα για τον DCCS modified

Σε αυτή την υποενότητα θα γίνει η ανάλυση των αποτελεσμάτων για τον DCCS modified, ίδια με αυτή που έγινε πιο πριν για τους DCCS και IPCS. Πιο συγκεκριμένα τα αποτελέσματα θα συγκριθούν με αυτά του DCCS.

Βελτίωση στο IPC$_{sum}$ - Βελτίωση επίδοσης συστήματος

Στο Σχήμα 5.17 φαίνεται η βελτίωση του IPC$_{sum}$ σε σχέση με την baseline για τον DCCS και DCCS modified.



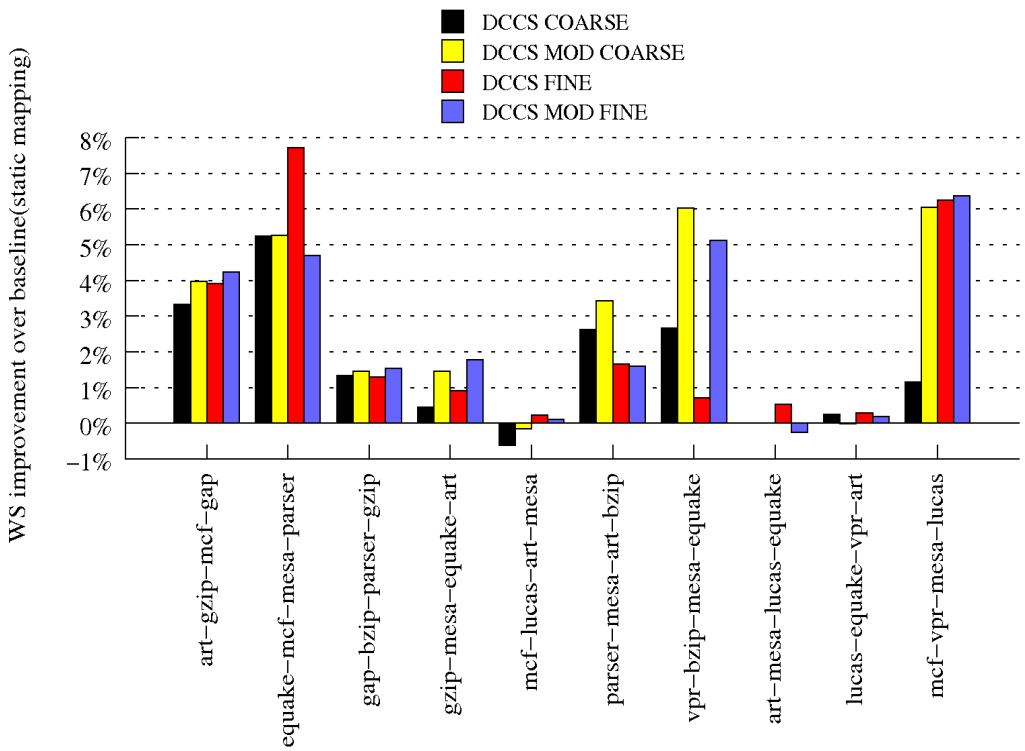
Σχήμα 5.17: Βελτίωση στην μετρική IPCsum σε σχέση με την baseline περίπτωση.

Ο DCCS modified παρουσιάζει παρόμοια συμπεριφορά με τον DCCS αφού η λογική με την οποία συνδρομολογεί τα νήματα είναι η ίδια, με μια μικρή διαφοροποίηση την οποία περιγράψαμε πιο πριν. Παρ' όλα αυτά για την coarse-grained πολιτική στους περισσότερους συνδυασμούς (συνδυασμοί 1,2,3,4,5,6,7) ο DCCS modified παρουσιάζει καλύτερη επίδοση σε σχέση με τον DCCS. Αυτό οφείλεται στο γεγονός ότι σε μεγαλύτερα διαστήματα κάποιοι μετρητές προσβάσεων υπερχειλίζουν πιο εύκολα δημιουργώντας στον DCCS το πρόβλημα που αναφέρθηκε προηγουμένως. Αντίθετα, ο DCCS modified υπολογίζοντας τον ακριβή βαθμό σύγκρουσης για κάθε συνδυασμό, μπορεί πια να προβεί σε ορθότερες αποφάσεις δρομολόγησης νημάτων. Αυτό δεν συμβαίνει στην περίπτωση της fine-grained πολιτικής όπου σε κάποιους συνδυασμούς (συνδυασμός 1,2,5,6,8) ο DCCS εμφανίζει καλύτερη επίδοση ενώ σε κάποιους άλλους (συνδυασμοί 3,4,7,10) ο DCCS modified. Για πιο μικρά διαστήματα οι υπερχειλίσεις στους μετρητές συμβαίνουν πιο δύσκολα και γι'αυτό η λεπτομέρεια που

εμφανίζει ο DCCS modified ίσως να μην χρειάζεται. Τέλος η καλύτερη επίδοση στην IPC_{sum} μετρική εμφανίστηκε στον Συνδυασμό 2 με πολιτική δρομολόγησης DCCS fine-grained.

Weighted Speedup - Βελτίωση στην μεμονωμένη επίδοση κάθε νήματος.

Το Σχήμα 5.18 παρουσιάζει τη βελτίωση στην επίδοση σύμφωνα με την WS μετρική για τους DCCS και DCCS modified.

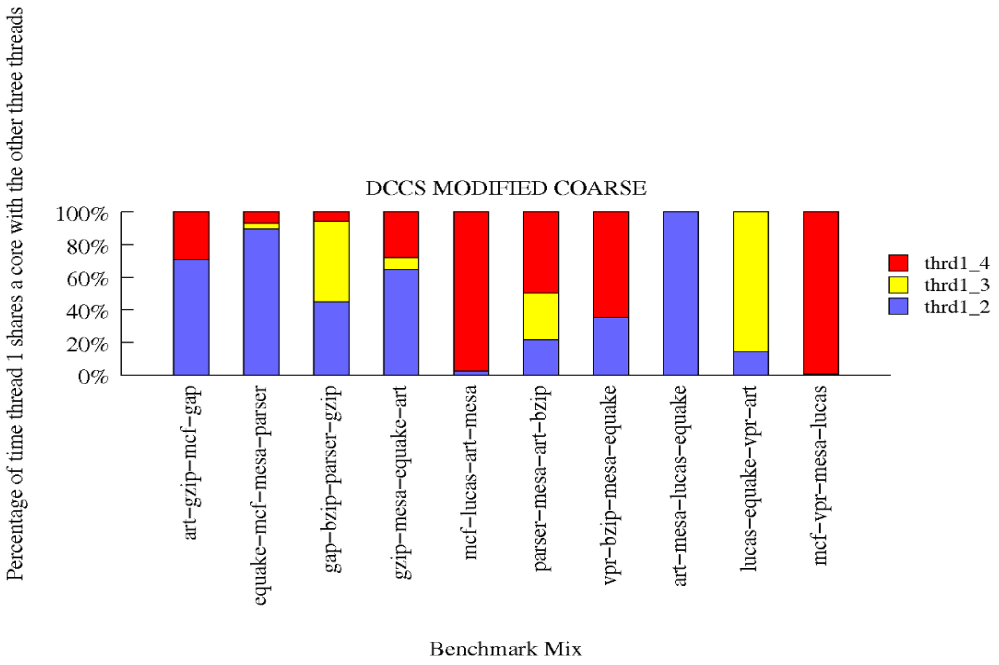


Σχήμα 5.18: Βελτίωση στην μετρική WS σε σχέση με την baseline περίπτωση.

Αντίστοιχα είναι τα αποτελέσματα και στην WS μετρική. Στην coarse-grained περίπτωση ο DCCS modified εμφανίζει υψηλότερες επιδόσεις σε αρκετές περιπτώσεις σε σχέση με τον DCCS (Συνδυασμοί 1,2,3,4,5,6,7,10) με χαρακτηριστικότερες αυτές στους Συνδυασμούς 7 και 10. Η αιτιολογία είναι η ίδια με αυτήν που δόθηκε προηγουμένως για την IPC_{sum} μετρική δηλαδή στο ότι για μεγαλύτερα διαστήματα δρομολόγησης (coarse-grained) λόγω του μεγάλου αριθμού προσβάσεων στην κρυφή μνήμη δεδομένων L1 είναι πιο εύκολο να εμφανιστούν αρκετές υπερχειλίσεις μετρητών για κάποια σετ, που μπορεί να οδηγήσουν σε τον DCCS σε λανθασμένες επιλογές. Αυτό δικαιολογείται από το γεγονός ότι στην fine-grained περίπτωση δεν συμβαίνουν συχνά πολλές υπερχειλίσεις μετρητών και έτσι η καλύτερη βελτίωση σε κάθε συνδυασμό δεν ανήκει αποκλειστικά στον DCCS modified. Έτσι σε κάποιες περιπτώσεις (Συνδυασμοί 1,5,6,8,9) ο DCCS συμπεριφέρεται καλύτερα ενώ σε κάποιες άλλες έχουμε μεγαλύτερη βελτίωση στον DCCS modified. Και πάλι η μεγαλύτερη βελτίωση παρατηρείται στον Συνδυασμό 2 και ανήκει στον DCCS fine-grained.

Ποσοστά συνδρομολόγησης νημάτων για τον DCCS modified

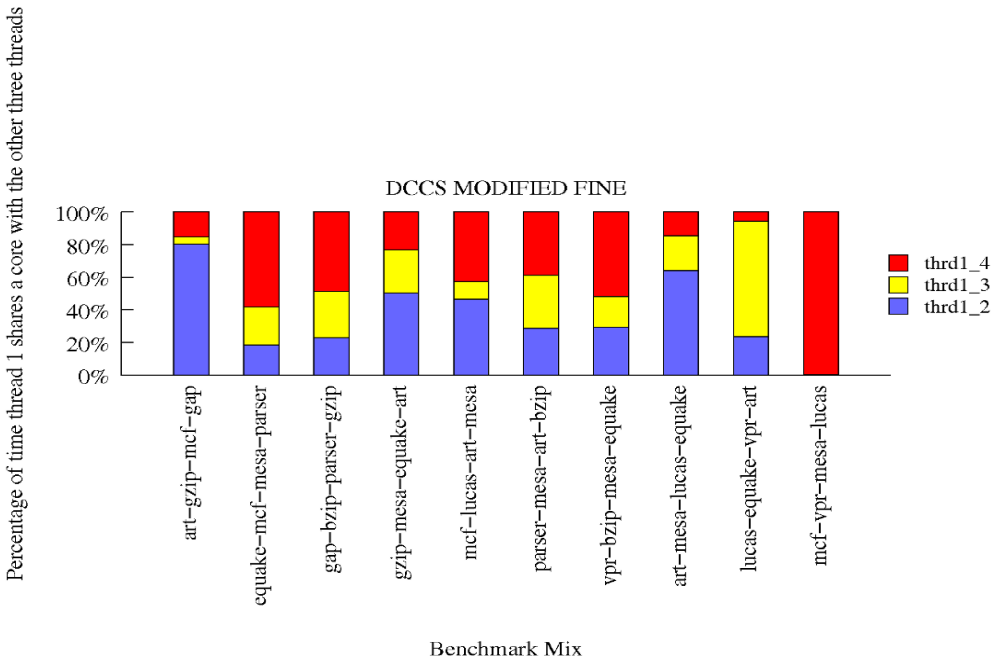
Σε αυτή την υποενότητα παρουσιάζονται τα ποσοστά της συνδρομολόγησης στην περίπτωση του αλγόριθμου DCCS modified για coarse-grained και fine-grained διαστήματα.
Όπως φαίνεται υπάρχουν κάποια ελαφρώς διαφοροποιημένα ποσοστά για κάποιους συνδυασμούς σε σχέση με τον DCCS, τα οποία δεν επιφέρουν τεράστια βελτίωση σε σχέση με την απλή εκδοχή του αλγόριθμου αφού ουσιαστικά στον modified η βελτίωση έγκειται στο ότι κάποιες φορές επιλέγεται η



Σχήμα 5.19: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του DCCS modified coarse grained.

λιγότερο χειρότερη περίπτωση. Και σε αυτό τον αλγόριθμο στην fine-grained περίπτωση εμφανίζονται κάποια ποσοστά συνδρομολόγησης που όπως είναι φυσικό η coarse-grained δεν βλέπει λόγω του μεγάλου χρονικού διαστήματος μέχρι την επόμενη δρομολόγηση. Αυτά τα ποσοστά όμως επιφέρουν βελτίωση στην επίδοση σε σχέση με το coarse grained και στις δύο μετρικές IPC_{sum} και WS_{sum}.

Αξιοσημείωτες είναι οι περιπτώσεις των Συνδυασμών 5 και 10 στην coarse-grained προσέγγιση. Οι συνδυασμοί που παρατηρούνται στην περίπτωση του DCCS και DCCS modified είναι εντελώς διαφορετικοί. Στον Συνδυασμό 5 που μάλλον είναι μια κατάσταση όπου υπάρχει αρκετή κοινή χρήση σετ κρυφής μνήμης μεταξύ όλων των νημάτων, αφοπλίζεται κυριολεκτικά ο DCCS. Οποιοσδήποτε συνδυασμός των νημάτων στους πυρήνες θα επιφέρει αρκετό συνωστισμό και πολλά misses. Όμως ο DCCS modified που καταφέρνει να υπολογίζει αυτό τον βαθμό του συνωστισμού όπως περιγράφηκε προηγουμένως επιλέγει να συνδρομολογήσει τον λιγότερο χειρότερο συνδυασμό σε αντίθεση με τον DCCS. Σαν αποτέλεσμα η μείωση στην επίδοση του DCCS coarse-grained είναι αρκετά μεγαλύτερη από ότι στον DCCS modified coarse-grained. Εδώ πρέπει να σταθούμε και να πούμε ότι σε αυτές τις δύσκολες περιπτώσεις είναι που φαίνεται η σημασία του DCCS modified και η κύρια διαφορά των δύο αλγορίθμων. Αξιοσημείωτη είναι συμπεριφορά στον Συνδυασμό 10 όπου τα ποσοστά συνδρομολόγησης είναι σχεδόν τα ίδια σε DCCS modified coarse-grained και fine-grained αλλά και DCCS fine-grained. Ο DCCS coarse-grained από την άλλη παρουσιάζει εντελώς διαφορετικό συνδυασμό συνδρομολογήσεων όπως αναφέρθηκε σε προηγούμενη υποενότητα. Έτσι παρόλο που στην IPC_{sum} μετρική οι τέσσερις πολιτικές παρουσιάζουν παρόμοια επίδοση κάτι τέτοιο δεν ισχύει για την WS. Σε αυτή την μετρική οι τρεις πρώτες εμφανίζουν αρκετά μεγαλύτερη βελτίωση σε σχέση με την τελευταία. Αυτή η συμπεριφορά του DCCS coarse-grained μας οδηγεί στην υποψία ότι ο Συνδυασμός 10 είναι ένας συνδυασμός που στην coarse-grained περίπτωση εμφανίζει συχνότερες υπερχειλίσεις στους μετρητές προσβάσεων στα σετ κρυφής μνήμης. Έτσι εφόσον στον DCCS η συχνότητα υπερχειλίσεων δεν λαμβάνεται υπόψη οδηγούμαστε σε λανθασμένες συνδρομολογήσεις οι οποίες μπορεί



Σχήμα 5.20: Ποσοστό του χρόνου που το νήμα 1 μοιράζεται τον πυρήνα με τα υπόλοιπα τρία νήματα στην περίπτωση του DCCS modified fine grained.

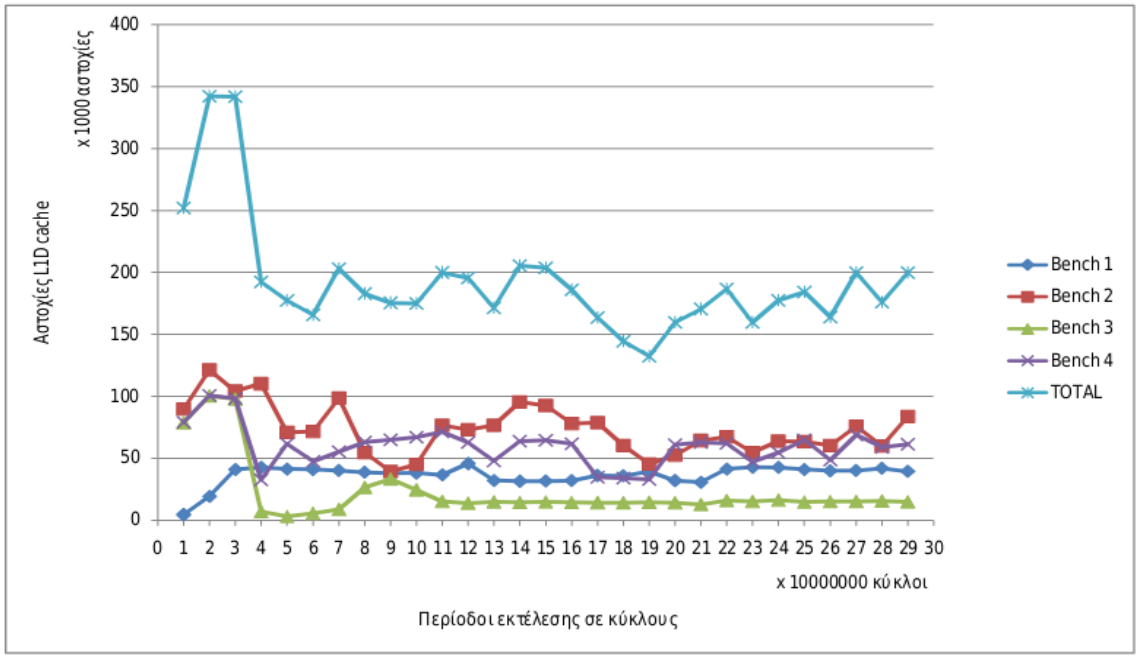
να μην επηρεάζουν σε μεγάλο βαθμό την IPC_{sum} μετρική πράγμα όμως που δεν ισχύει για την WS.

Μελέτη αστοχιών - συνδρομολογήσεων στην περίπτωση του equake-mcf-mesa-parser στον DCCS modified

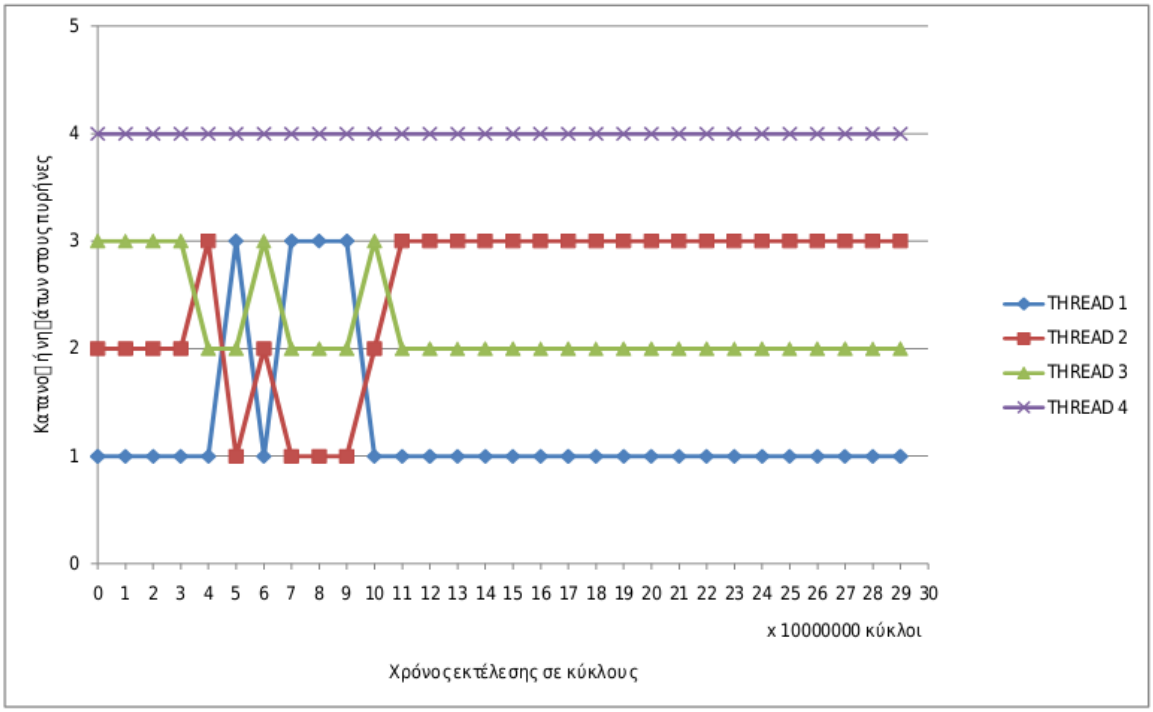
Τέλος στα Σχήματα 5.21 και 5.22 παρουσιάζονται τα αντίστοιχα διαγράμματα για την περίπτωση του equake-mcf-mesa-parser στον DCCS modified ενώ στο Σχήμα 5.23 γίνεται άμεση σύγκριση των αστοχιών ανά περίοδο για τις δύο πολιτικές δρομολόγησης.

Η συμπεριφορά που εμφανίζεται σε σχέση με την baseline περίπτωση είναι σχεδόν η ίδια με αυτή στον DCCS. Ουσιαστικά ο DCCS modified καταφέρνει να επιφέρει ελαφρώς καλύτερη βελτίωση σε σχέση με τον απλό DCCS. Όπως και στον DCCS, ο DCCS modified με την κατάλληλη συνδρομολόγηση νημάτων καταφέρνει να μειώσει τις αστοχίες των νημάτων στην L1D κρυφή μνήμη κατά τη διάρκεια της εκτέλεσης σε αντίθεση με τις αρχικές περιόδους όπου ο αλγόριθμος ξεκινά να εφαρμόζεται και οι αστοχίες είναι αυξημένες. Συνεπώς έχουμε λιγότερες συνολικές αστοχίες, γεγονός που μειώνει την καθυστέρηση στο σύστημά μας και κατά συνέπεια παρουσιάζει βελτίωση όσον αφορά την IPC_{sum} μετρική. Στο μεγαλύτερο διάστημά του ο αλγόριθμος προτιμά την συνδρομολόγηση των νημάτων 1 και 4 στον ίδιο πυρήνα και αντίστοιχα 2 και 3 όπως ακριβώς και στην απλή έκδοση DCCS με περίπου ίδια ποσοστά.

Στην περίπτωση της μεμονωμένης απόδοσης και στην περίπτωση του DCCS modified παρατηρείται βελτίωση όπως εξάλλου φανερώνει και το Σχήμα 5.18 και μάλιστα ελαφρώς καλύτερη από την απλή DCCS περίπτωση. Η βελτίωση και στις δύο μετρικές μπορεί να φανεί καλύτερα στο Σχήμα 5.23 όπου σε κάποιες περιόδους (160 M - 170 M , 170 M -180 M, 250 M - 260 M) οι συνολικές αστοχίες των νημάτων αλλά και κάποιων μεμονωμένων (στο διάστημα 160 M -170 M κύκλους για το νήμα 1, στο διάστημα 170 M - 180 M για το νήμα 2 ενώ στο διάστημα 250 M -260 M πάλι για το νήμα 2) είναι

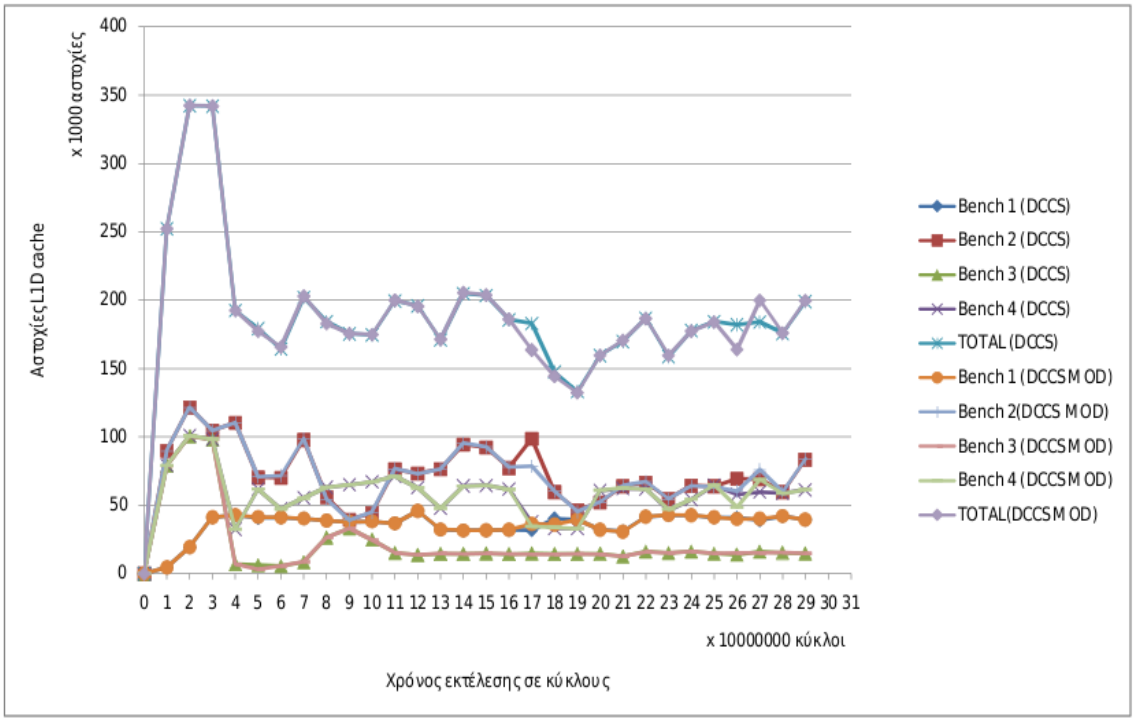


Σχήμα 5.21: Οι αστοχίες κρυφής μνήμης L1D κάθε νήματος και οι συνολικές σε κάθε περίοδο στον DCCS modified.



Σχήμα 5.22: Η κατανομή των νημάτων στους πυρήνες σε κάθε περίοδο στον DCCS modified.

λιγότερες στον DCCS modified σε σχέση με τον DCCS. Παρ'όλα αυτά σε μια τέτοια περίπτωση όπου



Σχήμα 5.23: Άμεση σύγκριση των αστοχιών L1D κρυφής μνήμης για DCCS και DCCS modified coarse-grained.

DCCS και DCCS modified επιφέρουν παρόμοια βελτίωση ίσως η αξία της modified έκδοσης να μην φαίνεται ξεκάθαρα. Στις ακραίες περιπτώσεις όπως αναφέρθηκε προηγουμένως, όπου όλα τα νήματα έχουν πρόσβαση στον ίδιο βαθμό σε κοινά σετ κρυφής μνήμης, ο DCCS modified συμπεριφέρεται ακόμη καλύτερα από τον DCCS.

Κεφάλαιο 6

Συμπεράσματα - Μελλοντική Εργασία

6.1 Γενικά

Η χρήση των CMP αποτελούμενων από πολυνηματικούς πυρήνες είναι η πιο δημοφιλής προσέγγιση σχεδίασης επεξεργαστών σήμερα και αναμένεται να επικρατήσει στην αγορά και για τα επόμενα χρόνια. Οι τελευταίοι επεξεργαστές στην αγορά κυρίως για χρήση ως server δημιουργήθηκαν σύμφωνα με αυτή την τεχνολογία. Φυσικά υπάρχουν και περιπτώσεις τέτοιων επεξεργαστών που αφορούν στην οικιακή χρήση αλλά ακόμα δεν βρίσκονται στο απόγειο της χρήσης τους για αυτό το σκοπό.

Η χρήση μιας τέτοιας αρχιτεκτονικής επιβάλλει την αντιμετώπιση αρκετών προκλήσεων. Τα κλασικά προγράμματα με το χαμηλό ποσοστό παραλληλοποίησης δύσκολα παρουσιάζουν βελτίωση σε αυτές τις αρχιτεκτονικές που σχεδιάστηκαν με κεντρικό άξονα την εκμετάλλευση της παραλληλίας. Η σχεδίαση παράλληλων προγραμμάτων (π.χ. με χρήση πολυνηματικού προγραμματισμού) είναι πια επιβεβλημένη. Όσον αφορά στο σχεδιαστικό κομμάτι,τεχνολογίες και διαμορφώσεις που χρησιμοποιούνταν στις αρχιτεκτονικές ενός πυρήνα, πλέον δεν έχουν θέση πια εδώ. Αντίθετα υπάρχει ανάγκη για σχεδίαση εξαρχής κάποιων στοιχείων και όχι απλή προσαρμογή τους στις σύγχρονες αρχιτεκτονικές. Επιπλέον καινούργιες έννοιες όπως π.χ. η συνάφεια κρυφής μνήμης, κάνουν αισθητή την παρουσία τους εμφανίζοντας περισσότερες δυσκολίες στη σχεδίαση. Τέλος στις πολυπύρηνες-πολυνηματικές αρχιτεκτονικές οι περισσότεροι πόροι μοιράζονται ανάμεσα σε πυρήνες και σε τελική ανάλυση στα νήματα. Χρειάζεται άρα περαιτέρω έρευνα για την ανακάλυψη πολιτικών και μηχανισμών που θα μεγιστοποιήσουν την ολική απόδοση αλλά που θα υπάρχει ισότητα νημάτων. Δηλαδή δεν θα εμφανίζεται το φαινόμενο "λιμοκτονίας" κάποιων νημάτων για χάρη της ολικής απόδοσης.

Αυτά τα δεδομένα επιβάλλουν την εφαρμογή οποιασδήποτε μεθόδου η οποία να εισάγει βελτίωση σε τέτοιες αρχιτεκτονικές. Η εύρεση των "καλύτερων γειτόνων" δηλαδή το πρόβλημα της συνδρομολόγησης των νημάτων στους επεξεργαστές - πυρήνες είναι επιβεβλημένη. Όπως είδαμε και από τα αποτελέσματα τις έρευνάς μας, υπήρξε αλγόριθμος δρομολόγησης που σε ένα σύστημα δύο πυρήνων, συνολικά τεσσάρων νημάτων κατάφερε να εισάγει βελτίωση συστήματος περίπου 12% απ'ότι στην στατική περίπτωση, για κάποιο συνδυασμό μετροπρογραμμάτων. Μια βελτίωση τέτοιου μεγέθους είναι κρισιμότατη και δεν μπορεί να περάσει απαρατήρητη. Ειδικά αν αναλογιστούμε ότι σε πιο μεγάλα συστήματα π.χ δύο πυρήνων - οκτώ νημάτων ή τεσσάρων πυρήνων - οκτώ νημάτων, η επιλογή των κατάλληλων συνδυασμών στους πυρήνες είναι ακόμα πιο σημαντική. Η στατική περίπτωση σε ένα τέτοιο σύστημα με περισσότερα νήματα μπορεί να εισάγει περισσότερες καθυστερήσεις και συγκρούσεις σε περίπτωση λανθασμένης συνδρομολόγησης νημάτων, σε αντίθεση με κάποιο αλγόριθμο ο οποίος μπορεί να συνδρομολογήσει τα νήματα με καλύτερο τρόπο. Άρα τα ποσοστά βελτίωσης που αναμένονται σε μεγαλύτερα συστήματα είναι ακόμη μεγαλύτερα από την περίπτωση που εξετάσαμε.

Οι επεξεργαστές αυτής της τεχνολογίας, που κυκλοφορούν σήμερα, δεν υποστηρίζουν κάποιο συγκεκριμένο αλγόριθμο δρομολόγησης σε επίπεδο υλικού. Αντίθετα οι μοναδικοί αλγόριθμοι που χρησιμοποιούνται είναι σε επίπεδο λειτουργικού όπως στην περίπτωση του Sun Niagara και του λειτουργικού Solaris. Φυσικά όμως η προσέγγιση της δρομολόγησης σε επίπεδο υλικού εισάγει καλύτερες βελτιώσεις αφού επιτρέπει την online χρήση μετρητών στο ίδιο το υλικό, στα σημεία που μπορεί να

προκληθούν μεγάλες καθυστερήσεις (π.χ. κρυφή μνήμη),σημεία που πιθανόν να βρίσκονται σε πολύ χαμηλό επίπεδο και άρα να είναι δύσβατα για το λειτουργικό. Η δρομολόγηση επίσης σε επίπεδο υλικού είναι αμεσότερη αφού γίνεται από τον ίδιο τον επεξεργαστή σε αντίθεση με την δρομολόγηση σε επίπεδο λειτουργικού όπου μεσολαβεί το λειτουργικό. Ίσως ένας συνδυασμός των δύο μεθόδων να επιφέρει καλύτερα αποτελέσματα από την χρήση μόνο του ενός, αλλά όσον αφορά στη χρήση ενός εκ των δύο,η περίπτωση της δρομολόγησης οδηγούμενης από το hardware φαίνεται να υπερέχει.

6.2 Συμπεράσματα

Σ' αυτή τη διπλωματική εργασία μελετήσαμε τρεις ουσιαστικά πολιτικές συνδρομολόγησης νημάτων. Την περίπτωση του DCCS (Data Cache Conflict Scheduler) ο οποίος στηρίζεται στην καταμέτρηση των προσβάσεων στα σετ της κρυφής μνήμης L1D από κάθε νήμα και ακολούθως συνδρομολογεί μαζί στον ίδιο πυρήνα τα νήματα με τις προσβάσεις στα λιγότερο κοινά σετ. Με αυτό τον τρόπο επιτυγχάνεται ο έλεγχος του συνωστισμού της κρυφής μνήμης L1D σε κάθε πυρήνα που μπορεί να αποτελέσει σημαντικό παράγοντα εισαγωγής καθυστέρησης. Ακολούθως έγινε μελέτη του IPCS (Instruction Per Cycle Scheduler) ο οποίος συνδρομολογεί σε μια περίοδο στον ίδιο πυρήνα τα νήματα τα οποία παρουσίασαν το υψηλότερο IPC την προηγούμενη περίοδο και αντίστοιχα μαζί σε άλλο εκείνα με το μικρότερο. Με αυτό τον τρόπο προσπαθεί να εμποδίσει τα πιο αργά νήματα από το να καθυστερούν τα πιο γρήγορα λόγω του ότι καταλαμβάνουν για πολύ χρόνο πόρους. Τέλος μελετήσαμε την περίπτωση της βελτιωμένης εκδοχής του αλγόριθμου DCCS (DCCS modified), την οποία και προτείνουμε. Η βασική της διαφορά έγκειται στο γεγονός ότι λαμβάνει υπόψη τον αριθμό των υπερχειλίσεων που συμβαίνουν στους μετρητές προσβάσεων στα σετ της κρυφής μνήμης. Έτσι για δύο νήματα, από τα οποία στο ένα ο μετρητής για κάποιο σετ υπερχείλισε 100 φορές ενώ στο άλλο 50, ο βαθμός σύγκρουσης του ζεύγους νημάτων θα είναι πλέον 150 αντί 1 όπως συνέβαινε στην απλή περίπτωση του DCCS. Έτσι επιλέγονται με μεγαλύτερη λεπτομέρεια και άρα αποδοτικότερα οι ομάδες των νημάτων σε κάθε πυρήνα για την επόμενη περίοδο.

Τα αποτελέσματα των πειραμάτων μας φανερώνουν μια καλύτερη επίδοση των αλγόριθμων της οικογένειας DCCS αφού ο κυρίαρχος παράγοντας εισαγωγής καθυστέρησης στο σύστημά μας είναι οι αστοχίες κρυφής μνήμης. Είναι λογικό λοιπόν αυτοί οι αλγόριθμοι των οποίων η φιλοσοφία έχει να κάνει με το συνωστισμό της κρυφής μνήμης, να εμφανίζουν καλύτερα αποτελέσματα. Ειδικά σε ένα εξομοιωτή όπως αυτός που διαμορφώσαμε και χρησιμοποιήσαμε, όπου ο κύριος παράγοντας εισαγωγής καθυστέρησης και διαμόρφωσης της επίδοσης είναι το υποσύστημα μνήμης τότε ήταν λογικό να αναμένονται τέτοιες βελτιώσεις από αυτές τις πολιτικές δρομολόγησης.

Αντίθετα ο IPCS επειδή δεν ελέγχει άμεσα τις προσβάσεις κάθε νήματος στην κρυφή μνήμη παρά μόνο το IPC τους σε κάθε περίοδο μπορεί να οδηγήσει σε συνδρομολογήσεις με μεγάλα ποσοστά αστοχίας κρυφής μνήμης λόγω της συνύπαρξης των νημάτων. Το IPC από μόνο του δεν μπορεί να αποτελέσει ικανοποιητικό κριτήριο συνδρομολόγησης νημάτων σε πυρήνα. Και αυτό γιατί μπορεί να έχουμε μεν τα πιο γρήγορα νήματα να τρέχουν μαζί αλλά δεν μπορούμε να ξέρουμε το ποσοστό κοινής πρόσβασης στην L1D κρυφή μνήμη και άρα την εμφάνιση αλλεπάλληλων συγκρούσεων και αστοχιών εκεί, με υψηλό κόστος στην επίδοση τους αλλά και αυτής του συστήματος. Κατά συνέπεια ο IPCS παρουσίασε αρκετά μικρότερα ποσοστά βελτίωσης σε αρκετούς συνδυασμούς μετροπρογραμμάτων που μελετήσαμε και σε κάποιες περιπτώσεις μάλιστα εμφάνισε χειρότερα αποτελέσματα και από την στατική δρομολόγηση.

Όλη αυτή η συμπεριφορά του IPCS ίσως να εξηγείται από το ότι αρχικά προτάθηκε στις SMT αρχιτεκτονικές ενός πυρήνα. Ουσιαστικά ο IPCS έπρεπε να αποφασίσει ποια νήματα έπρεπε να τρέχουν μαζί σε κάποια χρονική στιγμή πάνω στον μοναδικό SMT πυρήνα. Άρα το πρόβλημα που είχε να αντιμετωπίσει είχε περισσότερο χρονικό χαρακτήρα. Αντίθετα στην δική μας περίπτωση, το πρόβλημα έχει να κάνει περισσότερο με το που πρέπει να συνδρομολογηθούν και ποια νήματα, δηλαδή υπάρχει πια

και ο περιορισμός σε ποιον πυρήνα. Αυτός ο περιορισμός όμως εισάγει νέες παραμέτρους που πρέπει να μελετηθούν ώστε να παρθεί η απόφαση για μια συνδρομολόγηση, παραμέτρους που μια μεταφορά του IPCS από τις SMT μονοπύρηνες αρχιτεκτονικές στις πολυπύρηνες πολυνηματικές αρχιτεκτονικές πιθανόν να μην τις λαμβάνει υπόψη. Ένα παράδειγμα μιας τέτοιας παραμέτρου είναι ακριβώς αυτό που αντιμετωπίζει ο DCCS δηλαδή το πόσο φιλικά είναι τα νήματα που δρομολογούνται στον ίδιο πυρήνα όσον αφορά την χρήση της κρυφής μνήμης.

Όσον αφορά DCCS modified η αξία του φάνηκε κυρίως στις περιπτώσεις που ο DCCS αντιμετώπιζε πρόβλημα και άρα τα ποσοστά βελτίωσης που εισήγαγε και στις δύο μετρικές που μελετήθηκαν ήταν αρκετά χαμηλά ή ακόμα και αρνητικά. Ένα παράδειγμα τέτοιας περίπτωσης είναι όταν όλα τα νήματα του συνδυασμού παρουσιάζουν υψηλά ποσοστά κοινής χρήσης σετ κρυφής μνήμης μεταξύ τους. Μια τέτοια περίπτωση χρειάζεται περαιτέρω λεπτομέρεια στην καταγραφή των προσβάσεων στα σετ κρυφής μνήμης, μια λεπτομέρεια που μόνο η modified έκδοση μπορεί να προσφέρει καταμετρώντας τον αριθμό των υπερχειλίσεων που συμβαίνουν σε ένα μετρητή σετ και τελικά υπολογίζοντας τον ακριβή δείκτη σύγκρουσης μιας ομάδας νημάτων. Έτσι σε νοσηρές περιπτώσεις όπου ο DCCS δεν μπορεί λόγω του συνδυασμού των μετροπρογραμμάτων να προσφέρει σημαντική βελτίωση, χρήση του DCCS modified παρουσίασε καλύτερες επιδόσεις.

Σχετικά τέλος με το διάστημα δρομολόγησης, καλύτερα αποτελέσματα για τους αλγόριθμους της οικογένειας DCCS παρατηρήθηκαν στην fine-grained πολιτική. Αυτό είναι λογικό επακόλουθο του γεγονότος ότι σε μια τέτοια πολιτική τα στατιστικά που συμβάλλουν στην τελική απόφαση συνδρομολόγησης των νημάτων σε μια περίοδο συλλέγονται συχνότερα απ'ότι στην coarse-grained περίπτωση. Έτσι μπορούμε να οδηγηθούμε σε καλύτερα συμπεράσματα για την συμπεριφορά των νημάτων από περίοδο σε περίοδο.

Ενώ ο DCCS παρακολουθεί μόνο την συμπεριφορά των νημάτων σε σχέση με την κρυφή μνήμη δεδομένων L1 για το πάρσιμο αποφάσεων συνδρομολόγησης,στον IPCS παρακολουθείται η συμπεριφορά του IPC κάθε νήματος η οποία δεν εξαρτάται μόνο από ένα άλλα από περισσότερους παράγοντες όπως είναι ο συνωστισμός στο pipeline κάθε πυρήνα, οι αστοχίες στις κρυφές μνήμες L1 και L2 κτλ. καθιστώντας την πιο ευαίσθητη σε μεγάλες μεταβολές της συμπεριφοράς των νημάτων σε αυτούς τους παράγοντες από περίοδο σε περίοδο.

Στον IPCS τα αποτελέσματα της έρευνας που αφορούν στο ποια πολιτική από τις δύο είναι καλύτερη έδειξαν ότι αυτό κάθε φορά εξαρτάται από τον συνδυασμό των νημάτων. Στους συνδυασμούς όπου τα νήματα στην fine-grained περίπτωση παρουσιάζουν απότομες μεταβολές στο IPC από περίοδο σε περίοδο, οι βελτιώσεις που παρουσιάζονταν ήταν χειρότερες από τις αντίστοιχες στην coarse-grained. Αυτό εξηγείται από το γεγονός ότι παρουσιάζονταν ενδιάμεσες συνδρομολογήσεις όπου φαινομενικά ήταν σωστές όμως η συμπεριφορά του IPC των νημάτων στην επόμενη περίοδο τις αποδείκνυε λανθασμένες. Αυτό δεν συμβαίνει με την coarse grained πολιτική όπου λόγω των μεγάλων διαστημάτων δρομολόγησης δεν ήταν τόσο ευαίσθητη σε τέτοιες πιο μικρού διαστήματος μεταβολές οπότε και η τελική απόφαση συνδρομολόγησης για την επόμενη coarse-grained περίοδο δεν επηρεαζόταν. Σε αντίθετη περίπτωση που τα IPC των νημάτων δεν παρουσίαζαν απότομες μεταβολές στο IPC τους από περίοδο σε περίοδο, δηλαδή το IPC της επόμενης περιόδου δρομολόγησης ήταν σχετικά προβλέψιμο τότε η IPCS fine-grained πολιτική παρουσίασε καλύτερες βελτιώσεις σε σχέση με την coarse-grained.

6.3 Μελλοντική Εργασία

Η μελέτη των αλγόριθμων δρομολόγησης στην οποία προβήκαμε έχει δείξει ότι μπορούμε να πετύχουμε αρκετά σημαντική βελτίωση με την εύρεση του καταλληλότερου συνδυασμό των νημάτων σε κάθε πυρήνα. Υπάρχουν πολλοί παράγοντες που επηρεάζουν αυτή την επιλογή, οι περισσότεροι κινούνται γύρω από του πόρους του πυρήνα που μπορούν να γνωρίσουν μεγάλη καθυστέρηση λόγω

της συνύπαρξης περισσότερων του ενός νημάτων σε αυτόν Όπως καταλαβαίνετε μπορεί να βρεθεί ένα μεγάλο σύνολο αλγορίθμων που να εξειδικεύεται σε κάθε παράγοντα γενικά. Αλγόριθμοι που παρακολουθούν τον συνωστισμό στο pipeline και στους καταχωρητές, τον συνωστισμό στην κρυφή μνήμη L1 και L2 κτλ.

Αφού σχεδόν όλες οι μετρικές που χρησιμοποιούνται για αξιολόγηση της επίδοσης τέτοιων συστημάτων σχετίζονται σε τελική ανάλυση με την τιμή του IPC που θα εμφανίσει κάθε νήμα μετά το πέρας της εκτέλεσής του, ίσως να είναι πιο σωστό να μπορέσουμε να οδηγηθούμε στην ζητούμενη βελτίωση δημιουργώντας αλγόριθμους συνδρομολόγησης που να παρακολουθούν την τιμή του IPC για κάθε νήμα στην εκάστοτε περίοδο δρομολόγησης. Η προσέγγιση του IPCS που περιγράφηκε πιο πριν και χρησιμοποιήθηκε σύμφωνα με προηγούμενες έρευνες μπορεί να τύχει αρκετής σχεδιαστικής βελτίωσης. Η τιμή του IPC για κάθε νήμα στο τέλος κάθε περιόδου δεν είναι αρκετή για να χαρακτηρίσει αν η συνδρομολόγηση στο διάστημα το οποίο ακολουθεί θα είναι επιτυχής. Έτσι βρίσκοντας επιπλέον και τη διαφορά του IPC για κάθε νήμα μεταξύ δύο περιόδων και παίρνοντας το άθροισμα όλων των διαφορών μπορούμε να συγκρίνουμε, αν αυτή η ανακατανομή από τον ένα συνδυασμό νημάτων στον άλλο, όπως έχει επιβάλει ο IPCS, είχε τα επιθυμητά αποτελέσματα. Αν πράγματι αυτή η μεταβολή άξιζε τότε μπορεί η κατανομή των νημάτων στους πυρήνες να παραμείνει η ίδια. Αν όμως αυτό οδηγήσει σε χειρότερες μεμονωμένες τιμές IPC, τότε ίσως θα ήταν καλύτερα η συνδρομολόγηση να αλλάξει. Αυτή είναι μια βελτίωση στον IPCS η οποία πιστεύουμε μπορεί να προσφέρει περισσότερη βελτίωση στην εφαρμογή του.

Η συνδρομολόγηση νημάτων στους πυρήνες στις CMT αρχιτεκτονικές είναι ένα τεράστιο κεφάλαιο και οι εταιρείες επεξεργαστών οφείλουν να συνεχίσουν την έρευνα σε αυτό. Ελπίζουμε στο μέλλον να εμφανιστούν και επεξεργαστές οι οποίοι να υποστηρίζουν ένα τέτοιο αποδοτικό αλγόριθμο συνδρομολόγησης νημάτων, ίσως κάποια πιο βελτιωμένη έκδοση αυτών που παρουσιάστηκαν. Τα οφέλη που προσκομίζονται είναι αρκετά σε σχέση με το κόστος, αφού με απλή χρήση μετρητών στις ενδιαφέρουσες περιοχές του επεξεργαστή μπορούμε να πετύχουμε βελτίωση μέχρι και 12%.

Βιβλιογραφία

[1] J. Aas. Understanding the linux 2.6. 8.1 cpu scheduler. *Retrieved Oct*, 16, 2005.

[2] A. Alameldeen, B. Beckmann, R. Dickson, P. Harper, M. Martin, M. Marty, C. Mauer, K. Moore, M. Plakal, D. Sorin, et al. ISCA Tutorial.

[3] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, pages 29–40. ACM New York, NY, USA, 2006.

[4] F.A. Bower, D.J. Sorin, and L.P. Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. *IEEE Micro*, 28(3):17–25, 2008.

[5] S. Chen, P.B. Gibbons, M. Kozuch, V. Liaskovitis, A. Ailamaki, G.E. Blelloch, B. Falsafi, L. Fix, N. Hardavellas, T.C. Mowry, et al. Scheduling threads for constructive cache sharing on CMPs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 105–115. ACM New York, NY, USA, 2007.

[6] M. De Vuyst, R. Kumar, and DM Tullsen. Exploiting unbalanced thread scheduling for energy and performance on a CMP of SMT processors. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10, 2006.

[7] A. El-Moursy, R. Garg, D.H. Albonesi, and S. Dwarkadas. Compatible phase co-scheduling on a CMP of multi-threaded processors. In *Intl. Parallel and Distributed Processing Symp*. Citeseer, 2006.

[8] J. Engblom, D. Ekblom, and V. Ab. Simics: A commercially proven full-system simulation framework. In *Workshop on Simulation in European Space Programmes*. Citeseer, 2006.

[9] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum. Performance of multithreaded chip multiprocessors and implications for operating system design. In *USENIX 2005 Annual Technical Conference*, pages 395–398. Citeseer, 2005.

[10] A. Fedorova, M. Seltzer, and M.D. Smith. Cache-fair thread scheduling for multicore processors. *Division of Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-17-06*, 2006.

[11] A. Fedorova, M. Seltzer, and M.D. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 25–38. IEEE Computer Society Washington, DC, USA, 2007.

[12] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4), 2005.

[13] JL Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[14] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122. IEEE Computer Society Washington, DC, USA, 2004.

[15] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE micro*, 25(2):21–29, 2005.

[16] R. Kumar, DM Tullsen, NP Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, 2005.

[17] T. Li, D. Baumberger, D.A. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. ACM New York, NY, USA, 2007.

[18] PS Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[19] D.T. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1):4–15, 2002.

[20] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, and D.A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[21] C. McNairy and R. Bhatia. Montecito: A dual-core, dual-thread Itanium processor. *IEEE Micro*, 25(2):10–20, 2005.

[22] S. Parekh, S. Eggers, H. Levy, and J. Lo. Thread-sensitive scheduling for SMT processors. *University of Washington Technical Report*, pages 04–02, 2000.

[23] N. Rafique, W.T. Lim, and M. Thottethodi. Architectural support for operating system-driven CMP cache management.

[24] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, 2001.

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57. ACM New York, NY, USA, 2002.

[26] T. Sondag, V. Krishnamurthy, and H. Rajan. Predictive thread-to-core assignment on a heterogeneous multi-core processor. In *Proceedings of the 4th workshop on Programming languages and operating systems*. ACM New York, NY, USA, 2007.

[27] L. Spracklen and SG Abraham. Chip multithreading: Opportunities and challenges. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 248–252, 2005.

[28] S. Sridharan, B. Keck, R. Murphy, S. Chandra, and P. Kogge. Thread migration to improve synchronization performance. In *Workshop on Operating System Interference in High Performance Applications*. Citeseer, 2006.

[29] D. Tam, R. Azimi, and M. Stumm. Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 47–58. ACM New York, NY, USA, 2007.

[30] S. Version. Simics Programming Guide.

[31] S. Version. Simics User Guide for Unix.

[32] AB Virtutech. Simics full system simulator.

[33] V.M. Weaver and S.A. McKee. Using dynamic binary instrumentation to generate multi-platform simpoints: Methodology and accuracy. *Lecture Notes in Computer Science*, 4917:305, 2008.