ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Μη-Γραμμικοί Μετασχηματισμοί Αποθήκευσης και Τεχνικές Πρώιμης Ανάκλησης Δεδομένων για την Αξιοποίηση της Τοπικότητας Αναφοράς σε Σύγχρονες Αρχιτεκτονικές Μικροεπεξεργαστών με Πολυεπίπεδες Ιεραρχίες Μνημών

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ευαγγελία Γ. Αθανασάκη

Αθήνα, Ιούλιος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΑΚΤΙΚΟ ΕΞΕΤΑΣΗΣ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

της

Ευαγγελίας Γ. Αθανασάκη

Διπλωματούχου Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών Ε.Μ.Π. (2002)

Μη-Γραμμικοί Μετασχηματισμοί Αποθήκευσης και Τεχνικές Πρώιμης Ανάκλησης Δεδομένων για την Αξιοποίηση της Τοπικότητας Αναφοράς σε Σύγχρονες Αρχιτεκτονικές Μικροεπεξεργαστών με Πολυεπίπεδες Ιεραρχίες Μνημών

Τριμελής Συμβουλευτική επιτροπή: Παναγιώτης Τσανάκας, επιβλέπων
Γεώργιος Παπακωνσταντίνου
Νεκτάριος Κοζύρης

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την .....

.....	.....	.....
Π. Τσανάκας	Γ. Παπακωνσταντίνου	Ν. Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκ. Καθηγητής Ε.Μ.Π.
.....		.....
Τ. Σελλής		Α. Μπίλας
Καθηγητής Ε.Μ.Π.		Αναπ. Καθηγητής, Πανεπ. Κρήτης
.....		.....
Α. Σταφυλοπάτης		Ν. Παπασπύρου
Καθηγητής Ε.Μ.Π.		Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2006

.....

Ευαγγελία Γ. Αθανασάκη
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Ευαγγελία Γ. Αθανασάκη, 2006
Με επιφύλαξη παντός δικαιώματος - All rights reserved

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται η παρούσα σημείωση. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τη συγγραφέα.
Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτήν τη διατριβή εκφράζουν τη συγγραφέα και δεν πρέπει να θεωρηθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ένα από τα βασικότερα ζητήματα που έχουν να αντιμετωπίσουν οι αρχιτέκτονες υπολογιστών και οι συγγραφείς μεταγλωττιστών είναι η συνεχώς αυξανόμενη διαφορά επίδοσης μεταξύ του επεξεργαστή και της κύριας μνήμης. Προκειμένου να ξεπεραστεί το πρόβλημα αυτό χρησιμοποιούνται διάφοροι μετασχηματισμοί κώδικα, με στόχο τη μείωση του αριθμού των αστοχιών μνήμης και επομένως τη μείωση του μέσου χρόνου που καθυστερούν οι εφαρμογές περιμένοντας τα δεδομένα να φτάσουν από την κύρια μνήμη. Ο μετασχηματισμός υπερκόμβων χρησιμοποιείται ευρέως στους κώδικες φωλιασμένων βρόχων, αναδιατάσσοντας τη σειρά εκτέλεσης των επαναλήψεων, για τη βελτίωση της τοπικότητας των αναφορών σε δεδομένα μνήμης.

Η διατριβή αυτή συμβάλλει στην περαιτέρω αξιοποίηση της τοπικότητας αναφοράς για ιεραρχίες μνημών, βελτιστοποιώντας τον τρόπο αναφοράς επαναληπτικών κωδίκων στη μνήμη. Επιχειρεί να ελαχιστοποιήσει τον αριθμό των αστοχιών μνήμης, αναδιατάσσοντας τη σειρά αποθήκευσης των δεδομένων των πολυδιάστατων πινάκων στη μνήμη, ώστε να ακολουθεί τη σειρά προσπέλασής τους από τον κώδικα φωλιασμένων βρόχων, αφότου έχει εφαρμοστεί σε αυτόν μετασχηματισμός υπερκόμβων. Στη μέθοδο μη-γραμμικής αποθήκευσης που αναπτύσεται, τα δεδομένα των πινάκων χωρίζονται σε ομάδες (blocks) οι οποίες ταυτίζονται με τα tiles του μετασχηματισμού υπερκόμβων, και αποθηκεύονται σε διαδοχικές θέσεις μνήμης. Με τον τρόπο αυτό, η ακολουθία των προσπελάσεων δεδομένων από τον κώδικα βρόχων ευθυγραμμίζεται με τη σειρά αποθήκευσής τους στη μνήμη. Η μετατροπή της πολυδιάστατης δεικτοδότησης των πινάκων στη δεικτοδότηση των μη-γραμμικών διατάξεων δεδομένων που προκύπτουν, γίνεται ταχύτατα με απλές δυαδικές πράξεις πάνω σε "αραιωμένους" ακεραίους. Τα πειραματικά αποτελέσματα και οι προσομοιώσεις αποδεικνύουν ότι η συνολική επίδοση βελτιώνεται σημαντικά, χάρη στη μείωση των αστοχιών μνήμης, όταν συνδυάζεται ο μετασχηματισμός υπερκόμβων με τις μη-γραμμικές διατάξεις δεδομένων σε ομάδες και την δυαδική δεικτοδότηση των δεδομένων.

Η συνέπεια της προτεινόμενης μεθόδου σε ότι αφορά τη συνολική επιτάχυνση του χρόνου εκτέλεσης εξαρτάται σε μεγάλο βαθμό από την επιλογή του κατάλληλου μεγέθους tile. Στην παρούσα διατριβή αναλύεται η διαδικασία παραγωγής των αστοχιών κρυφής μνήμης και TLB στις

μη-γραμμικές διατάξεις δεδομένων σε ομάδες. Σύμφωνα με τα αποτελέσματα της ανάλυσης αυτής, επιλέγεται το βέλτιστο μέγεθος tile, με στόχο τη μέγιστη αξιοποίηση του μεγέθους της L1 κρυφής μνήμης και ταυτόχρονα την αποτροπή των αστοχιών λόγω συγκρούσεων στην κρυφή μνήμη. Αποδεικνύεται ότι σε βελτιστοποιημένους κώδικες, που έχει εφαρμοστεί πρώιμη ανάκληση δεδομένων, ξεδίπλωμα βρόχων, ανάθεση μεταβλητών σε προσωρινούς καταχωρητές και ευθυγράμμιση των διαφορετικών πινάκων, τα tiles, με μέγεθος ίσο με την χωρητικότητα της L1 κρυφής μνήμης, αξιοποιούν την L1 κρυφή μνήμη κατά βέλτιστο τρόπο, ακόμα και στις περιπτώσεις που προσπελαύνονται περισσότεροι από ένας πίνακες δεδομένων. Το προτεινόμενο μέγεθος tile είναι συγκριτικά με άλλες μεθόδους μεγαλύτερο, δίνοντας επιπλέον το πλεονέκτημα του μειωμένου αριθμού αστοχιών λόγω των λανθασμένων προβλέψεων διακλάδωσης των φωλιασμένων βρόχων.

Τέλος, ένα άλλο θέμα που δεν είχε μέχρι στιγμής εξεταστεί στην εως τώρα βιβλιογραφία είναι η επίδοση μίας εφαρμογής, όταν αυτή εκτελείται μόνη της σε μηχανήματα που είναι εξοπλισμένα με την τεχνική της Ταυτόχρονης Πολυνημάτωσης. Η τεχνική αυτή είχε προταθεί για τη βελτίωση του ρυθμού εκτέλεσης εντολών μέσω της ταυτόχρονης εκτέλεσης εντολών από διαφορετικά νήματα (τα οποία προέρχονται είτε από διαφορετικές εφαρμογές ή από παραλληλοποιημένες εφαρμογές). Οι πρόσφατες μελέτες έχουν δείξει ότι η ανομοιογένεια των ταυτόχρονα εκτελούμενων νημάτων είναι από τους παράγοντες που επηρεάζουν θετικά την επίδοση των εφαρμογών. Ωστόσο, η επιτάχυνση των παράλληλων εφαρμογών εξαρτάται σε μεγάλο βαθμό από τους μηχανισμούς συγχρονισμού και επικονωνίας μεταξύ των διαφορετικών, αλλά πιθανότατα εξαρτώμενων μεταξύ τους, νημάτων. Επιπλέον, καθώς τα διαφορετικά νήματα των παράλληλων εφαρμογών έχουν στις περισσοτέρες περιπτώσεις όμοιου τύπου εντολές (π.χ. πράξεις κινητής υποδιαστολής, ακέραιες πράξεις, άλματα, κλπ), κατά την εκτέλεσή τους στους κοινούς πορους του συστήματος συνοστίζονται και σειριοποιούνται κατά τη διέλευσή τους μέσα από συγκεκριμένες λειτουργικές μονάδες, με αποτέλεσμα να μην μπορεί να επιτευχθεί σημαντική επιτάχυνση των εφαρμογών. Στη διατριβή αυτή αποτιμώνται και συγκρίνονται η πρώιμη ανάκληση δεδομένων και ο παραλληλισμός επιπέδου νήματος (thread-level parallelism - TLP). Τέλος εξετάζεται η επίδραση των μηχανισμών συγχρονισμού στις πολυνηματικές παράλληλες εφαρμογές που εκτελούνται από επεξεργαστές που διαθέτουν ταυτόχρονη πολυνημάτωση.

Λέξεις-κλειδιά: Ιεραρχία Μνήμης, Μετασχηματισμός Υπερκόμβων (Tiling), Μη-γραμμικοί μετασχηματισμοί δεδομένων, Δυαδικές μάσκες, Πρώιμη ανάκληση δεδομένων, Ξεδίπλωμα βρόχων, Επιλογή Μεγέθους Tile, Ταυτόχρονη Πολυνημάτωση, Φορτίο μίας μόνο Εφαρμογής, Απελευθέρωση Πόρων Συστήματος από τα άεργα Νήματα.

Abstract

One of the key challenges computer architects and compiler writers are facing, is the increasing discrepancy between processor cycle times and main memory access times. To overcome this problem, program transformations that decrease cache misses are used, to reduce average latency for memory accesses. Tiling is a widely used loop iteration reordering technique for improving locality of references. Tiled codes modify the instruction stream to exploit cache locality for array accesses.

This thesis adds some intuition and some practical solutions to the well-studied memory hierarchy problem. We further reduce cache misses, by restructuring the memory layout of multi-dimensional arrays, that are accessed by tiled instruction code. In our method, array elements are stored in a blocked way, exactly as they are swept by the tiled instruction stream. We present a straightforward way to easily translate multi-dimensional indexing of arrays into their blocked memory layout using simple binary-mask operations. Indices for such array layouts are now easily calculated based on the algebra of dilated integers, similarly to morton-order indexing. Actual experimental and simulation results illustrate that execution time is greatly improved when combining tiled code with tiled array layouts and binary mask-based index translation functions.

The stability of the achieved performance improvements are heavily dependent on the appropriate selection of tile sizes, taking into account the actual layout of the arrays in memory. This thesis provides a theoretical analysis for the cache and TLB performance of blocked data layouts. According to this analysis, the optimal tile size that maximizes L1 cache utilization, should completely fit in the L1 cache, to avoid any interference misses. We prove that when applying optimization techniques, such as register assignment, array alignment, prefetching and loop unrolling, tile sizes equal to L1 capacity offer better cache utilization, even for loop bodies that access more than just one array. Increased self- or/and cross-interference misses are now tolerated through prefetching. Such larger tiles also reduce lost CPU cycles due to less mispredicted branches.

Another issue, that had not been thoroughly examined so far, is Simultaneous Multithreading (SMT) for single-program workloads. SMT has been proposed to improve system throughput by overlapping multiple (either multi-programmed or explicitly parallel) threads on a single wide-issue processor. Recent studies have demonstrated that heterogeneity of simultaneously executed applications can bring up significant performance gains due to SMT. However, the speedup of a single application that is parallelized into multiple threads, is often sensitive to the efficiency of synchronization and communication mechanisms between its separate, but possibly dependent, threads. Moreover, as these separate threads tend to put pressure on the same architectural resources, no significant speedup can be achieved. This thesis evaluates and contrasts software prefetching and thread-level parallelism (TLP) techniques. It also examines the effect of thread synchronization mechanisms on multithreaded parallel applications that are executed on a single SMT processor.

Keywords: Memory Hierarchy, Loop tiling, Blocked Array Layouts, Binary Masks, Prefetching, Loop Unrolling, Tile Size Selection, Simultaneous Multithreading, Single Program Workload, Resource Release by idle Threads.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Αντί Προλόγου

Η παρούσα διδακτορική διατριβή εκπονήθηκε στον Τομέα Τεχνολογίας Πληροφορικής και Υπολογιστών, της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, του Εθνικού Μετσόβιου Πολυτεχνείου. Περιλαμβάνει την έρευνα και τα συμπεράσματα που προέκυψαν κατά τη διάρκεια των μεταπτυχιακών σπουδών μου στο Εργαστήριο Υπολογιστικών Συστημάτων της σχολής αυτής. Η διατριβή αποτελείται από δύο μέρη: Το πρώτο είναι γραμμένο στα αγγλικά, προκειμένου να μπορεί να διαβαστεί από την ακαδημαϊκή κοινότητα εκτός Ελλάδας. Το δεύτερο μέρος αποτελεί περιληπτική μετάφραση του πρώτου στα ελληνικά.

Στις ακόλουθες γραμμές θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου σε όλους εκείνους που με βοήθησαν είτε άμεσα με την καθοδήγηση που μου προσέφεραν, είτε έμμεσα με την ηθική στήριξή τους και συνέβαλαν ουσιαστικά στην ολοκλήρωση της παρούσας διατριβής. Πρώτον από όλους θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Παναγιώτη Τσανάκα, καθώς εκείνος με μύησε στον κόσμο της Αρχιτεκτονικής Υπολογιστών, ως προπτυχιακή φοιτήτρια, και μου μετέδωσε την αγάπη του για το αντικείμενο, το οποίο επέλεξα στη συνέχεια να διερευνήσω κατά τη διάρκεια των μεταπτυχιακών σπουδών μου. Πρόκειται για τον άνθρωπο του οποίου η ήρεμη δύναμη, η ευγένεια και η ηθική έδινε στις δύσκολες καταστάσεις την ισορροπία και τη διέξοδο. Τον ευχαριστώ ιδιαίτερα για την εμπιστοσύνη με την οποία με περιέβαλε και την αυτοπεποίθηση που μου μετέδιδε.

Στη συνέχεια θα ήθελα να ευχαριστήσω το δεύτερο μέλος της συμβουλευτικής επιτροπής μου, τον καθηγητή Γεώργιο Παπακωνσταντίνου, ο οποίος ως κεφαλή του εργαστηρίου μου έδωσε τη δυνατότητα να βρίσκομαι σε ένα καλά δομημένο και άρτια οργανωμένο χώρο. Είναι ο άνθρωπος που με την τεχνογνωσία και την περιρρέουσα γνώση που μου εξασφάλισε, αποτέλεσε σημείο αναφοράς στο επιστημονικό του πεδίο. Τον ευχαριστώ θερμά, γιατί με την ακεραιότητα των προθέσεων και των ενεργειών του, αποτελούσε πάντα πόλο έλξης αντίστοιχα ακέραιων προπτυχιακών και μεταπτυχιακών φοιτητών και πολύτιμων συνεργατών.

Επιπλέον, νιώθω το χρέος να εκφράσω τις ειλικρινείς ευχαριστίες μου στο τρίτο μέλος της συμβουλευτικής επιτροπής μου, τον Επίκουρο καθηγητή Νεκτάριο Κοζύρη. Αισθάνομαι ιδιαίτερα τυχερή που στα τελευταία έτη των προπτυχιακών σπουδών μου είχα την ευκαιρία να ακούσω τις διαλέξεις του και αργότερα, κατά την εκπόνηση της διδακτορικής μου διατριβής, να συνεργαστώ μαζί του. Με το προσωπικό παράδειγμα της εργατικότητας, της μεθοδικότητας και της αδιάλειπτης ανανέωσης των γνώσεών του, αποτελούσε πάντοτε, για όλα τα μέλη του εργαστηρίου, κινητήρια

δύναμη προς τη συνεχή αναζήτηση ερεθισμάτων, που οδήγησαν το εργαστήριο στην αιχμή της τεχνολογίας. Ήταν ο άνθρωπος που διεύρυνε τους ορίζοντές μου, επιστημονικούς και κοινωνικούς, με τις παροτρύνσεις του να επισκεφθώ χώρες του εξωτερικού με κορυφαία πανεπιστήμια, να έρθω σε επαφή με τους πλέον αναγνωρισμένους επιστήμονες στο πεδίο της επιστήμης μου και με άλλους υποψήφιους διδάκτορες, που διέθεταν ανέλπιστο ζήλο για την έρευνά τους. Τον ευχαριστώ πραγματικά για το χρόνο που διέθεσε για να παρακολουθεί την ερευνητική μου πορεία, αλλά και για τις ώρες που αφιέρωσε παρευρισκόμενος στις συλλογικές συζητήσεις του εργαστηρίου.

Θα ήταν παράλειψη να μην ευχαριστήσω τα υπόλοιπα μέλη του εργαστηρίου, με πρώτους τον Νίκο Αναστόπουλο και τον Κορνήλιο Κούρτη, που η συνεργασία μαζί τους αποδείχτηκε ιδιαίτερα δημιουργική. Τα αποτελέσματα της παρούσας διατριβής θα ήταν σίγουρα φτωχότερα χωρίς την ουσιαστική συμβολή τους. Αν και νεώτερα μέλη του εργαστηρίου, η βαθιά γνώση του αντικειμένου που καλλιέργησαν από πολύ νωρίς, η εφευρετικότητα και η διερευνητική τους διάθεση, σε συνδυασμό με την αθόρυβη εργατικότητά τους, τους κατέστησε για μένα αναντικατάστατους συνοδοιπόρους. Εύχομαι το μέλλον να ανταμείψει τους κόπους τους και να ικανοποιήσει τα όνειρά τους.

Ακόμη, θα ήθελα να ευχαριστήσω τον Γιώργο Γκούμα, τον Άρη Σωτηρόπουλο, το Νίκο Δροσινό και την αδελφή μου, Μαρία Αθανασάκη. Αισθάνομαι ότι χωρίς τις γνώσεις και την παρουσία τους το εργαστήριο θα ήταν πολύ άδειο από τεχνογνωσία και ωριμότητα, αλλά και από ζωή, νέες ιδέες και αντίλογο, ψυχισμό: όλα εκείνα τα στοιχεία που σε τραβούν να περάσεις αμέτρητες ώρες μέσα στους τέσσερις τοίχους του εργαστηριακού περιβάλλοντος, νιώθοντας ότι ζεις αληθινή ζωή. Δεν θα μπορούσα να μην κάνω ιδιαίτερη αναφορά στην αδελφή μου, γιατί με έμαθε να μοιράζομαι, να αγαπάω, μου έδινε και μου δίνει μαθήματα προνοητικότητας, αποτελεί για μένα τον οδηγό προς ανώτερους στόχους, συμπληρώνει τις αδυναμίες μου, με κάνει να νιώθω πιο δυνατή και μόνο που είναι δίπλα μου. Τέλος, θα ήθελα να αναφέρω ξεχωριστά τον Βαγγέλη Κούκη, τον Αντώνη Χαζάπη, τον Αντώνη Ζήσιμο, το Γιώργο Τσουκαλά, το Γιώργο Βερυγάκη και όλα τα υπόλοιπα νέα ή παλιότερα μέλη του εργαστηρίου, που με την ενεργητικότητα, τις γνώσεις και το χαρακτήρα τους χτίζουν την κουλτούρα του Εργαστηρίου Υπολογιστικών Συστημάτων.

Τέλος, ευχαριστώ θερμά το Κοινωφελές Ίδρυμα Αλέξανδρος Ωνάσης για την οικονομική στήριξη που μου παρείχε μέσω μίας υποτροφίας μεταπτυχιακών σπουδών.

Η εργασία αυτή αφιερώνεται στην οικογένειά μου, σε όλους εκείνους που κατέχουν μία ξεχωριστή θέση στη ζωή μου.

Αθήνα, Ιούλιος 2006

Ευαγγελία Αθανασάκη

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Είναι πλέον ευρέως γνωστό, ότι το χάσμα μεταξύ της αρχικής καθυστέρησης αποστολής δεδομένων στον επεξεργαστή από τη μνήμη και της συχνότητας λειτουργίας του επεξεργαστή διαρκώς διευρύνεται [PJ03]. Έτσι, οι περισσότερες εφαρμογές καταναλώνουν ένα σημαντικό ποσοστό του συνολικού χρόνου εκτέλεσης, περιμένοντας τα δεδομένα να φτάσουν στον επεξεργαστή από την κύρια μνήμη. Για να αποτρέψουμε τα προβλήματα που πηγάζουν λόγω της διαφοράς επίδοσης των δύο συστατικών του υπολογιστικού συστήματος, το σύστημα μνήμης δομείται ιεραρχικά. Επιπλέον, εισάγονται μετασχηματισμοί κώδικα, προκειμένου να αυξηθεί η τοπικότητα των αναφορών στους επαναληπτικούς κώδικες εκτέλεσης, με συνέπεια οι περισσότερες αναφορές εντολών του επεξεργαστή σε λέξεις μνήμης να ικανοποιούνται από τα πιο ψηλά και γρηγορότερα επίπεδα της ιεραρχίας μνήμης.

Η παρούσα διατριβή αποτελεί μία προσέγγιση του προβλήματος, που κάποιοι το ονόμασαν τείχος μνήμης (memory wall [WM95]), από την πλευρά του λογισμικού, λαμβάνοντας υπόψη τα χαρακτηριστικά του υλικού σε ότι αφορά τη δομή του συστήματος μνήμης. Το παρόν κεφάλαιο περιγράφει στο εξής το κίνητρο, που αποτέλεσε την αφετηρία την εργασίας αυτής, και κάνει μία μικρή εισαγωγή στους στόχους της. Τέλος, δίνεται μία λίστα των στοιχείων όπου συμβολής της στο γενικότερο ερευνητικό πεδίο και παρουσιάζεται το σχεδιάγραμμα που θα ακολουθηθεί στα επόμενα κεφάλαια.

1.1 Το κίνητρο

Η υπολογιστική ισχύς των σύγχρονων μικροεπεξεργαστών διαρκώς αυξάνεται, καθώς μεγαλώνουν οι συχνότητες ρολογιού και αξιοποιείται ο παραλληλισμός σε πολλαπλά επίπεδα στο εσωτερικό τους. Ωστόσο, η επεξεργαστική ισχύς δεν είναι ο μοναδικός παράγοντας που επηρεάζει την επίδοση ενός υπολογιστικού συστήματος. Προκειμένου να αποκομισθούν στο μέγιστο τα οφέλη που προσφέρει ένας ταχύτατος υπολογιστής, θα πρέπει να συνοδεύεται από ένα αντίστοιχα γρήγορο σύστημα μνήμης.

Δυστυχώς, το σύστημα μνήμης έχει αποτελέσει το αδύνατο σημείο των συστημάτων υψηλών επιδόσεων, περιορίζοντας την ταχύτητα με την οποία τα δεδομένα φτάνουν στον επεξεργαστή και απομακρύνονται από αυτόν. Το χάσμα μεταξύ κύριας μνήμης και επεξεργαστή διευρύνεται κάθε χρόνο. Η επίδοση των επεξεργαστών αυξάνεται με ρυθμούς 55% το χρόνο, ενώ η αρχική καθυστέρηση προσπέλασης των δεδομένων που βρίσκονται στη μνήμη (latency) μειώνεται με ρυθμούς μόλις 7% [PJ03]. Μία οικονομική λύση στην ακόρεστη ανάγκη των προγραμμάτων για μεγάλης χωρητικότητας γρήγορη μνήμη, είναι η ιεραρχία μνήμης. Η αξία της ιεραρχίας μνήμης αυξάνεται με την πάροδο του χρόνου. Για παράδειγμα, το 1980 οι μικροεπεξεργαστές σχεδιάζονταν χωρίς καθόλου κρυφή μνήμη (cache), ενώ το 2005 τα περισσότερα συστήματα κατασκευάζονταν με 2 ή περισσότερα επίπεδα μνήμης. Η ιεραρχική οργάνωση των υποσυστημάτων μνήμης στηρίζεται στην αρχή της χρονικής και χωρικής τοπικότητας αναφορών και της επαναχρησιμοποίησης των δεδομένων των προγραμμάτων.

Ο σχεδιασμός της κρυφής μνήμης βασίζεται στη θεμελιώδη αρχή της τοπικότητας των αναφορών: την Χρονική και τη Χωρητική Τοπικότητα Δεδομένων. Αν θεωρήσουμε ότι η αρχή αυτή εφαρμόζεται με επιτυχία, το σύστημα αποθήκευσης δεδομένων των σύγχρονων επεξεργαστών θα έπρεπε θεωρητικά να προσεγγίζει την ταχύτητα του αρχείου προσωρινών καταχωρητών, και ταυτόχρονα να έχει τη χωρητικότητα του σκληρού δίσκου. Η αποτελεσματικότητα της ιεραρχίας μνήμης εξαρτάται όχι μόνο από το σχεδιασμό του υλικού, αλλά επίσης, από την ίδια την εφαρμογή, έτσι όπως προκύπτει μετά τις βελτιστοποιήσεις του μεταγλωττιστή. Ασφαλώς, οι κώδικες των πραγματικών εφαρμογών δεν θα μπορούσαν να έχουν την ιδανική τοπικότητα αναφορών. Οι περισσότερες αριθμητικές εφαρμογές έχουν κακή απόδοση, επειδή χειρίζονται μεγάλους όγκους δεδομένων, τα οποία ασφαλώς δεν είναι δυνατόν να χωρούν στα ανώτερα ιεραρχικά επίπεδα μνήμης. Ωστόσο, οι προσπελάσεις δεδομένων έχουν στις περισσότερες περιπτώσεις προβλέψιμα μοτίβα προσπέλασης, τα οποία, υπό συνθήκες, μπορούν να έχουν καλή τοπικότητα αναφορών. Αν εκμεταλλευτούμε σωστά την κανονικότητα των αναφορών, μπορούμε να περιορίσουμε στο ελάχιστο τις προσπελάσεις δεδομένων των οποίων η αναζήτηση φτάνει μέχρι την κύρια μνήμη. Για το λόγο αυτό, ένα σημαντικό κομμάτι της έρευνας αφιερώνεται στη βελτιστοποίηση κώδικα και την εφαρμογή διαφόρων τεχνικών μετασχηματισμού, προκειμένου να μειώσουν τις μεταφορές δεδομένων από την κύρια μνήμη, τόσο, που ο αριθμός τους να προσεγγίζει τον αριθμό των υποχρεωτικών αστοχιών δεδομένων (αστοχιών πρώτης αναφοράς σε αυτά).

Οι φωλιασμένοι βρόχοι αποτελούν συνήθως τα κομμάτια του κώδικα που καταναλώνουν το μεγαλύτερο ποσοστό επεξεργαστικής ισχύος και μνήμης, και τελικά το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης του προγράμματος. Τέτοιοι βρόχοι συναντώνται πολύ συχνά στις επιστημονικές εφαρμογές, για την επεξεργασία εικόνας, στη δυναμική των ρευστών, την ανάλυση γεωφυσικών δεδομένων και στην όραση υπολογιστών. Η βελτιστοποίηση της εκτέλεσης τέτοιων φωλιασμένων βρόχων, ισοδυναμεί ουσιαστικά με βελτιστοποίηση ολόκληρου του προγράμματος. Ωστόσο, οι βελτιστοποιήσεις με το χέρι (από τους προγραμματιστές) είναι σε γενικές γραμμές κακή πρακτική, επειδή ο κώδικας που τελικά προκύπτει είναι δύσκολο να ελεγχθεί, να εντοπιστούν

σφάλματα, ή να γίνουν προσθήκες-αλλαγές σε αυτόν από άλλον προγραμματιστή. Επιπλέον, είναι ακόμη πιο δύσκολο να έχει ευελιξία προσαρμογής σε άλλες αρχιτεκτονικές με διαφορετικά χαρακτηριστικά και απαιτήσεις.

Θα μπορούσαμε, λοιπόν, να εξαγάγουμε το συμπέρασμα ότι η βελτιστοποίηση κώδικα οφείλει να είναι το αποτέλεσμα μίας αυτόματης ή ημι-αυτόματης διαδικασίας. Από την άλλη μεριά, οι βελτιστοποιήσεις που γίνονται με το χέρι, για την προσαρμογή κωδίκων στις απαιτήσεις συγκεκριμένων μηχανημάτων, μπορούν να φέρουν τη μέγιστη δυνατή επίδοση. Δηλαδή, ακόμα και ένα αυτόματο εργαλείο βελτιστοποίησης κώδικα χρειάζεται να λαμβάνει πληροφορίες σχετικά με τα ιδιαίτερα χαρακτηριστικά της πλατφόρμας που πρόκειται να εκτελεστεί μία εφαρμογή, αλλά και σχετικά με τις ιδιαιτερότητες της συγκεκριμένης εφαρμογής.

Στόχος της παρούσας διατριβής είναι η ανάπτυξη μίας τεχνικής βελτιστοποίησης κώδικα, η οποία για τη βελτίωση της επίδοσης των αριθμητικών εφαρμογών, να εκμεταλλεύεται την τοπικότητα των αναφορών και την ιεραρχία μνήμης. Επικεντρώνουμε το ενδιαφέρον μας στην αναδιάταξη της θέσης αποθήκευσης των δεδομένων στη μνήμη, για την ελαχιστοποίηση των άεργων κύκλων του επεξεργαστή, εξαιτίας των αστοχιών δεδομένων. Προκειμένου να καταστήσουμε ικανή την αυτόματη παραγωγή κώδικα φωλιασμένων βρόχων, μετασχηματισμένου με tiling, ο οποίος θα προσπελάζει δεδομένα αποθηκευμένα στη μνήμη σε μη-γραμμικές διατάξεις, προτείνουμε μία μέθοδο υπολογισμού της διεύθυνσης των ζητούμενων δεδομένων μέσω μη-γραμμικών δεικτών. Υιοθετήσαμε κάποιο είδος μη-γραμμικών μετασχηματισμών ομαδοποίησης δεδομένων, και χρησιμοποιήσαμε την άλγεβρα των "αραιωμένων" ακεραίων, παρόμοια με τους αναδρομικούς πίνακες Morton. Πετύχαμε, λοιπόν, το συνδυασμό της βέλτιστης τοπικότητας δεδομένων, χάρη στους μη-γραμμικούς μετασχηματισμούς ομαδοποίησης δεδομένων, με μία γρήγορη μέθοδο εντοπισμού και προσπέλασης των ζητούμενων στοιχείων. Ο εντοπισμός της θέσης των ζητούμενων στοιχείων πραγματοποιείται ταχύτατα, μέσω υπολογισμών επί δυαδικούς τελεστές. Η προτεινόμενη μέθοδος επιτυγχάνει τη δραστική μείωση του συνολικού αριθμού των αστοχιών της κρυφής μνήμης, γιατί η σειρά αποθήκευσης των στοιχείων στη μνήμη ευθυγραμμίζεται με τη σειρά προσπέλασής τους από των μετασχηματισμένο με tiling κώδικα φωλιασμένων βρόχων. Τέλος, η όλη διαδικασία μετασχηματισμού υλοποιείται χωρίς να προσθέτουμε χρονοβόρες εντολές, που θα μπορούσαν να καθυστερούν την τελική επίδοση του συστήματος.

1.1.1 Μετασχηματισμοί Κώδικα

Μέσω των μετασχηματισμών κώδικα είναι δυνατόν να αυξηθεί το ποσοστό των προσπελάσεων θέσεων μνήμης που ικανοποιούνται από τις κρυφές μνήμες. Κατά αυτόν τον τρόπο, αλλάζει η δομή του επαναληπτικού κώδικα εκτέλεσης με στόχο τη βελτίωση της τοπικότητας δεδομένων των αναφορών. Η ανταλλαγή βρόχων (loop permutation), η αναστροφή βρόχου (loop reversal) και το λόξεμα βρόχων (loop skewing) επιτυγχάνουν την αλλαγή της σειράς εκτέλεσης των επαναλήψεων και επομένως της σειράς προσπέλασης των δεδομένων, βελτιώνοντας την τοπικότητα δεδομένων. Το ξεδίπλωμα βρόχων (loop unrolling) εκμεταλλεύεται την ύπαρξη προσωρινών καταχωρητών

(registers) και την αρχιτεκτονική αγωγού (pipeline), βελτιώνοντας τη χρονική τοπικότητα αναφορών σε δεδομένα [Jim99]. Η συγχώνευση βρόχων (loop fusion) ή η διάσπαση βρόχων (loop fission/distribution) μπορεί να βοηθήσει έμμεσα στην αξιοποίηση της επαναχρησιμοποίησης, δίνοντας τη δυνατότητα να εφαρμοστούν κάποιοι μετασχηματισμοί κώδικα που αρχικά δεν ήταν νόμιμοι [MCT96]. Ο μετασχηματισμός tiling και ο μετασχηματισμός επίδεσης δεδομένων (data shackling) [KPCM99] παρόλο που χειρίζονται το θέμα της τοπικότητας δεδομένων από διαφορετική σκοπιά, μετασχηματίζουν τη ροή ελέγχου του προγράμματος, μειώνοντας το μέγεθος του συνόλου δεδομένων το οποίο επεξεργάζεται το πρόγραμμα και επαναχρησιμοποιεί σε γειτονικές επαναλήψεις. Έτσι αξιοποιείται η χωρική και η χρονική επαναχρησιμοποίηση δεδομένων των πινάκων. Οι ορθομοναδιαίοι μετασχηματισμοί κώδικα (unimodular control transformations), όπως περιγράφονται στην πλέον δημοφιλή εργασία των Wolf και Lam [WL91], καθώς και οι συνδυαστικοί μετασχηματισμοί (compound transformations) των McKinley κλπ [MCT96], προσπαθούν να βρουν τον καλύτερο συνδυασμό τέτοιων μετασχηματισμών, οι οποίοι, χρησιμοποιούμενοι σε συνδυασμό με τον μετασχηματισμό tiling, διασφαλίζουν τη σωστή σειρά υπολογισμών, ενώ αυξάνεται η τοπικότητα των αναφορών στην κρυφή μνήμη.

Με τους μετασχηματισμούς βρόχων, αλλάζουμε τη σειρά με την οποία οι εντολές σε ένα φωλιασμένο βρόχο προσπελάζουν δεδομένα, αλλά όχι τη σειρά αποθήκευσης δεδομένων. Προσεγγίζοντας το πρόβλημα από την πλευρά των δεδομένων [Kan01], ο Kandemir προτείνει την ενοποίηση των πινάκων δεδομένων (array unification). Πρόκειται για μία τεχνική με την οποία πολλαπλοί πίνακες αποθηκεύονται σε έναν ενιαίο χώρο στη μνήμη σε διάταξη διαστρωμάτωσης (αποθηκεύοντας διαδοχικά σε στρώματα στοιχεία διαφορετικών πινάκων), ομαδοποιώντας τους πίνακες εκείνους που προσπελάζονται σε γειτονικές επαναλήψεις. Κάθε ομάδα μετασχηματίζεται κατάλληλα προκειμένου να επιτευχθεί η βέλτιστη χωρική τοπικότητα δεδομένων και να μειωθεί ο αριθμός των αστοχιών σύγκρουσης. Οι Rivera και Tseng στην εργασία [RT98a] χρησιμοποιούν παραγεμίσματα (padding) για να αποτρέψουν έναν μεγάλο αριθμό από αστοχίες σύγκρουσης. Τα παραγεμίσματα που εισάγονται μεταξύ των διαφορετικών πινάκων (inter-variable padding) μετακινούν τη θέση απεικόνισης του αρχικού στοιχείου κάθε πίνακα στην κρυφή μνήμη (τη διεύθυνση βάσης), ενώ τα παραγεμίσματα εντός των πινάκων (intra-variable padding) αλλάζουν τη διάσταση των πινάκων. Σε γραμμικά αλγεβρικά προβλήματα, για διάφορες χωρητικότητες κρυφών μνημών και για διαφορετικά μεγέθη πινάκων, οι κρίσιμες αποστάσεις μεταξύ στοιχείων διαφορετικών στηλών στους πίνακες, που αναμένεται να προκαλέσουν αστοχίες σύγκρουσης, υπολογίζονται μέσω του Ευκλείδειου αλγορίθμου (Euclidean gcd algorithm).

Σε ορισμένες περιπτώσεις, βελτιώνοντας την τοπικότητα των αναφορών για μία ομάδα πινάκων, με την επιλογή ενός συγκεκριμένου τρόπου αποθήκευσης μπορεί να επηρεαστεί αρνητικά ο αριθμός των αστοχιών της κρυφής μνήμης σε κάποιες άλλες αναφορές στους πίνακες αυτούς. Για το λόγο αυτό, προτάθηκε η συνδυασμένη εφαρμογή μετασχηματισμών βρόχων και δεδομένων στις εργασίες [CL95] και [KRC99]. Οι Cierniak και Li [CL95] παρουσιάζουν έναν αλγόριθμο βελτιστοποίησης της τοπικότητας των αναφορών στην κρυφή μνήμη, ο οποίος συνδυάζει τους μετασχηματισμούς

βρόχων με τους γραμμικούς μετασχηματισμούς πινάκων δεδομένων. Ένας ακόμη συστηματικός αλγόριθμος που ενοποιεί τους μετασχηματισμούς βρόχων και δεδομένων, παρουσιάστηκε από τους Kandemir κλπ στις εργασίες [KCS+99], [KRCB01], [KRC99], όπου δίνεται έμφαση στην αξιοποίηση της χωρικής επαναχρησιμοποίησης δεδομένων, προκειμένου να επιτευχθεί η βέλτιστη τοπικότητα των αναφορών.

Σε όλες τις προηγούμενες προσεγγίσεις υποθέταμε γραμμικές διατάξεις αποθήκευσης των δεδομένων στη μνήμη. Οι γλώσσες προγραμματισμού υποστηρίζουν δομές πολυδιάστατων πινάκων, οι οποίοι αποθηκεύονται στη μνήμη γραμμικά, είτε κατά γραμμές είτε κατά στήλες. Ωστόσο, οι γραμμικές διατάξεις δεδομένων στη μνήμη δεν ταιριάζουν με την ακολουθία προσπέλασης των δεδομένων από τους μετασχηματισμένους κώδικες φωλιασμένων βρόχων με χρήση tiling. Καθώς τέτοιοι κώδικες δίνουν έμφαση στην επαναχρησιμοποίηση δεδομένων που βρίσκονται σε μία υποπεριοχή (tile) του συνολικού πίνακα, θα ήταν ιδανικό αν τα δεδομένα του tile αυτού μπορούσαν να τοποθετηθούν σε συνεχόμενες θέσεις στη μνήμη. Με άλλα λόγια, δεδομένου ότι η επαναληπτική εκτέλεση εντολών και, κατά συνέπεια, η ακολουθία προσπέλασης δεδομένων είναι οργανωμένες σε tiles, θα ήταν λογικό να αποθηκεύαμε και τα δεδομένα που ζητούνται με την αντίστοιχη σειρά. Κατά αυτόν τον τρόπο, οι αναφορές των εντολών στη μνήμη θα ευθυγραμμίζονταν με τα δεδομένα. Πέρα από τη βελτίωση της τοπικότητας των αναφορών (δηλαδή μείωση υποχρεωτικών αστοχιών), οι αστοχίες σύγκρουσης μειώνονται στο ελάχιστο, ιδιαίτερα στις κρυφές μνήμες ευθείας απεικόνισης και τις συσχετιστικές μνήμης με μικρό βαθμό συσχέτισης. Αυτό συμβαίνει γιατί όλα τα στοιχεία των πινάκων που ανήκουν στο ίδιο tile βρίσκονται αποθηκευμένα σε συνεχόμενες θέσεις μνήμης, και επομένως αποφεύγεται η σύγκρουση μεταξύ των στοιχείων του ίδιου πίνακα.

1.1.2 Μη-γραμμικές Διατάξεις Αποθήκευσης Δεδομένων

Οι Chatterjee κλπ στις εργασίες τους [CJL+99], [CLPT99] μελέτησαν τα πλεονεκτήματα των μη γραμμικών διατάξεων ομαδοποίησης δεδομένων στη μνήμη και ποσοτικοποίησαν το κόστος υλοποίησής τους. Πρότειναν δύο οικογένειες τέτοιων διατάξεων. Και οι δύο οικογένειες διαιρούν τα στοιχεία των πινάκων σε tiles τέτοιου μεγέθους, που να ταιριάζει με τα χαρακτηριστικά της κρυφής μνήμης. Τα στοιχεία στο εσωτερικό κάθε tile αποθηκεύονται σειριακά (γραμμικά). Παρόλο που υποστηρίζουν ότι η προτεινόμενη τεχνική αυξάνει την επίδοση του συστήματος, σε ότι αφορά το χρόνο εκτέλεσης τα πειραματικά αποτελέσματα ανατρέπουν τη θεωρία. Η χρήση τετραδιάστατων πινάκων, λόγω της χρονοβόρας πολλαπλής αναφοράς στη μνήμη για τον υπολογισμό των δεικτών, φαίνεται να αντισταθμίζει το πλεονέκτημα των μη-γραμμικών διατάξεων δεδομένων, σχετικά με την αυξημένη τοπικότητα δεδομένων που επιτυγχάνουν. Ακόμα και αν μετατρέπαμε τους τετραδιάστατους πίνακες σε διδιάστατους, όπως πρότειναν οι Lin κλπ [LLC02], οι οποίοι απεικονίζουν τα στοιχεία από τη μία μορφή στην άλλη μέσω σχημάτων Karnaugh, η σωστή δεικτοδότηση των ζητούμενων στοιχείων πινάκων απαιτεί ακριβούς (σε υπολογιστικό χρόνο) δείκτες.

Οι μη γραμμικές διατάξεις δεδομένων χρησιμοποιήθηκαν επίσης από τους Wise κλπ στις εργασίες [WAFG01] και [WF99] σε συνδυασμό με την ιεραρχική δεικτοδότηση Ahnentafel και Morton.

Παρόλο που οι διατάξεις δενδρικής δομής τεσσάρων διακλαδώσεων φαίνεται να δίνει καλά αποτελέσματα στους αναδρομικούς αλγορίθμους, χάρη στη βελτιστοποιημένη δεικτοδότηση των στοιχείων των πινάκων, τα αποτελέσματα δεν είναι καλά στην περίπτωση μη αναδρομικών αλγορίθμων. Ιδιαίτερα λόγω του γεγονότος ότι η αναδρομικότητα της διάταξης δεδομένων φτάνει στο επίπεδο του ενός μόνο στοιχείου, εισάγεται επιπρόσθετη μείωση επίδοσης.

Οι μη γραμμικές διατάξεις δεδομένων αποδείχτηκε ότι ευνοούν την τοπικότητα δεδομένων σε όλα τα επίπεδα της ιεραρχίας μνήμης, συμπεριλαμβανομένου της L1 και L2 κρυφής μνήμης και των TLBs. Τα πειραματικά αποτελέσματα της εργασίας [RT99b], στην οποία λαμβάνεται υπόψη και αξιοποιείται η τοπικότητα δεδομένων στην L2 κρυφή μνήμη (όχι μόνο σε ένα επίπεδο της κρυφής μνήμης), επιδεικνύουν μειωμένο αριθμό αστοχιών κρυφής μνήμης. Ωστόσο, η συνολική βελτίωση του χρόνου εκτέλεσης δεν είναι ιδιαίτερα σημαντική. Επομένως, στοχεύοντας στη βελτιστοποίηση της τοπικότητας δεδομένων για την L1 κρυφή μνήμη, επιτυγχάνεται σχεδόν η βέλτιστη δυνατή επίδοση. Οι εργασίες που έχουν περιγραφεί μέχρι αυτό το σημείο, στοχεύουν στη βελτιστοποίηση της επίδοσης κυρίως της κρυφής μνήμης. Ωστόσο, όσο μεγαλώνει το μέγεθος των προβλημάτων, η επίδοση των TLBs γίνεται πιο σημαντική. Αν συμβεί υπερχείλιση του TLB [PHP02], θα μειωθεί δραστικά η συνολική επίδοση του συστήματος. Επομένως, κατά τη βελτιστοποίηση των εφαρμογών, η επίδοση του TLB θα πρέπει να λαμβάνεται υπόψη σε συνδυασμό με την επίδοση της κρυφής μνήμης. Οι Park κλπ στην εργασία [PHP03] εξάγουν ένα κατώτερο όριο για την επίδοση του TLB για συγκεκριμένες ακολουθίες προσπέλασης δεδομένων και αποδεικνύουν ότι οι μη-γραμμικές διατάξεις δεδομένων και οι διατάξεις κατά Morton (στους αναδρομικούς μόνο αλγορίθμους) φτάνουν το όριο αυτό. Οι διατάξεις αυτές με μέγεθος ομάδας δεδομένων ίσο με το μέγεθος σελίδας, ελαχιστοποιούν τον αριθμό των αστοχιών TLB. Λαμβάνοντας υπόψη όλα τα επίπεδα κρυφής μνήμης και το TLB, μπορεί να χρησιμοποιηθεί ένας αλγόριθμος επιλογής ενός στενού πεδίου εντός του οποίου ανήκει το βέλτιστο μέγεθος ομάδας δεδομένων.

Οπωσδήποτε, η αυτόματη εφαρμογή μη γραμμικών διατάξεων δεδομένων σε πραγματικούς μεταγλωττιστές είναι ιδιαίτερα περίπλοκη διαδικασία. Δεν αρκεί να αναγνωρισθεί η βέλτιστη μη-γραμμική διάταξη ομαδοποίησης δεδομένων για έναν συγκεκριμένο πίνακα. Θα πρέπει επιπλέον να παράγεται αυτόματα η αντιστοίχηση από τους πολυδιάστατους δείκτες που ελέγχουν τις επαναλήψεις των βρόχων, στη σωστή θέση της σειριακής μνήμης, όπου βρίσκονται αποθηκευμένα τα ζητούμενα δεδομένα. Οι μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων υπόσχονται μία πραγματικά πολύ καλή επίδοση, αν συνοδεύονται από μία ευφυή μέθοδο υπολογισμού της διεύθυνσης των δεδομένων.

1.1.3 Επιλογή του Βέλτιστου Μεγέθους και Σχήματος Tile

Οι πρώτες εργασίες [MCT96], [WL91] αναζητούσαν ένα μέγεθος tile με το οποίο το σύνολο των δεδομένων που επεξεργάζεται και επαναχρησιμοποιεί το πρόγραμμα σε γειτονικές επαναλήψεις, να χωράει στην κρυφή μνήμη, έτσι ώστε να ελαχιστοποιηθούν κατά το δυνατό οι συγκρούσεις χωρητικότητας. Για να ελαχιστοποιηθεί ταυτόχρονα η επιβάρυνση λόγω του tiling, το μέγεθος των

επιλεγόμενων tiles θα πρέπει να είναι το μέγιστο δυνατό που ικανοποιεί τις παραπάνω συνθήκες.

Οι μεταγενέστερες εργασίες [TFJ94] λαμβάνουν υπόψη και τις αστοχίες σύγκρουσης. Η συγκεκριμένη κατηγορία αστοχιών λαμβάνει χώρα όταν πολλά δεδομένα διεκδικούν την απεικόνισή τους στην ίδια θέση της κρυφής μνήμης, προκαλώντας έτσι εκτοπισμό δεδομένων από την κρυφή μνήμη, ακόμα και αν υπάρχει αρκετός χώρος σε αυτήν για την αποθήκευση του συνόλου των δεδομένων που επαναχρησιμοποιούνται. Κατά συνέπεια, η επιλογή των tiles δεν θα πρέπει να γίνεται μόνο με το κριτήριο της ελαχιστοποίησης των αστοχιών χωρητικότητας, αλλά επιπλέον, λόγω των μεγεθών μεγάλων πινάκων σε επιστημονικές εφαρμογές, θα πρέπει να μειώνονται κατά το δυνατόν οι αστοχίες σύγκρουσης, τόσο μεταξύ των στοιχείων των ίδιων πινάκων, όσο και μεταξύ διαφορετικών πινάκων [CM99], [CM95], [Ess93], [LRW91], [RT99a], [WL91].

Οι παραπάνω εργασίες επιτυγχάνουν μείωση των αστοχιών χωρητικότητας περιορίζοντας το μέγεθος των tiles. Επιλέγουν τέτοια tiles που το σύνολο των δεδομένων που περιέχουν να χωρούν εξ' ολοκλήρου στην κρυφή μνήμη. Για τη μείωση των αστοχιών σύγκρουσης μεταξύ των στοιχείων του ίδιου πίνακα, οι εργασίες [WMC96] και [MHCF98] χρησιμοποιούν ως μοντέλο το αποτελεσματικό μέγεθος της κρυφής μνήμης $q \times C$ $(q < 1)$, αντί του πραγματικού μεγέθους της C. Για τον ίδιο σκοπό οι εργασίες [CM99], [CM95], [LRW91] και [SL01] βρίσκουν συγκεκριμένα μεγέθη και σχήματα tiles που προκαλούν μειωμένο αριθμό αστοχιών σύγκρουσης. Η εργασία [CM95] λαμβάνει επιπλέον υπόψη της το μέγεθος της γραμμής της κρυφής μνήμης. Έτσι, (θεωρώντας -χωρίς βλάβη της γενικότητας- ότι τα στοιχεία των πινάκων αποθηκεύονται κατά στήλες) επιλέγει ως διάσταση της στήλης των πινάκων κάποιο πολλαπλάσιο της γραμμής της κρυφής μνήμης. Οι Lam κλπ στην εργασία τους [LRW91] επιλέγουν τετραγωνικά tiles διάστασης όχι μεγαλύτερη από $\sqrt{\frac{aC}{a+1}}$, όπου $a =$ ο βαθμός συσχέτισης της κρυφής μνήμης. Στην πράξη, η βέλτιστη επιλογή αφορά tiles που περιέχουν δεδομένα τα οποία κατά την αποθήκευσή τους στην κρυφή μνήμη, καλύπτουν ένα μικρό μόνο τμήμα της, συχνά λιγότερο από 10%. Επιπρόσθετα, όσο αυξάνει το μέγεθος της κρυφής μνήμης, μικραίνει το ποσοστό της που καλύπτεται από τα δεδομένα ενός tile.

Το επιθυμητό σχήμα των tiles ορίζεται ρητά σε κάποιες εργασίες όπως τις [Ess93], [CM99], [CM95], [WL91], [WMC96], [LRW91]. Τόσο η [WL91], όσο και η [LRW91], αναζητούν τετραγωνικά tiles. Αντίθετα, οι [CM99], [CM95] και [WMC96] βρίσκουν το βέλτιστο ορθογώνιο tile. Στην [Ess93], ως βέλτιστη λύση προτείνονται πολύ στενόμακρα tiles (αναζητείται ο μέγιστος αριθμός ολόκληρων στηλών του πίνακα που χωρούν στην κρυφή μνήμη). Το σχήμα των tiles και το ποσοστό της κρυφής μνήμης που χρησιμοποιείται από αυτά είναι δύο σημαντικοί παράγοντες που επηρεάζουν την επίδοση των αλγορίθμων. Για το λόγο αυτό λαμβάνονται υπόψη στις περισσότερες εργασίες, είτε έμμεσα μέσω του μοντέλου που χρησιμοποιούν, είτε άμεσα επιλέγοντας συγκεκριμένα σχήματα tiles. Τα πολύ πλατιά tiles είναι πολύ πιθανό να προκαλέσουν υπερχείλιση του TLB. Από την άλλη πλευρά, τα πολύ μακριά ή τα τετραγωνικά tiles μπορεί να έχουν χαμηλό ποσοστό χρησιμοποίησης της κρυφής μνήμης. Εκτός από τις στατικές τεχνικές υπολογισμού του μεγέθους των tiles, στην εργασία [KKO00] χρησιμοποιούνται επαναληπτικοί αλγόριθμοι προσέγγισης της βέλτιστης λύσης, οι οποίοι εκτελούνται στο χρόνο μεταγλώττισης. Η τεχνική αυτή μπορεί να επι-

τύχει πολύ σημαντική επιτάχυνση στο χρόνο εκτέλεσης των προγραμμάτων, όμως το μειονέκτημά της έγκειται στον πολύ μεγάλο χρόνο μεταγλώττισης, που απαιτείται για την παραγωγή και την υποθετική εκτέλεση (profile) πολλών εκδόσεων του αρχικού προγράμματος.

Δυστυχώς, η επίδοση των μετασχηματισμένων προγραμμάτων όπως διαμορφώνονται μετά από το tiling, δεν είναι σταθερή, όπως αποδεικνύουν οι εργασίες [PNDN99], [RT99a]. Αυτή η αστάθεια επίδοσης οφείλεται στα λεγόμενα παθολογικά μεγέθη πινάκων, όπου οι διαστάσεις των πινάκων είναι ίσες ή σχεδόν ίσες με κάποια δύναμη του δύο. Στο σημείο αυτό οι αστοχίες σύγκρουσης μεταξύ των στοιχείων των ίδιων πινάκων είναι ιδιαίτερα επίφοβες. Το παραγέμισμα των πινάκων [RT98a], [RT98b], [SL01] είναι μία τεχνική με την οποία, αυξάνοντας το μέγεθος των πινάκων, αποτρέπεται η δημιουργία αυτών των παθολογικών περιπτώσεων. Η τεχνική αυτή, παρόλο που σπαταλάει ένα μέρος του ωφέλιμου χώρου της μνήμης, τελικά σταθεροποιεί την επίδοση του προγράμματος. Τελικά, το ποσοστό της κρυφής μνήμης που χρησιμοποιείται από τα tiles στην περίπτωση των παραγεμισμένων πινάκων είναι αρκετά καλύτερο, καθώς έτσι μπορούν να αποφευχθούν τα πολύ μικρά tiles [RT99a]. Οι περισσότερες εργασίες διερευνούν και προτείνουν κάποιο συνδυασμό του μετασχηματισμού tiling και του παραγεμίσματος πινάκων, για να επιτύχουν καλή τοπικότητα δεδομένων και ταυτόχρονα σταθερότητα στην επίδοση των προγραμμάτων [HK04], [RT98b], [PNDN99]. Μία εναλλακτική μέθοδος για την αποφυγή των αστοχιών σύγκρουσης είναι η αντιγραφή ολόκληρων των tiles σε προσωρινούς καταχωρητές. Επιπλέον, το πρόγραμμα θα πρέπει να χρησιμοποιεί και να πραγματοποιεί την ανανέωση του περιεχομένου των δεδομένων απευθείας στους καταχωρητές αυτούς [Ess93], [LRW91], [TGJ93]. Τα δεδομένα των tiles στους καταχωρητές αποθηκεύονται σειριακά, σε συνεχόμενες θέσεις, και επομένως αποτρέπονται οι αστοχίες σύγκρουσης μεταξύ των στοιχείων του ίδιου πίνακα. Επιπλέον, αν οι καταχωρητές που περιέχουν τα στοιχεία διαφορετικών πινάκων ενοποιηθούν, δεσμεύοντας διαδοχικές θέσεις μνήμης, τότε μπορούν να αποτραπούν και οι αστοχίες σύγκρουσης μεταξύ των στοιχείων διαφορετικών πινάκων. Μέσω της τεχνικής της αντιγραφής (copying), η εργασία [LRW91] επιτυγχάνει να αξιοποιεί το μέγεθος ολόκληρης της κρυφής μνήμης, επιλέγοντας tiles μεγέθους $\sqrt{C} \times \sqrt{C}$. Ωστόσο, η επιβάρυνση στο χρόνο εκτέλεσης (δεδομένου ότι η αντιγραφή πραγματοποιείται κατά τη διάρκεια της εκτέλεσης του προγράμματος) μπορεί να γίνει απαγορευτική, αν η αντιγραφή χρειάζεται να γίνει παραπάνω από μία φορά για κάθε πίνακα.

Η υπερχείλιση του TLB είναι ένας ακόμη κρίσιμος παράγοντας που πρέπει να λαμβάνεται υπόψη, αφού κάθε αστοχία του TLB κοστίζει πολύ περισσότερους κύκλους ρολογιού από μία αστοχία της L1 κρυφής μνήμης. Επομένως, οι αστοχίες του TLB μπορούν να προκαλέσουν τραγικά πολλούς κύκλους καθυστέρησης. Ο μετασχηματισμός tiling μπορεί να εφαρμοστεί σε πολλαπλά επίπεδα της ιεραρχίας μνήμης. Οι συναρτήσεις που λαμβάνουν υπόψη τους μόνο ένα επίπεδο της ιεραρχίας, μπορούν να επιτύχουν ικανοποιητικά αποτελέσματα σε ότι αφορά τις αστοχίες του συγκεκριμένου επιπέδου, όχι όμως και τη συνολική επίδοση. Η εφαρμογή του μετασχηματισμού tiling πολλαπλών επιπέδων μπορεί να επιτύχει ικανοποιητική αξιοποίηση της τοπικότητας δεδομένων στα πολλαπλά επίπεδα της ιεραρχίας ταυτόχρονα. Τέτοιου είδους βελτιστοποιήσεις

προτείνονται στις εργασίες [CM95], [HK04] και σε μία έκδοση της [MHCF98], όπου χρησιμοποιούνται συναρτήσεις υπολογισμού του κόστους εκτέλεσης στα πολλαπλά επίπεδα της ιεραρχίας μνήμης. Η ελαχιστοποίηση του κόστους των συναρτήσεων πολλαπλών επιπέδων επιτυγχάνεται με εξισορρόπηση του σχετικού κόστους των αστοχιών της κρυφής μνήμης και των TLBs. Ο βέλτιστος μετασχηματισμός tiling θα πρέπει να ικανοποιεί τους περιορισμούς χωρητικότητας τόσο της κρυφής μνήμης όσο και των TLBs.

Γενικότερα, οι περισσότεροι αλγόριθμοι αναζητούν το μέγιστο δυνατό μέγεθος tile με το οποίο παράγεται ο ελάχιστος δυνατός αριθμός αστοχιών χωρητικότητας, εξαλείφονται οι αστοχίες σύγκρουσης μεταξύ στοιχείων του ίδιου πίνακα και, κατά το δυνατό, περιορίζονται οι αστοχίες σύγκρουσης μεταξύ στοιχείων διαφορετικών πινάκων. Σε πολλές περιπτώσεις δεν είναι δυνατόν να συνδυαστεί το υψηλό ποσοστό χρησιμοποίησης της κρυφής μνήμης [SL99] με έναν μικρό αριθμό αστοχιών. Κατά συνέπεια, κάθε αλγόριθμος αποτελεί έναν διαφορετικό τρόπο προσέγγισης του προβλήματος, όπου εξισορροπείται με διαφορετικό τρόπο και πιθανότατα σε διαφορετικό σημείο η χρησιμοποίηση της κρυφής μνήμης και ο αριθμός των αστοχιών.

Σημαντική, επίσης, δουλειά έχει γίνει στην ποσοτικοποίηση του συνολικού αριθμού των αστοχιών σύγκρουσης στις εργασίες [CM99], [GMM98], [GMM99], [HKN99], [Ver03] και [FST91]. Η συμπεριφορά της κρυφής μνήμης είναι ιδιαίτερα δύσκολο να αναλυθεί. Πολύ μικρές αλλαγές παραμέτρων που την αφορούν μπορεί να προκαλέσουν δυσανάλογα μεγάλες αλλαγές στο ρυθμό παραγωγής των αστοχιών [TFJ94], γεγονός που αποδεικνύει την ασταθή φύση των κρυφών μνημών. Συνήθως, η αξιολόγηση της συμπεριφοράς των κρυφών μνημών γινόταν μέσω προσομοιώσεων. Παρόλο που τα αποτελέσματα είναι ακριβή, ο χρόνος που απαιτείται για να τα εξάγουμε είναι πάντοτε πολύ μεγαλύτερος από το χρόνο εκτέλεσης των προγραμμάτων που προσομοιώνουμε. Για να ξεπεράσουμε τα προβλήματα αυτά, έχουν πλέον αναπτυχθεί αναλυτικά μοντέλα της συμπεριφοράς της κρυφής μνήμης συνδυασμένα με ευριστικούς αλγόριθμους, τα οποία καθοδηγούν τους μεταγλωττιστές στη βελτιστοποίηση των προγραμμάτων [GMM98], [GMM99], [RT98a] και [WL91], ή μελετούν την επίδοση της κρυφής μνήμης σε συγκεκριμένους τύπους αλγορίθμων, και ιδιαίτερα εκείνους όπου έχει εφαρμοστεί μετασχηματισμός tiling [CM99], [HKN99], [LRW91], [Mit00] και [Ver03].

Οι βελτιστοποιήσεις κώδικα, όπως η επιλογή του βέλτιστου μεγέθους tile, οι οποίες επιλέγονται με τη βοήθεια των τεχνικών πρόβλεψης του αριθμού των αστοχιών, απαιτούν έναν πραγματικά ακριβή υπολογισμό της συμπεριφοράς του κώδικα των προγραμμάτων. Επιπρόσθετα, θα πρέπει να λαμβάνεται υπόψη η πολυπλοκότητα που εισάγεται στον κώδικα λόγω του μετασχηματισμού tiling καθώς και οι επιπλέον λάθος προβλέψεις της απόφασης των εντολών άλματος. Ο αριθμός των αστοχιών είναι σε γενικές γραμμές μικρότερος στους κώδικες όπου έχει εφαρμοστεί μετασχηματισμός tiling. Το γεγονός αυτό δίνει μεγαλύτερη βαρύτητα σε λάθη πρόβλεψης, που σε άλλη περίπτωση θα φαίνονταν ασήμαντα σε σύγκριση με τις υπόλοιπες αιτίες που εισάγουν καθυστέρηση στην εκτέλεση του προγράμματος. Για το λόγο αυτό, η επιλογή του βέλτιστου συνδυασμού μετασχηματισμών θα πρέπει να λαμβάνει υπόψη της το συνδυασμό της θεωρητικής ανάλυσης της

συμπεριφοράς της κρυφής μνήμης, καθώς και των αποτελεσμάτων της πραγματικής εκτέλεσης των προγραμμάτων και των προσομοιώσεων.

Οι προηγούμενες προσεγγίσεις θεωρούσαν ότι η διάταξη αποθήκευσης των δεδομένων στη μνήμη ήταν γραμμική, αν και μελετούσαν κώδικες όπου είχε εφαρμοστεί μετασχηματισμός tiling. Επεδίωκαν την εύρεση του βέλτιστου μεγέθους tile, το οποίο πετύχαινε έναν συμβιβασμό μεταξύ της ελαχιστοποίησης των αστοχιών χωρητικότητας και των αστοχιών σύγκρουσης. Σε όλες τις εργασίες το μέγεθος του tile δεν ξεπερνούσε τη χωρητικότητα της κρυφής μνήμης (συνήθως αναφερόντουσαν στην L1 κρυφή μνήμη) ή της κρυφής μνήμης και του TLB ταυτόχρονα. Ωστόσο, όπως έχει αποδειχτεί στη βιβλιογραφία και θα παρουσιάσουμε στο κεφάλαιο 3, οι γραμμικές αυτές διατάξεις αποθήκευσης δεν ευνοούν την τοπικότητα δεδομένων στους κώδικες όπου έχει εφαρμοστεί μετασχηματισμός tiling, και προκαλούν έναν μεγάλο αριθμό αστοχιών σύγκρουσης, αυξάνοντας την επιβάρυνση που εισάγει το σύστημα μνήμης.

1.2 Συμβολή της Διατριβής

Ένα αναλυτικό μοντέλο της συμπεριφοράς των επιπέδων μνήμης και του συνόλου των επιδράσεων άλλων παραγόντων στην συνολική επίδοση κάθε εφαρμογής, θα μπορούσε να τροφοδοτήσει τους μεταγλωττιστές με πληροφορίες ή θα έδινε κατευθυντήριες γραμμές στους προγραμματιστές σχετικά με τις καταλληλότερες βελτιστοποιήσεις κώδικα (ανά περίπτωση αρχιτεκτονικής συστήματος). Ωστόσο, η καταγραφή των παραγόντων και η μοντελοποίηση αυτή αποτελούν διαδικασίες πραγματικά πολύπλοκες, ιδιαίτερα σε ότι αφορά το ευθύ συσχετισμό τους με συγκεκριμένες βελτιστοποιήσεις κώδικα, με στόχο την αυτόματη παραγωγή κώδικα. Η παρούσα διατριβή συμβάλλει στη αυτοματοποίηση της βελτιστοποίησης κώδικα, επικεντρώνοντας το ενδιαφέρον της στους μη-γραμμικούς μετασχηματισμούς δεδομένων αριθμητικών εφαρμογών. Ο αλγόριθμος βελτιστοποίησης λαμβάνει υπόψη του τις παραμέτρους της ιεραρχίας μνήμης κάθε συστήματος, προκειμένου να προσδιορίσει το βέλτιστο μέγεθος των ομάδων δεδομένων που επεξεργάζεται κάθε εφαρμογή σε γειτονικές εντολές.

Η συμβολή της εργασίας αυτής εντοπίζεται κυρίως σε τρία θέματα:

- Προτείνεται ένα σχήμα γρήγορης δεικτοδότησης των μη-γραμμικών διατάξεων ομαδοποίησης δεδομένων. Η συγκεκριμένη δεικτοδότηση αποτελεί το απαραίτητο συνοδευτικό κάθε μη-γραμμικού μετασχηματισμού δεδομένων, διαφορετικά, για τον εντοπισμό των ζητούμενων δεδομένων, απαιτείται ένας μεγάλος όγκος χρονοβόρων υπολογισμών. Ο συνδυασμός των μη-γραμμικών μετασχηματισμών ομαδοποίησης δεδομένων, με την προτεινόμενη γρήγορη δεικτοδότηση και βελτιστοποιήσεις κώδικα, και ιδιαίτερα το μετασχηματισμό tiling), αποδείχτηκε ιδιαίτερα αποδοτικός στις περιπτώσεις αριθμητικών εφαρμογών που περιέχουν μεγάλο όγκο υπολογισμών. Ο αλγόριθμος που αναπτύσσεται μπορεί να ενσωματωθεί σε κάποιο στατικό εργαλείο παραγωγής κώδικα, όπως οι μεταγλωττιστές.

- Προτείνεται μία απλή ευριστική τεχνική για την εύκολη εύρεση του βέλτιστου μεγέθους tile, για μετασχηματισμένους κώδικες όπου έχει εφαρμοστεί μόνο ένα επίπεδο tiling. Πρόκειται για το σημείο σύγκλισης των παραγόντων που επηρεάζουν ή καθορίζουν την επίδοση των πολλαπλών επιπέδων της ιεραρχίας μνήμης.
- Μελετούνται οι επιδράσεις διαφόρων τεχνικών βελτιστοποίησης κώδικα ανά περίπτωση αρχιτεκτονικής συστήματος. Η εργασία αυτή αντιστοιχεί ποιες βελτιστοποιήσεις συνταιριάζονται αποδοτικά με ποια χαρακτηριστικά υλικού. Τέλος, επισημαίνονται οι αδυναμίες του υλικού, και εφευρίσκονται οι ακολουθίες εντολών που μπορούν να επιτύχουν υψηλό ρυθμό εκτέλεσης, αποφεύγοντας τα σημεία στενωπού τη σωλήνωσης.

1.3 Οργάνωση Διατριβής

Η παρούσα διατριβή οργανώνεται ως εξής:

Το κεφάλαιο 2 περιγράφει το γενικό πλαίσιο της ιεραρχίας μνημών, τον τρόπο οργάνωσής της και τα χαρακτηριστικά των κρυφών μνημών. Παρουσιάζεται, επίσης, η ορολογία που αφορά τις τεχνικές βελτιστοποίησης κώδικα. Γίνεται ιδιαίτερη αναφορά στους μετασχηματισμούς βρόχων και τον μετασχηματισμό υπερκόμβων (tiling).

Το κεφάλαιο 3 παρουσιάζει το υπόβαθρο που στηρίζει το κύριο σώμα της παρούσας διατριβής, που είναι οι προϋπάρχοντες μη-γραμμικοί μετασχηματισμοί δεδομένων. Στη συνέχεια, περιγράφονται με λεπτομέρειες οι προτεινόμενοι μη-γραμμικοί μετασχηματισμοί ομαδοποίησης δεδομένων, οι οποίοι συνδυάζονται με ένα γρήγορο σχήμα δεικτοδότησης των πινάκων. Αναπτύσσεται ένα σύστημα δυαδικών μασκών, το οποίο χρησιμοποιείται για την εύρεση των στοιχείων των αρχικών πολυδιάστατων πινάκων, μέσα στους τελικούς μονοδιάστατους μετασχηματισμένους πίνακες.

Το κεφάλαιο 4 δίνει μία αναλυτική περιγραφή της συμπεριφοράς των κρυφών μνημών και του TLB δεδομένων, όταν εκτελούνται μετασχηματισμένοι κώδικες που περιέχουν πίνακες αποθηκευμένους στη μνήμη σύμφωνα με τους μη-γραμμικούς μετασχηματισμού ομαδοποίησης δεδομένων. Υπολογίζεται το βέλτιστο μέγεθος tile, που αξιοποιεί στο μέγιστο την χωρητικότητα των κρυφών μνημών και ταυτόχρονα αποτρέπει την παραγωγή υπερβολικά μεγάλου αριθμού από αστοχίες σύγκρουσης. Το βέλτιστο μέγεθος tile είναι το σημείο σύγκλισης των παραγόντων που επηρεάζουν την επίδοση του συστήματος μνήμης (αστοχίες κρυφών μνημών και TLB, καθώς και αστοχίες λόγω λάθος πρόβλεψης διακλάδωσης).

Το κεφάλαιο 5 περιγράφει την αρχιτεκτονική της Ταυτόχρονης Πολυνημάτωσης (Simultaneous Multithreading - SMT) και μελετά θέματα που σχετίζονται με την υλοποίηση πολυνηματικών εφαρμογών και το συγχρονισμό των ταυτόχρονα εκτελούμενων νημάτων. Επίσης, αναζητά τα όρια της επίδοσης παράλληλα εκτελούμενων ομοιογενών (σε ότι αφορά τη σύσταση των εντολών) νημάτων.

Το κεφάλαιο 6 παρουσιάζει τα αποτελέσματα πραγματικών πειραματικών μετρήσεων και προσομοιώσεων, αποτιμώντας τους προτεινόμενους από την παρούσα διατριβή μη-γραμμικούς μετα-

σχηματισμούς ομαδοποίησης δεδομένων (τους συγκρίνουμε με τους προγενέστερους μη-γραμμικούς μετασχηματισμούς δεδομένων). Επιπλέον, αποτιμάται η τεχνική επιλογής του βέλτιστου μεγέθους tile και η επίδοση της τεχνικής ταυτόχρονης πολυνημάτωσης σε βελτιστοποιημένους κώδικες.

Τέλος, το κεφάλαιο **??** δίνει τις τελικές παρατηρήσεις και τα συμπεράσματα.

1.4 Δημοσιεύσεις

ΔΙΕΘΝΗ ΣΥΝΕΔΡΙΑ

- Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis and Nectarios Koziris. Exploring the Capacity of a Modern SMT Architecture to Deliver High Scientific Application Performance. In *Proc. of the 2006 International Conference on High Performance Computing and Communications (HPCC-06)*, pages , Munich, Germany, Sep 2006. Lecture Notes in Computer Science.

- Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis and Nectarios Koziris. Exploring the Performance Limits of Simultaneous Multithreading for Scientific Codes. In *Proc. of the 2006 International Conference on Parallel Processsing (ICPP-06)*, pages , Columbus, OH, Aug 2006. IEEE Computer Society Press.

- Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis and Nectarios Koziris. Tuning Blocked Array Layouts to Exploit Memory Hierarchy in SMT Architectures. In *Proc. of the 10th Panhellenic Conference in Informatics*, pages , Volos, Greece, Nov 2005. Lecture Notes in Computer Science.

- Evangelia Athanasaki, Nectarios Koziris and Panayiotis Tsanakas. A Tile Size Selection Analysis for Blocked Array Layouts. In *Proc. of the 9-th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9), In conjuction with the 11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, pages 70–80, San Francisco, CA, Feb 2005. IEEE Computer Society Press.

- Evangelia Athanasaki and Nectarios Koziris. Fast Indexing for Blocked Array Layouts to Improve Multi-Level Cache Locality. In *Proc. of the 8-th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-8), In conjuction with the 10th International Symposium on High-Performance Computer Architecture (HPCA-10)*, pages 109–119, Madrid, Spain, Feb 2004. IEEE Computer Society Press.

- Evangelia Athanasaki and Nectarios Koziris. Improving Cache Locality with Blocked Array Layouts. In *Proceedings of the 12-th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP'04)*, pages 308–317, A Coruna, Spain, Feb. 2004. IEEE Computer Society Press.

- Evangelia Athanasaki and Nectarios Koziris. Blocked Array Layouts for Multilevel Memory Hierarchies. In *Proceedings of the 9th Panhellenic Conference in Informatics*, pages 193–207, Thessaloniki, Greece, Nov. 2003.

ΔΙΕΘΝΗ ΠΕΡΙΟΔΙΚΑ

- Evangelia Athanasaki and Nectarios Koziris. Fast Indexing for Blocked Array Layouts to Reduce Cache Misses. *International Journal of High Performance Computing and Networking (IJHPCN)*, 3(5/6): 417–433, 2005.

- Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis and Nectarios Koziris. Exploring the Performance Limits of Simultaneous Multithreading for Memory Intensive Applications. *The Journal of Supercomputing*, submitted.

ΚΕΦΑΛΑΙΟ 2

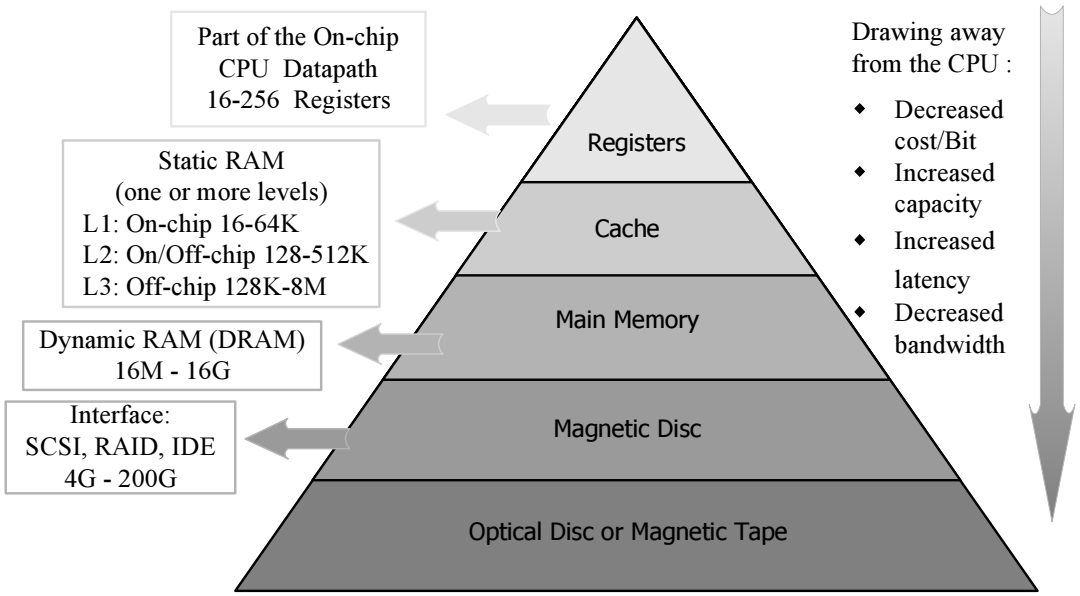
Βασικές Έννοιες

Το κεφάλαιο αυτό εισάγει τις βασικές έννοιες της ιεραρχίας μνήμης, σε ότι αφορά τον τρόπο οργάνωσής της. Επιπλέον, επεξηγούνται οι έννοιες της επαναχρησιμοποίησης και της τοπικότητας δεδομένων, οι οποίες είναι θεμελιώδεις για την κατανόηση της συμπεριφοράς του συστήματος μνήμης. Τέλος, δίνονται οι ορισμοί εννοιών που θα χρησιμοποιηθούν σε επόμενα κεφάλαια.

2.1 Η Ιεραρχία Μνήμης

Το σύστημα μνήμης οργανώνεται ιεραρχικά σε πολλαπλά επίπεδα (σχήμα 2.1). Στο ένα άκρο της ιεραρχίας βρίσκεται ο αποθηκευτικός χώρος του δίσκου, το οποίο έχει υψηλή πυκνότητα αποθήκευσης, χαμηλό κατασκευαστικό κόστος ανά bit λέξης μνήμης, και σχετικά μεγάλο χρόνο προσπέλασης. Στο άλλο άκρο βρίσκονται οι καταχωρητές του επεξεργαστή, οι οποίοι είναι λίγοι σε αριθμό, έχουν υψηλό κατασκευαστικό κόστος ανά bit λέξης μνήμης, αλλά επιτρέπουν πολύ γρήγορη πρόσβαση στα δεδομένα που περιέχουν. Επομένως, καθώς μετακινούμαστε από το δίσκο προς τους καταχωρητές, αυξάνει το κατασκευαστικό κόστος, μικραίνει η πυκνότητα αποθήκευσης, αλλά μικραίνει και ο χρόνος προσπέλασης. Τα επίπεδα της ιεραρχίας συνήθως είναι υποσύνολα των αμέσως μεγαλύτερων επιπέδων. Δηλαδή, όλα οι λέξεις που βρίσκονται σε ένα επίπεδο, πρέπει να είναι αποθηκευμένες και στο επόμενο ιεραρχικά επίπεδο. Ο στόχος είναι να αναπτυχθεί ένα σύστημα μνήμης με κόστος σχεδόν όσο αυτό του χαμηλότερου - πιο απομακρυσμένου από τη μνήμη - και φθηνότερου επιπέδου και ταχύτητα σχεδόν όση του ταχύτερου επιπέδου.

Κρυφή μνήμη (cache) είναι ένα υψηλής ταχύτητας σύστημα μνήμης, που περιέχει ένα μικρό υποσύνολο των λέξεων της κύριας μνήμης. Βρίσκεται μεταξύ του επεξεργαστή και της κύριας μνήμης και εκεί απευθύνεται μία αναζήτηση διεύθυνσης μνήμης, αν δεν ικανοποιηθεί από τους καταχωρητές του επεξεργαστή. Δεδομένου ότι η κρυφή μνήμη αποκρίνεται πολύ γρηγορότερα κατά τις αιτήσεις αναζήτησης δεδομένων, σε σχέση με την κύρια μνήμη, μπορεί να επιταχύνει την επίδοση του συστήματος μνήμης, αν τα δεδομένα και οι εντολές που χρησιμοποιούνται συχνά μπορούν να τοποθετηθούν εκεί, προκειμένου να αποστέλλονται γρήγορα στον επεξεργαστή, όποτε



Σχήμα 2.1: Η ιεραρχία μνήμης

αυτός τα χρειάζεται.

Η κρυφή μνήμη βελτιώνει την επίδοση του συστήματος μνήμης, και γενικότερα του υπολογιστικού συστήματος, αξιοποιώντας την *τοπικότητα* αναφοράς των προγραμμάτων σε δεδομένα:

- Σύμφωνα με τη *χρονική τοπικότητα δεδομένων (temporal locality)*, αν ο επεξεργαστής χρησιμοποιήσει (διαβάσει ή γράψει) μία λέξη μνήμης, είναι πολύ πιθανό στο άμεσο μέλλον να την επαναχρησιμοποιήσει. Για το λόγο αυτό, οι χρησιμοποιούμενες λέξεις τοποθετούνται στην κρυφή μνήμη, προκειμένου να υπάρχει η δυνατότητα γρήγορης προσπέλασής τους από τον επεξεργαστή κατά την επόμενη αναφορά σε αυτές.

- Σύμφωνα με τη *χωρική τοπικότητα δεδομένων (spatial locality)*, αν ο επεξεργαστής χρησιμοποιήσει μία λέξη, είναι πολύ πιθανό στο άμεσο μέλλον να χρησιμοποιήσει τις γειτονικές σε αυτήν λέξεις, και ειδικότερα εκείνες που ανήκουν στην ίδια ομάδα λέξεων (block).

Κατά την αναφορά σε μία διεύθυνση μνήμης (λέξη μνήμης), αν το ζητούμενο δεδομένο είναι αποθηκευμένο στην κρυφή μνήμη, τότε πρόκειται για μία *επιτυχία της κρυφής μνήμης (cache hit)*. Αν ο επεξεργαστής δε βρει το ζητούμενο δεδομένο στην κρυφή μνήμη, τότε έχουμε μία *αστοχία της κρυφής μνήμης (cache miss)*. Στην τελευταία περίπτωση, μία προκαθορισμένου μεγέθους ομάδα λέξεων (memory block) (η οποία συμπεριλαμβάνει και το ζητούμενο δεδομένο), ανακτάται από την κύρια μνήμη και αποθηκεύεται στην κρυφή μνήμη. Η ομάδα αυτή λέξεων είναι το ελάχιστο ποσό δεδομένων που μπορεί να μετακινηθεί από το ένα επίπεδο μνήμης στο άλλο. Στην περίπτωση των κρυφών μνημών, η ομάδα λέξεων (memory block) αντιστοιχεί και ονομάζεται εναλλακτικά *γραμμή δεδομένων της κρυφής μνήμης (cache line)*. Δηλαδή, ενώ οι αναφορές σε διευθύνσεις μνήμης ισοδυναμούν ουσιαστικά με αναφορές σε λέξεις μνήμης, η αναζήτησή τους γίνεται μέσα

στην ομάδα λέξεων στην οποία ανήκουν. Αν η ζητούμενη λέξη βρίσκεται στην κρυφή μνήμη, ολόκληρη η αντίστοιχη ομάδα λέξεων θα βρίσκεται αποθηκευμένη σε αυτήν, και η ζητούμενη λέξη θα καταλαμβάνει μία από τις θέσεις μνήμης που δεσμεύονται από την ομάδα λέξεων στην οποία ανήκει.

Ο χρόνος που απαιτείται για την επίλυση μίας αστοχίας δεδομένων εξαρτάται τόσο από την αρχική καθυστέρηση ανάγνωσης από τη μνήμη (latency) όσο και από το εύρος ζώνης, δηλαδή το ρυθμό μεταφοράς δεδομένων από τη μνήμη στον επεξεργαστή (bandwidth). Η αρχική καθυστέρηση καθορίζει το χρόνο που μεσολαβεί μεταξύ της αίτησης ανάκτησης μιας ομάδας λέξεων και της άφιξης στο υπό εξέταση επίπεδο μνήμης της πρώτης λέξης της συγκεκριμένης ομάδας. Το εύρος ζώνης καθορίζει το ρυθμό ανάκτησης των υπόλοιπων λέξεων της ομάδας. Μία αστοχία της κρυφής μνήμης αναγκάζει τον επεξεργαστή να αναβάλλει τις λειτουργίες του (να πραγματοποιήσει stall), μέχρι αυτή να επιλυθεί (αυτό ισχύει στην περίπτωση των επεξεργαστών που εκτελούν τις εντολές σειριακά -in order).

2.2 Οι Κρυφές Μνήμες

Οι κρυφές μνήμες σχεδιάστηκαν για να κρατούν αποθηκευμένα τα πιο συχνά χρησιμοποιούμενα δεδομένα. Βρίσκονται μεταξύ του επεξεργαστή και της κύριας μνήμης, για να ικανοποιούν τις αιτήσεις του επεξεργαστή για ανάκτηση δεδομένων. Ωστόσο, δεν είναι πάντα εφικτή η εύρεση των ζητούμενων δεδομένων σε αυτές, οπότε συμβαίνει, όπως προαναφέρθηκε, μία αστοχία δεδομένων. Οι αστοχίες δεδομένων χωρίζονται σε τρεις κατηγορίες:

- *Υποχρεωτικές αστοχίες (compulsory misses)*: Κατά την πρωταρχική αναφορά ενός προγράμματος σε ένα από τα στοιχεία μίας ομάδας λέξεων, τα δεδομένα ομάδας αυτής δεν είναι δυνατόν να βρίσκονται αποθηκευμένα στην κρυφή μνήμη. Προκειμένου να προσπελαστούν από τον επεξεργαστή, πρέπει να μεταφερθούν για πρώτη φορά στην κρυφή μνήμη. Στην περίπτωση αυτή, πρόκειται για μία υποχρεωτική αστοχία δεδομένων ή, όπως εναλλακτικά ονομάζεται, αστοχία αρχικοποίησης ή πρωταρχικής αναφοράς (cold-start ή first-reference miss).

- *Αστοχίες Χωρητικότητας (capacity misses)*: Αν η κρυφή μνήμη είναι πολύ μικρή (σε χωρητικότητα) για να κρατήσει το σύνολο των δεδομένων που επαναχρησιμοποιεί ένα πρόγραμμα, τότε κατά την επαναληπτική προσπέλαση δεδομένων, θα πρέπει αυτά να ξανα-μεταφέρονται στην κρυφή μνήμη από τα κατώτερα ιεραρχικά επίπεδα. Στην περίπτωση αυτή έχουμε αστοχίες χωρητικότητας.

  Για τη μείωση των αστοχιών χωρητικότητας, η προφανής λύση θα ήταν η αύξηση της συνολικής χωρητικότητας των κρυφών μνημών. Ασφαλώς, μία μεγάλου μεγέθους κρυφή μνήμη έχει υψηλό κατασκευαστικό κόστος. Το βασικό, όμως, μειονέκτημα είναι ότι αυξάνει ο

χρόνος προσπέλασής της. Για το λόγο αυτό, κατασκευάζονται μεγάλης χωρητικότητας κρυφές μνήμες κυρίως στο δεύτερο και το τρίτο επίπεδο της ιεραρχίας του συστήματος μνημών, οι οποίες τοποθετούνται εκτός του ολοκληρωμένου κυκλώματος του επεξεργαστή.

- *Αστοχίες σύγκρουσης (conflict misses)*: Οι αστοχίες σύγκρουσης δεδομένων είναι αποτέλεσμα του τρόπου οργάνωσης της κρυφής μνήμης, δηλαδή, του μηχανισμού τοποθέτησης των νέων δεδομένων που εισάγονται προς αποθήκευση σε αυτήν. Μία αστοχία σύγκρουσης μπορεί να συμβεί ακόμα και στην περίπτωση που υπάρχει διαθέσιμος χώρος στην κρυφή μνήμη. Συμβαίνει, όταν περισσότερες από μία γραμμές δεδομένων διεκδικούν την ίδια θέση μέσα στην κρυφή μνήμη.

  Τις αστοχίες σύγκρουσης τις διαχωρίζουμε σε αστοχίες που προκαλούνται λόγω σύγκρουσης στοιχείων, τα οποία προέρχονται από τον ίδιο πίνακα (self conflict misses) και σε αστοχίες που οφείλονται σε συγκρούσεις μεταξύ στοιχείων διαφορετικών πινάκων (cross-conflict misses).

2.3 Οργάνωση των Κρυφών Μνημών

Ανάλογα με τον τρόπο επιλογής της θέσης αποθήκευσης μίας ομάδας λέξεων στην κρυφή μνήμη, έχουμε έναν διαφορετικό τρόπο οργάνωσης:

- *Ευθείας απεικόνισης (direct mapped)* κρυφή μνήμη: Μία συγκεκριμένη ομάδα λέξεων μπορεί να τοποθετηθεί σε μία και μοναδική θέση. Η θέση αυτή καθορίζεται από τη διεύθυνση της ομάδας λέξεων ως εξής:

  (θέση απεικόνισης μέσα στην κρυφή μνήμη) = (διεύθυνση της ομάδας λέξεων) MOD (συνολικός αριθμός ομάδων λέξεων που χωρούν στην κρυφή μνήμη)

  Η υλοποίηση αυτού του τρόπου οργάνωσης είναι εύκολη στην πράξη και επεκτάσιμη τόσο για μεγάλα μεγέθη κρυφών μνήμων όσο και για γρήγορα ρολόγια. Ωστόσο, το μειονέκτημά του είναι ότι δεν μπορεί να αποτρέψει τις πολλαπλές συγκρούσεις δεδομένων που επαναχρησιμοποιούνται, όταν δύο ή περισσότερες ομάδες λέξεων διεκδικούν την ίδια θέση μέσα στην κρυφή μνήμη.

- *Πλήρους Συσχέτισης (fully associative)* κρυφή μνήμη: Στην περίπτωση αυτή κάθε ομάδα λέξεων μπορεί να τοποθετηθεί οπουδήποτε μέσα στην κρυφή μνήμη. Επομένως, δεν υπάρχουν πλέον αστοχίες λόγω σύγκρουσης δεδομένων, παρά μόνο αστοχίες χωρητικότητας και αρχικοποίησης. Το μειονέκτημα, όμως, είναι ότι λόγω των πολλαπλών πιθανών θέσεων τοποθέτησης μίας ομάδας λέξεων στην κρυφή μνήμη, ο πολύπλοκος μηχανισμός αναζήτησης, εκτός από ακριβός κατασκευαστικά, είναι και πολύ αργός, τόσο που δεν μπορεί να ακολουθήσει την αύξηση της συχνότητας ρολογιού του επεξεργαστή.

- *Συσχέτισης Συνόλου(set associative)* κρυφή μνήμη: Όταν μία ομάδα λέξεων μπορεί να τοποθετηθεί οπουδήποτε μέσα σε ένα περιορισμένο σύνολο θέσεων της κρυφής μνήμης, τότε πρόκειται για κρυφή μνήμη συσχέτισης συνόλου. Επομένως, η ομάδα λέξεων αντιστοιχίζεται πρώτα με το σύνολο της κρυφής μνήμης, σύμφωνα με τη διεύθυνσή της, και στη συνέχεια μπορεί να αποθηκευθεί οπουδήποτε μέσα στο σύνολο αυτό. Η αντιστοίχιση ομάδας λέξεων-συνόλου γίνεται ως εξής:

  (σύνολο απεικόνισης στην κρυφή μνήμη) = (διεύθυνση της ομάδας λέξεων) MOD (αριθμός συνόλων που υπάρχουν στην κρυφή μνήμη)

  Αν κάθε σύνολο μίας κρυφής μνήμης χωράει N διαφορετικές ομάδες λέξεων, τότε η κρυφή μνήμη ονομάζεται *συσχέτισης N-δρόμων (N-way set associative)*. Έτσι, μία κρυφή μνήμη ευθείας απεικόνισης είναι στην πραγματικότητα κρυφή μνήμη συσχέτισης ενός δρόμου. Επιπλέον, μία πλήρως συσχετιστική κρυφή μνήμη με συνολική χωρητικότητα M ομάδων λέξεων, είναι κρυφή μνήμη συσχέτισης M-δρόμων. Τέλος, σε ότι αφορά την επίδοση των κρυφών μνημών, μία κρυφή μνήμη συσχέτισης οκτώ δρόμων επιφέρει ουσιαστικά το ίδιο ποσοστό αστοχιών δεδομένων με μία πλήρως συσχετιστική κρυφή μνήμη.

Επιπλέον, ένας μηχανισμός που συνδυάζει την ταχεία απόκριση των κρυφών μνημών ευθείας απεικόνισης με την επίδοση των συσχετιστικών κρυφών μνημών, είναι οι *ψευδο-συσχετιστικές κρυφές μνήμες (pseudo-associative caches)*. Στη υλοποίηση αυτή, οι κρυφές μνήμες ακολουθούν το μηχανισμό αναζήτησης δεδομένων και την ταχύτητα απόκρισης των μνημών ευθείας απεικόνισης. Σε περίπτωση όμως αστοχίας δεδομένων, πριν την προώθηση της αναζήτησης στο αμέσως χαμηλότερο ιεραρχικά επίπεδο, ελέγχεται το περιεχόμενο μίας ακόμη θέσης μνήμης.

Οι ψευδο-συσχετιστικές μνήμες έχουν έναν γρήγορο και έναν αργό χρόνο επιτυχίας, που αντιστοιχούν στο χρόνο επιτυχίας της κρυφής μνήμης ευθείας απεικόνισης και στο χρόνο της ψευδο-επιτυχίας (επιτυχία στην εναλλακτική θέση μνήμης που ελέγχεται). Ο μόνος κίνδυνος είναι αν όλοι οι γρήγοροι χρόνοι επιτυχίας της κρυφής μνήμης ευθείας απεικόνισης μετατραπούν σε αργούς χρόνους επιτυχίας της ψευδο-συσχετιστικής κρυφής μνήμης. Για το λόγο αυτό, η διάκριση μεταξύ των θέσεων ενός συνόλου που ελέγχονται κατά το γρήγορο ή τον αργό χρόνο προσπέλασης, πρέπει να γίνεται με προσοχή.

Επιπρόσθετα, οι *κρυφές μνήμες θυμάτων (victim caches)*, βελτιώνουν την επίδοση των κρυφών μνημών ευθείας απεικόνισης με την προσθήκη μίας πολύ μικρής πλήρως συσχετιστικής κρυφής μνήμης. Η μνήμη αυτή τοποθετείται μεταξύ του πρώτου επιπέδου κρυφής μνήμης και του αμέσως επόμενου επιπέδου της ιεραρχίας και δέχονται τα δεδομένα που απομακρύνονται από την κρυφή μνήμη πρώτου επιπέδου, λόγω αστοχιών σύγκρουσης. Με τον τρόπο αυτό, γίνεται προσπάθεια να μειωθεί ο ρυθμός αστοχιών, τόσο που να προσεγγίζει αυτόν των συσχετιστικών μνημών. Ταυτόχρονα, ο χρόνος προσπέλασης των λέξεων διατηρείται μικρός, όσο των μνημών ευθείας απεικόνισης.

2.4 Μηχανισμοί Αντικατάστασης στις Κρυφές Μνήμες

Όταν συμβεί μία αστοχία μνήμης, η ομάδα λέξεων στην οποία ανήκει το ζητούμενο στοιχείο πρέπει να τοποθετηθεί στην κρυφή μνήμη. Για το σκοπό αυτό επιλέγεται μία θέση στην κρυφή μνήμη. Το πλεονέκτημα των κρυφών μνημών ευθείας απεικόνισης είναι ότι οι αποφάσεις του υλικού, σχετικά με την τοποθέτηση νέων ομάδων λέξεων σε αυτές, είναι εξαιρετικά απλουστευμένες: μόνο μία θέση είναι υποψήφια για την τοποθέτηση της νέας ομάδας λέξεων. Στις πλήρως συσχετιστικές κρυφές μνήμες καθώς και στις κρυφές μνήμες συνόλου συσχέτισης υπάρχουν πολλές υποψήφιες θέσεις. Οι στρατηγικές επιλογής μεταξύ αυτών των θέσεων είναι οι εξής:

- Τυχαία: Προκειμένου η κατανομή των δεδομένων στις υποψήφιες θέσεις να απλωθεί όσο το δυνατόν περισσότερο, γίνεται τυχαία επιλογή. Σε ορισμένα συστήματα παράγεται ένας ψευδοτυχαίος αριθμός απεικόνισης.

- Αντικατάσταση του λιγότερο πρόσφατα χρησιμοποιημένου δεδομένου (Least Recently Used - LRU): Προκειμένου να μειωθεί η πιθανότητα διαγραφής χρήσιμων δεδομένων, καταγράφεται η συχνότητα προσπέλασής τους. Με βάση την καταγραφή αυτή, κάθε φορά που πρέπει να διαγραφεί κάποιο στοιχείο για να αποθηκευτεί ένα νέο, επιλέγεται εκείνο που δεν έχει χρησιμοποιηθεί για μεγαλύτερο χρονικό διάστημα. Η επιλογή αυτή βασίζεται στην αρχή της τοπικότητας: ό,τι χρησιμοποιήθηκε πιο πρόσφατα, έχει μεγαλύτερη πιθανότητα να επαναχρησιμοποιηθεί σύντομα.

- Αντικατάσταση του παλαιότερου δεδομένου (first in, first out - FIFO): Ο μηχανισμός αυτός προσεγγίζει την ακρίβεια του LRU, αποφεύγοντας ωστόσο την πολυπλοκότητά του. Αντικαθιστά το δεδομένο που αποθηκεύτηκε πρώτο από όλα τα υπόλοιπα υποψήφια στην κρυφή μνήμη.

2.5 Μηχανισμοί Εγγραφής στις Κρυφές Μνήμες

Ιδιαίτερη προσοχή χρειάζεται η αλλαγή του περιεχομένου μίας θέσης μνήμης. Στην περίπτωση που η ομάδα λέξεων στην οποία ανήκει το εν λόγω στοιχείο βρίσκεται ήδη στην κρυφή μνήμη, δύο είναι οι βασικές επιλογές:

- Εγγραφή σε όλα τα επίπεδα (write through) - Ενημερώνονται όλα τα επίπεδα μνήμης (από το ανώτερο επίπεδο κρυφής μνήμης μέχρι την κύρια μνήμη) σχετικά με τη νέα τιμή του στοιχείου.

- Εγγραφή μόνο κατά τη διαγραφή (write back)- Η νέα τιμή του στοιχείου καταγράφεται μόνο στο ανώτερο ιεραρχικά επίπεδο μνήμης. Επομένως, η κύρια μνήμη δεν περιέχει τις έγκυρες τιμές των δεδομένων, αλλά ενημερώνεται μόνο όταν η ομάδα λέξεων που περιέχει τις νέες τιμές δεδομένων πρέπει να διαγραφεί από την κρυφή μνήμη.

Στην περίπτωση που η εγγραφή αφορά δεδομένα που δε βρίσκονται ήδη αποθηκευμένα στην κρυφή μνήμη, ακολουθείται ένας από τους παρακάτω μηχανισμούς:

- Εκχώρηση θέσης στη νέα ομάδα λέξεων (write allocate) - Δημιουργείται χώρος (allocate) στην κρυφή μνήμη και η ομάδα λέξεων στην οποία ανήκει το προς εγγραφή δεδομένο μεταφέρεται ολόκληρο εκεί. Στη συνέχεια, ακολουθείται η ίδια πορεία, όπως και στην επιτυχία δεδομένων.

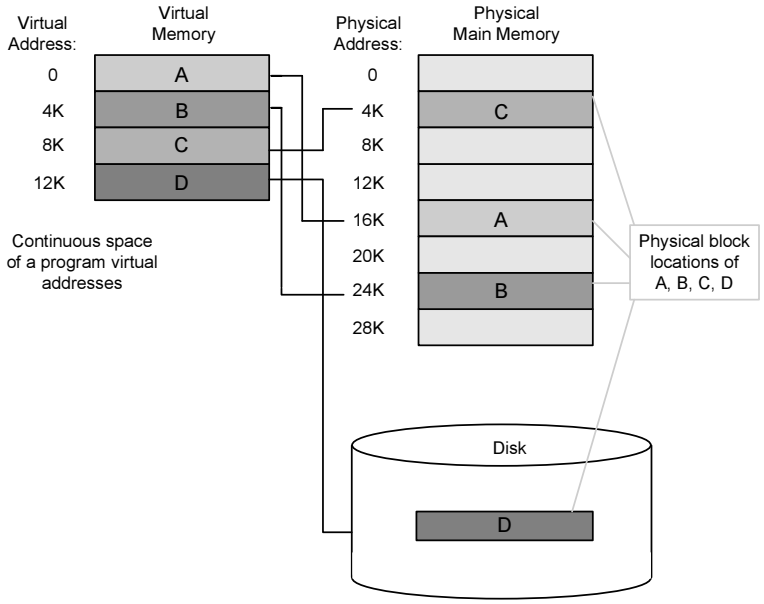
- Μη εκχώρηση θέσης στη νέα ομάδα λέξεων (write no allocate) - Στην περίπτωση αυτή, οι εγγραφές δεδομένων δεν επηρεάζουν την κρυφή μνήμη. Οι νέες τιμές των δεδομένων αποθηκεύονται κατευθείαν στην κύρια μνήμη, χωρίς να τροποποιηθεί το περιεχόμενο της κρυφής μνήμης.

2.6 Η Εικονική Μνήμη

Τα δεδομένα που χειρίζεται ένα πρόγραμμα, δεν είναι απαραίτητο να χωρούν στην κύρια μνήμη. Για το λόγο αυτό υπάρχει η *εικονική μνήμη (virtual memory)*, ώστε κάποια από τα δεδομένα να παραμένουν αποθηκευμένα στο δίσκο. Ο χώρος των διευθύνσεων χωρίζεται σε συγκεκριμένου μεγέθους ομάδες διευθύνσεων, που ονομάζονται *σελίδες (pages)*. Κάθε μία από τις σελίδες του τρέχοντος προγράμματος βρίσκονται είτε στην κύρια μνήμη είτε στο δίσκο. Όταν ο επεξεργαστής αναζητά δεδομένα τα οποία δε βρίσκονται αποθηκευμένα ούτε στην κρυφή ούτε στην κύρια μνήμη, τότε συμβαίνει μία *αστοχία σελίδας (page fault)*, οπότε ολόκληρη η σελίδα στην οποία περιέχονται τα ζητούμενα δεδομένα πρέπει να μεταφερθεί από το δίσκο στην κύρια μνήμη. Η επίλυση μίας αστοχίας σελίδας διαρκεί μεγάλο χρονικό διάστημα. Για το λόγο αυτό τη χειρίζεται το λογισμικό, και στο μεταξύ, ο επεξεργαστής ξεκινά κάποια άλλη διεργασία.

Ο επεξεργαστής κατά την εκτέλεση του προγράμματος γνωρίζει και επεξεργάζεται τις *εικονικές διευθύνσεις (virtual addresses)*. Για την προσπέλαση της μνήμης, οι εικονικές διευθύνσεις πρέπει να μεταφραστούν στις αντίστοιχες *φυσικές διευθύνσεις (physical addresses)*. Η μετάφραση των διευθύνσεων γίνεται μέσω μίας δομής, που ονομάζεται *πίνακας σελίδων (page table)*. Οι πίνακες σελίδων είναι συνήθως μεγάλοι και για το λόγο αυτό αποθηκεύονται στην κύρια μνήμη. Ωστόσο, αν για κάθε αναφορά στη μνήμη, και κατά συνέπεια για κάθε μετάφραση από την εικονική διεύθυνση του ζητούμενου στοιχείου στην αντίστοιχη φυσική, χρειαζόταν η προσπέλαση της κύριας μνήμης, η επιβάρυνση του χρόνου εκτέλεσης θα ήταν υπερβολικά υψηλή.

Βασιζόμενοι στην αρχή της τοπικότητας δεδομένων, δημιουργήθηκε μία κρυφή μνήμη, όπου αποθηκεύονται οι μεταφράσεις των πιο πρόσφατα χρησιμοποιούμενων διευθύνσεων. Αυτή η ειδικού σκοπού κρυφή μνήμη, έχει συνήθως οργάνωση πλήρους συσχέτισης και ονομάζεται *προσωρινός καταχωρητής μετάφρασης διευθύνσεων (Translation Look-aside Buffer: TLB)*.



Σχήμα 2.2: Η εικονική μνήμη

2.7 Ο Χώρος Επαναλήψεων

Κάθε διακριτή επανάληψη ενός φωλιασμένου βρόχου (nested loop) με βάθος n παριστάνεται με ένα n-διάστατο διάνυσμα:

$\vec{j} = (j_1, \ldots, j_n)$

Κάθε μία από τις συντεταγμένες του διανύσματος αυτού συσχετίζεται με έναν από τους βρόχους. Η αριστερότερη συντεταγμένη (j_1) αφορά τον εξωτερικότερο βρόχο, ενώ η δεξιότερη συντεταγμένη (j_n) τον εσωτερικότερο βρόχο.

Χώρος επαναλήψεων (iteration space) ενός φωλιασμένου βρόχου με βάθος n είναι ένα n-διάστατο πολύεδρο, το οποίο περικλείεται στα όρια των n βρόχων. Κάθε σημείο του n-διάστατου αυτού χώρου είναι μία διακριτή επανάληψη του εσωτερικού των φωλιασμένων βρόχων (loop body). Ο χώρος των επαναλήψεων παριστάνεται ως εξής:

$J_n = \{\vec{j}(j_1, \ldots, j_n) \mid j_i \in Z \wedge l_i \leq j_i \leq u_i, 1 \leq i \leq n\}$

2.8 Εξαρτήσεις Δεδομένων

Εξαρτήσεις δεδομένων παρουσιάζονται όταν, κατά τη διάρκεια μίας προκαθορισμένης ακολουθίας επαναλήψεων ενός προγράμματος, δύο αναφορές δεδομένων προσπελάζουν την ίδια θέση μνήμης. Μία τέτοια εξάρτηση μπορεί να είναι:

- *Πραγματική εξάρτηση δεδομένων (true dependence)* - η πρώτη αναφορά γράφει σε μία θέση μνήμης και η δεύτερη διαβάζει από αυτήν

- *Αντίστροφη εξάρτηση δεδομένων (anti-dependence)* - η πρώτη αναφορά διαβάζει το περιεχόμενο μίας θέσης μνήμης ενώ η δεύτερη γράφει σε αυτήν
- *Εξάρτηση εξόδου δεδομένων (output dependence)* - και οι δύο αναφορές γράφουν στην ίδια θέση μνήμης
- *Εξάρτηση εισόδου δεδομένων (input dependence)* - και οι δύο αναφορές διαβάζουν από την ίδια θέση μνήμης

Μετασχηματίζοντας έναν κώδικα, θα πρέπει να διατηρηθεί η σχετική σειρά των επαναλήψεων που συνδέονται με ένα από τα τρία πρώτα είδη εξαρτήσεων. Διαφορετικά, τα αποτελέσματα του προγράμματος θα είναι λανθασμένα. Η παραβίαση των εξαρτήσεων εισόδου δεδομένων δεν επιφέρει λανθασμένα αποτελέσματα. Ωστόσο, η αναγνώριση αυτού του είδους εξαρτήσεων εξυπηρετεί για την αποτίμηση της επαναχρησιμοποίησης δεδομένων.

Ως *διάνυσμα εξάρτησης (dependence vector)* μίας επανάληψης ενός προγράμματος με φωλιασμένους βρόχους ορίζουμε τη διαφορά του διανύσματος $\vec{j} = (j_1, \ldots, j_n)$ που εκφράζει τη συγκεκριμένη επανάληψη μείον το διάνυσμα $\vec{j'} = (j_1', \ldots, j_n')$, που εκφράζει την επανάληψη από την οποία εξαρτάται η $\vec{j}$ (τα αποτελέσματα της επανάληψης $\vec{j'}$ χρησιμοποιούνται άμεσα κατά τους υπολογισμούς που πραγματοποιούνται από την $\vec{j}$).

Κάθε διάνυσμα εξάρτησης αναπαριστάται ως ένα n-διάστατο διάνυσμα: $\vec{d_i} = (d_{i1}, \ldots, d_{in})$.

2.9 Μετασχηματισμοί Βρόχων

Κατά τη βελτιστοποίηση της επίδοσης ενός προγράμματος, σημαντικό κέρδος μπορεί να προκύψει από τα τμήματα του κώδικα στα οποία καταναλώνεται το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης. Πρόκειται συνήθως για επαναλαμβανόμενες δομές εκτέλεσης. Στην παρούσα συγγραφή θα ασχοληθούμε με n-διάστατους φωλιασμένους βρόχους που περιέχουν αναφορές σε στοιχεία n-διάστατων πινάκων. Προκειμένου να μειωθεί ο χρόνος εκτέλεσης τέτοιων φωλιασμένων βρόχων εφαρμόζονται διάφορες τεχνικές μετασχηματισμού κώδικα και δεδομένων, με στόχο τη βελτίωση της τοπικότητας δεδομένων.

Στους μετασχηματισμούς δεδομένων εντάσσεται η *ευθυγράμμιση των δεδομένων (array alignment)* ή *παραγέμισμα (padding)* των πινάκων. Ο δεύτερος μετασχηματισμός μεγαλώνει το μέγεθος των πινάκων εισάγοντας επιπλέον (περιττά) στοιχεία. Στην ίδια κατηγορία εντάσσεται και η αλλαγή της διάταξης αποθήκευσης των στοιχείων των πινάκων στη μνήμη (data array layout). Οι διαφορετικοί τρόποι αποθήκευσης θα αναλυθούν εκτενώς στο κεφάλαιο 3.

Οι *μετασχηματισμοί κώδικα* ή, όπως διαφορετικά ονομάζονται, οι *μετασχηματισμοί βρόχων (code ή loop transformations)* έχουν στόχο την αλλαγή της σειράς εκτέλεσης των επαναλήψεων, έτσι ώστε να γίνει καλύτερη αξιοποίηση της επαναχρησιμοποίησης δεδομένων, μέσω της ιεραρχίας μνήμης. Στις περισσότερες περιπτώσεις πρέπει να βρεθεί ένας κατάλληλος συνδυασμός των μετασχηματισμών αυτών, προκειμένου να επιτευχθεί η βέλτιστη επίδοση.

Στους μετασχηματισμούς βρόχων εντάσσεται, καταρχάς, η *ανταλλαγή βρόχων (loop interchange/permutation)* με την οποία αντιμετατίθενται δύο βρόχοι. Έτσι, αν στο αρχικό πρόγραμμα η σειρά προσπέλασης των δεδομένων δεν ακολουθούσε τη σειρά αποθήκευσής τους στη μνήμη, η απλή ανταλλαγή βρόχων μπορεί να επιτύχει σειριακή προσπέλαση των δεδομένων, επιτυγχάνοντας χωρική τοπικότητα δεδομένων.

Η *συγχώνευση βρόχων (loop fusion)* βελτιώνει τη δομή των προγραμμάτων στις περιπτώσεις προσπέλασης των ίδιων πινάκων δεδομένων μέσα σε διαφορετικούς φωλιασμένους βρόχους. Συγχωνεύοντας τους βρόχους, μειώνεται η απόσταση μεταξύ των διαδοχικών προσπελάσεων των ίδιων στοιχείων, βελτιώνοντας τη χρονική τοπικότητα δεδομένων.

Η *διάσπαση βρόχων (loop fission/distribution)* ωφελεί τα προγράμματα που περιέχουν μεγάλο όγκο εντολών και μεταβλητών μέσα στο σώμα του βρόχου επανάληψης. Με τη μέθοδο αυτή, ο αρχικός φωλιασμένος βρόχος διασπάται σε μία ακολουθία διαδοχικών φωλιασμένων βρόχων, καθένας από τους οποίους περιέχει ένα υποσύνολο των αρχικών εντολών. Επομένως, μειώνεται ο όγκος των δεδομένων που επεξεργάζεται και επαναχρησιμοποιεί το πρόγραμμα (working set) σε κάθε ένα από τα υποσύνολα, μειώνοντας έτσι την πιθανότητα αστοχιών χωρητικότητας και σύγκρουσης.

Το *λόξεμα βρόχων (loop skewing)* είναι ένας μετασχηματισμός που αλλάζει το σχήμα του χώρου επαναλήψεων. Δεν βελτιώνει από μόνος του την επίδοση ενός προγράμματος, αλλά δίνει τη δυνατότητα εφαρμογής άλλων τεχνικών στον μετασχηματισμένο πλέον φωλιασμένο βρόχο. Μετά την εφαρμογή του μετασχηματισμού αυτού, τα σχήματα των tiles που προκύπτουν είναι τραπεζοειδή (όχι πλέον τετράγωνα ή ορθογώνια). Οι καθυστερήσεις που προκύπτουν από την μη-κανονική δομή τους ή και από τον έλεγχο των πολύπλοκων συναρτήσεων που καθορίζουν τα όρια των tiles μπορεί να είναι σημαντικές. Για το λόγο αυτό, το λόξεμα βρόχων θα πρέπει να εφαρμόζεται με προσοχή στις περιπτώσεις που κρίνεται αναγκαίο.

Το *ξεδίπλωμα εσωτερικών ή εξωτερικών βρόχων (inner ή outer unrolling)* αυξάνει το βαθμό παραλληλισμού ενός προγράμματος, αντικαθιστώντας ένα τμήμα του βρόχου με πολλαπλά αντίγραφά του. Με τη μέθοδο αυτή μειώνεται η επιβάρυνση που εισάγουν οι επαναληπτικές εκτελέσεις των εντολών άλματος, λόγω των καθυστερήσεων που προκύπτουν μετά από κάθε λανθασμένη πρόβλεψη.

Η *αντικατάσταση βαθμωτών αναφορών σε πίνακες από μεταβλητές (scalar replacement)* βελτιώνει την τοπικότητα δεδομένων σε επίπεδο καταχωρητών. Η αντικατάσταση αυτή δίνει τη δυνατότητα της ανάθεσης των μεταβλητών αυτών σε καταχωρητές, κρατώντας το περιεχόμενό τους κοντά στον επεξεργαστή για άμεση επαναχρησιμοποίηση.

Τέλος, με το *μετασχηματισμό υπερκόμβων (tiling)*, ο αρχικός χώρος επαναλήψεων J_n διαμερίζεται σε πανομοιότυπους n-διάστατους χώρους, που ονομάζονται υπερκόμβοι (tiles). Με τον μετασχηματισμό αυτό, αντί να προσπελάζεται ολόκληρος ο όγκος δεδομένων, μειώνεται το εύρος των εσωτερικών βρόχων, και η προσπέλαση περιορίζεται μέσα στο tile, δηλαδή σε ένα μικρό τμήμα του συνολικού όγκου δεδομένων. Κατά αυτόν τον τρόπο, μεγιστοποιείται ο αριθμός των

προσβάσεων στα δεδομένα του κάθε tile πριν προχωρήσουμε στο επόμενο και τα δεδομένα του απομακρυνθούν από την κρυφή μνήμη.

ΚΕΦΑΛΑΙΟ 3

Μη γραμμικές διατάξεις αποθήκευσης

Στο συγκεκριμένο κεφάλαιο προκειμένου να καταστήσουμε ικανή την αυτόματη παραγωγή του μετασχηματισμένου σε tiles κώδικα, ο οποίος προσπελάζει της μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων προτείνουμε μία μέθοδο υπολογισμού της δεικτοδότησης των πινάκων. Υιοθετήσαμε ένα είδος μη-γραμμικής διάταξης ομαδοποίησης δεδομένων, καθώς και τη δεικτοδότηση των πινάκων με "αραιωμένους" ακεραίους, παρόμοια με τους αναδρομικούς πίνακες Morton. Επομένως, συνδυάζουμε την αυξημένη τοπικότητα δεδομένων, χάρη στις μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων, με μία τεχνική γρήγορης προσπέλασης δεδομένων, χάρη στο μηχανισμό υπολογισμού της θέσης αποθήκευσης των δεδομένων, με απλές δυαδικές (boolean) πράξεις, ο οποίος ενσωματώνεται στον εκτελούμενο κώδικα. Στην ενότητα 3.4 παρουσιάζεται η προτεινόμενη μέθοδος. Η μέθοδος αυτή είναι ιδιαίτερα αποτελεσματική στη μείωση των αστοχιών της κρυφής μνήμης, καθώς η διάταξη αποθήκευσης των δεδομένων των πινάκων στη μνήμη ακολουθεί με ακρίβεια τη σειρά προσπέλασης των δεδομένων από τον μετασχηματισμένο κώδικα με tiling και, επιπλέον, οι προσπελάσεις δεδομένων γίνονται χωρίς χρονοβόρους υπολογισμούς κατά το χρόνο εκτέλεσης.

3.1 Το πρόβλημα: Βελτίωση της τοπικότητας δεδομένων των πινάκων

Στην ενότητα αυτή περιγράφουμε τους λόγους που καθιστούν αναγκαία την εφαρμογή μετασχηματισμών βρόχων σε συνδυασμό με μετασχηματισμούς δεδομένων, προκειμένου να αξιοποιηθεί πλήρως η επαναχρησιμοποίηση δεδομένων. Περιγράφουμε βήμα προς βήμα όλες τις επιμέρους φάσεις βελτιστοποίησης, χρησιμοποιώντας το παράδειγμα του πολλαπλασιασμού πινάκων.

---

1. η μη βελτιστοποιημένη έκδοση

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
      C[i,j] += A[i,k] * B[k,j];
```

2. ο μετασχηματισμένος κώδικας με tiling

```
for (jj = 0; jj < N; jj += step)
  for (kk = 0; kk < N; kk += step)
    for (i = 0; i < N; i++)
      for (j = jj; (j < N && j < jj + step); j++)
        for (k = kk; (k < N && k < kk + step); k++)
          C[i,j] += A[i,k] * B[k,j];
```

3. μετασχηματισμοί βρόχων και δεδομένων

```
for (kk=0; kk < N; kk += step)
  for (jj = 0; jj < N; jj += step)
    for (i = 0; i < N; i++)
      for (k = kk; (k < N && k < kk + step); k++)
        for (j = jj; (j < N && j < jj + step); j++)
          Cr[i,j] += Ar[i,k] * Br[k,j];
```

4. μη γραμμικές διατάξεις ομαδοποίησης δεδομένων

```
for (ii = 0; ii < N; ii += step)
  for (kk = 0; kk < N; kk += step)
    for (jj = 0; jj < N; jj += step)
      for (i = ii; (i < N && i < ii + step); i++)
        for (k = kk; (k < N && k < kk + step); k++)
          for (j = jj; (j < N && j < jj + step); j++)
            Czz[i + j] += Azz[i + k] * Bzz[k + j];
```

Μετασχηματισμοί Βρόχων: Ο κώδικας #1 παρουσιάζει την τυπική, μη βελτιστοποιημένη έκδοση του πολλαπλασιασμού πινάκων. Ο μετασχηματισμός tiling (κώδικας #2) αναδιοργανώνει τη σειρά εκτέλεσης των επαναλήψεων, έτσι ώστε ο αριθμός των επαναλήψεων που μεσολαβούν μεταξύ διαδοχικών επαναχρησιμοποιήσεων δεδομένων και επομένως τα δεδομένα που προσπελάζονται στο διάστημα αυτό, μειώνονται. Έτσι, επιτυγχάνουμε να μην διαγράφονται τα χρήσιμα δεδομένα από το αρχείο καταχωρητών ή την κρυφή μνήμη, προτού επαναχρησιμοποιηθούν. Το μέγεθος του tile (η διάσταση *step*) επιλέγεται ανάλογα σε πιο επίπεδο μνήμης επιθυμούμε να επιτύχουμε τη μέγιστη τοπικότητα των αναφορών [LRW91]. Εκτεταμένη ανάλυση για την επιλογή του βέλτιστου μεγέθους του tile παρουσιάζεται στο κεφάλαιο 4.

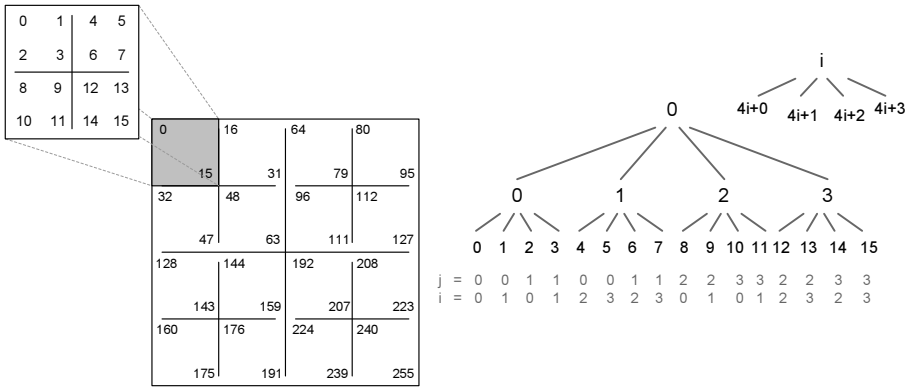
Μετασχηματισμοί Βρόχων και Δεδομένων: Οι μετασχηματισμοί βρόχων δεν μπορούν από μόνοι τους να επιτύχουν τη βέλτιστη δυνατή τοπικότητα αναφορών. Για το λόγο αυτό, καθίσταται αναγκαία η συνδυασμένη εφαρμογή των μετασχηματισμών βρόχων και δεδομένων. Στον κώδικα #2, ο βρόχος k προσπελάζει διαφορετικές γραμμές του πίνακα B. Με δεδομένη μία διάταξη αποθήκευσης κατά γραμμές, η χωρική τοπικότητα δεδομένων μπορεί να αξιοποιηθεί για τον πίνακα B κατά την εκτέλεση του εσωτερικότερου βρόχου k. Δίνοντας έμφαση στην αξιοποίηση της χωρικής τοπικότητας δεδομένων της ίδιας αναφοράς [KRC99] (καθώς η χρονική τοπικότητα μπορεί να θεωρηθεί ένα υποσύνολο της χωρικής τοπικότητας, ενώ η χωρική τοπικότητα μεταξύ διαφορετικών αναφορών είναι σπάνια), ο μετασχηματισμένος κώδικας παίρνει τη μορφή #3. Καταρχάς ρυθμίζουμε τη διάταξη αποθήκευσης των δεδομένων του πίνακα που βρίσκεται στην αριστερή πλευρά της ισότητας (LHS: Left Hand Side), δηλαδή του πίνακα C, επειδή σε κάθε επανάληψη τα δεδομένα του πίνακα αυτού και διαβάζονται και γράφονται, ενώ τα στοιχεία των πινάκων A και B μόνο διαβάζονται. Επιλέγοντας το βρόχο j ως τον εσωτερικότερο, η γρηγορότερα μεταβαλλόμενη διάσταση του πίνακα C θα πρέπει να ελέγχεται από το δείκτη αυτόν. Δεδομένου ότι στο φωλιασμένο βρόχο περιέχεται η αναφορά $C[i, j]$, ο πίνακας C θα πρέπει να αποθηκεύεται κατά γραμμές (Cr). Παρομοίως, ο πίνακας B, του οποίου η αναφορά είναι η $B[k, j]$, θα πρέπει επίσης να αποθηκεύεται κατά γραμμές (Br). Τέλος, η τοποθέτηση των βρόχων με τη σειρά ikj είναι η προτιμότερη, επειδή αξιοποιείται η χρονική τοπικότητα δεδομένων (μέσα στην ίδια την αναφορά) στον δεύτερο εσωτερικότερο βρόχο για τον πίνακα C. Επομένως, ο πίνακας A, με αναφορά την $A[i, k]$, θα πρέπει επίσης να αποθηκεύεται κατά γραμμές (Ar).

Μετασχηματισμοί Βρόχων και Μη-γραμμικοί Μετασχηματισμοί Δεδομένων: Προκειμένου να αξιολογήσουμε τα πλεονεκτήματα των μη-γραμμικών μετασχηματισμών δεδομένων, εξετάζουμε τον κώδικα #4. Υποθέτουμε ότι τα στοιχεία των τριών πινάκων αποθηκεύονται ακριβώς με τη σειρά που προσπελάζονται από το πρόγραμμα (ονομάζουμε τη διάταξη αυτή ZZ, όπως παρουσιάζεται αναλυτικά στην ενότητα 3.3). Η σειρά διάταξης των βρόχων παραμένει όπως στον κώδικα #3, εκτός από το ότι ο μετασχηματισμός tiling εφαρμόζεται επιπρόσθετα και στο βρόχο i, έτσι ώστε να σχηματιστούν και στους τρεις πίνακες ομοιόμορφα σε σχήμα tiles, απλοποιώντας τους υπολογισμούς που χρειάζονται για την εύρεση της θέσης κάθε στοιχείου.

3.2 Αναδρομική (Μη-γραμμική) Διάταξη Αποθήκευσης Πινάκων κατά Morton

Προκειμένου να κατανοήσουμε την έννοια της διάταξης κατά Morton [Wis01], μίας αναδρομικής σειράς αποθήκευσης των στοιχείων των πινάκων, παρουσιάζονται στη συνέχεια τα βασικά στοιχεία της άλγεβρας "αραιωμένων" ακεραίων. Ο Morton όρισε τη δεικτοδότηση των διδιάστατων πινάκων και κατέδειξε τη μετατροπή από και προς την καρτεσιανή δεικτοδότηση μέσω της αραίωσης των δυαδικών ψηφίων του δείκτη (bit interleaving) (σχήμα 3.1). Οι ορισμοί που ακολουθούν ισχύουν

μόνο για διδιάστατους πίνακες. Ένας d-διάστατος πίνακας παριστάνεται με ένα δέντρο βαθμού 2^d, και η δεικτοδότησή του ακολουθεί την ίδια φιλοσοφία.



Σχήμα 3.1: Δεικτοδότηση κατά γραμμές για έναν διδιάστατο πίνακα και η ανάλογη δεικτοδότηση για μία δενδροειδή διάταξη βαθμού 4

Ορισμός 3.1 *Ο ακέραιος* $\overrightarrow{b} = \sum_{k=0}^{w-1} 4^k$ *είναι η σταθερά 0x55555555 και ονομάζεται evenBits. Παρόμοια, ο αριθμός* $\overleftarrow{b} = 2\overrightarrow{b}$ *είναι η σταθερά 0xaaaaaaaa και ονομάζεται oddBits.*

Ο δεκαεξαδικός αριθμός $evenBits = 0x55555555$ έχει όλα τα άρτια bits ίσα με 1 και όλα τα περιττά bits μηδενικά. Ανάλογα, ο αριθμός $oddBits = evenBits << 1$ έχει την τιμή $0xaaaaaaaa$, με όλα τα άρτια bits ίσα με 0 και όλα τα περιττά bits ίσα με 1.

Ορισμός 3.2 *Η παράσταση της άρτιας αραίωσης (even-dilated) του δείκτη* $j = \sum_{k=0}^{w-1} j_k 2^k$ *είναι η* $\sum_{k=0}^{w-1} j_k 4^k$*, και συμβολίζεται* $\overrightarrow{j}$*. Η παράσταση της περιττής αραίωσης (odd-dilated) του δείκτη* $i = \sum_{k=0}^{w-1} i_k 2^k$ *είναι η* $2\overrightarrow{i}$ *και συμβολίζεται* $\overleftarrow{i}$*.*

Θεώρημα 3.1 *Κατά την παράσταση Morton, ο δείκτης για το* $< i, j >^{\text{οστό}}$ *στοιχείο ενός πίνακα είναι ο* $\overleftarrow{i} \vee \overrightarrow{j} = \overleftarrow{i} + \overrightarrow{j}$*.*

Ας υποθέσουμε έναν καρτεσιανό δείκτη γραμμών $\overleftarrow{i}$ με τα σημαντικά δυαδικά ψηφία του "αραιωμένα" κατά τη σταθερά $oddBits$ (έτσι ώστε να καταλαμβάνουν τις θέσεις των ψηφίων που είναι ίσα με 1 στη σταθερά $oddBits$) και έναν καρτεσιανό δείκτη $\overrightarrow{j}$ "αραιωμένο" κατά τη σταθερά $evenBits$ (έτσι ώστε τα σημαντικά του ψηφία να καταλαμβάνουν τις θέσεις των ψηφίων που ισούται με ένα στη σταθερά $evenBits$). Αν τα στοιχεία του πίνακα A αποθηκεύονται κατά τη διάταξη Morton, τότε το στοιχείο $[i, j]$ μπορεί να προσπελάζεται από την αναφορά $A[\overleftarrow{i} + \overrightarrow{j}]$ ανεξάρτητα από το μέγεθος του πίνακα.

Στη διάταξη Morton ο βρόχος

```
for (i=0; i < N; i++)   ...
```

μετατρέπεται ως εξής:

```
for(i_m=0; i_m < N_m; i_m=(i_m - evenBits)&evenBits)   ...
```

όπου $i_m = \overrightarrow{i}$ και $N_m = \overrightarrow{N}$

Η παράσταση των "αραιωμένων" ακεραίων χρησιμοποιείται ως εξής: Αν ένας δείκτης βρόχου χρησιμοποιείται μόνο ως δείκτης στηλών σε όλες τις αναφορές πινάκων που περιέχονται στο σώμα του φωλιασμένου βρόχου, τότε θα πρέπει να χρησιμοποιούμε περιττή αραίωση. Διαφορετικά, κατάλληλη είναι η άρτια αραίωση. Αν ο δείκτης χρησιμοποιείται και στους δύο ρόλους (ως δείκτης στηλών και ως δείκτης γραμμών) χρειαζόμαστε και τους δύο τύπους αραίωσης: η περιττή παράσταση προκύπτει με διπλασιασμό της τιμής της άρτιας αραίωσης.

Για το παράδειγμα του κώδικα #1, αφότου οι δείκτες i και j μεταφραστούν στις αντίστοιχες παραστάσεις τους $\overleftarrow{i}$ και $\overrightarrow{j}$, ο κώδικας του πολλαπλασιασμού πινάκων μετασχηματίζεται ως εξής:

```
#define evenIncrement(i)(i = ((i - evenBits)&evenBits))
#define oddIncrement(i)(i = ((i - oddBits)&oddBits))

  for (i=0; i < colsOdd; oddIncrement(i))
    for (j=0; j < rowsEven; evenIncrement(j))
      for (k=0; k < rowsAEven; evenIncrement(k))
        C[i + j]+ = A[i + k] * B[2 * k + j];
```

όπου $rowsEven$, $colsOdd$ and $rowsAEven$ είναι τα όρια των πινάκων όταν μετασχηματιστούν με την τεχνική της αραίωσης. Σημειώστε ότι ο δείκτης k χρησιμοποιείται τόσο ως δείκτης στηλών όσο και ως δείκτης γραμμών. Επομένως, μεταφράζεται στην παράσταση $\overrightarrow{k}$, και για το δείκτη στηλών του πίνακα B, χρησιμοποιείται η τιμή $2 * k$, η οποία ισούται με την παράσταση $\overleftarrow{k}$.

Παράδειγμα 3.1:

Έστω ότι αναζητούμε το στοιχείο

$$C[i, j] = C[3, 4]$$

Για τον υπολογισμό της θέσης αποθήκευσής του, όταν ο πίνακας C ακολουθεί τη διάταξη Morton, ακολουθούμε τη διαδικασία που περιγράφεται στη συνέχεια.

Ο δείκτης γραμμών j πρέπει να "αραιωθεί" κατά τη σταθερά $evenBits = 1010101_{<2>}$. Στη γραμμική του μορφή έχει την τιμή:

$$j = 4_{<10>} = 100_{<2>}$$

Αραίωση της παραπάνω τιμής σημαίνει ότι μεταξύ των δυαδικών ψηφίων του δείκτη j όπως αναγράφονται στη γραμμική παράσταση της τιμής του, θα πρέπει να παρεμβληθούν μηδενικά. Δηλαδή, όπου υπάρχει 1 στη σταθερα $evenBits$ είναι θέσεις που καταλαμβάνονται από τα δυαδικά ψηφία του δείκτη, είτε μηδενικά, είτε μονάδες. Άρα:

$$\overrightarrow{j} = x_3 0 x_2 0 x_1 0 x_0$$

όπου x_n, η τιμή του n-οστού δυαδικού ψηφίου του j.

$$\overrightarrow{j} = 0010000_{<2>} = 8_{<10>}$$

Ομοίως, ο δείκτης στηλών i θα πρέπει να "αραιωθεί" κατά τη σταθερά $oddBits = 0101010_{<2>}$. Στη γραμμική του μορφή έχει την τιμή:

$$i = 3_{<10>} = 011_{<2>}.$$
$$\overleftarrow{i} = 0 x_2 0 x_1 0 x_0 0$$

όπου x_n, η τιμή του n-οστού δυαδικού ψηφίου του i.

$$\overleftarrow{i} = 0001010_{<2>} = 10_{<10>}$$

Έτσι, το στοιχείο $C[i, j]$ είναι αποθηκευμένο στη θέση:

$$\overleftarrow{i} + \overrightarrow{j} = 0001010 + 0010000 = 0011010_{<2>}$$
$$\overleftarrow{i} + \overrightarrow{j} = 10_{<10>} + 8_{<10>} = 18_{<10>}.$$

Άρα, πρόκειται για το στοιχείο $C[18]$. ◇

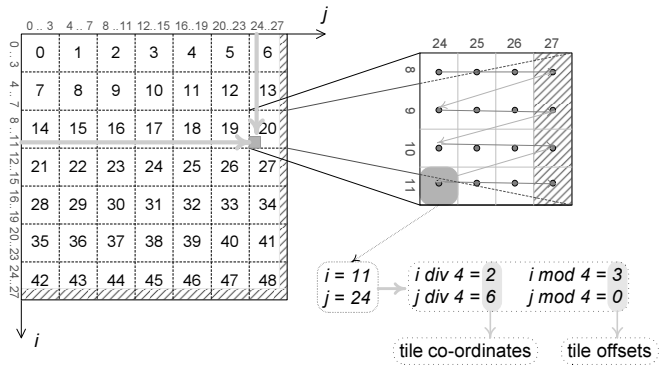
3.3 Μη-γραμμικές Διατάξεις Ομαδοποίησης Δεδομένων Πινάκων

Στην ενότητα αυτή παρουσιάζουμε τους μη-γραμμικούς μετασχηματισμούς δεδομένων. Τέτοιες διατάξεις δεδομένων χρησιμοποιούνταν από τους Chatterjee κλπ στις εργασίες τους [CJL+99], [CLPT99], προκειμένου να επιτύχουν καλύτερη επίδοση των κρυφών μνημών. Σύμφωνα με την εργασία [CJL+99], ένας διδιάστατος πίνακας μεγέθους $m \times n$ μπορεί να μετασχηματισθεί σε έναν πίνακα μεγέθους $\lceil \frac{m}{t_R} \rceil \times \lceil \frac{n}{t_C} \rceil$, κάθε στοιχείο του οποίου θα αποτελεί ένα tile μεγέθους $t_R \times t_C$. Ισοδύναμα, ο αρχικός διδιάστατος χώρος επαναλήψεων (i, j) απεικονίζεται σε έναν 4-διάστατο χώρο (t_i, t_j, f_i, f_j), όπως φαίνεται στα σχήματα 3.3 και 3.5. Η προτεινόμενη διάταξη του χώρου $L_F : (f_i, f_j)$ (tile offsets), που περιέχει τα δεδομένα των tiles, είναι γραμμική, δηλαδή τα στοιχεία αποθηκεύονται κατά γραμμές ή κατά στήλες, ανάλογα με τη σειρά προσπέλασής τους από τον κώδικα του προγράμματος. Η συνάρτηση μετασχηματισμού του χώρου $L_T : (t_i, t_j)$ (tile co-

ordinates), που περιέχει που περιέχει τις διευθύνσεις των tiles, μπορεί να είναι είτε γραμμική είτε να ακολουθεί τη διάταξη Morton.

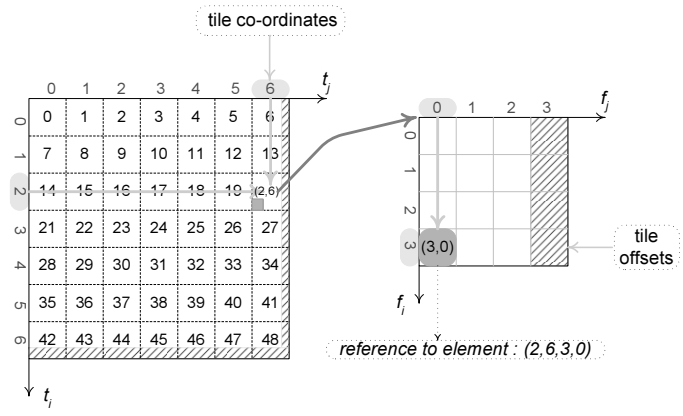
3.3.1 Η διάταξη L_{4D}

Στη διάταξη L_{4D}, τόσο η διάταξη L_F του εσωτερικού των tiles, όσο και η διάταξη L_T των διαφορετικών tiles είναι γραμμική. Στο παράδειγμά μας, οι επιλεγόμενες γραμμικές διατάξεις θα είναι κατά γραμμές, προκειμένου να ακολουθούν τη σειρά προσπέλασης των στοιχείων των πινάκων από τον κώδικα του πολλαπλασιασμού πινάκων. Ο αρχικός πίνακας μεγέθους 27×27 στοιχείων, διαιρείται σε tiles μεγέθους 4×4 στοιχεία. Στα όρια των πινάκων προστίθενται παραγεμίσματα (padding), έτσι ώστε να σχηματίζονται ολόκληρα tiles. Ο συγκεκριμένος πίνακας φαίνεται και στο σχήμα 3.2, όπου έχει σχεδιαστεί επιπλέον και η σειρά αποθήκευσης των tiles.



Σχήμα 3.2: Ο μετασχηματισμένος πίνακας σύμφωνα με τη διάταξη L_{4D}

Ο τελικός 4-διάστατος πίνακας σχεδιάζεται στο σχήμα 3.3. Ο 4-διάστατος χώρος αποθήκευσης των στοιχείων των πινάκων παριστάνεται ως ένα 2-διάστατο επίπεδο όπου βρίσκονται οι διευθύνσεις αποθήκευσης των tiles, οι οποίες αποτελούν ουσιαστικά δείκτες στο 2-διάστατο χώρο αποθήκευσης του εσωτερικού κάθε tile.



Σχήμα 3.3: Ο τελικός 4-διάστατος πίνακας κατά τη διάταξη L_{4D}

Η τιμή κάθε διάστασης για το στοιχείο (i, j) υπολογίζεται ως εξής (Θεωρούμε ότι το μέγεθος των tiles είναι $step \times step$):

$$\begin{aligned} \text{tile co-ordinates :} \quad & t_i = i \; div \; step \\ & t_j = j \; div \; step \\ \text{tile offsets :} \quad & f_i = i \; mod \; step \\ & f_j = j \; mod \; step \end{aligned}$$

Αν ο αρχικός κώδικας πολλαπλασιασμού πινάκων μετασχηματιστεί κατά tiling και στις τρεις διαστάσεις του, προκύπτει ένας φωλιασμένος βρόχος βάθους 6:

```
for (ii = 0; ii < N; ii + step)
  for (kk = 0; kk < N; kk + step)
    for (jj = 0; jj < N; jj + step)
      for (i = 0; (i < ii + step && i < N); i++)
        for (k = 0; (k < kk + step && k < N); k++)
          for (j = 0; (j < jj + step && j < N); j++)
            C[i][j]+ = A[i][k] * B[k][j];
```

Χρησιμοποιώντας τη διάταξη L_{4D}, η υλοποίηση του πολλαπλασιασμού πινάκων γίνεται ως εξής:

```
for (ii = 0; ii < NN; ii++) {
  stepi = (N > step * (ii + 1)?step : (N − ii * step));
  for (kk = 0; kk < NN; kk++) {
    stepk = (N > step * (kk + 1)?step : (N − kk * step));
    for (jj = 0; jj < NN; jj++) {
      stepj = (N > step * (jj + 1)?step : (N − jj * step));
      for (i=0; i < stepi; i++)
        for (k=0; k < stepk; k++)
          for (j=0; j < stepj; j++)
            C[ii][jj][i][j]+ = A[ii][kk][i][k] * B[kk][jj][k][j]; }
  }
}
```

όπου N, $step$ είναι τα όρια των πινάκων και των tiles αντίστοιχα, και $NN = \lceil \frac{N}{step} \rceil$.

Παράδειγμα 3.2:

Έστω ότι αναζητούμε το στοιχείο

$$C[i, j] = C[11, 24]$$

Για τον υπολογισμό της θέσης αποθήκευσής του, ακολουθούμε τη διαδικασία που περιγράφεται στη συνέχεια και απεικονίζεται στο σχήμα 3.2. Θεωρούμε ότι ο πίνακας C μεγέθους 27×27 ακολουθεί τη διάταξη L_{4D}, και είναι χωρισμένος σε tiles μεγέθους 4×4.

Tile coordinates:

$$t_i = i \; div \; step = 11 \; div \; 4 = 2_{<10>}$$
$$t_j = j \; div \; step = 24 \; div \; 4 = 6_{<10>}$$

Επομένως, το ζητούμενο στοιχείο βρίσκεται στο tile με συντεταγμένες $[2, 6]$

Tile offsets:

$$f_i = i \; mod \; step = 11 \; mod \; 4 = 3_{<10>}$$
$$f_j = j \; mod \; step = 24 \; mod \; 4 = 0_{<10>}$$

Επομένως, στο εσωτερικό του tile, το ζητούμενο στοιχείο βρίσκεται στη θέση $[3, 0]$

Στη διάταξη L_{4D}, το στοιχείο $C[11, 24]$ του αρχικού πίνακα, αποθηκεύεται σε έναν τετραδιάστατο πίνακα και πρέπει να αναζητηθεί στη θέση [2][6][3][0]. ◇

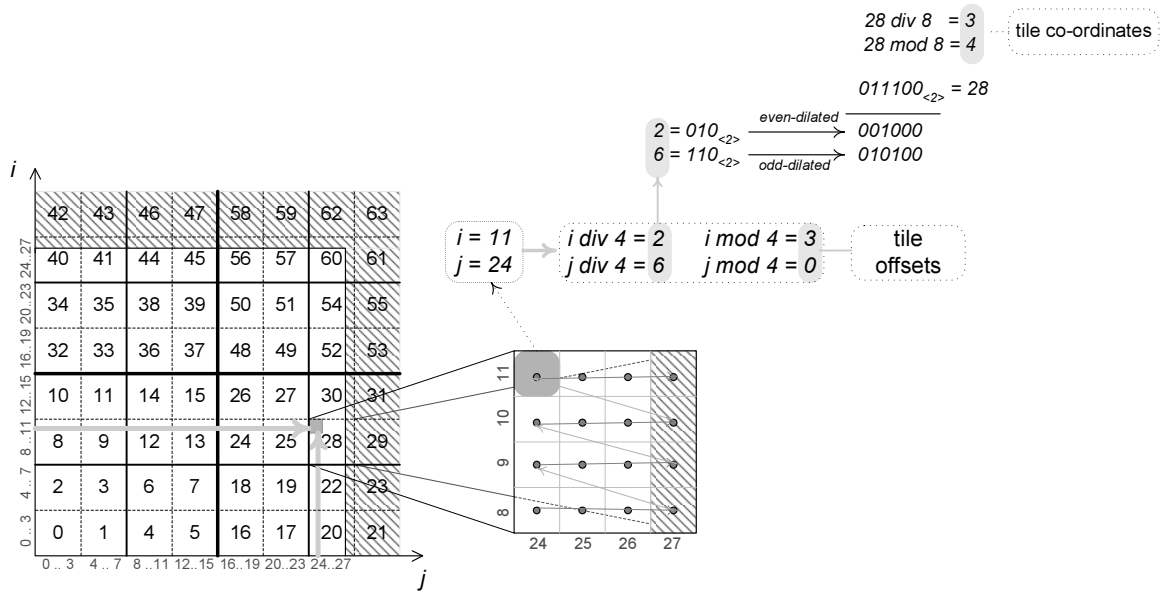
3.3.2 Η διάταξη L_{MO}

Στη διάταξη L_{MO}, η σειρά αποθήκευσης των διαφορετικών tiles στη μνήμη L_T ακολουθεί την σειρά Morton, ενώ η σειρά αποθήκευσης των στοιχείων στο εσωτερικό των tiles L_F είναι γραμμική (στο παράδειγμά μας επιλέγουμε αποθήκευση κατά γραμμές για να συμβαδίζει με τη ακολουθία προσπέλασης των στοιχείων αυτών). Ο αρχικός πίνακας είναι όμοιος με αυτόν της ενότητας 3.3.1 εκτός από τον αριθμό των στοιχείων που προστέθηκαν ως παραγεμίσματα. Στην περίπτωση του L_{MO}, τα παραγεμίσματα είναι περισσότερα, επειδή η διάσταση του "παραγεμισμένου" πίνακα θα πρέπει να είναι ίση με κάποια δύναμη του 2. Ο πίνακας αυτός απεικονίζεται στο σχήμα 3.4, όπου φαίνεται η σειρά αποθήκευσης των στοιχείων.

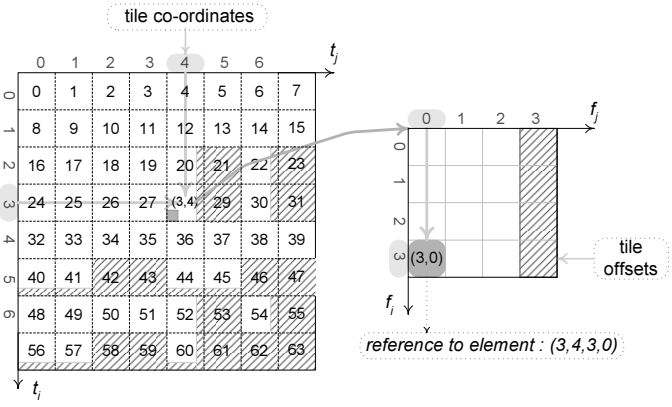
Ο τελικός 4-διάστατος πίνακας φαίνεται στο σχήμα 3.5. Σημειώστε ότι τα παραγεμίσματα δεν βρίσκονται πλέον στα όρια των πινάκων αλλά ανακατεύονται με τα χρήσιμα στοιχεία.

Η τιμή των διαστάσεων του πίνακα και των tiles για το στοιχείο (i, j) υπολογίζονται ως εξής (Θεωρούμε ότι το μέγεθος των tiles είναι $step \times step$):

tile co-ordinates : $tile_{num} = \overleftarrow{(i \; div \; step)} | \overrightarrow{(j \; div \; step)}$



Σχήμα 3.4: Ο μετασχηματισμός tiling του πίνακα για τη διάταξη L_{MO}



Σχήμα 3.5: Ο 4-διάστατος πίνακας κατά τη διάταξη L_{MO}

$$
\begin{aligned}
& t_i = tile_{num}\ div\ n_t \\
& t_j = tile_{num}\ mod\ n_t \\
\text{tile offsets :}\quad & f_i = i\ mod\ step \\
& f_j = j\ mod\ step
\end{aligned}
$$

όπου $\overleftarrow{x}$ και $\overrightarrow{x}$ είναι η παράσταση περιττής και άρτιας αραίωσης του x, αντίστοιχα και n_t είναι ο αριθμός των tiles που χωρούν ανά γραμμή ή ανά στήλη του "παραγεμισμένου" πίνακα (οι "παραγεμισμένοι" πίνακες είναι πάντοτε τετραγωνικοί, όπως απαιτεί η διάταξη Morton).

Για την εφαρμογή της διάταξης L_{MO}, ο κώδικας οφείλει να μετασχηματιστεί ως εξής:

```
for (ii = 0; ii < NNodd; ii = (ii − oddBits)&oddBits) {
  if (((ii − oddBits)&oddBits) == NNodd)
    stepi = (step_b == 0?step : step_b);
  else stepi = step;
  for (kk = 0; kk < NNeven; kk = (kk − evenBits)&evenBits) {
    if (((kk − evenBits)&evenBits) == NNeven)
      stepk = (step_b == 0?step : step_b);
    else stepk = step;
    x = ii|kk;
    tiA = x/NN;
    tkA = x%NN;
    for (jj = 0; jj < NN; jj++) {
      if (((jj − evenBits)&evenBits) == NNeven)
        stepj = (step_b == 0?step : step_b);
      else stepj = step;
      y = ii|jj;
      tiC = y/NN;
      tjC = y%NN;
      z = (kk << 1)|kk;
      tkB = z/NN;
      tjB = z%NN;
      for (i=0; i < stepi; i++)
        for (k=0; k < stepk; k++)
          for (j=0; j < stepj; j++)
            C[tiC][tjC][i][j]+ = A[tiA][tkA][i][k] ∗ B[tkB][tjB][k][j];
    }
  }
}
```

όπου $step\_b = N\%step$

$NNodd = \overleftarrow{NN}$

$NNeven = \overrightarrow{NN}$

Ο τελικός κώδικας δεν περιέχει χρονοβόρους υπολογισμούς για την εύρεση των ζητούμενων στοιχείων. Ακόμα και στη διάταξη L_{MO}, περιέχονται δυαδικές πράξεις, οι οποίες προσθέτουν την ελάχιστη δυνατή επιβάρυνση. Ωστόσο, η αναφορά σε 4-διάστατους πίνακας δίνει μεγάλους κώδικες assembly, και επομένως, επαναληπτικές εντολές φόρτωσης από τη μνήμη και πρόσθεσης (για την εύρεση της ζητούμενης διεύθυνσης), οι οποίες, όπως δείχνουν τα πειραματικά αποτελέσματα, είναι χρονοβόρες και μειώνουν σημαντικά τη συνολική επίδοση.

Παράδειγμα 3.3:

Έστω ότι αναζητούμε και πάλι το στοιχείο

$$C[i,j] = C[11,24]$$

Για τον υπολογισμό της θέσης αποθήκευσής του, ακολουθούμε τη διαδικασία που περιγράφεται στη συνέχεια και απεικονίζεται στο σχήμα 3.4. Θεωρούμε ότι ο πίνακας C μεγέθους 27×27 ακολουθεί τη διάταξη L_{MO}, και είναι χωρισμένος σε tiles μεγέθους 4×4.

Tile coordinates:

$$tile_{num} = \overleftarrow{(i\ div\ step)}|\overrightarrow{(j\ div\ step)} = \overleftarrow{(11 div 4)}|\overrightarrow{(24 div 4)} = \overleftarrow{2}\,|\,\overrightarrow{6}$$

Στη συνέχεια, "αραιώνουμε" τους παραπάνω δείκτες:

$$even - dilation\text{: } \overleftarrow{2_{<10>}} = \overleftarrow{010_{<2>}} = 001000_{<2>} = 8_{<10>}$$
$$odd - dilation\text{: } \overrightarrow{6_{<10>}} = \overrightarrow{110_{<2>}} = 010100_{<2>} = 20_{<10>}$$
$$tile_{num} = \overleftarrow{2}\,|\,\overrightarrow{6} = 8 + 20 = 28$$

$$t_i = tile_{num}\ div\ n_t = 28\ div\ 8 = 3_{<10>}$$
$$t_j = tile_{num}\ mod\ n_t = 28\ mod\ 8 = 4_{<10>}$$

Επομένως, το ζητούμενο στοιχείο βρίσκεται στο tile #28 με συντεταγμένες $[3,4]$

Tile offsets:

$$f_i = i\ mod\ step = 11\ mod\ 4 = 3_{<10>}$$
$$f_j = j\ mod\ step = 24\ mod\ 4 = 0_{<10>}$$

Επομένως, στο εσωτερικό του tile, το ζητούμενο στοιχείο βρίσκεται στη θέση $[3,0]$

Στη διάταξη L_{MO}, το στοιχείο $C[11,24]$ του αρχικού πίνακα, αποθηκεύεται σε έναν τετραδιάστατο πίνακα και πρέπει να αναζητηθεί στη θέση [3][4][3][0] (σχήμα 3.5). ◇

3.4 Η προσέγγισή μας: μη-γραμμική διάταξη αποθήκευσης με χρήση δυαδικών μασκών

Κατά την εκτέλεση μετασχηματισμένου κώδικα με tiling, η σειρά προσπέλασης των στοιχείων των πινάκων δεν ακολουθεί τη σειρά αποθήκευσης των στοιχείων αυτών στη μνήμη στις γραμμικές διατάξεις δεδομένων. Στους μετασχηματισμένους κώδικες, προσπελάζονται διαδοχικά δεδομένα που ανήκουν στο ίδιο tile. Επομένως, προκειμένου να βελτιστοποιηθεί η τοπικότητα δεδομένων στις κρυφές μνήμες, τα στοιχεία του πίνακα που ανήκουν στο ίδιο tile απαιτείται να αποθηκεύονται

σε γειτονικές θέσεις μνήμης. Η ιδανική περίπτωση για την αξιοποίηση της χωρικής τοπικότητας δεδομένων, είναι η ευθυγράμμιση της ακολουθίας προσπέλασης με τη διάταξη αποθήκευσης των δεδομένων στη μνήμη.

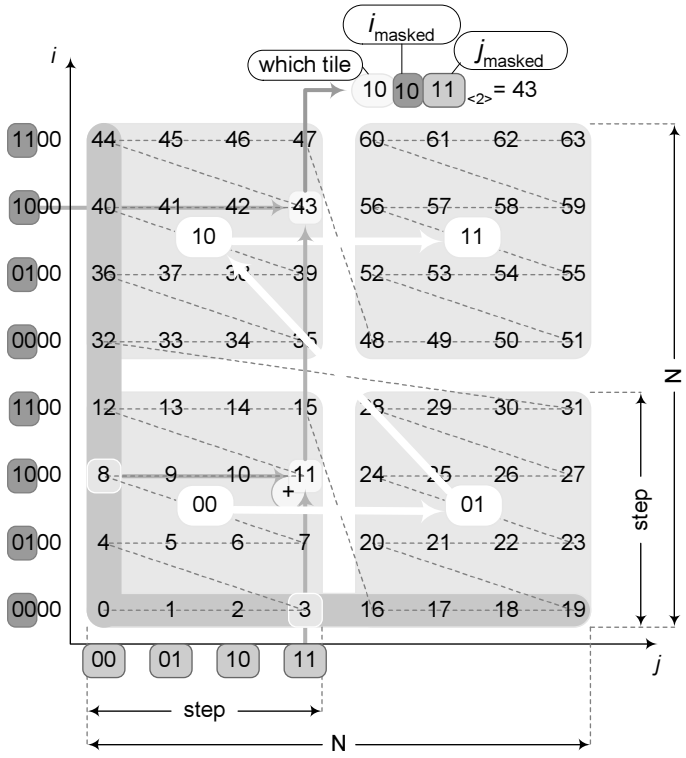
Η μη-γραμμική διάταξη L_{4D} αποθηκεύει τα δεδομένα των πινάκων με τρόπο αποδοτικό σε ότι αφορά την τοπικότητα δεδομένων που προκύπτει. Αποθηκεύονται με τη σειρά που τα προσπελάζει ο μετασχηματισμένος κώδικας, με χρήση tiling. Ωστόσο οι τετραδιάστατοι πίνακες που χρησιμοποιούνται κατά την υλοποίηση του L_{4D} είναι αργοί σε επίδοση, γιατί απαιτούν τον υπολογισμό της τιμής τεσσάρων διαφορετικών δεικτών και τετραπλή αναφορά στη μνήμη, για την προσπέλαση κάθε στοιχείου του πίνακα.

Από την άλλη πλευρά, οι πίνακες Morton έχουν μία γρήγορη στον υπολογισμό της δεικτοδότηση των δεδομένων των πινάκων, χρησιμοποιώντας την άλγεβρα των "αραιωμένων" ακεραίων. Οι ίδιοι οι πίνακες, ωστόσο, είναι πλήρως αναδρομικοί, και η σειρά αποθήκευσης των στοιχείων δεν ταιριάζει με τους μη αναδρομικούς κώδικες (δεν επιφέρει καλή τοπικότητα κατά τις αναφορές σε δεδομένα). Επιπλέον, οι μη αναδρομικοί κώδικες είναι δύσκολο ή και μη αποδοτικό να μετατραπούν σε αναδρομικούς.

Στο κεφάλαιο αυτό υιοθετούμε τις μη-γραμμικές διατάξεις δεδομένων και επεκτείνουμε τη θεωρία δεικτοδότησης με "αραιωμένους" ακεραίους, για να εφαρμόζεται στην περίπτωση των μη αναδρομικών κωδίκων. Επομένως, συνδυάζουμε την αποδοτική μη-γραμμική διάταξη αποθήκευσης των στοιχείων των πινάκων στη μνήμη (που ταιριάζει με τη σειρά προσπέλασης των στοιχείων από τον μετασχηματισμένο κώδικα με χρήση tiling), με ένα γρήγορο σχήμα δεικτοδότησης των στοιχείων των μετασχηματισμένων πινάκων που προκύπτουν. Η προτεινόμενη διάταξη αποθήκευσης απεικονίζεται στο σχήμα 3.6 για έναν πίνακα μεγέθους 8×8 στοιχείων, ο οποίος έχει διαιρεθεί σε tiles μεγέθους 4×4 στοιχείων. Στο σχήμα, η γκρι διακεκομμένη γραμμή ακολουθεί τη σειρά προσπέλασης των στοιχείων του πίνακα, και η αρίθμηση υποδεικνύει τη σειρά αποθήκευσής τους στη μνήμη. Γίνεται φανερό ότι η σειρά προσπέλασης συμφωνεί με τη σειρά αποθήκευσης των δεδομένων στη μνήμη. Τα στοιχεία των πινάκων κατά την αποθήκευσή τους στη μνήμη ομαδοποιούνται σε tiles, όπως ακριβώς στον μετασχηματισμό tiling του κώδικα. Το συγκεκριμένο παράδειγμα απεικονίζει τη διάταξη αποθήκευσης δεδομένων "ZZ" (τόσο η αποθήκευση των διαδοχικών στοιχείων στο εσωτερικό ενός tile, όσο και η διαδοχή των διαφορετικών, γειτονικών tiles γίνεται κατά γραμμές).

Η ονοματοδοσία των διαφορετικών διατάξεων βασίζεται στη σχηματική παράσταση της σειράς αποθήκευσης στη μνήμη των στοιχείων των πινάκων. Για την περίπτωση της διάταξης "ZZ": στο εσωτερικό των tiles η αποθήκευση γίνεται κατά γραμμές (σε διάταξη που σχηματικά μοιάζει με Z) και η μετάβαση από tile σε tile γίνεται επίσης κατά γραμμές (σε σχήμα Z).

Ο μετασχηματισμένος κώδικας που προσπελάζει τα στοιχεία ενός πίνακα A κατά τη διάταξη ZZ είναι ο εξής:

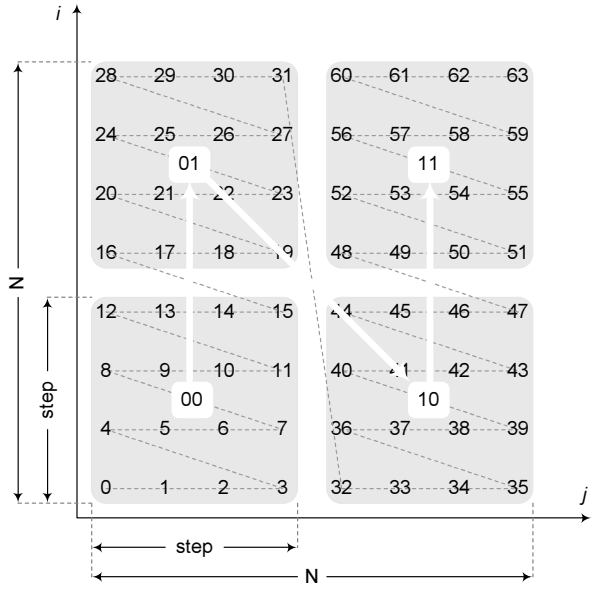


Σχήμα 3.6: Η διάταξη ZZ

"ZZ"
```
for (ii=0; ii < N; ii+=step)
  for (jj=0; jj < N; jj+=step)
    for (i=ii; (i < ii + step && i < N); i++)
      for (j = jj; (j < jj+step && j < N); j++)
        A[i,j]=...;
```

Τα υπόλοιπα τρία είδη μετασχηματισμών δεδομένων φαίνονται στα σχήματα 3.7, 3.8 και 3.9. Για παράδειγμα, σύμφωνα με τη διάταξη "NN", τόσο η αποθήκευση στο εσωτερικό των tiles όσο και η μετάβαση από tile σε tile γίνεται κατά στήλες (σε σχήμα N). Οι αντίστοιχοι κώδικες προσπέλασης αναγράφονται στη συνέχεια:

"NZ"
```
for (jj=0; jj < N; jj+=step)
  for (ii=0; ii < N; ii+=step)
    for (i = ii; (i < ii + step && i < N); i++)
      for (j = jj; (j < jj + step && j < N); j++)
        A[i,j]=...;
```

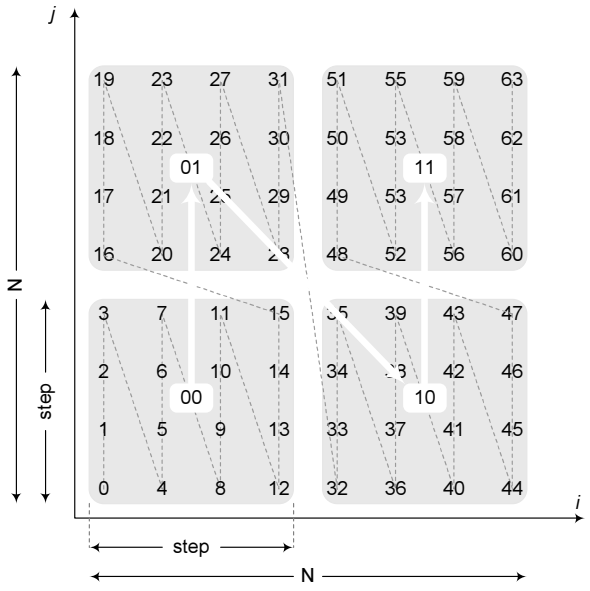


Σχήμα 3.7: Η διάταξη NZ

"NN"
```
for (ii=0; ii < N; ii+=step)
  for (jj=0; jj < N; jj+=step)
    for (i = ii; (i < ii + step && i < N); i++)
      for (j = jj; (j < jj + step && j < N); j++)
        A[i,j]=...;
```

"ZN"
```
for (jj=0; jj < N; jj+=step)
  for (ii=0; ii < N; ii+=step)
    for (i = ii; (i < ii + step && i < N); i++)
      for (j = jj; (j < jj+step && j < N); j++)
        A[i,j]=...;
```

Οι σημερινοί μεταγλωττιστές υποστηρίζουν μόνο τις γραμμικές διατάξεις δεδομένων (αποθήκευση των διδιάστατων ή γενικότερα των πολυδιάστατων πινάκων κατά γραμμές ή κατά στήλες). Για το λόγο αυτό, στις μη γραμμικές διατάξεις που περιγράψαμε, τα στοιχεία των πινάκων πρέπει να αποθηκεύονται σε μονοδιάστατους πίνακες. Για τη δεικτοδότησή τους χρησιμοποιούνται ακέραιοι δείκτες των οποίων οι τιμές είναι κατάλληλα "αραιωμένες" (dilated integers). Η δεικτοδότηση κατά Morton [WF99] δεν εφαρμόζεται στην περίπτωσή μας, γιατί προϋποθέτει αναδρομικό μετασχηματισμό tiling. Στην περίπτωσή μας, η προτεινόμενη μη γραμμική διάταξη δεδομένων εφαρμόζει μετασχηματισμό tiling για ένα μόνο επίπεδο της ιεραρχίας μνήμης. Έτσι, κατασκευάζουμε δυαδικές μάσκες για τη μορφοποίηση των δεικτών, πιο πολύπλοκες από τα *oddBits* και *evenBits* του Morton, αλλά γρήγορες στον υπολογισμό τους, γιατί περιλαμβάνουν μόνο λογικές πράξεις επί ακεραίων αριθμών.



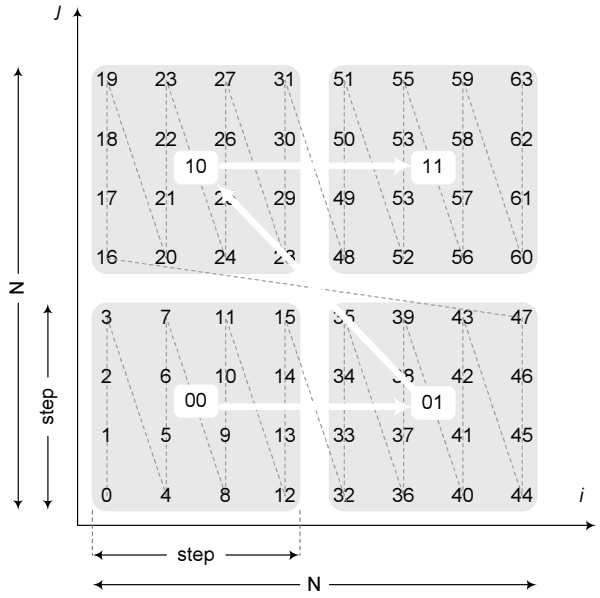
Σχήμα 3.8: Η διάταξη NN

3.5 Η Θεωρία Μασκών

Θεωρούμε έναν πίνακα $A[i,j]$ μεγέθους $N_i \times N_j$ στοιχείων, ο οποίος έχει διαιρεθεί σε tiles μεγέθους $step_i \times step_j$. Για τη μορφοποίηση των δεικτών, χρησιμοποιούνται οι μάσκες που φαίνονται στον πίνακα που ακολουθεί.

Ο αριθμός των συνεχόμενων 0 και 1 που αποτελούν τα επιμέρους τμήματα των μασκών, καθορίζεται από τις συναρτήσεις $m_x = log(step_x)$ και $t_x = log\left(\frac{N_x}{step_x}\right)$, όπου η διάσταση $step$ του tile επιλέγεται να είναι κάποια δύναμη του 2. Επιπρόσθετα, αν η διάσταση του πίνακα N δεν είναι δύναμη του 2, δεσμεύεται για τον πίνακα A χώρος που αντιστοιχεί στον αμέσως μεγαλύτερο πίνακα διάστασης $N = 2^n$. Τα πλεονάζοντα στοιχεία παίρνουν τυχαίες τιμές (αποτελούν ουσιαστικά "παραγεμίσματα" - padding). Η αύξηση του μεγέθους του πίνακα δεν επιβαρύνει το συνολικό χρόνο εκτέλεσης του προγράμματος, γιατί τα πλεονάζοντα στοιχεία δεν προσπελάζονται κατά την εκτέλεση του κώδικα.

Κάθε στοιχείο του αρχικού πίνακα $A[i,j]$ αποθηκεύεται σε μία θέση του μονοδιάστατου μετασχηματισμένου πίνακα. Η θέση αυτή είναι η $[i_m + j_m] = [i_m | j_m]$, όπου οι δείκτες i_m, j_m προκύπτουν με φιλτράρισμα/"αραίωμα" (μέσω κατάλληλων μασκών) της τιμής των αρχικών δεικτών i, j.



Σχήμα 3.9: Η διάταξη ZN

μετασχηματισμός ZZ

δείκτης γραμμών	00..0	11..1	00..0	11..1
δείκτης στηλών	11..1	00..0	11..1	00..0
	$\leftarrow t_i \rightarrow$	$\leftarrow t_j \rightarrow$	$\leftarrow m_i \rightarrow$	$\leftarrow m_j \rightarrow$

μετασχηματισμός NZ

δείκτης γραμμών	11..1	00..0	00..0	11..1
δείκτης στηλών	00..0	11..1	11..1	00..0
	$\leftarrow t_j \rightarrow$	$\leftarrow t_i \rightarrow$	$\leftarrow m_i \rightarrow$	$\leftarrow m_j \rightarrow$

μετασχηματισμός NN

δείκτης γραμμών	11..1	00..0	11..1	00..0
δείκτης στηλών	00..0	11..1	00..0	11..1
	$\leftarrow t_j \rightarrow$	$\leftarrow t_i \rightarrow$	$\leftarrow m_j \rightarrow$	$\leftarrow m_i \rightarrow$

μετασχηματισμός ZN

δείκτης γραμμών	00..0	11..1	11..1	00..0
δείκτης στηλών	11..1	00..0	00..0	11..1
	$\leftarrow t_i \rightarrow$	$\leftarrow t_j \rightarrow$	$\leftarrow m_j \rightarrow$	$\leftarrow m_i \rightarrow$

εξαρτάται από τον τρόπο $\hookleftarrow$ $\hookrightarrow$ εξαρτάται από τη διάταξη
μετάβασης από tile σε tile του εσωτερικού των tiles

3.6 Υλοποίηση

Ας υποθέσουμε ότι τα δεδομένα ενός πίνακα $A[i,j]$ αποθηκεύονται κατά τη διάταξη ZZ (κεφάλαιο 3.4), σύμφωνα με τη σειρά προσπέλασής τους από μία μετασχηματισμένη ακολουθία εντολών. Το σχήμα 3.10 παρουσιάζει έναν μετασχηματισμό δεδομένων ενός διδιάστατου πίνακα σε μονοδιάστατη μη-γραμμική διάταξη αποθήκευσης. Για τη δεικτοδότηση των μονοδιάστατων πλέον πινάκων είναι σημαντικό να μην απαιτούνται πολύπλοκοι και χρονοβόροι υπολογισμοί, οι οποίοι επιβαρύνουν τη συνολική επίδοση του εκάστοτε προγράμματος που χρησιμοποιεί μη-γραμμικούς μετασχηματισμούς δεδομένων. Για το λόγο αυτό, αντί να μορφοποιούνται ("αραιώνονται") οι αντίστοιχοι δείκτες κάθε φορά που γίνεται αναφορά σε στοιχεία μετασχηματισμένων πινάκων, οι δείκτες παίρνουν εξαρχής μη γραμμικές τιμές. Οι τιμές των δεικτών i και j που ελέγχουν αντίστοιχα τις στήλες και τις γραμμές του πίνακα A (στο παράδειγμά μας το μέγεθος πίνακα είναι $= 8 \times 8$, και η διάσταση του tile $step = 4$), φαίνονται στον πίνακα που ακολουθεί.

γραμμική αρίθμηση δεικτών για τις στήλεςγραμμές :		0	1	2	3	4	5	6	7
μορφοποιημένοι δείκτες	column i :	0	4	8	12	32	36	40	44
πινάκων βρόχων	row j :	0	1	2	3	16	17	18	19

Παράδειγμα 3.4:

Έστω ότι αναζητούμε το στοιχείο

$$A[i,j] = A[2,3]$$

Για τον υπολογισμό της θέσης αποθήκευσής του, ακολουθούμε τη διαδικασία που περιγράφεται στη συνέχεια και απεικονίζεται στο σχήμα 3.10 και 3.11. Θεωρούμε ότι ο πίνακας A μεγέθους $N_i \times N_j = 8 \times 8$ ακολουθεί τη διάταξη ZZ, και είναι χωρισμένος σε tiles μεγέθους $step_i \times step_j = 4 \times 4$.

Στο παράδειγμά μας η κατάλληλη δυαδική μάσκα (ενότητα 3.5) για την εύρεση της θέσης των στοιχείων του πίνακα A, είναι:

για το δείκτη γραμμών j, $m_j = 010011$

για το δείκτη στηλών i, $m_i = 101100$

Φιλτράροντας τις τιμές 0-7 των γραμμικών δεικτών (οι τιμές που παίρνουν οι δείκτες i και j για τη δεικτοδότηση των γραμμικών διατάξεων δεδομένων), μέσω των μασκών αυτών (όπως καταγράφεται αναλυτικά στο σχήμα 3.11), προκύπτουν οι τιμές δεικτοδότησης των μη-γραμμικών διατάξεων δεδομένων.

$$j_m = masked_{010011}(3_{<10>} = masked_{010011}(011_{<2>} = 000011_{<2>} = 3_{<10>}$$

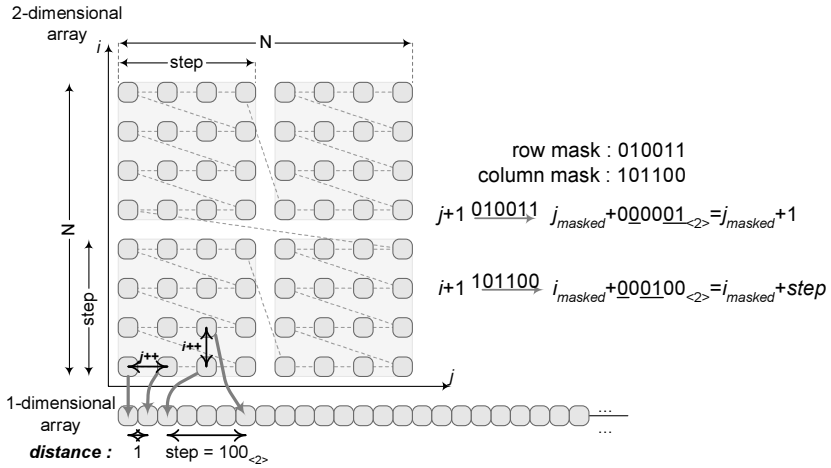
$$i_m = masked_{101100}(2_{<10>} = masked_{101100}(010_{<2>} = 001000_{<2>} = 8_{<10>}$$

Τελικά, η εύρεση της θέσης του ζητούμενου στοιχείου γίνεται απλά με πρόσθεση των δύο δεικτών:

$$A[i_m + j_m] = A[8 + 3] = A[11]$$

Όπου i_m και j_m είναι οι μορφοποιημένοι δείκτες i και j, αντίστοιχα. ◇

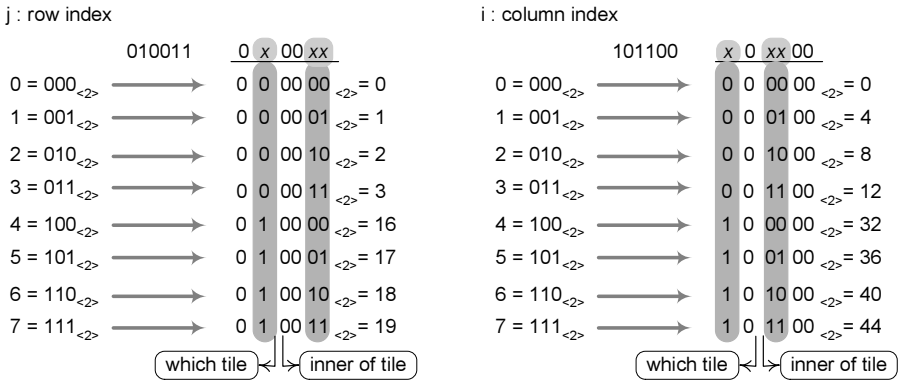
Στην πράξη, οι μη γραμμικές τιμές των δεικτών είναι ίσες με τη θέση (μέσα στο μονοδιάστατο μετασχηματισμένο πίνακα) των στοιχείων της πρώτης στήλης και της πρώτης γραμμής (σχήμα 3.6). Επομένως, για την αναφορά στο στοιχείο της 2ης γραμμής και 3ης στήλης του αρχικού πίνακα, δηλαδή του $A[2,3]$, οι τιμές των δεικτών για τον μονοδιάστατο πίνακα θα πρέπει να είναι i=8, j=3.



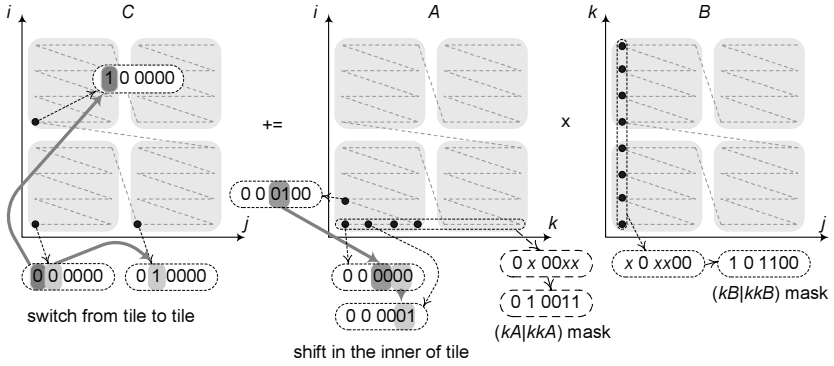
Σχήμα 3.10: Μετατροπή ενός 2-διάστατου πίνακα σε 1-διάστατο κατά τη διάταξη ZZ

3.7 Παράδειγμα : Πολλαπλασιασμός Πινάκων

Σύμφωνα με τους Kandemir κλπ στην εργασία [KRC99], η βέλτιστη τοπικότητα δεδομένων επιτυγχάνεται όταν ο κώδικας μετασχηματιστεί ως εξής:



Σχήμα 3.11: Μετατροπή των γραμμικών δεικτών για γραμμές και στήλες στις αντίστοιχες "αραιωμένες" τιμές τους μέσω κατάλληλων μασκών



Σχήμα 3.12: Ο Πολλαπλασιασμός πινάκων: $C[i,j] += A[i,k] * B[k,j]$

```
for (ii=0; ii < 8; ii+=4)
  for (kk=0; kk < 8; kk+=4)
    for (jj=0; jj < 8; jj+=4)
      for (i = ii; (i < ii+4 && i < 8); i++)
        for (k = kk; (k < kk+4 && k < 8); k++)
          for (j = jj; (j < jj+4 && j < 8); j++)
            C[i,j]+=A[i,k]*B[k,j];
```

Χρησιμοποιούμε μία τροποποιημένη έκδοση του [KRC99], εφαρμόζοντας μετασχηματισμό tiling σε όλους τους βρόχους. Έτσι, το τελικό βάθος του φωλιασμένου βρόχου που προκύπτει στον μετασχηματισμένο κώδικα είναι 6, αντί 5. Η υλοποίηση της θεωρίας μασκών είναι απλούστερη στην περίπτωση αυτήν, γιατί όλοι οι πίνακες διαιρούνται σε τετραγωνικά tiles ίδιων διαστάσεων. Έτσι, και οι τρεις πίνακες A, B, C προσπελάζονται κατά τη διάταξη ZZ και τα στοιχεία του δεικτοδοτούνται από ομοιόμορφους μη-γραμμικούς (φιλτραρισμένους) δείκτες (σχήμα 3.12).

Στο φωλιασμένο κώδικα του παραδείγματός μας, οι τρεις εσωτερικότεροι βρόχοι

(i, j, k) ελέγχουν την σειρά προσπέλασης των δεδομένων στο εσωτερικό των tiles. Τα 4 λιγότερο σημαντικά ψηφία των μασκών εμπεριέχουν τη δεικτοδότηση όλων των επαναλήψεων των συγκεκριμένων βρόχων, όπως φαίνεται στο σχήμα 3.11. Οι τρεις εξωτερικότεροι βρόχοι (ii, kk, jj) ελέγχουν τη μετάβαση από tile σε tile, και τα 2 πιο σημαντικά ψηφία των μασκών καθορίζουν τη μετακίνηση από το ένα tile στο γειτονικό του.

δείκτες :	έλεγχος	πίνακας(ες)	κατάλληλη μάσκα
i, ii :	στηλών	A & C	μάσκα στηλών (101100)
j, jj :	γραμμών	B & C	μάσκα γραμμών (010011)
k, kk :	στηλών	B	μάσκα στηλών (για τους kB, kkB)
k, kk :	γραμμών	A	μάσκα γραμμών (για τους $kA, kkjA$)

Πίνακας 3.1: Δεικτοδότηση πινάκων

Επομένως, ο δείκτης i παίρνει τιμές εντός του πεδίου $xx00$ όπου $x = (0 \text{ or } 1)$ και ο ii παίρνει τιμές στο διάστημα $x00000$. Το άθροισμα $i|ii = x0xx00$ δίνει τις επιθυμητές τιμές για τις στήλες των πινάκων A και C. Ομοίως, ο δείκτης j παίρνει τιμές στο διάστημα $00xx$ και ο jj στο διάστημα $0x0000$, και έτσι το άθροισμα $(j|jj)$ είναι η επιθυμητή τιμή για τις γραμμές του B και του C. Ο δείκτης k δεν ελέγχει το ίδιο είδος διάστασης στους πίνακες στους οποίους εμπλέκεται, όπως φαίνεται και στον πίνακα 3.1. Έτσι, για τη δεικτοδότηση του πίνακα A, η κατάλληλη μάσκα είναι μία μάσκα γραμμών, δηλαδή $kA \in 00xx$ και $kkA \in 0x0000$. Από την άλλη πλευρά, για τη δεικτοδότηση του πίνακα B η κατάλληλη μάσκα είναι μία μάσκα στηλών. Επομένως, $kB \in xx00$ και $kkB \in x00000$. Μία χρήσιμη παρατήρηση είναι ότι η μετατροπή ενός δείκτη γραμμών σε δείκτη στηλών (στην διάταξη ZZ) γίνεται εύκολα, με μη χρονοβόρους υπολογισμούς ως εξής: $kB = kA << logstep$ και $kkB = kkA << log\left(\frac{N}{step}\right)$.

Για το παράδειγμά μας,

$ibound$=column_mask=$101100_{<2>} = 44$,

$iiincrement = 100000_{<2>} = 32 = 4 \times 4 << \frac{N}{step}$,

$iincrement = 100_{<2>} = 4 = step$

and $ireturn = min\{\text{column\_mask}, ii|1100_{<2>}\}$.

Ο μετασχηματισμένος κώδικας με τις μορφοποιημένες τιμές δεικτών έχει τη μορφή που φαίνεται παρακάτω:

```
for (ii=0; ii < ibound; ii+=iiincrement){
  itilebound=(ii|imask)+1;
  ireturn=(ibound < itilebound?ibound : itilebound);
  for (kk=0; kk < kjbound; kk+=kkjjincrement){
    ktilebound=(kk|kjmask)+1;
    kreturn=(kjbound < ktilebound?kjbound : ktilebound);
    kkB=kk << logNxy;
    for (jj=0; jj < kjbound; jj+=kkjjincrement) {
      jtilebound=(jj|kjmask)+1;
      jreturn=(kjbound < jtilebound?kjbound : jtilebound);
      for (i=ii; i < ireturn; i+=iincrement)
        for (k=kk; k < kreturn; k+ = kjincrement) {
          kB=(k & (step_1))<< logstep;
          ktB=kkB|kB;
          xA=i|k;
          for (j=jj; j < jreturn; j+=kjincrement)
            C[i|j]+=A[xA] * B[ktB|j];
        }
    }
  }
}
```

3.8 Ο αλγόριθμος

Για την εύρεση του βέλτιστου συνδυασμού μετασχηματισμών κώδικα και δεδομένων, με στόχο τη μεγιστοποίηση της επίδοσης, παρουσιάζουμε μία στρατηγική βασισμένη στον αλγόριθμο του [KRC99], προσαρμοσμένο στις ανάγκες των μη γραμμικών διατάξεων δεδομένων. Σημειώνουμε ότι ο αλγόριθμός μας βρίσκει επίσης τη βέλτιστη διάταξη δεδομένων για κάθε πίνακα, λαμβάνοντας υπόψη όλα τα στιγμιότυπά του σε ολόκληρο το πρόγραμμα, ακόμα και αν πρόκειται για στιγμιότυπα που βρίσκονται σε διαφορετικούς φωλιασμένους βρόχους. Κατά αυτόν τον τρόπο δε χρειάζεται να δημιουργούμε πολλαπλά αντίγραφα για τους πίνακες που έχουν περισσότερα από ένα στιγμιότυπα. Επομένως, υπάρχει ένας και μοναδικός μετασχηματισμένος πίνακας για κάθε έναν από τους πίνακες του προγράμματος, αποφεύγοντας τις χρονοβόρες μετατροπές από τη μία μορφή διάταξης στην άλλη. Η προτεινόμενη διαδικασία εύρεσης του βέλτιστου συνδυασμού μετασχηματισμών είναι η ακόλουθη:

- Κατασκευάζουμε έναν πίνακα R μεγέθους $r \times l$, όπου $r =$ *αριθμός των διαφορετικών αναφορών σε πίνακες* και $l =$ *βάθος του φωλιασμένου βρόχου* (πριν την εφαρμογή του μετασχηματισμού tiling). Αν υπάρχουν δύο πανομοιότυπες αναφορές σε έναν πίνακα, δεν χρειάζεται

να εισάγουμε νέα γραμμή για αυτήν. Σημειώνουμε απλά έναν αστερίσκο, για να υποδεικνύει προτεραιότητα στη βελτιστοποίηση της αναφοράς στο συγκεκριμένο πίνακα, λόγω των πολλαπλών εμφανίσεών της. Για το παράδειγμά μας, ο πίνακας R είναι ο εξής:

$$R = \begin{array}{c} \begin{matrix} i & k & j \end{matrix} \\ \left[\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \text{C**} \\ \text{A} \\ \text{B} \end{array}$$

Σημειώνουμε επίσης, ότι οι πίνακες που βρίσκονται στο αριστερό μέρος μίας παράστασης (ο πίνακας C του παραδείγματός μας) έχουν μεγαλύτερη βαρύτητα, επειδή σε κάθε επανάληψη το περιεχόμενο των στοιχείων τους διαβάζεται και γράφεται.

Κάθε στήλη του R αντιπροσωπεύει έναν από τους δείκτες του φωλιασμένου βρόχου. Τα στοιχεία του R παίρνουν τιμή ίση με 1 όταν ο αντίστοιχος δείκτης ελέγχει κάποια από τις διαστάσεις του συγκεκριμένου πίνακα. Διαφορετικά, τα στοιχεία του R παίρνουν τιμή ίση με 0.

- Βελτιστοποιούμε πρώτα τη διάταξη των στοιχείων του πίνακα με το μεγαλύτερο αριθμό αναφορών στο φωλιασμένο βρόχο. (Πρόκειται για τον πίνακα C του παραδείγματός μας.) Οι εφαρμοζόμενοι μετασχηματισμοί βρόχων θα πρέπει να είναι τέτοιοι που να φέρνουν στην εσωτερικότερη θέση έναν από τους δείκτες που ελέγχουν μία διάσταση του πίνακα. Δηλαδή, η γραμμή του R που αντιστοιχεί στον πίνακα C θα πρέπει να πάρει τη μορφή $(x, \ldots, x, 1)$, όπου $x = 0$ ή 1. Για το σκοπό αυτό, ίσως χρειαστεί αντιμετάθεση στηλών (εφόσον βέβαια είναι επιτρεπτή από τις εξαρτήσεις δεδομένων η αντιμετάθεση των αντίστοιχων βρόχων). Ο επιλεγμένος δείκτης είναι προτιμότερο να είναι το μόνο στοιχείο που ελέγχει τη συγκεκριμένη διάσταση, και επιπλέον, να μην εμφανίζεται σε οποιαδήποτε άλλη διάσταση του πίνακα C.

  Στη συνέχεια, προκειμένου να αξιοποιηθεί η χωρική τοπικότητα δεδομένων της αναφοράς αυτής, ο πίνακας C θα πρέπει να αποθηκευτεί στη μνήμη κατά τη διάσταση (έστω την z-οστή) που ελέγχεται από τον εσωτερικότερο δείκτη του φωλιασμένου βρόχου. Ασφαλώς, όλοι οι δείκτες που ελέγχουν κάποια διάσταση του C θα πρέπει να ελεγχθούν εναλλακτικά αν ικανοποιούν τις παραπάνω συνθήκες και είναι ωφέλιμο να μπουν στην εσωτερικότερη θέση.

- Ρυθμίζουμε τις υπόλοιπες αναφορές σε πίνακες δεδομένων με σειρά προτεραιότητας. Ο στόχος είναι να έρθουν όσο το δυνατόν περισσότερες γραμμές του R στη μορφή $(x, \ldots, x, 1)$. Κατά αυτόν τον τρόπο, αν έστω η y-οστή διάσταση του A ελέγχει από έναν δείκτη βρόχου,

κατά πανομοιότυπο τρόπο όπως η z-οστή διάσταση του C, τότε αποθήκευσε τον πίνακα A έτσι ώστε η ταχύτερα μεταβαλλόμενη διάστασή του να είναι η y-οστή. Αν δεν υπάρχει τέτοια διάσταση για τον A, τότε θα πρέπει να μετασχηματιστεί η αναφορά σε αυτήν, έτσι ώστε να πάρει τη μορφή $A[*,\ldots,*,f(i_{in-1}),*,\ldots,*]$, όπου $f(i_{in-1})$ είναι συνάρτηση του δεύτερου εσωτερικότερου βρόχου i_{in-1} και άλλων δεικτών εκτός του εσωτερικότερου i_{in}. Το σύμβολο $*$ προσδιορίζει όρους ανεξάρτητους, τόσο από τον i_{in-1}, όσο και τον i_{in}. Επομένως, η γραμμή του R που αφορά τον πίνακα A θα πρέπει να πάρει τη μορφή $(x,\ldots,x,1,0)$, έτσι ώστε να αξιοποιηθεί η χωρική τοπικότητα δεδομένων κατά μήκος του βρόχου i_{in-1}. Αν δεν είναι δυνατόν ένας τέτοιος μετασχηματισμός, θα πρέπει να επιχειρήσουμε το ίδιο για τον τρίτο εσωτερικότερο δείκτη i_{in-2}, κ.ο.κ. Αν δεν επιτύχουμε την επιθυμητή μορφή με κανέναν από τους δείκτες, τότε αυτοί διατάσσονται τυχαία.

- Μετά από έναν πλήρη μετασχηματισμό βρόχων και την αντίστοιχη επιλογή της διάταξης αποθήκευσης των πινάκων, καταγράφεται η λύση και δοκιμάζεται ο επόμενος εναλλακτικός συνδυασμός. Μεταξύ όλων των δυνατών λύσεων επιλέγεται εκείνη που αξιοποιεί τη χωρική τοπικότητα δεδομένων στον εσωτερικότερο βρόχο, για το μεγαλύτερο αριθμό αναφορών σε πίνακες.

- Όταν έχει πλέον καθοριστεί η τελική σειρά των βρόχων του κώδικα, εφαρμόζουμε τον μετασχηματισμό tiling. Σε αυτό το στάδιο, μπορεί πλέον να καθοριστεί πλήρως η διάταξη αποθήκευσης των στοιχείων των πινάκων. Σε κάθε μία από τις αναφορές σε πίνακες δεδομένων, η διάσταση που ορίστηκε στο πρώτο βήμα του αλγορίθμου ως η γρηγορότερα μεταβαλλόμενη, πρέπει να παραμείνει τέτοια σε ότι αφορά το εσωτερικό των tiles, δηλαδή θα αποτελεί τη διάσταση κατά μήκος της οποίας τα στοιχεία του πίνακα αποθηκεύονται σειριακά στο εσωτερικό των tiles. Επομένως, για έναν διδιάστατο πίνακα, η αναφορά στα δεδομένα του οποίου έχει τη μορφή $C[*,i_{in}]$, η διάταξη αποθήκευσης στο εσωτερικό του tile θα πρέπει να γίνεται κατά γραμμές. Επομένως, στο μη-γραμμικό μετασχηματισμό δεδομένων θα χρησιμοποιηθεί μία από τις διατάξεις τύπου xZ (η διάταξη ZZ ή η διάταξη NZ). Αν η αναφορά στα δεδομένα του πίνακα είναι της μορφής $C[i_{in},*]$, τότε η αποθήκευση των στοιχείων του πίνακα στο εσωτερικό του tile θα πρέπει να γίνεται κατά στήλες. Στο μη-γραμμικό μετασχηματισμό θα χρησιμοποιηθεί μία από τις διατάξεις τύπου xN (η διάταξη ZN ή η διάταξη NN). Η μετάβαση από tile σε tile, και επομένως το πρώτο γράμμα του ονόματος του εφαρμοζόμενου μη-γραμμικού μετασχηματισμού δεδομένων, εξαρτάται από το μετασχηματισμό του tiling που έχει επιλεχθεί. Σε ότι αφορά το παράδειγμα του διδιάστατου πίνακα C, αν στο δείκτη που ελέγχει τη διάσταση που σημειώνεται με * δεν έχει εφαρμοστεί tiling τότε θα επιλεγεί ένας από τους μετασχηματισμούς NZ ή ZN, αντίστοιχα. Διαφορετικά, θα επιλεγεί ένας από τους ZZ ή NN αντίστοιχα. Στις περισσότερες περιπτώσεις, η εφαρμογή του μετασχηματισμού tiling σε όλους τους βρόχους φέρνει ομοιομορφία στα σχήματα και στα μεγέθη των tiles. Κατά αυτόν τον τρόπο, η μορφοποίηση των δεικτών γίνεται ευκολότερα,

χρησιμοποιώντας λιγότερους υπολογισμούς. Το μέγεθος των επιλεγόμενων tiles εξαρτάται από τη χωρητικότητα του επιπέδου της κρυφής μνήμης που επιδιώκουμε να αξιοποιήσουμε, όπως περιγράφεται αναλυτικά στο κεφάλαιο 4.

3.9 Σύνοψη

Στις γραμμικές διατάξεις δεδομένων που προσπελάζονται από μετασχηματισμένους κώδικες με tiling, η ακολουθία προσπέλασης δε συμβαδίζει με τη σειρά αποθήκευσης των δεδομένων στη μνήμη. Σαν αποτέλεσμα του γεγονότος αυτού δεν αξιοποιείται η τοπικότητα δεδομένων, και επομένως, η επίδοση των μετασχηματισμένων δεδομένων δεν είναι η βέλτιστη. Στο κεφάλαιο αυτό εξετάσαμε την αποτελεσματικότητα των μη γραμμικών διατάξεων ομαδοποίησης δεδομένων και προτείναμε μία τεχνική δεικτοδότησης με χρήση δυαδικών μασκών. Παρόλο που ο τελικός μετασχηματισμένο κώδικας είναι πιο σύνθετος, η τεχνική δεικτοδότησης υλοποιείται με χαμηλό υπολογιστικό κόστος, σε ότι αφορά το χρόνο εκτέλεσης (αξιοποιώντας την άλγεβρα των "αραιωμένων" ακεραίων). Τα πειραματικά αποτελέσματα του κεφαλαίου 6, τόσο του χρόνου εκτέλεσης, όσο και της προσομοίωσης, επιδεικνύουν την αποτελεσματικότητα της προτεινόμενης τεχνικής.

ΚΕΦΑΛΑΙΟ 4

Επιλογή του βέλτιστου μεγέθους tile - Θεωρητική Ανάλυση

4.1 Εισαγωγή

Ο μετασχηματισμός tiling είναι ένας από τους πιο ευρέως χρησιμοποιούμενους μετασχηματισμούς κώδικα. Στο κεφάλαιο 3 μελετήσαμε τις μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων και προτείναμε τη δεικτοδότηση των πινάκων μέσω ενός απλού (στην υλοποίηση και στο χρόνο εκτέλεσης) σχήματος διευθυνσιοδότησης. Οι μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων σχεδόν εξαλείφουν τις αστοχίες σύγκρουσης μεταξύ των στοιχείων του ίδιου πίνακα, ενώ οι αστοχίες σύγκρουσης μεταξύ στοιχείων διαφορετικών πινάκων μπορούν να μειωθούν στο ελάχιστο, με κατάλληλη ευθυγράμμιση των πινάκων μεταξύ τους. Κατά αυτόν τον τρόπο, άλλοι παράγοντες που στις γραμμικές διατάξεις δεδομένων θεωρούνταν αμελητέοι, στην περίπτωσή μας μπορούν να καθορίσουν τη συμπεριφορά της κρυφής μνήμης και του TLB. Τέτοιοι παράγοντες είναι η πολυπλοκότητα του κώδικα, ο αριθμός των λανθασμένων προβλέψεων της απόφασης των εντολών άλματος και το ποσοστό της χρησιμοποίησης της κρυφής μνήμης. Καταφέραμε να μειώσουμε την πολυπλοκότητα του κώδικα σε ότι αφορά την προσπέλαση των δεδομένων που αποθηκεύονται στις μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων, χρησιμοποιώντας την τεχνική της γρήγορης δεικτοδότησης με τις δυαδικές μάσκες (κεφάλαιο 3). Τα πειραματικά αποτελέσματα αποδεικνύουν ότι επιτυγχάνουμε ελαχιστοποίηση του χρόνου εκτέλεσης όταν αξιοποιείται πλήρως η L1 κρυφή μνήμη. Στο σημείο της βέλτιστης επίδοσης το σύνολο των δεδομένων που περιέχονται σε ένα tile γεμίζουν πλήρως την L1 κρυφή μνήμη, ενώ το σύνολο των δεδομένων που επεξεργάζεται το πρόγραμμα στους εσωτερικότερους βρόχους ξεπερνούν λίγο το μέγεθος του συγκεκριμένου επιπέδου κρυφής μνήμης. Τέτοια μεγάλα tiles μειώνουν, επιπλέον, και τον αριθμό των λανθασμένων προβλέψεων της απόφασης των εντολών άλματος.

Στο κεφάλαιο αυτό μελετάμε τη συμπεριφορά των διαφόρων επιπέδων της κρυφής μνήμης και

των TLBs. Πιο συγκεκριμένα εξετάζουμε τις αστοχίες δεδομένων κατά την εκτέλεση του μικροπρογράμματος πολλαπλασιασμού πινάκων. Το μεγαλύτερο μέρος της ανάλυσης αφιερώνεται στην αρχιτεκτονική ενός Sun UltraSPARC II. Το μηχάνημα αυτό διαθέτει κρυφές μνήμες ευθείας απεικόνισης, το οποίο επιφέρει έναν μεγάλο αριθμό από αστοχίες δεδομένων λόγω σύγκρουσης και πολύπλοκες παραστάσεις υπολογισμού των αστοχιών. Καθόλη τη διάρκεια του κεφαλαίου θα θεωρούμε ότι τα στοιχεία των πινάκων αποθηκεύονται σύμφωνα με τις μη γραμμικές διατάξεις που περιγράφηκαν στο κεφάλαιο 3. Επομένως, τα δεδομένα που προσπελάζονται σε διαδοχικές επαναλήψεις του εσωτερικότερου βρόχου, βρίσκονται αποθηκευμένα σε γειτονικές θέσεις στη μνήμη. Οι μη γραμμικές διατάξεις δεδομένων απαλείφουν τις συγκρούσεις μεταξύ των στοιχείων του ίδιου πίνακα (self-conflict misses). Επιπλέον, υπενθυμίζουμε ότι εξετάζονται μόνο τα τετραγωνικά tiles για λόγους συμμετρίας και για την απλοποίηση του παραγόμενου μετασχηματισμένου κώδικα. Σαν αποτέλεσμα, η βελτιστοποίηση κώδικα φωλιασμένων βρόχων και η επιλογή του βέλτιστου μεγέθους tile, θα πρέπει να επικεντρώνεται στην ελαχιστοποίηση των υπόλοιπων παραγόντων που επηρεάζουν την επίδοση των προγραμμάτων. Η ανάλυση που ακολουθεί είναι μία προσπάθεια εντοπισμού των παραγόντων αυτών.

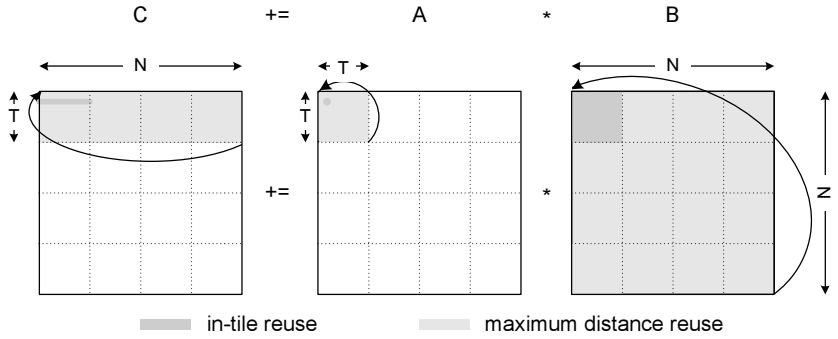
4.2 Η Επαναχρησιμοποίηση Δεδομένων στον Κώδικα του Πολλαπλασιασμού Πινάκων

Στη συνέχεια του παρόντος κεφαλαίου θα χρησιμοποιούμε το βελτιστοποιημένο κώδικα του πολλαπλασιασμού πινάκων, όπως περιγράφεται στις εργασίες [KRC99], [RT98b], [AK04a]: $C+ = A * B$

```
for (ii=0; ii < N; ii+=T)
  for (kk=0; kk < N; kk+=T)
    for (jj=0; jj < N; jj+=T)
      for (i = ii; (i < ii+T && i < N); i++)
        for (k = kk; (k < kk+T && k < N); k++)
          for (j = jj; (j < jj+T && j < N); j++)
            C[i,j]+=A[i,k]*B[k,j];
```

Το σχήμα 4.1 παρουσιάζει την επαναχρησιμοποίηση ομάδων δεδομένων για τους τρεις πίνακες, κατά την εκτέλεση του μετασχηματισμένου πολλαπλασιασμού πινάκων. Η επαναχρησιμοποίηση δεδομένων στο εσωτερικό κάθε tile (για οποιονδήποτε από τους τρεις πίνακες) πραγματοποιείται στους εσωτερικότερους βρόχους (i, k, j). Για παράδειγμα, η αναφορά $C[i,j]$ επαναχρησιμοποιεί δεδομένα κατά μήκος του βρόχου k, όπου μία ολόκληρη γραμμή ενός tile προσπελάζεται

επαναληπτικά. Η μέγιστη απόσταση επαναχρησιμοποίησης περικλείει τα δεδομένα που επαναχρησιμοποιούνται κατά μήκος των τριών εξωτερικότερων βρόχων (ii, kk, jj). Για παράδειγμα, η αναφορά $A[i,k]$ επαναχρησιμοποιεί δεδομένα κατά μήκος του βρόχου jj, ο οποίος περιέχει ένα ολόκληρο tile του A.



Σχήμα 4.1: Η επαναχρησιμοποίηση δεδομένων στους τρεις πίνακες του Πολλαπλασιασμού Πινάκων, όταν έχει εφαρμοστεί μετασχηματισμός Tiling

Το Παράρτημα **??** περιέχει τον πίνακα συμβόλων που θα χρησιμοποιήσουμε στο παρών κεφάλαιο.

4.3 Οι Αστοχίες της L1 Κρυφής Μνήμης Δεδομένων

Μελετάμε την περίπτωση όπου το πρώτο επίπεδο (L1) κρυφής μνήμης έχει οργάνωση ευθείας απεικόνισης. Λόγω των ενδεχόμενων πολλαπλών αστοχιών σύγκρουσης, η ακριβής καταγραφή της συμπεριφοράς μίας κρυφής μνήμης ευθείας απεικόνισης είναι δυσκολότερη. Στην περίπτωση αυτή, θα πρέπει να λάβουμε υπόψη μας τη σχετική θέση αποθήκευσης (alignment) των τριών πινάκων (που εμπλέκονται στον πολλαπλασιασμό πινάκων). Μία τυχαία επιλογή της σχετικής θέσης των πινάκων μπορεί να προκαλέσει υπερβολικά μεγάλο αριθμό αστοχιών σύγκρουσης. Η επιλογή της σχετικής θέσης των πινάκων στην L1 κρυφή μνήμη δεν επηρεάζει τον αριθμό των αστοχιών της L2 κρυφής μνήμης. Η L2 κρυφή μνήμη έχει αρκετά μεγαλύτερη χωρητικότητα από την L1 κρυφή μνήμη. Επομένως, προσθέτοντας μερικά πολλαπλάσια του C_{L1} στη σχετική θέση των πινάκων που επιλέχθηκε για να ικανοποιεί τις απαιτήσεις της L1 κρυφής μνήμης, μπορούμε να πετύχουμε την επιθυμητή σχετική θέση κατά την αντιστοίχιση των θέσεων μνήμης στην L2 κρυφή μνήμη, χωρίς να επηρεάσουμε την αντιστοίχιση στην L1 κρυφή μνήμη.

Οι παράγραφοι που ακολουθούν περιέχουν μία ανάλυση της συμπεριφοράς της L1 κρυφής μνήμης, για διάφορα μεγέθη πινάκων.

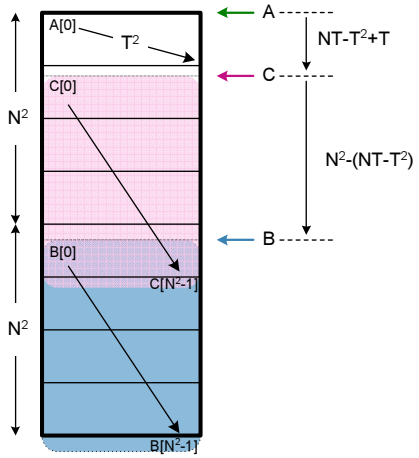
4.3.1 Περισσότεροι από ένας πίνακας χωρούν στην L1 κρυφή μνήμη: $N^2 < C_{L1}$

Για δεδομένο μέγεθος κρυφής μνήμης, εξετάζουμε την περίπτωση που το μέγεθος των πινάκων είναι αρκετά μικρό, ώστε να μπορούν να αποθηκευθούν τα επαναχρησιμοποιούμενα δεδομένα και

των τριών πινάκων A, B, C στην κρυφή μνήμη. Ο αριθμός όλων των αστοχιών μηδενίζεται, εκτός, ασφαλώς, από τις υποχρεωτικές αστοχίες (κατά την πρωταρχική αναφορά σε μία γραμμή δεδομένων). Η επιλογή των σχετικών θέσεων αποθήκευσης στη μνήμη και των τριών πινάκων και, κατά συνέπεια, οι θέσεις απεικόνισής τους στην κρυφή μνήμη, είναι εύκολη υπόθεση, όταν $3N^2 \leq C_{L1}$. Ωστόσο, όταν $2N^2 = C_{L1}$ οι σχετικές θέσεις απεικόνισης των πρώτων στοιχείων κάθε πίνακα θα πρέπει να επιλεγούν με προσοχή. Σε ότι αφορά τους πίνακες A, C, η ελάχιστη απόσταση απεικόνισης των στοιχείων $A[0]$ και $C[0]$, η οποία διασφαλίζει μηδενικές αστοχίες σύγκρουσης μεταξύ των δύο πινάκων στην L1 κρυφή μνήμη, είναι $NT - \left(T^2 - T\right)$ στοιχεία ($< C_{L1}$). Για τους πίνακες C και B, η ελάχιστη απόσταση απεικόνισης των στοιχείων $C[0]$ και $B[0]$ πρέπει να είναι $N^2 - \left(NT - T^2\right)$ στοιχεία, όπως φαίνεται στο σχήμα (σχήμα 4.2). Για να αποτραπούν οι αστοχίες σύγκρουσης μεταξύ των πινάκων A, B θα πρέπει ο κοινός χώρος αποθήκευσης των στοιχείων των δύο πινάκων να είναι $A \bigcap B \leq NT - T^2 + T - L_1$. Η συνθήκη αυτή ικανοποιείται από την προτεινόμενη διάταξη, αφού $A \bigcap B = T$ στοιχεία.

Σε κάθε περίπτωση, ο συνολικός αριθμός αστοχιών είναι ίσος με τον αριθμό των υποχρεωτικών αστοχιών:

$M_A = M_B = M_C = \frac{N^2}{L_1}$

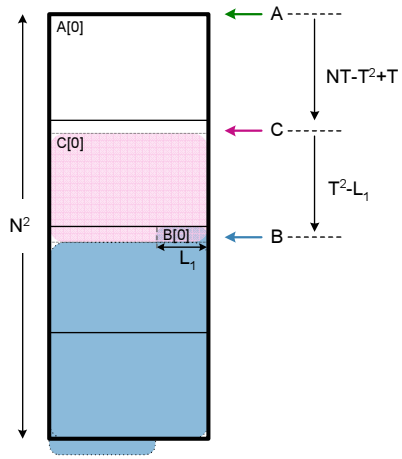


Σχήμα 4.2: Ευθυγράμμιση των πινάκων A, B, C, όταν $N^2 \leq C_{L1}$

4.3.2 Ένας μόνος πίνακας χωράει στην L1 κρυφή μνήμη: $C_{L1} = N^2$

Στην περίπτωση αυτή υπάρχει χώρος στην L1 κρυφή μνήμη μόνο για έναν πίνακα ($N^2 = C_{L1}$). Επομένως, προκειμένου να αποτρέψουμε έναν μεγάλο αριθμό από αστοχίες σύγκρουσης, επιλέγουμε και πάλι με προσοχή τις σχετικές θέσεις αποθήκευσης των τριών πινάκων στη μνήμη. Σε ότι αφορά τους πίνακες A, C, η ελάχιστη απόσταση απεικόνισης των στοιχείων $A[0]$ και $C[0]$, επιλέγεται όπως ακριβώς στην παράγραφο 4.3.1. Για τους πίνακες C και B, επιλέγουμε την απόσταση μεταξύ των στοιχείων $C[0]$ και $B[0]$ να είναι $T^2 - L_1$ στοιχεία. Η απόσταση αυτή δεν επιτρέπει στις αναφορές $B[k, j]$ και $C[i, j]$, για οποιαδήποτε επανάληψη, να προσπελάζουν ταυτόχρονα την

ίδια γραμμή (cache line) της κρυφής μνήμης. Σύμφωνα με αυτήν την ευθυγράμμιση των στοιχείων των τριών πινάκων, ο αριθμός των αστοχιών για κάθε μία από τις τρεις αναφορές πινάκων είναι:



Σχήμα 4.3: Ευθυγράμμιση των πινάκων A, B, C, όταν $C_{L1} = N^2$

$M_A = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τον πίνακα B
$M_B = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τους πίνακες A, C
$M_C = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τον πίνακα B

Στις διαδοχικές επαναλήψεις του βρόχου kk, οι αναφορές στους πίνακες A και C προσπελάζουν μόλις από μία γραμμή από tiles (x tiles $= NT$ στοιχεία). Αντίθετα, η αναφορά στον πίνακα B προσπελάζει συνολικά N^2 στοιχεία, δηλαδή ολόκληρη την L1 κρυφή μνήμη. Επομένως, συγκρούεται με τα στοιχεία των πινάκων A και C. Κάθε σύγκρουση με στοιχεία του πίνακα A διαγράφει τα T^2 στοιχεία του A που επαναχρησιμοποιούνται στο βρόχο jj. Οι συγκρούσεις με τον πίνακα C προκαλούν τη διαγραφή των $T \cdot N$ στοιχείων του C που επαναχρησιμοποιούνται στο βρόχο kk. Η επόμενη αναφορά στα στοιχεία που διαγράφηκαν από την L1 κρυφή μνήμη, θα αποτελέσει μία αστοχία κρυφής μνήμης. Οπότε θα πρέπει να ξαναμεταφερθούν τα στοιχεία στην L1 κρυφή μνήμη και να αποθηκευτούν σε αυτήν. Αυτή η επανα-αποθήκευση θα επαναληφθεί $x - 1$ φορές για τον πίνακα A (σε κάθε μία από τις x επαναλήψεις του βρόχου ii, εκτός από 1: όταν η σύγκρουση συμβαίνει κατά την τελευταία επαναχρησιμοποιήση των T^2 στοιχείων του A, όπου δε χρειάζεται επανα-αποθήκευση των στοιχείων του πίνακα, γιατί δεν πρόκειται να ξαναγίνει αναφορά σε αυτά, Για την προτεινόμενη ευθυγράμμιση, αυτή η εξαιρούμενη επανάληψη είναι η πρώτη επανάληψη του βρόχου ii). Στον πίνακα C, η επανα-αποθήκευση των στοιχείων που διαγράφονται γίνεται $x - 1$ φορές (η πρώτη επανάληψη του βρόχου ii εξαιρείται, όπως και στον πίνακα A).

Επιπρόσθετα, όποτε δύο tiles των πινάκων A και B απεικονίζονται στην ίδια περιοχή της κρυφής μνήμης (1 φορά σε κάθε επανάληψη του βρόχου ii), προσπελάζεται ταυτόχρονα η ίδια γραμμή της κρυφής μνήμης για L_1 διαδοχικές επαναλήψεις (από τις T^2 των βρόχων k, j). Η σύγκρουση αυτή επιφέρει L_1 αστοχίες σε κάθε έναν από τους πίνακες A, B. Κατά τη διάρκεια κάθε μίας i επανάληψης, μία ολόκληρη γραμμή ενός tile (T στοιχεία) του πίνακα B διαγράφεται από προσπελάσεις στον πίνακα A και αντίστροφα. Αυτή η γραμμή του tile θα πρέπει να ξανα-

φορτωθεί στην κρυφή μνήμη, δηλαδή έχουμε επιπλέον $\frac{T}{L}$ αστοχίες σε κάθε έναν από τους πίνακες A και B. Η παραπάνω ακολουθία συγκρούσεων επαναλαμβάνεται T φορές κατά την εκτέλεση των επαναλήψεων του βρόχου i, 1 φορά στους βρόχους jj και kk, και x φορές στο βρόχο ii.

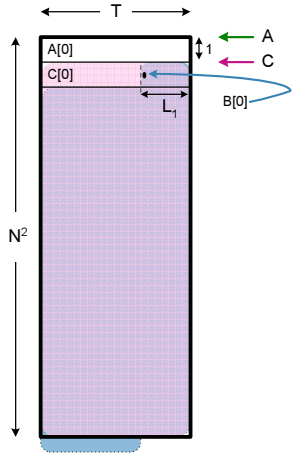
Από την άλλη πλευρά, δύο tiles των πινάκων C και B δεν απεικονίζονται ποτέ ταυτόχρονα στην ίδια περιοχή της L1 κρυφής μνήμης. Ωστόσο, κατά την εκτέλεση των επαναλήψεων του βρόχου kk, και ενώ η αναφορά στον πίνακα C προσπελάζει στοιχεία μίας σειράς από tiles (NT στοιχεία), διαγράφει στοιχεία του πίνακα B. Κατά συνέπεια προκύπτουν $2\frac{NT}{L_1}$ αστοχίες (NT στοιχεία διαγράφονται από κάθε προηγούμενη επανάληψη του βρόχου ii και NT στοιχεία εξαιτίας της παρούσας επανάληψης). Η επαναληπτική φόρτωση λαμβάνει χώρα $x-1$ φορές (κατά την εκτέλεση όλων των x επαναλήψεων του βρόχου ii εκτός από την πρώτη).

$M_A = \frac{N^2}{L_1} + \frac{T^2}{L_1} \cdot (x-1) + \left(L_1 + \frac{T}{L_1}\right) \cdot T \cdot x = \frac{N^2}{L_1} + N \cdot \left(L_1 + \frac{T}{L_1}\right) - \frac{T^2}{L_1}$

$M_B = \frac{N^2}{L_1} + \left(L_1 + \frac{T}{L_1}\right) \cdot T \cdot x + \frac{2NT}{L_1} \cdot (x-1) = \frac{N^2}{L_1} + N \cdot \left(L_1 + \frac{T}{L_1}\right) + \frac{2N^2}{L_1} - \frac{2NT}{L_1}$

$M_C = \frac{N^2}{L_1} + \frac{NT}{L_1} \cdot (x-1) = \frac{N^2}{L_1} + \frac{NT}{L_1} \cdot \left(\frac{N}{T} - 1\right)$

Αν $T = N$, η απόσταση απεικόνισης των στοιχείων $C[0]$ και $A[0]$ ορίζεται να είναι $T = N$ στοιχεία, όσα και επαναχρησιμοποιούμενα στοιχεία κατά την επαναληπτική εκτέλεση του βρόχου k.



Σχήμα 4.4: Ευθυγράμμιση των πινάκων A, B, C, όταν $C_{L1} = N^2$, με $T = N$

Με παρόμοιο τρόπο προκύπτει ότι:

$M_A = \frac{N^2}{L_1} + L_1 \cdot (N-1)$

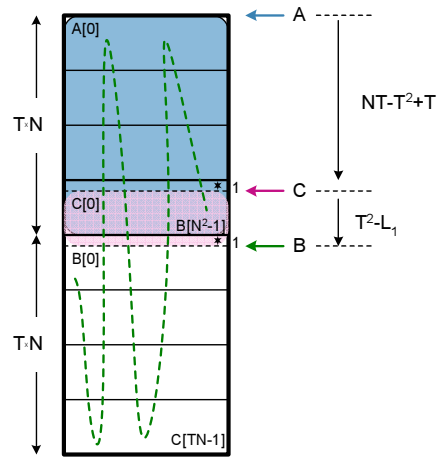
$M_B = \frac{N^2}{L_1} + L_1 \cdot (N-1) + \frac{N}{L_1} \cdot (N-1) + \frac{2N}{L_1} \cdot (N-1) = \frac{N^2}{L_1} + L_1 \cdot (N-1) + \frac{3N}{L_1} \cdot (N-1)$

$M_C = \frac{N^2}{L_1} + \frac{N}{L_1} \cdot (N-1)$

4.3.3 Μία σειρά από tiles χωράει στην κρυφή μνήμη: $N^2 > C_{L1}$, $T \cdot N < C_{L1}$

Στην περίπτωση αυτή, δύο τουλάχιστον σειρές από tiles χωρούν στην L1 κρυφή μνήμη.

Στους πίνακες A, C μπορεί να αξιοποιηθεί η επαναχρησιμοποίηση δεδομένων τόσο στο εσωτερικό κάθε tile όσο και ολόκληρων tiles, γιατί η L1 κρυφή μνήμη μπορεί να χωρέσει T^2 και $T \cdot N$



Σχήμα 4.5: Ευθυγράμμιση των πινάκων A, B, C, όταν $N^2 > C_{L1}$ και $T \cdot N < C_{L1}$

στοιχεία αντίστοιχα. Από την άλλη πλευρά, στον πίνακα B μπορεί να αξιοποιηθεί μόνο η επαναχρησιμοποίηση στο εσωτερικό των tiles, καθώς για την αξιοποίηση ολόκληρων tiles απαιτείται χωρητικότητα N^2 στοιχείων ($\geq C_{L1}$). Οι αποστάσεις απεικόνισης των στοιχείων $A[0]$, $C[0]$ και $C[0]$, $B[0]$ θα πρέπει να είναι όπως στην παράγραφο 4.3.2 για $T \neq N$ (σχήμα 4.5).

Ο αριθμός των αστοχιών είναι:

$M_A = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τον B

$M_B = \frac{N^3}{T \cdot L_1}$ + αστοχίες σύγκρουσης με τον A

$M_C = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τον B

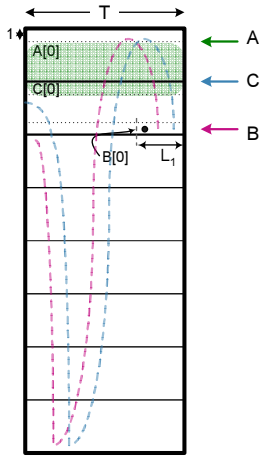
Κατά τις επαναληπτικές εκτελέσεις του βρόχου kk, και ενώ οι αναφορές στους πίνακες A και C προσπελάζουν στοιχεία μίας μόλις σειράς από tiles (x tiles $= NT$ στοιχεία ο κάθε πίνακας), η αναφορά στον πίνακα B προσπελάζει όλες τις γραμμές της L1 κρυφής μνήμης $\frac{N^2}{C_{L1}}$ φορές (προσπελάζονται N^2 στοιχεία του πίνακα B), οπότε συγκρούεται με τους πίνακες A και C. Κάθε σύγκρουση διαγράφει T^2 στοιχεία σε κάθε έναν από τους πίνακες A και C, και επομένως πρέπει να ξανα-φορτωθούν στην κρυφή μνήμη $\frac{N^2}{C_{L1}} \cdot x$ φορές. Ο ακριβής αριθμός είναι $\frac{N^2}{C_{L1}} \cdot \left(x - \frac{N^2}{C_{L1}}\right) + \left(\frac{N^2}{C_{L1}} - 1\right) \cdot \frac{N^2}{C_{L1}} = \frac{N^2}{C_{L1}} \cdot (x - 1)$: σε $\frac{N^2}{C_{L1}}$ επαναλήψεις (από σύνολο x επαναλήψεων του βρόχου kk) μία από τις συγκρούσεις μεταξύ των A και C συμβαίνει κατά τη διάρκεια της πρώτης χρησιμοποίησης ενός από τα tiles του A. Επομένως, η συγκεκριμένη αστοχία έχει ήδη υπολογιστεί στις υποχρεωτικές αστοχίες. Οπότε η επανα-φόρτωση χρειάζεται να γίνει μόνο $\left(\frac{N^2}{C_{L1}} - 1\right)$ φορές για τις περιπτώσεις αυτές.

Επιπρόσθετα, οποτεδήποτε δύο των πινάκων A και B απεικονίζονται στην ίδια περιοχή της κρυφής μνήμης ($\frac{N^2}{C_{L1}}$ φορές κατά τις επαναληπτικές εκτελέσεις του βρόχου kk), προκύπτουν $\left(L_1 + \frac{T}{L}\right)$ αστοχίες και στους δύο πίνακες A και B, όπως περιγράφτηκε στην παράγραφο 4.3.2. Η παραπάνω ακολουθία συγκρούσεων επαναλαμβάνεται T φορές κατά την εκτέλεση των επαναλήψεων του βρόχου i, 1 φορά στο βρόχο jj, $\frac{N^2}{C_{L1}}$ φορές στο βρόχο kk και x φορές στο βρόχο ii. Παρόμοιο φαινόμενο δεν εμφανίζεται στις συγκρούσεις μεταξύ των πινάκων A και C, χάρη στη επιλεγόμενη απόσταση απεικόνισης των πινάκων:

$$M_A = \frac{N^2}{L_1} + \frac{T^2}{L_1} \cdot \frac{N^2}{C_{L1}} \cdot (x-1) + \left(L_1 + \frac{T}{L}\right) \cdot T \cdot \frac{N^2}{C_{L1}} x = \frac{N^2}{L_1} + \frac{N^3}{C_{L1}} \cdot \left(L_1 + \frac{2T}{L_1}\right) - \frac{N^2 T^2}{L_1 C_{L1}}$$
$$M_B = \frac{N^3}{TL_1} + \left(L_1 + \frac{T}{L}\right) \cdot T \cdot \frac{N^2}{C_{L1}} x = \frac{N^3}{TL_1} + \frac{N^3}{C_{L1}} \cdot \left(L_1 + \frac{T}{L_1}\right)$$
$$M_C = \frac{N^2}{L_1} + \frac{T^2}{L_1} \cdot \frac{N^2}{C_{L1}} \cdot (x-1) = \frac{N^2}{L_1} + \frac{N^2 T^2}{L_1 C_{L1}} \cdot \left(\frac{N}{T} - 1\right)$$

4.3.4 Ένα tile για κάθε έναν από τους τρεις πίνακες χωράνε στην κρυφή μνήμη: $3T^2 < C_{L1} \leq T \cdot N$ $(N^2 > C_{L1})$

Στην περίπτωση αυτή, τρία ολόκληρα tiles χωράνε στην κρυφή μνήμη, ένα για κάθε έναν από τους τρεις πίνακες. Η απόσταση των $A[0]$ και $C[0]$ επιλέγεται να είναι $T^2 - T$ στοιχεία, έτσι ώστε η επαναχρησιμοποίηση στο εσωτερικό των tiles του πίνακα C δεν θα επιφέρει οποιαδήποτε σύγκρουση με στοιχεία του A (σχήμα 4.6). Η απόσταση απεικόνισης μεταξύ των $C[0]$, $B[0]$ θα πρέπει να είναι τουλάχιστον $T^2 - L_1$ στοιχεία, όπως και στην παράγραφο 4.3.2.



Σχήμα 4.6: Ευθυγράμμιση των πινάκων A, B, C, όταν $3T^2 < C_{L1} \leq T \cdot N$

Για τους πίνακες B και C η επαναχρησιμοποίηση δεδομένων κατά την επαναληπτική εκτέλεση των βρόχων ii και kk αντίστοιχα δεν είναι δυνατόν να αξιοποιηθεί, καθώς δεν υπάρχει αρκετή χωρητικότητα στη L1 κρυφή μνήμη για την αποθήκευση N^2 και $T \cdot N$ στοιχείων αντίστοιχα. Ακόμα και όταν $C_{L1} = T \cdot N$, όπου τα επαναχρησιμοποιούμε στοιχεία του πίνακα C εντός του βρόχου kk χωράνε στην L1 κρυφή μνήμη, οι προσπελάσεις στοιχείων του πίνακα B απομακρύνουν τα στοιχεία του πίνακα C και αποτρέπουν την αξιοποίηση της επαναχρησιμοποίησής τους. Η σύγκρουση μεταξύ της απεικόνισης των στοιχείων των πινάκων B και C δεν επιφέρει επιπρόσθετες αστοχίες πέρα από το ότι προκαλεί αλληλο-διαγραφή από την κρυφή μνήμη. Ο αριθμός των αστοχιών είναι:

$M_B = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον A

$M_C = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον A

Από την άλλη πλευρά, αξιοποιείται η επαναχρησιμοποίηση δεδομένων του πίνακα A στο βρόχο jj (η επαναχρησιμοποίηση αυτή περιλαμβάνει μόλις T^2 στοιχεία):

$M_A = \frac{N^2}{L_1}$ + αστοχίες σύγκρουσης με τους B, C

Κατά τη διαδοχική εκτέλεση των επαναλήψεων του βρόχου jj, και ενώ η αναφορά στον πίνακα A προσπελάζει στοιχεία που ανήκουν στο ίδιο tile, οι αναφορές στους B και C προσπελάζουν όλες τις γραμμές της L1 κρυφής μνήμης (προσπελάζονται x tiles για κάθε έναν από τους πίνακες B,C $= NT$ στοιχεία). Επομένως, και οι δύο αυτοί πίνακες συγκρούονται με στοιχεία του πίνακα A $\frac{NT}{C_{L1}}$ φορές. Σε κάθε σύγκρουση T^2 στοιχεία του πίνακα A απομακρύνονται από την κρυφή μνήμη, και κατά συνέπεια πρέπει να ξανα-φορτωθούν $2 \cdot \frac{NT}{C_{L1}} \cdot x^2$ φορές ($\frac{NT}{C_{L1}}$ φορές λόγω των συγκρούσεων με τον πίνακα B και $\frac{NT}{C_{L1}}$ φορές λόγω των συγκρούσεων με τον πίνακα C, πολλαπλασιασμένες με τον αριθμό x^2 των επαναλήψεων των βρόχων ii και kk). Ο ακριβής αριθμός των επαναληπτικών φορτώσεων των στοιχείων του πίνακα A στην κρυφή μνήμη είναι $2\frac{NT}{C_{L1}} \cdot \left(x - \frac{NT}{C_{L1}}\right) \cdot x + \left(2\frac{NT}{C_{L1}} - 1\right) \cdot \frac{NT}{C_{L1}} \cdot x = \frac{NT}{C_{L1}} \cdot (2x-1) \cdot x$: σε $\frac{NT}{C_{L1}}$ επαναλήψεις (από σύνολο x επαναλήψεων) του βρόχου kk μία από τις συγκρούσεις μεταξύ των A και C συμβαίνει κατά τη διάρκεια της τελευταίας επαναχρησιμοποίησης ενός από τα tiles του A, και επομένως δεν υπάρχει λόγος να ξανα-φορτωθούν τα στοιχεία του στην κρυφή μνήμη.

Επιπρόσθετα, οποτεδήποτε δύο tiles από τους A και B απεικονίζονται στην ίδια περιοχή της κρυφής μνήμης ($\frac{NT}{C_{L1}}$ φορές κατά την επαναληπτική εκτέλεση του βρόχου jj), προκύπτουν $\left(L_1 + \frac{T}{L}\right)$ αστοχίες και στους δύο πίνακες A και B, όπως περιγράφτηκε στην παράγραφο 4.3.2. Ο παραπάνω αριθμός αστοχιών επαναλαμβάνεται T φορές στο βρόχο i, $\frac{NT}{C_{L1}}$ φορές στο βρόχο jj και x^2 φορές στους βρόχους kk και ii. Παρόμοιο φαινόμενο δεν παρατηρείται στις συγκρούσεις των πινάκων A και C, καθώς οι 2 γραμμές της κρυφής μνήμης που προσπελάζονται από τις αναφορές στους πίνακες αυτούς έχουν απόσταση τουλάχιστον T στοιχεία, λόγω της επιλεγμένης ευθυγράμμισης. Τελικά:

$$M_A = \frac{N^2}{L_1} + \frac{T^2}{L_1}\frac{NT}{C_{L1}} \cdot (2x-1) \cdot x + \left(L_1 + \frac{T}{L}\right) \cdot T \cdot \frac{NT}{C_{L1}} \cdot x^2 = \frac{N^2}{L_1} \cdot \left(1 - \frac{T^2}{C_{L1}}\right) + \frac{N^3}{C_{L1}} \cdot \left(L_1 + \frac{3T}{L_1}\right)$$
$$M_B = \frac{N^3}{TL_1} + \left(L_1 + \frac{T}{L}\right) \cdot T \cdot \frac{NT}{C_{L1}} \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3}{C_{L1}}\left(L_1 + \frac{T}{L_1}\right)$$
$$M_C = \frac{N^3}{TL_1}$$

4.3.5 Λιγότερα από τρία ολόκληρα tiles χωράνε στην κρυφή μνήμη: $N^2 > C_{L1}$, $T^2 \leq C_{L1} < 3T^2$

Επιλέγουμε την απόσταση των στοιχείων $A[0]$ και $C[0]$ να είναι Τ στοιχεία, δηλαδή, μία ολόκληρη γραμμή δεδομένων ενός tile. Για τους πίνακες B και C, προκειμένου να επιτύχουμε τον ελάχιστο δυνατό αριθμό συγκρούσεων στην κρυφή μνήμη, η απόσταση αποθήκευσης μεταξύ των πρώτων στοιχείων κάθε ενός από τους πίνακες θα πρέπει να είναι τουλάχιστον $T^2 - L_1$ στοιχεία (σχήμα 4.7).

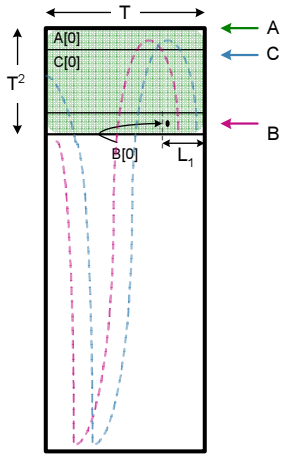
Μόνο η επαναχρησιμοποίηση στο εσωτερικό του tile μπορεί να αξιοποιηθεί στους τρεις πίνακες κατά την εκτέλεση των επαναλήψεων των τριών εσωτερικότερων βρόχων. Επομένως:

$M_A = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον B

$M_B = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τους A, C

$M_C = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον B

Οι συγκρούσεις μεταξύ των αναφορών στους πίνακες A και B συμβαίνουν κατά παρόμοιο



Σχήμα 4.7: Ευθυγράμμιση των πινάκων A, B, C, όταν $N^2 > C_{L1}$, $T^2 \leq C_{L1} < 3T^2$

τρόπο με την προηγούμενη περίπτωση. Διαφέρουν μόνο ως προς τη συχνότητα των συγκρούσεων: δύο tiles των πινάκων A και B απεικονίζονται στις ίδιες θέσεις της κρυφής μνήμης $\frac{T^2}{C_{L1}} \cdot x$ φορές κατά την εκτέλεση των διαδοχικών επαναλήψεων του βρόχου jj ($\frac{T^2}{C_{L1}} \leq 1$).

Σε ότι αφορά τις συγκρούσεις μεταξύ των πινάκων B και C, κατά την εκτέλεση των διαδοχικών επαναλήψεων του βρόχου i, μία ολόκληρη γραμμή ενός tile (T στοιχεία) του πίνακα B διαγράφονται από την κρυφή μνήμη λόγω αποθήκευσης στοιχείων του C, και αντίστροφα. Αυτή η γραμμή του tile θα πρέπει να επανα-μεταφερθεί στην κρυφή μνήμη, το οποίο σημαίνει $\frac{T}{L}$ επιπρόσθετες αστοχίες τόσο στον πίνακα B όσο και στον C. Η παραπάνω ακολουθία συγκρούσεων επαναλαμβάνεται T φορές κατά την εκτέλεση όλων των επαναλήψεων του βρόχου i, $\frac{T^2}{C_{L1}} \cdot x$ φορές κατά τις διαδοχικές επαναλήψεις του βρόχου jj και x^2 φορές κατά τις διαδοχικές επαναλήψεις των βρόχων kk και ii.

$$M_A = \frac{N^3}{TL_1} + L_1 \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3 L_1}{C_{L1}}$$
$$M_B = \frac{N^3}{TL_1} + \left(L_1 + \frac{T}{L_1}\right) \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 + \frac{T}{L_1} \cdot T \cdot \left(\frac{T^2}{C_{L1}} x\right) \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3}{C_{L1}} \left(L_1 + \frac{2T}{L_1}\right)$$
$$M_C = \frac{N^3}{TL_1} + \frac{T}{L_1} \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3 T}{C_{L1} L_1}$$

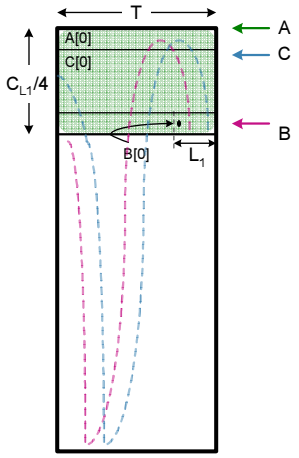
4.3.6 Ένα ολόκληρο tile δεν χωράει στην κρυφή μνήμη: $N^2 > C_{L1}$, $T^2 > C_{L1} > T$

Προκειμένου να ελαχιστοποιήσουμε τον αριθμό των συγκρούσεων μεταξύ των αναφορών σε στοιχεία διαφορετικών πινάκων, επιλέγουμε $C[0] - A[0] = T$ στοιχεία και $B[0] - C[0] = \frac{C_{L1}}{4} - L_1$ στοιχεία (σχήμα 4.8).

Όπως και στην προηγούμενη περίπτωση, δεν είναι δυνατόν να αξιοποιηθεί η επαναχρησιμοποίηση ολόκληρων tiles. Επιπρόσθετα, στον πίνακα B, η επαναχρησιμοποίηση στο εσωτερικό των tiles (κατά την εκτέλεση των διαδοχικών επαναλήψεων του βρόχου i) δεν μπορεί να αξιοποιηθεί. Κατά συνέπεια, ο συνολικός αριθμός των αστοχιών για κάθε έναν από τους πίνακες είναι:

$M_A = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον B

$M_B = \frac{N^3}{L_1}$ + αστοχίες σύγκρουσης με τον A



Σχήμα 4.8: Ευθυγράμμιση των πινάκων A, B, C, όταν $N^2 > C_{L1}$, $T^2 > C_{L1} > T$

$M_C = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τον B

Υπολογίζουμε τον αριθμό των αστοχιών σύγκρουσης παρόμοια με την προηγούμενη περίπτωση. Στον πίνακα B δεν αξιοποιείται καθόλου η επαναχρησιμοποίηση δεδομένων στο εσωτερικό των tiles του. Επομένως οι συγκρούσεις των αναφορών στον πίνακα B με τις αναφορές στους πίνακες A και C, επιφέρουν τη διαγραφή από την κρυφή μνήμη ολόκληρων γραμμών του εκάστοτε επαναχρησιμοποιούμενου tile (T στοιχεία), μόνο στους πίνακες A και C.

$M_A = \frac{N^3}{TL_1} + L_1 \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3 L_1}{C_{L1}}$
$M_B = \frac{N^3}{L_1} + L_1 \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 = \frac{N^3}{L_1} + \frac{N^3 L_1}{C_{L1}}$
$M_C = \frac{N^3}{TL_1} + \frac{T}{L_1} \cdot T \cdot \frac{T^2 x}{C_{L1}} \cdot x^2 = \frac{N^3}{TL_1} + \frac{N^3 T}{C_{L1} L_1}$

4.3.7 Μία γραμμή ενός tile υπερβαίνει σε μέγεθος τη χωρητικότητα της κρυφής μνήμης: $N^2 > C_{L1}$, $T \geq C_{L1}$

Επιπρόσθετα με την προηγούμενη περίπτωση, στον πίνακα C, η επαναχρησιμοποίηση στο εσωτερικό των tiles (κατά την επαναληπτική εκτέλεση του βρόχου k) δεν μπορεί να αξιοποιηθεί. Κατά συνέπεια, ο συνολικός αριθμός αστοχιών για κάθε πίνακα είναι:

$M_A = \frac{N^3}{TL_1}$ + αστοχίες σύγκρουσης με τους B, C

array B: $M_B = \frac{N^3}{L_1}$ + αστοχίες σύγκρουσης με τον A

array C: $M_C = \frac{N^3}{L_1}$ + αστοχίες σύγκρουσης με τον A

Επιλέγοντας $B[0] - C[0] \geq L_1$ στοιχεία, δεν υπάρχει οποιαδήποτε αστοχία σύγκρουσης μεταξύ των δύο πινάκων. Ενώ η αναφορά $A[i,k]$ προσπελάζει συνεχώς το ίδιο στοιχείο κατά την εκτέλεση των διαδοχικών επαναλήψεων του βρόχου j, οι αναφορές στους πίνακες B και C διασχίζουν ολόκληρη την κρυφή μνήμη, καθώς προσπελάζουν ακολουθιακά T στοιχεία. Κάθε ένα από αυτά συγκρούεται με το στοιχείο του πίνακα A $\frac{T}{C_{L1}}$ φορές, και προκύπτουν L_1 αστοχίες. Η διαδικασία αυτή επαναλαμβάνεται $T^2 \cdot x^3$ φορές κατά την εκτέλεση των βρόχων k, i, jj, kk και ii. Κατά συνέπεια, ο αριθμός των αστοχιών σύγκρουσης (για κάθε έναν από τους συγκρουόμενους πίνακες)

είναι $\frac{T}{C_{L1}} \cdot L_1 \cdot T^2x^3$:

$$M_A = \frac{N^3}{TL_1} + 2\frac{T}{C_{L1}} \cdot L_1 \cdot T^2x^3 = \frac{N^3}{TL_1} + \frac{2N^3L_1}{C_{L1}}$$
$$M_B = \frac{N^3}{L_1} + \frac{T}{C_{L1}} \cdot L_1 \cdot T^2x^3 = \frac{N^3}{L_1} + \frac{N^3L_1}{C_{L1}}$$
$$M_C = \frac{N^3}{L_1} + \frac{T}{C_{L1}} \cdot L_1 \cdot T^2x^3 = \frac{N^3}{L_1} + \frac{N^3L_1}{C_{L1}}$$

4.3.8 Σύνοψη των αστοχιών της L1 κρυφής μνήμης δεδομένων

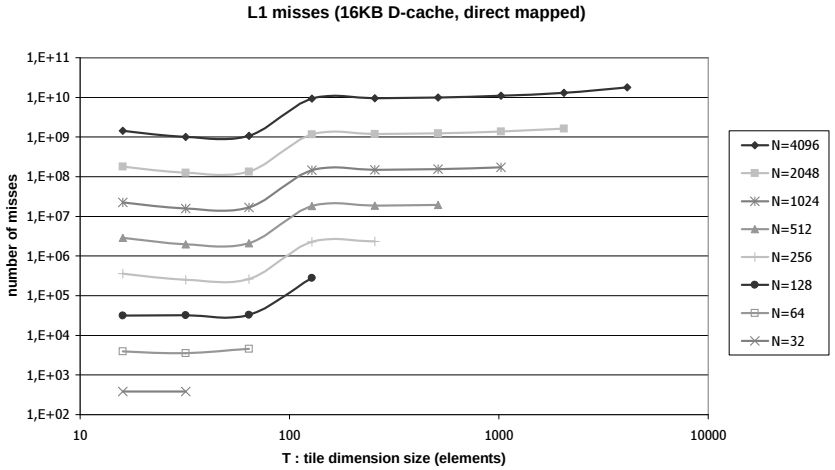
Ο πίνακας 4.1 συνοψίζει τον συνολικό αριθμό αστοχιών M_1 $(= M_A + M_B + M_C)$ της L1 κρυφής μνήμης δεδομένων, για διάφορα μεγέθη προβλημάτων.

requirements	$\mathbf{M_1}$
$N^2 < C_{L1}$	$\frac{3N^2}{L_1}$
$N^2 = C_{L1},\ T \neq N$	$\frac{6N^2}{L_1} + 2NL_1 - \frac{T^2+NT}{L_1}$
$N^2 = C_{L1},\ T = N$	$\frac{7N^2}{L_1} + 2NL_1 - \left(2L_1 + \frac{4N}{L_1}\right)$
$N^2 > C_{L1},\ T \cdot N < C_{L1}$	$\frac{2N^2}{L_1}\left(1 - \frac{T^2}{C_{L1}}\right) + \frac{N^3}{TL_1} + + \frac{N^3}{C_{L1}}\left(2L_1 + \frac{4T}{L_1}\right)$
$3T^2 < C_{L1} \leq T \cdot N$	$\frac{N^2}{L_1}\left(1 - \frac{T^2}{C_{L1}}\right) + \frac{2N^3}{TL_1} + + \frac{N^3}{C_{L1}}\left(2L_1 + \frac{4T}{L_1}\right)$
$T^2 \leq C_{L1} < 3T^2$	$\frac{3N^3}{TL_1} + \frac{N^3}{C_{L1}}\left(2L_1 + \frac{3T}{L_1}\right)$
$T^2 > C_{L1} > T$	$\frac{N^3}{L_1} + \frac{N^3}{TL_1}\left(2 + \frac{T^2}{C_{L1}}\right) + \frac{2N^3L_1}{C_{L1}}$
$T \geq C_{L1}$	$\frac{N^3}{TL_1} + \frac{2N^3}{L_1} + \frac{4N^3L_1}{C_{L1}}$

Πίνακας 4.1: Οι εξισώσεις υπολογισμού των αστοχιών της L1 κρυφής μνήμης δεδομένων

Η γραφική αναπαράσταση των εξισώσεων αυτών φαίνεται στο σχήμα 4.9, για τα αρχιτεκτονικά χαρακτηριστικά του UltraSPARC II.

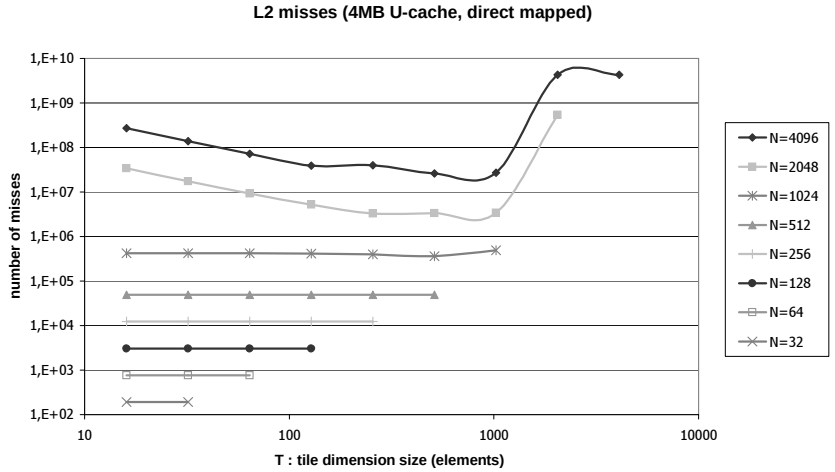
Ο ελάχιστος αριθμός αστοχιών, για όλα τα μεγέθη πινάκων, λαμβάνει χώρα όταν $T = 32$, όπου τρία ολόκληρα tiles χωράνε στην κρυφή μνήμη (ένα για κάθε έναν από τους τρεις πίνακες) Όταν $T = 64$ $(T = \sqrt{C_{L1}})$, ένα μόνο tile χωράει στην κρυφή μνήμη, αλλά αξιοποιείται ολόκληρη η L1 κρυφή μνήμη. Στο σημείο αυτό, ο αριθμός των αστοχιών μεγαλώνει ελαφρώς, αλλά συνεχίζει να βρίσκεται πολύ κοντά στην ελάχιστη τιμή του.



Σχήμα 4.9: Ο αριθμός των αστοχιών της L1 κρυφής μνήμης για διαφορετικά μεγέθη πινάκων και tiles, όταν εφαρμόζεται η ευθυγράμμιση που περιγράφτηκε στις αντίστοιχες παραγράφους

4.4 Οι Αστοχίες της L2 Κρυφής Μνήμης

Στην αρχιτεκτονική του UltraSPARC II, το επίπεδο της L2 κρυφής μνήμης είναι ευθείας απεικόνισης, όπως και η L1 κρυφή μνήμη. Κατά συνέπεια, παραλείπουμε την αναλυτική περιγραφή του τρόπου υπολογισμού του ακριβούς αριθμού αστοχιών, και παρουσιάζουμε μόνο το αντίστοιχο διάγραμμα (σχήμα 4.10). Σημειώνουμε ότι η L2 κρυφή μνήμη είναι ενοποιημένη (για εντολές και δεδομένα). Ωστόσο, για σχετικά μεγάλες κρυφές μνήμες, ο αριθμός των αστοχιών αυξάνει ανεπαίσθητα (λιγότερο από 1%) σε σύγκριση με μία κρυφή μνήμη ίσου μεγέθους που αφιερώνεται αποκλειστικά για δεδομένα.



Σχήμα 4.10: Ο αριθμός των αστοχιών της L2 κρυφής μνήμης για διάφορα μεγέθη πινάκων και tiles

Ο αριθμός των αστοχιών της L2 κρυφής μνήμης, για όλα τα μεγέθη πινάκων, λαμβάνει ελάχιστη τιμή όταν $T = 512$, και αυξάνει ελαφρά όταν $T = 1024$, όπου χρησιμοποιείται ολόκληρη η L2

κρυφή μνήμη και μόλις ένα ολόκληρο tile χωράει σε αυτήν.

4.5 Οι Αστοχίες στο TLB Δεδομένων

Αυτό το επίπεδο μνήμης είναι συνήθως πλήρους συσχέτισης. Επομένως, δε χρειάζεται να ασχοληθούμε με τη σχετική θέση των δεδομένων στη μνήμη.

4.5.1 Οι διευθύνσεις 3 γραμμών από tiles χωρούν στο TLB: $N^2 \geq E \cdot P$, $3T \cdot N < E \cdot P$

Στην περίπτωση αυτή οι διευθύνσεις τριών γραμμών από tiles χωράνε στο TLB.

Για τους πίνακες A και C, μπορεί να αξιοποιηθεί η επαναχρησιμοποίηση κατά μήκος των βρόχων jj και kk, αντίστοιχα, αφού υπάρχει αρκετός διαθέσιμος χώρος για την αποθήκευση $\frac{T^2}{P}$ και $\frac{T \cdot N}{P}$ διευθύνσεων, αντίστοιχα. Κατά συνέπεια, ο συνολικός αριθμός αστοχιών για τους δύο πίνακες ισούται με τον υποχρεωτικό αριθμό αστοχιών (αστοχίες πρώτης αναφοράς), δηλαδή ισούται με $\frac{N^2}{P}$, για κάθε έναν από τους δύο πίνακες.

Από την άλλη πλευρά, η επαναχρησιμοποίηση των στοιχείων του πίνακα B δεν είναι δυνατόν να αξιοποιηθεί στο μέγιστο, καθώς ο αναγκαίος αριθμός εγγραφών για αυτό το σκοπό (για την αποθήκευση $\frac{N^2}{P}$ διευθύνσεων) ξεπερνάει την χωρητικότητα του TLB. Κατά την επαναληπτική εκτέλεση του βρόχου ii, οι επαναχρησιμοποιούμενες διευθύνσεις δεν βρίσκονται πλέον στο TLB και επομένως πρέπει να επαναφορτωθούν από τη μνήμη. Ωστόσο, δεν υπάρχει οποιαδήποτε αστοχία σύγκρουσης με τους πίνακες A και C, καθώς σύμφωνα με την πολιτική LRU (least recently used) οι λιγότερο πρόσφατα χρησιμοποιούμενες σελίδες είναι εκείνες του πίνακα B.

$M_A = \frac{N^2}{P}$, $M_B = \frac{N^2}{P} \cdot x = \frac{N^3}{T \cdot P}$, $M_C = \frac{N^2}{P}$

4.5.2 Οι διευθύνσεις 2 τουλάχιστον γραμμών από tiles χωρούν στο TLB: $N^2 > E \cdot P$, $T \cdot N < E \cdot P < 3T \cdot N$

Για τον πίνακα C, η επαναχρησιμοποίηση κατά την επαναληπτική εκτέλεση του βρόχου kk δεν είναι δυνατόν να αξιοποιηθεί, καθώς δεν υπάρχει αρκετός διαθέσιμος χώρος στο TLB για την αποθήκευση $\frac{T \cdot N}{P}$ διευθύνσεων.

$M_A = \frac{N^2}{P}$, $M_B = \frac{N^3}{T \cdot P}$, $M_C = x \times \frac{N^2}{P} = \frac{N^3}{T \cdot P}$

4.5.3 Οι διευθύνσεις τουλάχιστον 2 ολόκληρων tiles χωρούν στο TLB: $N^2 > E \cdot P$, $T^2 < E \cdot P < 3T^2$

Προκειμένου να αξιοποιηθεί η επαναχρησιμοποίηση δεδομένων του πίνακα A ($\frac{T^2}{P}$ διευθύνσεις) κατά τις επαναληπτικές εκτελέσεις του βρόχου jj, υπάρχει αρκετός χώρος στο TLB. Ωστόσο, οι

εγγραφές του TLB γεμίσουν λόγω των πολλών διαφορετικών σελίδων που προσπελάζονται από τις προσπελάσεις του πίνακα B. Οι διευθύνσεις που αντιστοιχούν σε στοιχεία του πίνακα A δεν είναι οι πιο πρόσφατα χρησιμοποιούμενες, με αποτέλεσμα να διαγράφονται από το TLB.

$M_A = \frac{N^3}{T \cdot P}$, $M_B = \frac{N^3}{T \cdot P}$, $M_C = \frac{N^3}{T \cdot P}$

4.5.4 Οι διευθύνσεις 1 ολόκληρου tile δεν χωρούν στο TLB: $N^2 > E \cdot P$, $T^2 > E \cdot P > T$

Η επαναχρησιμοποίηση δεδομένων του πίνακα B δεν είναι δυνατόν να αξιοποιηθεί κατά την επαναληπτική εκτέλεση του βρόχου i: η χωρητικότητα του TLB δεν επαρκεί για την αποθήκευση $\frac{T^2}{P}$ διευθύνσεων:

$M_A = \frac{N^3}{T \cdot P}$, $M_B = \frac{N^3}{P}$, $M_C = \frac{N^3}{T \cdot P}$

4.5.5 Οι διευθύνσεις 1 ολόκληρης γραμμής του tile δεν χωρούν στο TLB: $N^2 > E \cdot P$, $T \geq E \cdot P$

Η επαναχρησιμοποίηση δεδομένων του πίνακα C δεν είναι δυνατόν να αξιοποιηθεί κατά την εκτέλεση των διαδοχικών επαναλήψεων του βρόχου k: η χωρητικότητα του TLB δεν επαρκεί για την αποθήκευση $\frac{T}{P}$ διευθύνσεων:

$M_A = \frac{N^3}{T \cdot P}$, $M_B = \frac{N^3}{P}$, $M_C = \frac{N^3}{P}$

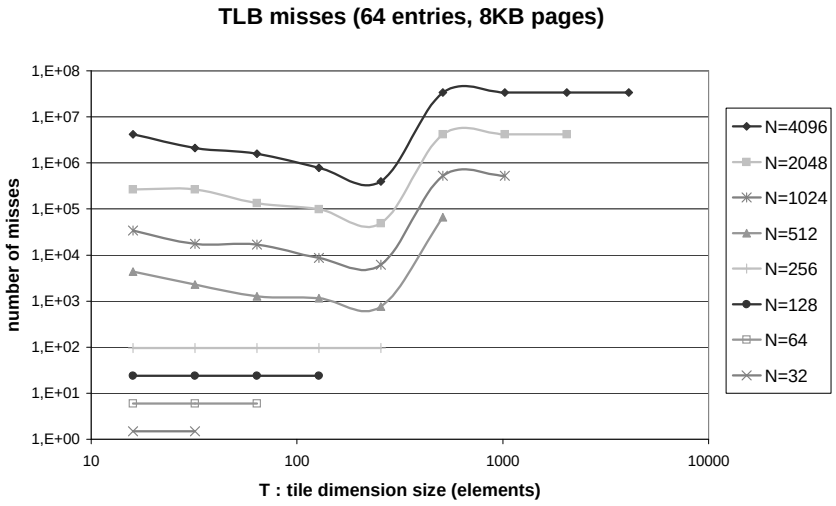
4.5.6 Σύνοψη των αστοχιών του TLB

Ο Πίνακας 4.2 συνοψίζει τον συνολικό αριθμό των αστοχιών δεδομένων του TLB M_{TLB} για τα διάφορα μεγέθη προβλημάτων.

μέγεθος πινάκων tiles	M_{TLB}
$N^2 < E \cdot P$	$3\frac{N^2}{P}$
$3T \cdot N \leq E \cdot P$	$2\frac{N^2}{P} + \frac{N^3}{T \cdot P}$
$T \cdot N < E \cdot P < 3T \cdot N$	$\frac{N^2}{P} + 2\frac{N^3}{T \cdot P}$
$T^2 < E \cdot P < 3T^2$	$3\frac{N^3}{T \cdot P}$
$T^2 > E \cdot P > T$	$2\frac{N^3}{T \cdot P} + \frac{N^3}{P}$
$T > E \cdot P$	$\frac{N^3}{T \cdot P} + 2\frac{N^3}{P}$

Πίνακας 4.2: Οι αστοχίες δεδομένων στο TLB

Το σχήμα 4.11 απεικονίζει τον αριθμό των αστοχιών δεδομένων στο TLB, σύμφωνα με την ανάλυση που προηγήθηκε για τα χαρακτηριστικά της αρχιτεκτονικής του UltraSPARC II.



Σχήμα 4.11: Ο αριθμός των αστοχιών του TLB για διάφορα μεγέθη πινάκων και tiles

Ο αριθμός των αστοχιών του TLB για όλα τα μεγέθη πινάκων, χρησιμοποιώντας τις παραμέτρους της αρχιτεκτονικής του UltraSPARC II, ελαχιστοποιείται όταν $T = 256$. Στο σημείο αυτό οι διευθύνσεις των σελίδων που περιέχουν τα δεδομένα ενός ολόκληρου tile χωράνε στις εγγραφές του TLB, και επομένως η επαναχρησιμοποίηση δεδομένων εντός του tile μπορεί να αξιοποιηθεί.

4.6 Αστοχίες στην Πρόβλεψη Διακλάδωσης

Τα επιπρόσθετα επίπεδα των φωλιασμένων βρόχων και τα μικρά σε μέγεθος tiles προκαλούν αύξηση του αριθμού των λανθασμένων προβλέψεων διακλάδωσης. Προκειμένου να αποφύγουμε μία επίδοση που θα αποκλίνει από τη βέλτιστη εξαιτίας της αύξησης αυτής, ενσωματώνουμε την επίπτωση των λανθασμένων προβλέψεων διακλάδωσης στο μοντέλο της συμπεριφοράς του υπό διερεύνηση συστήματος.

Χρησιμοποιούμε μία τροποποιημένη έκδοση της συνάρτησης που περιγράφεται από τον Vera στο [Ver03]. Θεωρούμε έναν φωλιασμένο βρόχο βάθους n. Ο χώρος των επαναλήψεων περιγράφεται από το διάνυσμα $I = \{I_1, \ldots, I_n\}$. Με βάση την υπόθεση ότι οι σημερινοί μηχανισμοί πρόβλεψης διακλάδωσης αποτυγχάνουν μόνο κατά το τέλος της εκτέλεσης ενός βρόχου, ο αριθμός των προσδοκώμενων λαθών πρόβλεψης διακλάδωσης υπολογίζεται ως εξής:

$M_{br} = \sum_{j=1}^{j \leq n} \prod_{i=1}^{i<j} I_i$

Ο πολλαπλασιασμός πινάκων στον οποίο έχουμε εφαρμόσει τους προτεινόμενους μετασχηματισμούς, καθώς και το σχήμα γρήγορης δεικτοδότησης , διαθέτει έναν φωλιασμένο βρόχο βάθους 6 με $I = \{\frac{N}{T}, \frac{N}{T}, \frac{N}{T}, T, T, T\}$. Κατά συνέπεια, ο αριθμός των αστοχιών πρόβλεψης διακλάδωσης είναι για το παράδειγμά μας:

$M_{br_6} = 1 + \frac{N}{T} + \left(\frac{N}{T}\right)^2 + \left(\frac{N}{T}\right)^3 + \left(\frac{N}{T}\right)^3 \times T + \left(\frac{N}{T}\right)^3 \times T^2$

Σε σύγκριση έναν φωλιασμένο βρόχο βάθους 5, όταν δεν έχει εφαρμοστεί μετασχηματισμός tiling σε όλους τους βρόχους (και πιο συγκεκριμένα στο βρόχο i), δεν υπάρχει σημαντική διαφορά σε ότι αφορά των αριθμό των αστοχιών πρόβλεψης διακλάδωσης. Δηλαδή, η αύξηση του βάθους του φωλιασμένου βρόχου όπως επιβάλλεται για την εφαρμογή του σχήματος γρήγορης δεικτοδότησης, δεν επιφέρει αξιοσημείωτη μείωση της επίδοσης:

$I = \{\frac{N}{T}, \frac{N}{T}, N, T, T\}$

και

$M_{br_5} = \sum_{j=1}^{j\leq 5} \prod_{i=1}^{i<j} I_i =$

$= 1 + \frac{N}{T} + \left(\frac{N}{T}\right)^2 + \left(\frac{N}{T}\right)^2 \times N + \left(\frac{N}{T}\right)^2 \times N \times T$

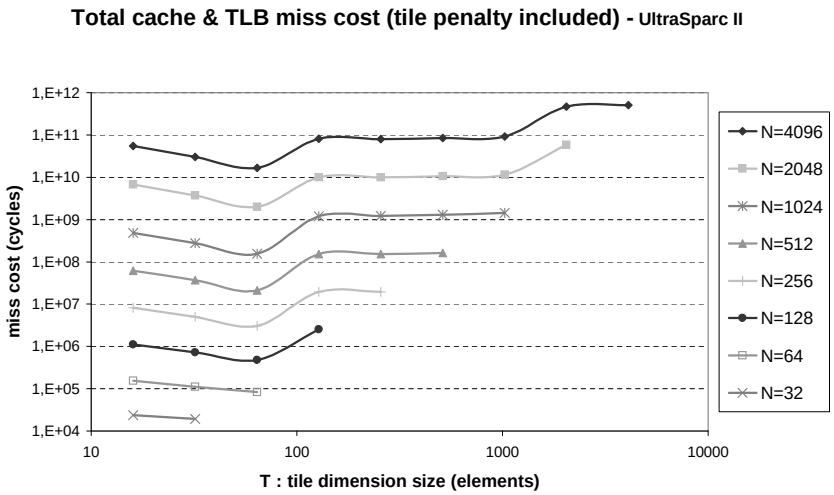
$\Delta M = M_{br_6} - M_{br_5} = \left(\frac{N}{T}\right)^3$

Για $N = 1024$ και $T = 32$, η αύξηση του ποσοστού $\left(\frac{\Delta M}{M_{br_5}}\right)$ είναι λιγότερο από 0.01%.

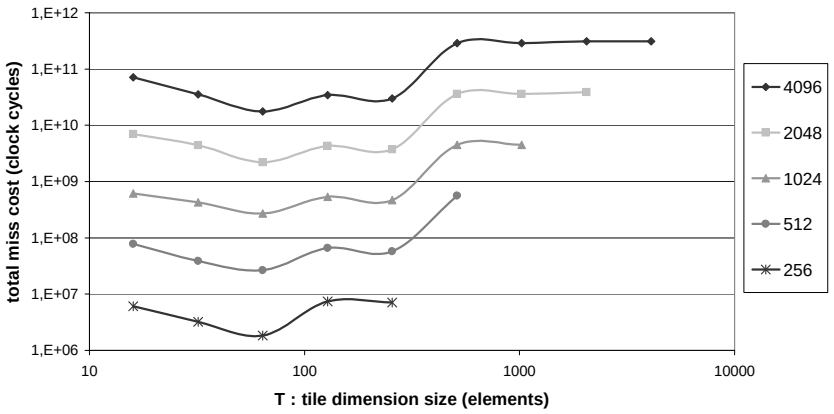
4.7 Συνολικό Κόστος των Αστοχιών

Λαμβάνοντας υπόψη την αρχική καθυστέρηση ανάγνωσης δεδομένων (miss penalty) από τα διαφορετικά επίπεδα της ιεραρχίας μνήμης, καθώς και τους άεργους κύκλους του επεξεργαστή λόγω λανθασμένης πρόβλεψης διακλάδωσης, προκύπτει η συνολική καθυστέρηση αστοχιών μνήμης και διακλάδωσης του σχήματος 4.12. Σημειώνουμε ότι έχουμε υπολογίσει μόνο τις αστοχίες δεδομένων της L2 κρυφής μνήμης, παρόλο που η L2 κρυφή μνήμη είναι ενοποιημένη για δεδομένα και εντολές. Ωστόσο, η αύξηση των αστοχιών δεδομένων στο συγκεκριμένο επίπεδο μνήμης, λόγω συγκρούσεων με εντολές, είναι πραγματικά αμελητέα σε σχέση με τον συνολικό αριθμό αστοχιών του συστήματος της ιεραρχίας.

Σύμφωνα με το σχήμα 4.12, στην αρχιτεκτονική του UltraSPARC II, η επίδοση του συστήματος φτάνει στο μέγιστο όταν $T = 64$ ($T = \sqrt{C_{L1}}$). Πρόκειται για το μέγιστο μέγεθος tile του οποίου τα δεδομένα χωρούν εξ' ολοκλήρου στην L1 κρυφή μνήμη. Στο σημείο της βέλτιστης επίδοσης, οι αστοχίες της L1 κρυφής μνήμης είναι μία τάξη μεγέθους περισσότερες σε σχέση με τις αστοχίες της L2 κρυφής μνήμης και τρεις τάξεις μεγέθους περισσότερες σε σχέση με τις αστοχίες του TLB. Επιπρόσθετα, μπορούν ακόμα και να τετραπλασιαστούν, όταν τα δεδομένα των tiles υπερχειλίζουν τη L1 κρυφή μνήμη. Κατά συνέπεια, παρόλο που οι αστοχίες της L1 κρυφής μνήμης κοστίζουν σημαντικά λιγότερους κύκλους ρολογιού σε σύγκριση με τα άλλα επίπεδα μνήμης, επιβαρύνουν περισσότερο την επίδοση του συστήματος. Επομένως είναι ο κυρίαρχος παράγοντας στην συνολική συμπεριφορά του συστήματος μνήμης. Σημειώνουμε, επίσης, ότι στο σημείο που επιτυγχάνεται βέλτιστη επίδοση ($T = \sqrt{C_{L1}}$), οι αστοχίες της L1 κρυφής μνήμης δεν ελαχιστοποιούνται. Παρατηρείται μία μικρή αύξηση σε σχέση με το αμέσως μικρότερο μέγεθος tile. Από την άλλη πλευρά, οι αστοχίες της L2 κρυφής μνήμης και του TLB, καθώς και ο αριθμός των λανθασμένων προβλέψεων διακλάδωσης μειώνονται. Η συνδυασμένη αυτή μείωση αντισταθμίζει την μικρή αύξηση των αστοχιών της L1 κρυφής μνήμης. Οι αστοχίες της L1 κρυφής μνήμης τελικά αποτελούν τον κυρίαρχο παράγοντα στη συμπεριφορά του συνολικού συστήματος μνήμης, καθώς



Σχήμα 4.12: Το συνολικό κόστος των αστοχιών στην αρχιτεκτονική UltraSPARC II



Σχήμα 4.13: Το συνολικό κόστος των αστοχιών στην αρχιτεκτονική Pentium III

η απότομη αύξηση των αστοχιών του συγκεκριμένου επιπέδου, λόγω υπερχείλισης της L1 κρυφής μνήμης (όταν $T > \sqrt{C_{L1}}$), δεν μπορεί να αντισταθμιστεί από κάποια μείωση σε οποιοδήποτε άλλο επίπεδο μνήμης. Η απότομη αυτή αύξηση καθορίζει ότι το βέλτιστο μέγεθος tile είναι $\sqrt{C_{L1}}$.

Τέλος, αξίζει να παρατηρήσουμε ότι η αρχιτεκτονική του UltraSPARC II έχει μία αρκετά μεγάλη L2 κρυφή μνήμη (4Mbytes). Το γεγονός αυτό μειώνει σημαντικά τον αριθμό των αστοχιών που λαμβάνουν χώρα στο επίπεδο αυτό μνήμης, και επομένως επιτείνει την κυριαρχία των αστοχιών της L1 κρυφής μνήμης σε ότι αφορά την συνολική επίδοση του συστήματος.

Η αρχιτεκτονική του UltraSPARC II διαθέτει κρυφές μνήμες ευθείας απεικόνισης. Όπως έγινε φανερό στις παραγράφους 4.3 και 4.4, η οργάνωση αυτή έχει σαν επίπτωση έναν μεγάλο αριθμό από αστοχίες σύγκρουσης. Στην περίπτωση που οι κρυφές μνήμες είναι συσχέτισης συνόλου, οι αστοχίες σύγκρουσης μπορούν να εξαλειφθούν. Ακόμη και η συσχέτιση συνόλου δύο δρόμων επαρκεί για την περίπτωση του πολλαπλασιασμού πινάκων όπου προσπελάζονται δεδομένα τριών διαφορετικών πινάκων. Η κατάλληλη ευθυγράμμιση μπορεί να αποτρέψει τη διεκδίκηση της

ίδιας γραμμής κρυφής μνήμης από περισσότερους από δύο πίνακες ταυτόχρονα. Επομένως, όσο το επιτρέπει η χωρητικότητα της κρυφής μνήμης, η επαναχρησιμοποίηση δεδομένων μπορεί να αξιοποιηθεί. Στην περίπτωση των κρυφών μνημών συσχέτισης συνόλου, οι εξισώσεις που προσδιορίζουν τον αριθμό των αστοχιών ακολουθούν τη λογική της παραγράφου 4.5 (εξάλλου, το TLB είναι μία κρυφή μνήμη πλήρους συσχέτισης).

Το σχήμα 4.13 απεικονίζει την συμπεριφορά της ιεραρχίας μνήμης του Pentium III (Coppermine), ο οποίος διαθέτει συσχετιστικές κρυφές μνήμες.

4.8 Σύνοψη

Ένας μεγάλος αριθμός εργασιών της βιβλιογραφίας αναζητούν το βέλτιστο μέγεθος και σχήμα για τα χρησιμοποιούμενα tiles, στους κώδικες που περιέχουν φωλιασμένους βρόχους και έχει εφαρμοστεί σε αυτούς ο μετασχηματισμός tiling. Στο συγκεκριμένο κεφάλαιο, εξετάσαμε την επίδοση του συστήματος μνήμης, στην περίπτωση που εκτελούνται φωλιασμένοι βρόχοι με μη-γραμμική διάταξη ομαδοποίησης δεδομένων, και η δεικτοδότηση των πινάκων γίνεται μέσω δυαδικών μασκών. Με τη θεωρητική ανάλυση βρήκαμε ότι σε συστήματα κρυφών μνημών ευθείας απεικόνισης, με μεγάλο μέγεθος του επιπέδου L2 της κρυφής μνήμης, οι αστοχίες της L1 κρυφής μνήμης καθορίζουν τη συνολική επίδοση του συστήματος. Οι τεχνικές πρώιμης ανάκλησης σε συνδυασμό με άλλες τεχνικές βελτιστοποίησης κώδικα, επιτυγχάνουν βέλτιστη επίδοση όταν το μέγεθος του tile είναι $T_1 = \sqrt{C_{L1}}$. Στο σημείο αυτό οι αστοχίες του επιπέδου L1 της κρυφής μνήμης αυξάνουν ελαφρώς σε σχέση με την ελάχιστη τιμή τους που σημειώνεται στο αμέσως μικρότερο μέγεθος tile. Ωστόσο, η μείωση των αστοχιών του επιπέδου L2 και του TLB, καθώς και οι μείωση των λανθασμένων προβλέψεων της απόφασης των εντολών διακλάδωσης, αντισταθμίζουν την μικρή αυτή αύξηση και δίνουν βέλτιστη συνολική επίδοση. Τα θεωρητικά αυτά αποτελέσματα επιβεβαιώνονται από τις μετρήσεις χρόνου και την προσομοίωση της εκτέλεσης του κεφαλαίου 6.

ΚΕΦΑΛΑΙΟ 5

Ταυτόχρονη Πολυνημάτωση

Η ταυτόχρονη πολυνημάτωση (Simultaneous Multi-threading: SMT) είναι μία τεχνική που επιτρέπει την ταυτόχρονη εκτέλεση πολλαπλών νημάτων (είτε του ίδιου παραλληλοποιημένου προγράμματος, είτε διαφορετικών προγραμμάτων) στον ίδιο φυσικό επεξεργαστή. Ο στόχος των αρχιτεκτονικών ταυτόχρονης πολυνημάτωσης είναι η επικάλυψη των καθυστερήσεων, που προκύπτουν λόγω επικοινωνίας με τη μνήμη και εντός των λειτουργικών μονάδων του επεξεργαστή. Αυτό επιτυγχάνεται με την αξιοποίηση των άεργων κύκλων του επεξεργαστή, που θα προέκυπταν από την εκτέλεση των εντολών ενός μόνο νήματος, μέσω της ταυτόχρονης εκτέλεσης εντολών από άλλα νήματα. Τα ταυτόχρονα εκτελούμενα νήματα μπορεί να προέρχονται, είτε από ένα παράλληλο πρόγραμμα, είτε από διαφορετικά προγράμματα.

Η πρόσφατη βιβλιογραφία έχει αποδείξει ότι η τεχνική της ταυτόχρονης πολυνημάτωσης μπορεί να επιτύχει σημαντική επιτάχυνση στο χρόνο εκτέλεσης των εφαρμογών, όταν οι εφαρμογές αυτές είναι ανομοιογενείς και επομένως χρησιμοποιούν διαφορετικές λειτουργικές μονάδες του επεξεργαστή. Πέρα από αυτό, η επίδοση των παράλληλων εφαρμογών που εκτελούνται σε επεξεργαστές με δυνατότητες ταυτόχρονης πολυνημάτωσης, εξαρτάται σε μεγάλο βαθμό από την αποδοτικότητα των μηχανισμών συγχρονισμού και επικοινωνίας μεταξύ των διαφορετικών, αλλά πιθανότατα εξαρτώμενων μεταξύ τους νημάτων. Επιπρόσθετα, στις περισσότερες περιπτώσεις οι παράλληλες εφαρμογές παράγουν νήματα, τα οποία έχουν ανάγκη από τις ίδιες λειτουργικές μονάδες. Αυτό ασφαλώς προκαλεί συμφόρηση σε συγκεκριμένες μονάδες του επεξεργαστή και ουσιαστικά αποτρέπει την επιτάχυνση του χρόνου εκτέλεσης των εφαρμογών.

Στο κεφάλαιο αυτό, συγκρίνουμε τις τεχνικές της πρώιμης ανάκλησης δεδομένων και της παραλληλοποίησης σε επίπεδο νήματος, για εφαρμογές που περιέχουν μεγάλο όγκο υπολογισμών εντός επαναληπτικών δομών, βελτιστοποιημένων, σύμφωνα με τις τεχνικές που έχουν περιγραφεί στα κεφάλαια 3 και 4. Τέλος, εξετάζουμε τα όρια επίδοσης της αρχιτεκτονικής ταυτόχρονης πολυνημάτωσης, εκτελώντας ομοιογενή νήματα που περιέχουν τους συνηθέστερους τύπους εντολών.

5.1 Εισαγωγή

Οι τεχνικές βελτιστοποίησης κώδικα που μελετήσαμε στα προηγούμενα κεφάλαια καθώς και η πρόοδος που σημειώνεται στην τεχνολογία της κρυφής μνήμης, μπορούν να περιορίσουν ως ένα σημείο το χάσμα επίδοσης μεταξύ επεξεργαστών και κύριας μνήμης. Από την άλλη πλευρά, η ταχύτητα των επεξεργαστών συνεχίζει να αυξάνει γρηγορότερα από την ταχύτητα επικοινωνίας με τη μνήμη, παράλληλα αυξάνεται ο όγκος των δεδομένων προς επεξεργασία καθώς και η πολυπλοκότητα των συμβατικών εφαρμογών, γεγονός που θέτει ένα πάνω όριο στη δυνατότητα παραλληλισμού των εφαρμογών σε επίπεδο εντολών.

Μία άλλη προσέγγιση στη προσπάθεια μείωσης του χάσματος μεταξύ επεξεργαστή και μνήμης είναι η τεχνική της ταυτόχρονης πολυνημάτωσης [HKN+92], [TEE+96], [TEL95]. Πρόκειται για μία τεχνική σε επίπεδο υλικού, η οποία, όπως προαναφέρθηκε, επιτρέπει σε εντολές από διαφορετικά ανεξάρτητα νήματα να εισάγονται στις λειτουργικές μονάδες του επεξεργαστή και να εκτελούνται ταυτόχρονα στον ίδιο κύκλο ρολογιού. Με αυτόν τον τρόπο, εκμεταλλευόμενοι τον παραλληλισμό σε επίπεδο εντολής περισσότερων του ενός νημάτων, αυξάνονται οι δυνατότητες αξιοποίησης όλων των λειτουργικών μονάδων του επεξεργαστή. Υπάρχουν, δηλαδή, αρκετά περισσότερες διαθέσιμες εντολές προς εκτέλεση σε κάθε κύκλο ρολογιού, επομένως, αρκετά καλύτερες πιθανότητες εξάλειψης των άεργων χρόνων εκτέλεσης και μεγαλύτερη ευελιξία [TEE+96].

Οι επεξεργαστές που διαθέτουν την τεχνική της ταυτόχρονης πολυνημάτωσης μπορούν να ξεπεράσουν σε επίδοση όλους τους προγενέστερους επεξεργαστές, επειδή συνδυάζουν τα χαρακτηριστικά των υπερβαθμωτών αρχιτεκτονικών (όπου διαδοχικές εντολές μίας διεργασίας μπορούν να εκτελούνται ταυτόχρονα μέσα στις λειτουργικές μονάδες του επεξεργαστή), και των αρχιτεκτονικών πολυνημάτωσης (όπου οι εντολές πολλών νημάτων μπορούν να εναλλάσσονται στον επεξεργαστή, προκειμένου να κρυφτούν οι άεργοι κύκλοι εκτέλεσης, οι οποίοι προκύπτουν λόγω των καθυστερήσεων που εισάγουν οι αναγνώσεις από τη μνήμη, και λόγω των εξαρτήσεων μεταξύ των εντολών). Στην τεχνική της ταυτόχρονης πολυνημάτωσης όλα τα νήματα είναι ενεργά και ανταγωνίζονται μεταξύ τους σε κάθε κύκλο για τους διαθέσιμους πόρους του επεξεργαστή. Πρόκειται, δηλαδή, για ένα δυναμικό διαμοιρασμό των λειτουργικών μονάδων του επεξεργαστή μεταξύ των ενεργών νημάτων, με το οποίο καταπολεμούνται οι δύο βασικές αιτίες που αναστέλλουν την αύξηση του ρυθμού εκτέλεσης των εντολών: οι μακρόχρονες καθυστερήσεις και ο περιορισμένος παραλληλισμός των εφαρμογών σε επίπεδο εντολής. Από την άλλη πλευρά, αυτή η αυξημένη ευελιξία της τεχνικής ταυτόχρονης πολυνημάτωσης έχει κάποιο κόστος. Καταρχάς, οι επεξεργαστές που εκτελούν πολλές εντολές ταυτόχρονα είναι περισσότερο επιρρεπείς στην υπερχείλιση των κρυφών μνημών. Επιπλέον, ο διαμοιρασμός των δομικών/λειτουργικών μονάδων του επεξεργαστή δεν γίνεται σε όλα τα επίπεδα με δυναμικό τρόπο. Η ουρά εντολών (instruction queue), ο προσωρινός καταχωρητής αναδιάταξης εντολών (reorder buffer), η ουρά αποθήκευσης δεδομένων (store queue), κ.α. διαμοιράζονται στατικά, το οποίο μπορεί να επηρεάσει αρνητικά τις εφαρμογές που έχουν αυξημένο ρυθμό εκτέλεσης. Ο στατικός αυτός διαμοιρασμός των πόρων του επεξεργα-

στή μπορεί να επηρεάζει αρνητικά την επίδοση όμοιων (από άποψη σύστασης εντολών) νημάτων, αλλά μετριάζει τις μεγάλες καθυστερήσεις στις περιπτώσεις ανόμοιων νημάτων (με διαφορετική δυναμική ως προς τη δέσμευση λειτουργικών μονάδων) [TT03].

Παράλληλα με την ταυτόχρονη πολυνημάτωση, η τεχνική της πρώιμης ανάκλησης δεδομένων [LM96], [LM99], [MG91] μπορεί να κρύψει ως ένα σημείο τους άεργους κύκλους εκτέλεσης λόγω ανάκτησης δεδομένων από τη μνήμη. Σε αντίθεση με την ταυτόχρονη πολυνημάτωση, που αξιοποιεί τους άεργους κύκλους εκτέλεσης, που προκύπτουν από τις αστοχίες της κρυφής μνήμης, η πρώιμη ανάκληση προλαμβάνει τις αστοχίες αυτές, προβλέποντας ποια δεδομένα απαιτούνται προς επεξεργασία στο μέλλον, και μεταφέροντάς τα στην κρυφή μνήμη αρκετά νωρίτερα, ώστε να είναι διαθέσιμα τη στιγμή που θα τα χρειαστεί ο επεξεργαστής. Αν η πρώιμη ανάκληση πραγματοποιηθεί αρκετά νωρίς, αλλά όχι υπερβολικά νωρίς, τόσο που τα δεδομένα να διαγραφούν από την κρυφή μνήμη λόγω σύγκρουσης πριν προφτάσουν να χρησιμοποιηθούν, τότε η τεχνική αυτή μπορεί να κρύψει τέλεια την καθυστέρηση λόγω αναγκαστικής ανάγνωσης δεδομένων από την κύρια μνήμη.

Ωστόσο, υπάρχουν και σε αυτήν την περίπτωση μειονεκτήματα, παρόλο που, λόγω της συγκάλυψης των καθυστερήσεων που προκύπτουν από τις αστοχίες μνήμης, ο ρυθμός εκτέλεσης εντολών αυξάνεται. Καταρχάς, αυξάνονται οι απαιτήσεις σε ότι αφορά το ρυθμό μεταφοράς δεδομένων από τη μνήμη, επειδή η πρώιμη ανάκληση δεδομένων αυξάνει την μετακίνηση δεδομένων μεταξύ κρυφής και κύριας μνήμης. Επιπλέον, αυξάνονται οι απαιτήσεις χωρητικότητας της κρυφής μνήμης, επειδή ένας μεγάλος αριθμός δεδομένων μεταφέρονται στην κρυφή μνήμη από την κύρια προκειμένου να χρησιμοποιηθούν από τις επερχόμενες εντολές προς εκτέλεση. Τέλος, αυξάνεται ο συνολικός αριθμός των εκτελούμενων εντολών, γεμίζοντας τη σωλήνωση του επεξεργαστή με χρονοβόρες εντολές. Οι εντολές αυτές υπολογίζουν ποια δεδομένα θα χρησιμοποιηθούν μελλοντικά από τις επερχόμενες εντολές (που εκτελούν τον πραγματικό όγκο δουλειάς που χρειάζεται η εφαρμογή), για να τα καλέσουν πρώιμα στην κρυφή μνήμη [BAYT04].

Οι τεχνικές πρώιμης ανάκλησης που υλοποιούνται σε επίπεδο υλικού [Che95], [JG97], [Jou90] έχουν το πλεονέκτημα ότι δεν απαιτούν υποστήριξη από κάποιον πολύπλοκο ή ειδικής χρήσης μεταγλωττιστή, ούτε εισάγουν επιπρόσθετες εντολές στην εφαρμογή. Αντίθετα, αξιοποιούν την πληροφορία που προκύπτει κατά το χρόνο εκτέλεσης της εφαρμογής, βασιζόμενες στην υπόθεση ότι οι μελλοντικές προσπελάσεις στη μνήμη θα ακολουθούν το ίδιο μοτίβο με τις προγενέστερες προσπελάσεις.

Οι τεχνικές πρώιμης ανάκλησης που υλοποιούνται σε επίπεδο λογισμικού [MG91], έχουν μικρότερες απαιτήσεις σε ότι αφορά το ρυθμό μεταφοράς δεδομένων μεταξύ κρυφής και κύριας μνήμης, γιατί πραγματοποιεί στοχευμένες και εξειδικευμένες για την κάθε εφαρμογή πρώιμες ανακλήσεις δεδομένων στην κρυφή μνήμη και προκαλεί λιγότερες συγκρούσεις με χρήσιμα δεδομένα, σε σχέση με την υλοποίηση της πρώιμης ανάκλησης σε επίπεδο υλικού. Από την άλλη πλευρά, εισάγεται ένας σημαντικός αριθμός εντολών για τον υπολογισμό της θέσης των δεδομένων που θα χρησιμοποιηθούν μελλοντικά από τις επερχόμενες εντολές της εφαρμογής, και για την πρώιμη ανάκληση των δεδομένων αυτών στην κρυφή μνήμη. Σε κάποιες περιπτώσεις, οι επιπρόσθετες

αυτές εντολές μπορεί να είναι τόσο χρονοβόρες, ώστε ο χρόνος που απαιτείται για την εκτέλεσή τους, να υπερσκελίζει το χρόνο που κερδίζουμε λόγω της συγκάλυψης των καθυστερήσεων από την πρώιμη ανάκληση δεδομένων.

Οι επιδόσεις της πρώιμης ανάκλησης δεδομένων σε επίπεδο υλικού και σε επίπεδο λογισμικού έχει βρεθεί ότι είναι περίπου ισοδύναμες [CB94]. Βέβαια, η πρώιμη ανάκληση σε επίπεδο υλικού είναι "ακριβότερη" κατασκευαστικά, τόσο από άποψη των επιπλέον απαιτήσεων στο ρυθμό μεταφοράς δεδομένων μεταξύ κρυφής και κύριας μνήμης, όσο και σε ότι αφορά το συνολικό κόστος του συστήματος. Οι τεχνικές αυτές, τόσο σε επίπεδο υλικού όσο και σε επίπεδο λογισμικού, μπορούν να συνεργαστούν πολύ καλά με την τεχνική της ταυτόχρονης πολυνημάτωσης. Στο συνδυασμό των δύο τελευταίων, με στόχο την αύξηση της επίδοσης όταν μία μόνο εφαρμογή εκτελείται στον επεξεργαστή, θα επικεντρωθούμε στη συνέχεια στο παρόν κεφάλαιο.

Από την εισαγωγή της έννοιας της ταυτόχρονης πολυνημάτωσης σε ερευνητικό επίπεδο μέχρι σήμερα, που με κάποιες παραλλαγές έχει υλοποιηθεί πλέον στους σύγχρονους επεξεργαστές, έχουν προταθεί δύο τεχνικές με στόχο την αύξηση της επίδοσης της κάθε εφαρμογής μέσω της αξιοποίησης των πολλαπλών νημάτων που υποστηρίζει ο επεξεργαστής: ο παραλληλισμός σε επίπεδο νήματος (thread-level parallelism - TLP) και η υποθετική εκτέλεση εντολών (speculative precomputation - SPR) [TWN04]. Με τον παραλληλισμό σε επίπεδο εντολής, οι σειριακοί κώδικες παραλληλίζονται, έτσι ώστε ο συνολικός όγκος υπολογισμών να μοιραστεί σε έναν αριθμό από ανεξάρτητα τμήματα, τα οποία θα εκτελεστούν από διαφορετικά νήματα. Στην υποθετική εκτέλεση εντολών, οι εφαρμογές διαθέτουν, εκτός από τα συνηθισμένα νήματα που εκτελούν τους υπολογισμούς, επιπλέον νήματα που εκτελούν υποθετικές εντολές, δηλαδή, πραγματοποιούν πρώιμη ανάκληση υποθετικών δεδομένων στην κοινή κρυφή μνήμη, τα οποία πιθανότατα να χρησιμοποιηθούν στη συνέχεια από τα "συγγενή" κύρια νήματα, που εκτελούν τους υπολογισμούς της εφαρμογής. Με αυτόν τον τρόπο, καλύπτεται η καθυστέρηση που εισάγει η ανάγνωση δεδομένων από την κύρια μνήμη και μειώνεται ο αριθμός των αστοχιών δεδομένων [WWW+02], [KLW+04], [TWN04]. Βέβαια, επιβάλλεται να είμαστε ιδιαίτερα προσεκτικοί όταν μετασχηματίζουμε της εφαρμογές, με σκοπό να εκτελεστούν από πολλαπλά παράλληλα νήματα σε αρχιτεκτονικές SMT (που διαθέτουν ταυτόχρονη πολυνημάτωση). Όταν ενεργοποιηθεί ο μηχανισμός ταυτόχρονης πολυνημάτωσης, οι δομικές και λειτουργικές μονάδες του επεξεργαστή (reservation station, renaming registers, fetch/decode units) μοιράζονται μεταξύ των λογικών επεξεργαστών και επομένως υπάρχει πιθανότητα αυξημένου ανταγωνισμού για συγκεκριμένους πόρους του επεξεργαστή. Ακόμα και όταν ένα νήμα είναι αδρανές, μπορεί να οδηγήσει στην εκκένωση της μονάδας διαμοιρασμού των εντολών στις λειτουργικές μονάδες του επεξεργαστή (dispatch unit), εξαιτίας του στατικού διαμοιρασμού των μονάδων ανάγνωσης και αποκωδικοποίησης των εντολών (fetch and decode units) [SU96].

Στη βιβλιογραφία, έχουν προταθεί πολυάριθμες τεχνικές πρώιμης ανάκλησης δεδομένων που βασίζονται στην ταυτόχρονη πολυνημάτωση. Σε αυτό το σημείο αναφέρουμε τους Zilles και Sohi (Speculative Slices) [ZS01], Roth και Sohi (Data Driven Multithreading) [RS01], Luk (Software Controlled Pre-Execution) [Luk01], Annavaram κ.α. (Data Graph Precomputation)

[APD01], Moshovos κ.α. (Slice-Processors) [MPB01], Collins κ.α. (Speculative Precomputation) [CWT+01], Kim κ.α. (Helper-Threads) [KLW+04] και Sundaramoorthy κ.α. (Slipstream Processors) [SPR00]. Η βασική ιδέα σε όλες τις παραπάνω εργασίες είναι η αξιοποίηση των ανενεργών νημάτων για την εκτέλεση υποθετικών εντολών προκειμένου να επιταχυνθεί ο ρυθμός εκτέλεσης στο κύριο νήμα. Οι υποθετικές εντολές των βοηθητικών νημάτων στοχεύουν στον εντοπισμό μελλοντικών αστοχιών δεδομένων και στην πρώιμη ανάκληση των δεδομένων αυτών, αρκετά νωρίτερα από την προσπέλασή τους από το κύριο νήμα, ώστε να κρυφτεί η καθυστέρηση της ανάγνωσής τους από την κύρια μνήμη. Η υλοποίηση της πρώιμης ανάκλησης δεδομένων έχει αρκετά κοινά σημεία στις παραπάνω εργασίες. Ο μεταγλωττιστής προδιαγράφει, είτε στατικά, είτε χρησιμοποιώντας το προφίλ της εφαρμογής, τις αναγνώσεις από την κύρια μνήμη (εντολές φόρτωσης στη μνήμη), οι οποίες έχουν αυξημένη πιθανότητα να αστοχήσουν στην κρυφή μνήμη. Τέτοιου είδους αναγνώσεις από τη μνήμη (delinquent loads), μπορούν να αναγνωριστούν και δυναμικά σε επίπεδο υλικού [CTWS01], [WWW+02], προκαλώντας τη εκκίνηση υποθετικών-βοηθητικών νημάτων. Η τεχνική της υποθετικής εκτέλεσης εντολών, στοχεύει σε αστοχίες κρυφής μνήμης δύσκολα προβλέψιμες, εξαρτώμενες από το αποτέλεσμα προηγούμενων εντολών ή έμμεσων αναφορών. Συνήθως τέτοιες εντολές ανάγνωσης είναι δύσκολο να τις χειριστούν οι προγενέστερες τεχνικές πρώιμης ανάκλησης δεδομένων, τόσο σε επίπεδο υλικού όσο και λογισμικού.

Το μεγαλύτερο μέρος των προγενέστερων εργασιών, βασίζουν τα αποτελέσματά τους σε προσομοιώσεις μοντέλων ταυτόχρονης πολυνημάτωσης. Στα πειράματα με πραγματικούς επεξεργαστές εξοπλισμένους με την τεχνική υπερ-νημάτωσης (Intel Hyper-Threading - HT) [BP04], [KLW+04], [TT03], δεν έχουν επιτευχθεί σημαντικές επιταχύνσεις στις παράλληλες εφαρμογές, σε σύγκριση με την σειριακή εκδοχή της εφαρμογής. Η υπεροχή των αρχιτεκτονικών ταυτόχρονης πολυνημάτωσης εξαρτάται από το είδος της εφαρμογής και από το πόσο βελτιστοποιημένη είναι [MCFT99]. Για παράδειγμα, ο βελτιστοποιημένος πολλαπλασιασμός πινάκων αξιοποιεί ήδη με τον καλύτερο τρόπο τους προσωρινούς καταχωρητές και επαναχρησιμοποιεί τα δεδομένα που βρίσκονται ήδη στην κρυφή μνήμη. Σε αυτό το επίπεδο βελτιστοποίησης, τα όποια επιπρόσθετα νήματα θα επηρεάσουν αρνητικά την επίδοση της μικρο-εφαρμογής. Από την άλλη πλευρά, ένας πολλαπλασιασμός πινάκων με αφελή υλοποίηση, ο οποίος δεν έχει καλή αξιοποίηση της τοπικότητας των αναφορών, θα ωφεληθεί από την παραλληλοποίηση της μικρο-εφαρμογής σε επίπεδο νήματος.

Το κεφάλαιο αυτό οργανώνεται στη συνέχεια ως εξής: Η παράγραφος 5.2 περιγράφει την υλοποίηση των μεθόδων σε επίπεδο λογισμικού, που χρησιμοποιούνται για την αξιοποίηση της τεχνικής της ταυτόχρονης πολυνημάτωσης. Η παράγραφος 5.3 παρουσιάζει τις λεπτομέρειες και τα αποτελέσματα της ταυτόχρονης εκτέλεσης ομοιογενών νημάτων στον επεξεργαστή Intel Xeon DP, εξοπλισμένος με την τεχνική της υπερ-νημάτωσης, εντοπίζοντας τα σημεία σύγκρουσης των εκτελούμενων ροών από εντολές κατά τη διεκδίκηση των λειτουργικών μονάδων του επεξεργαστή. Τέλος, κλείνουμε με την παράγραφο 5.4.

5.2 Υλοποίηση

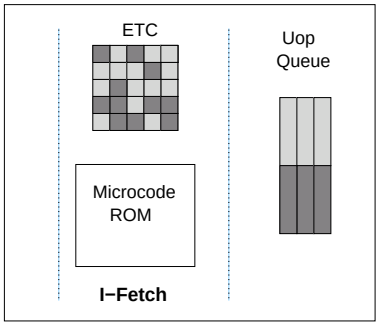
Παραλληλισμός σε επίπεδο νήματος (TLP)

Παραλληλίζοντας τις εφαρμογές σε επίπεδο νήματος, μοιράζουμε το υπολογιστικό φορτίο σε δύο νήματα. Τα νήματα αυτά ανταγωνίζονται για τους κοινούς πόρους του συστήματος με στόχο την ελαχιστοποίηση των άεργων κύκλων εκτέλεσης. Ωστόσο, η παραλληλοποίηση των εφαρμογών χρειάζεται προσοχή σε ότι αφορά τις καθυστερήσεις που μπορούν εύκολα να εισαχθούν, αν η υλοποίηση δεν σχεδιαστεί επιμελώς. Καταρχάς πρέπει να ληφθεί υπόψη ο συγχρονισμός μεταξύ των νημάτων. Οι εξαρτήσεις μεταξύ των διαφορετικών νημάτων μπορούν να οδηγήσουν σε δραματική επιβράδυνση των εφαρμογών. Στο σημείο αυτό χρειάζεται να ανατρέξουμε σε προγενέστερες εργασίες που μελετούν τη δρομολόγηση και αναλύουν τις εξαρτήσεις μεταξύ των διαφορετικών διεργασιών. Ωστόσο, η περίπτωση της παραλληλοποίησης σε επίπεδο νήματος, όπου οι επεξεργαστικοί πόροι του συστήματος δεν αυξάνονται με την αύξηση των διεργασιών/νημάτων, αλλά απλώς κατανέμονται μεταξύ αυτών, είναι ιδιάζουσα. Ο Πίνακας 5.1 και το σχήμα 5.1 δείχνουν τους πόρους του επεξεργαστή, όπως αυτοί μοιράζονται μεταξύ των λογικών επεξεργαστών του συστήματος, όταν ενεργοποιείται η τεχνική υπερ-νημάτωσης σε επεξεργαστές Intel (Intel Hyper-Threading).

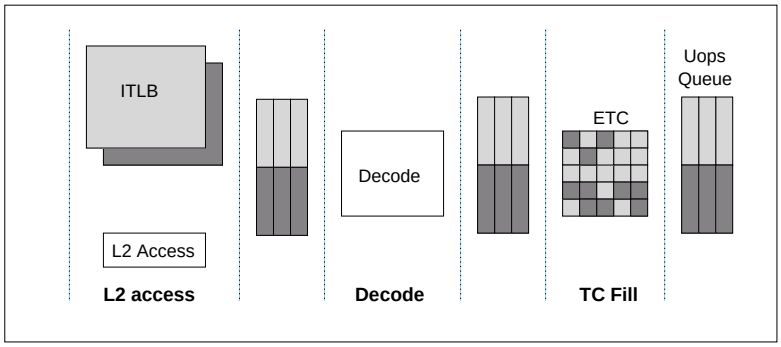
Shared	Execution units, Trace cache, L1 D-cache, L2 cache, DTLB, Global history array, Allocator, Microcode ROM, μop retirement logic, IA-32 instruction decode, Instruction scheduler, Instruction fetch logic
Duplicated	Processor architecture state, Instruction pointers, Rename Logic, ITLB, Streaming buffers, Return stack buffer, Branch history buffer
Partitioned	μop queue, Memory instruction queue, Reorder buffer, General instruction queue

Πίνακας 5.1: Η διαχείριση του υλικού στους επεξεργαστές της Intel που διαθέτουν την τεχνολογία Υπερνημάτωσης

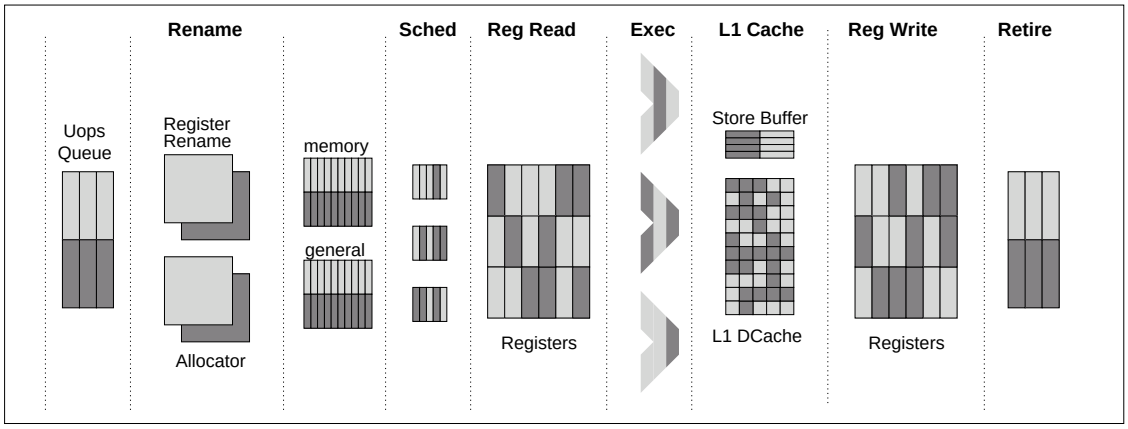
Ο χρόνος και οι πόροι του συστήματος που σπαταλούνται για το συγχρονισμό των νημάτων, όπως αναφέρεται εκτενέστερα στην συνέχεια της παραγράφου 5.2, δεν θα πρέπει να υπερβαίνει το χρόνο που κερδίζουμε μέσω της παράλληλης διεκδίκησης των πόρων του συστήματος από τα πολλαπλά νήματα. Ειδικά στην περίπτωση των βελτιστοποιημένων εφαρμογών με κυρίαρχο το υπολογιστικό φορτίο, όπου έχουν κατά το δυνατό ελαχιστοποιηθεί οι άεργοι κύκλοι εκτέλεσης,



(α) Το μονοπατι εκτέλεσης κατά την ευστοχία δεδομένων της trace cache



(β) Το μονοπατι εκτέλεσης κατά την αστοχία δεδομένων της trace cache



(γ) Το εκτός-σειράς μονοπάτι εκτέλεσης των εντολών

Σχήμα 5.1: Ο διαχωρισμός των πόρων του συστήματος μεταξύ των ταυτόχρονα εκτελούμενων νημάτων στην αρχιτεκτονική των επεξεργαστών της Intel εξοπλισμένων με την τεχνική της υπερνημάτωσης

υπάρχει ο κίνδυνος της συνολικής επιβράδυνσης των εφαρμογών, όταν οι συνεχείς συγκρούσεις επάνω στις ίδιες λειτουργικές μονάδες και ο στατικός διαχωρισμός κάποιων δομικών μονάδων του συστήματος εισάγουν δυσανάλογα μεγάλες καθυστερήσεις.

Η υποθετική εκτέλεση εντολών

Η υποθετική εκτέλεση εντολών(πρώιμη ανάκληση δεδομένων με αξιοποίηση της ταυτόχρονης πολυνημάτωσης) είναι τεχνική κατάλληλη για εφαρμογές που δεν μπορούν να παραλληλοποιηθούν εύκολα ή αποδοτικά. Το βοηθητικό νήμα εκτέλεσης περιέχει μόνο τις κρίσιμες εντολές, δηλαδή εκείνες που προσπελάζουν τον μεγαλύτερο όγκο των δεδομένων που ενδέχεται να προκαλέσει αστοχίες κρυφής μνήμης. Στην υλοποίησή μας το βοηθητικό νήμα δεν επηρεάζει τη ροή εκτέλεσης του κύριου νήματος. Ο στόχος είναι, εκμεταλλευόμενοι τους άεργους κύκλους των λειτουργικών μονάδων, να πραγματοποιηθεί έγκαιρα η πρώιμη ανάκληση δεδομένων, τα οποία θα βρίσκονται στην κρυφή μνήμη, όταν θα τα χρειαστεί το κύριο νήμα που εκτελεί τους υπολογισμούς της εφαρμογής.

Στην υλοποίηση της υποθετικής εκτέλεσης εντολών χρειάζεται ιδιαίτερη προσοχή σε δύο βα-

σικά σημεία:

- Καταρχάς, πρέπει να ληφθεί υπόψη η απόσταση μεταξύ της επερχόμενης εντολής φόρτωσης (που ενδέχεται να καταλήξει σε αστοχία κρυφής μνήμης) και της εντολής πρώιμης ανάκλησης της αντίστοιχης θέσης μνήμης. Η απόσταση αυτή (prefetch distance), όπως έχει ήδη αναφερθεί, θα πρέπει να είναι αρκετά μεγάλη, ώστε τα δεδομένα που καλούνται πρώιμα στην κρυφή μνήμη να προφτάσουν να έρθουν σε αυτήν, πριν αστοχήσει η επερχόμενη εντολή ανάγνωσης της εφαρμογής. Ωστόσο, δεν θα πρέπει να είναι ούτε υπερβολικά μεγάλη, τόσο που τα δεδομένα που έχουν φτάσει πρώιμα στην κρυφή μνήμη να εκτοπιστούν από αυτήν λόγω σύγκρουσης επόμενων δεδομένων που θα μεταφερθούν στην ίδια θέση, πριν προλάβουν να χρησιμοποιηθούν από την ροή εντολών της εφαρμογής. Αναφερόμενοι σε επαναληπτικούς κώδικες που περιέχουν φωλιασμένους βρόχους, η βέλτιστη (ελάχιστη) απόσταση πρώιμης ανάκλησης, σύμφωνα με την εργασία [MLG92], (σε αριθμό επαναλήψεων) είναι: $\frac{L}{I}$

  όπου L η καθυστέρηση (latency) μεταφοράς των δεδομένων από την κύρια μνήμη σε εκείνο το επίπεδο κρυφής μνήμης που στοχεύει η πρώιμη ανάκληση

  και I ο χρόνος που χρειάζεται το συντομότερο δυνατό μονοπάτι στο εσωτερικό ενός βρόχου για να εκτελεστεί η εντολή ανάγνωσης του ζητούμενου δεδομένου.

  Παράλληλα με την απόσταση της πρώιμης ανάκλησης, θα πρέπει να ληφθεί υπόψη και ο όγκος των δεδομένων που μπορούν να ληφθούν πρώιμα στην κρυφή μνήμη χωρίς να εκτοπίσουν από την κρυφή μνήμη χρήσιμα δεδομένα (cache pollution). Ιδιαίτερα στην περίπτωση που η πρώιμη ανάκληση δεδομένων γίνεται από ένα βοηθητικό νήμα, το οποίο οφείλει να συγχρονίζεται με το κύριο νήμα που εκτελεί τους υπολογισμούς, μας ενδιαφέρει να προσδιορίσουμε το μέγιστο δυνατό όγκο δεδομένων που θα εναρμονίζεται με τη χωρητικότητα εκείνου του επιπέδου της κρυφής μνήμης στο οποίο στοχεύει η πρώιμη ανάκληση. Το να προσδιορίσουμε το μέγιστο δυνατό όριο είναι σημαντικό, γιατί με αυτόν τον τρόπο μπορούμε να ελαχιστοποιήσουμε την επικοινωνία μεταξύ κύριου και βοηθητικού νήματος.

  Η προσέγγιση που θα υιοθετήσουμε είναι η εξής: ο όγκος των πρώιμα καλούμενων δεδομένων πρέπει να κυμαίνεται μεταξύ $\frac{1}{A}$ και $\frac{1}{2}$, όπου A ο βαθμός συσχέτισης της κρυφής μνήμης. Το όριο $\frac{1}{A}$ επιλέχθηκε από τους [TWN04]. Ωστόσο, στην περίπτωση των Intel Xeon DP μηχανημάτων που θα χρησιμοποιήσουμε για την εκτέλεση πειραματικών μετρήσεων, το όριο αυτό είναι αρκετά μικρό ($A = 8$) και θα εισήγαγε σημαντικές καθυστερήσεις λόγω του συχνού συγχρονισμού που θα απαιτούνταν μεταξύ του νήματος υπολογισμών και του βοηθητικού νήματος. Αναφερόμαστε στην L2 κρυφή μνήμη και στο εξής θα στοχεύσουμε στις αστοχίες του συγκεκριμένου επιπέδου. Οι καθυστερήσεις της L1 κρυφής μνήμης μπορούν να καλυφθούν αποτελεσματικά μέσω των συνηθισμένων τεχνικών βελτιστοποίησης κώδικα, που αναφέρθηκαν στα κεφάλαια 3 και 4, της εκτός σειράς εκτέλεσης των εντολών και της πρώιμης ανάκλησης δεδομένων που υλοποιείται σε επίπεδο υλικού. Εξάλλου, το μικρό μέγεθος της L1 κρυφής μνήμης θα απαιτούσε τέτοια συχνότητα του συγχρονισμού, που το

κόστος του καθιστά απαγορευτική την υλοποίηση της πρώιμης ανάκλησης για την L1 κρυφή μνήμη μέσω της τεχνικής ταυτόχρονης πολυνημάτωσης.

Στο εξής το όριο που επιλέγουμε είναι πάντοτε μεγαλύτερο από $\frac{1}{A}$, και ποτέ δεν ξεπερνά το $\frac{1}{2}$ της κρυφής μνήμης. Με αυτόν τον τρόπο καταφέρνουμε να ρυθμίσουμε την απόσταση μεταξύ της πρώιμης ανάκλησης δεδομένων και της πραγματικής χρησιμοποίησής τους από το κύριο νήμα εκτέλεσης: Όταν το βοηθητικό νήμα που πραγματοποιεί την πρώιμη ανάκληση φέρει στην κρυφή μνήμη μία ολόκληρη ομάδα δεδομένων (κομμάτι προκαθορισμένου μεγέθους), σταματάει και περιμένει έως ότου το κύριο νήμα εκτέλεσης αρχίσει να χρησιμοποιεί τα δεδομένα της ομάδας αυτής. Τότε το βοηθητικό νήμα αρχίζει την πρώιμη ανάκληση της επόμενης ομάδας δεδομένων.

- Το δεύτερο εξίσου σημαντικό στοιχείο που πρέπει να προσέξουμε είναι ο διαμοιρασμός των πόρων του επεξεργαστή μεταξύ των νημάτων (βοηθητικού νήματος και κύριου νήματος υπολογισμών). Το βοηθητικό νήμα που εκτελεί την πρώιμη ανάκληση των εντολών έχει πολύ μικρότερες ανάγκες σε ότι αφορά τις λειτουργικές μονάδες του επεξεργαστή σε σύγκριση με το κύριο νήμα. Παρόλα αυτά, ακόμη και ένα άεργο νήμα μπορεί, καθώς δεσμεύει υποχρεωτικά τους στατικά διαμοιραζόμενους πόρους, να επιβραδύνει τη ροή εκτέλεσης των εντολών που πραγματοποιούν τους υπολογισμούς.

  Για το λόγο αυτό επιλέχθηκε η καθολική απενεργοποίηση του βοηθητικού νήματος (sleeping mode) και του δεύτερου λογικού επεξεργαστή για όσο χρόνο το βοηθητικό νήμα είναι άεργο και απλά περιμένει το κύριο νήμα να φτάσει στο σημείο όπου θα ξεκινήσει να χρησιμοποιεί τα δεδομένα που ανήκουν στην τελευταία ομάδα των πρώιμα καλούμενων δεδομένων. Στο σημείο αυτό, στέλνεται σήμα επανεκκίνησης του βοηθητικού νήματος και ξεκινάει η πρώιμη ανάκληση της επόμενης ομάδας δεδομένων. Απενεργοποιώντας εντελώς τον έναν λογικό επεξεργαστή (αυτόν που εκτελούσε το βοηθητικό νήμα) εξασφαλίζουμε ότι όλοι οι πόροι του συστήματος θα αφιερώνονται στο ένα και μοναδικό κύριο νήμα εκτέλεσης, με στόχο την επίτευξη της κατά το δυνατό βέλτιστης ροής εντολών μέσα στη σωλήνωση.

  Ασφαλώς, η ενεργοποίηση και η απενεργοποίηση του ενός λογικού επεξεργαστή έχει κάποιο, μη αμελητέο, κόστος. Το κόστος, όμως, αυτό είναι πολύ μικρότερο από την επιβράδυνση που προκύπτει όταν το άεργο βοηθητικό νήμα συνυπάρχει με το κύριο νήμα και δεσμεύει χρήσιμους πόρους του επεξεργαστή, χωρίς ουσιαστικά να τους αξιοποιεί για ένα σημαντικό χρονικό διάστημα. Επομένως, οφείλουμε να είμαστε προσεκτικοί στην εφαρμογή της μεθόδου αυτής. Στο κεφάλαιο 6, μετριούνται οι άεργοι κύκλοι εκτέλεσης που προέκυπταν σε κάθε σημείο των εφαρμογών, και εφαρμόζεται η προτεινόμενη τεχνική μόνο στα σημεία εκείνα που προκύπτει μετρήσιμο όφελος.

Ο συγχρονισμός την νημάτων

Για την ελαχιστοποίηση της επιβάρυνσης που εισάγει ο συγχρονισμός των νημάτων, χρησιμοποιήθηκαν οι εντολές `pause` και `halt`. Η πρώτη συστήνεται από την Intel [Int01], γιατί απομακρύνει τις εντολές του άεργου νήματος από τη σωλήνωση προκειμένου να μην δεσμεύουν τις λειτουργικές μονάδες του συστήματος. Δεν αποδεσμεύονται όμως οι στατικά διαμοιραζόμενοι πόροι του συστήματος, και επομένως δεν υπάρχει η δυνατότητα αξιοποίησής τους από το ενεργό νήμα. Για το λόγο αυτό χρησιμοποιείται εναλλακτικά η εντολή συστήματος `halt`, με την οποία απενεργοποιείται εντελώς το άεργο νήμα, απελευθερώνοντας όλους τους πόρους του συστήματος και εκτελώντας το ένα και μοναδικό νήμα με απενεργοποιημένη τη δυνατότητα ταυτόχρονης πολυνημάτωσης. Ασφαλώς, στην περίπτωση αυτή, η επανεκκίνηση του άεργου νήματος έχει αυξημένο κόστος. Άρα, μπορεί να εφαρμοστεί μόνο όταν ο χρόνος αναμονής είναι αρκετά μεγάλος, τόσος που το κέρδος από την απελευθέρωση όλων των πόρων του συστήματος υπερβαίνει το κόστος επανεκκίνησης του άεργου νήματος. Για την περαιτέρω ανάλυση των θεμάτων συγχρονισμού ο αναγνώστης μπορεί να βρει περισσότερες λεπτομέρειες στην εργασία [AAKK06b].

5.3 Αναζήτηση των ορίων του παραλληλισμού επιπέδου νήματος (TLP) και εντολής (ILP)

Προκειμένου να καθορίσουμε τα όρια παραλληλισμού που μπορούν να αξιοποιηθούν αποδοτικά στα πλαίσια των επεξεργαστών Xeon που είναι εξοπλισμένοι με την τεχνική υπερ-νημάτωσης, πειραματιστήκαμε με ομοιογενή νήματα εντολών. Κατασκευάσαμε ανεξάρτητα μεταξύ τους νήματα, το καθένα από τα οποία περιείχε ένα μόνο είδος από τις βασικές αριθμητικές εντολές (add,sub,mul,div), ή μία από τις εντολές ανάγνωσης/εγγραφής στη μνήμη (load, store), τόσο με ακέραιες μεταβλητές όσο και με μεταβλητές κινητής υποδιαστολής.

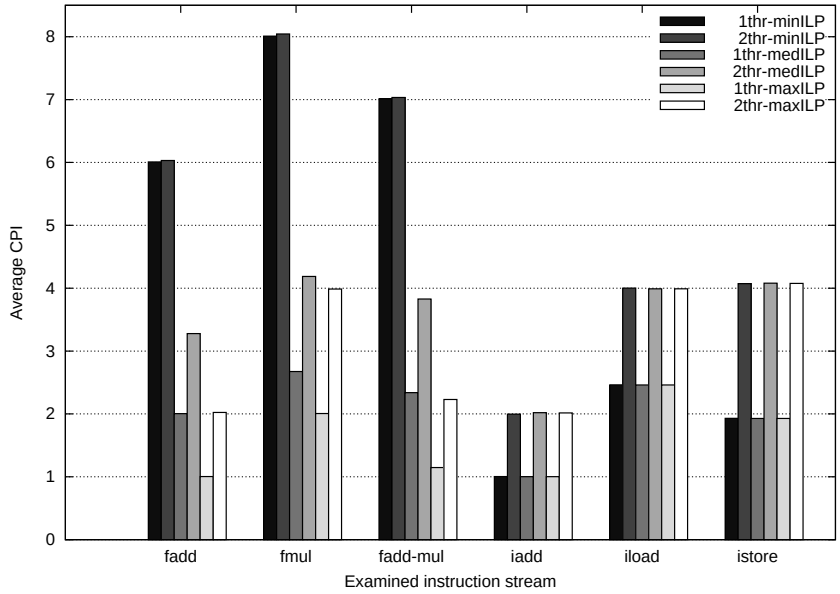
Στα πειράματα που ακολουθούν ελέγχαμε το βαθμό παραλληλισμού επιπέδου εντολής (ILP) των νημάτων μεγαλώνοντας ή μικραίνοντας τον αριθμό των προσωρινών καταχωρητών όπου η ακολουθία των εντολών που εκτελούμε αποθηκεύει τα αποτελέσματά της (T). Βασιζόμαστε, δηλαδή, στη σειριοποίηση των εντολών που επιβάλουν οι εξαρτήσεις WAW (write after write). Εξετάσαμε τρεις διαφορετικούς βαθμούς παραλληλισμού επιπέδου εντολής: τον ελάχιστο (min: $|T|$=1), όπου οι διαδοχικές εντολές είναι ευθέως εξαρτώμενες η μία από την άλλη, τον ενδιάμεσο (med: $|T|$=3), όπου υπάρχει ελευθερία τριών ανεξάρτητων διαδοχικών εντολών, και τον μέγιστο (max: $|T|$=6), όπου ο αριθμός των ανεξάρτητων καταχωρητών ισούται με το πλάτος της σωλήνωσης στο σημείο διανομής των εντολών στις λειτουργικές μονάδες.

	CPI					
	min ILP		***med ILP***		***max ILP***	
instr.	***1thr***	***2thr***	***1thr***	***2thr***	***1thr***	***2thr***
fadd	6.01	6.03	2.01	3.28	**1.00**	2.02
fmul	8.01	8.04	2.67	4.19	2.01	**3.99**
faddmul	7.01	7.03	2.34	3.83	1.15	**2.23**
fdiv	**45.06**	99.90	45.09	107.05	45.10	107.43
fload	1049.05	2012.62	1049.06	2012.43	1049.05	**2011.86**
fstore	1050.67	1982.99	1050.68	1983.07	1050.67	**1982.93**
iadd	1.01	**1.99**	1.01	2.02	1.00	2.02
imul	11.02	11.05	11.03	**11.05**	11.03	11.05
idiv	76.18	78.76	76.19	**78.71**	76.18	78.73
iload	2.46	4.00	2.46	3.99	2.46	**3.99**
istore	1.93	4.07	1.93	4.08	**1.93**	4.07

Πίνακας 5.2: Μέσο CPI για διαφορετικούς βαθμούς παραλληλισμού σε επίπεδο νήματος και εντολών για τις ευρέως χρησιμοποιούμενες ροες εντολών

5.3.1 Ταυτόχρονη εκτέλεση όμοιου τύπου ακολουθιών από εντολές

Αρχικά, εκτελούμε κάθε μία από τις ομογενείς ακολουθίες εντολών ξεχωριστά για κάθε έναν από τους τρεις διαφορετικούς βαθμούς παραλληλισμού, μόνες τους σε έναν επεξεργαστή, έτσι ώστε να απασχολούν αποκλειστικά τις λειτουργικές μονάδες του συστήματος (στήλες $1thr$ του Πίνακα 5.2). Στη δεύτερη φάση, εκτελούμε ταυτόχρονα δύο ανεξάρτητα μεταξύ τους νήματα, του ίδιου βαθμού παραλληλισμού (στήλες $2thr$ του Πίνακα 5.2). Κάθε ένα από τα νήματα αυτά προσδένεται αποκλειστικά σε έναν συγκεκριμένο λογικό επεξεργαστή, για να αποφύγουμε εσφαλμένες εκτιμήσεις λόγω απροσδιόριστων μετακινήσεων των ταυτόχρονα εκτελούμενων νημάτων μεταξύ των λογικών επεξεργαστών. Τα αποτελέσματα που προκύπτουν δίνουν μία ένδειξη ως προς τον τρόπο που αλληλεπιδρούν τα ταυτόχρονα εκτελούμενα νήματα κατά την διεκδίκηση των κοινών πόρων του συστήματος. Ενδεικτική της αποτελεσματικότητας της παραλληλοποίησης των εφαρμογών ως προς την επίδοσή τους είναι επίσης η σύγκριση της επίδοσης των εκδόσεων $1thr$, ενός συγκεκριμένου βαθμού παραλληλισμού με τις εκδόσεις, $2thr$, με τον υποδιπλάσιο βαθμό παραλληλισμού. Θεωρούμε δηλαδή ότι, κατά την παραλληλοποίηση των εφαρμογών, μειώνεται ο εγγενής βαθμός παραλληλισμού, επειδή οι ανεξάρτητες μεταξύ τους εντολές μοιράζονται στα δύο διαφορετικά νήματα εκτέλεσης. Έστω για παράδειγμα, μία συγκεκριμένη ακολουθία εντολών με μέσο όρο αριθμού κύκλων ρολογιού ανά εντολή (CPI) $C_{2thr-medILP} > 2 \times C_{1thr-maxILP}$. Στην περίπτωση αυτή, δεν θα πρέπει να περιμένουμε επιτάχυνση κατά την παραλληλοποίηση μίας εφαρμογής που περιέχει σε μεγάλο ποσοστό αυτό το είδους της εντολής. Τα δεδομένα του πίνακα 5.2



Σχήμα 5.2: Μέσο CPI για διαφορετικούς βαθμούς παραλληλισμού σε επίπεδο νήματος και εντολών για τις ευρέως χρησιμοποιούμενες ροες εντολών

εμφανίζονται με έντονους χαρακτήρες στα σημεία που σημειώνεται η καλύτερη επίδοση ανά γραμμή του πίνακα (για κάθε διαφορετικό είδος ακολουθίας εντολών). Το διάγραμμα 5.2 παρουσιάζει σχηματικά τα αποτελέσματα του πίνακα.

5.3.2 Ταυτόχρονη εκτέλεση διαφορετικού τύπου ακολουθιών από εντολές

Ο πίνακας 5.3 παρουσιάζει τα αποτελέσματα που προκύπτουν από την ταυτόχρονη εκτέλεση ακολουθιών εντολών διαφορετικού τύπου. Όπως προαναφέρθηκε, εξετάζουμε και εδώ ζεύγη νημάτων με όμοιο βαθμό παραλληλισμού. Ο συντελεστής επιβράδυνσης αντιπροσωπεύει το λόγο του CPI όταν δύο νήματα εκτελούνται ταυτόχρονα, προς το CPI του νήματος της πρώτης στήλης του πίνακα, όταν αυτό εκτελείται μόνο του. Παρατηρήθηκε ότι στις περιπτώσεις των νημάτων που περιέχουν μεταβλητές ακέραιου τύπου, ο ρυθμός εκτέλεσης εντολών δεν μεταβαλλόταν με την αλλαγή του βαθμού παραλληλισμού επιπέδου εντολής. Τα δεδομένα του πίνακα 5.3 εμφανίζονται με έντονους χαρακτήρες στα σημεία που εμφανίζεται η καλύτερη επίδοση ανά γραμμή του πίνακα (για κάθε διαφορετικό είδος ακολουθίας εντολών). Τα διαγράμματα 5.3 και 5.4 παρουσιάζουν σχηματικά τα αποτελέσματα του πίνακα.

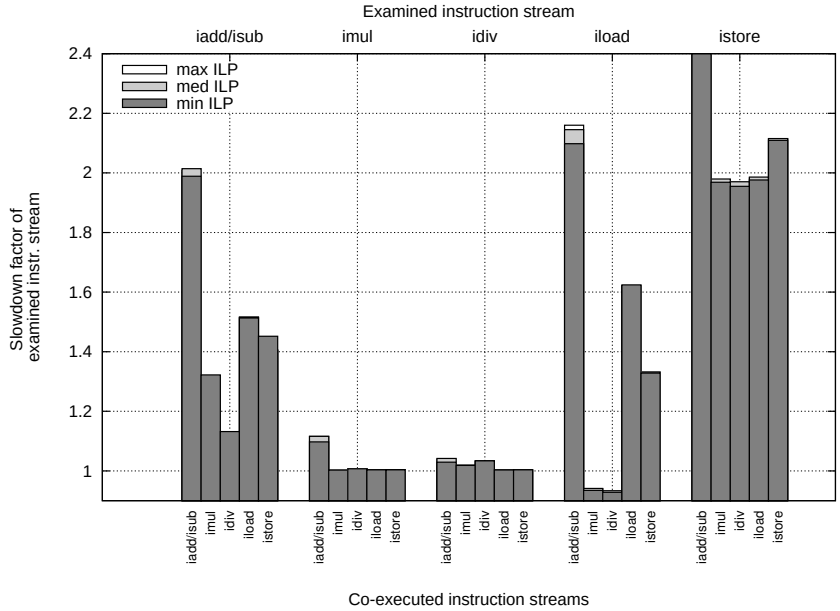
5.4 Σύνοψη

Το κεφάλαιο αυτό παρουσίασε τις δυνατότητες της τεχνικής ταυτόχρονης πολυνημάτωσης, όταν χρησιμοποιείται για τη βελτίωση της επίδοσης μίας εφαρμογής που έχει στη διάθεσή της όλους τους πόρους του συστήματος. Εξετάσαμε δύο τεχνικές: την παραλληλοποίηση των εφαρμογών σε

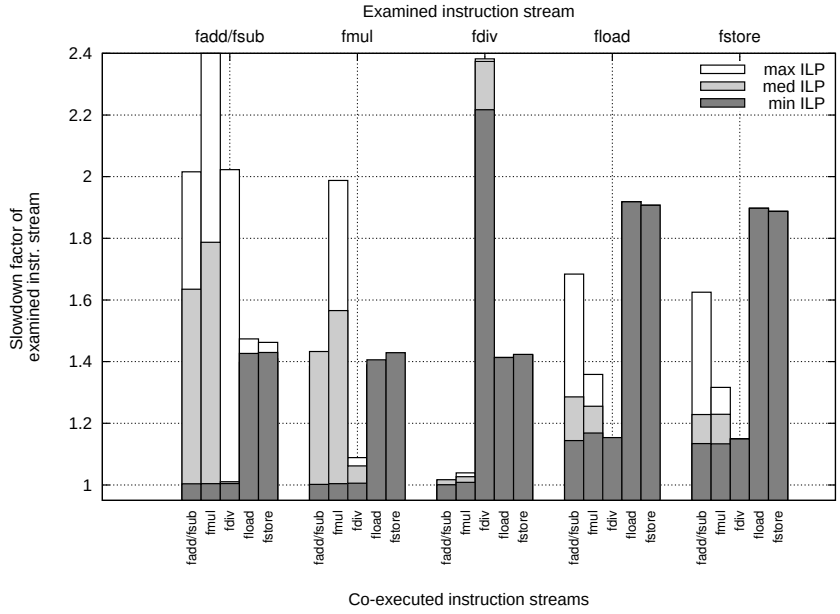
		Co-executed Instruction Streams				
	ILP	***fadd***	***fmul***	***fdiv***	***fload***	***fstore***
fadd	min:	1.004	1.004			
	med:	1.635	1.787	1.010	1.398	1.409
	max:	**2.016**	**2.801**	**2.023**	1.474	1.462
fmul	min:	1.002	1.004	1.006		
	med:	1.433	1.566	1.062	1.391	1.393
	max:	1.384	**1.988**			
fdiv	min:			**2.217**		
	med:	1.017	1.027	**2.374**	1.413	1.422
fload	min:	1.144	1.169			
	med:	1.286	1.255	1.153	**1.919**	**1.907**
	max:	1.684	1.358			
fstore	min:	1.134	1.133			
	med:	1.229	1.229	1.150	**1.897**	**1.887**
	max:	1.625	1.316			
	ILP	***iadd***	***imul***	***idiv***	***iload***	***istore***
iadd	med:	**2.014**	1.316	1.117	1.515	1.405
imul	med:	1.116	1.002	1.008	1.003	1.004
idiv	med:	1.042	1.019	1.033	1.003	1.003
iload	med:	**2.145**	0.941	0.934	1.621	1.331
istore	min:	4.072				
	med:	**4.299**	1.979	1.970	1.986	2.115
	max:	2.160	0.941	0.934	1.622	1.331

Πίνακας 5.3: Οι συντελεστές επιβράδυνσης κατά την ταυτόχρονη εκτέλεση νημάτων που επεξεργάζονται ακεραίους αριθμούς και αριθμούς κινητής υποδιαστολής

επίπεδο νήματος και την υποθετική εκτέλεση εντολών.



Σχήμα 5.3: Οι συντελεστές επιβράδυνσης κατά την ταυτόχρονη εκτέλεση νημάτων που επεξεργάζονται ακεραίους αριθμούς



Σχήμα 5.4: Οι συντελεστές επιβράδυνσης κατά την ταυτόχρονη εκτέλεση νημάτων που επεξεργάζονται αριθμούς κινητής υποδιαστολής

ΚΕΦΑΛΑΙΟ 6

Πειραματικά αποτελέσματα

Το κεφάλαιο αυτό παρουσιάζει τα αποτελέσματα των πραγματικών πειραματικών μετρήσεων και των προσομοιώσεων, όπως προέκυψαν κατά την εκτέλεση 7 διαφορετικών μετρο-προγραμμάτων, χρησιμοποιώντας τις πλατφόρμες του παραρτήματος Β. Το σύνολο των μετρήσεων χωρίζεται σε τρία μέρη: Στην ενότητα 6.1, υλοποιούνται τα διαφορετικά σχήματα μη-γραμμικών μετασχηματισμών δεδομένων και συγκρίνονται με την επίδοση του προτεινόμενου μη-γραμμικού μετασχηματισμού ομαδοποίησης δεδομένων. Στη συνέχεια, στην ενότητα 6.2, αναζητούμε το βέλτιστο μέγεθος tile για τους μη-γραμμικούς μετασχηματισμούς ομαδοποίησης δεδομένων. Τέλος, η ενότητα 6.3 συνδυάζει τους βελτισοποιημένους κώδικες, που έχουν περιγραφεί στα κεφάλαια 3 και 4, στις αρχιτεκτονικές ταυτόχρονης πολυνημάτωσης, με την πρώιμη ανάκληση δεδομένων και την παραλληλοποίηση σε επίπεδο νήματος. Στην ενότητα αυτή εξετάχονται, επιπλέον, και εφαρμογές με τυχαία μοτίβα προσπέλασης δεδομένων.

6.1 Η μη γραμμική διάταξη δεδομένων

6.1.1 Το Περιβάλλον Εκτέλεσης των Πειραματικών Μετρήσεων

Στην παράγραφο αυτήν παρουσιάζουμε τα πειραματικά αποτελέσματα που προέκυψαν από την εκτέλεση των εξής μετρο-προγραμμάτων: Matrix Multiplication, LU-decomposition, SSYR2K, SSYMM και STRMM. Οι αντίστοιχοι κώδικες φαίνονται στο Παράρτημα C. Εκτελούνται δύο τύποι πειραμάτων: πραγματικές μετρήσεις του χρόνου εκτέλεσης και προσομοιώσεις όπου χρησιμοποιείται το εργαλείο SimpleScalar (sim-out-order) [LW94]. Τα πειράματα εκτελέστηκαν σε τρεις διαφορετικές πλατφόρμες: σε ένα μηχάνημα SUN UltraSPARC II 450, σε ένα πολυεπεξεργαστικό σύστημα SGI/Cray Origin2000, και σε έναν προσωπικό υπολογιστή Athlon XP 2600+. Τα χαρακτηριστικά των μηχανημάτων αναγράφονται στον πίνακα Β.1 του Παραρτήματος Β.

Για τον UltraSPARC II και τον SGI Origin χρησιμοποιήσαμε τον cc compiler, πρώτα χωρίς optimization flags (cc -xO0), προκειμένου να μελετήσουμε την καθαρή επίδραση των διαφορετικών

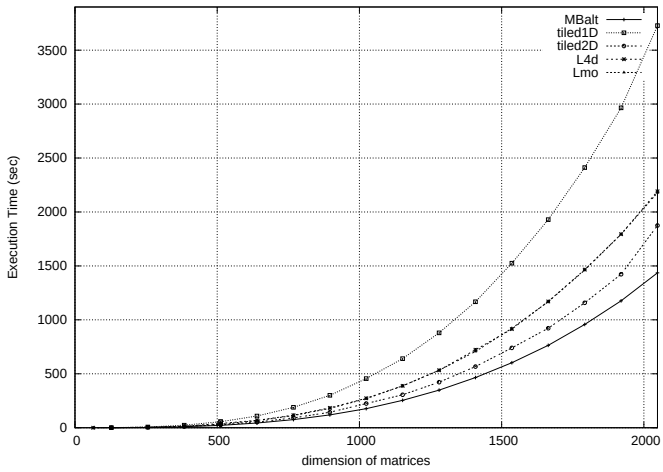
διατάξεων δεδομένων. Προσπαθήσαμε να αποκομίσουμε μία καθαρή αίσθηση σχετικά με την επιτάχυνση που μπορούν να επιφέρουν οι μη-γραμμικοί μετασχηματισμοί ομαδοποίησης δεδομένων, ανεξάρτητα της επίδρασης των βελτιστοποιήσεων του compiler. Στη συνέχεια, χρησιμοποιήσαμε το υψηλότερο επίπεδο βελτιστοποίησης (-fast -xtarget=native), το οποίο συμπεριλαμβάνει ευθυγραμμίσεις δεδομένων στη μνήμη, ξεδίπλωμα βρόχων, software pipeline και βελτιστοποιήσεις στην εκτέλεση πράξεων με αριθμούς κινητής υποδιαστολής (floating point). Τα πειράματα πραγματοποιήθηκαν για πολλά διαφορετικά μεγέθη πινάκων. Η διάσταση (N) των πινάκων κυμαινόταν στο διάστημα 16 έως 2048 στοιχεία, και τα μεγέθη των tiles (*step*) στο διάστημα 16 έως N στοιχεία. Για να αποκαλύψουμε τη δυναμική του προτεινόμενου αλγόριθμου βελτιστοποίησης πειραματιστήκαμε με όγκους δεδομένων που χωρούν ή δεν χωρούν ολόκληροι στα διάφορα επίπεδα κρυφής μνήμης.

Στον Athlon XP χρησιμοποιήθηκε ο μεταγλωττιστής gcc , πρώτα χωρίς καθόλου βελτιστοποίηση (gcc -O0) και στη συνέχεια, στο υψηλότερο επίπεδο βελτιστοποίησης (-O3).

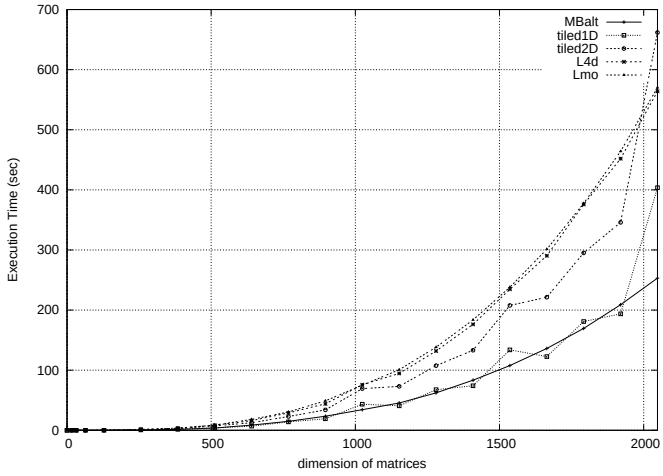
Υλοποιήσαμε 5 διαφορετικές εκδόσεις από τους κώδικες: τη μη-γραμμική διάταξη ομαδοποίησης δεδομένων πινάκων χρησιμοποιώντας τη θεωρία των μασκών για τη δεικτοδότησή τους (**B**locked **a**rray **L**ayou**t**s using **M**asks: MBaLt), τη διάταξη L_{MO} (Lmo) και τη διάταξη L_{4D} (L4d) κατά την προσέγγιση του Chatterjee, και τη μέθοδο του Kandemir [KRC99] με γραμμική διάταξη δεδομένων, χρησιμοποιώντας διδιάστατους (tiled2D) και μονοδιάστατους πίνακες (tiled1D). Οι μετρούμενοι χρόνοι δεν συμπεριλαμβάνουν το χρόνο μετασχηματισμού των πινάκων από τις γραμμικές διατάξεις στις μη-γραμμικές διατάξεις ομαδοποίησης, επειδή για κάθε πίνακα επιλέγεται ένα και μοναδικό στιγμιότυπο διάταξης, το οποίο χρησιμοποιείται καθ'όλη τη διάρκεια της εκτέλεσης του βελτιστοποιημένου κώδικα, από όλες τις αναφορές στον συγκεκριμένο πίνακα. Επομένως, δεν χρειάζεται μετασχηματισμός και επαναφορά του πίνακα στην αρχική του μορφή, το οποίο θα προκαλούσε καθυστέρηση.

6.1.2 Μετρήσεις του Χρόνου Εκτέλεσης

Στον κώδικα του πολλαπλασιασμού πινάκων (Matrix Multiplication), συγκρίναμε την μέθοδό μας (MBaLt), με εκείνη που προτάθηκε στη βιβλιογραφία στην εργασία [KRC99]. Σε όλες τις περιπτώσεις, τα διαγράμματα παρουσιάζουν τις τιμές του χρόνου εκτέλεσης που σημειώνονται στο βέλτιστο (κατά περίπτωση) μέγεθος tile. Στο σχήμα 6.1 ο χρόνος εκτέλεσης του MBaLt εμφανίζει επιτάχυνση σχεδόν 25% σε σύγκριση με το χρόνο εκτέλεσης του προτεινόμενου από τον Kandemir κώδικα, για διδιάστατους πίνακες (tiled2D). Στην πραγματικότητα, δεδομένου ότι οι σύγχρονοι μεταγλωττιστές δεν υποστηρίζουν μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων και κατά συνέπεια, στην υλοποίηση της μεθόδου μας χρησιμοποιούμε μονοδιάστατους πίνακες, μία δίκαιη σύγκριση θα πρέπει να αφορούσε μονοδιάστατους πίνακες, δηλαδή την έκδοση tiled1D. Στην περίπτωση αυτή, η έκδοση MBaLt είναι καλύτερη κατά 60%. Η υλοποίηση του Chatterjee έχει ακόμα χειρότερη επίδοση από την έκδοση tiled2D, παρόλο που η διάταξη δεδομένων L_{4D} είναι στην πράξη ίδια με αυτήν της έκδοσης MBaLt. Η διαφορά έγκειται στο ότι στην L_{4D} χρησιμοποιούνται



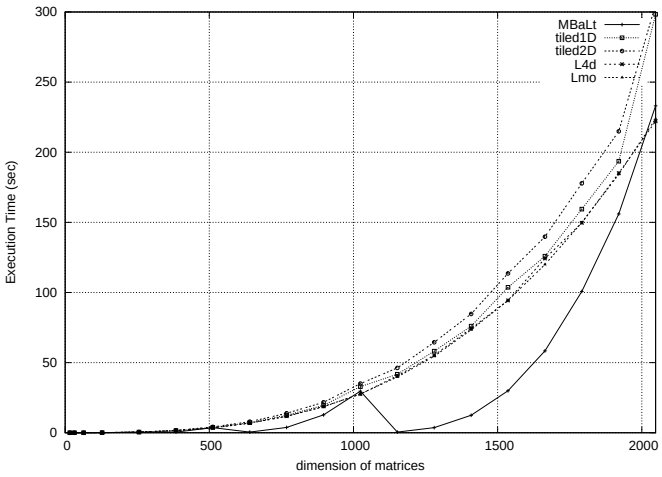
Σχήμα 6.1: Συνολικά πειραματικά αποτελέσματα του πολλαπλασιασμού πινάκων (-xO0, UltraSPARC)



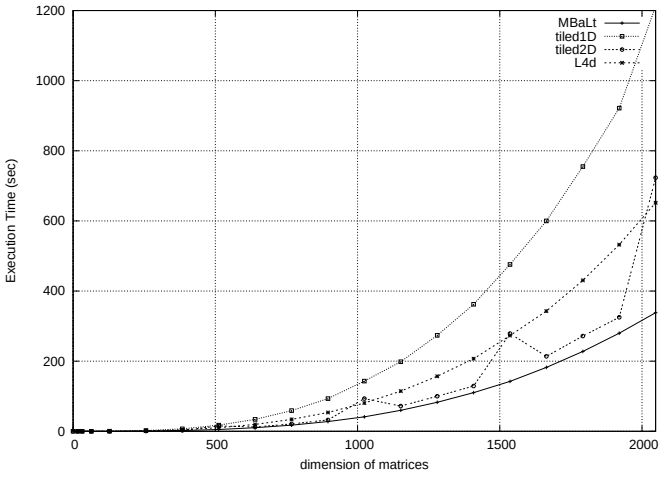
Σχήμα 6.2: Συνολικά πειραματικά αποτελέσματα του πολλαπλασιασμού πινάκων (-fast, UltraSPARC)

4-διάστατοι πίνακες, αντί των μονοδιάστατων της έκδοσης MBaLt. Η μείωση της επίδοσης στους 4-διάστατους πίνακες οφείλεται στο περισσότερο πολύπλοκο σχήμα τοποθέτησης και εύρεσης των στοιχείων στη μνήμη.

Χρησιμοποιώντας το επίπεδο βελτιστοποίησης -fast (σχήμα 6.2), ο κώδικας MBaLt έχει σταθερή επίδοση. Η γραφική παράσταση του σχήματος 6.2 αυξάνει ομαλά με την αύξηση του μεγέθους των πινάκων, καθώς έχουν εξαλειφθεί οι αστοχίες σύγκρουσης που θα μπορούσαν να προκαλέσουν απρόβλεπτες καθυστερήσεις. Η έκδοση tiled (ιδιαίτερα σε ότι αφορά τους κώδικες LU-decomposition και SSYR2K), είναι επιρρεπής σε τέτοιου είδους μη-προβλέψιμες αστοχίες, καθώς τα αντίστοιχα διαγράμματα επιδεικνύουν απότομες διακυμάνσεις του χρόνου εκτέλεσης για συγκεκριμένα μεγέθη πινάκων. Ένα επιπρόσθετο συμπέρασμα που προκύπτει συγκρίνοντας τα σχήματα 6.1 και 6.2, είναι ότι, όσο απλούστερος είναι ο αρχικός κώδικας, τόσο καλύτερη βελτιστοποίηση μπορεί να επιτύχει το επίπεδο -fast του μεταγλωττιστή. Επομένως, στη βελτιστοποίηση



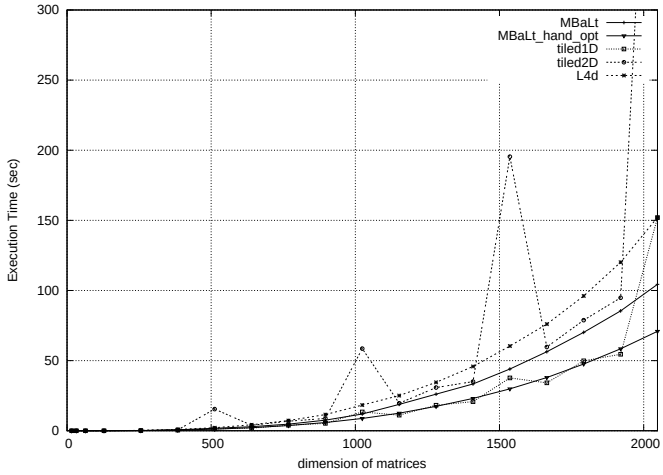
Σχήμα 6.3: Συνολικά πειραματικά αποτελέσματα του πολλαπλασιασμού πινάκων (-fast, SGI Origin)



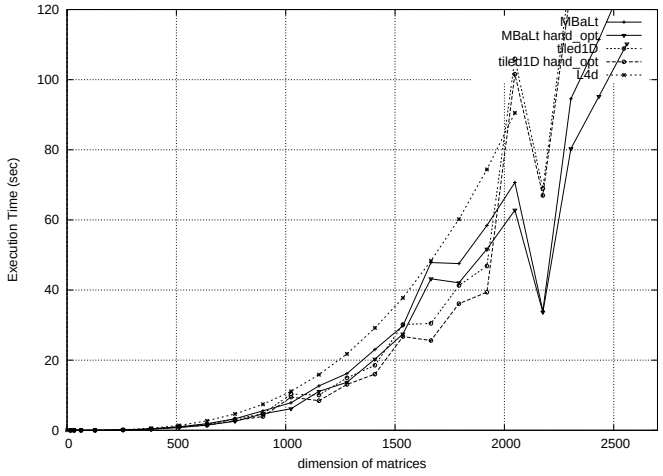
Σχήμα 6.4: Συνολικά πειραματικά αποτελέσματα του LU-decomposition (-xO0, UltraSPARC)

πολύπλοκων κωδίκων οι τυποποιημένες βελτιστοποιήσεις των μεταγλωττιστών δεν επαρκούν. Η εφαρμογή μετασχηματισμών βελτιστοποίησης με το χέρι (π.χ. ξεδίπλωμα βρόχων και πρώιμη ανάκληση δεδομένων) σε συνδυασμό με το επίπεδο βελτιστοποίησης -fast, αποδεικνύεται ως ο καλύτερος συνδυασμός για την επίτευξη της μέγιστης δυνατής βελτιστοποίησης της επίδοσης (βλέπε την επίδοση της έκδοσης MBaLt με εφαρμογή βελτιστοποιήσεων με το χέρι: MBaLt-hand_opt του σχήματος 6.6).

Παρόλο που ο κώδικας MBalt είναι μεγαλύτερος από την άποψη του αριθμού εντολών που περιλαμβάνει, η εκτέλεσή του διαρκεί λιγότερο χρόνο, λόγω των δυαδικών πράξεων που χρησιμοποιούνται για τον υπολογισμό των διευθύνσεων ανάγνωσης από τη μνήμη. Επιπρόσθετα, το παραγέμισμα των πινάκων δεν επηρεάζει αρνητικά την επίδοση των μετρο-προγραμμάτων, καθώς ο χρόνος εκτέλεσης αυξάνεται ομαλά με την αύξηση της πραγματικής διάστασης των πινάκων (δηλαδή, ο χρόνος εκτέλεσης δεν έχει απότομες αυξομειώσεις ανάλογα με το ποσό των παραγε-



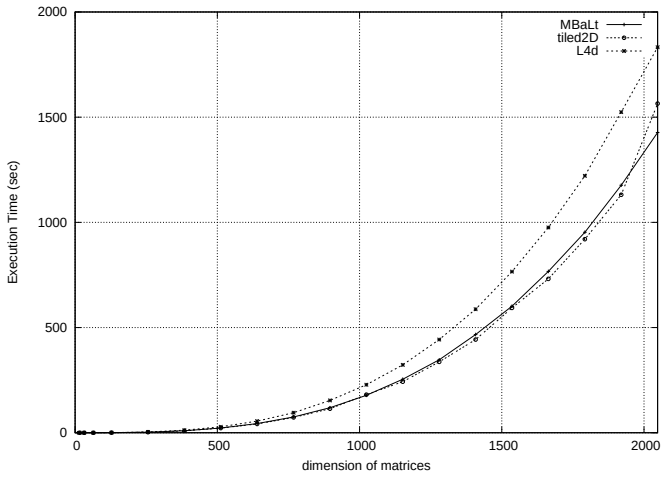
Σχήμα 6.5: Συνολικά πειραματικά αποτελέσματα του LU-decomposition (-fast, UltraSPARC)



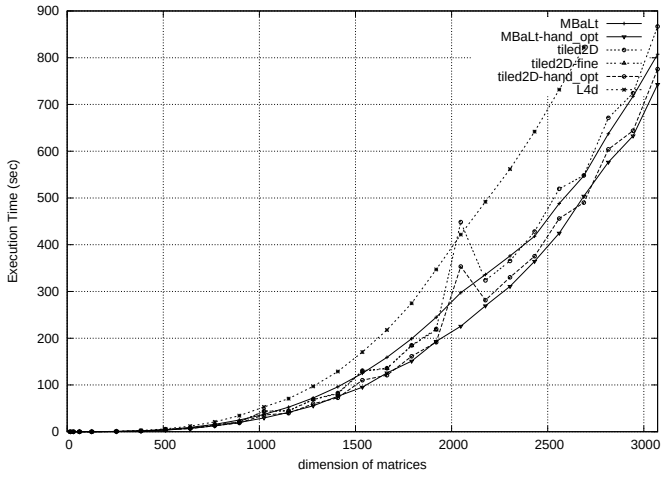
Σχήμα 6.6: Συνολικά πειραματικά αποτελέσματα του LU-decomposition για μεγάλα μεγέθη πινάκων και για κώδικες βελτιστοποιημένους με το χέρι (-fast, SGI Origin)

μισμάτων που εισάγονται). Μελετώντας με προσοχή τον κώδικα assembly που προκύπτει από τη μεταγλώττιση των αρχικών προγραμμάτων, οι μονοδιάστατοι πίνακες υλοποιούνται με απλούστερο άρα και ταχύτερο τρόπο, σε σχέση με τους πολυδιάστατους. Οι μονοδιάστατοι πίνακες χρειάζονται λιγότερες αναφορές στη μνήμη για την εύρεση των ζητούμενων στοιχείων τους, καθώς δεν απαιτούν ενδιάμεσες αναφορές στη μνήμη για την επίλυση των πολυδιάστατων. Από την άλλη πλευρά, η εύρεση της θέσης αποθήκευσης ενός στοιχείου του πίνακα στις μη-γραμμικές διατάξεις δεδομένων απαιτούν έναν σημαντικό αριθμό υπολογισμών. Αυτό που επιτύχαμε με την τεχνική της γρήγορης δεικτοδότησης είναι η χρησιμοποίηση μονοδιάστατων πινάκων αποφεύγοντας τη χρονοβόρα διαδικασία υπολογισμού διευθύνσεων, κάνοντας ιδιαίτερα ελκυστική την υιοθέτηση μη-γραμμικών διατάξεων δεδομένων στα προγράμματα.

Τα παραπάνω αποτελέσματα επιβεβαιώνονται από την εκτέλεση των μετρο-προγραμμάτων LU-decomposition, SSYR2K, SSYMM και STRMM (σχήματα 6.4 - 6.14). Επιπρόσθετα, παρατηρού-



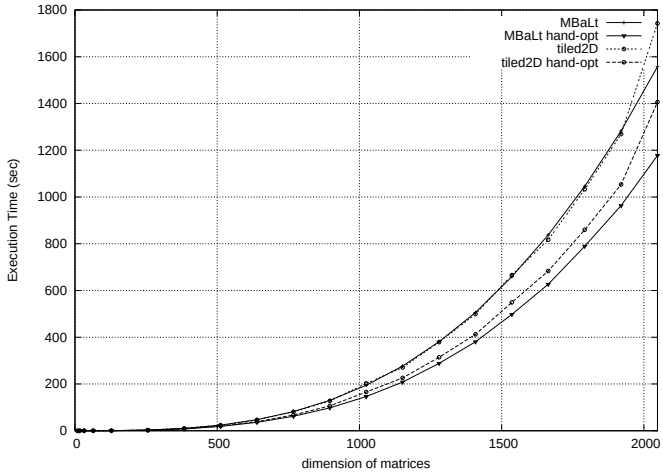
Σχήμα 6.7: Συνολικά πειραματικά αποτελέσματα του SSYR2K (-xO0, UltraSPARC)



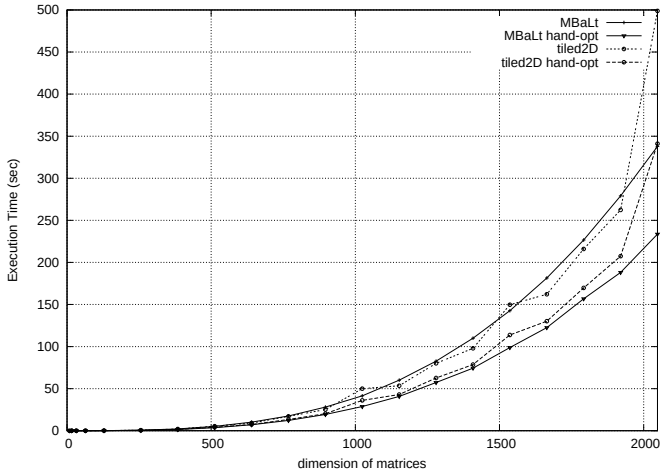
Σχήμα 6.8: Συνολικά πειραματικά αποτελέσματα του SSYR2K (-fast, UltraSPARC)

με ότι κατά την εκτέλεση του κώδικα LU-decomposition, η χρήση των μη γραμμικών διατάξεων ομαδοποίησης δεδομένων σε συνδυασμό με τη γρήγορη δεικτοδότηση που προτείνουμε, όχι μόνο παρέχει μία 15% μείωση του χρόνου εκτέλεσης (όταν δεν έχει εφαρμοστεί κανενός είδους βελτιστοποίηση κώδικα από τον μεταγλωττιστή), αλλά επιπλέον, εξομαλύνονται οι απότομες αυξομειώσεις του χρόνου εκτέλεσης που προκαλούνται στον απλό μετασχηματισμένο κώδικα με tiling, λόγω των αστοχιών σύγκρουσης. Η βελτίωση του χρόνου εκτέλεσης είναι ακόμα καλύτερη (μπορεί να φτάσει το 30%) με τη χρήση του επιπέδου βελτιστοποίησης -fast. Παρατηρώντας ότι η έκδοση tiled1D (στους κώδικες πολλαπλασιασμός πινάκων και LU-decomposition) έχει καλύτερη επίδοση συγκρινόμενη με την έκδοση MBaLt, όταν εφαρμόζεται η βελτιστοποίηση -fast, πειραματιστήκαμε με μεγαλύτερους πίνακες από 2048×2048. Το σχήμα 6.6 δείχνει ότι η έκδοση MBaLt είναι καλύτερη.

Στο μετρο-πρόγραμμα SSYR2K (στα σχήματα 6.7 και 6.8), πραγματοποιήσαμε μία αναλυτική αναζήτηση του βέλτιστου χρόνου εκτέλεσης για κάθε μέγεθος πίνακα, μεταβάλλοντας το μέγεθος



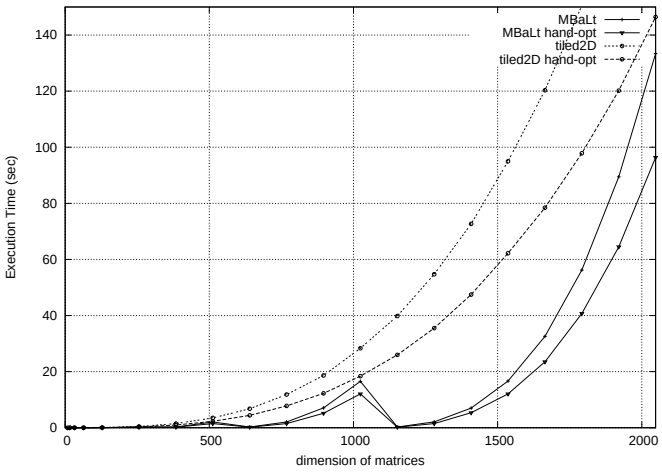
Σχήμα 6.9: Συνολικά πειραματικά αποτελέσματα του SSYMM (-xO0, UltraSPARC)



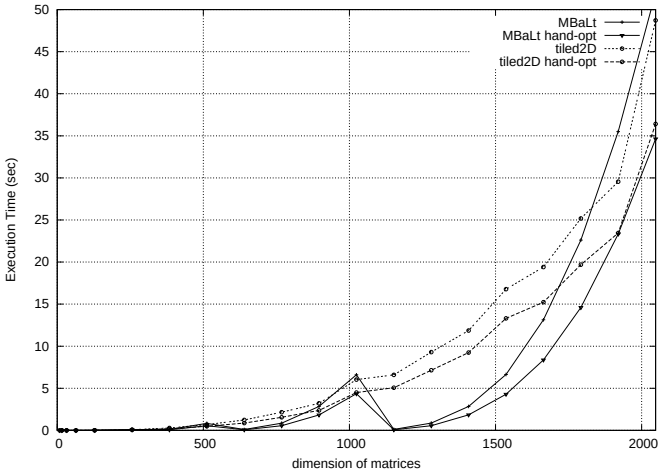
Σχήμα 6.10: Συνολικά πειραματικά αποτελέσματα του SSYMM (-fast, UltraSPARC)

των tiles εντός ενός πυκνού πλήθους τιμών (όχι πλέον μόνο μεταξύ δυνάμεων του 2), στις γραμμικές διατάξεις δεδομένων, στην έκδοση tiled2D-fine. Σημειώνουμε ότι σε όλα τα προηγούμενα πειράματα χρησιμοποιούσαμε μεγέθη tiles ίσα με κάποια δύναμη του 2, προκειμένου να συμφωνούν με τις απαιτήσεις της υλοποίησης MBaLt. Από τα σχήματα 6.7 και 6.8 γίνεται φανερό ότι η επίδοση των εκδόσεων tiled και tiled-fine είναι στην πράξη πανομοιότυπες. Από τους χρόνους εκτέλεσης αποδεικνύεται ότι δεν υπήρξε καμία επιβάρυνση του χρόνου εκτέλεσης σε όλες τις χρησιμοποιούμενες εκδόσεις tiled λόγω του ότι τα επιλεγόμενα μεγέθη tiles ήταν δυνάμεις του 2.

Τέλος, σε ότι αφορά το βέλτιστο μέγεθος tile, οι εκδόσεις MBaLt έχουν καλύτερη επίδοση για μικρά μεγέθη tile, τέτοια που να επιτρέπουν στο σύνολο δεδομένων που περιέχουν να χωρούν στην L1 κρυφή μνήμη. Τα βέλτιστα μεγέθη tile έχουν σταθερή τιμή ανεξάρτητα από το μέγεθος των πινάκων. Από την άλλη πλευρά, οι εκδόσεις tiled (tiled1D και tiled2D) επιτυγχάνουν ελαχιστοποίηση του χρόνου εκτέλεσης σε πολλά διαφορετικά μεγέθη tiles, καθώς μεταβάλλεται το μέγεθος των πινάκων. Αναλυτική διερεύνηση του θέματος αυτού γίνεται στο κεφάλαιο 4.



Σχήμα 6.11: Συνολικά πειραματικά αποτελέσματα του SSYMM (-O0, Athlon XP)

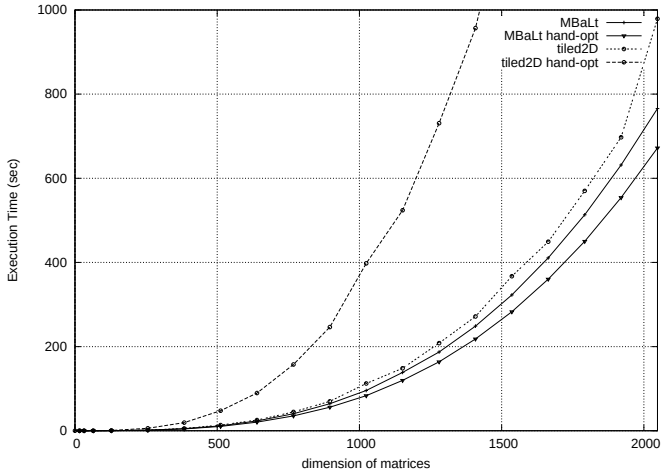


Σχήμα 6.12: Συνολικά πειραματικά αποτελέσματα του SSYMM (-O3, Athlon XP)

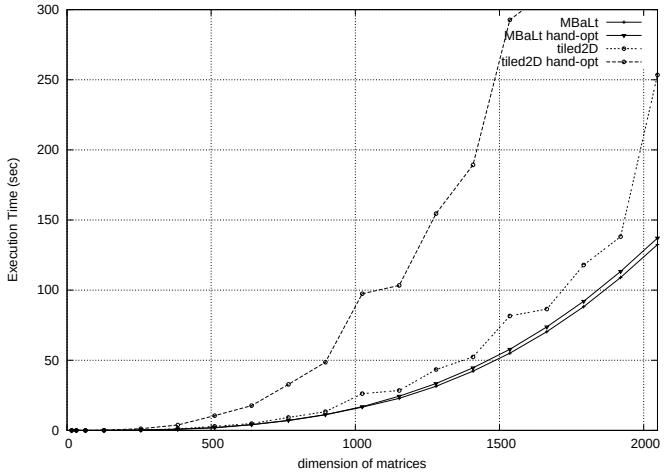
6.1.3 Αποτελέσματα της προσομοίωσης

Προκειμένου να επαληθεύσουμε τα συμπεράσματα που προέκυψαν από τις μετρήσεις του χρόνου εκτέλεσης, εκτελέσαμε τις εκδόσεις των προγραμμάτων του πολλαπλασιασμού πινάκων (MBaLt, tiled2D, tiled1D) και του LU-decomposition (non-tiled, tiled, MBaLt) με το εργαλείο προσομοίωσης SimpleScalar 2.0, τόσο για τα χαρακτηριστικά του UltraSPARC μηχανήματος όσο και του SGI Origin. Μετρήσαμε τις αστοχίες δεδομένων της L1 κρυφής μνήμης (dl1), τις αστοχίες της ενοποιημένης L2 μνήμης (ul2), καθώς και τις αστοχίες του TLB δεδομένων (dtlb), για διάφορες τιμές της διάστασης των πινάκων N στην περιοχή 16 έως 1024 στοιχεία και για μέγεθος tile *step* από 8 έως N στοιχεία, στο τυπικό επίπεδο βελτιστοποίησης του μεταγλωττιστή. Η κλίμακα του κατακόρυφου άξονα των διαγραμμάτων της παραγράφου αυτής είναι λογαριθμική.

Τα σχήματα 6.15 έως 6.17 παρουσιάζουν τον αριθμό των αστοχιών για διάφορα μεγέθη πινάκων. Σε όλες τις περιπτώσεις, η έκδοση MBaLt δίνει τον μικρότερο αριθμό αστοχιών. Ο αριθμός των προσπελάσεων στη μνήμη μετρήθηκαν κατά την προσομοίωση της εκτέλεσης των προγραμ-



Σχήμα 6.13: Συνολικά πειραματικά αποτελέσματα του STRMM (-xO0, UltraSPARC)

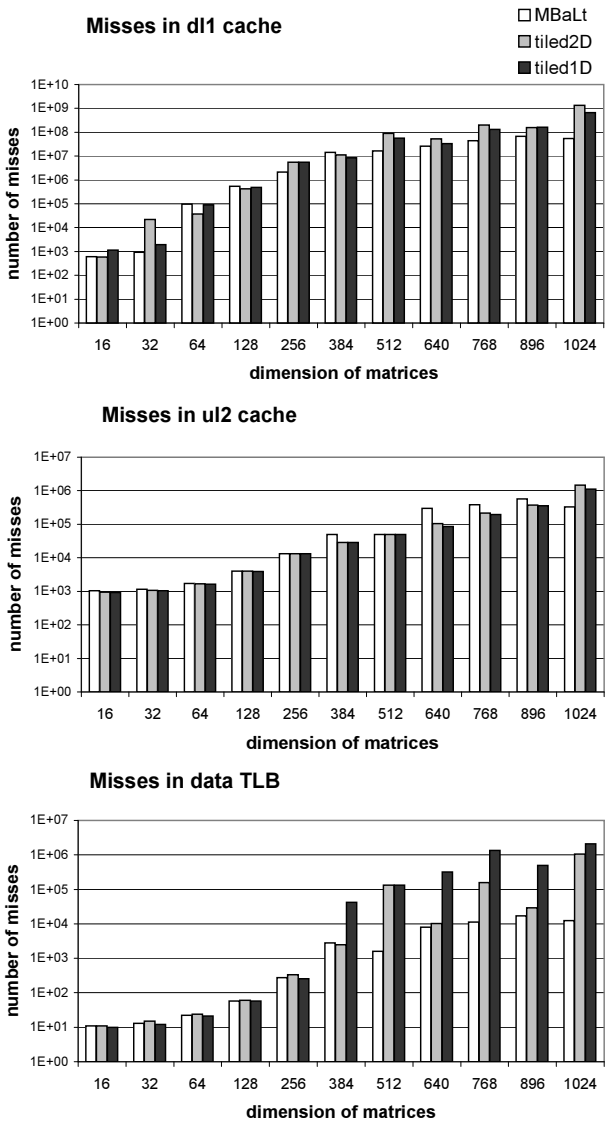


Σχήμα 6.14: Συνολικά πειραματικά αποτελέσματα του STRMM (-fast, UltraSPARC)

μάτων, αλλά δεν προσφέρουν επιπλέον συμπεράσματα. Έτσι, τα διαγράμματα περιορίζονται στην παρουσίαση μόνο του αριθμού των αστοχιών.

6.2 Επιλογή του βέλτιστου μεγέθους Tile

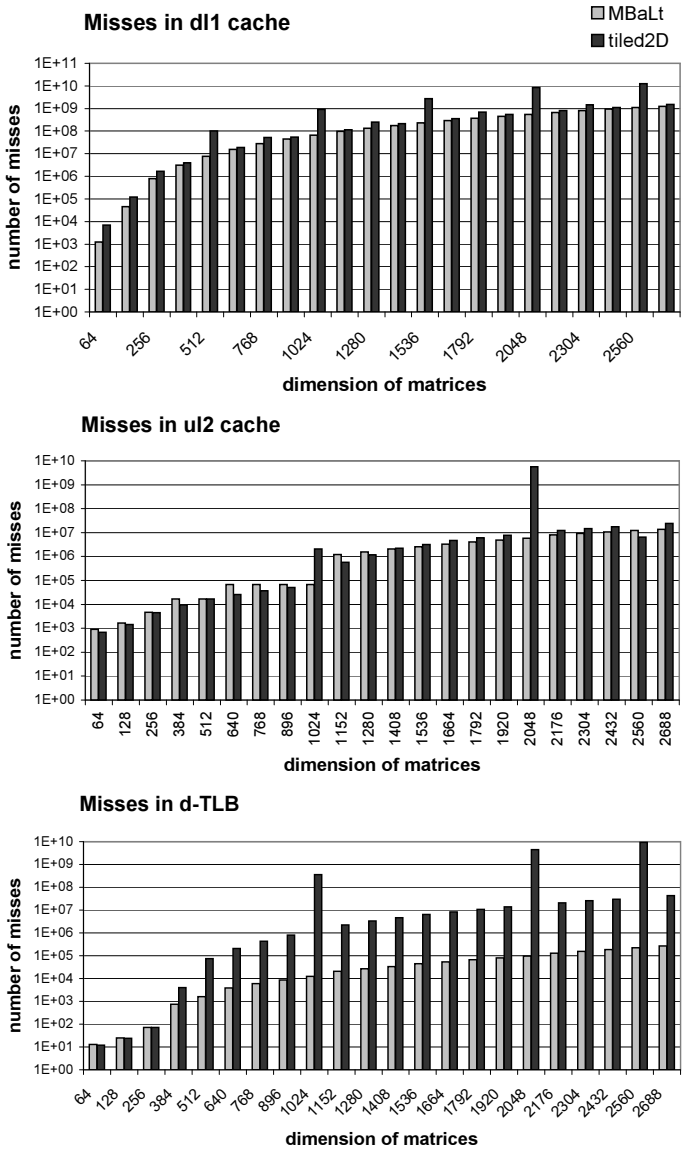
Στην ενότητα αυτή παρουσιάζουμε τα πειραματικά αποτελέσματα που προκύπτουν από την εκτέλεση των μετρο-προγραμμάτων Matrix Multiplication, LU-decomposition, SSYR2K, SSYMM and STRMM. Οι αντίστοιχοι κώδικες φαίνονται στο Παράρτημα C. Όπως και στο προηγούμενο κεφάλαιο, εκτελούνται δύο τύποι πειραμάτων: πραγματικές μετρήσεις του χρόνου εκτέλεσης και προσομοιώσεις όπου χρησιμοποιείται το εργαλείο SimpleScalar (sim-out-order) [LW94]. Τα πειράματα εκτελέστηκαν σε τρεις διαφορετικές πλατφόρμες: σε ένα μηχάνημα SUN UltraSPARC II 450, σε ένα συμμετρικό πολυεπεξεργαστικό σύστημα Pentium III (SMP), και σε έναν προσωπικό υπολογιστή Athlon XP 2600+. Τα χαρακτηριστικά των μηχανημάτων αναγράφονται στους



Σχήμα 6.15: Αριθμός αστοχιών στην L1 κρυφή μνήμη δεδομένων, την ενοποιημένη L2 κρυφή μνήμη και στο TLB δεδομένων για τον πολλαπλασιασμό πινάκων (UltraSPARC)

πίνακες B.1 και B.2 του Παραρτήματος B.

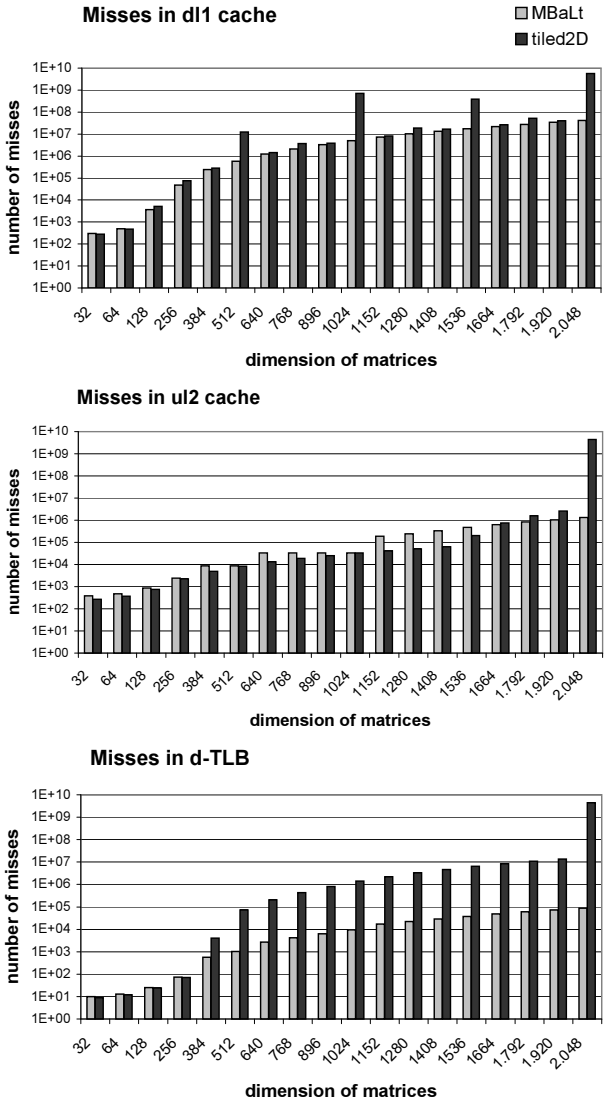
Για τον UltraSPARC II χρησιμοποιήσαμε τον cc compiler, στο υψηλότερο επίπεδο βελτιστοποίησης (-fast -xtarget=native). Τα πειράματα πραγματοποιήθηκαν για πολλά διαφορετικά μεγέθη πινάκων. Στα Pentium III και Athlon XP χρησιμοποιήθηκε ο μεταγλωττιστής gcc στο υψηλότερο επίπεδο βελτιστοποίησης (-O3). Η διάσταση (N) των πινάκων κυμαινόταν στο διάστημα 64 έως 2048 στοιχεία, και τα μεγέθη των tiles (*step*) στο διάστημα 16 έως N στοιχεία. Για να αποκαλύψουμε τη δυναμική του προτεινόμενου αλγόριθμου βελτιστοποίησης πειραματιστήκαμε με όγκους δεδομένων που χωρούν ή δεν χωρούν ολόκληροι στα διάφορα επίπεδα κρυφής μνήμης.



Σχήμα 6.16: Αστοχίες στην L1 κρυφή μνήμη δεδομένων, την ενοποιημένη L2 κρυφή μνήμη και στο TLB δεδομένων για το LU-decomposition (UltraSPARC)

6.2.1 Πειραματική Επαλήθευση των Θεωρητικών Αποτελεσμάτων

Καταρχήν, εκτελέσαμε το μετρο-πρόγραμμα του πολλαπλασιασμού πινάκων, στο οποίο χρησιμοποιούσαμε μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων σε συνδυασμό με την τεχνική της γρήγορης δεικτοδότησης. Τα πειραματικά αποτελέσματα πιστοποιούν τη συμπεριφορά της κρυφής μνήμης και του TLB που υπολογίσαμε θεωρητικά στην παράγραφο 4.1: οι αστοχίες της L1 κρυφής μνήμης φαίνεται ότι αποτελούν τον κυρίαρχο παράγοντα στην συνολική επίδοση του συστήματος (η βέλτιστη επίδοση επιτυγχάνεται όταν $T = 64$). Στο σχήμα 6.18 απεικονίζεται μόνο ένας περιορισμένος αριθμός τιμών, έτσι ώστε η παρουσίασή τους να γίνει ευανάγνωστη.

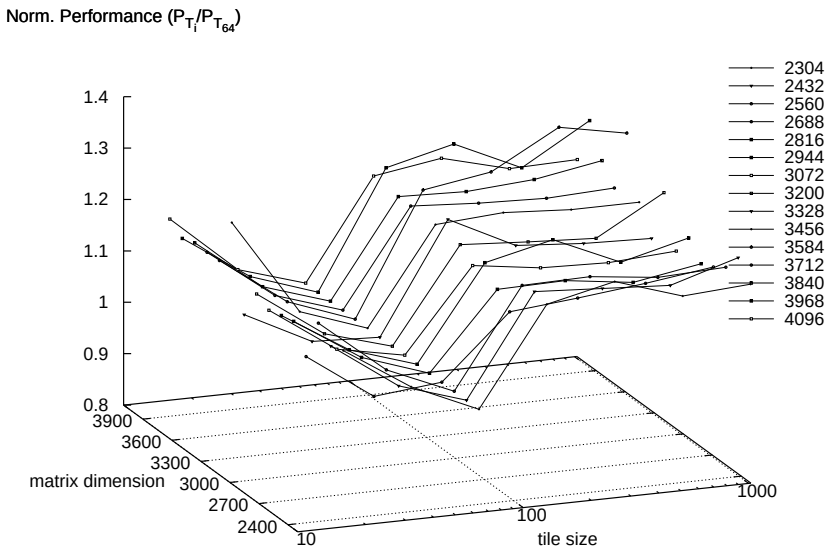


Σχήμα 6.17: Αστοχίες στην L1 κρυφή μνήμη δεδομένων, την ενοποιημένη L2 κρυφή μνήμη και στο TLB δεδομένων για το LU-decomposition (SGI Origin)

6.2.2 Η επίδοση της έκδοσης MBaLt: Επιλογή μεγέθους tile

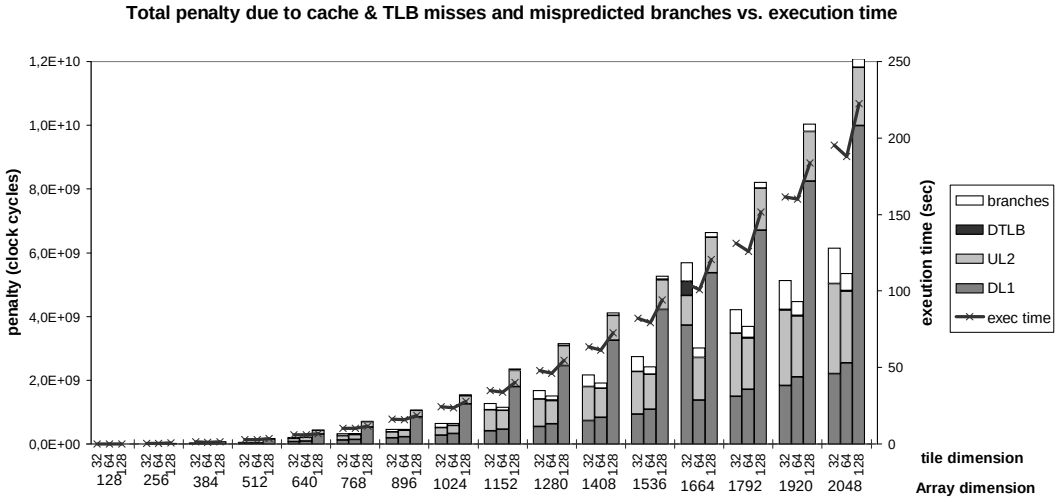
Τα πειραματικά αποτελέσματα του χρόνου εκτέλεσης και της προσομοίωσης της βελτιστοποιημένης έκδοσης του Πολλαπλασιασμού Πινάκων φαίνεται στο σχήμα 6.19. Οι μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων όταν συνδυαστούν κατάλληλα με τις τεχνικές βελτιστοποίησης των φωλιασμένων βρόχων, δίνουν βέλτιστο μέγεθος tile $T = \sqrt{C_{L1}}$. Στην έκδοση MBaLt, μπορούμε να χρησιμοποιήσουμε μόνο τετραγωνικά tiles των οποίων η διάσταση είναι ίση με κάποια δύναμη του δύο. Επομένως, στην αρχιτεκτονική του μηχανήματος UltraSPARC II, το βέλτιστο μέγεθος tile είναι $T = 64$.

Σημειώνουμε ότι τα αποτελέσματα της προσομοίωσης επαληθεύουν πλήρως τους θεωρητικούς υπολογισμούς: Στο σημείο της βέλτιστης επίδοσης, οι αστοχίες της L1 κρυφής μνήμης αυξάνουν ελαφρά σε σχέση με την ελάχιστη τιμή τους (που σημειώνεται στο αμέσως μικρότερο μέγεθος



Σχήμα 6.18: Ο χρόνος εκτέλεσης του μετρο-προγράμματος του Πολλαπλασιασμού Πινάκων για διάφορα μεγέθη πινάκων και tile (UltraSPARC, -fast)

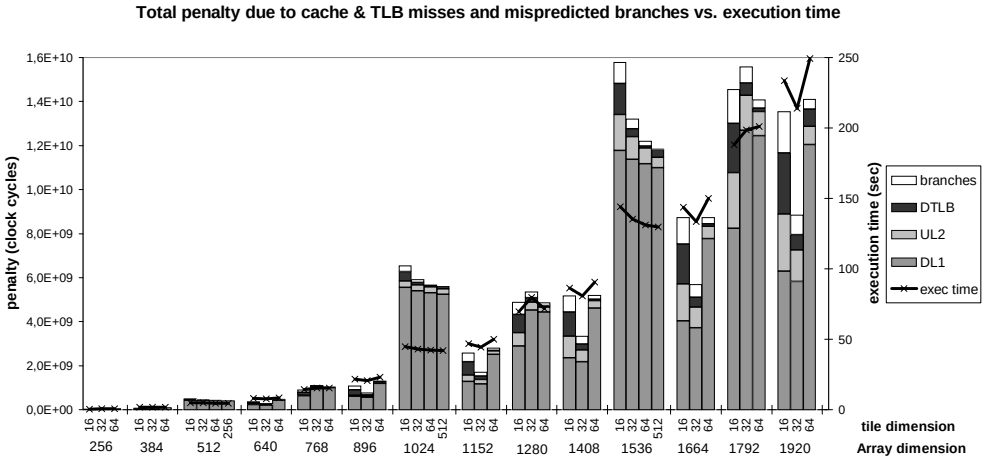
tile). Από την άλλη πλευρά, η μείωση των αστοχιών της L2 κρυφής μνήμης, του TLB και της λάθος πρόβλεψης της απόφασης των εντολών άλματος αντισταθμίζει την μικρή αυτή αύξηση των αστοχιών της L1 κρυφής μνήμης. Σαν συνέπεια, στο σημείο αυτό επιτυγχάνεται η ελαχιστοποίηση του συνολικού αριθμού κύκλων ρολογιού, που σπαταλώνται σε καθυστερήσεις και επομένως η ελαχιστοποίηση του χρόνου εκτέλεσης του προγράμματος. Όταν το μέγεθος του tile ξεπεράσει τη χωρητικότητα της L1 κρυφής μνήμης, η L1 κρυφή μνήμη υπερχειλίζει και οι αστοχίες που σημειώνονται στο επίπεδο αυτό αυξάνουν δραματικά. Το γεγονός αυτό, αποκλείει τα μεγαλύτερα tiles από μία καλή επίδοση.



Σχήμα 6.19: Συνολική επιβάρυνση της επίδοσης λόγω των αστοχιών της L1 κρυφής μνήμης, της L2 κρυφής μνήμης και του TLB δεδομένων στον Πολλαπλασιασμό Πινάκων όπου έχει χρησιμοποιηθεί μη-γραμμική διάταξη ομαδοποίησης δεδομένων και η τεχνική της γρήγορης δεικτοδότησης των πινάκων. Απεικονίζεται επίσης ο πραγματικός χρόνος του προγράμματος (UltraSPARC)

6.2.3 Η έκδοση MBaLt έναντι των γραμμικών διατάξεων δεδομένων

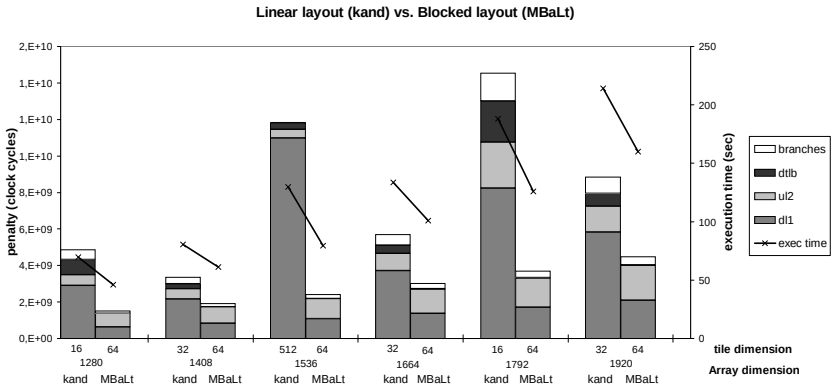
Στην παράγραφο αυτή συγκρίνουμε την επίδοση της έκδοσης MBaLt (Blocked array layouts with use of binary masks) με την πλήρως βελτιστοποιημένη έκδοση των γραμμικών διατάξεων δεδομένων ([KRC99]). Διερευνούμε τους ακριβής λόγους στους οποίους οφείλεται η καλύτερη επίδοση των μη-γραμμικών διατάξεων ομαδοποίησης δεδομένων.



Σχήμα 6.20: Η συνολική επιβάρυνση του χρόνου εκτέλεσης λόγω καθυστερήσεων και ο πραγματικός χρόνος εκτέλεσης για τον Πολλαπλασιασμό Πινάκων (γραμμική διάταξη δεδομένων - UltraSPARC)

Όπως μπορούμε ξεκάθαρα να διακρίνουμε στο σχήμα 6.20, οι γραμμικές διατάξεις δεδομένων έχουν ασταθή (μη-ομαλή) επίδοση για τα διαφορετικά μεγέθη πινάκων. Σημειώνουμε ότι επιλέγουμε ως διάταξη αποθήκευσης των στοιχείων των πινάκων στη μνήμη, είτε κατά γραμμές, είτε κατά στήλες, έτσι ώστε να ταιριάζει κατά το δυνατόν με τη σειρά προσπέλασης των στοιχείων από τον κώδικα του προγράμματος. Η αστάθεια αυτή αποδίδεται στις δύσκολα προβλέψιμες αστοχίες σύγκρουσης. Οι τεχνικές, όπως το παραγέμισμα των πινάκων ή η αντιγραφή, που εφαρμόζονται στην περίπτωση των γραμμικών διατάξεων δεδομένων για να αποφευχθούν τα παθολογικά μεγέθη πινάκων, πρέπει να εφαρμόζονται με προσοχή και ύστερα από επισταμένη διερεύνηση. Είναι, ωστόσο, περιττές στην περίπτωση των μη-γραμμικών διατάξεων ομαδοποίησης δεδομένων. Επομένως, η βελτιστοποίηση των φωλιασμένων βρόχων είναι μία περισσότερο τυποποιημένη διαδικασία όταν χρησιμοποιούνται μη-γραμμικές διατάξεις ομαδοποίησης δεδομένων.

Το σχήμα 6.21 δείχνει αναλυτικά τα σημεία που προκαλούν μειωμένη επίδοση στην περίπτωση των γραμμικών διατάξεων δεδομένων σε σύγκριση με τις μη-γραμμικές διατάξεις. Το συγκεκριμένο σχήμα παρουσιάζει ένα περιορισμένο εύρος τιμών, προκειμένου να γίνουν περισσότερο ευδιάκριτες οι διαφορές. Και στις δύο περιπτώσεις (MBaLt & kand), επιλέγουμε και παρουσιάζουμε τα αποτελέσματα της περίπτωσης όπου σημειώνεται η καλύτερη επίδοση χρόνου. Γίνεται φανερό, ότι οι αστοχίες της L1 κρυφής μνήμης δεδομένων στις γραμμικές διατάξεις δεδομένων μπορούν να γίνουν ακόμα και μία τάξη μεγέθους περισσότερες σε σχέση με την έκδοση MBaLt, εξαιτίας των αστοχιών σύγκρουσης. Στην περίπτωση των γραμμικών διατάξεων δεδομένων είναι



Σχήμα 6.21: Η σχετική επίδοση των δύο διαφορετικών διατάξεων δεδομένων (UltraSPARC)

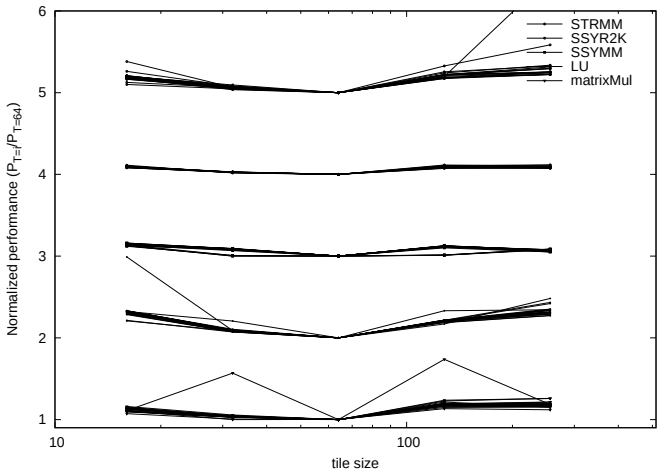
ιδιαίτερα επίπονη διαδικασία η πρόβλεψη αυτής της κατηγορίας των αστοχιών και επομένως, δύσκολα μπορούν να αποφευχθούν. Οι αστοχίες του TLB δεδομένων μειώνονται επίσης σημαντικά στην έκδοση MBaLt (στην πραγματικότητα φτάνουν την ελάχιστη δυνατή τιμή τους, δεδομένων των αστοχιών χωρητικότητας): Η ακολουθία προσπέλασης των δεδομένων από τον κώδικα του προγράμματος ταιριάζει ακριβώς με την διάταξη αποθήκευσής τους στη μνήμη. Εξάλλου τα στοιχεία που ανήκουν στο ίδιο tile αποθηκεύονται σε συνεχόμενες θέσεις μνήμης και καλύπτουν $\frac{T^2}{P}$ διαδοχικές σελίδες. Στην χειρότερη περίπτωση, μπορεί να καλύπτουν οριακά $\frac{T^2}{P}+1$ σελίδες. Οι αστοχίες στην L2 κρυφή μνήμη αυξάνονται ελαφρώς στην έκδοση MBaLt. Ωστόσο η βελτίωση της επίδοσης λόγω του μειωμένου αριθμού αστοχιών στην L1 κρυφή μνήμη και στο TLB δεν μπορεί να αναιρεθεί από τη μικρή αύξησή τους στην L2 κρυφή μνήμη. Τέλος, ο αριθμός των λάθος προβλέψεων στην απόφαση των εντολών άλματος είναι μειωμένος στην έκδοση MBaLt, γιατί οι γραμμικές διατάξεις δεδομένων επιτυγχάνουν ελαχιστοποίηση του συνολικού χρόνου εκτέλεσης για μικρά μεγέθη tiles, γιατί σε αυτό το σημείο μπορούν να μειωθούν οι αστοχίες σύγκρουσης.

6.2.4 Περισσότερα Πειραματικά Αποτελέσματα

Για την επαλήθευση των θεωρητικών αποτελεσμάτων, εκτελέσαμε πειράματα με τους κώδικες 5 διαφορετικών μετρο-προγραμμάτων σε τρεις διαφορετικές πλατφόρμες. Τα μετρο-προγράμματα είναι τα Matrix Multiplication, SSYR2K, SSYMM και STRMM από τις ρουτίνες BLAS3 και το LU-decomposition.

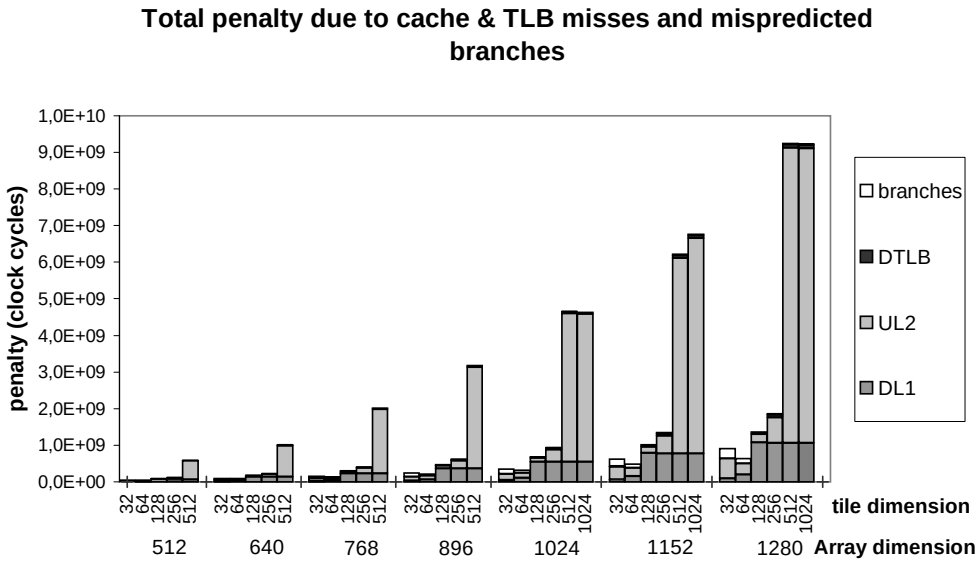
Στην αρχιτεκτονική του UltraSPARC II, όπως πιστοποιείται από το σχήμα 6.22, και στις πέντε περιπτώσεις μετρο-προγραμμάτων, ο χρόνος εκτέλεσης ελαχιστοποιείται όταν $T = 64$. Πρόκειται για το μέγεθος tile που χωράει ακριβώς στην ευθείας απεικόνισης L1 κρυφή μνήμη δεδομένων (πρόκειται για το σημείο που μόλις μπορεί ακόμα να αποφευχθεί η υπερχείλιση της L1 κρυφής μνήμης δεδομένων). Το σχήμα 6.22 απεικονίζει την κανονικοποιημένη επίδοση των 5 μετρο-προγραμμάτων, τα οποία έχουν μετατοπιστεί στον κατακόρυφο άξονα έτσι ώστε να μην υπερκαλύπτονται τα διαγράμματα των διαφορετικών προγραμμάτων. Δηλαδή, $\frac{\text{επίδοση όταν } T_N=(\text{οποιαδήποτε τιμή})}{\text{επίδοση όταν } T_N=64}+$

x, όπου $x = \{0, 1, 2, 3, 4\}$, $T_N \in [8, N]$ και $N \in [64, 4096]$. Στο σχήμα 6.22, απεικονίζονται οι τιμές του $T_N \in [16, 256]$, με έμφαση στην περιοχή της βέλτιστης επίδοσης.



Σχήμα 6.22: Κανονικοποιημένη επίδοση των 5 μετρο-προγραμμάτων για διάφορα μεγέθη πινάκων και tiles (UltraSPARC)

Η πλατφόρμα του Athlon XP και του Pentium III έχουν διαφορετικά αρχιτεκτονικά χαρακτηριστικά. Ο βαθμός συσχέτισης (που είναι μεγαλύτερος του 1) των κρυφών μνημών του Athlon XP μπορεί να απαλείψει τις αστοχίες σύγκρουσης. Κατά συνέπεια, ο αριθμός των αστοχιών της L1 κρυφής μνήμης μειώνεται σημαντικά. Η L2 κρυφή μνήμη είναι επίσης συσχετιστική. Από την άλλη πλευρά, έχει πολύ μικρότερη συνολική χωρητικότητα σε σχέση με το αντίστοιχο επίπεδο του UltraSPARC, και επομένως προκαλούνται σε αυτήν πολλές περισσότερες αστοχίες. Σε κάθε περίπτωση το ελάχιστο άθροισμα των αστοχιών στα επίπεδα L1+L2 της κρυφής μνήμης δεν μπορεί να επιτευχθεί πουθενά αλλού παρά μόνο για μέγεθος tile τέτοιο, που τα δεδομένα που περιέχει να χωράνε στην L1 κρυφή μνήμη. Η κατάσταση αυτή έχει ως αποτέλεσμα και πάλι την κυριαρχία των αστοχιών της L1 κρυφής μνήμης στην συνολική επίδοση (βλ. σχήμα 6.23). Μόνο για μία μειονότητα μεγεθών πινάκων (σχήμα 6.24) η βέλτιστη επίδοση επιτυγχάνεται για μέγεθος tiles ίσο με την αμέσως μεγαλύτερη τιμή από την C_{L1}. Στη γενική περίπτωση το βέλτιστο μέγεθος tile είναι εκείνο που ικανοποιεί την απαίτηση: $T^2 = C_{L1}$. Τα διαγράμματα επίδοσης απεικονίζονται στα σχήματα 6.24 και 6.25. Σημειώνουμε ότι στα πειράματα της πλατφόρμας Pentium III (σχήμα 6.24) είναι $x = \{0, 4, 8, 12, 16\}$ ενώ στην πλατφόρμα του Athlon XP (σχήμα 6.25), είναι $x = \{0, 2, 4, 6, 8\}$. Αυτές είναι οι ελάχιστες δυνατές τιμές που σε κάθε περίπτωση αποτρέπουν την επικάλυψη των διαγραμμάτων από διαφορετικά μετρο-προγράμματα. Και στα δύο σχήματα οι απεικονιζόμενες τιμές είναι για $T_N \in [32, 512]$, επικεντρώνοντας το ενδιαφέρον μας στην περιοχή της βέλτιστης επίδοσης.



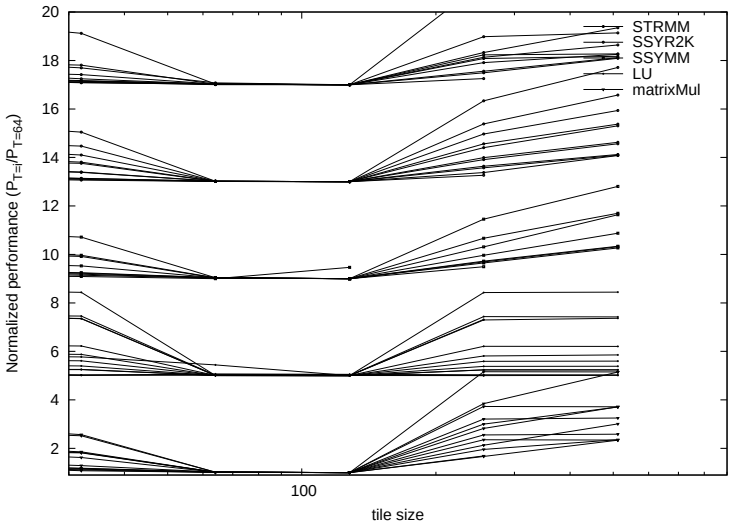
Σχήμα 6.23: Συνολική Επίδοση του Πολλαπλασιασμού Πινάκων (Pentium III)

6.3 Η τεχνική της ταυτόχρονης πολυνημάτωσης

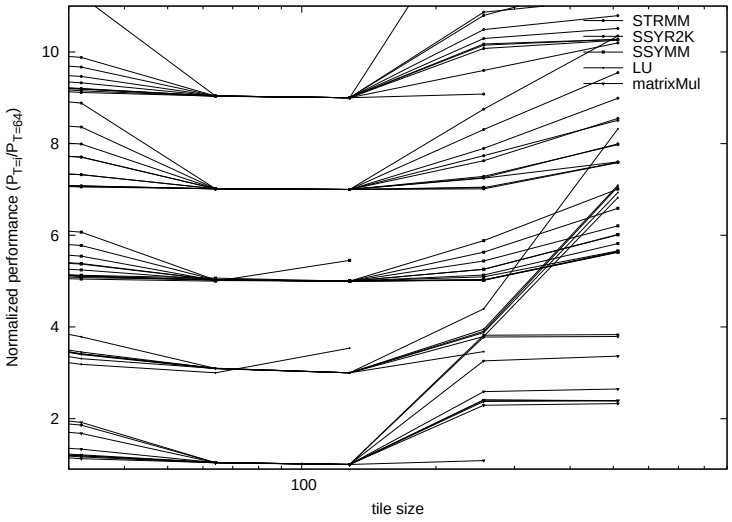
Η παράγραφος που ακολουθεί παρουσιάζει τα αποτελέσματα που προέκυψαν από την εφαρμογή των τεχνικών που περιγράφηκαν στο κεφάλαιο 5 σε εφαρμογές που επεξεργάζονται μεγάλο όγκο δεδομένων. Πρόκειται επομένως για εφαρμογές που περιέχουν μεγάλη ποσότητα υπολογισμών και ταυτόχρονα συχνές αναγνώσεις δεδομένων από τη μνήμη. Σε κάθε μία από τις εφαρμογές αυτές καταμετρήσαμε τα εξής στοιχεία:

- **Αστοχίες της L2 κρυφής μνήμης**: Πρόκειται για τις εντολές φόρτωσης και τις αιτήσεις για αποκλειστική χρήση δεδομένων (Request for Ownership - RFO), όπως εμφανίζονται στο διάδρομο επικοινωνίας με τη μνήμη. Στις εκδόσεις προγραμμάτων που εφαρμόζεται η τεχνική της ταυτόχρονης πολυνημάτωσης καταμετράται το άθροισμα των αστοχιών L2 κρυφής μνήμης για τα δύο νήματα. Μόνο στην περίπτωση που το ένα από τα δύο νήματα εκτελεί απλώς πρώιμη ανάκληση δεδομένων, καταμετρούμε τις αστοχίες μόνο του νήματος που πραγματοποιεί τους υπολογισμούς.

- **Άεργοι κύκλοι των λειτουργικών μονάδων**: Στην περίπτωση αυτή καταμετράμε τους κύκλους ρολογιού κατά τους οποίους η ακολουθία των εντολών ενός νήματος βρίσκεται κολλημένη στον allocator (διανεμητής), περιμένοντας να αδειάσει κάποια θέση μέσα στον προσωρινό καταχωρητή αποθήκευσης εντολών (store buffer). Ο allocator, η δομική μονάδα που διαβάζει τις μετροεντολές από την ουρά που βρίσκονται αρχικά τοποθετημένες και τις διανέμει στους προσωρινούς καταχωρητές των λειτουργικών μονάδων. Η μετρική αυτή δίνει μία ένδειξη των συγκρούσεων που πραγματοποιούνται μεταξύ των νημάτων (αλλά και εντός του ίδιου νήματος) κατά τη διεκδίκηση των λειτουργικών μονάδων του συστήματος.



Σχήμα 6.24: Pentium III - Κανονικοποιημένη επίδοση των πέντε μετρο-προγραμμάτων για διάφορα μεγέθη πινάκων και tiles



Σχήμα 6.25: Athlon XP - Κανονικοποιημένη επίδοση των πέντε μετρο-προγραμμάτων για διάφορα μεγέθη πινάκων και tiles

- **Πραγματικός αριθμός μετροεντολών που ολοκληρώνονται**: Μη πραγματικές (bogus) είναι οι εντολές που ακυρώνονται λόγω λανθασμένης πρόβλεψης. Σε όλες τις περιπτώσεις, καταμετρούμε το άθροισμα των εντολών που ολοκληρώνονται για τα δύο ταυτόχρονα εκτελούμενα νήματα.

Στην παράγραφο αυτή παρουσιάζουμε τα αποτελέσματα των μετρήσεων με τους αριθμητικούς κώδικες του Πολλαπλασιασμού Πινάκων (MM) και LU decomposition, καθώς και δύο από τους μετροκώδικες NAS, τους CG [1] και BT [2]. Στους κώδικες MM και LU, παρουσιάζουμε τα απο-

[1] Ο κώδικας CG λύνει ένα μη δομημένο αραιό γραμμικό σύστημα με τη μέθοδο των συζυγών παραγώγων (conjugate gradient). Πρόκειται για μετρο-κώδικα που προσπελάζει μεγάλο όγκο δεδομένων με ακανόνιστα μοτίβα εκτέλεσης.

[2] Ο κώδικας BT επιλύει τριγωνικά συστήματα ομάδων 5×5, χρησιμοποιώντας τη μέθοδο πεπερασμένων διαφορών (finite differences). Πρόκειται για μετροκώδικα που προσπελάζει μεγάλο όγκο δεδομένων, αλλά έχει μεγαλύτερη

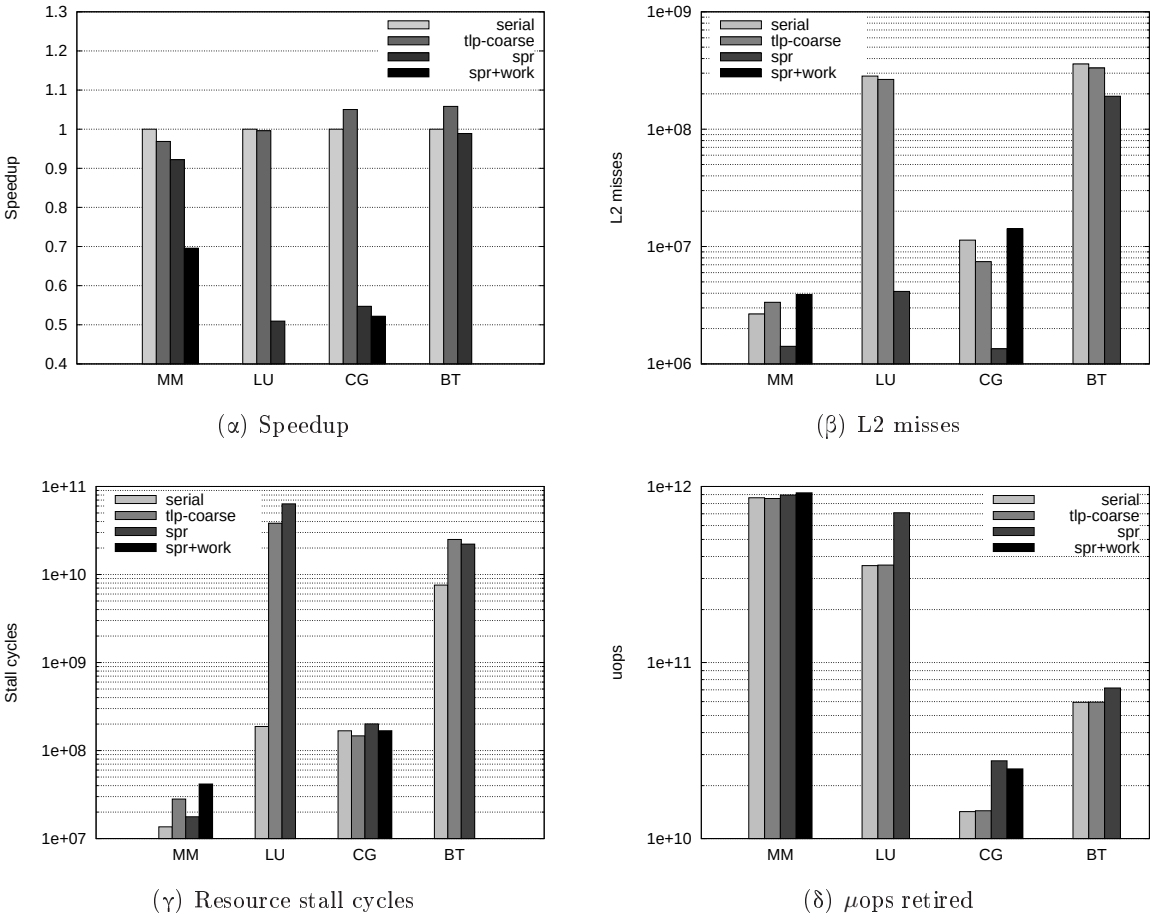
τελέσματα για μέγεθος πινάκων 4096×4096 (οι μετρήσεις για τα διαφορετικά μεγέθη πινάκων δεν προσέφεραν επιπρόσθετα ή διαφοροποιημένα συμπεράσματα, αφού η σχετική επίδοση των μεθόδων που υλοποιήθηκαν έδωσαν παρόμοια σχετικά αποτελέσματα). Για τον ίδιο λόγο, στους κώδικες CG και BT παρουσιάζουμε τα αποτελέσματα για προβλήματα τάξης Class A. Όπως και στις προηγούμενες μετρήσεις, για τους κώδικες MM και LU εφαρμόσαμε τον μετασχηματισμό υπερκόμβων με μέγεθος ομάδας δεδομένων (tile) τέτοιο που να χωράει πλήρως στην L1 κρυφή μνήμης, μιας και αυτό δίνει και στην περίπτωση των επεξεργαστών Xeon τη βέλτιστη επίδοση. Επιπλέον, εφαρμόσαμε μη γραμμικό μετασχηματισμό δεδομένων με χρήση των δυαδικών μασκών που περιγράφτηκαν στο κεφάλαιο 3 και επιπλέον ξεδίπλωμα βρόχων. Η υλοποίηση των CG και BT βασίστηκε στις εκδόσεις OpenMP C της σουίτας NPB 2.3, σύμφωνα με το πρόγραμμα Omni OpenMP Compiler Project [Ope03], αλλά μετασχηματίστηκαν κατάλληλα για την υλοποίηση πολυνηματικών εκδόσεων. Το βασικό χαρακτηριστικό και στους δύο κώδικες που επιλέχθηκαν (CG και BT) είναι ότι περιέχουν ακανόνιστα μοτίβα προσπέλασης μνήμης.

Οι εκδόσεις που υλοποιούν την παραλληλοποίηση των εφαρμογών με χρήση πολυνημάτωσης χωρίζουν το σύνολο των υπολογισμών με μεγάλα κομμάτια (coarse-grain: **tlp-coarse**): στα διαφορετικά νήματα ανατίθεται ο υπολογισμός διαδοχικών ομάδων δεδομένων (tiles), εξασφαλίζοντας έτσι ότι δεν θα υπάρχουν συγκρούσεις δεδομένων στην κρυφή μνήμη. Πειραματιστήκαμε με κάποιους από τους κώδικες χωρίζοντας τους υπολογισμούς σε μικρά τμήματα (fine-grain), αλλά τα αποτελέσματα ήταν ιδιαίτερα αποθαρρυντικά. Πραγματοποιούμε έναν στατικό κατά το δυνατό ισο-μοίρασμα του συνολικού όγκου υπολογισμών στα δύο νήματα. Ειδικά στην περίπτωση του πολλαπλασιασμού πινάκων, στα διαφορετικά νήματα ανατίθενται διαφορετικά κομμάτια του πίνακα C (πίνακας καταχώρησης αποτελεσμάτων) και επομένως αποφεύγουμε κάθε είδους συγχρονισμό μεταξύ των νημάτων, στις παράλληλες εκδόσεις.

Οι εκδόσεις που υλοποιούν την υποθετική εκτέλεση εντολών, χρησιμοποιούν τις μεθόδους συγχρονισμού που περιγράφτηκαν στο κεφάλαιο 5 και αναλυτικότερα στην εργασία [AAKK06b]. Στην έκδοση **spr** το σύνολο των υπολογισμών υλοποιείται από το ένα νήμα, ενώ το δεύτερο πραγματοποιεί πρώιμη ανάκληση των δεδομένων που θα χρειαστεί το κύριο νήμα κατά το επόμενο κομμάτι υπολογισμών. Τέλος, υλοποιήσαμε μία ακόμα υβριδική έκδοση (**spr+work**), όπου το σύνολο των υπολογισμών χωρίζεται σε μικρότερα τμήματα. Και τα δύο νήματα εκτελούν υπολογισμούς στο ίδιο κομμάτι δεδομένων (διαχωρισμός υπολογισμών σε μικρά κομμάτια: fine-grain), ενώ ταυτόχρονα το ένα από τα δύο νήματα καλεί πρώιμα τα δεδομένα που θα χρειαστούν στο επόμενο κομμάτι υπολογισμών. Ο fine-grain διαχωρισμός των υπολογισμών επιβάλλεται από το γεγονός ότι η πρώιμη ανάκληση που εκτελεί το ένα νήμα πρέπει να φροντίσει για τα δεδομένα που θα χρειαστούν και τα δύο νήματα στην επόμενη φάση υπολογισμών. Η υβριδική αυτή έκδοση υλοποιήθηκε μόνο στους μετροκώδικες MM και CG.

Το διάγραμμα 6.26 παρουσιάζει τα αποτελέσματα που προέκυψαν από την εκτέλεση των παραπάνω εκδόσεων μετροκωδίκων. Όπως γίνεται φανερό, η τεχνική της ταυτόχρονης πολυνημάτωσης

τοπικότητα δεδομένων σε σχέση με τον CG.



(α) Speedup

(β) L2 misses

(γ) Resource stall cycles

(δ) μops retired

Σχήμα 6.26: Πειραματικά Αποτελέσματα κατά την εφαρμογή της Ταυτόχρονης Πολυνημάτωσης

όπως υλοποιείται στην πλατφόρμα των επεξεργαστών Intel Xeon όπου ενσωματώνεται η τεχνολογία υπερ-νημάτωσης δεν μπορεί να επιταχύνει το χρόνο εκτέλεσης των εφαρμογών σε σχέση με τη σειριακή έκδοσή τους. Μόνο στην περίπτωση των κωδικών NAS έχουμε επιτάχυνση της τάξης $5\% - 6\%$, για τις εκδόσεις TLP. Στις εκδόσεις όπου υλοποιείται η υποθετική εκτέλεση εντολών, επιτυγχάνεται σημαντικότατη μείωση του αριθμού των αστοχιών της L2 κρυφής μνήμης, αλλά δεν είναι αρκετό για την αντιστάθμιση των επιπρόσθετων καθυστερήσεων που εισάγονται από τις επιπλέον υποθετικές εντολές που εκτελούνται, από το συγχρονισμό των νημάτων και τις συγκρούσεις των εντολών επάνω στις ίδιες λειτουργικές μονάδες του συστήματος (παράγραφος 6.3.1). Εξάλλου, ο ρυθμός εκτέλεσης εντολών είναι στις περιπτώσεις των κωδικών MM και LU αρκετά υψηλός (IPC=1.38 στην περίπτωση του MM). Επομένως, είναι πραγματικά πολύ δύσκολο να επιτύχουμε επιτάχυνση του συνολικού χρόνου εκτέλεσης χωρίς να μειώσουμε τον συνολικό αριθμό εντολών των προγραμμάτων. Το διάγραμμα 6.26(δ) επιβεβαιώνει ότι στις εκδόσεις SPR αυξάνεται ο συνολικός αριθμός μικροεντολών που εκτελούνται. Ειδικά στις περιπτώσεις των LU και CG, ο συνολικός αριθμός των μικροεντολών είναι σχεδόν διπλάσιος σε σχέση με τη σειριακή έκδοση (*serial*) των εφαρμογών.

6.3.1 Επιπρόσθετες Παρατηρήσεις

		Instrumented thread		
	EXECUTION UNIT	***serial***	***tlp***	***spr***
MM	ALU0+ALU1:	27.06%	26.26%	37.56%
	FP_ADD:	11.70%	11.82%	0.00%
	FP_MUL:	11.70%	11.82%	4.13%
	MEM_LOAD:	38.76%	27.00%	58.30%
	MEM_STORE:	12.07%	12.02%	20.75%
	Total instructions:	4590588278	2270133929	202876770
LU	ALU0+ALU1:	38.84%	38.84%	38.16%
	FP_ADD:	11.15%	11.15%	0.00%
	FP_MUL:	11.15%	11.15%	0.00%
	MEM_LOAD:	49.24%	49.24%	38.40%
	MEM_STORE:	11.24%	11.24%	22.78%
	Total instructions:	3205661399	1622610935	3264715031
CG	ALU0+ALU1:	28.04%	23.95%	49.93%
	FP_ADD:	8.83%	7.49%	0.00%
	FP_MUL:	8.86%	7.53%	0.00%
	FP_MOVE:	17.05%	14.05%	0.00%
	MEM_LOAD:	36.51%	45.71%	19.09%
	MEM_STORE:	9.50%	8.51%	9.54%
	Total instructions:	11934228188	7069734891	166842453
BT	ALU0+ALU1:	8.06%	8.06%	12.06%
	FP_ADD:	17.67%	17.67%	0.00%
	FP_MUL:	22.04%	22.04%	0.00%
	FP_MOVE:	10.51%	10.51%	0.00%
	MEM_LOAD:	42.70%	42.70%	44.70%
	MEM_STORE:	16.01%	16.01%	42.94%
	Total instructions:	44973276097	22486809710	8398026979

Πίνακας 6.1: Ο βαθμός χρησιμοποίησης των λειτουργικών μονάδων του επεξεργαστή για ένα νήμα ανά περίπτωση

Τα αποτελέσματα της παραγράφου 5.3 επιβεβαιώνουν τα αποτελέσματα που προέκυψαν στο παρόν κεφάλαιο. Αναφερόμενοι στην περίπτωση των μετρο-προγραμμάτων MM και LU: Όταν δύο νήματα εκτελούνται παράλληλα σε έναν επεξεργαστή SMT, αξιοποιώντας την τεχνολογία της ταυτόχρονης υπερνημάτωσης, αναμένεται ότι θα προκύψουν σημαντικές καθυστερήσεις στην εκτέλεση εντολών κινητής υποδιαστολής, όταν αυτές συνυπάρχουν με εντολές φόρτωσης και αποθήκευσης δεδομένων όμοιου τύπου, οι οποίες δεν αστοχούν στην κρυφή μνήμη. Επιπρόσθετα, δεν υπάρχει η δυνατότητα παράλληλης εκτέλεσης πράξεων πάνω σε ακέραιους αριθμούς, λόγω συγκρούσεων επάνω στους κοινούς πόρους του συστήματος. Έτσι, οι εντολές που ελέγχουν τους δείκτες των φωλιασμένων βρόχων, και διπλασιάζονται όταν το υπολογιστικό φορτίο μοιράζεται

σε δύο νήματα, επιβαρύνουν τη συνολική επίδοση του συστήματος. Αντίθετα, οι κώδικες NAS έχουν υψηλό ποσοστό αστοχιών στην κρυφή μνήμη. Τέτοιες εντολές φόρτωσης και αποθήκευσης δεν αποτελούν σημαντικό παράγοντα επιβράδυνσης των εντολών που συνυπάρχουν μαζί τους και εκτελούν υπολογισμούς. Εξάλλου, οι άεργοι κύκλοι εκτέλεσης που προκύπτουν λόγω των αστοχιών μπορούν με την ταυτόχρονη πολυνημάτωση να αξιοποιηθούν με αποτέλεσμα την επιτάχυνση των προγραμμάτων. Βέβαια, η επιτάχυνση αυτή δεν είναι σημαντική, και σε κάποιες περιπτώσεις μη αξιόλογη, καθώς οι εντολές φόρτωσης και αποθήκευσης επιβραδύνουν η μία την άλλη, όταν συνυπάρχουν στη σωλήνωση του συστήματος.

Ο πίνακας 6.1 παρουσιάζει τα σχετικά ποσοστά χρησιμοποίησης των πιο πολυάσχολων λειτουργικών μονάδων του συστήματος, κατά την εκτέλεση των πειραμάτων της παρούσας ενότητας. Στην πρώτη στήλη εμφανίζονται τα αποτελέσματα κατά την εκτέλεση της σειριακής έκδοσης των προγραμμάτων. Η δεύτερη στήλη περιέχει τα στατιστικά στοιχεία των εκδόσεων TLP, για το ένα μόνο από τα δύο νήματα εκτέλεσης. Το υπολογιστικό φορτίο είναι σε όλες τις περιπτώσεις κατά το δυνατό ισο-μοιρασμένο, και επομένως τα αποτελέσματα για τα δύο νήματα είναι σχεδόν ταυτόσημα. Εξάλλου, σε γενικές γραμμές η κατανομή αυτή των δύο πρώτων στηλών είναι περίπου ίδια, δεδομένου ότι κατά την παραλληλοποίηση των εφαρμογών δεν αλλάζουμε το είδος τον εντολών, απλά τις κατανέμουμε σε δύο διαφορετικά νήματα. Η τρίτη στήλη παρουσιάζει τα στατιστικά των εκδόσεων SPR, για το βοηθητικό νήμα εκτέλεσης. Το κύριο νήμα εκτέλεσης των υπολογισμών έχει την ίδια κατανομή με τη σειριακή έκδοση του προγράμματος. Τα στατιστικά του πίνακα αναφέρονται στο ποσοστό (επί του συνόλου των εντολών του προγράμματος) των εντολών κάθε έκδοσης προγράμματος, που απασχολούν την κάθε μία από τις λειτουργικές μονάδες του συστήματος. Οι μετρήσεις έγιναν αποτυπώνοντας το προφίλ των προγραμμάτων μέσω του εργαλείου Pin (binary instrumentation tool) [LCM+05].

Στη περίπτωση του Πολλαπλασιασμού Πινάκων το πιο χαρακτηριστικό στοιχείο είναι ο μεγάλος αριθμός λογικών εντολών που χρησιμοποιούνται. Περίπου το 25% του συνόλου των εντολών, τόσο στη σειριακή έκδοση, όσο και στην έκδοση TLP, είναι λογικές εντολές λόγω της χρήσης δυαδικών μασκών για τον υπολογισμό της θέσης των μη γραμμικά αποθηκευμένων δεδομένων των πινάκων (κεφάλαιο 3). Ο επεξεργαστής Xeon περιέχει δύο αριθμητικές/λογικές μονάδες (ALU units: ALU0 και ALU1 [Int]). Ωστόσο, μόνο η μονάδα ALU0 μπορεί να εκτελέσει τις λογικές εντολές. Επομένως, η εκτέλεση των εντολών αυτών σειριοποιούνται αναγκαστικά, ανεξάρτητα από το βαθμό παραλληλισμού του προγράμματος.

Οι κώδικες NAS περιέχουν έναν σημαντικό αριθμό από εντολές fp-move, οι οποίες δρομολογούνται αναγκαστικά μέσω της πόρτας 0, του συστήματος, την ίδια με τη μονάδα ALU0. Η αυξημένη αυτή κίνηση ουσιαστικά ακυρώνει τη διπλή ταχύτητα της μονάδας ALU0. Η επιτάχυνση της έκδοσης TLP του κώδικα BT μπορεί να αποδοθεί στον μικρό αριθμό εντολών ALU, σε συνδυασμό με τον όχι πολύ αυξημένο αριθμό εντολών fp-move. Παρόμοια, η μη επιβράδυνση της έκδοσης TLP του κώδικα LU μπορούν επίσης να αποδοθούν στο διαμοιρασμό των αριθμητικών/λογικών εντολών στις δύο (διπλής ταχύτητας) μονάδες ALU. Βέβαια, και στις δύο περιπτώσεις, ο μεγάλος αριθμός άεργων κύκλων εκτέλεσης στις σειριακές εκδόσεις είναι που δίνει το προβάδισμα στις

εκδόσεις TLP.

Επιπρόσθετα, όπως άλλωστε ήταν αναμενόμενο, ένα σημαντικό ποσοστό των εντολών είναι πράξεις υπολογισμών με δεδομένα κινητής υποδιαστολής. Σημειώνουμε ότι τέτοιου είδους εντολές δεν συνυπάρχουν με καλά αποτελέσματα στις περιπτώσεις αυξημένου βαθμού παραλληλισμού.

Τέλος, οι εκδόσεις SPR αυξάνουν τον συνολικό αριθμό εντολών που πρέπει να εκτελεστούν από τις ήδη πολυάσχολες λειτουργικές μονάδες του συστήματος (ιδιαίτερα οι ALUs και οι μονάδες ανάγνωσης/εγγραφής στη μνήμη). Το παραπάνω, σε συνδυασμό με την επιβάρυνση που προκύπτει από το συγχρονισμό των δύο νημάτων εμποδίζει τη συνολική επιτάχυνση του συστήματος. Ιδιαίτερα, στις περιπτώσεις των βελτιστοποιημένων προγραμμάτων, όπως ο Πολλαπλασιασμός Πινάκων, δεν υπάρχουν ουσιαστικά άεργοι κύκλοι εκτέλεσης για να αξιοποιηθούν από το βοηθητικό νήμα εποικοδομητικά. Αντίθετα, στην περίπτωση του κώδικα BT, όπου υπάρχουν σημαντικός αριθμός αστοχιών δεδομένων, η έκδοση SPR δεν επιφέρει επιβράδυνση της επίδοσης.

ΚΕΦΑΛΑΙΟ 7

Επίλογος

Το χάσμα μεταξύ της επίδοσης των επεξεργαστών και των συστημάτων μνήμης διαρκώς αυξάνεται, με αποτέλεσμα η πλειοψηφία των εφαρμογών να σπαταλούν ένα μεγάλο ποσοστό του συνολικού χρόνου εκτέλεσης άεργες, περιμένοντας δεδομένα από τη μνήμη να φτάσουν μέχρι το αρχείο καταχωρητών του επεξεργαστή. Η ελαχιστοποίηση ή η συγκάλυψη του μέσου χρόνου αναμονής για την άφιξη των δεδομένων από τη μνήμη αποτελεί έναν από τους βασικότερους στόχους της έρευνας στο πεδίο των υψηλών επιδόσεων. Η ιεραρχία μνήμης υλοποιήθηκε για αυτόν ακριβώς το λόγο: για την προσωρινή αποθήκευση των πρόσφατα χρησιμοποιούμενων δεδομένων σε θέσεις κοντά στο σημείο επεξεργασίας τους. Και ενώ οι κρυφές μνήμες μειώνουν την μέση καθυστέρηση για την ανάγνωση δεδομένων από τη μνήμη, οι τεχνικές βελτιστοποίησης κώδικα επιχειρούν να εκμεταλλευτούν την τοπικότητα των αναφορών στους επαναληπτικούς κώδικες, για να μειώσουν τον αριθμό των αστοχιών στις κρυφές μνήμες.

Στην παρούσα διατριβή διατυπώθηκε το ερώτημα πόσο αποτελεσματικές μπορούν να είναι οι στατικές τεχνικές βελτιστοποίησης κώδικα στα μονο-επεξεργαστικά συστήματα. Ασχοληθήκαμε με την εφαρμογή μη-γραμμικών μετασχηματισμών ομαδοποίησης δεδομένων, σε αριθμητικούς κώδικες με επαναληπτικές εκτελέσεις εντολών. Σε κάθε περίπτωση, δεν αρκεί η εφαρμογή του βέλτιστου μη-γραμμικού μετασχηματισμού δεδομένων. Είναι απαραίτητη μία αυτόματη και ταχεία δεικτοδότηση των μη-γραμμικά αποθηκευμένων στοιχείων στη σειριακή μνήμη, μετασχηματίζοντας τους πολυδιάστατους γραμμικούς δείκτες. Οι μη-γραμμικοί μετασχηματισμοί δεδομένων μπορούν να εφαρμοστούν με επιτυχία, όταν συνδυάζονται με ένα γρήγορο σχήμα δεικτοδότησης.

7.1 Συμβολή της Διατριβής - Περίληψη

Τα βασικότερα συμπεράσματα που προέκυψαν από τη διατριβή αυτή είναι τα ακόλουθα:

1. Η αυτόματη παραγωγή μετασχηματισμένου κώδικα με χρήση tiling και μη-γραμμικών μετασχηματισμών ομαδοποίησης δεδομένων, απαιτούν μία ταχύτατη και απλή στην υλοποίησή της μέθοδο για τον υπολογισμό των δεικτών που εντοπίζουν τα δεδομένα των πινάκων στη

σειριακή μνήμη. Προτείναμε τέσσερεις διαφορετικούς νέους μη-γραμμικούς μετασχηματισμούς ομαδοποίησης δεδομένων, και εφαρμόσαμε σε αυτούς μία δεικτοδότηση που βασίζεται στην αριθμητική των "αραιωμένων" ακεραίων, επεκτείνοντας αυτήν που χρησιμοποιείται στους αναδρομικούς πίνακες Morton. Συνδυάζουμε, επομένως, την αυξημένη τοπικότητα αναφοράς δεδομένων, χάρη στη μη-γραμμική αποθήκευση των στοιχείων των πινάκων, με την γρήγορη εύρεση των στοιχείων αυτών, με απλούς δυαδικούς υπολογισμούς. Ο μετασχηματισμένος κώδικας με tiling, δεν προσπελαύνει τα δεδομένα των πινάκων σειριακά, αλλά τα ομαδοποιεί και επαναχρησιμοποιεί τις ομάδες αυτές. Ο μη-γραμμικός μετασχηματισμός δεδομένων τα διατάσσει όπως ακριβώς τα ζητάει ο μετασχηματισμένος κώδικας με tiling, και η γρήγορη δεικτοδότησή τους μας εξασφαλίζει ότι θα μπορούμε να βρούμε ταχύτατα τη θέση αποθήκευσης των ζητούμενων δεδομένων στη μνήμη. Με τη μέθοδο αυτή πετυχαίνουμε δραστική μείωση των αστοχιών μνήμης, καθώς η τοπικότητα δεδομένων είναι βέλτιστη. Το αποτέλεσμα είναι η επιτάχυνση του συνολικού χρόνου εκτέλεσης, αφού δεν υπάρχει χρονική επιβάρυνση λόγω της σύνθετης αναζήτησης των μη-γραμμικά αποθηκευμένων δεδομένων.

2. Παρουσιάζουμε τη θεωρητική ανάλυση της συμπεριφοράς των διαφορετικών επιπέδων κρυφής μνήμης και του TLB, κατά την εκτέλεση μετασχηματισμένων κωδίκων με μη-γραμμικούς μετασχηματισμούς δεδομένων. Σύμφωνα με την ανάλυση αυτή, το βέλτιστο μέγεθος tile που αξιοποιεί στο μέγιστο τη χωρητικότητα της L1 κρυφής μνήμης, πρέπει να έχει τις μέγιστες δυνατές διαστάσεις που του επιτρέπουν να χωράει ολόκληρο στην L1 κρυφή μνήμη και να μην επιφέρει την παραγωγή αστοχιών σύγκρουσης. Αποδεικνύουμε ότι στους βελτιστοποιημένους κώδικες, όπου έχει εφαρμοστεί ανάθεση μεταβλητών σε προσωρινούς καταχωρητές, ευθυγράμμιση των διαφορετικών πινάκων που εμπλέκονται στο πρόγραμμα, πρώιμη ανάκληση δεδομένων και ξεδίπλωμα βρόχων, το μέγεθος του tile πρέπει να ισούται με το μέγεθος της L1 κρυφής μνήμης. Το μέγεθος αυτό αξιοποιεί στο μέγιστο τη χωρητικότητα της L1 κρυφής μνήμης ακόμα και στις περιπτώσεις που ο κώδικας προσπελαύνει δεδομένα από περισσότερα του ενός πίνακα. Η όποια αύξηση των αστοχιών σύγκρουσης κρύβεται μέσω της πρώιμης ανάκλησης δεδομένων. Αυτά τα σχετικά (συγκρινόμενα με ό,τι είχε προταθεί στη βιβλιογραφία) μεγάλα tiles, μειώνουν και την απώλεια κρίσιμων κύκλων εκτέλεσης, εξαιτίας της λάθος πρόβλεψης διακλάδωσης που λαμβάνει χώρα στο τέλος κάθε επαναληπτικού βρόχου.

3. Αποτιμήσαμε την αποτελεσματικότητα των προτεινόμενων μεθόδων συνδυαζόμενες και με άλλες βελτιστοποιήσεις κώδικα σε έναν αριθμό από διαφορετικές πλατφόρμες. Τα πειραματικά αποτελέσματα και οι προσομοιώσεις υποδεικνύουν ότι τα χαρακτηριστικά της αρχιτεκτονικής κάθε πλατφόρμας ευνοούν και υπαγορεύουν συγκεκριμένες βελτιστοποιήσεις κώδικα. Στο σημείο αυτό, αξίζει να αναφέρουμε ότι ακόμα και οι επεξεργαστές που είναι εξοπλισμένοι με την τεχνική της ταυτόχρονης πολυνημάτωσης έχουν καλύτερη επίδοση όταν εκτελείται σε αυτούς ένα και μοναδικό βελτιστοποιημένο νήμα. Η αξιοποίηση των υπολογιστικών πόρων και επομένως ο ρυθμός εκτέλεσης εντολών γίνεται με σχεδόν βέλτιστο τρόπο,

τόσο ώστε οι τεχνικές παραλληλοποίησης σε επίπεδο νήματος δεν έφεραν αξιόλογη βελτίωση στους αριθμητικούς επαναληπτικούς κώδικες. Οι τεχνικές πολυνημάτωσης μπορουν να αποδειχτούν αποτελεσματικές μόνο σε κώδικες με τυχαία μοτίβα προσπέλασης δεδομένων. Στους κώδικες αυτούς ο ρυθμός ολοκλήρωσης εντολών είναι μειωμένος και οι εντολές από τα διαφορετικά νήματα δεν καθυστερούν ανταγωνιζόμενες για τις μοιραζόμενες λειτουργικές μονάδες του επεξεργαστή.

Appendices

ΠΑΡΑΡΤΗΜΑ A

Πίνακας Συμβόλων

Explanation	symbol
L1 cache :	C_{L1}
L1 cache line :	L_1
L1 miss penalty:	p_{L1}
total L1 cache misses :	M_1
L2 cache :	C_{L2}
L2 cache line :	L_2
L2 miss penalty:	p_{L2}
total L2 cache misses :	M_2
TLB entries (L1):	E
TLB entries (L2):	E_2
page size :	P
TLB miss penalty:	p_{TLB}
total TLB misses :	M_{TLB}
mispred. branch penalty:	c_{br}
array size :	N
tile size :	T
tiles per array line :	$x = \frac{N}{T}$

Πίνακας A.1: Πίνακας Συμβόλων

ΠΑΡΑΡΤΗΜΑ B

Η Αρχιτεκτονική των Μηχανημάτων Εκτέλεσης των Πειραματικών Μετρήσεων

	UltraSPARC	Pentium III	SGI Origin R10K
CPU freq. :	400MHz	800MHz	250MHz
L1 data cache :	16KB	16KB	32 KB
L1 associativity:	direct-mapped	4-way set asssoc.	2-way set assoc.
L1 cache line :	32B	32B	64B
L1 miss penalty:	8 cc	4 cc	6 cc
L2 cache :	4MB	256KB	4MB
L2 cache :	direct-mapped	8-way set asssoc.	2-way set assoc.
L2 cache line :	64B	64B	128B
L2 miss penalty:	84 cc	60 cc	100 cc
TLB entries (L1):	64 addresses	64 addresses	64 addresses
TLB entries (L2):			
TLB asssociativity:	fully assoc.	4-way set asssoc.	fully assoc.
page size :	8KB	4KB	32KB
TLB miss penalty:	51 cc	5 cc	57 cc
mispredicted branch penalty:	4 cc	10-15 cc	4 cc

Πίνακας B.1: Πίνακας αρχιτεκτονικών χαρακτηριστικών

	Athlon XP	Xeon
CPU freq. :	2GHz	2,8GHz
L1 cache :	64KB	16KB
	2-way set asssoc.	8-way set asssoc.
L1 line :	64B	64/128B
L1 miss penalty:	3 cc	4 cc
L2 cache :	256KB	1MB
	16-way set asssoc.	8-way set asssoc.
L2 line :	64B	64B
L2 miss penalty:	20 cc	18 cc
TLB entries (L1):	40 addresses	64 addresses
TLB entries (L2):	256 addresses	
page size :	4KB	4KB
TLB miss penalty:	3 cc	30 cc
mispredicted branch penalty:	10 cc	20 cc

Πίνακας Β.2: Πίνακας αρχιτεκτονικών χαρακτηριστικών

ΠΑΡΑΡΤΗΜΑ C

Κώδικες Προγραμμάτων

Παρουσιάζουμε Α. του αρχικούς κώδικες (όπου έχει εφαρμοστεί μόνο αναδιάταξη των βρόχων προκειμένου να επιτευχθεί η βέλτιστη σχετική θέση τους) και Β. τους βελτιστοποιημένους-μετασχηματισμένους κώδικες, αμέσως πριν την εφαρμογή των μη-γραμμικών διατάξεων δεδομένων με δεικτοδότηση μέσω δυαδικών μασκών.

C.1 Matrix Multiplication

A. (We consider that all arrays are row-wise stored)

```
for (i = 0; i < N; i++)
  for (k = 0; k < N; k++)
    for (j = 0; j < N; j++)
      C[i][j]+ = A[i][k] * B[k][j];
```

B.

```
for (ii = 0; ii < N; ii + step)
  for (kk = 0; kk < N; kk + step)
    for (jj = 0; jj < N; jj + step)
      for (i = 0; (i < ii + step && i < N); i++)
        for (k = 0; (k < kk + step && k < N); k++)
          for (j = 0; (j < jj + step && j < N); j++)
            C[i][j]+ = A[i][k] * B[k][j];
```

C.2 LU decomposition

A. (We consider that all arrays are column-wise stored)

```
for (k = 0; k < N − 1; k++){
  for (i = k + 1; i < N; i++)
    A[i][k] = A[i][k]/A[k][k];
  for (j = k + 1; j < N; j++)
    for (i = k + 1; i < N; i++)
      A[i][j]− = A[i][k] ∗ A[k][j];
}
```

B.

```
for (kk = 0; kk < N − 1; kk+ = step)
  for (jj = kk; jj < N; jj+ = step)
    for (ii = kk; ii < N; ii+ = step){
      if (ii > kk && jj > kk)
        kreturn=(N < (kk + step)?N : (kk + step));
      else kreturn = (N < (kk + step)?N : (kk + step)) − 1;
      for (k = kk; k < kreturn; k++){
        jstart = (k + 1 > jj?k + 1 : jj);
        istart = (k + 1 > ii?k + 1 : ii);
        if (jstart == k + 1)
          for (i = istart; i < ireturn; i++){
            A[i][k] = A[i][k]/A[k][k];
        for (j = jstart; (j < jj + step && j < N); j++)
          for (i = istart; (i < ii + step && i < N); i++)
            A[i][j]− = A[i][k] ∗ A[k][j];
      }
    }
```

C.3 STRMM: Product of Triangular and Square Matrix

A. (We consider that all arrays are row-wise stored)

```
for (i = 0; i < N − 1; i++)
  for (k = i + 1; k < N; k++)
    for (j = 0; j < N; j++)
      B[i][j]+ = A[i][k] ∗ B[k][j];
```

B.

```
for (ii = 0; ii < N − 1; ii+ = step)
  for (kk = ii; kk < N; kk+ = step)
    for (jj = 0; jj < N; jj+ = step)
      for (i = ii; (i < ii + step &$ i < N − 1); i + +){
        kstart = (i + 1 > kk?i + 1 : kk);
        for (k = kstart; (k < kk + step && k < N); k + +)
          for (j = jj; (j < jj + step && j < N); j + +)
            B[i][j]+ = A[i][k] ∗ B[k][j];
      }
```

C.4 SSYMM: Symmetric Matrix-Matrix Operation

A. (We consider that all arrays are column-wise stored)

```
for (j = 0; j < N; j + +)
  for (i = 0; i < N; i + +)}
    for (k = 0; k < i; k + +){
      C[k][j]+ = A[k][i] ∗ B[i][j];
      C[i][j]+ = A[k][i] ∗ B[k][j];
    }
    C[i][j]+ = A[i][i] ∗ B[i][j];
  }
```

B.

```
for (jj = 0; jj < N; jj+ = step)
  for (ii = 0; ii < N; ii+ = step)
    for (kk = 0; kk <= ii; kk+ = step)
      for (j = jj; (j < jj + step && j < N); j + +)
        for (i = ii; (i < ii + step && i < N); i + +){
          for (k = kk; (k < kk + step && k < i); k + +){
            C[k][j]+ = A[k][i] ∗ B[i][j];
            C[i][j]+ = A[k][i] ∗ B[k][j];
          }
          if (ii == kk)
            C[i][j]+ = A[i][i] ∗ B[i][j];
        }
```

C.5 SSYR2K: Symmetric Rank 2k Update

A. (We consider that all array C is row-wise stored, while arrays A, B are column-wise stored)

```
for (k = 0; k < N; k++)
  for (i = 0; i < N; i++)
    for (j = i; j < N; j++)
      C[i][j]+= B[j][k]*A[i][k] + A[j][k]*B[i][k];
```

B.

```
for (kk = 0; kk < N; kk+= step)
  for (ii = 0; ii < N; ii+= step)
    for (jj = ii; jj < N; jj+= step)
      for (k = kk; (k < kk + step && k < N); k++)
        for (i = ii; (i < ii + step && i < N); i++){
          jstart = (i > jj?i : jj);
          for (j = jstart; (j < jj + step && j < N); j++)
            C[i][j]+= B[j][k]*A[i][k] + A[j][k]*B[i][k];
        }
```

Bibliography

[AAKK05] Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis, and Nectarios Koziris. Tuning Blocked Array Layouts to Exploit Memory Hierarchy in SMT Architectures. In *Proceedings of the 10th Panhellenic Conference in Informatics*, Volos, Greece, Nov. 2005.

[AAKK06a] Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis, and Nectarios Koziris. Exploring the Capacity of a Modern smt Architecture to Deliver High Scientific Application Performance. In *Proc. of the 2006 International Conference on High Performance Computing and Communications (HPCC-06)*, Munich, Germany, Sep 2006. Lecture Notes in Computer Science.

[AAKK06b] Evangelia Athanasaki, Nikos Anastopoulos, Kornilios Kourtis, and Nectarios Koziris. Exploring the Performance Limits of Simultaneous Multithreading for Scientific Codes. In *Proc. of the 2006 International Conference on Parallel Processsing (ICPP-06)*, Columbus, Ohio, Aug 2006. IEEE Computer Society Press.

[AK03] Evangelia Athanasaki and Nectarios Koziris. Blocked Array Layouts for Multi-level Memory Hierarchies. In *Proceedings of the 9th Panhellenic Conference in Informatics*, pages 193–207, Thessaloniki, Greece, Nov. 2003.

[AK04a] Evangelia Athanasaki and Nectarios Koziris. Fast Indexing for Blocked Array Layouts to Improve Multi-Level Cache Locality. In *Proc. of the 8-th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-8), In conjuction with the 10th International Symposium on High-Performance Computer Architecture (HPCA-10)*, pages 109–119, Madrid, Spain, Feb 2004. IEEE Computer Society Press.

[AK04b] Evangelia Athanasaki and Nectarios Koziris. Improving Cache Locality with Blocked Array Layouts. In *Proceedings of the 12-th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP'04)*, pages 308–317, A Coruna, Spain, Feb. 2004. IEEE Computer Society Press.

[AK05] Evangelia Athanasaki and Nectarios Koziris. Fast Indexing for Blocked Array Layouts to Reduce Cache Misses. *International Journal of High Performance Computing and Networking (IJHPCN)*, 3(5/6):417–433, 2005.

[AKT05] Evangelia Athanasaki, Nectarios Koziris, and Panayiotis Tsanakas. A Tile Size Selection Analysis for Blocked Array Layouts. In *Proc. of the 9-th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9), In conjuction with the 11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, pages 70–80, San Francisco, CA, Feb 2005. IEEE Computer Society Press.

[APD01] M. Annavaram, J. Patel, and E. Davidson. Data Prefetching by Dependence Graph Precomputation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA '01)*, pages 52–61, Göteborg, Sweden, July 2001.

[BAYT04] Abdel-Hameed A. Badawy, Aneesh Aggarwal, Donald Yeung, and Chau-Wen Tseng. The Efficacy of Software Prefetching and Locality Optimizations on Future Memory Systems. *The Journal of Instruction-Level Parallelism*, 6:1–35, June 2004.

[BP04] James R. Bulpin and Ian A. Pratt. Multiprogramming Performance of the Pentium 4 with Hyper-Threading. In *Proceedings of the Third Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD 2004) held in conjunction with ISCA ŏ04*, page 53ŭ62, Munich, Germany, June 2004.

[CB94] T.-F. Chen and J.-L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In *Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA '94)*, pages 223–232, Chicago, IL, Apr 1994.

[Che95] T. Chen. An Effective Programmable Prefetch Engine for On-Chip Caches. In *Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-28)*, pages 237–242, Ann Arbor, MI, Dec 1995.

[CJL+99] Siddhartha Chatterjee, Vibhor V. Jain, Alvin R. Lebeck, Shyam Mundhra, and Mithuna Thottethodi. Nonlinear Array Layouts for Hierarchical Memory Systems. In *Proc. of the 13th ACM Int. Conf. on Supercomputing (ICS '99)*, pages 444–453, Rhodes, Greece, June 1999.

[CL95] Michael Cierniak and Wei Li. Unifying Data and Control Transformations for Distributed Shared-Memory Machines. In *Proc. of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, pages 205–217, La Jolla, CA, June 1995.

[CLPT99] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive Array Layouts and Fast Parallel Matrix Multiplication. In *Proc.*

of the 11th Annual Symp. on Parallel Algorithms and Architectures (SPAA '99), pages 222–231, Saint Malo, France, June 1999.

[CM95] Stephanie Coleman and Kathryn S. McKinley. Tile Size Selection Using Cache Organization and Data Layout. In *Proc. of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, pages 279–290, La Jolla, CA, June 1995.

[CM99] Jacqueline Chame and Sungdo Moon. A Tile Selection Algorithm for Data Locality and Cache Interference. In *Proc. of the 13th ACM Int. Conf. on Supercomputing (ICS '99)*, pages 492–499, Rhodes, Greece, June 1999.

[CTWS01] Jamison D. Collins, Dean M. Tullsen, Hong Wang, and John P. Shen. Dynamic Speculative Precomputation. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-34)*, pages 306–317, Austin, TX, Dec 2001.

[CWT+01] Jamison D. Collins, Hong Wang, Dean M. Tullsen, Christopher Hughes, Yong-Fong Lee, Dan Lavery, and John P. Shen. Speculative Precomputation: Long-Range Prefetching of Delinquent Loads. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA '01)*, pages 14–25, Göteborg, Sweden, July 2001.

[Ess93] K. Esseghir. Improving Data Locality for Caches. Master's thesis, Department of Computer Science, Rice University, Houston, Texas, Sept 1993.

[FST91] Jeanne Ferrante, V Sarkar, and W Thrash. On Estimating and Enhancing Cache Effectiveness. In *Proceedings of the 4th International Workshop on Languages and Compilers for Parallel Computing*, Aug 1991.

[GMM98] Somnath Ghosh, Margaret Martonosi, and Sharad Malik. Precise Miss Analysis for Program Transformations with Caches of Arbitrary Associativity. In *Proc. of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, pages 228–239, San Jose, CA, Oct 1998.

[GMM99] Somnath Ghosh, Margaret Martonosi, and Sharad Malik. Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(4):703–746, July 1999.

[HK04] Chung-Hsing Hsu and Ulrich Kremer. A Quantitative Analysis of Tile Size Selection Algprithms. *The Journal of Supercomputing*, 27(3):279–294, Mar 2004.

[HKN+92] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa. An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads. In *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA '92)*, pages 136–145, Gold Coast, Australia, May 1992.

[HKN99] John S. Harper, Darren J. Kerbyson, and Graham R. Nudd. Analytical Modeling of Set-Associative Cache Behavior. *IEEE Transactions Computers*, 48(10):1009–1024, Oct 1999.

[Int] Intel Corporation. *IA-32 Intel Architecture Optimization Reference Manual*. Order Number: 248966-012.

[Int01] Intel Corporation. *Using Spin-Loops on Intel Pentium 4 Processor and Intel Xeon Processor*, May 2001. Order Number: 248674-002.

[JG97] D. Joseph and D. Grunwald. Prefetching using Markov Predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA '97)*, pages 252–263, Denver, CO, June 1997.

[Jim99] Marta Jimenez. *Multilevel Tiling for Non-Rectangular Iteration Spaces*. PhD thesis, Universitat Politecnica de Catalunya, May 1999.

[Jou90] Norman P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '90)*, pages 364–373, Seattle, WA, May 1990.

[Kan01] Mahmut Taylan Kandemir. Array Unification: A Locality Optimization Technique. In *Proc. of the 10th International Conference on Compiler Construction (CC'01)*, pages 259–273, Genova, Italy, Apr 2001.

[KCS+99] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam. A Linear Algebra Framework for Automatic Determination of Optimal Data Layouts. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):115–135, Feb 1999.

[KKO00] T. Kisuki, P.M.W. Knijnenburg, and M.F.P. OőBoyle. Combined Selection of Tile Sizes and Unroll Factors Using Iterative Compilation. In *Proc. of the 2000 International Conference on Parallel Architectures and Compilation Techniques (PACT'00)*, pages 237–248, Philadelphia, Pennsylvania, Oct 2000.

[KLW+04] Dongkeun Kim, Shih-Wei Liao, Perry H. Wang, Juan del Cuvillo, Xinmin Tian, Xiang Zou, Hong Wang, Donald Yeung, Milind Girkar, and John Paul Shen. Physical

experimentation with prefetching helper threads on intel's hyper-threaded processors. In *Proceedings of the 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO 2004)*, pages 27–38, San Jose, CA, Mar 2004.

[KPCM99] Induprakas Kodukula, Keshav Pingali, Robert Cox, and Dror Maydan. An Experimental Evaluation of Tiling and Shackling for Memory Hierarchy Management. In *Proc. of the 13th ACM International Conference on Supercomputing (ICS '99)*, pages 482–491, Rhodes, Greece, June 1999.

[KRC99] M. Kandemir, J. Ramanujam, and Alok Choudhary. Improving Cache Locality by a Combination of Loop and Data Transformations. *IEEE Transactions on Computers*, 48(2):159–167, Feb 1999.

[KRCB01] M. Kandemir, J. Ramanujam, A. Choudhary, and P. Banerjee. A Layout-conscious Iteration Space Transformation Technique. *IEEE Transactions on Computers*, 50(12):1321–1336, Dec 2001.

[LCM+05] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *SIGPLAN Not.*, 40(6):190–200, 2005.

[LLC02] Chun-Yuan Lin, Jen-Shiuh Liu, and Yeh-Ching Chung. Efficient Representation Scheme for Multidimensional Array Operations. *IEEE Transactions on Computers*, 51(03):327–345, Mar 2002.

[LM96] Chi-Keung Luk and Todd Mowry. Compiler-Based Prefetching for Recursive Data Structures. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 222–233, Boston, MA, Oct 1996.

[LM99] Chi-Keung Luk and Todd Mowry. Automatic Compiler-Inserted Prefetching for Poiner-based Applications. *IEEE Transactions on Computers*, 48(2):134–141, Feb 1999.

[LRW91] Monica S. Lam, Edward E. Rothberg, and Michael E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. In *Proc. of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, Santa Clara, CA, April 1991.

[Luk01] Chi-Keung Luk. Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA '01)*, pages 40–51, Göteborg, Sweden, July 2001.

[LW94] Alvin R. Lebeck and David A. Wood. Cache Profiling and the SPEC Benchmarks: A Case Study. *IEEE Computer*, 27(10):15–26, Oct 1994.

[MCFT99] Nicholas Mitchell, Larry Carter, Jeanne Ferrante, and Dean Tullsen. ILP versus TLP on SMT. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, Nov 1999.

[MCT96] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving Data Locality with Loop Transformations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(04):424–453, July 1996.

[MG91] Todd Mowry and Anoop Gupta. Tolerating Latency Trough Software-Controlled Prefetching in Shared-Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, 12(2):87–106, June 1991.

[MHCF98] Nicholas Mitchell, Karin Högstedt, Larry Carter, and Jeanne Ferrante. Quantifying the Multi-Level Nature of Tiling Interactions. *International Journal of Parallel Programming*, 26(6):641–670, Dec 1998.

[Mit00] Nicholas Matthew Mitchell. *Guiding Program Transformations with Modal Performance Models.* PhD thesis, University of California, San Diego, Aug 2000.

[MLG92] Todd C. Mowry, Monica S. Lam, and Anoop Gupta. Design and Evaluation of a Compiler Algorithm for Prefetching. In *Proc. of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS V)*, pages 62–73, Boston, MA, Oct 1992.

[MPB01] A. Moshovos, D. Pnevmatikatos, and A. Baniasadi. Slice Processors: an Implementation of Operation-Based Prediction. In *Proceedings of the 15th International Conference on Supercomputing (ICS '01)*, pages 321–334, Sorrento, Italy, June 2001.

[Ope03] Omni OpenMP Compiler Project. Released in the International Conference for High Performance Computing, Networking and Storage (SC'03), Nov 2003.

[PHP02] Neungsoo Park, Bo Hong, and Viktor Prasanna. Analysis of Memory Hierarchy Performance of Block Data Layout. In *Proc. of the International Conference on Parallel Processing (ICPP 2002)*, pages 35–44, Vancouver, Canada, Aug 2002.

[PHP03] Neungsoo Park, Bo Hong, and Viktor Prasanna. Tiling, Block Data Layout, and Memory Hierarchy Performance. *IEEE Transactions on Parallel and Distributed Systems*, 14(07):640–654, July 2003.

[PJ03] D. Patterson and J.Hennessy. *Computer Architecture. A Quantitative Approach*, pages 373–504. Morgan Kaufmann Pub., San Francisco, CA, 3rd edition, 2003.

[PNDN99] Preeti Ranjan Panda, Hiroshi Nakamura, Nikil D. Dutt, and Alexandru Nicolau. Augmenting Loop Tiling with Data Alignment for Improved Cache Performance. *IEEE Transactions on Computers*, 48(2):142–149, Feb 1999.

[RS01] A. Roth and G. Sohi. Speculative Data-Driven Multithreading. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA '01)*, pages 37–48, Nuevo Leone, Mexico, Jan 2001.

[RT98a] Gabriel Rivera and Chau-Wen Tseng. Data Transformations for Eliminating Conflict Misses. In *Proc. of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation (PLDI'98)*, pages 38–49, Montreal, Canada, June 1998.

[RT98b] Gabriel Rivera and Chau-Wen Tseng. Eliminating Conflict Misses for High Performance Architectures. In *Proc. of the 12th International Conference on Supercomputing (SC'98)*, pages 353–360, Melbourne, Australia, July 1998.

[RT99a] Gabriel Rivera and Chau-Wen Tseng. A Comparison of Compiler Tiling Algorithms. In *Proc. of the 8th International Conference on Compiler Construction (CC'99)*, pages 168–182, Amsterdam, The Netherlands, March 1999.

[RT99b] Gabriel Rivera and Chau-Wen Tseng. Locality Optimizations for Multi-Level Caches. In *Proc. of the 1999 ACM/IEEE Conference on Supercomputing (SC'99)*, CDROM article no. 2, Portland, OR, Nov 1999.

[SL99] Yonghong Song and Zhiyuan Li. New Tiling Techniques to Improve Cache Temporal Locality. In *Proc. of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation (PLDI'99)*, pages 215–228, Atlanta, Georgia, May 1999.

[SL01] Yonghong Song and Zhiyuan Li. Impact of Tile-Size Selection for Skewed Tiling. In *Proc. of the 5-th Workshop on Interaction between Compilers and Architectures (INTERACT'01)*, Monterrey, Mexico, Jan 2001.

[SPR00] Karthik Sundaramoorthy, Zachary Purser, and Eric Rotenberg. Slipstream Processors: Improving both Performance and Fault Tolerance. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 257–268, Cambridge, MA, Nov 2000.

[SU96] Ulrich Sigmund and Theo Ungerer. Identifying Bottlenecks in a Multithreaded Superscalar Microprocessor. In *Proceedings of the 2nd International Euro-Par Conference on Parallel Processing-Volume II (Euro-Par '96)*, pages 797–800, Lyon, France, Aug 1996.

[TEE+96] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA '96)*, pages 191–202, Philadelphia, PA, May 1996.

[TEL95] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA '95)*, pages 392–403, Santa Margherita Ligure, Italy, June 1995.

[TFJ94] O. Temam, C. Fricker, and W. Jalby. Cache Interference Phenomena. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 261–271, Nashville, Tennessee, May 1994.

[TGJ93] O. Temam, E. D. Granston, and W. Jalby. To Copy or Not to Copy: A Compile-Time Technique for Assessing When Data Copying Should be Used to Eliminate Cache Conflicts. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing (SC'93)*, pages 410–419, Portland, OR, Nov 1993.

[TT03] Nathan Tuck and Dean M. Tullsen. Initial Observations of the Simultaneous Multithreading Pentium 4 Processor. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT '03)*, New Orleans, LA, Sep 2003.

[TWN04] Filip Blagojevic Tanping Wang and Dimitrios S. Nikolopoulos. Runtime Support for Integrating Precomputation and Thread-Level Parallelism on Simultaneous Multithreaded Processors. In *Proceedings of the 7th ACM SIGPLAN Workshop on Languages, Compilers, and Runtime Support for Scalable Systems (LCR'2004)*, Houston, TX, Oct 2004.

[Ver03] Xavier Vera. *Cache and Compiler Interaction (how to analyze, optimize and time cache behaviour).* PhD thesis, Malardalen University, Jan 2003.

[WAFG01] David S. Wise, Gregory A. Alexander, Jeremy D. Frens, and Yuhong Gu. Language Support for Morton-order Matrices. In *Proc. of the 2001 ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'01)*, pages 24–33, Snowbird, Utah, USA, June 2001.

[WF99] David S. Wise and Jeremy D. Frens. Morton-order Matrices Deserve Compilers' Support. TR533, CS Dept., Indiana University, Nov 1999.

[Wis01] David S. Wise. Ahnentafel Indexing into Morton-ordered Arrays, or Matrix Locality for Free. In *1st Euro-Par 2000 Int. Workshop on Cluster Computing*, pages 774–783, Munich, Germany, June 2001.

[WL91] Michael E. Wolf and Monica S. Lam. A Data Locality Optimizing Algorithm. In *Proc. of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30–44, Toronto, Ontario, Canada, June 1991.

[WM95] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[WMC96] Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen. Combining Loop Transformations Considering Caches and Scheduling. In *Proc. of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 274–286, Paris, France, Dec 1996.

[WWW+02] H. Wang, P. Wang, R.D. Weldon, S.M. Ettinger, H. Saito, M. Girkar, S.S-W. Liao, and J. Shen. Speculative Precomputation: Exploring the Use of Multithreading for Latency. *Intel Technology Journal*, 6(1):22–35, Feb 2002.

[ZS01] C. Zilles and G. Sohi. Execution-Based Prediction Using Speculative Slices. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA '01)*, pages 2–13, Göteborg, Sweden, July 2001.