ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Υλοποίηση και βελτιστοποίηση αλγορίθμων για πολυπύρηνα συστήματα και επιταχυντές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γιώργου Δ. Ζαχαριάδη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Υλοποίηση και βελτιστοποίηση αλγορίθμων για πολυπύρηνα συστήματα και επιταχυντές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γιώργου Δ. Ζαχαριάδη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 25η Ιουλίου 2014

.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Αν. Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

.....
Γεώργιος Δ. Ζαχαριάδης

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

Περίληψη

Τα τελευταία χρόνια, το τοπίο στα σύγχρονα συστήματα παράλληλης επεξεργασίας έχει αλλάξει σε μεγάλο βαθμό. Η βελτίωση των υπολογιστικών επιδόσεων έχει επέλθει από τη σημαντική αύξηση των επεξεργαστικών πυρήνων και όχι από την αύξηση της συχνότητας λειτουργίας, όπως συνέβαινε σχεδόν τις δύο τελευταίες δεκαετίες.

Συνεπώς, στις μέρες μας οι πολυπύρηνοι επεξεργαστές (multicore) φιλοξενούν μέχρι και 12 ή 15 πυρήνες ανά επεξεργαστή. Επιπλέον, αρχιτεκτονικές επιταχυντών (manycore) σαν τις κάρτες γραφικών, οι οποίες ενσωματώνουν πολλούς απλούστερους επεξεργαστικούς πυρήνες, προσφέρουν πολύ υψηλές υπολογιστικές δυνατότητες και σημαντικές ευκαιρίες για εφαρμογές παράλληλου προγραμματισμού. Όμως, η δυσκολία στον προγραμματισμό τους και τα διαφορετικά προγραμματιστικά μοντέλα που ακολουθούν, αποτελούν μεγάλη πρόκληση για τους προγραμματιστές.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η υλοποίηση και η βελτιστοποίηση των αλγορίθμων Floyd Warshall και LU Decomposition σε συστήματα πολυπύρηνων επεξεργαστών, σε κάρτες γραφικών της εταιρίας Nvidia και στη νέα manycore αρχιτεκτονική της εταιρίας Intel (Many Integrated Core). Στόχος είναι τόσο η αντιμετώπιση των προκλήσεων που παρουσιάζονται, όσο και η σύγκριση και η αξιολόγηση των παραπάνω διαφορετικών αρχιτεκτονικών.

Λέξεις Κλειδιά

Πολυπύρηνοι επεξεργαστές, Υπολογισμοι Γενικού Σκοπού σε Κάρτες Γραφικών (GPGPU), Κάρτα Γραφικών (GPU), CUDA, αρχιτεκτονική Nvidia Fermi, αρχιτεκτονική Nvidia Kepler, Intel Many Integrated Core (MIC), Intel Xeon Phi, Floyd Warshall, LU Decomposition

Abstract

In recent years, the landscape of modern parallel processing systems has changed greatly. Advancement in computer performance has been coming through hardware parallelism instead of from the clock-rate increases that we enjoyed for roughly 20 years.

As a result, multicore processors have taken us from one core per processor up to 12 or 15 cores per processor today. Moreover, manycore architectures like GPU accelerators, which incorporate many simpler processor cores, offer very high computational capabilities and significant opportunities for parallel programming. However, the difficulties in programming such devices and their unfamiliar programming models present a major challenge for developers.

The purpose of the current diploma thesis is the implementation and optimization of Floyd Warshall and LU Decomposition algorithms on multicore architectures, Nvidia GPUs and Intel's new manycore architecture (Many Integrated Core). The goal is to address the challenges presented and to compare and evaluate all these different architectures.

Keywords

Multicore, Manycore, General Purpose computing on GPUs (GPGPU), Graphics Processing Unit (GPU), CUDA, Nvidia Fermi, Nvidia Kepler, Intel Many Integrated Core (MIC), Intel Xeon Phi, Floyd Warshall, LU Decomposition

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη του Καθηγητή Ε.Μ.Π. Νεκτάριου Κοζύρη.

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Κοζύρη τόσο για την ευκαιρία που μου έδωσε να έρθω σε επαφή με τον επιστημονικό τομέα των υπολογιστικών συστημάτων όσο και για τις γνώσεις και την έμπνευση που μου προσέφερε κατά τη διάρκεια των σπουδών μου.

Στη συνέχεια, θα ήθελα να ευχαριστήσω ιδιαιτέρως τον Λέκτορα Γιώργο Γκούμα και την Υποψήφια Διδάκτωρ Νικέλα Παπαδοπούλου για την καθοδήγησή τους και τις πολύτιμες συμβουλές τους κατά την εκπόνηση της παρούσας διπλωματικής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου και τους φίλους μου για την αμέριστη συμπαράσταση και την υποστήριξή τους όλα αυτά τα χρόνια.

Τέλος, θα ήθελα να ευχαριστήσω την εταιρία Nvidia για την πρόσβαση που μου παρείχε στο HPC Cluster της.

Γιώργος Ζαχαριάδης

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγικά

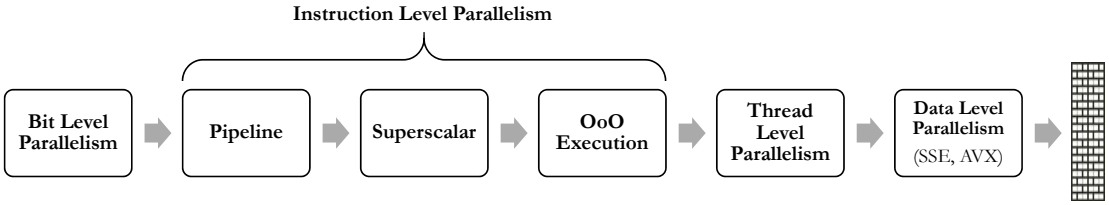
Ο όρος «παράλληλη επεξεργασία» ήταν ανέκαθεν σημαντικός για την επιστήμη των υπολογιστών και συνεχίζει να συγκεντρώνει μεγάλο μέρος της έρευνας. Παλαιότερα, ήταν ταυτισμένος μόνο με μεγάλους υπερυπολογιστές (supercomputers). Στις μέρες μας, έχει αρχίσει να γίνεται όλο και πιο δημοφιλής σε πολλούς τομείς εκτός αυτού της επιστήμης των υπολογιστών. Αιτία για όλα αυτά είναι η ευρύτατη χρήση επεξεργαστών που διαθέτουν πολλαπλούς πυρήνες σε προσωπικούς υπολογιστές και άλλες ηλεκτρονικές συσκευές της καθημερινότητας.

Τα περισσότερα προβλήματα μπορούν να διαιρεθούν σε μικρότερα τμήματα/υποπροβλήματα. Ο βαθμός της διαίρεσης εξαρτάται κυρίως από τη φύση του προβλήματος και των δεδομένων. Αυτό το χαρακτηριστικό εκμεταλλεύεται η παράλληλη επεξεργασία. Προσπαθεί με τη χρησιμοποίηση πολλών επεξεργαστικών μονάδων, που επικοινωνούν μεταξύ τους υπό καθεστώς συνεργασίας, να επιλύσει μεγάλα προβλήματα σε μικρότερο χρόνο [3]. Αποτελεί ένα αρκετά δύσκολο εγχείρημα καθώς έρχεται αντιμέτωπη με πολλαπλά ζητήματα. Από την πλευρά του υλικού (hardware) μερικά από αυτά είναι ο αριθμός, η ισχύς και η πολυπλοκότητα των επεξεργαστικών μονάδων, ο τρόπος με τον οποίο θα επικοινωνούν και θα συνεργάζονται μεταξύ τους. Επίσης, σημαντικό ζήτημα είναι η ενέργεια που καταναλώνουν τόσο οι επεξεργαστικές μονάδες, όσο και τα υπόλοιπα δομικά στοιχεία που απαιτούνται για την αποδοτική επικοινωνία και συνεργασία τους. Φυσικά, όλα τα προηγούμενα από μόνα τους δεν έχουν κανένα νόημα χωρίς τη συμβολή του προγραμματιστή (software). Προκειμένου να εκμεταλλευτεί στο έπακρο το 100% των δυνατοτήτων που του παρέχει το hardware, πρέπει να κατανοήσει τον τρόπο λειτουργίας και συγχρονισμού των μονάδων επεξεργασίας και τον τρόπο που επικοινωνούν με τη μνήμη. Το έργο του για την κατασκευή ενός αποδοτικού παράλληλου προγράμματος παραμένει δύσκολο, παρά τις προόδους που έχουν συντελεστεί τα τελευταία χρόνια.

1.1. Η παραλληλία πριν τους επεξεργαστές πολλαπλών πυρήνων

Η ιδέα του παραλληλισμού προϋπάρχει σχεδόν από την εποχή της ανάπτυξης των πρώτων επεξεργαστών. Χρειάστηκαν πολλά στάδια βελτιστοποιήσεων για την εκμετάλλευση της παραλληλίας σε επίπεδο ενός επεξεργαστικού πυρήνα πριν φτάσουμε στους πολυπύρηνους επεξεργαστές που γνωρίζουμε σήμερα.

Στη συνέχεια, θα αναφερθούν επιγραμματικά τα στάδια αυτά, τα οποία παρουσιάζονται στο *Σχήμα 1.1*.



Σχ. 1.1: Στάδια εκμετάλλευσης παραλληλισμού σε επίπεδο ενός επεξεργαστικού πυρήνα

1.1.1. Παραλληλισμός σε επίπεδο bit (Bit Level Parallelism)

Από τη δεκαετία του '70 μέχρι περίπου τα μέσα της δεκαετίας του '80, η αύξηση των επιδόσεων των επεξεργαστών ερχόταν μέσω της αύξησης του μήκους λέξης (word). Αρχικά, οι επεξεργαστές 4-bit έδωσαν τη θέση τους σε επεξεργαστές 8-bit, οι οποίοι στη συνέχεια αντικαταστάθηκαν από επεξεργαστές 16-bit. Με κάθε διπλασιασμό του μήκους λέξης, μειώνεται ο αριθμός των εντολών που απαιτούνται για τον υπολογισμό της ίδιας πράξης.

Τελικά, στις αρχές του 1980 παρουσιάστηκαν επεξεργαστές με μέγεθος λέξης 32-bit. Παρέμειναν σε αυτό το μέγεθος για αρκετά χρόνια μέχρι τις αρχές του 2000 όπου παρουσιάστηκαν οι πρώτοι επεξεργαστές ευρείας χρήσης με μέγεθος λέξης 64-bit. Υπήρχαν ήδη επεξεργαστές 64-bit που απευθύνονταν κυρίως για επιστημονικές εφαρμογές και μεγάλα υπολογιστικά συστήματα. Πέρα από λόγους επιδόσεων, σημαντική αφορμή για τη μετάβαση από τα 32-bit στα 64-bit ήταν η ανάγκη για μεγαλύτερο χώρο διευθύνσεων καθώς το όριο των 2^{32} $(4\,GB)$ δεν ήταν πια αρκετό. [2]

1.1.2. Παραλληλισμός σε επίπεδο εντολής (Instruction Level Parallelism)

Ένας δεύτερος τρόπος παραλληλισμού που άρχισε να εφαρμόζεται από τα μέσα της δεκαετίας του '80 ήταν σε επίπεδο εντολής (ILP). Η ιδέα είναι να εκτελούνται εντολές παράλληλα, όσο αυτό είναι δυνατό. Υπήρξαν αρκετές βελτιώσεις και καινοτομίες μέχρι να φτάσουμε στους σημερινούς επεξεργαστές.

Τεχνική Σωλήνωσης (Pipelining)

Όταν εισέρχεται μια εντολή στον επεξεργαστή διέρχεται από αρκετά στάδια μέχρι να ολοκληρωθεί η εκτέλεσή της. Καθένα από αυτά τα στάδια απαιτεί ένα κύκλο ρολογιού. Πριν την εφαρμογή της τεχνικής της σωλήνωσης, η εντολή που ήταν προς εκτέλεση σε κάποιο στάδιο του επεξεργαστή μονοπωλούσε και τα υπόλοιπα στάδια μέχρι να ολοκληρωθεί. Συνεπώς, η ταχύτητα ολοκλήρωσης εντολών ανά κύκλο ρολογιού ήταν πάρα πολύ μικρή. Με την τεχνική της σωλήνωσης, στόχος είναι σε κάθε κύκλο ρολογιού να εισέρχεται μια νέα εντολή προς εκτέλεση ώστε ανά πάσα στιγμή να χρησιμοποιούνται όλα τα στάδια του επεξεργαστή (από διαφορετικές εντολές). Με αυτό τον τρόπο, η ταχύτητα ολοκλήρωσης εντολών ανά κύκλο ρολογιού αυξάνεται σημαντικά.

Υπερβαθμωτοί Επεξεργαστές (Superscalar)

Με τον καιρό, όσο εξελισσόταν ο τομέας της τεχνολογίας των υλικών και οι τεχνολογίες κατασκευής ολοκληρωμένων κυκλωμάτων, τόσο αυξανόταν ο αριθμός των τρανζίστορ ανά μονάδα επιφάνειας υλικού. Συνεπώς, κατέστη δυνατή η ενσωμάτωση περισσότερων λειτουργικών μονάδων (functional units) στον επεξεργαστή. Με αυτό τον τρόπο, μπορούσε να εκτελεί περισσότερες από μια εντολές κατά τη διάρκεια ενός κύκλου ρολογιού, εκδίδοντας πολλαπλές εντολές, που είναι ανεξάρτητες μεταξύ τους, στις πολλαπλές λειτουργικές μονάδες. Οι εντολές οι οποίες έχουν εξαρτήσεις μεταξύ τους εκτελούνται σειριακά. Οι επεξεργαστές αυτοί ονομάζονται Υπερβαθμωτοί Επεξεργαστές.

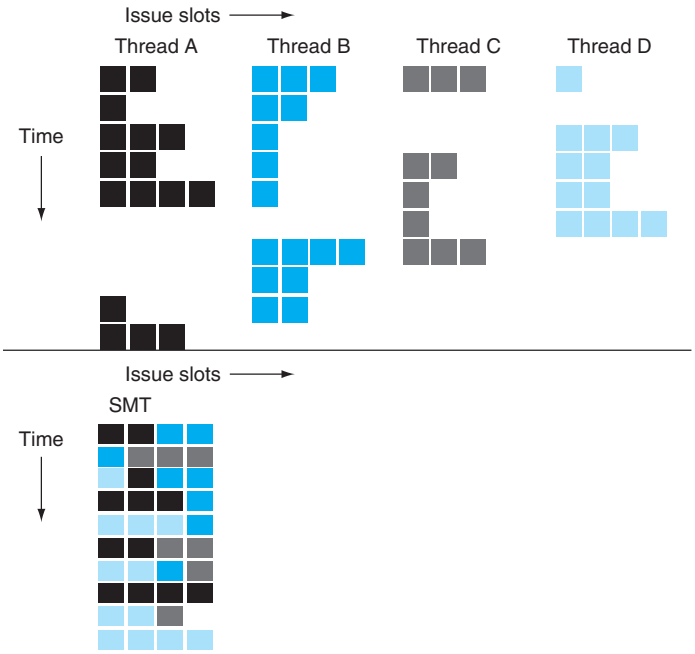
Εκτέλεση εντολών εκτός σειράς (Out-of-Order execution)

Στη συνέχεια, οι επεξεργαστές που βασίζονται σε υπερβαθμωτές αρχιτεκτονικές αρχίζουν να υποστηρίζουν μια νέα καινοτομία, η οποία είναι η εκτέλεση εντολών εκτός σειράς. Οι εντολές συνεχίζουν να εισέρχονται στο pipeline σειριακά (in-order issue) αλλά στη συνέχεια η εκτέλεσή τους παύει να είναι σειριακή και χρησιμοποιείται δυναμική δρομολόγηση (dynamic scheduling). Με αυτό τον τρόπο, αν μια εντολή δεν είναι έτοιμη προς εκτέλεση (π.χ. περιμένει κάποιο όρισμα από την κύρια μνήμη) ενώ μια επόμενη ανεξάρτητη εντολή είναι έτοιμη, τότε εκείνη εκτελείται. Έτσι, αποφεύγεται η αναμονή (stall), που σε αντίθετη περίπτωση θα ήταν αναγκαστική. Εδώ να σημειωθεί ότι, παρά την εκτέλεση εντολών εκτός σειράς, τα αποτελέσματα των εντολών παραδίδονται σειριακά (in-order commit), με την ίδια σειρά με την οποία εισήλθαν οι εντολές στο pipeline.

1.1.3. Παραλληλισμός σε επίπεδο νημάτων (Thread Level Parallelism)

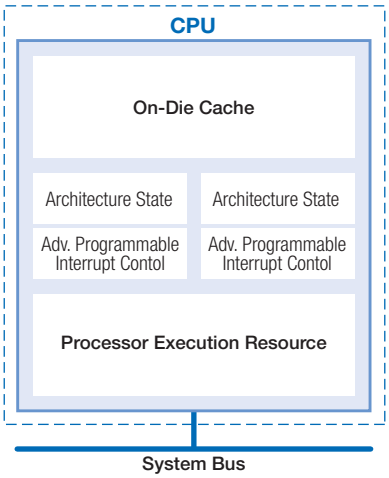
Μετά την καθιέρωση των βελτιστοποιήσεων που αναφέρθηκαν προηγουμένως, η εμπειρία έδειξε ότι, παρά την εκτέλεση εντολών εκτός σειράς και της υποθετικής εκτέλεσης εντολών (speculative execution) με βάση κάποιον προβλέπτη διακλαδώσεων (branch predictor), δεν ήταν δυνατή η πλήρης εκμετάλλευση όλων των πόρων του επεξεργαστή. Ένα νήμα δεν μπορεί να παρέχει τον απαραίτητο παραλληλισμό σε επίπεδο εντολών ώστε να χρησιμοποιούνται όλοι οι επεξεργαστικοί πόροι στο μέγιστο δυνατόν. Σε περιπτώσεις όπως η καθυστέρηση λόγω μεταφοράς δεδομένων από την κύρια μνήμη ή κάποια λανθασμένη πρόβλεψη διακλάδωσης, η αναμονή (stall) είναι αναπόφευκτη.

Συνεπώς, οι προσπάθειες κατευθύνθηκαν στην εκμετάλλευση του παραλληλισμού σε επίπεδο νημάτων (TLP). Η κυριότερη τεχνική είναι η τεχνική SMT (Simultaneous Multithreading). Ο επεξεργαστής μπορεί να εκτελεί ταυτόχρονα περισσότερα του ενός νήματα. Με αυτό τον τρόπο, αν ένα νήμα οδηγηθεί σε κατάσταση αναμονής, τότε μπορεί να εκτελεστεί κάποιο άλλο ώστε να μη μείνουν οι υπολογιστικές μονάδες του επεξεργαστή ανεκμετάλλευτες. Η εναλλαγή των νημάτων πρέπει να γίνεται όσο το δυνατόν πιο γρήγορα.



Σχ. 1.2: Τρόπος εκτέλεσης τεσσάρων ανεξάρτητων νημάτων σε υπερβαθμωτό επεξεργαστή που υποστηρίζει SMT [4]

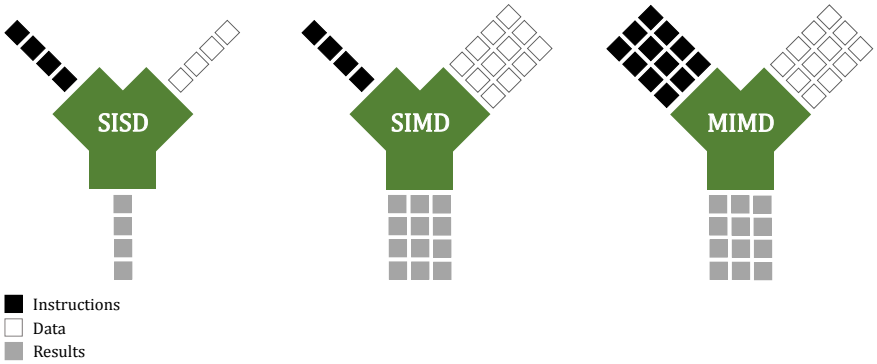
Χαρακτηριστικό παράδειγμα εφαρμογής της τεχνικής του SMT είναι η τεχνολογία της Intel που ονομάζεται Hyper-Threading Technology (HTT). Συγκεκριμένα τμήματα του επεξεργαστή υπάρχουν σε παραπάνω από ένα αντίτυπα (όσα και τα νήματα που υποστηρίζει) ώστε να διατηρείται η αρχιτεκτονική κατάσταση (architectural state) κάθε νήματος και να γίνεται γρήγορα η εναλλαγή μεταξύ τους, όπως φαίνεται στο *Σχήμα 1.3*. Πρωτοπαρουσιάστηκε το 2002 αρχικά στους επεξεργαστές της σειράς Xeon και στη συνέχεια στους επεξεργαστές της σειράς Pentium 4 υποστηρίζοντας 2 νήματα. Μετέπειτα, χρησιμοποιήθηκε σε αρκετούς επεξεργαστές της εταιρίας. Η αύξηση των επιδόσεων που προσφέρει εξαρτάται από το είδος των προγραμμάτων που εκτελούνται και φτάνει μέχρι και το 30% σε συγκεκριμένα σενάρια. [5]



Σχ. 1.3: Αρχιτεκτονική επεξεργαστή που υποστηρίζει Hyper-Threading Technology [5]

1.1.4. Παραλληλισμός σε επίπεδο δεδομένων (Data Level Parallelism)

Ένας άλλος τρόπος αύξησης της επεξεργαστικής ισχύος των επεξεργαστών αποτελεί ο παραλληλισμός σε επίπεδο δεδομένων, δηλαδή η εκτέλεση στον ίδιο κύκλο ρολογιού της ίδιας εντολής σε περισσότερα δεδομένα. Αυτή η τεχνική προϋπάρχει από τα πρώτα χρόνια ανάπτυξης των υπολογιστών και χρησιμοποιείται σε ειδικούς επεξεργαστές επιστημονικού σκοπού που ονομάζονται Vector Processors ή ελληνιστί Διανυσματικοί Επεξεργαστές.



Σχ. 1.4: Αρχιτεκτονικές SISD, SIMD και MIMD

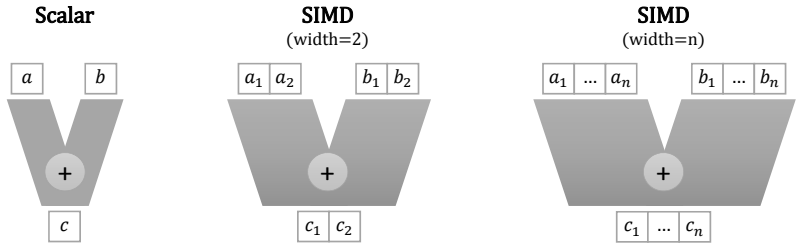
Σε αυτό το σημείο κρίνεται απαραίτητο να αναφερθεί ο διαχωρισμός των αρχιτεκτονικών υπολογιστών με βάση την ταξινόμηση του Flynn (Flynn's taxonomy) [11]. Κριτήριο της ταξινόμησης αποτελεί ο αριθμός των ταυτόχρονων ροών (streams) εντολών και των ταυτόχρονων ροών δεδομένων. Αναλυτικότερα:

- SISD (Single Instruction, Single Data): Αρχιτεκτονικές στις οποίες δεν υπάρχει κάποιο επίπεδο παραλληλισμού και όλα γίνονται σειριακά καθώς υπάρχει μόνο μια σειριακή ροή εντολών η οποία εφαρμόζεται σε μία σειριακή ροή δεδομένων.
- SIMD (Single Instruction, Multiple Data): Αρχιτεκτονικές στις οποίες υπάρχει μόνο μια σειριακή ροή εντολών αλλά η οποία εφαρμόζεται ταυτόχρονα σε πολλές ροές δεδομένων. Είναι εξαιρετικά αποδοτικές σε προβλήματα που απαιτούν επεξεργασία μεγάλου όγκου δεδομένων. Σε αυτή την κατηγορία ανήκουν ορισμένες σημαντικές εντολές της x86 αρχιτεκτονικής οι οποίες εκμεταλλεύονται τον παραλληλισμό σε επίπεδο δεδομένων, οι κάρτες γραφικών και οι αρχιτεκτονικές διανυσματικών επεξεργαστών.
- MISD (Multiple Instruction, Single Data): Αρχιτεκτονικές στις οποίες υπάρχουν ταυτόχρονα πολλές διαφορετικές ροές εντολών οι οποίες εφαρμόζονται σε μία μόνο ροή δεδομένων. Δεν υπάρχει κάποια εμπορική αρχιτεκτονική που να εμπίπτει σε αυτή την κατηγορία, αν και ορισμένοι τοποθετούν εδώ τους διανυσματικούς επεξεργαστές. Τέτοιες αρχιτεκτονικές έχουν χρησιμοποιηθεί μόνο συστήματα όπου η ανοχή σφαλμάτων (fault tollerance) είναι σημαντική καθώς οι ίδιες εντολές εκτελούνται σε διαφορετικά συστήματα και συγκρίνονται τα αποτελέσματα.
- MIMD (Multiple Instruction, Multiple Data): Αρχιτεκτονικές στις οποίες υπάρχουν ταυτόχρονα πολλές διαφορετικές ροές εντολών οι οποίες εφαρμόζονται σε πολλές διαφορετικές ροές δεδομένων. Στην κατηγορία αυτή ανήκουν τα περισσότερα σύγχρονα συστήματα παράλληλων

υπολογιστών, δηλαδή οι πολυπύρηνοι επεξεργαστές, τα συστήματα πολλαπλών επεξεργαστών καθώς και οι συστοιχίες των υπερυπολογιστών.

x86 SIMD Extensions

Η εκμετάλλευση του παραλληλισμού σε επίπεδο δεδομένων σε εμπορικούς επεξεργαστές γενικής χρήσης γίνεται με την επέκταση του σετ εντολών x86 (x86 ISA) και την προσθήκη νέων SIMD εντολών, δηλαδή διανυσματικών εντολών που ακολουθούν τη φιλοσοφία να εφαρμόζονται ταυτόχρονα σε περισσότερα δεδομένα. Παράδειγμα εκτέλεσης μιας πράξης πρόσθεσης από SIMD εντολές ακολουθεί στο *Σχήμα 1.5*.



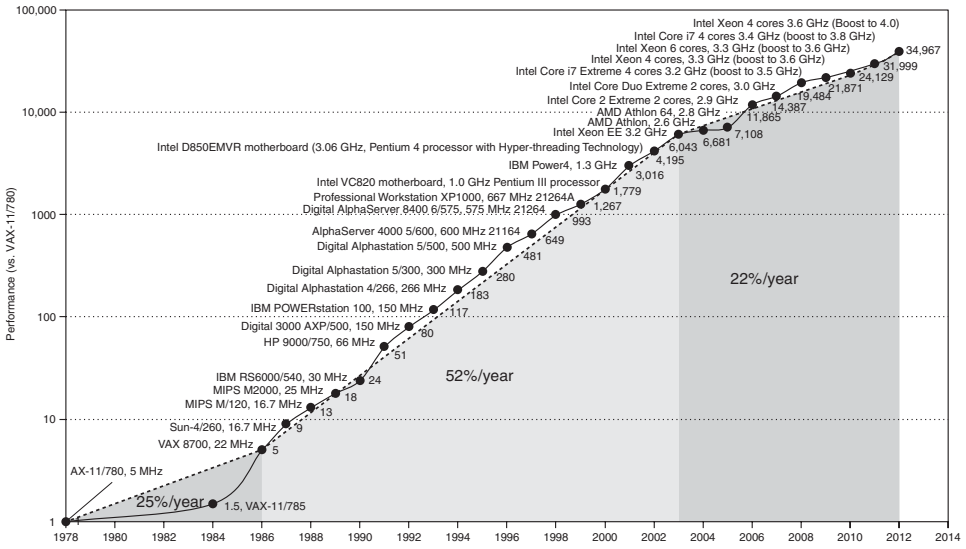
Σχ. 1.5: Εκτέλεση της ίδιας πράξης από Scalar και SIMD αρχιτεκτονικές

Η πρώτη προσπάθεια έγινε από την Intel το 1997 όταν παρουσίασε το σετ εντολών MMX στους επεξεργαστές Pentium. Τα μειονεκτήματα των εντολών αυτών ήταν ότι εφαρμόζονταν μόνο σε ακέραια δεδομένα και όχι κινητής υποδιαστολής ενώ ταυτόχρονα δεν εισήγαγαν νέους καταχωρητές αλλά χρησιμοποιούσαν τους ήδη υπάρχοντες καταχωρητές για πράξεις κινητής υποδιαστολής. Για να αντιμετωπιστούν αυτά τα προβλήματα, το 1999 παρουσιάστηκε από την Intel και την AMD το πακέτο επέκτασης εντολών SSE (Streaming SIMD Extensions), το οποίο αρχικά λειτουργούσε για πράξεις κινητής υποδιαστολής και χρησιμοποιούσε ξεχωριστούς καταχωρητές. Για τους λόγους αυτούς έγινε αρκετά πιο δημοφιλές από το πακέτο MMX. Στη διάρκεια των ετών παρουσιάστηκαν αρκετές νέες επεκτάσεις (SSE2, SSE3, SSSE3, SSE4) οι οποίες προσέθεσαν περισσότερες εντολές και μεγαλύτερους καταχωρητές 128 bit.

Στις αρχές του 2011, η Intel παρουσίασε το πακέτο επέκτασης εντολών AVX (Advanced Vector Extensions). Αποτελεί εξέλιξη των επεκτάσεων SSE και διπλασιάζει το μέγεθος των ειδικών καταχωρητών σε 256 bit. Έτσι, μπορεί να γίνει ταυτόχρονη επεξεργασία τεσσάρων αριθμών κινητής υποδιαστολής διπλής ακρίβειας (double precision, 64 bit) και οκτώ αριθμών απλής ακρίβειας (single precision, 32 bit). Επιπλέον, εισάγει εντολές που δέχονται 3 ορίσματα αντί για 2.

Τέλος, στους επεξεργαστές αρχιτεκτονικής ARM, που στις μέρες μας γίνονται όλο και πιο δημοφιλείς, υποστηρίζονται SIMD εντολές από το πακέτο επέκτασης εντολών NEON (ή αλλιώς Advanced SIMD extension).

1.2. Αιτίες της στροφής προς τους επεξεργαστές πολλαπλών πυρήνων



Σχ. 1.6: Αύξηση της επίδοσης των επεξεργαστών από τα μέσα της δεκαετίας του 1980 έως σήμερα (με βάση το μετροπρόγραμμα SPECint) [4]

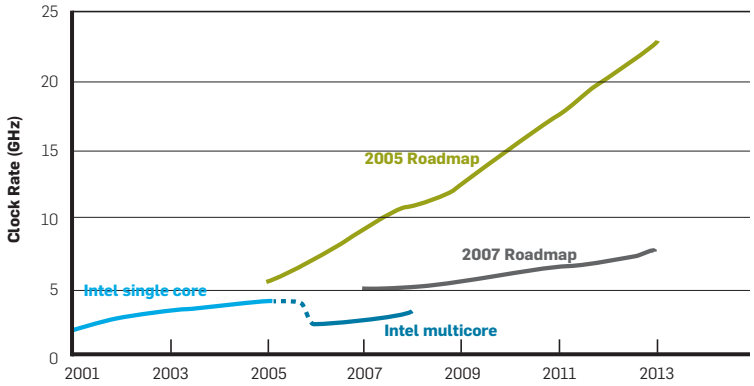
Μέχρι περίπου το 2003, οι καινοτομίες και οι βελτιστοποιήσεις που αναφέρθηκαν παραπάνω, καθώς και οι εξελίξεις στον τομέα της κατασκευής ολοκληρωμένων κυκλωμάτων, οδηγούσαν κάθε χρόνο σε σημαντική αύξηση των επιδόσεων των επεξεργαστών. Ειδικότερα, όπως φαίνεται στο *Σχήμα 1.6*, η ετήσια αύξηση της επεξεργαστικής ισχύος άγγιζε κατά μέσο όρο το 52% και παρέμεινε σταθερή για παραπάνω από 2 δεκαετίες. Οι προβλέψεις ανέφεραν ότι ο συγκεκριμένος ρυθμός αύξησης θα συνεχιζόταν και τα επόμενα χρόνια. Τελικά, αυτό διαψεύστηκε και μέχρι σήμερα περιορίστηκε σε περίπου 22% ανά έτος κατά μέσο όρο.

Η κύρια αιτία αυτής της μείωσης είναι ότι οι επιδόσεις σε επίπεδο ενός επεξεργαστικού πυρήνα δεν κατάφεραν να διατηρήσουν τον υψηλό ρυθμό εξέλιξης. Κατά συνέπεια, οι κατασκευαστές οδηγήθηκαν στην ενσωμάτωση περισσότερων του ενός επεξεργαστικών πυρήνων, εγκαινιάζοντας την εποχή των «multicore» αρχιτεκτονικών. Οι λόγοι είναι αρκετοί και συνοψίζονται σε τρεις κατηγορίες: τα προβλήματα στην κλιμάκωση της έκλυσης ισχύος (power wall), οι δυσκολίες στην περαιτέρω εκμετάλλευση του παραλληλισμού σε επίπεδο εντολής (ILP wall) και η μικρή αύξηση της ταχύτητας της κύριας μνήμης (memory wall).

1.2.1. Power wall

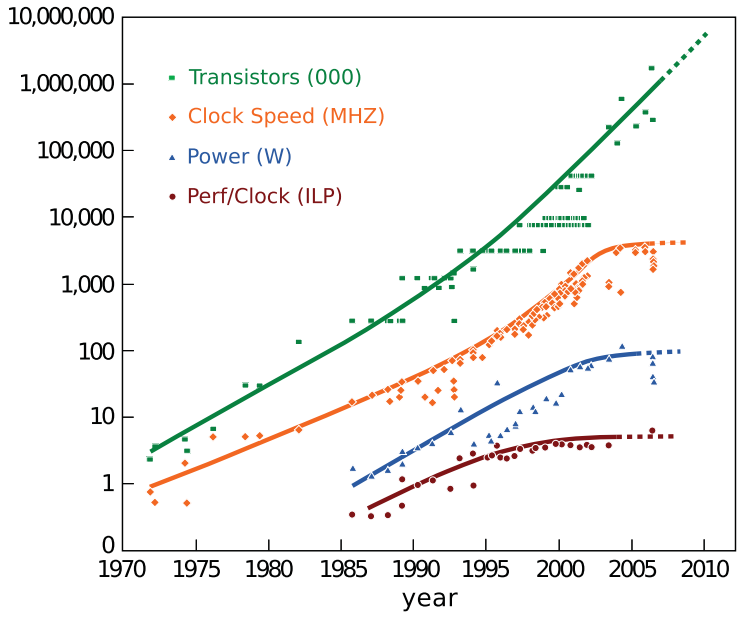
Αρχικά, η κατανάλωση των πρώτων επεξεργαστών κυμαινόταν σε μόλις μερικά Watt, ενώ οι τεχνολογίες ολοκλήρωσης βρίσκονταν σε πολύ πρώιμο στάδιο. Η κυρίαρχη τεχνολογία κατασκευής ολοκληρωμένων κυκλωμάτων ονομάζεται CMOS και χρησιμοποιείται μέχρι και σήμερα. Η κατανάλωση ενέργειας ενός επεξεργαστή είναι ανάλογη της συχνότητας λειτουργίας πολλαπλασιασμένης με την τάση λειτουργίας υψωμένη στο τετράγωνο. Η εξέλιξη της συγκεκριμένης τεχνολογίας μειώνει συνεχώς το μέγεθος των τρανζίστορ και την τάση λειτουργίας τους και συνεπώς την ενέργεια που καταναλώνουν. Έτσι, οι κατασκευαστές επεξεργαστών μπορούσαν

να ενσωματώνουν όλο και περισσότερα τρανζίστορ στο ίδιο εμβαδόν υλικού και ταυτόχρονα να αυξάνουν τη συχνότητα του ρολογιού. Τελικά, από γενιά σε γενιά η συνολική κατανάλωση μεγάλωνε αλλά το ίδιο έκαναν και οι επιδόσεις. Η τεχνική αυτή αποτέλεσε τον έναν από τους δύο βασικούς πυλώνες στους οποίους στηρίχτηκε η μεγάλη αύξηση των επιδόσεων των επεξεργαστών.



Σχ. 1.7: Συχνότητες επεξεργαστών της Intel σε σχέση με τις προβλέψεις των εκδόσεων του 2005 και του 2007 του International Technology Roadmap for Semiconductors [7]

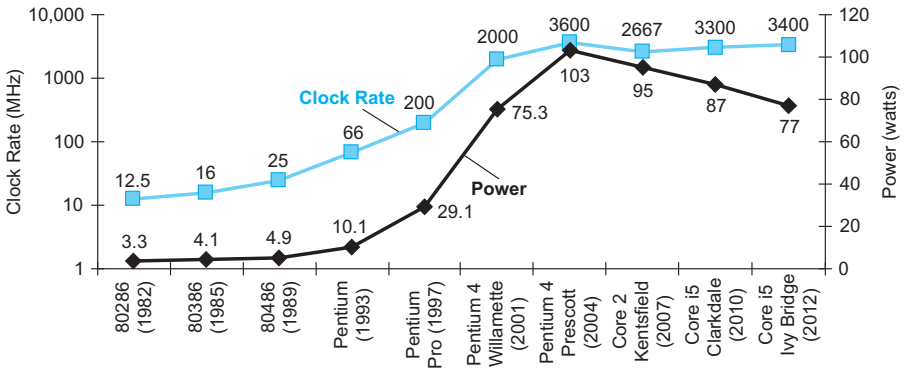
Στις αρχές του 2000, οι κατασκευαστές οραματίζονταν επεξεργαστές με μεγάλες συχνότητες λειτουργίας, ακόμα και διψήφιες. Όπως φαίνεται στο *Σχήμα 1.7*, οι προβλέψεις του 2005, με τα δεδομένα της εποχής, ανέφεραν ότι μέχρι το 2008 οι συχνότητες λειτουργίας θα έχουν ξεπεράσει τα $10\,GHz$, κάτι το οποίο δεν έγινε ποτέ. Ακόμα και οι μετριοπαθείς προβλέψεις του 2007 απέχουν αρκετά από τις συχνότητες που τελικά επιτεύχθηκαν.



Σχ. 1.8: Διαγράμμα αριθμού τρανζίστορ, συχνότητας λειτουργίας, κατανάλωσης ενέργειας και επιδόσεων ανά κύκλο ρολογιού (ILP)

Όπως φαίνεται στο *Σχήμα 1.8*, ο ρυθμός αύξησης της καταναλισκόμενης ενέργειας συμπίπτει με τον ρυθμό αύξησης της συχνότητας λειτουργίας. Αρχικά, η κατανάλωση ενέργειας βρισκόταν σε χαμηλά επίπεδα και συνεπώς υπήρχαν μεγάλα περιθώρια ανόδου. Όμως, όσο αυξανόταν η συνολική κατανάλωση ενέργειας, τόσο δυσκολότερη γινόταν η απαγωγή της εκλυόμενης θερμότητας. Επίσης, ο ρυθμός μείωσης της ενέργειας που καταναλώνουν τα τρανζίστορ δεν ήταν αρκετός ώστε να αντισταθμίσει την αύξηση της συχνότητας του ρολογιού και του πλήθους των τρανζίστορ. Αυτό είχε ως αποτέλεσμα η ψύξη των επεξεργαστών να αποτελέσει πολύ σημαντικό πρόβλημα στην περαιτέρω κλιμάκωση της τεχνικής αυτής, ειδικά όταν η κατανάλωση ισχύος ανήλθε σε τριψήφιους αριθμούς (σε Watt).

Τότε ήταν που οι κατασκευαστές συνειδητοποίησαν πως η συχνότητα λειτουργίας δεν ήταν δυνατόν να βελτιωθεί παραπάνω και συνεπώς έπρεπε να βρεθεί άλλος τρόπος ενίσχυσης των επιδόσεων. Η φιλοσοφία που επικράτησε στηρίζεται στο ότι είναι προτιμότερο να υπάρχουν περισσότεροι επεξεργαστικοί πυρήνες χαμηλότερα χρονισμένοι παρά λιγότεροι με υψηλότερους χρονισμούς. Όταν σταμάτησε να αυξάνεται η συχνότητα του ρολογιού και η κατανάλωση ενέργειας, τότε άρχιζαν να παρουσιάζονται επεξεργαστές με παραπάνω από έναν πυρήνες. Επιπλέον, αξίζει να σημειωθεί ότι παρά την αλλαγή της φιλοσοφίας ως προς τον τρόπο αύξησης των επιδόσεων, ο νόμος του Moore[1] συνεχίζει να ισχύει.



Σχ. 1.9: Συχνότητες λειτουργίας και κατανάλωση ενέργειας επεξεργαστών της Intel από το 1982 έως το 2012 [4]

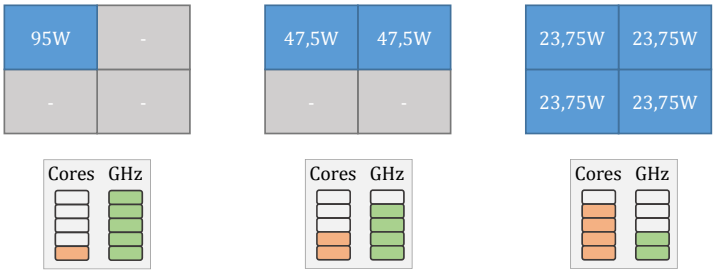
Πλέον, οι κατασκευαστές εκμεταλλεύονται τις προόδους των τεχνολογιών ολοκλήρωσης ώστε να ενσωματώνουν περισσότερους επεξεργαστικούς πυρήνες και μεγαλύτερες κρυφές μνήμες (cache). Ταυτόχρονα, προσπαθούν να βελτιστοποιούν συνεχώς την αποδοτικότητα των επεξεργαστών ως προς την κατανάλωση ενέργειας (power efficiency) με στόχο τη μείωση της κατανάλωσης αλλά ταυτόχρονα τη διατήρηση της συχνότητας λειτουργίας όσο το δυνατόν υψηλότερα. Αυτό γίνεται εμφανές στο *Σχήμα 1.9*. Η Intel, μετά τα σημαντικά προβλήματα θερμικής συμπεριφοράς που αντιμετώπισε με την αρχιτεκτονική Prescott, αναγκάστηκε να εγκαταλείψει τη σειρά επεξεργαστών Pentium 4 και να οδηγηθεί στη γενιά Core 2. Η αλλαγή σηματοδοτούσε τη στροφή προς ένα απλούστερο και αποδοτικότερο pipeline και την ενσωμάτωση περισσότερων

[1] Ο νόμος του Moore (συνιδρυτή της εταιρίας Intel) αναφέρει ότι ο αριθμός των τρανζίστορ στους μικροεπεξεργαστές θα διπλασιάζεται κάθε 2 χρόνια. Αργότερα, η πρόβλεψη αυτή τροποποιήθηκε σε 18 μήνες.

πυρήνων χαμηλότερα χρονισμένων. Η φιλοσοφία αυτή διατηρήθηκε και στις επόμενες γενιές Core με έμφαση σε περαιτέρω βελτιώσεις ενεργειακής αποδοτικότητας.

Turbo Boost

Σημαντική καινοτομία αποτελεί η εμφάνιση από την Intel της τεχνολογίας Intel Turbo Boost η οποία παρουσιάστηκε για πρώτη φορά στα τέλη του 2008 στην αρχιτεκτονική Nehalem. Η τεχνολογία αυτή αφορά τη δυναμική μεταβολή του χρονισμού των πυρήνων του επεξεργαστή σε πραγματικό χρόνο ανάλογα με το επεξεργαστικό φορτίο. Βασικό κριτήριο για το ανώτατο όριο αύξησης της συχνότητας αποτελεί η κατανάλωση ενέργειας η οποία δεν πρέπει ποτέ να ξεπεράσει τις προδιαγραφές του επεξεργαστή (power envelope). Επιπρόσθετα, συνυπολογίζονται τα μέγιστα όρια θερμοκρασίας και ρεύματος που μπορεί να αντλήσει ο επεξεργαστής.



Σχ. 1.10: Σχηματική αναπαράσταση της λειτουργίας του Turbo Boost σε έναν τετραπύρηνο επεξεργαστή με 95W ανώτατη προδιαγραφή κατανάλωσης (TDP)

Αρχικά, σκοπός ήταν να βελτιωθεί η επίδοση εφαρμογών που δεν εκμεταλλεύονται παραπάνω από έναν πυρήνα. Αυτό επιτυγχάνεται με αύξηση της συχνότητα ενός μόνο πυρήνα παραπάνω από τη βασική συχνότητα λειτουργίας και ταυτόχρονα η μείωση της συχνότητας (ή και απενεργοποίηση) των υπόλοιπων πυρήνων ώστε να αντισταθμιστεί η αυξημένη κατανάλωση. Στη συνέχεια, επεκτάθηκε ώστε να λειτουργεί για οποιοδήποτε υποσύνολο ενεργών πυρήνων, όπως φαίνεται στη σχηματική αναπαράσταση του *Σχήματος 1.10*.

1.2.2. ILP wall

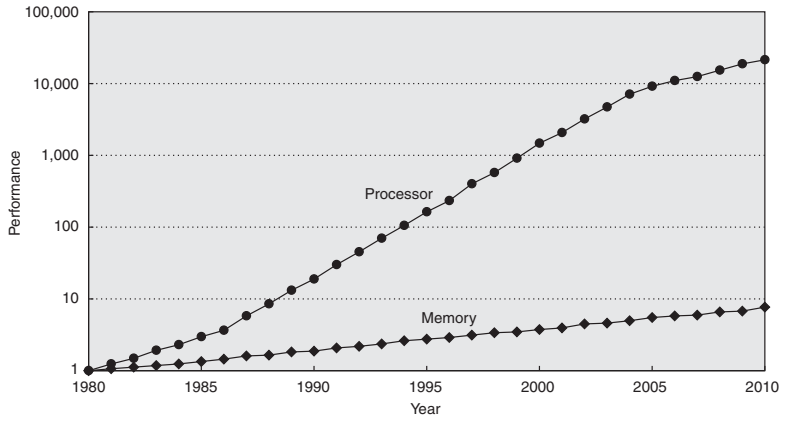
Ο δεύτερος πυλώνας στον οποίο στηρίχτηκε η εξέλιξη των επεξεργαστών τα προηγούμενα χρόνια είναι η εκμετάλλευση του παραλληλισμού σε επίπεδο εντολής (ILP). Με τη συνεχή βελτιστοποίηση των τεχνικών που αναλύθηκαν στην *Υποενότητα 1.1.2*, οι επεξεργαστές προσέφεραν όλο και μεγαλύτερες επιδόσεις από γενιά σε γενιά. Όπως φαίνεται ξεκάθαρα στο διάγραμμα του *Σχήματος 1.8*, μέχρι τις αρχές του 2000 παρατηρείται συνεχής και εντυπωσιακή αύξηση του ILP. Μέχρι τότε, η κατανάλωση ενέργειας δεν είχε φτάσει σε απαγορευτικά επίπεδα, οπότε υπήρχε δυνατότητα να αναπτυχθούν περαιτέρω οι τεχνικές που απαιτούν επιπλέον υλικό για να υλοποιηθούν. Ειδικότερα, η εκτέλεση εντολών εκτός σειράς (OoO Execution) απαιτεί σημαντικό αριθμό επιπλέον τρανζίστορ ώστε να υλοποιηθούν οι απαραίτητες δομές. Συνεπώς, πρέπει να υπάρχουν περιθώρια τόσο σε επίπεδο υλικού όσο και σε επίπεδο κατανάλωσης ισχύος.

Όλες οι τεχνικές, όταν βρίσκονται σε πρώιμα στάδια, είναι πιο εύκολο να αναπτυχθούν και να εξελιχθούν. Όσο όμως προχωράει η βελτιστοποίηση τους, γίνεται συνεχώς πιο δύσκολο και πιο

δαπανηρό να διατηρηθούν οι αρχικοί ρυθμοί ανάπτυξης. Από κάποιο σημείο και έπειτα, είτε είναι αδύνατη η περαιτέρω εξέλιξη λόγω αρκετών παραγόντων (π.χ. εξάντληση διαθέσιμων πόρων), είτε η εξέλιξη είναι πολύ μικρή για να αντισταθμίσει το κόστος που απαιτείται. Στις αρχές του 2000, αυτό συνέβη με τις περισσότερες τεχνικές που αναφέρθηκαν, οπότε ανακόπτονται οι μεγάλοι ρυθμοί αύξησης του ILP των προηγούμενων ετών. Από τότε μέχρι σήμερα οι επιπλέον βελτιώσεις είναι μικρότερες σε έκταση και κυρίως επωφελούνται της προόδου στον τομέα της βελτιωμένης ενεργειακής απόδοσης.

1.2.3. Memory wall

Η τρίτη αιτία που αποτελεί εμπόδιο στην εξέλιξη των επεξεργαστών είναι το γεγονός ότι η ταχύτητα της κύριας μνήμης (RAM) δεν μπόρεσε να ακολουθήσει τους εκθετικούς ρυθμούς αύξησης των επιδόσεων των επεξεργαστών. Έτσι, το κενό επιδόσεων ανάμεσα σε μνήμη και επεξεργαστή αύξανε με εκθετικό ρυθμό και συνεχίζει να αυξάνει μέχρι σήμερα. Με την εκμετάλλευση των τεχνολογιών ολοκλήρωσης, οι κατασκευαστές προσπαθούν να αντισταθμίσουν αυτό το πρόβλημα ενσωματώνοντας συνεχώς μεγαλύτερες κρυφές μνήμες (cache) οργανωμένες σε πολλαπλά επίπεδα.



Σχ. 1.11: Διαφορά επιδόσεων (performance gap) μεταξύ μνήμης RAM και επεξεργαστών ξεκινώντας από το 1980 [10]

Επιπλέον, όσο ο αριθμός των πυρήνων στους επεξεργαστές αυξάνεται, τόσο το πρόβλημα γίνεται εντονότερο καθώς πρέπει να μοιραστούν τον ίδιο «δρόμο» προς την μνήμη. Ειδικά σε εφαρμογές που απαιτούν πολλές και ακανόνιστες προσβάσεις στη μνήμη για να μεταφέρουν τα δεδομένα προς επεξεργασία, η ταχύτητα της μνήμης είναι το συχνότερο σημείο που εμφανίζεται συμφόρηση (bottleneck) και εν τέλει ο καθοριστικός παράγοντας που προσδιορίζει τις επιδόσεις που επιτυγχάνονται.

1.3. Από τις Multicore στις Manycore αρχιτεκτονικές

Στις μέρες μας, οι επεξεργαστές που απευθύνονται προς το ευρύ καταναλωτικό κοινό αριθμούν το πολύ μέχρι 6 ή 8 επεξεργαστικούς πυρήνες. Αντιθέτως, οι επεξεργαστές που προορίζονται για διακομιστές και συστήματα υψηλών επιδόσεων (High Performance Computing, HPC) ενσωματώνουν σημαντικά μεγαλύτερο αριθμό επεξεργαστικών πυρήνων και μεγαλύτερες κρυφές μνήμες. Σε αυτό βοήθησε η πρόοδος στις τεχνολογίες ολοκλήρωσης και η συνεχής εξέλιξη των καινοτομιών μείωσης και εξοικονόμησης ενέργειας.

	Αρ. Πυρήνων	**L3 Cache**	**TDP**	**Μεθ. Ολοκλ.**
Intel Core i7 Haswell	έως 4	8 MB	84 W	22 nm
Intel Core i7 Ivy Bridge-E	έως 6	15 MB	130 W	22 nm
Intel Xeon E5 v2	έως 12	30 MB	έως 115 W	22 nm
Intel Xeon E7 v2	έως 15	37.5 MB	έως 155 W	22 nm
AMD Opteron Piledriver	έως 16	16 MB	έως 140 W	32 nm

Πίν. 1.1: Προδιαγραφές σύγχρονων x86 επεξεργαστών [12, 13]

Οι απαιτήσεις για μεγαλύτερη επεξεργαστική ισχύ ήταν πάντα και είναι ακόμα μεγάλες, ειδικά από το HPC κομμάτι της αγοράς. Πριν από μια πενταετία, οι επεξεργαστές δεν αριθμούσαν το σημαντικό αριθμό επεξεργαστικών πυρήνων που φαίνεται στο *Σχήμα 1.1* αλλά αρκετά λιγότερους (μέχρι 4). Επιπλέον, η ταχύτητα της μνήμης (όπως αναφέρθηκε στην *Υποενότητα 1.2.3*) περιόριζε σημαντικά τις επιδόσεις και δεν επέτρεπε την πλήρη αξιοποίηση της υπολογιστικής δύναμης των επεξεργαστών. Η αναζήτηση περισσότερης επεξεργαστικής ισχύος οδήγησε στην εκμετάλλευση μιας υπολογιστικής συσκευής που μέχρι εκείνη τη στιγμή, αν και ισχυρή από επεξεργαστική άποψη, είχε χρήση μόνο στον τομέα των 3D γραφικών. Αυτή η συσκευή ήταν η κάρτα γραφικών (Graphics Processing Unit, GPU).

1.3.1. Κάρτες Γραφικών

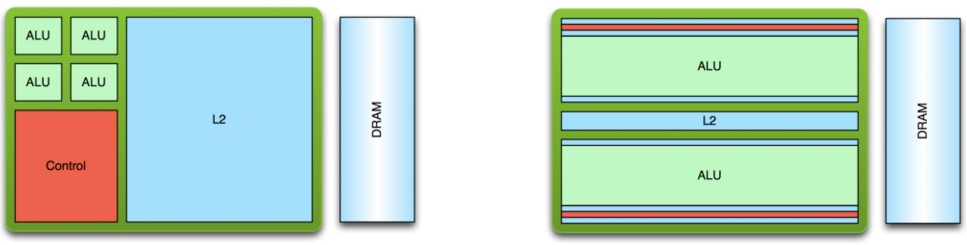
Η ανάγκη για ρεαλιστικότερα τρισδιάστατα γραφικά υψηλής ανάλυσης οδηγούσε σε σημαντική αύξηση της υπολογιστικής ισχύος των καρτών γραφικών από γενιά σε γενιά. Παρά ταύτα, οι κάρτες γραφικών δεν μπορούσαν να χρησιμοποιηθούν για άλλους υπολογισμούς γενικού σκοπού αλλά μόνο για συγκεκριμένες πράξεις ειδικού σκοπού που είναι απαραίτητες στο κομμάτι της 3D γραφικής απεικόνισης. Αυτό άλλαξε το 2008 όταν κυκλοφόρησε από την εταιρία Nvidia η πρώτη γενικού σκοπού προγραμματιζόμενη κάρτα γραφικών ακολουθώντας την αρχιτεκτονική Nvidia CUDA (Compute Unified Device Architecture). Ήταν η αρχή της σημαντικής ανάπτυξης που γνωρίζει ο τομέας των υπολογισμών γενικού σκοπού σε κάρτες γραφικών ή αλλιώς GPGPU (General Purpose computing on GPUs). Ενσωμάτωνε για πρώτη φορά προγραμματιζόμενους

vertex shaders και παρείχε ένα κατάλληλο προγραμματιστικό μοντέλο και τα απαραίτητα εργαλεία (CUDA SDK) για την ανάπτυξη εφαρμογών που θα εκτελούνταν όχι σε κάποιον επεξεργαστή αλλά σε κάρτες γραφικών.

Οι πρώτες GPUs γενικού σκοπού είχαν αρκετούς περιορισμούς, μειωμένη ακρίβεια πράξεων και επιπλέον τα εργαλεία ανάπτυξης εφαρμογών ήταν αρκετά δύστροπα για τον προγραμματιστή με αρκετές «αγκυλώσεις». Από τότε όμως οι κάρτες γραφικών γνωρίζουν σημαντική εξέλιξη και από γενιά σε γενιά παρουσιάζονται συνεχείς βελτιώσεις που αφορούν τόσο το κομμάτι των επιδόσεων όσο και το κομμάτι της ευκολίας προγραμματισμού. Στον τομέα του GPGPU εισήλθε και η δεύτερη μεγάλη εταιρία κατασκευής καρτών γραφικών ATI (πλέον AMD) παρουσιάζοντας αρχικά το Stream SDK. Στη συνέχεια παρουσιάστηκε το OpenCL (Open Computing Language) το οποίο αποτελεί ένα framework για την ανάπτυξη εφαρμογών που εκτελούνται σε ετερογενείς αρχιτεκτονικές, ανάμεσα στις οποίες είναι και οι κάρτες γραφικών. Κυρίως υποστηρίζεται από τις κάρτες γραφικών της AMD, ενώ πλέον και από τις κάρτες της Nvidia. Το OpenCL μαζί με το CUDA αποτελούν τις δύο πιο δημοφιλείς πλατφόρμες ανάπτυξης εφαρμογών για GPUs, με τη διαφορά ότι το CUDA είναι πλατφόρμα κλειστού κώδικα και συμβατή μόνο με κάρτες γραφικών της Nvidia. Άλλη λιγότερο δημοφιλής εναλλακτική είναι το DirectCompute της εταιρίας Microsoft.

CPU vs GPU

Η πρώτη μεγάλη διαφορά φιλοσοφίας ανάμεσα σε επεξεργαστές και κάρτες γραφικών είναι ότι οι κάρτες γραφικών αποτελούνται από πολύ μικρότερους και απλούστερους επεξεργαστικούς πυρήνες σε σχέση με τους επεξεργαστές, με αποκλειστικό προορισμό την εκτέλεση υπολογιστικών πράξεων. Ανήκουν στην κατηγορία των «manycore» αρχιτεκτονικών, δηλαδή αρχιτεκτονικών που αποτελούνται από δεκάδες, εκατοντάδες ή χιλιάδες απλούς επεξεργαστικούς πυρήνες, οι οποίοι ουσιαστικά ενσωματώνουν μόνο αριθμητικές μονάδες εκτέλεσης πράξεων (ALUs). Εκμεταλλεύονται τα οφέλη της λογικής των SIMD αρχιτεκτονικών στο έπακρο, προσφέρουν πολύ μεγαλύτερη υπολογιστική ισχύ από τους επεξεργαστές και δεν έχουν ανάγκη τα πολύπλοκα front end που χρειάζονται οι επεξεργαστές για την αποκωδικοποίηση των εντολών, την πρόβλεψη διακλαδώσεων, την εκτέλεση εντολών εκτός σειράς κ.ά. Οι απλοί πυρήνες καταναλώνουν σημαντικά λιγότερη ενέργεια ανά πυρήνα και καταλαμβάνουν πολύ μικρή επιφάνεια υλικού στο chip. Συνεπώς είναι δυνατή η ενσωμάτωση πολύ μεγάλου αριθμού, διατηρώντας την κατανάλωση ενέργειας σε λογικά πλαίσια.

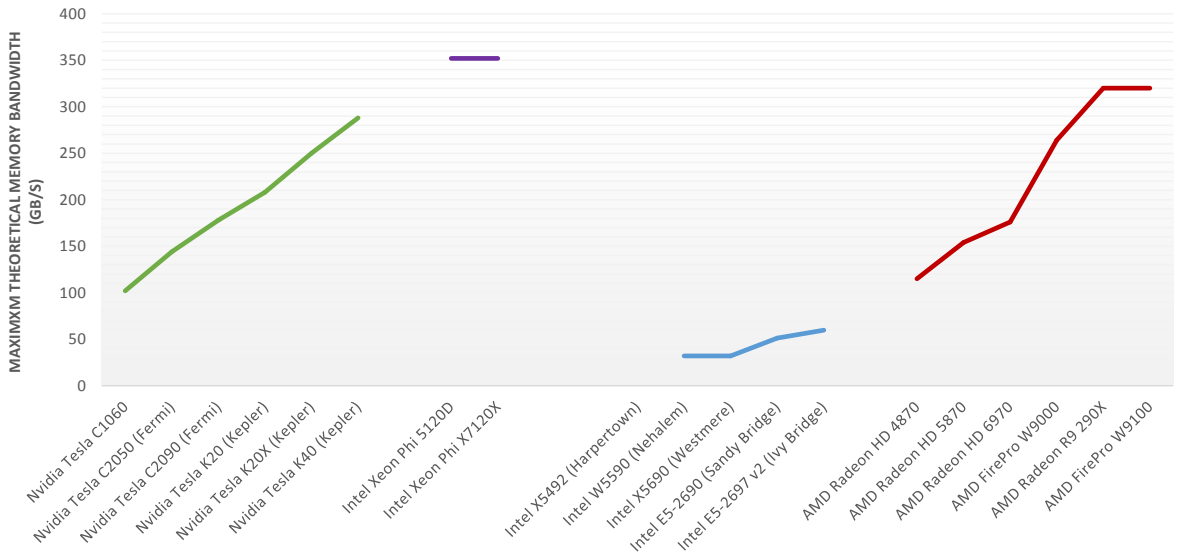


(a) CPU (b) GPU

Σχ. 1.12: Διαφορετική φιλοσοφία χρήσης της διαθέσιμης επιφάνειας των chip ανάμεσα σε επεξεργαστές και κάρτες γραφικών

Η δεύτερη διαφορά είναι ο τρόπος με τον οποίο αντιμετωπίζουν το πρόβλημα της αργής πρόσβασης στη μνήμη (memory latency) και μοιραία του bottleneck που δημιουργεί. Οι επεξεργαστές προσπαθούν να «κρύψουν» το latency των προσβάσεων στη μνήμη ενσωματώνοντας μεγάλες και πολυεπίπεδες μνήμες cache, οι οποίες καταναλώνουν τη μερίδα του λέοντος της επιφάνειας του chip και αρκετή ενέργεια. Αντίθετα, οι κάρτες γραφικών προσπαθούν να «κρύψουν» το latency δημιουργώντας και εκτελώντας χιλιάδες νήματα ταυτόχρονα. Έτσι, όταν κάποια νήματα αναμένουν δεδομένα από την μνήμη, κάποια αλλά είναι έτοιμα προς εκτέλεση και εκτελούνται κ.ο.κ. Συνεπώς, το συντριπτικό τμήμα του chip αφιερώνεται σε αριθμητικές μονάδες εκτέλεσης πράξεων. Επιπλέον, υπάρχουν μικρές μνήμες ανά ομάδες πυρήνων, τις οποίες συνήθως διαχειρίζεται ο προγραμματιστής ενώ το τμήμα αποκωδικοποίησης και δρομολόγησης των εντολών είναι εξαιρετικά απλό.

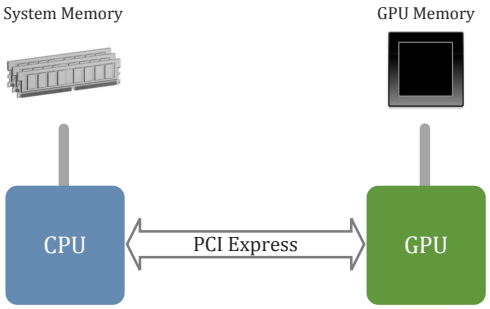
Ο αριθμός των νημάτων που μπορούν να διαχειριστούν ταυτόχρονα είναι τόσο μεγάλος ώστε κατορθώνουν να επιτύχουν το στόχο τους. Η ταυτόχρονη όμως πρόσβαση στη μνήμη από τόσα νήματα απαιτεί μεγάλο εύρος ζώνης (bandwidth), σημαντικά μεγαλύτερο σε σχέση με τους επεξεργαστές. Γι' αυτό το λόγο, ενσωματώνουν στο chip πολλαπλούς ελεγκτές μνήμης (memory controllers) ώστε να προσφέρουν συνολικό εύρος ζώνης προς την μνήμη (aggregate memory bandwitdh) που κυμαίνεται σε εκατοντάδες GB/s, όπως φαίνεται και στο *Σχήμα 1.13*.



Σχ. 1.13: Διάγραμμα σύγκρισης μέγιστου εύρους ζώνης προς τη μνήμη [13, 19, 12]

Η σύνδεση των καρτών γραφικών στο σύστημα γίνεται μέσω του διαύλου PCI Express, όπως φαίνεται στο *Σχήμα 1.14*. Αυτό συνεπάγεται ότι τα δεδομένα του προβλήματος που καλείται να λύσει η κάρτα γραφικών πρέπει να αντιγραφούν από την κύρια μνήμη του συστήματος στη μνήμη της κάρτας γραφικών και όταν ολοκληρωθεί η εκτέλεση να επιστρέψουν τα αποτελέσματα στην κύρια μνήμη με την αντίθετη διαδικασία. Το γεγονός ότι η μνήμη των καρτών γραφικών είναι σε πολλές περιπτώσεις μικρότερη των απαιτήσεων αναγκάζει τους προγραμματιστές να αντιγράφουν συχνά δεδομένα από και προς την μνήμη της κάρτας γραφικών κατά τη διάρκεια της εκτέλεσης. Αυτό δημιουργεί σημαντικά προβλήματα bottleneck διότι το εύρος ζώνης του

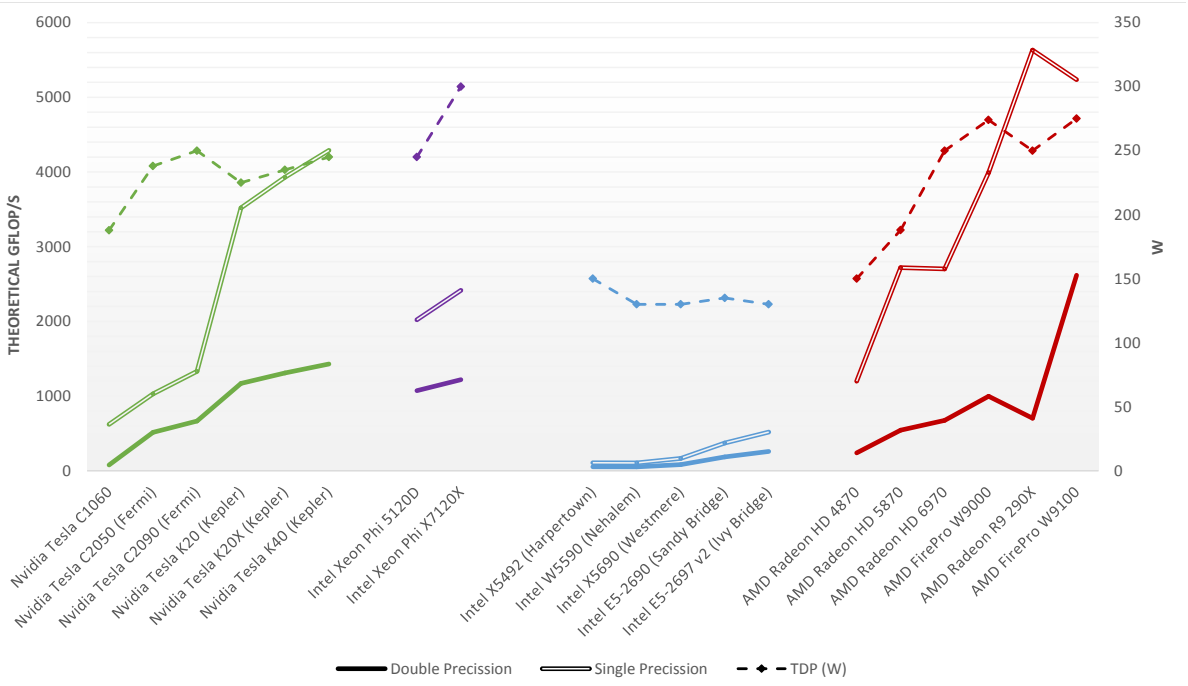
διαύλου PCI Express είναι περιορισμένο. Η σύνδεση μέσω 16 lanes (x16) παρέχει έως 8 GB/s ανά κατεύθυνση στην έκδοση 2.0 του PCI Express και έως 16 GB/s ανά κατεύθυνση στην έκδοση 3.0 [20].



Σχ. 1.14: Σύνδεση CPU και GPU μέσω του διαύλου PCI Express

Επιδόσεις

Στο *Σχήμα 1.15* φαίνεται ένα συγκεντρωτικό διάγραμμα που συγκρίνονται οι μέγιστες θεωρητικές επιδόσεις επεξεργαστών της Intel με κάρτες γραφικών των εταιριών Nvidia και AMD καθώς και με τη «manycore» εκδοχή της Intel (Intel Xeon Phi) που θα αναφερθεί πιο αναλυτικά στην επόμενη ενότητα. Επιπλέον, στη δευτερεύουσα στήλη του διαγράμματος φαίνονται οι ανώτερες προδιαγραφές κατανάλωσης (TDP) σε Watt.



Σχ. 1.15: Διάγραμμα σύγκρισης μέγιστων θεωρητικών επιδόσεων για πράξεις κινητής υποδιαστολής απλής και διπλής ακρίβειας και κατανάλωσης ενέργειας [13, 19, 12]

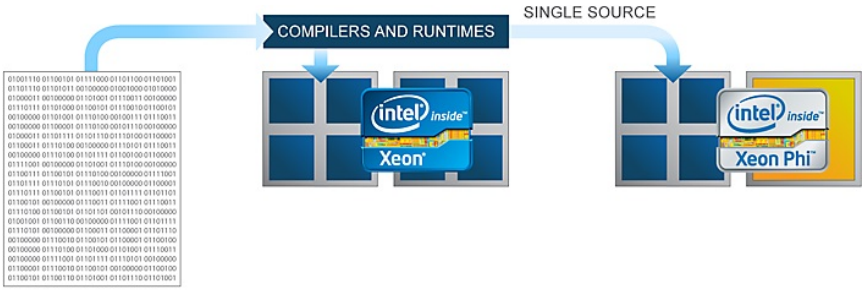
Παρατηρείται αμέσως η τεράστια διαφορά στις θεωρητικές επιδόσεις ανάμεσα σε επεξεργαστές και κάρτες γραφικών, άμεση συνέπεια των πολλών αριθμητικών μονάδων (ALUs) των καρτών γραφικών καθώς και τα πλαίσια κατανάλωσης ενέργειας μέσα στα οποία πραγματοποιούνται οι μέγιστες επιδόσεις. Φυσικά, η επίτευξή τους στην πράξη δεν είναι εύκολη και επιτυγχάνεται με κόπο και για συγκεκριμένου τύπου εφαρμογές. Επιπλέον, η δημιουργία αποδοτικών προγραμμάτων για κάρτες γραφικών δεν είναι κάτι απλό αφού απαιτεί εις βάθος εξοικείωση και γνώση τόσο του hardware όσο και του μοντέλου προγραμματισμού, το οποίο είναι σημαντικά διαφορετικό (έως δύστροπο για αρκετούς) σε σχέση με τους επεξεργαστές.

Παρόλα αυτά, έχουν γίνει τεράστια βήματα προόδου με στόχο τη διευκόλυνση των προγραμματιστών και τον εμπλουτισμό των δυνατοτήτων των καρτών γραφικών. Συνεπώς, σε πολλές περιπτώσεις το τελικό αποτέλεσμα αντισταθμίζει τις όποιες δυσκολίες και προβλήματα προκύπτουν κατά την ανάπτυξη των εφαρμογών.

1.3.2. Intel MIC

Η εταιρία Intel εισήλθε στο κομμάτι της αγοράς των manycore αρχιτεκτονικών το Νοέμβριο του 2012 όταν παρουσίασε την πρώτη της εμπορική προσπάθεια που ακούει στο όνομα Intel Xeon Phi, βασιζόμενο στην αρχιτεκτονική Intel MIC (Intel Many Integrated Core). Κατά την προώθηση του η εταιρία το αποκαλεί co-processor, δηλαδή συνεπεξεργαστή. Είχαν προηγηθεί αρκετές πειραματικές προσπάθειες, χωρίς όμως να παρουσιαστεί κάποιο εμπορικό προϊόν.

Πρόκειται για ένα καινοτόμο προϊόν το οποίο αποτελείται από 61 απλούς επεξεργαστικούς πυρήνες Pentium, τροποποιημένους κατάλληλα με πλάτους 512-bit SIMD units για την εκτέλεση αριθμητικών πράξεων και υποστήριξη 4-way SMT. Συνδέονται μέσω ενός αμφίδρομου διαδρόμου τύπου δακτυλιδιού (bi-directional ring bus). Η πιο γρήγορη έκδοση προσφέρει μέγιστες ονομαστικές επιδόσεις που αγγίζουν τα 1.2 TFLOP/s για πράξεις κινητής υποδιαστολής διπλής ακρίβειας και 2.4 TFLOP/s για απλής ακρίβειας.



Σχ. 1.16: Κοινός πηγαίος κώδικας μετάξυ CPU και Phi [21]

Το γεγονός ότι ενσωματώνει πυρήνες Pentium προσφέρει ένα σημαντικό πλεονέκτημα σε σχέση με τις κάρτες γραφικών, ότι εκτελεί x86 κώδικα. Συνεπώς, οι ήδη υλοποιημένες εφαρμογές που εκτελούνται στους επεξεργαστές μπορούν να εκτελεστούν με ελάχιστες αλλαγές στο Phi, χωρίς να χρειάζεται να ξαναυλοποιηθούν από την αρχή (όπως γίνεται με τις κάρτες γραφικών). Επιπλέον, σύμφωνα με την εταιρία οι βελτιστοποιήσεις που απαιτούνται για την αποδοτική εκτέλεση των προγραμμάτων στο Phi δεν είναι τόσο πολύπλοκες όσο στις GPUs.

Αυτό αποτελεί ένα από τα κυριότερα επιχειρήματα κατά την προώθηση του προϊόντος. Ένα κομμάτι της διπλωματικής εργασίας θα προσπαθήσει να διερευνήσει κατά πόσον αυτό επιτυγχάνεται στην πράξη.

1.4. Σκοπός και δομή της διπλωματικής εργασίας

Στόχος της παρούσας διπλωματικής εργασίας αποτελεί η σύγκριση και η αξιολόγηση των διαφορετικών αρχιτεκτονικών παράλληλου προγραμματισμού που έχουν προαναφερθεί με κριτήρια εκτός από τις επιδόσεις, την ευκολία προγραμματισμού, την κατανάλωση ενέργειας, το κόστος του εξοπλισμού κ.ά.

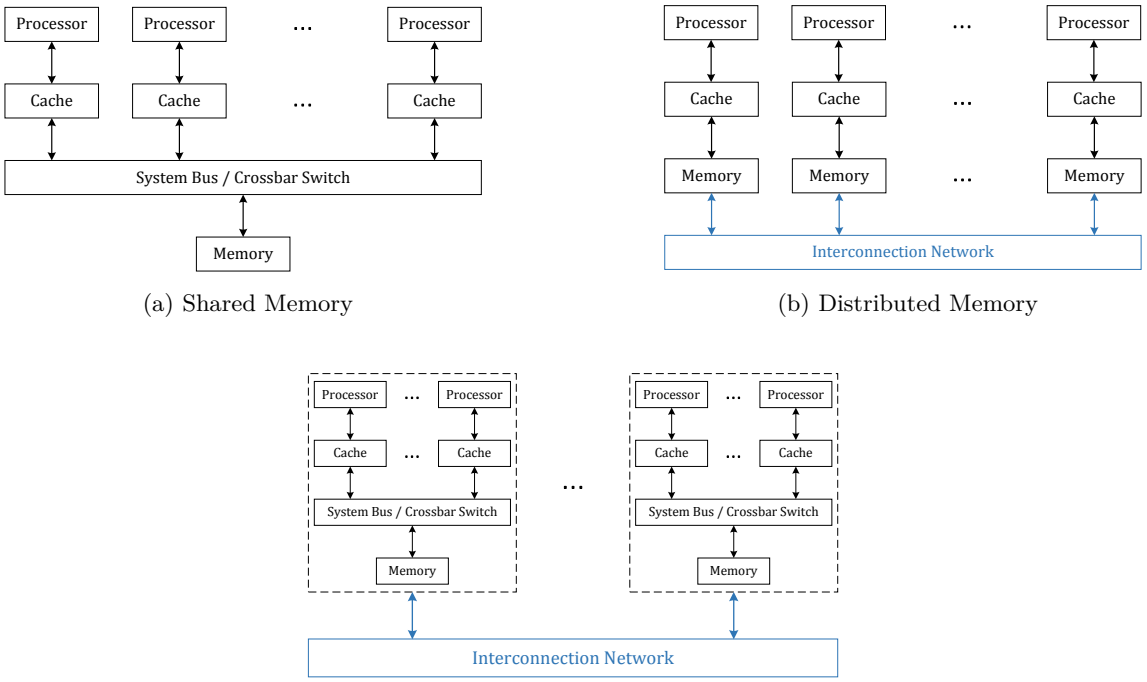
Στο Κεφάλαιο 2, αναλύονται οι αρχιτεκτονικές και τα προγραμματιστικά μοντέλα που θα χρησιμοποιηθούν στη διπλωματική εργασία. Στο Κεφάλαιο 3, παρουσιάζονται οι δύο αλγόριθμοί οι οποίοι θα υλοποιηθούν στις διάφορες αρχιτεκτονικές. Οι υλοποιήσεις και τα αποτελέσματα παρουσιάζονται στα Κεφάλαια 4 και 5 για κάθε αλγόριθμο αντίστοιχα. Τέλος, στο Κεφάλαιο 6, γίνεται μία μικρή σύνοψη και συζητούνται τα συμπεράσματα που προκύπτουν.

Κεφάλαιο 2

Παράλληλος Προγραμματισμός

Σε αυτό το κεφάλαιο θα παρατεθούν οι διαφορετικές αρχιτεκτονικές παράλληλου προγραμματισμού οι οποίες χρησιμοποιούνται και αξιολογούνται στην παρούσα διπλωματική εργασία. Θα αναλυθούν τα προγραμματιστικά μοντέλα στα οποία βασίζονται καθώς και η λογική με βάση την οποία έχουν σχεδιαστεί.

2.1. Προγραμματισμός σε πολλούς επεξεργαστές



(a) Shared Memory

(b) Distributed Memory

(c) Hybrid

Σχ. 2.1: Κατηγορίες παράλληλων αρχιτεκτονικών πολλαπλών επεξεργαστών

Όπως αναφέρθηκε στην *Υποενότητα 1.1.4*, η πλειοψηφία των σύγχρονων συστημάτων παράλληλων υπολογιστών ανήκουν στην κατηγορία αρχιτεκτονικών MIMD. Όπως φαίνεται στο *Σχήμα 2.1*, τα συστήματα αυτά χωρίζονται σε τρεις κατηγορίες, με κριτήριο τον τρόπο οργάνωσης της μνήμης σε σχέση με τους επεξεργαστές:

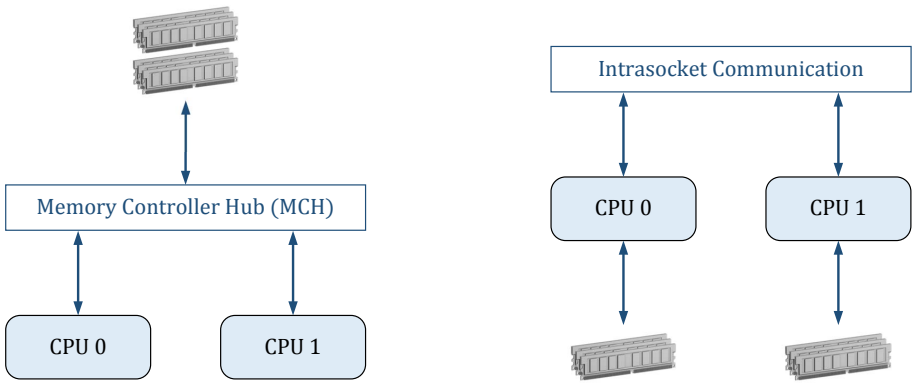
Αρχιτεκτονικές Μοιραζόμενης Μνήμης (Shared Memory): Όλοι οι επεξεργαστές έχουν πρόσβαση σε ολόκληρη τη μνήμη του συστήματος. Μπορούν να διατηρούν τοπική ιεραρχία επιπέδων κρυφής μνήμης ή να μοιράζονται κάποιο επίπεδο, συνήθως το τελευταίο (last level cache). Η επικοινωνία με την κύρια μνήμη γίνεται μέσω κάποιου διαύλου (bus) ή αποδοτικότερα μέσω crossbar switch. Το προγραμματιστικό μοντέλο προγραμματισμού τέτοιων συστημάτων ονομάζεται μοντέλο μοιραζόμενης μνήμης και αποτελεί το πρώτο μοντέλο με το οποίο ασχολείται η παρούσα διπλωματική εργασία.

Αρχιτεκτονικές Κατανεμημένης Μνήμης (Distributed Memory): Κάθε επεξεργαστής, εκτός από τοπική ιεραρχία κρυφής μνήμης, διαθέτει τοπική κύρια μνήμη και έχει πρόσβαση μόνο σε αυτή. Η πρόσβαση σε δεδομένα που βρίσκονται στις τοπικές μνήμες των υπόλοιπων επεξεργαστών γίνεται με χρήση κάποιου προτύπου μεταφοράς δεδομένων το οποίο υλοποιείται μέσω ειδικού δικτύου διασύνδεσης (λ.χ. Ethernet, Infiniband, Myrinet). Τέτοια συστήματα προγραμματίζονται με χρήση του προγραμματιστικού μοντέλου που ονομάζεται μοντέλο ανταλλαγής μηνυμάτων και δημοφιλέστερο πρότυπο αποτελεί το MPI (Message Passing Interface).

Υβριδικές Αρχιτεκτονικές (Hybrid architectures): Στην κατηγορία αυτή ανήκουν οι αρχιτεκτονικές που συνδυάζουν τις δύο προαναφερθείσες. Δηλαδή, κόμβοι που υλοποιούν αρχιτεκτονική μοιραζόμενης μνήμης συνδέονται μέσω ειδικού δικτύου διασύνδεσης, υλοποιώντας αρχιτεκτονική κατανεμημένης μνήμης. Στην κατηγορία αυτή ανήκουν όλοι οι σύγχρονοι υπερυπολογιστές.

2.1.1. Αρχιτεκτονικές Μοιραζόμενης Μνήμης

Όπως ήδη έχει αναφερθεί, στις αρχιτεκτονικές μοιραζόμενης μνήμης όλοι οι επεξεργαστές μοιράζονται έναν κοινό φυσικό χώρο διευθύνσεων μνήμης (physical memory address space). Ανάλογα με τον τρόπο που οργανώνεται η μνήμη, οι αρχιτεκτονικές μοιραζόμενης μνήμης χωρίζονται σε: *Uniform Memory Access (UMA)* και *Non-Uniform Memory Access (NUMA)*.



(a) UMA (b) NUMA

Σχ. 2.2: Κατηγορίες αρχιτεκτονικών μοιραζόμενης μνήμης

Στα συστήματα UMA, η πρόσβαση στην κύρια μνήμη απαιτεί περίπου τον ίδιο χρόνο (latency) ανεξάρτητα από το ποιος επεξεργαστής αιτείται την πρόσβαση και ανεξάρτητα από το ποιο memory module περιέχει τα αιτούμενα δεδομένα. Το μειονέκτημα είναι ότι το διαθέσιμο εύρος ζώνης (bandwidth) προς τη μνήμη μοιράζεται σε όλους του επεξεργαστές. Αυτό συνεπάγεται ότι δεν επιτυγχάνεται καλή κλιμάκωση για μεγάλο αριθμό επεξεργαστών.

Αντίθετα, στα συστήματα NUMA, κάθε επεξεργαστής διαθέτει τοπικά ένα τμήμα της κύριας μνήμης και συνδέεται με αυτή. Οι επεξεργαστές συνδέονται μεταξύ τους με ένα ειδικό σύστημα διασύνδεσης ώστε να παραμένει ο χώρος διευθύνσεων κοινός για όλους. Όμως, ο χρόνος κάθε πρόσβασης στη μνήμη δεν είναι ίδιος. Είναι γρήγορος για δεδομένα που είναι αποθηκευμένα στην τοπική μνήμη κάθε επεξεργαστή ενώ πιο αργός για δεδομένα που βρίσκονται στην τοπική μνήμη άλλου επεξεργαστή. Παρά το γεγονός ότι αυτό προκαλεί επιπλέον δυσκολία στον προγραμματιστή, τα συστήματα NUMA κλιμακώνουν καλύτερα για μεγαλύτερο αριθμό επεξεργαστών ενώ προσφέρουν σημαντικά αυξημένο συνολικό εύρος ζώνης προς τη μνήμη (aggregate memory bandwidth).

Συνέπεια κρυφής μνήμης (Cache coherency)

Κάθε επεξεργαστής μπορεί να τροποποιεί δεδομένα στην cache του χωρίς να ενημερώσει την κύρια μνήμη. Το γεγονός ότι κάθε επεξεργαστής διατηρεί τοπική ιεραρχία κρυφής μνήμης συνεπάγεται ότι μπορεί να υπάρχουν πολλά αντίγραφα της ίδιας θέσης μνήμης στις κρυφές μνήμες των επεξεργαστών. Προκειμένου να μη δημιουργηθεί ασυνέπεια στα μοιραζόμενα δεδομένα, γίνεται χρήση ενός πρωτοκόλλου συνέπειας κρυφής μνήμης (cache coherency protocol). Συστήματα που υλοποιούν τέτοια πρωτόκολλα ονομάζονται ccUMA (cache coherent UMA) και ccNUMA ανάλογα με τον διαχωρισμό που προαναφέρθηκε.

Υπάρχουν δύο κατηγορίες τέτοιων πρωτοκόλλων. Στην πρώτη κατηγορία όλοι οι ελεγκτές κρυφής μνήμης (cache controller) επικοινωνούν μέσω ενός δικτύου διασύνδεσης και για κάθε εγγραφή κάποιου επεξεργαστή ενημερώνονται οι υπόλοιποι ώστε αν έχουν αντίγραφο της συγκεκριμένης θέσης μνήμης να το «ακυρώσουν». Επιπλέον, «κρυφακούνε» (snooping) στο δίκτυο για κάθε αίτηση ανάγνωσης κάποιας θέσης μνήμης ώστε να την παράσχουν εκείνοι σε περίπτωση που υπάρχει έγκυρο αντίγραφο στην cache τους. Η δεύτερη κατηγορία αφορά πρωτόκολλα συνέπειας τύπου καταλόγου (directory). Κρατούνται στοιχεία για τις θέσεις μνήμες που βρίσκονται στις caches των επεξεργαστών είτε σε ένα κεντρικό σημείο είτε κατανεμημένα σε κάθε επεξεργαστή (distributed directories). Τα πρωτόκολλα τύπου καταλόγου προσφέρουν καλύτερη κλιμάκωση και χρησιμοποιούνται κυρίως σε ccNUMA συστήματα.

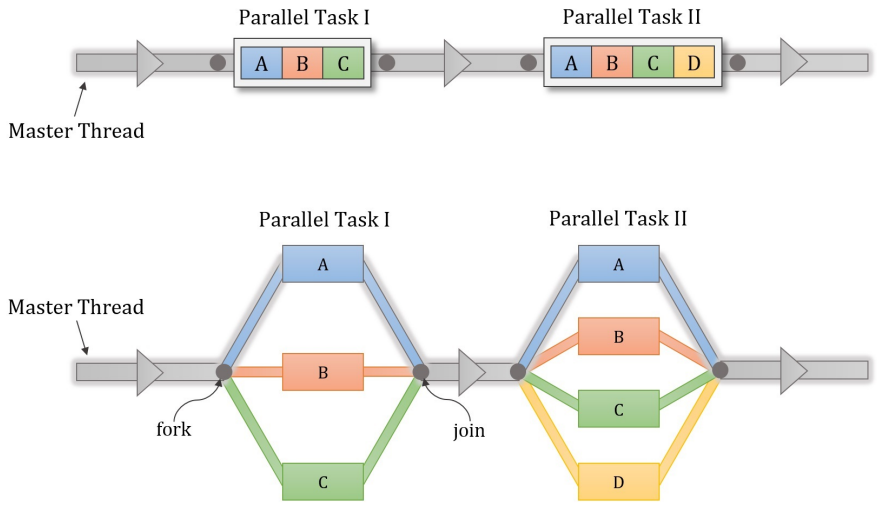
2.1.2. OpenMP

Το OpenMP (Open Specifications for Multi Processing) [17] αποτελεί ένα πολύ δημοφιλές πρότυπο παράλληλου προγραμματισμού (Application Programming Interface, API), το οποίο δίνει τη δυνατότητα δημιουργίας παράλληλων προγραμμάτων για συστήματα μοιραζόμενης μνήμης. Ακολουθεί το προγραμματιστικό μοντέλο με το οποίο προγραμματίζονται τέτοια συστήματα. Τα προγράμματα που αναπτύσσονται με χρήση του OpenMP μπορούν να εκτελεστούν σε διαφορετικές αρχιτεκτονικές και έχουν μεγάλη ικανότητα κλιμάκωσης. Είναι αρκετά απλό και εύκολο για τον προγραμματιστή και υποστηρίζει τις γλώσσες προγραμματισμού C/C++ και Fortran. Με αφετηρία τη σειριακή υλοποίηση του προγράμματος, ο προγραμματιστής ορίζει τα τμήματα του προγράμματος που επιθυμεί να εκτελεστούν παράλληλα υποδεικνύοντάς τα στον κώδικα με χρήση ειδικών εντολών οδηγιών (directives). [18]

Fork - Join

Η βασική μονάδα εκτέλεσης είναι το νήμα (thread) και κάθε διεργασία αποτελείται από ένα ή παραπάνω νήματα. Το OpenMP ακολουθεί το μοντέλο Fork - Join. Αρχικά, μία εφαρμογή OpenMP ξεκινά με ένα μόνο νήμα, το οποίο ονομάζεται master thread. Όταν το πρόγραμμα εισέρχεται σε μία περιοχή την οποία έχει ορίσει ο προγραμματιστής να εκτελεστεί παράλληλα (παράλληλη περιοχή, parallel region), τότε δημιουργούνται (fork) αρκετά νήματα τα οποία εκτελούνται παράλληλα μεταξύ τους. Όταν ολοκληρωθεί η εκτέλεση της παράλληλης περιοχής τότε όλα τα νήματα τερματίζουν και συγχρονίζονται (join) και συνεχίζει μόνο το master thread. Στη συνέχεια, επαναλαμβάνεται η ίδια διαδικασία για κάθε παράλληλη περιοχή που συναντάται στον κώδικα.

Βάσει των υποδείξεων του προγραμματιστή (directives), ο μεταγλωττιστής μετατρέπει κατάλληλα τον σειριακό κώδικα (single-threaded) σε πολυνηματικό (multi-threaded) για τις παράλληλες περιοχές. Στο *Σχήμα 2.3* φαίνεται μια σχηματική περιγραφή του μοντέλου Fork - Join.

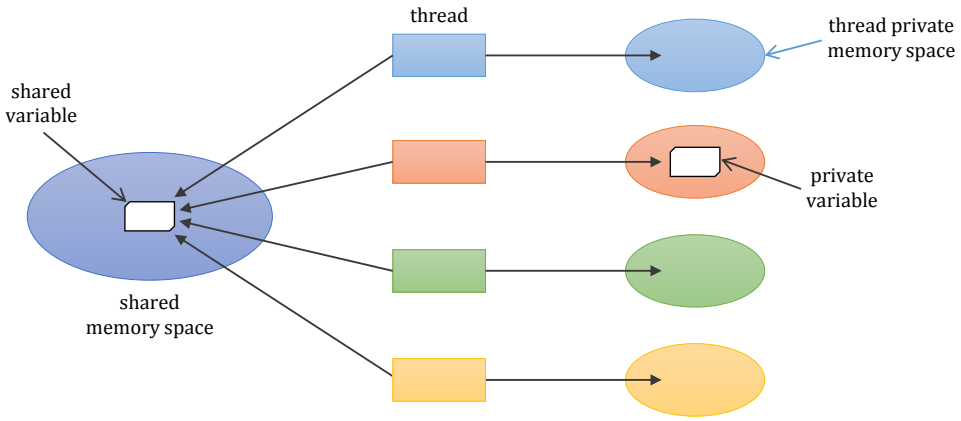


Σχ. 2.3: Μοντέλο Fork - Join

Το OpenMP δεν μπορεί να εγγυηθεί ότι μία παράλληλη εκτέλεση ενός κώδικα είναι συνεπής με την αντίστοιχη σειριακή εκτέλεση αφού δεν είναι εγγυημένο ότι η είσοδος και η έξοδος είναι συγχρονισμένες. Ο συγχρονισμός των νημάτων γίνονται με ευθύνη του προγραμματιστή. Το OpenMP παρέχει αρκετές οδηγίες και συναρτήσεις συγχρονισμού οι οποίες πρέπει να χρησιμοποιηθούν σωστά για να είναι εξασφαλισμένη η ορθότητα του προγράμματος.

Μοντέλο Μνήμης

Το μοντέλο μνήμης του OpenMP είναι ένα χαλαρό μοντέλο μοιραζόμενης μνήμης και παρέχει δύο είδη μνήμης στα νήματα: μοιραζόμενη και ιδιωτική. Ένα τμήμα της κύριας μνήμης είναι μοιραζόμενο (shared memory space) και σε αυτό έχουν πρόσβαση όλα τα νήματα. Μπορεί να χρησιμοποιηθεί για ανταλλαγή δεδομένων μεταξύ των νημάτων καθώς και για το συγχρονισμό τους. Επιπλέον, κάθε νήμα έχει τον προσωπικό/ιδιωτικό του χώρο στη μνήμη (thread private memory space) και μόνο εκείνο έχει πρόσβαση σε αυτόν.



Σχ. 2.4: Μοντέλο μνήμης OpenMP

Όταν ο προγραμματιστής ορίζει μία παράλληλη περιοχή, μπορεί να ορίσει αν οι μεταβλητές που εμπεριέχονται σε αυτή θα είναι μοιραζόμενες ή ιδιωτικές. Κάθε αναφορά σε μοιραζόμενη μεταβλητή είναι μία αναφορά στην ίδια την μεταβλητή, ενώ μια αναφορά σε ιδιωτική μεταβλητή είναι μια αναφορά σε ένα τοπικό αντίγραφο της μεταβλητής που βρίσκεται αποθηκευμένο στην ιδιωτική μνήμη του νήματος. Για την προσπέλαση των μοιραζόμενων μεταβλητών από διαφορετικά νήματα απαιτείται συγχρονισμός και αυτό είναι ευθύνη του προγραμματιστή. Το OpenMP παρέχει τη δυνατότητα η πρόσβαση σε μοιραζόμενες μεταβλητές να είναι ατομική ενέργεια (atomic operation) για κάθε νήμα και να μην χρειάζεται «χειροκίνητη» παρέμβαση από τον προγραμματιστή.

OpenMP directives

Όπως αναφέρθηκε προηγουμένως, ο προγραμματιστής χρησιμοποιεί κατάλληλες οδηγίες (directives) που του παρέχει το OpenMP για να δημιουργήσει παράλληλα προγράμματα.

Όλα τα directives στη γλώσσα προγραμματισμού C έχουν την εξής μορφή:

```
#pragma omp construct [clause [clause]...]
```

όπου construct είναι το όνομα της οδηγίας και clauses είναι προαιρετικές φράσεις.

#pragma omp parallel

Το parallel directive αποτελεί τη βασικότερη οδηγία από όλες καθώς εκείνη ορίζει μια παράλληλη περιοχή στον κώδικα, δηλαδή το τμήμα που θα εκτελεστεί παράλληλα.

Κώδικας 2.1 Παράδειγμα parallel directive

```
#include <omp.h>

main()
{
    ...
    #pragma omp parallel num_threads(4)
    {

        ...

    }
    ...
    omp_set_num_threads(4);

    #pragma omp parallel shared(...) private(...)
    {

        ...

    }
    ...
}
```

Ο αριθμός των νημάτων που δημιουργούνται κατά την είσοδο σε μια παράλληλη περιοχή μπορεί να καθοριστεί από τέσσερις παράγοντες, οι οποίοι κατά σειρά προτεραιότητας είναι:

- Η φράση (clause) `num_threads()`
- Η χρήση της συνάρτησης βιβλιοθήκης `omp_set_num_threads()`
- Η τιμή της μεταβλητής περιβάλλοντος (enviromental variable) OMP_NUM_THREADS
- Προκαθορισμένη τιμή με χρήση της φράσης default (συνήθως ο αριθμός των επεξεργαστικών πυρήνων του συστήματος)

Συγχρονισμός

- `#pragma omp barrier`: Η οδηγία barrier συγχρονίζει όλα τα νήματα μεταξύ τους. Απαιτεί από κάθε νήμα να σταματήσει προσωρινά την εκτέλεσή του στο σημείο όπου υπάρχει η οδηγία barrier, μέχρις ότου όλα τα νήματα φτάσουν σε αυτό το σημείο. Στη συνέχεια, όλα τα νήματα ξεκινούν παράλληλα, από εκείνο το σημείο, την εκτέλεσή του κώδικα που ακολουθεί.
- `#pragma omp master`: Η οδηγία αυτή ορίζει ένα τμήμα κώδικα το οποίο θα εκτελεστεί από το mater thread μόνο. Δεν υπάρχει κάποιο φράγμα στο τέλος του για τα υπόλοιπα νήματα, τα οποία απλά προσπερνούν την οδηγία αυτή.
- `#pragma omp critical`: Η οδηγία critical καθορίζει μια περιοχή η οποία πρέπει να εκτελεστεί μόνο από ένα νήμα κάθε φορά. Κυρίως χρησιμοποιείται για να οριστεί μια κρίσιμη περιοχή (critical region). Σε μια κρίσιμη περιοχή, μόνο μια διεργασία μπορεί να γράψει ή να διαβάσει μια μοιραζόμενη μεταβλητή, διασφαλίζοντας έτσι την ακεραιότητα αυτής της μεταβλητής.
- `#pragma omp atomic`: Η οδηγία atomic καθορίζει ότι μια συγκεκριμένη θέση μνήμης πρέπει να ενημερώνεται ατομικά (atomic operation) από κάθε νήμα, μη επιτρέποντας σε πολλά νήματα να ενημερώσουν ταυτόχρονα τη συγκεκριμένη θέση μνήμης. Έτσι, απαγορεύει σε οποιοδήποτε νήμα να διακόψει κάποιο άλλο νήμα που βρίσκεται στη διαδικασία προσπέλασης ή αλλαγής της τιμής μιας μεταβλητής μοιραζόμενης μνήμης.

Είδη μεταβλητών

Με χρήση κατάλληλων φράσεων ο προγραμματιστής ορίζει το είδος (μοιραζόμενες ή ιδιωτικές) των μεταβλητών που εμπεριέχονται στην παράλληλη περιοχή. Οι συχνότερα χρησιμοποιούμενες είναι:

- `private`: Με την φράση αυτή ορίζονται ιδιωτικές μεταβλητές για κάθε νήμα. Αυτό σημαίνει ότι κάθε νήμα έχει δικό του αντίγραφο της μεταβλητής στην ιδιωτική μνήμη του. Όλες οι αναφορές στις private μεταβλητές μετατρέπονται σε αναφορές στα αντίγραφα των μεταβλητών. Τα αντίγραφα των μεταβλητών δεν αρχικοποιούνται κατά τη δημιουργία των νημάτων.
- `firstprivate`: Όμοια με προηγουμένως, με τη διαφορά ότι τα αντίγραφα αρχικοποιούνται στην τιμή που είχε η μεταβλητή πριν την είσοδο στη παράλληλη περιοχή.
- `lastprivate`: Πρόκειται για το αντίθετο της firstprivate, δηλαδή η τιμή της μεταβλητής διατηρείται κατά την έξοδο από τη παράλληλη περιοχή και είναι ίση με την τιμή που πήρε η μεταβλητή στην εκτέλεση της τελευταίας (με λεξικογραφική έννοια) παράλληλης εκτέλεσης.
- `shared`: Με τη φράση αυτή ορίζονται μεταβλητές ως μοιραζόμενες σε όλα τα νήματα. Αυτό σημαίνει ότι αν ένα νήμα αλλάξει την τιμή μιας τέτοιας μεταβλητής, η αλλαγή θα είναι ορατή σε όλα τα νήματα. Τα νήματα δεν έχουν τοπικό αντίγραφο της μεταβλητής αυτής, χρησιμοποιούν τη θέση της στη μοιραζόμενη μνήμη. Ο προγραμματιστής είναι υπεύθυνος να φροντίσει να μην δημιουργήσουν προβλήματα ταυτόχρονων αναγνώσεων και εγγραφών της ίδιας μεταβλητής από πολλά νήματα. Οι shared μεταβλητές διατηρούν την τιμή τους κατά την έξοδο από την παράλληλη περιοχή.

#pragma omp for

Αποτελεί directive διαμοιρασμού εργασίας και διαμοιράζει τις επαναλήψεις ενός βρόχου for στα νήματα της τρέχουσας παράλληλης περιοχής. Δηλαδή, δε δημιουργεί νέα νήματα και πρέπει να εμπεριέχεται μέσα σε μια παράλληλη περιοχή.

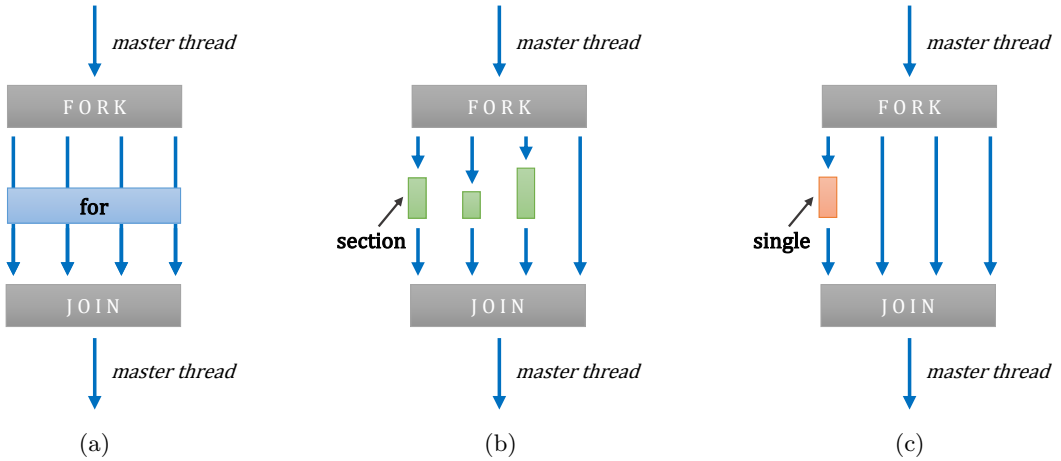
Κώδικας 2.2 Παράδειγμα for directive

```
#include <omp.h>

main()
{
    ...
    #pragma omp parallel
        #pragma omp for schedule(static)
        for(i=0; i<N; i++)
        {
            a[i] = b[i] + c[i];
        }
    ...
}
```

Ο τρόπος με τον οποίο γίνεται ο διαμοιρασμός των επαναλήψεων ανάμεσα στα νήματα εξαρτάται από τη φράση schedule:

- `static`: Οι επαναλήψεις του βρόχου χωρίζονται σε ίσα τμήματα μεγέθους chunk και μοιράζονται στατικά στα νήματα (πριν αρχίσει η εκτέλεση, κάθε νήμα γνωρίζει πόσες και ποιες επαναλήψεις θα εκτελέσει). Αν δεν οριστεί το chunk, τότε γίνεται ισομεγέθης διαμοιρασμός, αν αυτό βέβαια είναι εφικτό.
- `dynamic`: Με αυτήν την παράμετρο οι επαναλήψεις χωρίζονται σε ίσα τμήματα μεγέθους chunk. Αν δεν οριστεί το μέγεθος, κάθε τμήμα έχει μία επανάληψη του κώδικα. Κάθε φορά που ένα νήμα ολοκληρώνει την εκτέλεση του τμήματος που του έχει ανατεθεί, του ανατίθεται δυναμικά ένα άλλο τμήμα.
- `guided`: Το μέγεθος κάθε τμήματος μειώνεται εκθετικά, καθώς γίνονται οι αναθέσεις των κομματιών του χώρου επανάληψης. Η τιμή chunk ορίζει το μικρότερο κομμάτι στο οποίο μπορεί να σπάσει. Η προκαθορισμένη τιμή είναι 1.

Τέλος, η φράση `nowait`, αν υπάρχει, αναιρεί το φράγμα που υπάρχει στο τέλος της οδηγίας for.



Σχ. 2.5: Directives διαμοιρασμού εργασίας

#pragma omp sections

Η οδηγία sections είναι μια μη επαναληπτική περιοχή διαμοιρασμού εργασίας. Καθορίζει ότι τα εσωκλειόμενα τμήματα κώδικα θα διαμοιραστούν μεταξύ των νημάτων της ομάδας. Μια οδηγία sections μπορεί να περιέχει περισσότερες από μία, ανεξάρτητες, οδηγίες section. Κάθε section εκτελείται μια φορά από ένα νήμα της ομάδας, ενώ διαφορετικά sections εκτελούνται από διαφορετικά νήματα. Στο τέλος κάθε οδηγίας section υπονοείται κάποιο φράγμα που θα βοηθήσει στο συγχρονισμό των νημάτων, εκτός κι αν χρησιμοποιηθεί η φράση nowait οπότε τα νήματα δεν περιμένουν για συγχρονισμό μετά το πέρας της εργασίας τους.

Κώδικας 2.3 Παράδειγμα sections directive

```
#pragma omp parallel
{
    ...
    #pragma omp sections
    {
        #pragma omp section
            f1();
        #pragma omp section
            f2();
    }
    ...
}
```

#pragma omp single

Η οδηγία single καθορίζει ότι ο κώδικας που εσωκλείεται σε αυτή θα εκτελεστεί μόνο από ένα νήμα. Το νήμα που θα φτάσει πρώτο στη single οδηγία είναι εκείνο που θα εκτελέσει τον κώδικα. Τα νήματα που δεν εκτελούν την οδηγία, περιμένουν στο τέλος του εσωκλειόμενου κώδικα, εκτός αν χρησιμοποιηθεί η συνθήκη nowait οπότε τα νήματα δεν θα περιμένουν για να συγχρονιστούν.

OpenMP Tasks

Η έκδοση 3.0 του OpenMP που κυκλοφόρησε το 2008 εισήγαγε έναν νέο τρόπο για παραλληλοποίηση με χρήση tasks. Με τη χρήση των tasks μπορεί να οριστεί ένα κομμάτι κώδικα εντός της παράλληλης περιοχής το οποίο μπορεί να εκτελεστεί κάποια στιγμή και όχι κατ' ανάγκη από το νήμα που την όρισε. Παρέχουν την δυνατότητα ευκολότερης παραλληλοποίησης αναδρομικών συναρτήσεων και γενικά εφαρμογών που παράγουν δουλειά δυναμικά. Επίσης, με τη βοήθεια των tasks μπορεί σε αρκετές περιπτώσεις να αποφευχθεί η χρήση εμφωλευμένων παράλληλων περιοχών που μπορούν να δημιουργήσουν προβλήματα με ορισμένους compilers.

Η δημιουργία νέου task γίνεται με το εξής directive:

```
#pragma omp task [clause [clause]...]
```

Το νήμα που συναντά το παραπάνω directive δημιουργεί ένα νέο task με τον κώδικα που εμπεριέχεται στο directive και το τοποθετεί σε ένα pool στο οποίο βρίσκονται τα tasks προς εκτέλεση. Αυτά εκτελούνται από τα διαθέσιμα νήματα. Επίσης, ένα task μπορεί να δημιουργήσει νέα tasks.

Ο συγχρονισμός των tasks γίνεται με τη χρήση barrier που ορίζεται από το παρακάτω directive:

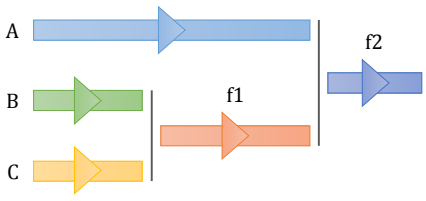
```
#pragma omp taskwait
```

Το task που συναντάει ένα taskwait barrier σταματάει την εκτέλεσή του μέχρι όλα τα tasks που έχει δημιουργήσει να ολοκληρώσουν την εκτέλεσή τους. Αυτό ισχύει μόνο για τα άμεσα παιδιά του task και όχι για τα παιδιά δεύτερης γενιάς.

Παρακάτω ακολουθεί παράδειγμα που φαίνεται αναλυτικά η δημιουργία και η δρομολόγηση των tasks.

Κώδικας 2.4 Παράδειγμα OpenMP tasks

```
int a, b, c, x, y;
#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp task shared(a)
        a = A();

        #pragma omp task shared(b, c, x)
        {
            #pragma omp task shared(b)
            b = B();
            #pragma omp task shared(c)
            c = C();
            #pragma omp taskwait

            #pragma omp task
            x = f1(b, c);
        }
        #pragma omp taskwait
        #pragma omp task
        y = f2(a, x);
    }
}
```



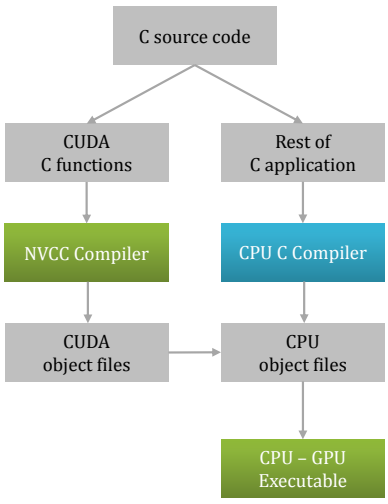
2.2. Προγραμματισμός σε Κάρτες Γραφικών

Η δεύτερη κατηγορία παράλληλων αρχιτεκτονικών με την οποία ασχολείται η παρούσα διπλωματική εργασία είναι εκείνη των καρτών γραφικών. Όπως έχει ήδη αναφερθεί, οι κάρτες γραφικών αντλούν τη δύναμή τους από τις πολλές υπολογιστικές μονάδες που διαθέτουν και από τη δυνατότητα ταυτόχρονης εκτέλεσης τεράστιου αριθμού νημάτων. Για το λόγο αυτό, οι εφαρμογές που μπορούν να εκμεταλλευτούν τις δυνατότητες των καρτών γραφικών είναι εκείνες οι οποίες εκτελούν πολλές αριθμητικές πράξεις σε μεγάλο όγκο δεδομένων και μπορούν να παραλληλοποιηθούν μαζικά σε πάρα πολλά νήματα.

Όλες οι υλοποιήσεις θα γίνουν χρησιμοποιώντας το CUDA SDK της εταιρίας Nvidia. Για την αξιολόγηση των αποτελεσμάτων θα χρησιμοποιηθούν κάρτες γραφικών της εταιρίας από την επιστημονική σειρά Tesla, οι οποίες απευθύνονται ειδικά στο HPC κομμάτι της αγοράς και είναι σχεδιασμένες προς αυτή την κατεύθυνση. Θα εξερευνηθούν οι δύο πιο σύγχρονες αρχιτεκτονικές της εταιρίας: *Fermi* (Compute Capability 2.0, 2.1) και *Kepler* (Compute Capability 3.0, 3.5).

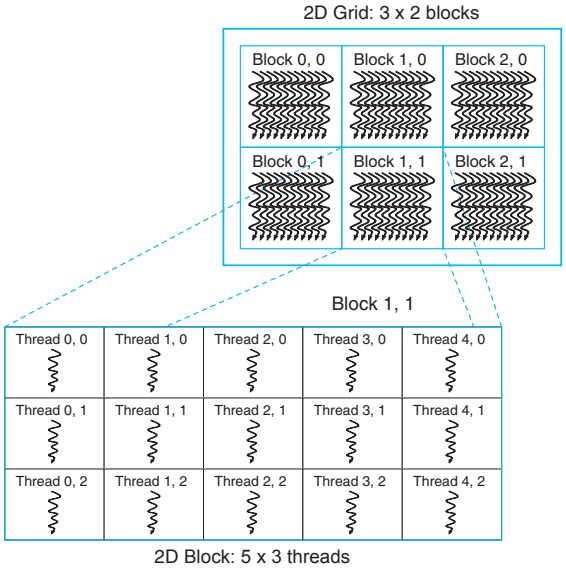
2.2.1. Προγραμματιστικό μοντέλο CUDA

Η πλατφόρμα CUDA της Nvidia προσφέρει μια επέκταση της γλώσσας προγραμματισμού C και παρέχει ειδικό compiler και κατάλληλες συναρτήσεις για την αλληλεπίδραση του προγραμματιστή με την κάρτα γραφικών (π.χ. αποστολή προγραμμάτων προς εκτέλεση, μεταφορά δεδομένων από και προς την κάρτα γραφικών, κλπ).



Σχ. 2.6: Μεταγλώττιση προγράμματος CUDA

Τα τμήματα του πηγαίου κώδικα που προορίζονται για εκτέλεση στην κάρτα γραφικών ονομάζονται *kernels* και μεταγλωττίζονται από τον nvcc compiler ενώ το υπόλοιπο πρόγραμμα που προορίζεται για τον επεξεργαστή μεταγλωττίζεται από τον gcc compiler. Στη συνέχεια, ενοποιούνται σε ένα κοινό εκτελέσιμο αρχείο. Αρχικά, ξεκινάει η εκτέλεση του προγράμματος στον επεξεργαστή και το περιβάλλον εκτέλεσης CUDA αναλαμβάνει να προωθήσει τα τμήματα του κώδικα που προορίζονται για την κάρτα γραφικών.



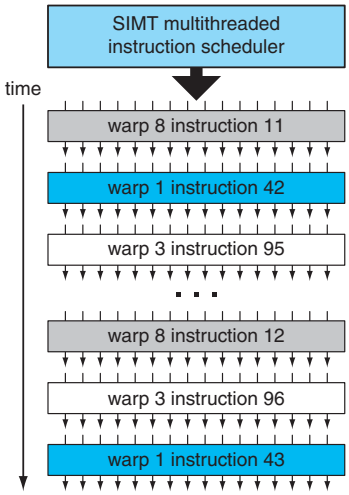
Σχ. 2.7: Οργάνωση των CUDA threads [4]

Κάθε kernel εκτελείται σε ολόκληρη την κάρτα γραφικών και αποτελείται από χιλιάδες νήματα τα οποία ονομάζονται CUDA threads. Τα threads ενός kernel χωρίζονται σε blocks και με τη σειρά τους τα blocks οργανώνονται σε ένα grid, όπως φαίνεται στο *Σχήμα 2.7*. Κάθε thread έχει ένα μοναδικό τοπικό index μέσα στο block και κάθε block έχει επίσης μοναδικό index μέσα στο grid. Τόσο τα block όσο και τα grid μπορούν να έχουν μία, δύο ή τρεις το πολύ διαστάσεις.

SIMT

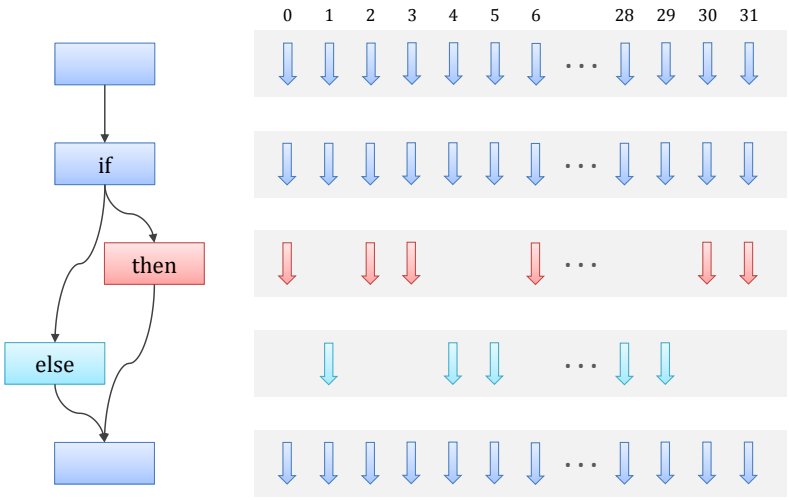
Οι κάρτες γραφικών της Nvidia ακολουθούν το προγραμματιστικό μοντέλο SIMT (Single Instruction, Multiple Threads). Όλα τα threads προς εκτέλεση οργανώνονται σε ομάδες που ονομάζονται warps και αποτελούνται από 32 threads. Η οργάνωση των threads στα warps γίνεται με τέτοιο τρόπο ώστε κάθε warp να περιλαμβάνει threads με συνεχόμενα thread ID.

Η δρομολόγηση γίνεται σε επίπεδο warps. Κάθε στιγμή ο δρομολογητής επιλέγει warps των οποίων η επόμενη εντολή είναι έτοιμη για να εκτελεστεί και τα δρομολογεί. Το state (registers, program counters, κλπ) καθενός warp διατηρείται on chip σε όλη τη φάση της εκτέλεσης εξαιτίας των χιλιάδων registers που διαθέτουν οι πολυεπεξεργαστές των καρτών γραφικών. Έτσι, σε αντίθεση με τους επεξεργαστές, το context switching γίνεται με μηδενικό κόστος (on the fly). Εξαιτίας αυτού και επειδή τα warps είναι ανεξάρτητα μεταξύ τους, ο δρομολογητής μπορεί να επιλέγει διαφορετικό warp σε κάθε επόμενο issue, όπως φαίνεται στο *Σχήμα 2.8*.



Σχ. 2.8: Δρομολόγηση και εκτέλεση warp [4]

Το warp εκτελεί μια εντολή σε κάθε κύκλο ρολογιού και είναι η ίδια για όλα τα threads που το αποτελούν. Παρόλα αυτά, κάθε thread διαθέτει δικό του program counter. Συνεπώς, είναι δυνατή η χρήση conditional branch και η διαφοροποίηση των execution path των threads. Όμως, η εκτέλεση τους είναι διαφορετική από τους επεξεργαστές. Όπως φαίνεται και στο *Σχήμα 2.9*, όταν συναντάται ένα conditional branch τότε τα execution path των threads αποκλίνουν (diverge) αλλά εκτελείται σειριακά κάθε τμήμα του branch. Έτσι, εκτελείται αρχικά το «then» τμήμα του branch και όποια threads δεν εμπίπτουν σε αυτό απλά απενεργοποιούνται προσωρινά. Στη συνέχεια, γίνεται το ίδιο και για το «else» τμήμα. Όταν ολοκληρωθεί η εκτέλεση του branch, όλα τα threads συγκλίνουν και συνεχίζουν την εκτέλεση του κοινού execution path. Τα warps που περιέχουν threads που αποκλίνουν ονομάζονται divergent warps. Όπως γίνεται εμφανές, η ύπαρξη διαφορετικών execution paths εντός των warps έχει σημαντική επίπτωση στις επιδόσεις λόγω της σειριακής εκτέλεσης των paths.



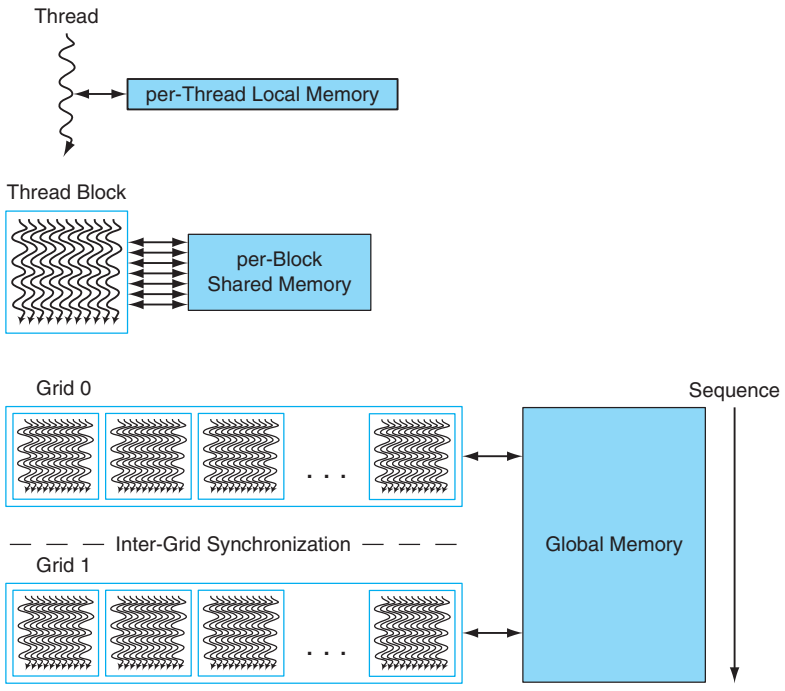
Σχ. 2.9: Εκτέλεση conditional branch από ένα warp

Το προγραμματιστικό μοντέλο SIMT μοιάζει αρκετά με το μοντέλο SIMD με τη διαφορά ότι ένας SIMT πολυεπεξεργαστής μπορεί να εκτελεί ανεξάρτητα πολλά threads σε αντίθεση με έναν SIMD επεξεργαστή που τα εκτελεί πάντα μαζί σε συγχρονισμένα groups. Η αρχιτεκτονική SIMT αναζητεί παραλληλισμό σε επίπεδο δεδομένων (DLP) ανάμεσα στα ανεξάρτητα threads εν αντιθέσει με την SIMD που απαιτεί ο παραλληλισμός σε επίπεδο δεδομένων να είναι ρητά δηλωμένος από το πρόγραμμα (σε κάθε vector instuction). Συνδυάζει τα πλεονεκτήματα και την αποδοτικότητα των SIMD αρχιτεκτονικών και ταυτόχρονα επιτρέπει ευκολότερο προγραμματισμό αφού ο προγραμματιστής γράφει thread-parallel κώδικα και όχι data-parallel.

Μοντέλο μνήμης

Το μοντέλο μνήμης της αρχιτεκτονικής CUDA παρέχει 3 διαφορετικά επίπεδα μνήμης με διαφορετική εμβέλεια το καθένα.

- Αρχικά, κάθε thread έχει την ιδιωτική του τοπική μνήμη (*local memory*) και χρησιμοποιείται για τοπικές μεταβλητές που δεν χωράνε στους καταχωρητές καθώς και για αποθήκευση των stack frames και για register spilling. Βρίσκεται στη μνήμη RAM της κάρτας γραφικών, δηλαδή εκτός του chip.
- Κάθε block έχει πρόσβαση στη μοιραζόμενη μνήμη (*shared memory*) και είναι ορατή σε όλα τα threads που περιέχονται στο block. Η διάρκεια ζωής της είναι όση είναι η διάρκεια ζωής του block. Βρίσκεται εντός του chip και παρέχει ταχύτατη πρόσβαση. Χρησιμοποιείται για ανταλλαγή και διαμοίραση δεδομένων μεταξύ των threads και η έξυπνη χρήση της είναι πολύ σημαντική για την επίτευξη καλών επιδόσεων.
- Τέλος, όλα τα νήματα όλων των blocks (δηλαδή όλο το grid) έχουν πρόσβαση στην καθολική μνήμη (*global memory*), η οποία βρίσκεται εκτός chip στη μνήμη RAM της κάρτας γραφικών.

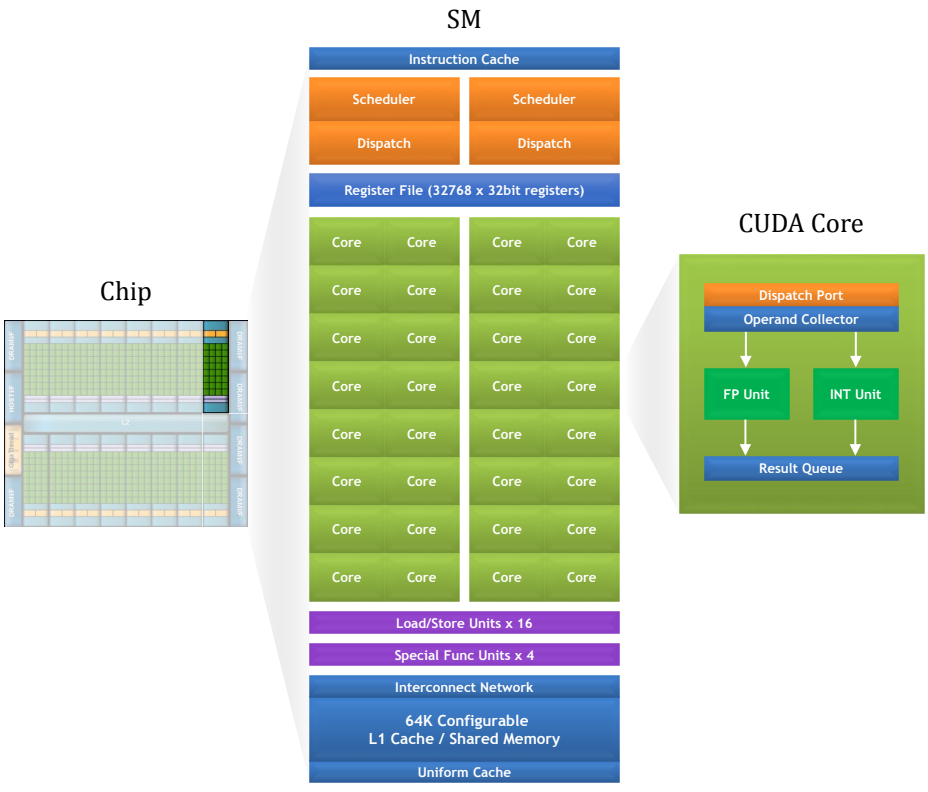


Σχ. 2.10: Μοντέλο μνήμης της αρχιτεκτονικής CUDA [4]

2.2.2. Αρχιτεκτονική Fermi

Στα τέλη του 2009, η εταιρία Nvidia παρουσίασε τη νέα τότε αρχιτεκτονική καρτών γραφικών με το όνομα Fermi και αποτελούσε μεγάλη αναβάθμιση από τις προηγούμενες γενιές. Ήταν η πρώτη αρχιτεκτονική με σαφή προσανατολισμό προς το compute κομμάτι της αγοράς και προσέφερε στους προγραμματιστές περισσότερες δυνατότητες και ευελιξία κατά τη συγγραφή εφαρμογών.

Σχ. 2.11: Block διάγραμμα του chip της αρχιτεκτονικής Fermi



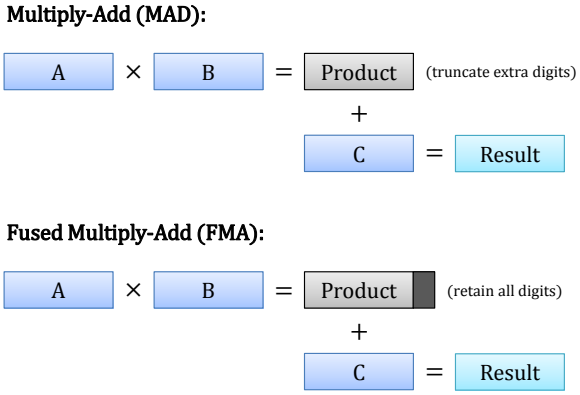
Σχ. 2.12: Διάρθρωση ενός SM της αρχιτεκτονικής Fermi

Streaming Multiprocessor

Το βασικό δομικό στοιχείο των καρτών γραφικών αρχιτεκτονικής Fermi είναι οι πολυεπεξεργαστές Streaming Multiprocessors (SM). Ανάλογα με το μοντέλο της κάρτας γραφικών είναι ενεργοποιημένος διαφορετικός αριθμός από SM. Στο *Σχήμα 2.12* φαίνεται η διάρθρωση ενός Streaming Multiprocessor.

Αναλυτικότερα, κάθε SM αποτελείται από:

32 επεξεργαστικούς πυρήνες CUDA Cores: Κάθε CUDA Core, ή αλλιώς Streaming Processors (SP), διαθέτει πλήρως pipelined αριθμητική μονάδα εκτέλεσης ακέραιων πράξεων (ALU), η οποία υποστηρίζει ακέραιους αριθμούς 64 bit. Επιπλέον, διαθέτει μονάδα εκτέλεσης πράξεων κινητής υποδιαστολής (FPU). Οι 32 CUDA Cores είναι νοητά χωρισμένοι σε δύο ομάδες εκτέλεσης των 16.



Σχ. 2.13: Εντολές MAD και FMA

Η Fermi αρχιτεκτονική είναι η πρώτη που υποστηρίζει το πρότυπο αριθμών κινητής υποδιαστολής IEEE 754-2008 και παρέχει εντολή για εκτέλεση fused multiply-add (FMA) πράξης τόσο για αριθμούς κινητής υποδιαστολής απλής ακρίβειας όσο και για διπλής ακρίβειας. Μπορεί να εκτελεί μία πράξη κινητής υποδιαστολής απλής ακρίβειας ανά κύκλο ρολογιού ενώ απαιτεί δύο κύκλους αν πρόκειται για πράξη διπλής ακρίβειας. Η υποστήριξη της εντολής FMA αποτελεί σημαντική βελτίωση σε σχέση με την multiply-add (MAD) που υποστήριζαν οι προηγούμενες αρχιτεκτονικές, αφού δεν υπάρχει απώλεια ακρίβειας κατά την πράξη της πρόσθεσης, όπως φαίνεται στο *Σχήμα 2.13*.

16 Load/Store units: Οι 16 μονάδες Load/Store κάθε SM επιτρέπουν τον υπολογισμό διευθύνσεων προέλευσης και προορισμού για 16 threads ανά κύκλο ρολογιού. Οι πράξεις load και store γίνονται τόσο από και προς την κύρια μνήμη RAM όσο και την κρυφή μνήμη.

4 Special Function units (SFU): Οι μονάδες SFU εκτελούν transcendental εντολές, όπως ημίτονο (sin), συνημίτονο (cos) ή τετραγωνική ρίζα. Εκτελούν μία εντολή ανά κύκλο ρολογιού. Συνεπώς, για την εκτέλεση όλων των threads ενός warp απαιτούνται 8 κύκλοι ρολογιού.

32K Register File: Κάθε SM διαθέτει τεράστιο register file που αριθμεί 32768 καταχωρητές των 32 bit. Η ύπαρξη τόσο μεγάλου αριθμού καταχωρητών είναι απαραίτητη ώστε να γίνεται το context switching σε μηδενικό χρόνο. Κάθε thread μπορεί να χρησιμοποιήσει το πολύ 63 καταχωρητές.

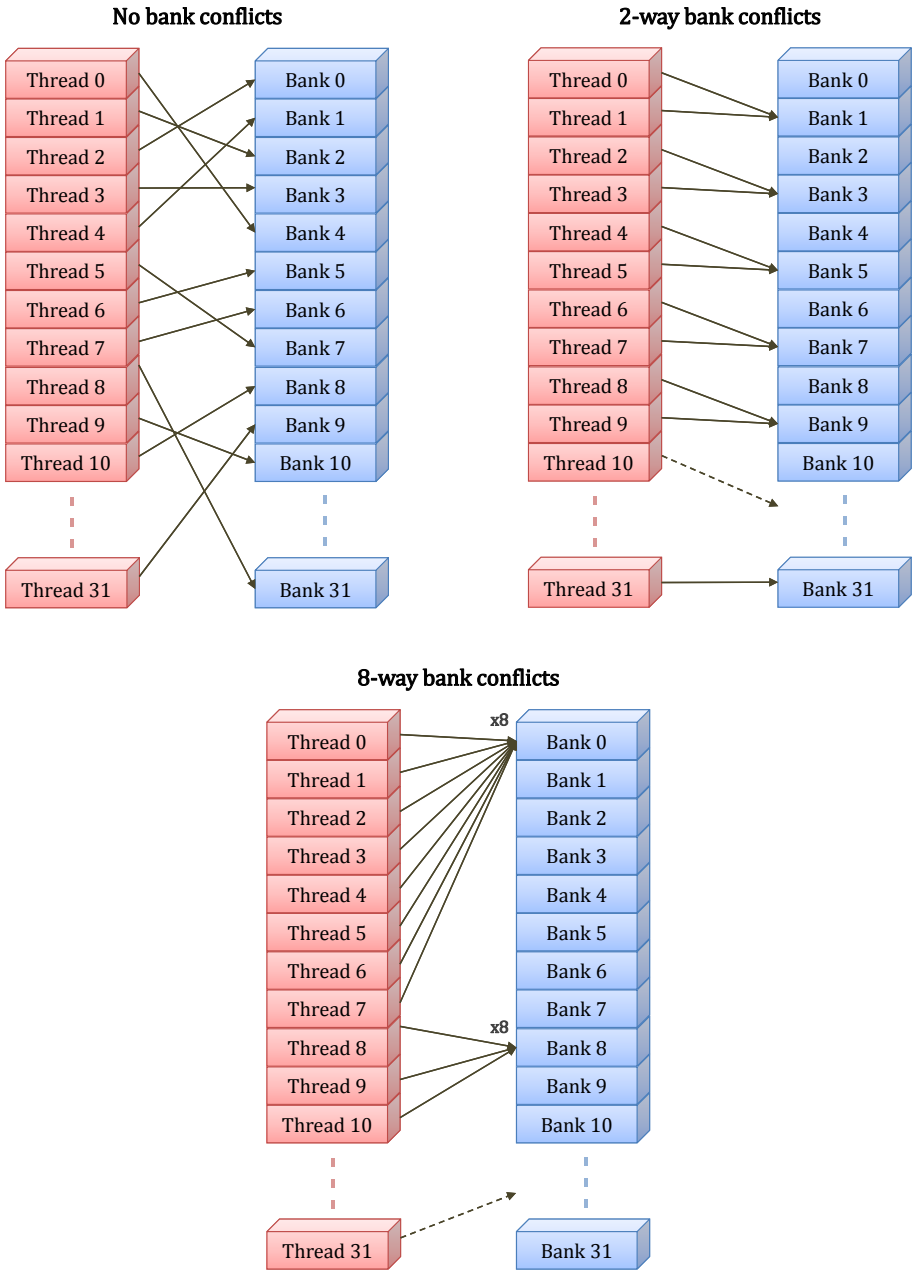
64 KB μνήμης διαμορφώσιμη ως Shared Memory και L1 cache: Συνολικά σε κάθε SM υπάρχουν 64 KB τοπικής γρήγορης μνήμης στην οποία έχουν πρόσβαση όλοι οι CUDA Cores. Ο προγραμματιστής μπορεί να επιλέξει το χωρισμό της μνήμης σε Shared Memory και L1 cache επιλέγοντας ανάμεσα σε δύο διαμορφώσεις: είτε 48 KB για την Shared Memory και 16 KB για L1 cache είτε το ανάποδο. Το τι θα αποθηκεύεται στην Shared Memory ορίζεται από τον προγραμματιστή, οπότε μπορεί να χαρακτηριστεί ως μια «χειροκίνητα» ελεγχόμενη κρυφή μνήμη.

Dual Warp Scheduler: Στους SM γίνεται η δρομολόγηση των warps προς εκτέλεση. Κάθε SM διαθέτει διπλούς warp schedulers και δύο instruction dispatch units έτσι ώστε δύο διαφορετικά warps να μπορούν να δρομολογηθούν και να εκτελεστούν ταυτόχρονα. Οι δρομολογητές επιλέγουν δύο warps τα οποία είναι έτοιμα να εκτελεστούν και κάνουν issue μια εντολή από κάθε warp σε μια από της δύο ομάδες εκτέλεσης των 16 CUDA Cores. Επειδή τα warps είναι ανεξάρτητα μεταξύ τους, δεν υπάρχει η ανάγκη για έλεγχο εξαρτήσεων μεταξύ τους και συνεπώς επιτυγχάνονται μεγάλες επιδόσεις δρομολόγησης.

Shared Memory

Η χρήση της ταχύτατης on-chip Shared Memory από τον προγραμματιστή είναι καταλυτική για την επίτευξη υψηλών επιδόσεων. Ορίζεται σε επίπεδο block και είναι ορατή σε όλα τα threads του block. Όπως ήδη αναφέρθηκε προηγουμένως, το μέγεθος της μπορεί να ορισθεί ίσο με 48 KB ή 16 KB. Η πρόσβαση στη Shared Memory γίνεται μέσω 32 banks. Τα δεδομένα αποθηκεύονται με τέτοιο τρόπο ώστε διαδοχικές 32 bit λέξεις να αποθηκεύονται σε διαδοχικά banks.

Η μεταφορά δεδομένων από διαφορετικά banks γίνεται ταυτόχρονα με μέγιστο εύρος ανά bank ίσο με 32 bit ανά 2 κύκλους ρολογιού. Όμως, όταν συμπίπτουν ταυτόχρονα δύο ή παραπάνω αιτήσεις πρόσβασης στο ίδιο bank, τότε συμβαίνει αυτό που ονομάζεται bank conflict και οι αιτήσεις δεν εξυπηρετούνται ταυτόχρονα αλλά σειριακά. Το γεγονός αυτό μπορεί να έχει σημαντική επίπτωση στις επιδόσεις του προγράμματος και συνεπώς ο προγραμματιστής οφείλει να λάβει υπ' όψιν του τον τρόπο με τον οποίο οργανώνεται η Shared Memory. Η μέγιστη απόδοση επιτυγχάνεται όταν διαφορετικά threads διαβάσουν ή γράφουν σε διαφορετικά banks. Στο *Σχήμα 2.14* φαίνονται περιπτώσεις αποδοτικής και μη πρόσβασης.



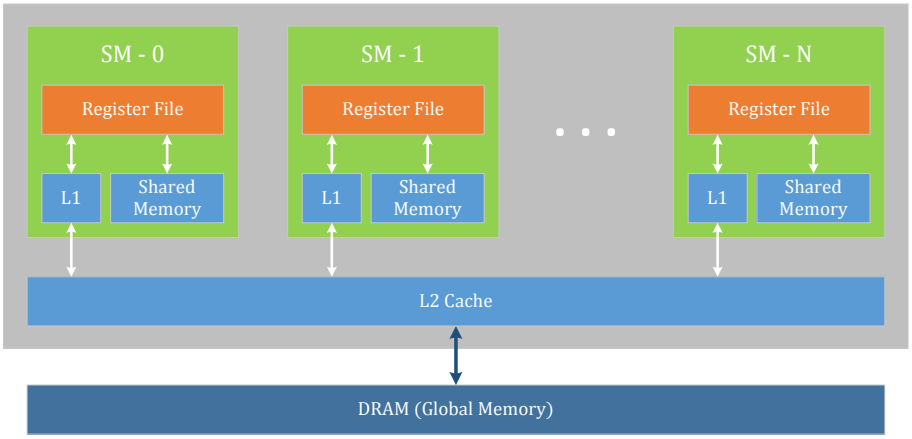
Σχ. 2.14: Bank conflicts κατά την πρόσβαση στη Shared Memory

Ιεραρχία Μνήμης

Η κύρια μνήμη RAM της κάρτας γραφικών συνδέεται με το chip μέσω έξι καναλιών DRAM πλάτους 64 bit και υποστηρίζεται μνήμη τεχνολογίας GDDR5. Το μέγιστο συνολικό εύρος ζώνης που επιτυγχάνεται ανέρχεται στα 177 GB/s. Επιπρόσθετα, υπάρχει L2 cache μεγέθους 768 KB, στην οποία έχουν πρόσβαση όλοι οι SM του chip. Η σύνδεση με το υπόλοιπο σύστημα γίνεται μέσω διαύλου PCI Express 2.0 με μέγιστο εύρος ζώνης 8 GB/s ανά κατεύθυνση.

Η αρχιτεκτονική Fermi είναι η πρώτη που υποστηρίζει ECC προστασία στη μεταφορά δεδομένων από και προς τη μνήμη RAM, χαρακτηριστικό που έχει ξεκάθαρο προσανατολισμό προς το compute κομμάτι της αγοράς και υποστηρίζεται μόνο από τις επαγγελματικές σειρές καρτών γραφικών Tesla και Quadro. Επιπλέον, όλες οι μεταφορές δεδομένων στις on-chip

μνήμες (L1, Shared Memory, L2) προστατεύονται επίσης με ECC. Στο *Σχήμα 2.15* φαίνεται το διάγραμμα ιεραρχίας όλων των επιπέδων μνήμης.



Σχ. 2.15: Ιεραρχία μνήμης αρχιτεκτονικής Fermi

Στον παρακάτω πίνακα παρουσιάζονται αναλυτικά όλα τα είδη μνήμης που υποστηρίζει η πλατφόρμα CUDA, η φυσική μνήμη στην οποία αποθηκεύονται καθώς επίσης η εμβέλεια και το ύψος του latency.

Memory	Stored in	Latency	Scope
register	register file	1 cycle	thread
local	DRAM	400 - 600 cycles	thread
shared	Shared Memory	1 - 32 cycles	block
global	DRAM	400 - 600 cycles	grid
constant	DRAM	400 - 600 cycles	grid

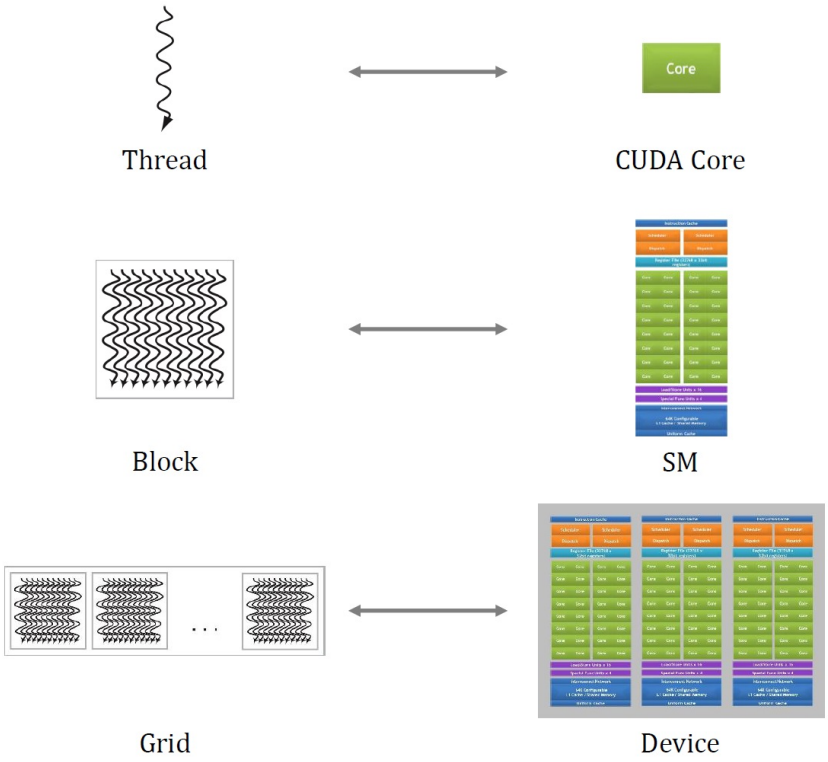
Πίν. 2.1: Είδη μνήμης της πλατφόρμας CUDA

Μοντέλο εκτέλεσης

Όταν μία εφαρμογή αποστέλλεται στην κάρτα γραφικών, όλα τα threads έχουν οργανωθεί σε blocks και τα blocks σε ένα grid το οποίο τελικά ανατίθεται στην κάρτα γραφικών για εκτέλεση. Ανάλογα με τον αριθμό των threads που περιέχουν τα blocks, κάθε SM αναλαμβάνει να εκτελέσει συγκεκριμένο αριθμό από blocks. Οι SM έχουν συγκεκριμένη χωρητικότητα από threads τα οποία μπορούν να είναι ενεργά κάθε στιγμή. Ο μέγιστος αριθμός ενεργών thread ανέρχεται σε 1536 threads (συνεπώς το πολύ 48 warps) ενώ κάθε block μπορεί να περιέχει μέχρι 1024 threads το πολύ. Ο μέγιστος αριθμός blocks ανά SM είναι 8 blocks. Ο προγραμματιστής

πρέπει να επιλέξει κατάλληλο αριθμό threads ανά block ώστε να επιτύχει βέλτιστη κατανομή των blocks και 100% occupancy των SM.

Στη συνέχεια, ο δρομολογητής κάθε πολυεπεξεργαστή SM (GigaThread scheduler) αναλαμβάνει να δρομολογήσει τα warps των threads από τα block που του έχουν ανατεθεί και τελικά κάθε CUDA Core αναλαμβάνει να εκτελέσει ένα thread κάθε στιγμή. Στο *Σχήμα 2.16* αποτυπώνεται η κατανομή που περιγράφηκε.



Σχ. 2.16: Μοντελο εκτέλεσης CUDA

2.2.3. Αρχιτεκτονική Kepler

Το 2012 η εταιρία Nvidia παρουσίασε την διάδοχη της αρχιτεκτονικής Fermi η οποία ονομάζεται Kepler. Η νέα αρχιτεκτονική προσφέρει θεαματική αύξηση των επιδόσεων, διατηρώντας την κατανάλωση στα ίδια επίπεδα. Αυτό οφείλεται στη χρήση λιθογραφίας 28 nm σε σχέση με την λιθογραφία 40 nm που χρησιμοποιούσαν τα chip της αρχιτεκτονικής Fermi. Επιπλέον, εισαγάγει πολλά νέα χαρακτηριστικά και καινοτομίες με στόχο την βελτίωση των compute δυνατοτήτων της κάρτας γραφικών. Θα αναφερθούν συνοπτικά μόνο οι αλλαγές εκείνες οι οποίες αφορούν την παρούσα διπλωματική εργασία.

Νέα αρχιτεκτονική Streaming Multiprocessor (SMX)

Η κυριότερη αλλαγή της αρχιτεκτονικής Kepler είναι ότι αναβαθμίζει σημαντικά τους πολυεπεξεργαστές SM της γενιάς Fermi και πλέον ονομάζονται SMX. Στο *Σχήμα 2.17* φαίνεται η διάρθρωση ενός SMX.

Σχ. 2.17: Διάρθρωση ενός SMX της αρχιτεκτονικής Kepler

Οι αλλαγές ενός Kepler SMX, εν συγκρίσει με έναν Fermi SM, είναι οι εξής:

192 single precision CUDA Cores: Ο αριθμός των CUDA Cores αυξάνεται σημαντικά από 32 σε 192. Κάθε CUDA Core εξακολουθεί να διαθέτει πλήρως pipelined αριθμητική μονάδα εκτέλεσης ακέραιων πράξεων (ALU) και πράξεων κινητής υποδιαστολής απλής ακρίβειας (FPU) με υποστήριξη του προτύπου αριθμών κινητής υποδιαστολής IEEE 754-2008 και επίσης της εντολής FMA. Όμως, σε αντίθεση με την αρχιτεκτονική Fermi, δεν εκτελεί πράξεις κινητής υποδιαστολής διπλής ακρίβειας, παρέχοντας ειδικές μονάδες για αυτό το σκοπό.

64 double precision units: Σε κάθε SMX ενσωματώνονται 64 μονάδες εκτέλεσης πράξεων κινητής υποδιαστολής διπλής ακρίβειας, διπλασιάζοντας τις επιδόσεις σε σχέση με την αρχιτεκτονική Fermi. Αυτό έγινε λόγω της μεγάλης ζήτησης από το HPC κομμάτι της αγοράς για υψηλές επιδόσεις σε αυτόν τον τομέα.

Quad Warp Scheduler: Πλέον, κάθε SMX ενσωματώνει τέσσερις warp schedulers και οκτώ instruction dispatch units. Έτσι, μπορεί να δρομολογεί και να εκτελεί ταυτόχρονα τέσσερα warps (αντί για δύο). Οι δρομολογητές επιλέγουν τέσσερα warps τα οποία είναι έτοιμα να εκτελεστούν και μπορούν να κάνουν issue δύο ανεξάρτητες μεταξύ τους εντολές ανά warp.

64K Register File: Το μέγεθος του register file διπλασιάζεται και φτάνει τους 65536 καταχωρητές των 32 bit. Η αύξηση του ήδη μεγάλου αριθμού καταχωρητών της αρχιτεκτονικής Fermi είναι απαραίτητη για την υποστήριξη των περισσότερων ταυτοχρόνως εκτελούμενων warps. Επίσης, πλέον κάθε thread μπορεί να χρησιμοποιήσει μέχρι 255 καταχωρητές.

32 Load/Store units: Οι μονάδες Load/Store διπλασιάζονται από 16 σε 32 ανά SMX, προκειμένου να ανταποκριθούν στον αυξημένο αριθμό των επεξεργαστικών μονάδων και των warps που μπορούν να εκτελούνται ταυτόχρονα.

32 Special Function units (SFU): Οι μονάδες SFU οκταπλασιάζονται από 4 σε 32.

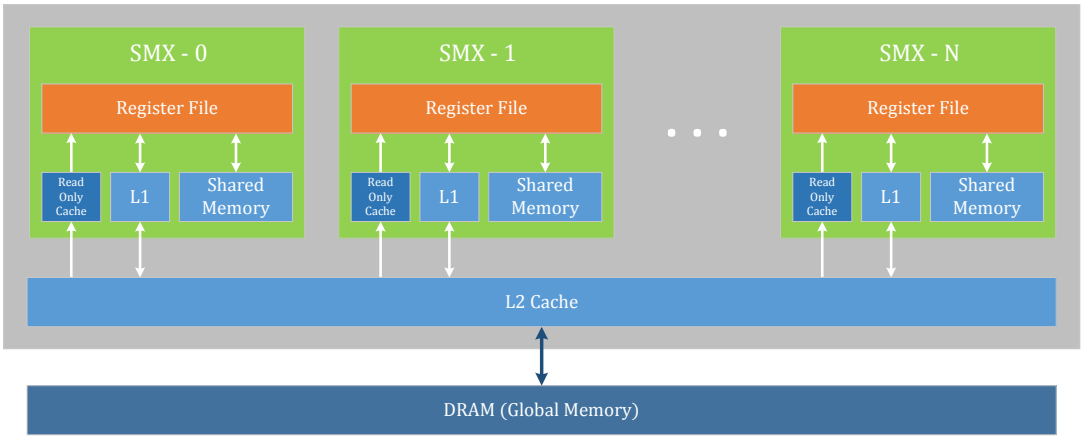
64 KB μνήμης διαμορφώσιμη ως Shared Memory και L1 cache: Το μέγεθος της Shared Memory διατηρείται στα 64 KB. Όμως, στις επιλογές διαμόρφωσης του μεγέθους των Shared Memory / L1 Cache προστίθεται η επιλογή 32 KB / 32 KB, εκτός από τις 48 KB / 16 KB και 16 KB / 48 KB. Επίσης, διπλασιάζεται το εύρος ζώνης κάθε bank της Shared Memory.

48 KB Read-Only Data Cache: Τέλος, επιπροσθέτως της L1 cache, η αρχιτεκτονική Kepler εισαγάγει μια κρυφή μνήμη 48 KB ανά SMX για την αποθήκευση δεδομένων, η οποία είναι μόνο για ανάγνωση (Read-Only). Στην αρχιτεκτονική Fermi, η συγκεκριμένη μνήμη ήταν προσβάσιμη μόνο από τα Texture Units και ο προγραμματιστής έπρεπε να δηλώσει ρητά ως textures τα δεδομένα που ήθελε να φορτώσει, γεγονός το οποίο είχε αρκετούς περιορισμούς και επιπλέον επίπεδο δυσκολίας.

Ιεραρχία Μνήμης

Η κύρια μνήμη RAM της κάρτας γραφικών συνεχίζει να συνδέεται με το chip μέσω έξι καναλιών DRAM πλάτους 64 bit και υποστηρίζεται ταχύτερη μνήμη τεχνολογίας GDDR5. Συνεπώς, το μέγιστο συνολικό εύρος ζώνης που επιτυγχάνεται ανέρχεται στα 288 GB/s. Επιπρόσθετα, το μέγεθος της L2 cache αυξάνεται από 768 KB σε 1536 KB και το εύρος ζώνης της ανά κύκλο ρολογιού διπλασιάζεται. Η σύνδεση με το υπόλοιπο σύστημα γίνεται πλέον μέσω διαύλου PCI Express 3.0 με μέγιστο εύρος ζώνης 16 GB/s ανά κατεύθυνση.

Όπως και στην αρχιτεκτονική Fermi, υποστηρίζεται ECC προστασία στη μεταφορά δεδομένων από και προς όλα τα επίπεδα μνήμης.



Σχ. 2.18: Ιεραρχία μνήμης αρχιτεκτονικής Kepler

Μοντέλο εκτέλεσης

Πλέον, ο μέγιστος αριθμός ενεργών thread ανά SMX ανέρχεται σε 2048 threads, ενώ κάθε block συνεχίζει να περιέχει μέχρι 1024 threads το πολύ. Όμως, ο μέγιστος αριθμός blocks ανά SMX διπλασιάζεται από 8 σε 16 blocks.

2.2.4. Γλώσσα προγραμματισμού CUDA C

Όπως έχει ήδη αναφερθεί, η CUDA C είναι μια επέκταση της γλώσσας προγραμματισμού C και παρέχει ειδικό compiler για τη μεταγλώττιση εφαρμογών στην κάρτα γραφικών. Επιπλέον, παρέχει κατάλληλες συναρτήσεις και οδηγούς για την αλληλεπίδραση με την κάρτα γραφικών, την αντιγραφή δεδομένων από και προς την κάρτα καθώς και την αποστολή του τμήματος του κώδικα για εκτέλεση στην κάρτα γραφικών.

Στον *Κώδικα 2.5* που ακολουθεί φαίνεται ένα απλό παράδειγμα προγράμματος γραμμένο σε CUDA C που προσθέτει δύο διανύσματα (a, b) σε ένα τρίτο (c).

Κώδικας 2.5 Παράδειγμα προγράμματος σε CUDA C για πρόσθεση διανυσμάτων

```
#define N (1024*1024)
#define THREADS_PER_BLOCK 256

__global__ void add(int *a, int *b, int *c)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    c[index] = a[index] + b[index];
}

int main()
{
    int *a, *b, *c;           // host copies of a, b, c
    int *d_a, *d_b, *d_c;     // device copies of a, b, c

    // Host space allocation and initialization
    ...

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    // Copy inputs to device
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

    // Launch add() kernel on GPU
    dim3 grid(N / THREADS_PER_BLOCK);
    dim3 block(THREADS_PER_BLOCK);
    add<<<grid, block>>>(d_a, d_b, d_c);

    // Copy result back to host
    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

    // Cleanup
    free(a); free(b); free(c);
    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    return 0;
}
```

Αρχικά, με τη συνάρτηση `cudaMalloc()` δεσμεύεται χώρος στη μνήμη της κάρτας γραφικών για την αποθήκευση των δύο διανυσμάτων που θα προστεθούν και για το διάνυσμα που θα αποθηκευτεί το αποτέλεσμα.

Στη συνέχεια, με τη συνάρτηση `cudaMemcpy()` αντιγράφονται τα δεδομένα των διανυσμάτων από τη μνήμη του συστήματος στη μνήμη της κάρτας γραφικών. Το τέταρτο όρισμα της συνάρτησης δηλώνει την κατεύθυνση αντιγραφής δεδομένων, δηλαδή:

- `cudaMemcpyHostToDevice`: δηλώνει αντιγραφή δεδομένων από την κύρια μνήμη RAM του συστήματος (Host) προς τη μνήμη της κάρτας γραφικών (Device).
- `cudaMemcpyDeviceToHost`: δηλώνει αντιγραφή δεδομένων από τη μνήμη της κάρτας γραφικών (Device) προς την κύρια μνήμη RAM του συστήματος (Host).

Έπειτα, γίνεται η κλήση της συνάρτησης `add()` που είναι το τμήμα του κώδικα που τελικά εκτελείται στην κάρτα γραφικών. Η λέξη κλειδί `__global__` δηλώνει ότι πρόκειται για συνάρτηση η οποία εκτελείται στην κάρτα γραφικών και καλείται από τον επεξεργαστή (Host). Η κλήση γίνεται με μια εντολή της μορφής:

```
function<<<GRID, BLOCK>>>( ... )
```

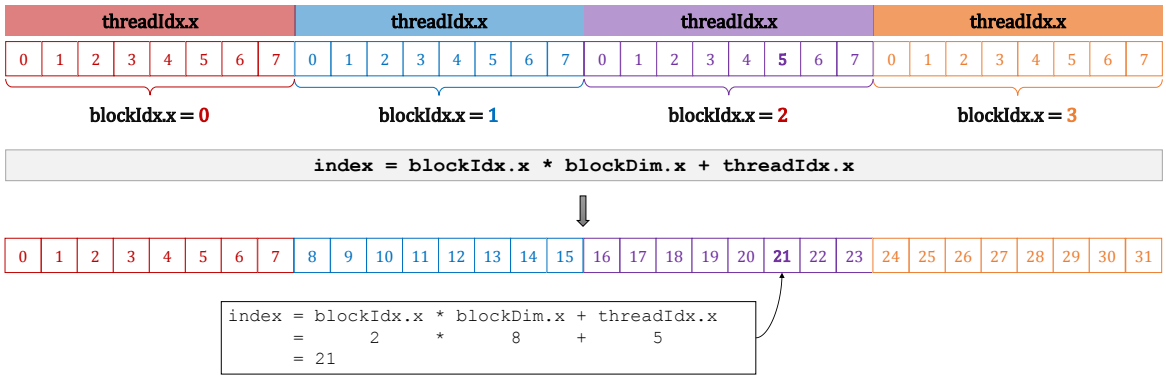
όπου:

- `GRID`: δηλώνει τον τρόπο με τον οποίο οργανώνονται τα blocks στο grid, δηλαδή τις διαστάσεις του grid. Η μεταβλητή `GRID` είναι τύπου `dim3` (ενσωματωμένου vector τύπου της CUDA) και μπορεί να έχει μία, δύο ή τρεις διαστάσεις.
- `BLOCK`: δηλώνει τον τρόπο με τον οποίο οργανώνονται τα threads σε κάθε block. Ομοίως, είναι τύπου `dim3` και μπορεί να έχει έως 3 διαστάσεις.

Στη συνέχεια, ξεκινάει η εκτέλεση της συνάρτησης `add()` στην κάρτα γραφικών. Κάθε thread εκτελεί τον ίδιο κώδικα και επεξεργάζεται ένα στοιχείο κάθε διανύσματος. Ο προσδιορισμός του στοιχείου που επεξεργάζεται κάθε thread υπολογίζεται με βάση κάποιες ενσωματωμένες και αυτόματα ορισμένες μεταβλητές που παρέχει η CUDA:

- `dim3 gridDim`: διαστάσεις του grid σε blocks
- `dim3 blockDim`: διαστάσεις των blocks σε threads
- `dim3 blockIdx`: index του block μέσα στο grid
- `dim3 threadIdx`: index του thread μέσα στο block

Στο παράδειγμα του *Κώδικα 2.5*, τόσο τα blocks όσο και το grid έχουν μία μόνο διάσταση. Στο *Σχήμα 2.19* φαίνεται ο τρόπος με τον οποίο κάθε thread υπολογίζει ποιο στοιχείο των διανυσμάτων θα επεξεργαστεί.



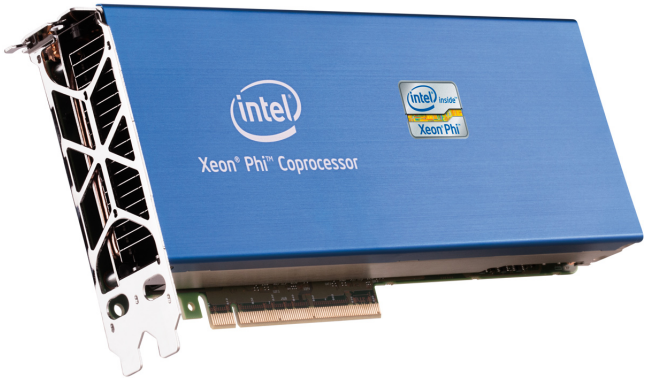
Σχ. 2.19: Παράδειγμα υπολογισμού index ανά thread

Τέλος, μετά την ολοκλήρωση της εκτέλεσης στην κάρτα γραφικών, αντιγράφονται πίσω στην μνήμη του συστήματος τα αποτελέσματα της εκτέλεσης.

2.3. Προγραμματισμός σε Intel Xeon Phi

Ο συνεπεξεργαστής (co-processor) Intel Xeon Phi αποτελεί το πρώτο προϊόν που βασίζεται στην αρχιτεκτονική Many Integrated Core (MIC) της εταιρίας Intel και φέρει την κωδική ονομασία Knights Corner. Στην καρδιά του βρίσκεται ένα chip το οποίο αποτελείται από επεξεργαστικούς πυρήνες Pentium, με αρκετές τροποποιήσεις που θα αναλυθούν παρακάτω.

Η προσέγγιση της αρχιτεκτονικής MIC είναι η ενσωμάτωση επεξεργαστικών πυρήνων μεσαίου μεγέθους, πολυπλοκότητας και κατανάλωσης. Οι πυρήνες Pentium βρίσκονται ανάμεσα στους πολύ μικρούς και απλούς πυρήνες των καρτών γραφικών και στους πολύ μεγάλους και πολύπλοκους πυρήνες των επεξεργαστών. Η προσέγγιση αυτή προσφέρει το πλεονέκτημα ότι πρόκειται για αρχιτεκτονική x86. Έτσι, μπορεί να επωφεληθεί από τη χρήση των ήδη γνωστών x86 προγραμματιστικών μοντέλων. Επιπλέον, ήδη γραμμένες εφαρμογές για x86 επεξεργαστές μπορούν να εκτελεστούν με μικρές τροποποιήσεις.



Σχ. 2.20: Intel Xeon Phi

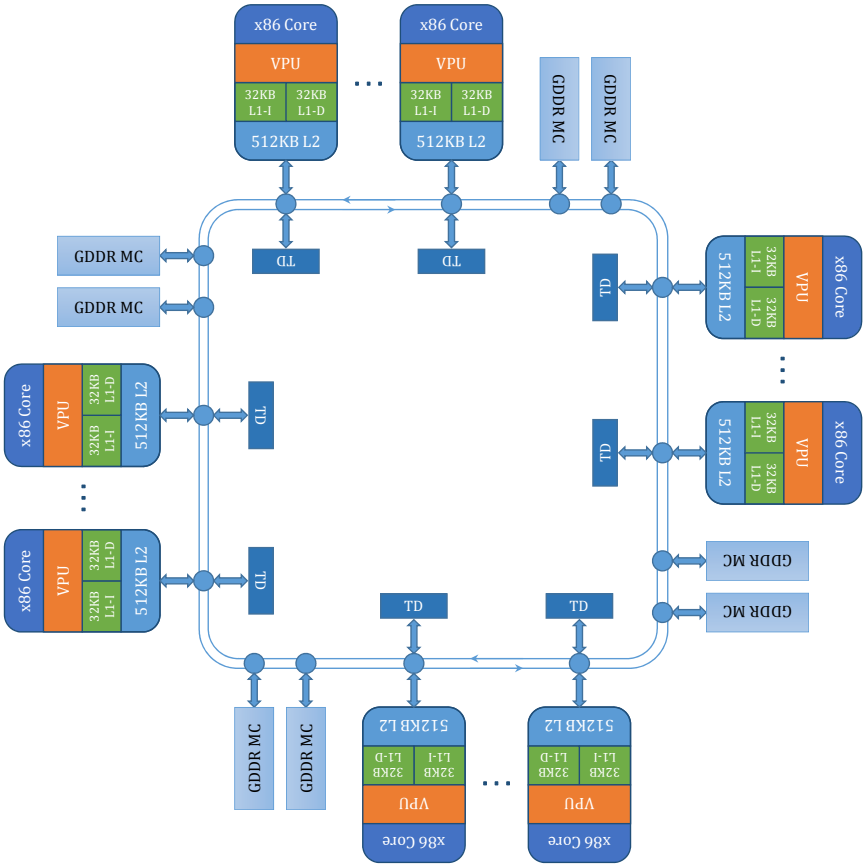
Συνδέεται μέσω διαύλου PCI Express όπως οι κάρτες γραφικών, αλλά ακολουθεί αρκετά διαφορετικό σχεδιασμό. Σε αντίθεση με τις κάρτες γραφικών, ενσωματώνει δικό του λειτουργικό σύστημα που βασίζεται στο Linux. Χρησιμοποιεί γρήγορη μνήμη DRAM τεχνολογίας GDDR5.

2.3.1. Αρχιτεκτονική του Xeon Phi

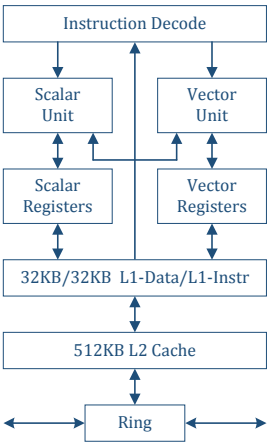
Ο Xeon Phi αποτελείται από:

- x86 επεξεργαστικούς πυρήνες βασισμένους σε πυρήνες Pentium (το γρηγορότερο μοντέλο ενσωματώνει 61 πυρήνες)
- κρυφή μνήμη L2 ανά πυρήνα
- 8 ελεγκτές μνήμης (MC) GDDR5, παρέχοντας μέγιστο συνολικό εύρος ζώνης ίσο με 352 GB/s και υποστηρίζοντας μέχρι 16 GB μνήμης
- διπλής κατεύθυνσης δίαυλο διασύνδεσης τύπου δαχτυλιδιού (bi-directional ring bus) με πολύ υψηλό εύρος ζώνης
- τις απαραίτητες δομές (Tag Directories, TD) ώστε να εξασφαλίζεται η συνέπεια (cache coherency) ανάμεσα σε όλες οι L2 cache.

Ο δίαυλος διασύνδεσης συνδέει όλους τους πυρήνες μεταξύ τους καθώς επίσης με τους ελεγκτές μνήμης και το I/O interface. Η σύνδεση με το υπόλοιπο σύστημα γίνεται μέσω διαύλου PCI Express 2.0, παρέχοντας μέγιστο εύρος ζώνης 8 GB/s ανά κατεύθυνση. Στο *Σχήμα 2.21* φαίνεται ένα block διάγραμμα του Phi ενώ στο *Σχήμα 2.22* φαίνεται η διάρθρωση κάθε επεξεργαστικού πυρήνα που ενσωματώνει ο Xeon Phi.



Σχ. 2.21: Block διάγραμμα του Xeon Phi



Σχ. 2.22: Διάρθρωση ενός πυρήνα του Xeon Phi

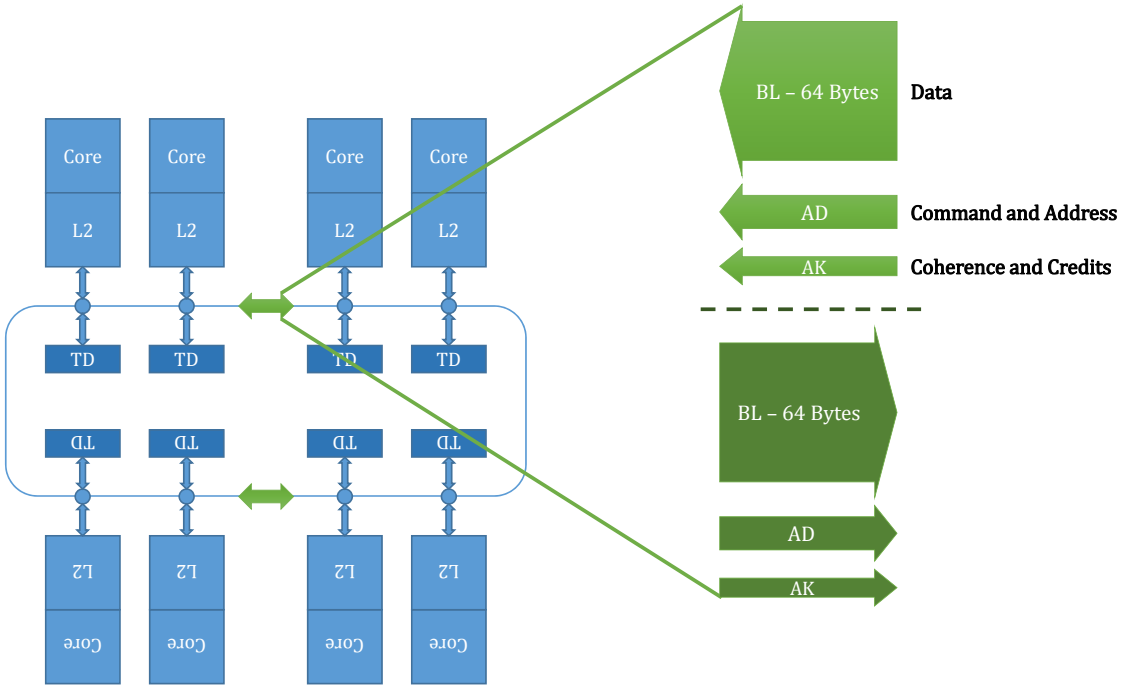
x86 Xeon Phi Core

Κάθε επεξεργαστικός πυρήνας x86 του Xeon Phi αποτελείται από 4 κύρια μέρη:

- Ένα in-order και dual-issue pipeline με υποστήριξη 4-way simultaneous multi-threading (SMT). Το γεγονός ότι είναι in-order (και όχι out-of-order) εξοικονομεί χώρο στο υλικό καθώς και ενέργεια ώστε να είναι εφικτός ο συνδυασμός μέχρι και 61 πυρήνων σε ένα chip.
- Μια SIMD μονάδα διανυσματικής επεξεργασίας (Vector Processing Unit, VPU) πλάτους 512 bit. Μπορεί να εκτελεί 16 πράξεις κινητής υποδιαστολής απλής ακρίβειας, 8 πράξεις διπλής ακρίβειας ή 16 πράξεις ακεραίων 32 bit ανά κύκλο ρολογιού ενώ υποστηρίζει την πράξη Fused-Multiply Add (FMA). Περιέχει register file που αποτελείται από 32 καταχωρητές πλάτους επίσης 512 bit. Δεν υποστηρίζονται οι παλαιότερες SIMD εντολές της Intel (MMX, SSE, AVX) αλλά υποστηρίζεται ένα νέο πακέτο SIMD εντολών με ονομασία AVX-512, ειδικά σχεδιασμένων για το Xeon Phi. Τέλος, περιέχει ειδική μονάδα εκτέλεσης transcendental πράξεων (Extended Math Unit, EMU).
- Κρυφές μνήμες L1 data και L1 instruction μεγέθους 32 KB η κάθε μία.
- Κρυφή μνήμη L2 μεγέθους 512 KB, η οποία είναι πλήρως συνεπής (fully coherent) με τις υπόλοιπες L2 κρυφές μνήμες των άλλων πυρήνων. Η συνέπεια εξασφαλίζεται με χρήση tag directories (TD). Κάθε tag directory αναλαμβάνει ένα συγκεκριμένο τμήμα της μνήμης ενώ όλα τα μηνύματα ανάμεσα στις L2 cache ανταλλάσσονται μέσω του διαύλου διασύνδεσης.

Bi-directional ring

Κάθε κατεύθυνση του διαύλου διασύνδεσης των πυρήνων αποτελείται από 3 διαφορετικά και ανεξάρτητα μεταξύ τους δαχτυλίδια (rings), όπως φαίνεται στο *Σχήμα 2.23*.



Σχ. 2.23: Αναπαράσταση του διαύλου διασύνδεσης διπλής κατεύθυνσης [22]

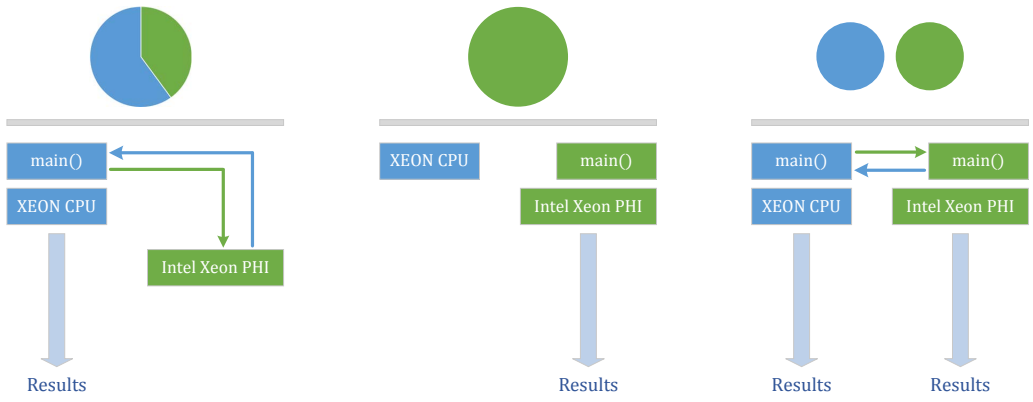
Το πρώτο και μεγαλύτερο δαχτυλίδι (Data ring) είναι αυτό μέσω του οποίου μεταφέρονται τα δεδομένα. Λόγω του μεγάλου αριθμού των πυρήνων και της ανάγκης για υψηλό εύρος ζώνης, υποστηρίζει μεγάλο block μεταφοράς ίσο με 64 Bytes. Το δεύτερο δαχτυλίδι (Command and Address ring) είναι σημαντικά μικρότερο και χρησιμοποιείται για την αποστολή εντολών ανάγνωσης/εγγραφής και διευθύνσεων μνήμης. Τέλος, το τρίτο και τελευταίο δαχτυλίδι (Coherence and Credits ή αλλιώς acknowledgement ring) είναι το μικρότερο από τα τρία και χρησιμοποιείται για την ανταλλαγή μηνυμάτων ελέγχου και του μηχανισμού συνέπειας των κρυφών μνημών L2.

2.3.2. Μοντέλο εκτέλεσης

Ο προγραμματισμός του Xeon Phi γίνεται με τα ίδια προγραμματιστικά μοντέλα τα οποία χρησιμοποιούνται και στους x86 επεξεργαστές. Υποστηρίζει το δημοφιλές πρότυπο παράλληλου προγραμματισμού OpenMP. Η Intel παρέχει κατάλληλους compilers, εργαλεία και οδηγούς για την ανάπτυξη εφαρμογών για τον Xeon Phi και την αλληλεπίδραση με τη συσκευή.

Υπάρχουν τρεις τρόποι να χρησιμοποιηθεί η συσκευή από τον προγραμματιστή. Αναλυτικότερα:

Offload to Phi: Ο προγραμματιστής αντιμετωπίζει τη συσκευή με τον ίδιο τρόπο με τον οποίο αντιμετωπίζονται οι κάρτες γραφικών. Η εφαρμογή αρχικά ξεκινάει την εκτέλεση στον επεξεργαστή. Η Intel παρέχει κατάλληλες οδηγίες (pragma directives) προς τον compiler για την αντιγραφή δεδομένων από και προς τη συσκευή και την εκτέλεση συγκεκριμένων τμημάτων κώδικα στο Xeon Phi (offload). Στη συνέχεια, όμοια με τις κάρτες γραφικών, το αποτέλεσμα από την εκτέλεση στη συσκευή επιστρέφει στην κύρια μνήμη του συστήματος και συνεχίζει η αρχική εκτέλεση, όπως φαίνεται στην *Εικόνα 2.24a*.



Offload (a) **Native** (b) **Symmetric** (c)

Σχ. 2.24: Μοντέλα εκτέλεσης του Intel Xeon Phi

Native Execution: Όπως αναφέρθηκε προηγουμένως, ο Xeon Phi ενσωματώνει δικό του λειτουργικό σύστημα βασισμένο στο Linux. Ο προγραμματιστής μεταγλωττίζει την εφαρμογή του ώστε να μπορεί να εκτελεστεί εγγενώς (natively) στον Xeon Phi. Δηλαδή, μπορεί να αντιμετωπιστεί ως ένας ξεχωριστός υπολογιστής ακολουθώντας το μοντέλο μοιραζόμενης μνήμης. Η μνήμη του Xeon Phi χρησιμοποιείται ως κύρια μνήμη από την εφαρμογή. Αυτός ο τρόπος λειτουργίας χρησιμοποιείται από την παρούσα διπλωματική εργασία.

Symmetric MPI: Τέλος, ο επεξεργαστής και ο Xeon Phi μπορούν να λειτουργήσουν συμμετρικά ως MPI targets. Απαιτούνται δύο ξεχωριστά εκτελέσιμα αρχεία, ένα μεταγλωττισμένο για εκτέλεση στον επεξεργαστή και το άλλο για εκτέλεση στη συσκευή. Ο προγραμματιστής οφείλει να προσέξει το μοίρασμα του επεξεργαστικού φόρτου ανάμεσα στους «μεγάλους» πυρήνες του επεξεργαστή (ή των επεξεργαστών) και τους «μικρούς» πυρήνες του Xeon Phi.

2.4. Πειραματικός Εξοπλισμός - Πλατφόρμες Δοκιμών

Στην παρούσα διπλωματική εργασία αξιολογούνται τρεις διαφορετικές αρχιτεκτονικές παράλληλου προγραμματισμού (CPU, GPU, Intel MIC - Xeon Phi), οι οποίες περιγράφηκαν προηγουμένως. Χρησιμοποιούνται «εκπρόσωποι» από κάθε αρχιτεκτονική. Στη συνέχεια, παρατίθεται χωριστά κατά αρχιτεκτονική όλος ο εξοπλισμός ο οποίος θα χρησιμοποιηθεί.

2.4.1. Συστήματα πολλαπλών επεξεργαστών

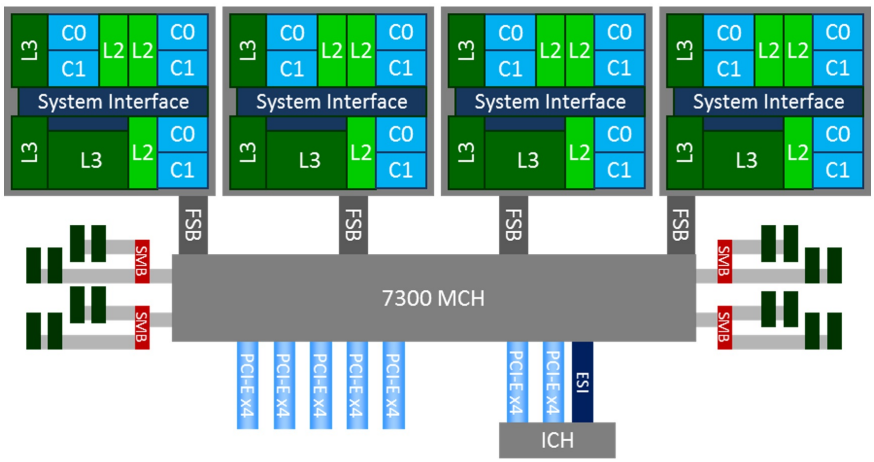
Θα χρησιμοποιηθούν δύο διαφορετικά συστήματα πολλαπλών επεξεργαστών, καθένα από τα οποία αποτελείται από 4 επεξεργαστές. Το πρώτο και λιγότερο σύγχρονο βασίζεται στην αρχιτεκτονική Dunnington της Intel, ενώ το δεύτερο και πιο σύγχρονο βασίζεται στην αρχιτεκτονική Sandy Bridge - EP της ίδιας εταιρίας.

Dunnington

Η αρχιτεκτονική Dunnington της Intel ανήκει στην κατηγορία των Uniform Memory Access (UMA) αρχιτεκτονικών και υποστηρίζει επεξεργαστές μέχρι 6 πυρήνες ο καθένας, βασισμένους στην αρχιτεκτονική Penryn.

Η σύνδεση των επεξεργαστών με τον Memory Controller Hub (MCH) γίνεται μέσω διαύλου FSB (Front Side Bus), ο οποίος δεν ανήκει στην κατηγορία διαύλων υψηλού εύρους ζώνης. Την εποχή όμως που παρουσιάστηκε η πλατφόρμα Dunnington (2008) δεν υπήρχαν πολλές εναλλακτικές.

Στο *Σχήμα 2.25* φαίνεται η δομή της αρχιτεκτονικής Dunnington.



Σχ. 2.25: Αρχιτεκτονική Dunnington [24]

Το σύστημα που θα χρησιμοποιηθεί αποτελείται από τέσσερις εξαπύρηνους επεξεργαστές Intel Xeon X7460 χρονισμένους στα 2.66 GHz. Κάθε πυρήνας έχει προσωπικές κρυφές μνήμες L1 Data και L1 Instruction μεγέθους 32 KB έκαστη. Ανά δύο πυρήνες μοιράζονται κρυφή μνήμη L2 μεγέθους 3 MB η κάθε μία. Επιπλέον, όλοι οι πυρήνες έχουν πρόσβαση σε κρυφή μνήμη L3 μεγέθους 16 MB. Τέλος, έχει μέγιστη κατανάλωση ενέργειας ίση με 130 W. Στον *Πίνακα 2.2* φαίνονται αναλυτικά τα χαρακτηριστικά των εν λόγω επεξεργαστών.

Intel Xeon X7460			
Number of cores	6	**L1 Cache**	6 × 32KB / 32KB
Clock Speed	2.66 GHz	**L2 Cache**	3 × 3 MB
FSB Speed	1066 MHz	**L3 Cache**	16 MB
Lithografy	45 nm	**Latest SIMD Support**	SSE4.1
TDP	130 W	**Microarchitecture**	Penryn

Πίν. 2.2: Χαρακτηριστικά του επεξεργαστή Intel Xeon X7460

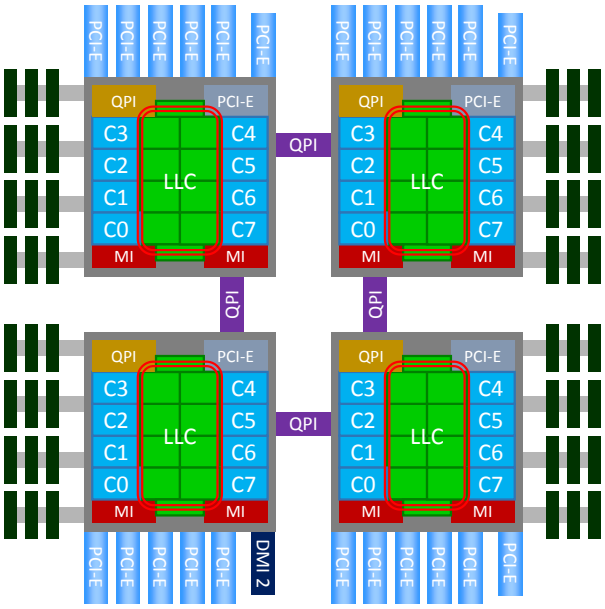
Sandy Bridge - EP

Η αρχιτεκτονική Sandy Bridge - EP της Intel ανήκει στην κατηγορία των Non-Uniform Memory Access (NUMA) αρχιτεκτονικών και υποστηρίζει επεξεργαστές μέχρι 8 πυρήνες ο καθένας, βασισμένους στην ομώνυμη αρχιτεκτονική της εταιρίας.

Κάθε επεξεργαστής διαθέτει ενσωματωμένο ελεγκτή μνήμης (Intergrated Memory Controller, IMC). Η διασύνδεση των επεξεργαστών μεταξύ τους γίνεται μέσω της δεύτερης γενιάς του διαύλου Intel QuickPath Interconnect 1.1 (QPI), ο οποίος, ανάλογα με τη συχνότητα λειτουργίας, προσφέρει εύρος ζώνης μέχρι 16 GB/s ανά κατεύθυνση.

Η διασύνδεση και η επικοινωνία των επεξεργαστικών πυρήνων εντός του chip γίνεται με χρήση ενός διπλής κατεύθυνσης διαύλου τύπου δαχτυλιδιού (bi-directional ring).

Στο *Σχήμα 2.26* φαίνεται η δομή της αρχιτεκτονικής Sandy Bridge - EP και η εσωτερική διασύνδεση των πυρήνων των επεξεργαστών.



Σχ. 2.26: Αρχιτεκτονική Sandy Bridge - EP [24]

Το σύστημα που θα χρησιμοποιηθεί αποτελείται από τέσσερις οκταπύρηνους επεξεργαστές Intel Xeon E5-4620 χρονισμένους στα 2.2 GHz, που υποστηρίζουν την τεχνολογία Intel Hyper-Treading. Κάθε πυρήνας έχει προσωπικές κρυφές μνήμες L1 Data και L1 Instruction μεγέθους 32 KB έκαστη και L2 μεγέθους 256 KB. Επιπλέον, όλοι οι πυρήνες μοιράζονται κρυφή μνήμη L3 μεγέθους 16 MB.

Επιπρόσθετα, υποστηρίζει την τεχνολογία Intel Turbo Boost (με μέγιστη συχνότητα 2.6 GHz). Τέλος, υποστηρίζει το x86 SIMD σετ εντολών AVX (πλάτους 256 bit) και έχει μέγιστη κατανάλωση ενέργειας 95 W.

Στον *Πίνακα 2.3* φαίνονται αναλυτικά τα χαρακτηριστικά των εν λόγω επεξεργαστών.

Intel Xeon E5-4620			
Number of cores	8	**L1 Cache**	8 × 32KB / 32KB
Clock Speed	2.2 GHz (max 2.6)	**L2 Cache**	8 × 256 KB
QPI Speed	7.2 GT/s	**L3 Cache**	16 MB
Lithografy	32 nm	**Latest SIMD Support**	SSE4.2, AVX
TDP	95 W	**Microarchitecture**	Sandy Bridge - E

Πίν. 2.3: Χαρακτηριστικά του επεξεργαστή Intel Xeon E5-4620

Μέγιστες θεωρητικές επιδόσεις (FLOP/s)

Στον *Πίνακα 2.4* φαίνονται οι μέγιστες θεωρητικές επιδόσεις πράξεων κινητής υποδιαστολής απλής και διπλής ακρίβειας. Υπολογίζονται με βάση τους παρακάτω τύπους:

Penryn (SSE):

- 4 Double Precision (DP) FLOPs/κύκλο ανά πυρήνα
- 8 Single Precision (SP) FLOPs/κύκλο ανά πυρήνα

Sandy Bridge (AVX):

- 8 Double Precision (DP) FLOPs/κύκλο ανά πυρήνα
- 16 Single Precision (SP) FLOPs/κύκλο ανά πυρήνα

(GFLOPS/s)	**Single Precision (32 bit)**	**Double Precison (64 bit)**
Intel Xeon x7460 (6C)	127.68	63.84
Intel Xeon E5-4620 (8C)	281.6	140.8
4P Intel Xeon x7460	510.72	255.36
4P Intel Xeon E5-4620	1126.4	563.2

Πίν. 2.4: Μέγιστες θεωρητικές επιδόσεις (GFLOP/s)

2.4.2. Κάρτες Γραφικών

Θα χρησιμοποιηθούν κάρτες γραφικών των δύο αρχιτεκτονικών Fermi και Kepler της εταιρίας Nvidia, που αναλύθηκαν στην *Ενότητα 2.2*. Από την αρχιτεκτονική Fermi χρησιμοποιείται η κάρτα γραφικών Tesla M2050 με 3 GB μνήμη και από την αρχιτεκτονική Kepler η Tesla K40 με 12 GB μνήμη, η οποία είναι η γρηγορότερη και πιο σύγχρονη κάρτα της αρχιτεκτονικής Kepler. Στον *Πίνακα 2.5* παρατίθενται αναλυτικά τα χαρακτηριστικά των προαναφερθέντων καρτών γραφικών.

	Nvidia Tesla M2050	**Nvidia Tesla K40**
Peak SP	1030 GFLOP/s	4290 GFLOP/s
Peak DP	515 GFLOP/s	1430 GFLOP/s
Memory Size	3 GB GDDR5	12 GB GDDR5
Memory Bandwidth	144 GB/s	288 GB/s
# of CUDA Cores	448	2880
TDP	237 W	245 W
PCIe Gen	2.0	3.0
Architecture	Fermi (40 nm)	Kepler (28 nm)

Πίν. 2.5: Χαρακτηριστικά καρτών γραφικών Tesla M2050 και Tesla K40

2.4.3. Intel Xeon Phi

Ως εκπρόσωπος της αρχιτεκτονικής Intel MIC χρησιμοποιείται το μοντέλο Intel Xeon Phi 5120D. Στον *Πίνακα 2.6* παρατίθενται αναλυτικά τα χαρακτηριστικά του συγκεκριμένου μοντέλου.

	Intel Xeon Phi 5120D
Peak SP	2022 GFLOP/s
Peak DP	1011 GFLOP/s
Memory Size	8 GB
Memory Bandwidth	352 GB/s
# of Cores	60 (max 240 threads)
Clock Speed	1.053 GHz
TDP	245 W
PCIe Gen	2.0
Total Cache	30 MB
Turbo Enabled	No

Πίν. 2.6: Χαρακτηριστικά Intel Xeon Phi 5120D

Κεφάλαιο 3

Αλγόριθμοι

Στο κεφάλαιο αυτό θα παρατεθούν οι δύο αλγόριθμοι οι οποίοι χρησιμοποιούνται στην παρούσα διπλωματική εργασία και παραλληλοποιούνται στις αρχιτεκτονικές που αναφέρθηκαν στο Κεφάλαιο 2.

3.1. Floyd - Warshall

3.1.1. Ορισμοί

Γράφος (ή γράφημα) είναι ένα διατεταγμένο ζεύγος $G = (V, E)$ αποτελούμενο από το σύνολο V του οποίου τα στοιχεία ονομάζονται κόμβοι ή κορυφές και το σύνολο E του οποίου τα στοιχεία ονομάζονται ακμές και αποτελούν ζεύγη από κορυφές. Ένας γράφος είναι μη-κατευθυνόμενος όταν τα ζεύγη κορυφών του συνόλου E είναι μη διατεταγμένα, δηλαδή οι ακμές δεν έχουν προσανατολισμό. Κατ' αντιστοιχία, ένας γράφος είναι κατευθυνόμενος όταν τα ζεύγη κορυφών είναι διατεταγμένα, δηλαδή οι ακμές είναι κατευθυνόμενες (έχουν προσανατολισμό). Όταν κάθε ακμή αντιστοιχεί με ένα πραγματικό αριθμό τότε πρόκειται για γράφο με βάρη και ο αριθμός αυτός ονομάζεται βάρος της ακμής.

Ο αριθμός των ακμών ενός γράφου βρίσκεται στο διάστημα $[0, |V|^2]$. Όταν ο αριθμός αυτός βρίσκεται πιο κοντά στο 0, πρόκειται για αραιό γράφο. Αντίθετα, όταν βρίσκεται πιο κοντά στο $|V|^2$, πρόκειται για πυκνό γράφο.

Η χρησιμότητα των γράφων στην πληροφορική είναι μεγάλη καθώς πολλά διαφορετικά προβλήματα μπορούν να μοντελοποιηθούν με τη βοήθεια των γράφων και της θεωρίας γραφημάτων εν γένει.

3.1.2. Τρόποι αναπαράστασης γράφων

Υπάρχουν διάφοροι τρόποι με τους οποίους μπορεί να αναπαρασταθεί και να αποθηκευτεί ένας γράφος. Οι δύο πιο συνηθισμένοι είναι οι εξής:

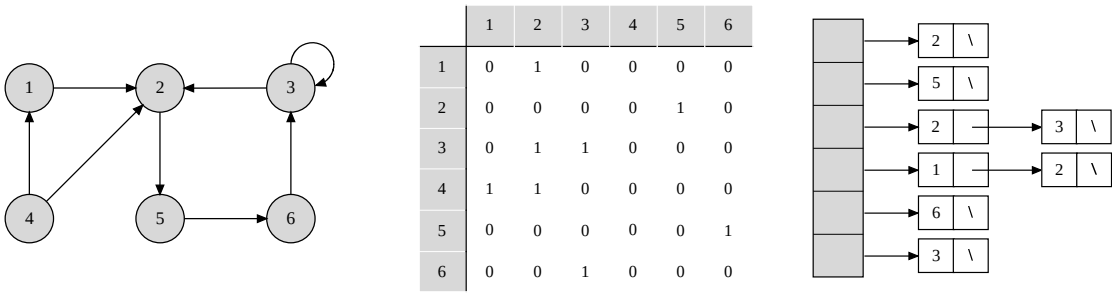
Πίνακας γειτνίασης: Ο γράφος αποθηκεύεται με τη μορφή ενός τετραγωνικού πίνακα μεγέθους $|V| \times |V|$ όπου σε κάθε κορυφή αντιστοιχείται ένας αριθμός (από το 1 μέχρι $|V|$) και κάθε γραμμή και στήλη του πίνακα αντιπροσωπεύει την αντίστοιχη κορυφή, όπως φαίνεται στο *Σχήμα 3.1b*.

Σε κάθε θέση (i, j) του πίνακα αποθηκεύεται:
- 0, εάν δεν υπάρχει ακμή από την κορυφή i προς την κορυφή j

- 1 ή η τιμή του βάρους, εάν υπάρχει ακμή από την κορυφή i προς την κορυφή j

Ο αριθμός των θέσεων μνήμης που απαιτείται είναι $|V|^2$ ανεξαρτήτως από τον αριθμό των ακμών. Πλεονέκτημα αυτού του τρόπου αναπαράστασης είναι ότι υπάρχει άμεσος έλεγχος για την ύπαρξη ακμής. Επίσης, γίνεται προφανές ότι ο τρόπος αυτός από άποψη χώρου στη μνήμη συνίσταται για πυκνούς γράφους.

Λίστα γειτνίασης: Σε κάθε κορυφή του γράφου αντιστοιχείται μία συνδεδεμένη λίστα με όλες τις κορυφές με τις οποίες συνδέεται, όπως φαίνεται στο *Σχήμα 3.1c*. Σε περίπτωση γράφου με βάρη, σε κάθε στοιχείο της λίστας αποθηκεύεται και η τιμή του βάρους. Ο αριθμός των θέσεων μνήμης που απαιτείται είναι $|E|$. Ο τρόπος αυτός προτιμάται σε περιπτώσεις αραιών γράφων.



	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	1	1	0	0	0
4	1	1	0	0	0	0
5	0	0	0	0	0	1
6	0	0	1	0	0	0

Σχ. 3.1: Τρόποι αναπαράστασης γράφων

Γενικά, η επιλογή του τρόπου αναπαράστασης εξαρτάται από τις ιδιαιτερότητες κάθε αλγορίθμου, την πυκνότητα του γράφου και φυσικά το διαθέσιμο χώρο στη μνήμη.

3.1.3. Το πρόβλημα

Δίνεται ένα εμβαρές κατευθυνόμενο γράφημα $G(V, E)$, στο οποίο το βάρος κάθε ακμής αναπαριστά το κόστος μετάβασης από την κορυφή αφετηρίας στην κορυφή προορισμού. Ζητούμενο του προβλήματος είναι να βρεθεί το ελάχιστο κόστος μετάβασης από κάθε κορυφή σε κάθε άλλη κορυφή του γραφήματος. Στη διεθνή βιβλιογραφία συναντάται ως *All Pairs Shortest Path (APSP)*.

Ο γνωστότερος αλγόριθμος που επιλύει το πρόβλημα αυτό είναι ο αλγόριθμος Floyd-Warshall. Υπολογίζει και συγκρίνει όλα τα πιθανά μονοπάτια ανάμεσα σε κάθε ζευγάρι κορυφών. Το επιτυγχάνει βελτιώνοντας σταδιακά την εκτίμηση του συντομότερου μονοπατιού ανάμεσα σε δύο κόμβους, μέχρι η εκτίμηση να γίνει βέλτιστη. Τελικά, ελέγχεται κάθε συνδυασμός ακμών.

Διατυπώθηκε στη σημερινή του μορφή από τον Robert Floyd το 1962. Όμως, είναι πρακτικά ο ίδιος αλγόριθμος με εκείνους που δημοσίευσαν οι Bernard Roy το 1959 και Stephen Warshall το 1962 για την επίλυση του προβλήματος του μεταβατικού κλεισίματος μίας σχέσης (transitive closure). Το συγκεκριμένο πρόβλημα είναι ισοδύναμο με το πρόβλημα APSP. Κατά τη δημοσίευση των αλγορίθμων τους, τόσο ο Floyd όσο και ο Warshall δεν έκαναν καμία αναφορά

περί δυναμικού προγραμματισμού. Παρά ταύτα, και οι δύο αλγόριθμοι είχαν «γεύση» δυναμικού προγραμματισμού και πρέπει να θεωρούνται εφαρμογή της τεχνικής αυτής [1].

3.1.4. Ο αλγόριθμος

Έστω γράφος G με V κόμβους αριθμημένους από 1 έως n. Επίσης, έστω ένα υποσύνολο κόμβων $\{1, 2, \ldots, k\}$ για κάποιο k. Για κάθε ζευγάρι κόμβων $i, j \epsilon V$, ελέγχεται το κόστος όλων των πιθανών μονοπατιών από τον i προς τον j με ενδιάμεσους κόμβους στο υποσύνολο $\{1, 2, \ldots, k\}$ και έστω p το μονοπάτι με το ελάχιστο κόστος. Ο αλγόριθμος Floyd-Warshall εκμεταλλεύεται τη σχέση ανάμεσα στο μονοπάτι p και το ελάχιστο μονοπάτι από τον i προς τον j με ενδιάμεσους κόμβους στο υποσύνολο $\{1, 2, \ldots, k-1\}$:

- Αν ο κόμβος k δεν ανήκει στο μονοπάτι p, τότε όλοι οι ενδιάμεσοι κόμβοι του μονοπατιού p ανήκουν στο υποσύνολο $\{1, 2, \ldots, k-1\}$. Συνεπώς, ένα ελάχιστο μονοπάτι από τον i προς τον j με ενδιάμεσους κόμβους στο υποσύνολο $\{1, 2, \ldots, k-1\}$ είναι ταυτόχρονα και ελάχιστο μονοπάτι από τον i προς τον j με ενδιάμεσους κόμβους στο υποσύνολο $\{1, 2, \ldots, k\}$.
- Αν ο κόμβος k ανήκει στο μονοπάτι p, τότε έστω p_1 το μονοπάτι από τον i προς τον k και p_2 το μονοπάτι από τον k προς τον j. Άρα, το μονοπάτι $i \overset{p}{\rightsquigarrow} j$ αναλύεται σε $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$. Επειδή ο κόμβος k δεν είναι ενδιάμεσος κόμβος του μονοπατιού p_1, όλοι οι ενδιάμεσοι κόμβοι του p_1 ανήκουν στο υποσύνολο $\{1, 2, \ldots, k-1\}$. Συνεπώς, το p_1 είναι ένα ελάχιστο μονοπάτι από τον i προς τον k με όλους τους ενδιάμεσους κόμβους να ανήκουν στο υποσύνολο $\{1, 2, \ldots, k-1\}$. Αντίστοιχα, το p_2 είναι ένα ελάχιστο μονοπάτι από τον k προς τον j με όλους τους ενδιάμεσους κόμβους να ανήκουν στο υποσύνολο $\{1, 2, \ldots, k-1\}$.

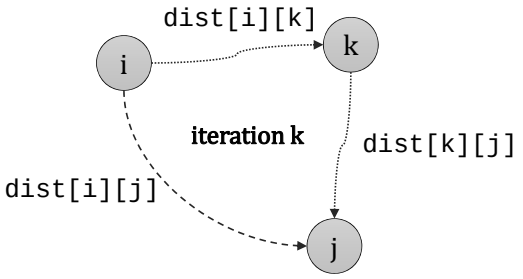
Με βάση τις παραπάνω παρατηρήσεις, ορίζεται μια αναδρομική σχέση για τον υπολογισμό των ελάχιστον μονοπατιών. Έστω $d_{ij}^{(k)}$ το κόστος ενός ελάχιστου μονοπατιού από τον κόμβο i προς τον κόμβο j με ενδιάμεσους κόμβους στο υποσύνολο $\{1, 2, \ldots, k\}$. Αν $k = 0$, τότε πρόκειται για ένα μονοπάτι από τον i προς τον j με κανέναν ενδιάμεσο κόμβο. Δηλαδή, μονοπάτι με ακριβώς μια ακμή, η οποία προφανώς είναι η (i, j). Άρα, $d_{ij}^{(0)} = w_{ij}$ (όπου w_{ij} το βάρος της ακμής (i, j)). Το $d_{ij}^{(k)}$ ορίζεται αναδρομικά από την εξής σχέση:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{εάν } k = 0 \\ min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{εάν } k \geq 1 \end{cases}$$

Τελικά, το κόστος του ελάχιστου μονοπατιού από τον κόμβο i προς τον κόμβο j (για κάθε i, j) είναι ίσο με $d_{ij}^{(n)}$. Στον *Κώδικα 3.1* φαίνεται ο αλγόριθμος Floyd-Warshall.

Κώδικας 3.1 Αλγόριθμος Floyd-Warshall

```
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if (dist[i][k] + dist[k][j] < dist[i][j]) then
                dist[i][j] = dist[i][k] + dist[k][j]
```



3.2. LU decomposition (Παραγοντοποίηση LU)

Ο δεύτερος αλγόριθμος με τον οποίο ασχολείται η παρούσα διπλωματική εργασία είναι ο αλγόριθμος της παραγοντοποίησης LU (LU decomposition ή LU factorization). Πρόκειται για τη διάσπαση ενός πίνακα σε έναν άνω τριγωνικό (upper) και έναν κάτω τριγωνικό (lower), το γινόμενο των οποίων ισούται με τον αρχικό πίνακα:

$$\underbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}}_{U}$$

Η παραγοντοποίηση LU είναι χρήσιμη για την επίλυση γραμμικών συστημάτων της μορφής $A\mathbf{x} = \mathbf{b}$:

$$A\mathbf{x} = \mathbf{b} \iff L(U\mathbf{x}) = \mathbf{b} \iff \begin{cases} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{cases}$$

3.2.1. Ο αλγόριθμος

Ο αλγόριθμος για την παραγοντοποίηση LU ενός τετραγωνικού πίνακα προκύπτει από τις μαθηματικές σχέσεις που ακολουθούν.

Αρχικά, ισχύει ότι:

$$A = LU \iff a_{ij} = \sum_{r=0}^{n-1} l_{ir} u_{rj} \quad (\forall\, i, j)$$

Στη συνέχεια, θεωρώντας ότι $i \leq j$, προκύπτει:

$$a_{ij} = \sum_{r=0}^{n-1} l_{ir}u_{rj} = \sum_{r=0}^{i} l_{ir}u_{rj} \iff \quad (l_{ir} = 0 \text{ για } r > i)$$

$$a_{ij} = \sum_{r=0}^{i-1} l_{ir}u_{rj} + l_{ii}u_{ij} = \sum_{r=0}^{i-1} l_{ir}u_{rj} + u_{ij} \iff$$

$$u_{ij} = a_{ij} - \sum_{r=0}^{i-1} l_{ir}u_{rj} \quad \text{για } i \leq j$$

Ομοίως:

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{r=0}^{j-1} l_{ir}u_{rj}\right) \quad \text{για } i > j$$

Ορίζεται ο ενδιάμεσος πίνακας $A^{(k)}$ του σταδίου k $(0 \leq k \leq n)$ ως εξής:

$$a_{ij}^{(k)} = a_{ij} - \sum_{r=0}^{k-1} l_{ir}u_{rj}$$

Χρησιμοποιώντας την παραπάνω σημειογραφία είναι προφανές ότι $A^{(0)} = A$ και $A^{(n)} = 0$. Επιπλέον:

$$u_{ij} = a_{ij} - \sum_{r=0}^{i-1} l_{ir}u_{rj} \iff u_{ij} = a_{ij}^{(i)}, \quad (i \leq j)$$

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{r=0}^{j-1} l_{ir}u_{rj}\right) \iff l_{ij} = \frac{a_{ij}^{(j)}}{u_{jj}}, \quad (i > j)$$

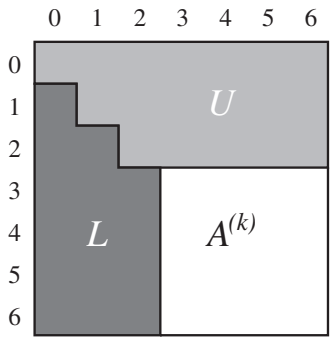
Από τις παραπάνω σχέσεις προκύπτει ο αλγόριθμος του *Κώδικα 3.2*, στον οποίο η αποθήκευση των πινάκων L και U γίνεται in place στον πίνακα A. Η πολυπλοκότητα του αλγορίθμου είναι τάξης $O(n^3)$.

Κώδικας 3.2 Αλγόριθμος LU decomposition

```
for k from 0 to N-1
    for i from k+1 to N-1
        A[i][k] = A[i][k] / A[k][k]
    for i from k+1 to N-1
        for j from k+1 to N-1
            A[i][j] = A[i][j] - A[i][k] * A[k][j]
```

$$\underbrace{\begin{bmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{bmatrix}}_{A} \rightsquigarrow \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & u_{nn} \end{bmatrix}$$

Στο *Σχήμα 3.2* φαίνεται η κατάσταση του πίνακα A κατά την έναρξη του ενδιάμεσου σταδίου $k = 3$. Οι γραμμές 0, 1 και 2 του πίνακα U έχουν ήδη υπολογιστεί, όπως επίσης και οι γραμμές 0, 1 και 2 του πίνακα L.



Σχ. 3.2: Ενδιάμεσο στάδιο εκτέλεσης αλγορίθμου LU Decomposition

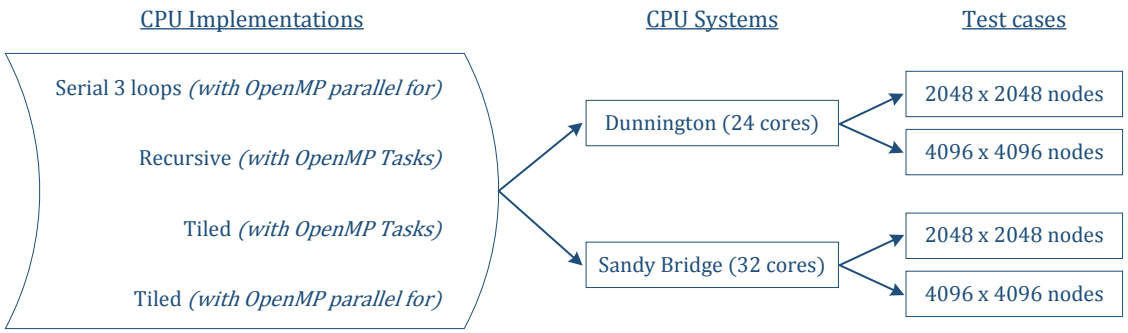
Κεφάλαιο 4

Υλοποίηση αλγορίθμου Floyd-Warshall

4.1. Υλοποίηση σε συστήματα πολλαπλών επεξεργαστών

Ακολουθούν τρεις διαφορετικές υλοποιήσεις για συστήματα πολλαπλών επεξεργαστών που έχουν παραλληλοποιηθεί με χρήση του OpenMP. Μεταγλωττίζονται με χρήση του Intel C Compiler 14.0 ο οποίος ενσωματώνει την έκδοση 4.0 του OpenMP και με επίπεδο βελτιστοποιήσεων -O3. Χρησιμοποιούνται ειδικές εντολές οδηγιών προς τον μεταγλωττιστή της Intel για διανυσματοποίηση (vectorization) όπου είναι εφικτό.

Διεξάγονται μετρήσεις στα δύο συστήματα που αναφέρθηκαν στην *Υποενότητα 2.4.1* και χρησιμοποιούνται γράφοι μικρού και μεγάλου μεγέθους με αριθμό κόμβων 2048×2048 και 4096×4096 αντίστοιχα, όπως φαίνεται στο *Σχήμα 4.1*. Οι γράφοι αποθηκεύονται στη μνήμη με τη μορφή πίνακα γειτνίασης και αναπαρίστανται με τύπο `float`, δηλαδή με αριθμούς κινητής υποδιαστολής απλής ακρίβειας (32 bit). Τα εκτελέσιμα που προορίζονται για το σύστημα με επεξεργαστές Dunnington μεταγλωττίζονται με υποστήριξη των SIMD εντολών SSE 4.1 ενώ εκείνα που προορίζονται για το σύστημα με επεξεργαστές Sandy Bridge με υποστήριξη των SIMD εντολών AVX.



Σχ. 4.1: Παράλληλες υλοποιήσεις αλγορίθμου Floyd-Warshall για συστήματα επεξεργαστών

Μετρήσεις

Dunnington: Αρχικά, διεξάγονται μετρήσεις στο σύστημα Dunnington για αυξανόμενο αριθμό OpenMP threads ώστε να υπολογιστούν οι επιδόσεις και η κλιμάκωση κάθε υλοποίησης. Κάθε thread αντιστοιχεί σε έναν επεξεργαστικό πυρήνα. Επίσης, για δεδομένο αριθμό threads, λαμβάνονται διαφορετικές κατανομές των threads στους επεξεργαστικούς πυρήνες των τεσσάρων επεξεργαστών (packages) του συστήματος, χρησιμοποιώντας τη δυνατότητα του OpenMP για ρητό καθορισμό του CPU Affinity. Αναλυτικότερα:

- Packed: Οι ενεργοί πυρήνες κατανέμονται σε όσο λιγότερους επεξεργαστές γίνεται. Για παράδειγμα, αν υπάρχουν 4 threads τότε συγκεντρώνονται όλα σε έναν επεξεργαστή.
- Spread: Οι ενεργοί πυρήνες κατανέμονται σε όσο περισσότερους επεξεργαστές γίνεται. Έτσι, στο παράδειγμα των 4 threads, ανατίθεται ένα thread ανά επεξεργαστή (package).

Επίσης, όπως έχει ήδη αναφερθεί, οι επεξεργαστικοί πυρήνες των επεξεργαστών αρχιτεκτονικής Dunnington μοιράζονται ανά δύο την ίδια κρυφή μνήμη L2. Συνεπώς, ορίζονται επιπλέον κατανομές «same_L2» και «different_L2» ανάλογα με το αν οι ενεργοί πυρήνες επιλέγονται έτσι ώστε να μοιράζονται τις κρυφές μνήμες L2 ή όχι.

Στο *Σχήμα 4.2* φαίνεται το affinity κάθε πυρήνα όλων των επεξεργαστών του συστήματος και στον *Πίνακα 4.1* όλες οι διαφορετικές κατανομές που λαμβάνονται υπ' όψιν κατά τις μετρήσεις.

Σχ. 4.2: Κατανομή των affinities όλων των πυρήνων του συστήματος Dunnington

		Selected CPUs
2 S	2 thr, spread	0, 3
2 P s_L2	2 thr, packed, same_L2	0, 12
2 P d_L2	2 thr, packed, different_L2	0, 1
4 S	4 thr, spread	0, 3, 6, 9
4 P	4 thr, packed	0, 12, 1, 13
8 S s_L2	8 thr, spread, same_L2	0, 12, 3, 15, 6, 18, 9, 21
8 S d_L2	8 thr, spread, different_L2	0, 1, 3, 4, 6, 7, 9, 10
8 P s_L2	8 thr, packed, same_L2	0, 12, 1, 13, 2, 14, 3, 15
8 P d_L2	8 thr, packed, different_L2	0, 12, 1, 13, 2, 14, 3, 4
12 S s_L2	12 thr, spread, same_L2	0, 12, 3, 15, 6, 18, 9, 21, 1, 4, 7, 10
12 S d_L2	12 thr, spread, different_L2	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
12 P	12 thr, packed	0, 1, 2, 12, 13, 14, 3, 4, 5, 15, 16, 17
16 S	16 thr, spread	0, 1, 12, 13, 3, 4, 15, 16, 6, 7, 18, 19, 9, 10, 21, 22
16 P	16 thr, packed	0, 1, 2, 12, 13, 14, 3, 4, 5, 15, 16, 17, 6, 7, 18, 19
24	24 thr	0 - 23

Πίν. 4.1: Κατανομές νημάτων σε επεξεργαστικούς πυρήνες του συστήματος Dunnington

Sandy Bridge: Στο σύστημα με επεξεργαστές αρχιτεκτονικής Sandy Bridge λαμβάνονται κατανομές spread και packed όμοια με προηγουμένως. Όμως, δεν εξετάζονται κατανομές σε σχέση με την κρυφή μνήμη $L2$ αφού κάθε πυρήνας έχει προσωπική και δεν είναι μοιραζόμενη.

Στο *Σχήμα 4.3* φαίνεται το affinity κάθε πυρήνα όλων των επεξεργαστών του συστήματος. Υπενθυμίζεται ότι, λόγω της υποστήριξης της τεχνολογίας Hyper-Threading, κάθε φυσικός πυρήνας αναγνωρίζεται από το λειτουργικό σύστημα ως δύο λογικοί. Στον *Πίνακα 4.2* φαίνονται όλες οι διαφορετικές κατανομές που εξετάζονται κατά τις μετρήσεις.

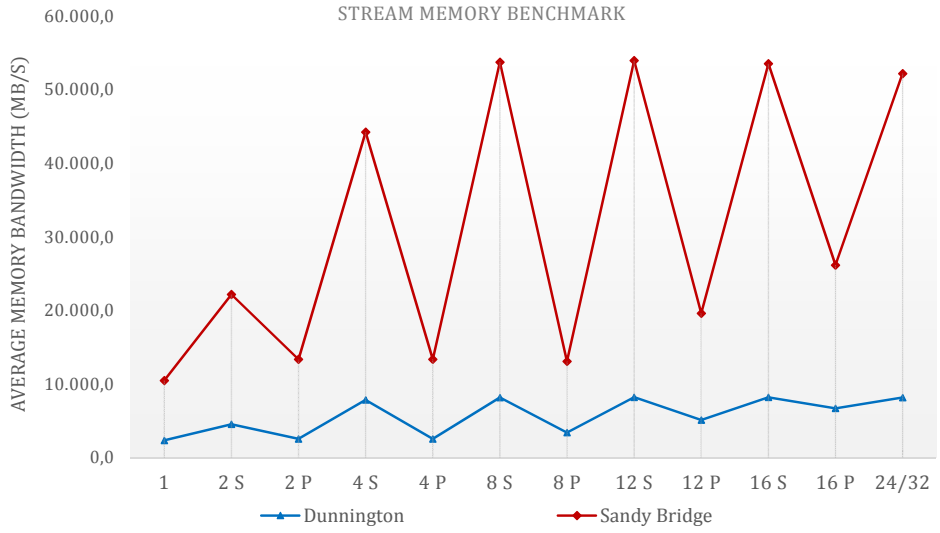
Σχ. 4.3: Κατανομή των affinities όλων των πυρήνων του συστήματος Sandy Bridge

		Selected CPUs
2 S	2 thr, spread	0, 8
2 P	2 thr, packed	0, 1
4 S	4 thr, spread	0, 8, 16, 24
4 P	4 thr, packed	0, 1, 2, 3
8 S	8 thr, spread	0, 1, 8, 9, 16, 17, 24, 25
8 P	8 thr, packed	0, 1, 2, 3, 4, 5, 6, 7
12 S	12 thr, spread	0, 1, 2, 8, 9, 10, 16, 17, 18, 24, 25, 26
12 P	12 thr, packed	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
16 S	16 thr, spread	0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, 19, 24, 25, 26, 27
16 P	16 thr, packed	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
32	32 thr	0-31
64	64 thr	0-63

Πίν. 4.2: Κατανομές νημάτων σε επεξεργαστικούς πυρήνες του συστήματος Sandy Bridge

Memory Bandwidth

Για τη μέτρηση του διαθέσιμου memory bandwidth κάθε συστήματος χρησιμοποιείται το γνωστό μετροπρόγραμμα Stream [25]. Στο *Σχήμα 4.4* φαίνονται τα αποτελέσματα για κάθε υπολογιστικό σύστημα και για όλες τις κατανομές spread και packed που περιγράφηκαν προηγουμένως.



Σχ. 4.4: Χωρισμός πίνακα σε υποπίνακες και διαδοχικές αναδρομικές κλήσης σε κάθε επίπεδο της αναδρομής

Γίνεται αμέσως αντιληπτή η μεγάλη διαφορά ανάμεσα στα δύο συστήματα όσον αφορά το μέγιστο εύρος ζώνης προς την κύρια μνήμη. Αιτία για αυτό αποτελεί η UMA αρχιτεκτονική του συστήματος Dunnington και η χρήση του πιο αργού διαύλου FSB για την επικοινωνία των επεξεργαστών με τον Memory Controller Hub, όπως φαίνεται στο *Σχήμα 2.25* του *Κεφαλαίου 2.*

Επίσης, παρατηρείται το μεγάλο προβάδισμα των spread κατανομών σε σχέση με τις packed (ειδικά για το σύστημα Sandy Bridge), κάτι το οποίο είναι αναμενόμενο λόγω των περισσότερων επεξεργαστών (packages) που χρησιμοποιούν οι spread κατανομές.

4.1.1. Σειριακή υλοποίηση

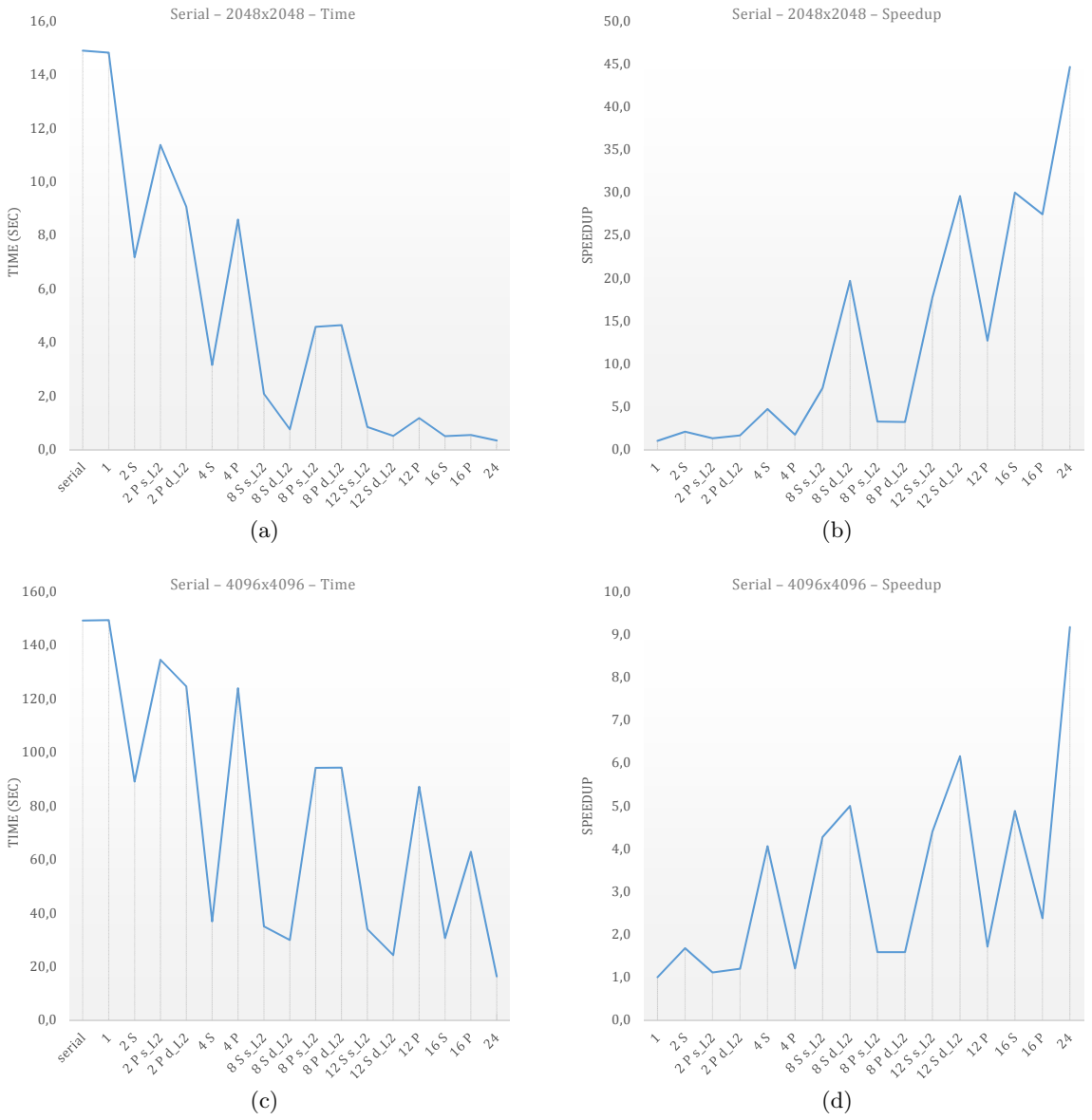
Αρχικά, παραλληλοποιείται η απλή σειριακή υλοποίηση του αλγορίθμου, όπως φαίνεται στον *Κώδικα 4.1*. Με κατάλληλη οδηγία προς τον μεταγλωττιστή γίνεται vectorization του εσωτερικού loop (j-loop) του κώδικα.

Κώδικας 4.1 Σειριακή υλοποίηση αλγορίθμου Floyd-Warshall

```
#pragma omp parallel firstprivate(k)
for(k = 0; k < N; k++)
    #pragma omp for private(j)
    for(i = 0; i < N; i++)
        for(j = 0; j < N; j++)
            A[i][j] = min(A[i][j], A[i][k] + A[k][j]);
```

Αποτελέσματα Dunnington

Στο *Σχήμα 4.5* φαίνονται τα διαγράμματα χρόνου και speedup των μετρήσεων στο σύστημα Dunnington για γράφους διαστάσεων 2048×2048 και 4096×4096 κόμβων. Ο μικρός γράφος καταλαμβάνει χώρο στη μνήμη ίσο με $2048 \times 2048 \times 4B = 16\,MB$ ενώ αντίθετα ο μεγάλος γράφος καταλαμβάνει $4096 \times 4096 \times 4B = 64\,MB$.

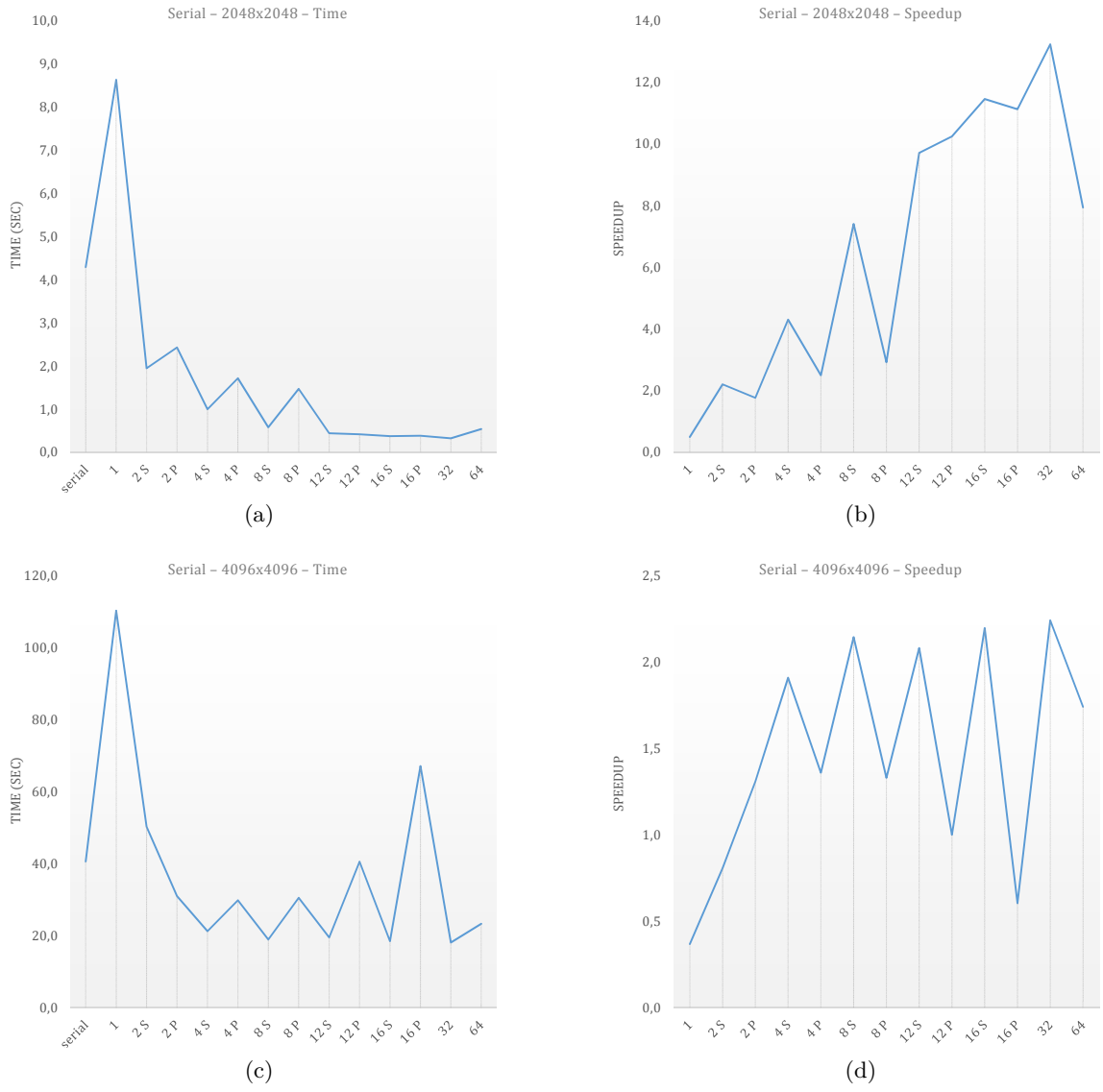


Σχ. 4.5: Αποτελέσματα μετρήσεων σειριακής υλοποίησης στο σύστημα Dunnington

Για το μικρό γράφο παρατηρείται speedup μεγαλύτερο από τον αριθμό των επεξεργαστών, δηλαδή υπεργραμμικό (superlinear). Αυτό οφείλεται στο λεγόμενο cache effect. Οι κρυφές μνήμες των επεξεργαστών Dunnington είναι πολύ μεγάλες σε μέγεθος και κάθε επεξεργαστής φέρει συνολικά $L2 : 3 \times 3MB + L3 : 16MB = 25\,MB$ κρυφής μνήμης επιπέδων $L2$ και $L3$. Συνεπώς, χωράει με ευκολία όλος ο γράφος στις κρυφές μνήμες κάθε επεξεργαστή και οι χρόνοι προσπέλασης των δεδομένων μειώνονται δραστικά. Έτσι, προκύπτει ένα επιπρόσθετο σημαντικό speedup λόγω του φαινομένου αυτού επιπλέον από αυτό που προκύπτει με την αύξηση των επεξεργαστικών πυρήνων. Το παραπάνω γίνεται ακόμα πιο εμφανές στις packed κατανομές όπου οι επιδόσεις είναι σημαντικά χειρότερες από τις αντίστοιχες spread επειδή η συνολικά διαθέσιμη κρυφή μνήμη είναι αρκετά μικρότερη. Επίσης, στις κατανομές different_L2 οι επιδόσεις είναι καλύτερες από τις αντίστοιχες same_L2 λόγω του ίδιου φαινομένου. Τέλος, μια ακόμα αιτία για το προβάδισμα των spread κατανομών είναι το υψηλότερο εύρος ζώνης προς την κύρια μνήμη.

Για να προκύψουν σωστά συμπεράσματα για την κλιμάκωση του αλγορίθμου, εξετάζονται τα αποτελέσματα του μεγάλου γράφου. Πλέον, το speedup είναι σημαντικά μειωμένο και απέχει πολύ από το ιδανικό. Βασικότερη αιτία είναι η φύση του αλγορίθμου που κάνει χωρίς χωρική τοπικότητα προσβάσεις στη μνήμη. Επίσης, όμοια με προηγουμένως, παρατηρούνται πολύ καλύτερες επιδόσεις στις spread και same_L2 κατανομές.

Αποτελέσματα Sandy Bridge



Σχ. 4.6: Αποτελέσματα μετρήσεων σειριακής υλοποίησης στο σύστημα Sandy Bridge

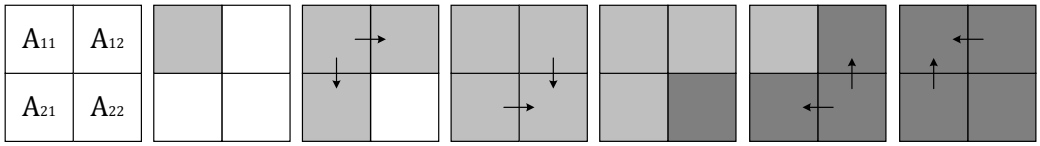
Εδώ πρέπει να σημειωθεί ότι το σύστημα με αρχιτεκτονική Sandy Bridge ακολουθεί αρχιτεκτονική NUMA και συνεπώς η πρόσβαση στη μνήμη δεν απαιτεί τον ίδιο χρόνο για όλους τους επεξεργαστές. Κατά την υλοποίηση του αλγορίθμου, εξετάζεται μόνο η γενική περίπτωση και δεν λαμβάνεται υπ' όψιν η NUMA φιλοσοφία του συστήματος κατά το allocation του πίνακα γειτνίασης στη μνήμη, συνεπώς αποθηκεύεται εξ' ολοκλήρου στην κύρια μνήμη ενός επεξεργαστή. Για το λόγο αυτό, το speedup που παρατηρείται είναι εξαιρετικά κακό για τον μεγάλο

γράφο και κάπως βελτιωμένο για τον μικρό γράφο, απόρροια του cache effect. Για του ίδιους λόγους που αναλύθηκαν προηγουμένως, προκύπτει προβάδισμα των spread έναντι των packed κατανομών.

4.1.2. Αναδρομική υλοποίηση

Η παραλληλοποίηση της απλής υλοποίησης του αλγορίθμου με τα 3 loops δεν προσφέρει σημαντικό speedup. Αυτό οφείλεται στη memory bounded φύση του αλγορίθμου, αφού σε κάθε επανάληψη πρέπει να μεταφέρεται από την κύρια μνήμη ολόκληρος ο πίνακας ενώ οι πράξεις που γίνονται είναι απλές από άποψη επεξεργαστικού φορτίου (μία σύγκριση και μία πρόσθεση). Για τον λόγο αυτό, έχουν προταθεί δύο εναλλακτικές μέθοδοι υλοποίησης για καλύτερη εκμετάλλευση της κρυφής μνήμης.

Η πρώτη είναι η *αναδρομική* (*recursive*) υλοποίηση η οποία εφαρμόζει αναδρομικά αυτόματο blocking σε κάθε επίπεδο, με βάση μία δεδομένη τιμή του μεγέθους του block. Ο αλγόριθμος φαίνεται στον *Κώδικα 4.2* ενώ στο *Σχήμα 4.7* φαίνεται ο τρόπος που χωρίζεται ο πίνακας σε κάθε επίπεδο της αναδρομής. Επιπλέον, φαίνονται οι εξαρτήσεις που υπάρχουν ανάμεσα στη σειρά με την οποία γίνονται οι αναδρομικές κλήσεις για την επεξεργασία κάθε υποπίνακα.



Σχ. 4.7: Χωρισμός πίνακα σε υποπίνακες και διαδοχικές αναδρομικές κλήσης σε κάθε επίπεδο της αναδρομής

Η αρχική αναδρομική κλήση γίνεται ως `FWR(A, A, A)` ενώ σε κάθε επίπεδο της αναδρομής τα ορίσματα `A`, `B`, `C` δείχνουν σε ίδιους ή διαφορετικούς υποπίνακες του αρχικού πίνακα γειτνίασης. Επιλέγοντας κατάλληλο μέγεθος block, ώστε να χωράει στην κρυφή μνήμη του επεξεργαστή, επιτυγχάνονται οι μέγιστες επιδόσεις.

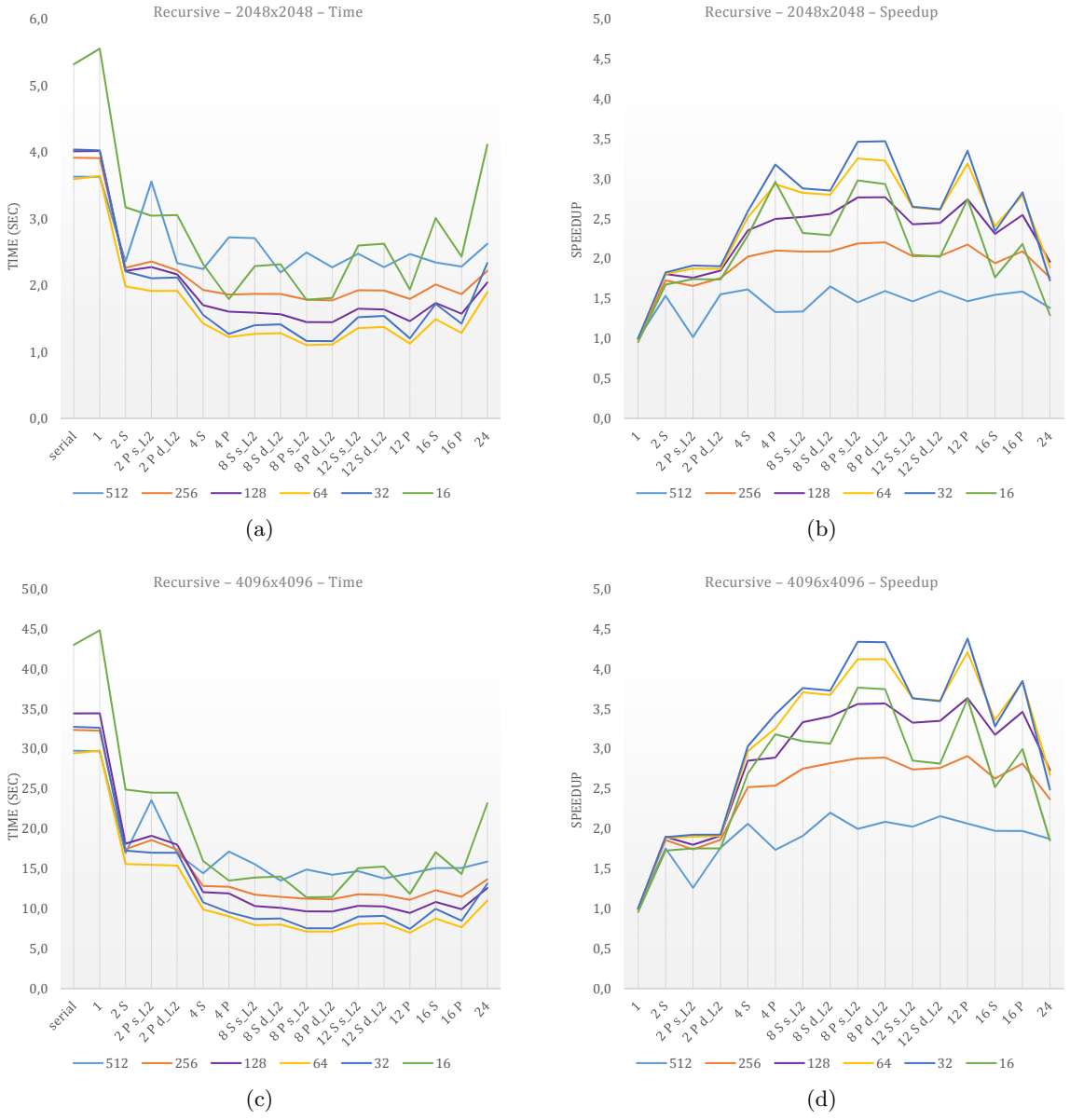
Η παραλληλοποίηση του αλγορίθμου γίνεται με τη χρήση OpenMP tasks, όπως φαίνεται στον *Κώδικα 4.2*. Σύμφωνα με τις εξαρτήσεις δεδομένων, η επεξεργασία των υποπινάκων A_{12} και A_{21} μπορεί να γίνει παράλληλα. Αυτό το γεγονός εκμεταλλευόμαστε κατά την παραλληλοποίηση και η χρήση των tasks αποδεικνύεται χρήσιμη. Επίσης, με κατάλληλη οδηγία προς τον μεταγλωττιστή γίνεται vectorization του εσωτερικού loop (j-loop) του κώδικα της συνάρτηση `FWI`.

Κώδικας 4.2 Αναδρομική (recursive) υλοποίηση αλγορίθμου Floyd-Warshall με OpenMP tasks

```
FWI(A, B, C):

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

FWR(A, B, C):

if (base case)
    FWI(A, B, C)
else
{
    FWR(A11, B11, C11);

    #pragma omp task
    FWR(A12, B11, C12);
    #pragma omp task
    FWR(A21, B21, C11);
    #pragma omp taskwait

    FWR(A22, B21, C12);
    FWR(A22, B21, C12);

    #pragma omp task
    FWR(A21, B21, C11);
    #pragma omp task
    FWR(A12, B11, C12);
    #pragma omp taskwait

    FWR(A11, B11, C11);
}
```

Αποτελέσματα Dunnington

Στο *Σχήμα 4.8* φαίνονται τα αποτελέσματα των μετρήσεων της αναδρομικής υλοποίησης στο σύστημα Dunnington για τις διάφορες κατανομές που περιγράφηκαν και για γράφους μεγέθους 2048×2048 και 4096×4096 κόμβων. Λαμβάνονται μετρήσεις για διάφορες τιμές μεγέθους του block που κυμαίνονται από 512×512 στοιχεία έως 16×16 στοιχεία.



(a) (b) (c) (d)

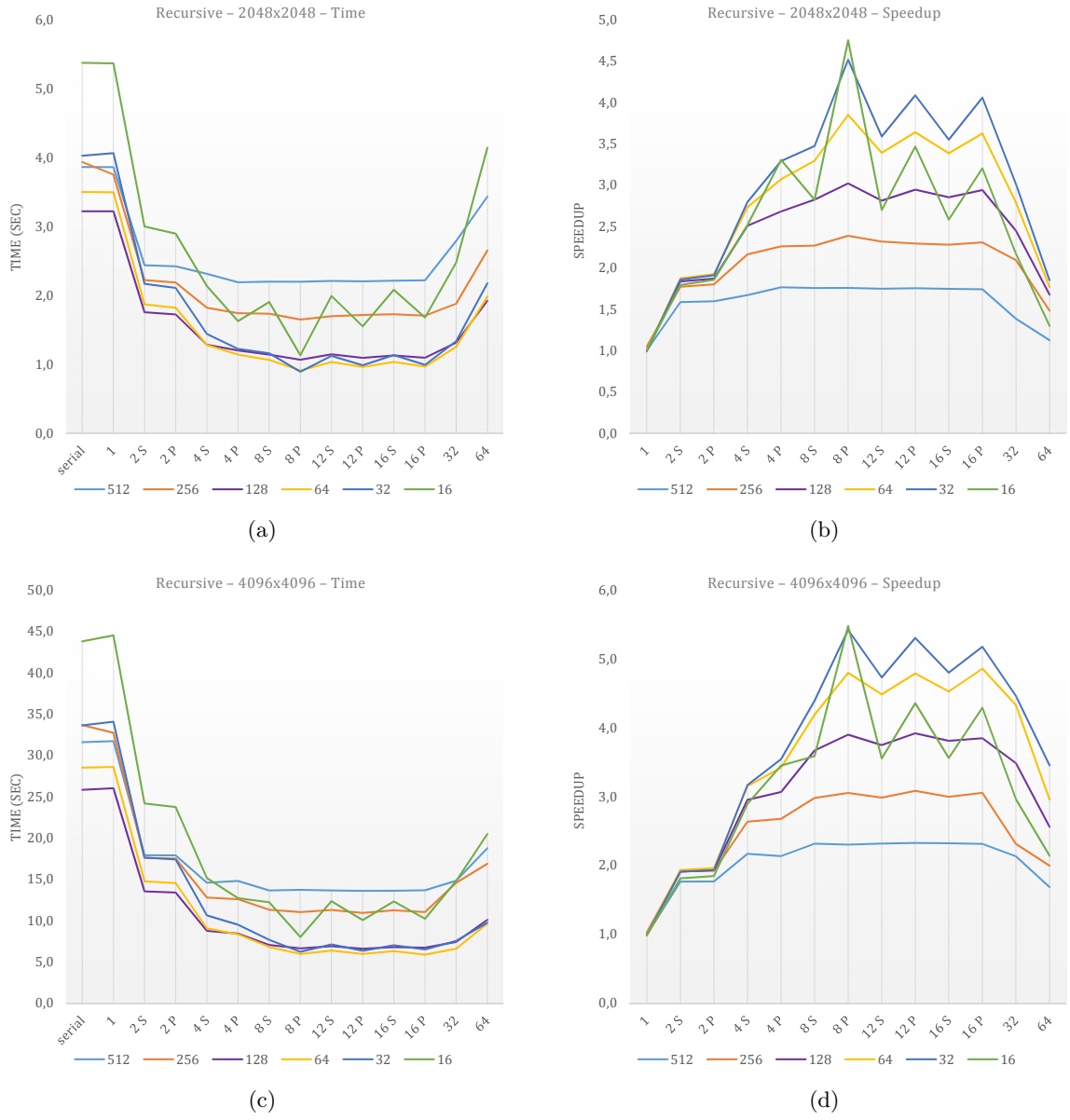
Σχ. 4.8: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στο σύστημα Dunnington

Παρατηρείται αμέσως σημαντική βελτίωση των χρόνων σε σχέση με την απλή υλοποίηση των τριών loops, τόσο στη σειριακή όσο και στις παράλληλες εκτελέσεις. Επίσης, αντίθετα με προηγουμένως, για ίδιο αριθμό threads οι packed κατανομές δίνουν καλύτερα αποτελέσματα σε σχέση με τις αντίστοιχες spread, αφού πλέον σημασία δεν έχει τόσο το μέγεθος της κρυφής μνήμης αλλά πόσα blocks χωράνε σε αυτές ώστε να τα μοιράζονται περισσότεροι επεξεργαστικοί πυρήνες. Τα καλύτερα αποτελέσματα προκύπτουν για μέγεθος block ίσο με $64 \times 64 \times 4B =$

$16\,KB$. Αυτό είναι αναμενόμενο αν ληφθεί υπ' όψιν ότι η κρυφή μνήμη $L1$ έχει μέγεθος $32\,KB$. Όμως, η κλιμάκωση του αλγορίθμου δεν είναι καλή, κάτι το οποίο οφείλεται στο ότι μικρό μόνο κομμάτι του κώδικα μπορεί να παραλληλοποιηθεί.

Αποτελέσματα Sandy Bridge

Αντίστοιχα με προηγουμένως, στο *Σχήμα 4.9* φαίνονται τα αποτελέσματα των μετρήσεων της αναδρομικής υλοποίησης στο σύστημα Sandy Bridge.



(a) (b) (c) (d)

Σχ. 4.9: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στο σύστημα Sandy Bridge

Και σε αυτά τα αποτελέσματα παρατηρείται η ίδια συμπεριφορά που αναλύθηκε προηγουμένως για το σύστημα Dunnington, με τα καλύτερα αποτελέσματα χρόνου να προκύπτουν για το ίδιο μέγεθος block.

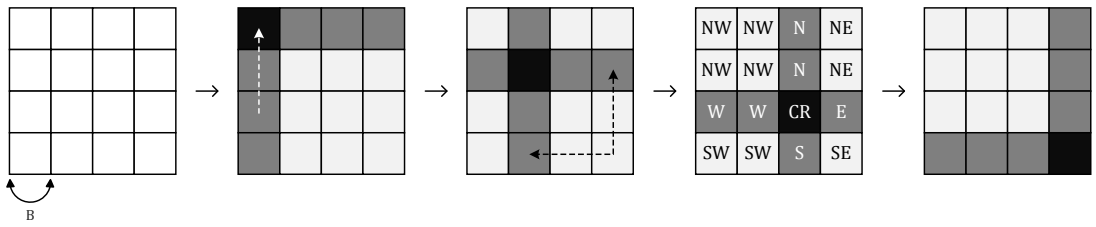
4.1.3. Tiled υλοποίηση

Η δεύτερη εναλλακτική μέθοδος υλοποίησης που έχει προταθεί για καλύτερη εκμετάλλευση της κρυφής μνήμης είναι η λεγόμενη «*Tiled*» υλοποίηση. Η τεχνική του tiling χρησιμοποιείται συχνά για την επίτευξη υψηλής επαναχρησιμοποίησης των δεδομένων σε αλγορίθμους με loops όπως επίσης για τη μείωση της μεταφοράς δεδομένων από και προς την κύρια μνήμη.

Αρχικά, επιλέγεται ένα μέγεθος block ίσο με B και ο πίνακας χωρίζεται σε tiles (πλακίδια) μεγέθους $B \times B$, όπως φαίνεται στο *Σχήμα 4.10*. Στην $k-$οστή επανάληψη:

- Πρώτα ενημερώνεται το διαγώνιο (k, k) tile (μαύρο, CR tile του *Σχήματος 4.10*). Η ενημέρωση γίνεται ανεξάρτητα και δεν υπάρχει εξάρτηση δεδομένων από κάποιο άλλο tile.
- Στη συνέχεια, ενημερώνονται τα tiles που βρίσκονται στην $k-$οστή γραμμή και στην $k-$οστή στήλη (σκούρο γκρι, $\{N, S, E, W\}$) και υπάρχει εξάρτηση δεδομένων από το αρχικό διαγώνιο tile που έχει ήδη ενημερωθεί. Οι ενημερώσεις μπορούν να γίνουν παράλληλα αφού είναι ανεξάρτητες μεταξύ τους και εξαρτώνται μόνο από το διαγώνιο tile.
- Τέλος, όταν έχει ολοκληρωθεί το προηγούμενο στάδιο, ενημερώνονται τα υπόλοιπα tiles του πίνακα (ανοιχτό γκρι, $\{NW, SW, NE, SE\}$). Υπάρχει εξάρτηση δεδομένων για κάθε tile από τα αντίστοιχα $\{N, S, E, W\}$ tiles που βρίσκονται στην ίδια γραμμή και στήλη. Ομοίως με προηγουμένως, οι ενημερώσεις μπορούν να γίνουν παράλληλα μεταξύ τους.

Αν το μέγεθος του πίνακα είναι N, συνολικά απαιτούνται $k = {}^{N}/_{B}$ επαναλήψεις. Τα διακεκομμένα βέλη του *Σχήματος 4.10* δείχνουν τις εξαρτήσεις των δεδομένων για κάθε είδος tile. Με την τεχνική του tiling μειώνεται η κίνηση από και προς την κύρια μνήμη κατά B φορές. Η κατάλληλη επιλογή του μεγέθους του B σε σχέση με το μέγεθος της κρυφής μνήμης δίνει τα καλύτερα αποτελέσματα. Στον *Κώδικα 4.3* φαίνεται η tiled σειριακή υλοποίηση του αλγορίθμου που περιγράφηκε παραπάνω.



Σχ. 4.10: Εφαρμογή τεχνικής tiling στον αλγόριθμο Floyd-Warshall

Κώδικας 4.3 Tiled σειριακή υλοποίηση αλγορίθμου Floyd-Warshall

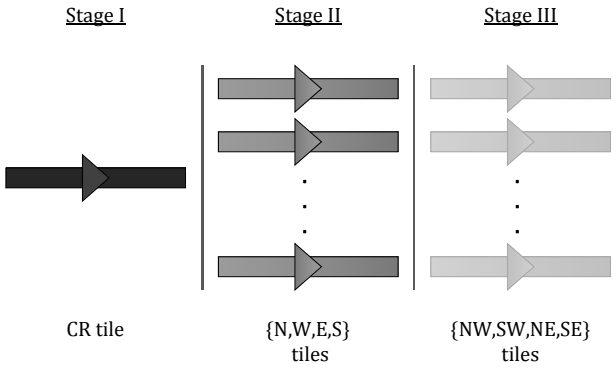
```
for (k = 0; k < N; k+=B) {
    FW(CR);
    for tile in E, W, N, S
        FW(tile);
    for tile in NE, NW, SE, SW
        FW(tile);
}
```

Παραλληλοποίηση με OpenMP Tasks

Ο πρώτος τρόπος με τον οποίο μπορεί να παραλληλοποιηθεί η tiled υλοποίηση είναι με χρήση OpenMP tasks, όπως φαίνεται στον *Κώδικα 4.4*.

Κώδικας 4.4 Tiled υλοποίηση αλγορίθμου Floyd-Warshall παραλληλοποιημένη με OpenMP tasks

```
for (k = 0; k < N; k+=B)
{
    FW(CR);

    foreach tile in (E, W, N, S)
        #pragma omp task
        FW(tile);

    #pragma omp taskwait

    foreach tile in (E)
        #pragma omp task
        foreach tile in (N)
            FW(NE);

    foreach tile in (W)
        #pragma omp task
        foreach tile in (N)
            FW(NW);

    foreach tile in (E)
        #pragma omp task
        foreach tile in (S)
            FW(SE);

    foreach tile in (W)
        #pragma omp task
        foreach tile in (S)
            FW(SW);

    #pragma omp taskwait
}
```



Η δημιουργία και ο συγχρονισμός των tasks γίνεται με τέτοιο τρόπο ώστε να γίνονται σεβαστές οι εξαρτήσεις δεδομένων που υπάρχουν ανάμεσα στα 3 στάδια κάθε επανάληψης. Δοκιμάστηκε επίσης μια εναλλακτική υλοποίηση του *Κώδικα 4.4*, στην οποία κατά το τρίτο στάδιο δημιουργείται ένα ξεχωριστό task για κάθε $\{NW, SW, NE, SE\}$ tile αλλά τα αποτελέσματα ήταν αρκετά χειρότερα εξαιτίας του αυξημένου overhead δημιουργίας και δρομολόγησης τόσων πολλών tasks και του πολύ μικρού επεξεργαστικού φορτίου κάθε tasks. Για το λόγο αυτό, δεν συμπεριλαμβάνονται μετρήσεις από αυτή την υλοποίηση.

Παραλληλοποίηση με OpenMP parallel for

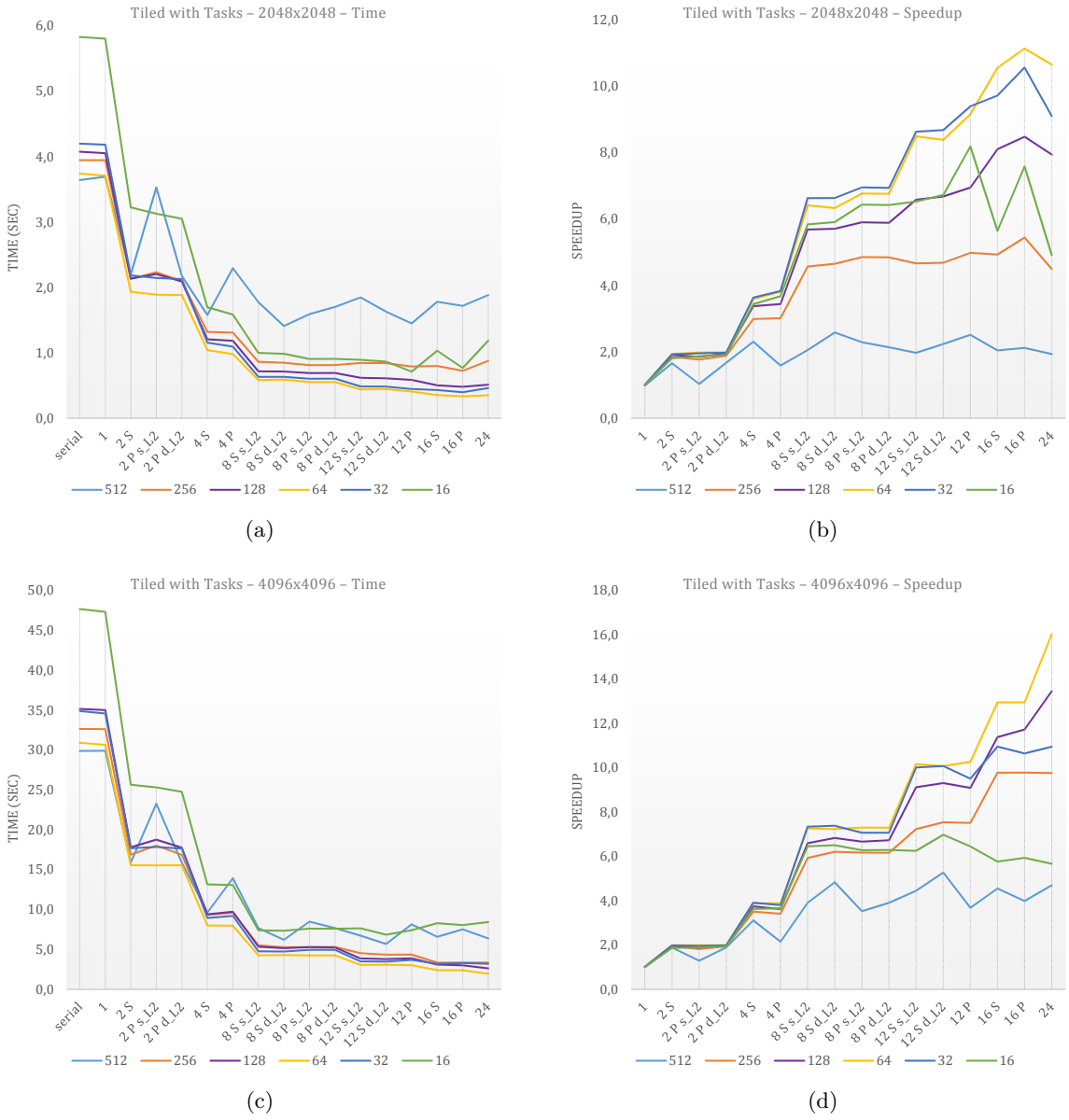
Ο δεύτερος τρόπος με τον οποίο μπορεί να παραλληλοποιηθεί η tiled υλοποίηση είναι παραλληλοποιώντας όλα τα for loops κάθε επανάληψης με χρήση της κατάλληλης οδηγίας του OpenMP, όπως φαίνεται στον *Κώδικα 4.5*.

Κώδικας 4.5 Tiled υλοποίηση αλγορίθμου Floyd-Warshall παραλληλοποιημένη με OpenMP parallel for

```
for (k = 0; k < N; k+=B)
{
    FW(CR);

    #pragma omp for nowait
    foreach tile in (E, W, N, S)
        FW(tile);

    #pragma omp barrier

    #pragma omp for nowait
    foreach tile in (E)
        foreach tile in (N)
            FW(NE);

    #pragma omp for nowait
    foreach tile in (W)
        foreach tile in (N)
            FW(NW);

    #pragma omp for nowait
    foreach tile in (E)
        foreach tile in (S)
            FW(SE);

    #pragma omp for nowait
    foreach tile in (W)
        foreach tile in (S)
            FW(SW);

    #pragma omp barrier
}
```

Αποτελέσματα Dunnington

OpenMP tasks: Στο *Σχήμα 4.11* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης με χρήση OpenMP tasks στο σύστημα Dunnington για τις διάφορες κατανομές που περιγράφηκαν και για γράφους μεγέθους 2048×2048 και 4096×4096 κόμβων. Λαμβάνονται μετρήσεις για διάφορες τιμές B του block που κυμαίνονται από $B = 512$ έως $B = 16$.



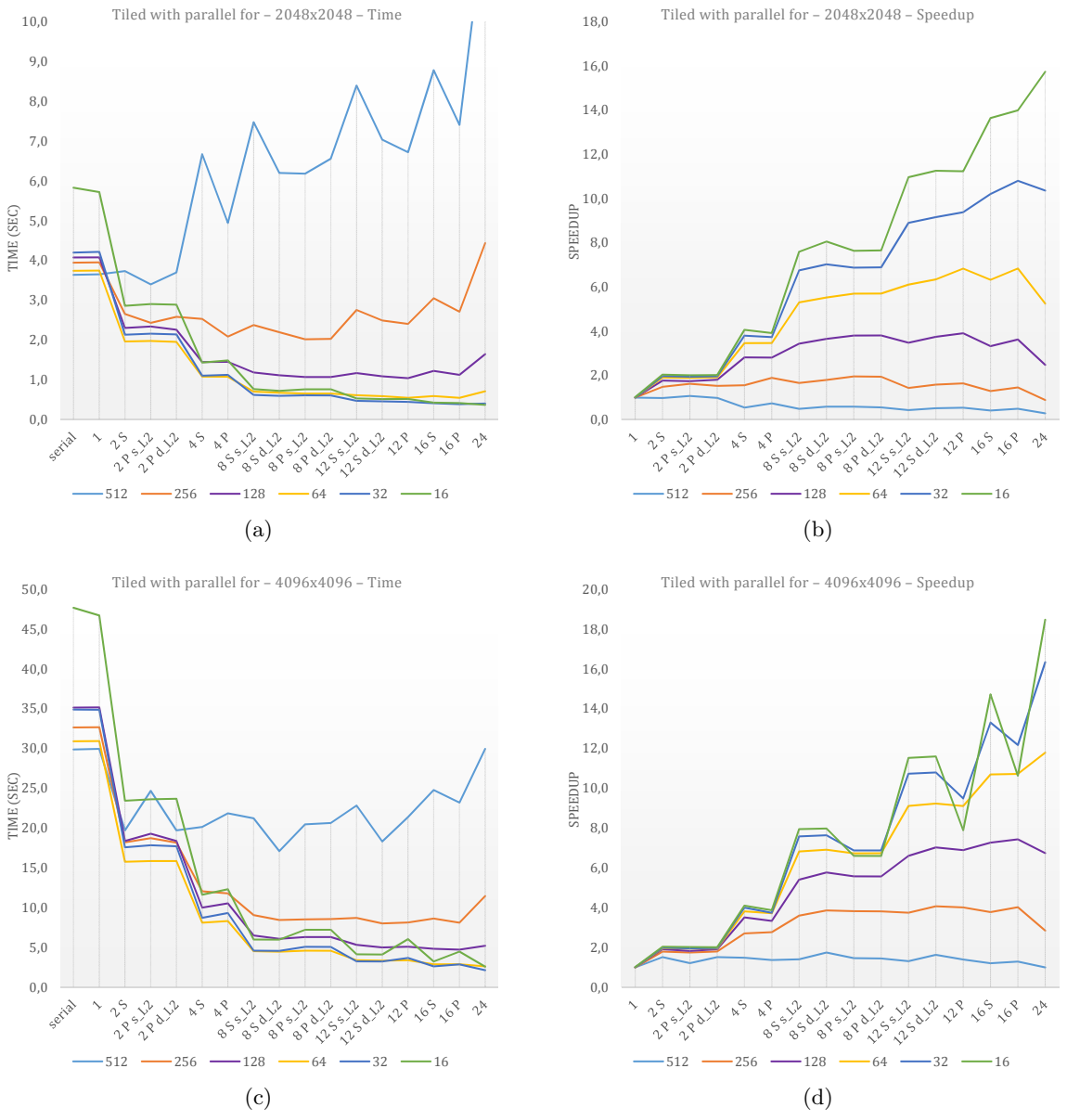
(a) (b) (c) (d)

Σχ. 4.11: Αποτελέσματα μετρήσεων tiled υλοποίησης με χρήση OpenMP tasks στο σύστημα Dunnington

Στην υλοποίηση αυτή βλέπουμε για πρώτη φορά πολύ βελτιωμένες επιδόσεις σε σχέση με τις προηγούμενες υλοποιήσεις. Η καλύτερη επίδοση προκύπτει για $B = 64$, δηλαδή μέγεθος block ίσο με $64 \times 64 \times 4B = 16\,KB$. Το γεγονός αυτό είναι αναμενόμενο με βάση την κρυφή μνήμη $L1$ μεγέθους $32\,KB$. Το αμέσως μεγαλύτερο μέγεθος block $B = 128$ ($128 \times 128 \times 4B = 64\,KB$), μπορεί να μειώνει στο μισό τα δεδομένα που μεταφέρονται από τη μνήμη σε σχέση με το $B = 64$,

αλλά ταυτόχρονα υστερεί αφού δεν χωράει ολόκληρο στην $L1$ κρυφή μνήμη. Για τα μη ακραία μεγέθη του B, δεν προκύπτει καμία διαφορά ανάμεσα σε packed και spread κατανομές ίδιου αριθμού threads. Τέλος, η κλιμάκωση της υλοποίησης κρίνεται αρκετά καλή και σημαντικά βελτιωμένη σε σχέση με τις προαναφερθείσες υλοποιήσεις.

OpenMP parallel for: Στο *Σχήμα 4.12* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης με χρήση OpenMP parallel for στο σύστημα Dunnington.



(a) (b) (c) (d)

Σχ. 4.12: Αποτελέσματα μετρήσεων tiled υλοποίησης με χρήση OpenMP parallel for στο σύστημα Dunnington

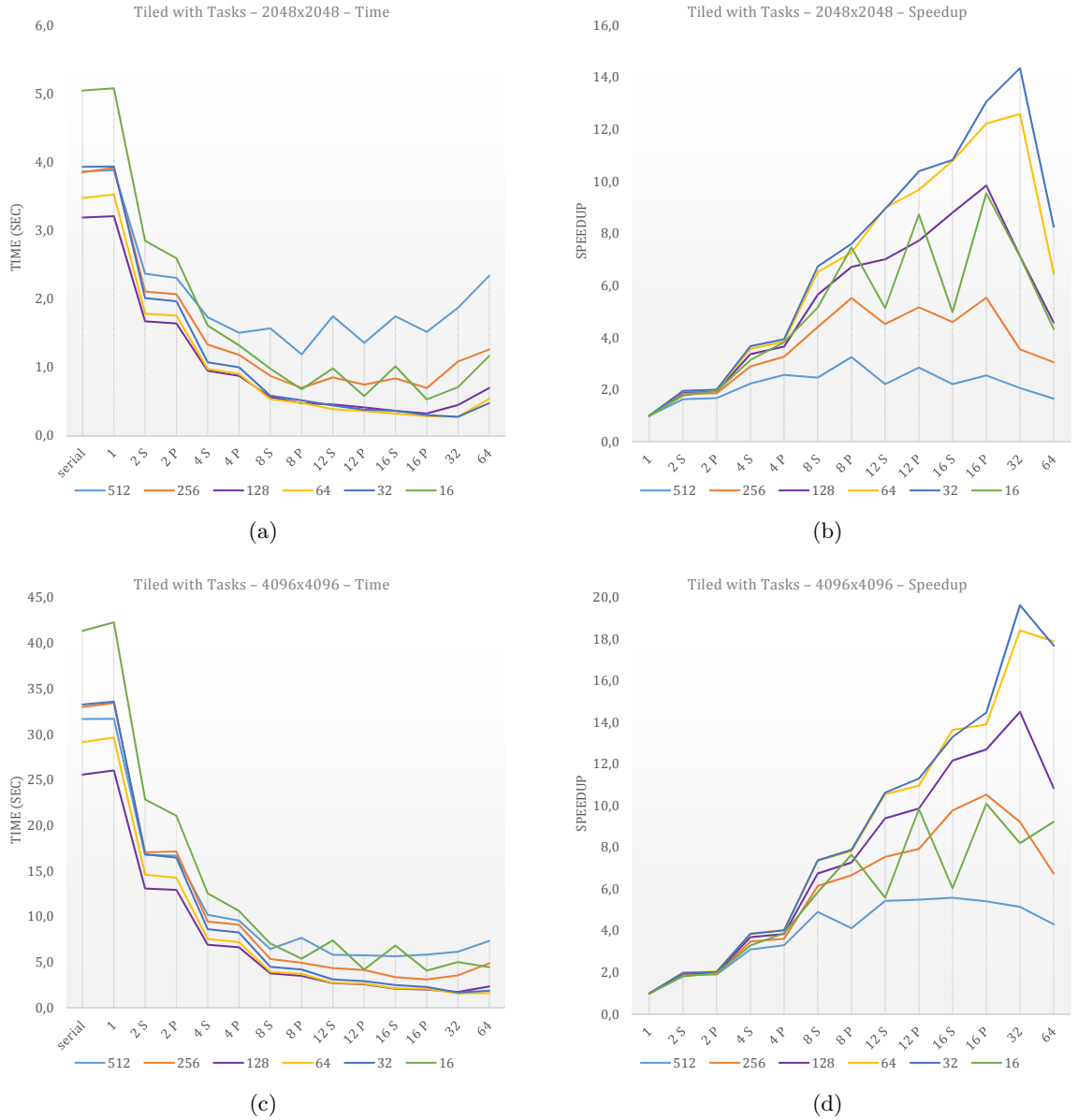
Και σε αυτή την υλοποίηση παρατηρείται όμοια συμπεριφορά με εκείνη της υλοποίησης με tasks, με τους καλύτερους χρόνους όμως ελαφρώς αυξημένους, όπως φαίνεται στον *Πίνακα 4.3*.

Tiled (24 threads)	2048×2048			4096×4096		
	$B = 64$	$B = 32$	$B = 16$	$B = 64$	$B = 32$	$B = 16$
OpenMP tasks	*0,351s*	$0,461s$	$1,187s$	*1,926s*	$3,189s$	$8,419s$
OpenMP parallel for	$0,713s$	$0,405s$	$0,370s$	$2,622s$	$2,136s$	$2,581s$

Πίν. 4.3: Καλύτεροι χρόνοι παράλληλων tiled υλοποιήσεων στο σύστημα Dunnington

Αποτελέσματα Sandy Bridge

OpenMP tasks: Στο *Σχήμα 4.13* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης με χρήση OpenMP tasks στο σύστημα Sandy Bridge.



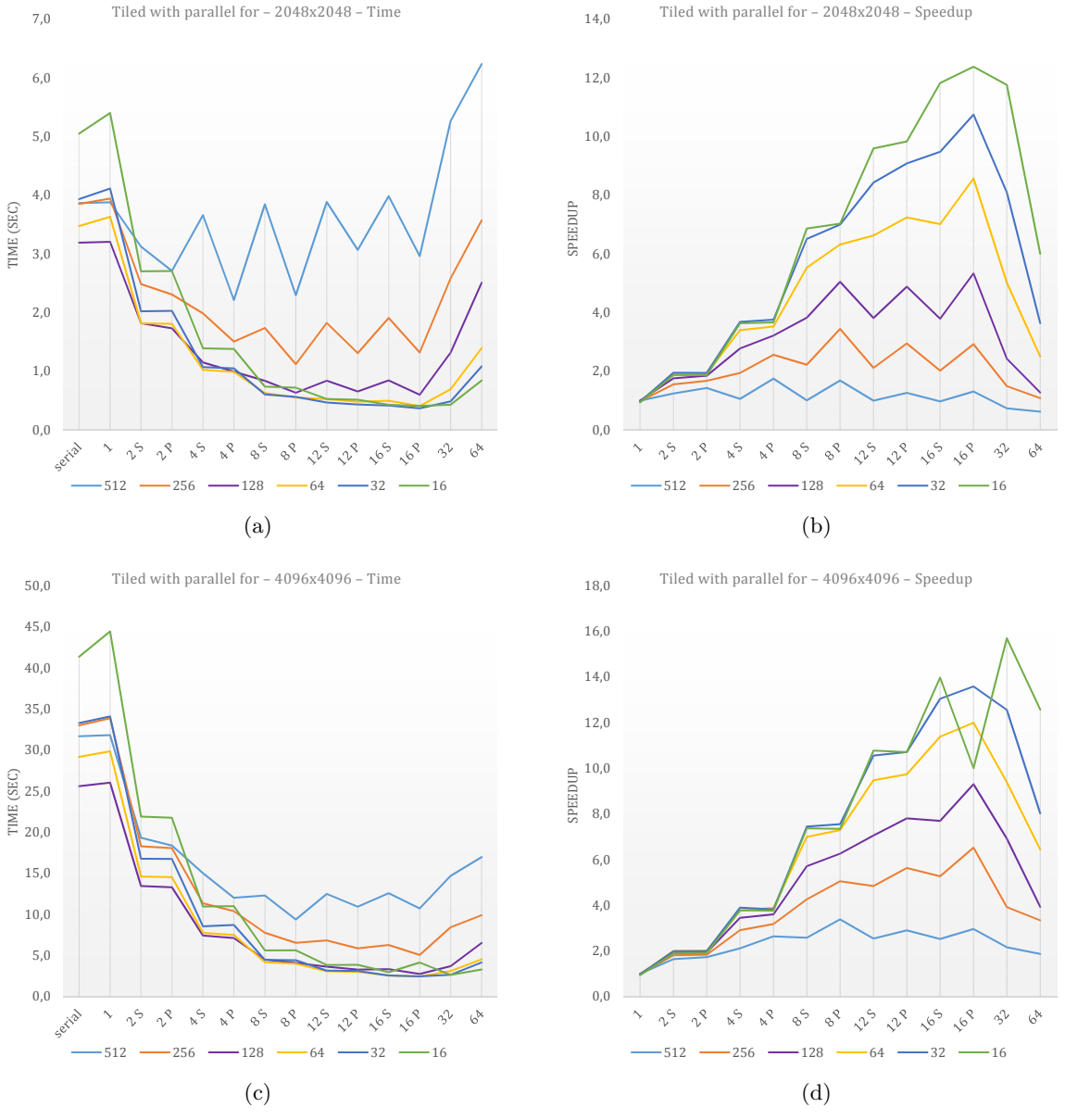
(a) (b)

(c) (d)

Σχ. 4.13: Αποτελέσματα μετρήσεων tiled υλοποίησης με χρήση OpenMP tasks στο σύστημα Sandy Bridge

Και στο σύστημα Sandy Bridge, η tiled υλοποίηση με tasks παρουσιάζει εξίσου καλά αποτελέσματα με προηγουμένως τόσο ως προς τους χρόνους όσο και ως προς την κλιμάκωση.

OpenMP parallel for: Στο *Σχήμα 4.14* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης με χρήση OpenMP parallel for στο σύστημα Sandy Bridge.



Σχ. 4.14: Αποτελέσματα μετρήσεων tiled υλοποίησης με χρήση OpenMP parallel for στο σύστημα Sandy Bridge

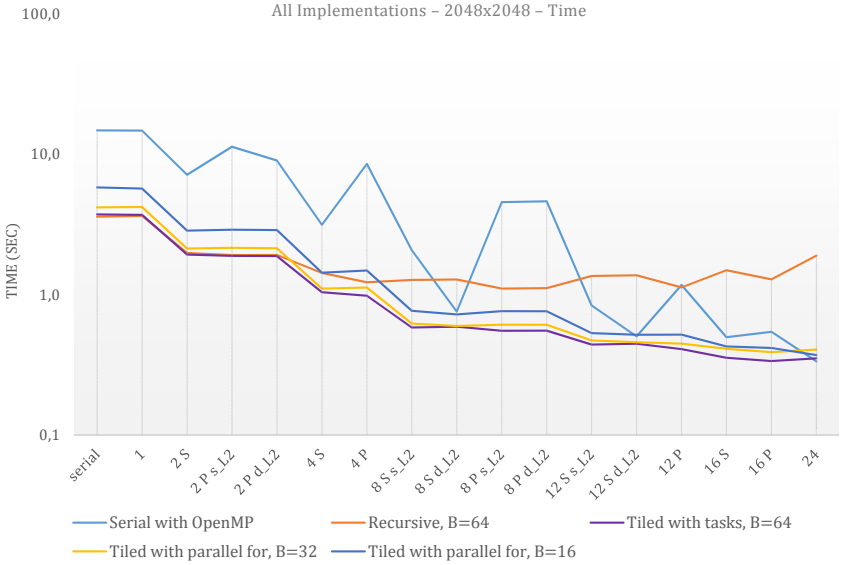
Τέλος, και σε αυτή την υλοποίηση παρουσιάζεται παρόμοια εικόνα με εκείνη που περιγράφηκε για το σύστημα Dunnington, με τη διαφορά ότι η υλοποίηση με tasks είναι αρκετά πιο γρήγορη από εκείνη με parallel for, όπως φαίνεται στον *Πίνακα 4.4*.

Tiled (32 threads)	2048×2048			4096×4096		
	$B = 64$	$B = 32$	$B = 16$	$B = 64$	$B = 32$	$B = 16$
OpenMP tasks	*0,276s*	*0,274s*	$0,709s$	*1,583s*	$1,695s$	$5,039s$
OpenMP parallel for	$0,692s$	$0,486s$	$0,443s$	$3,105s$	$2,650s$	$2,634s$

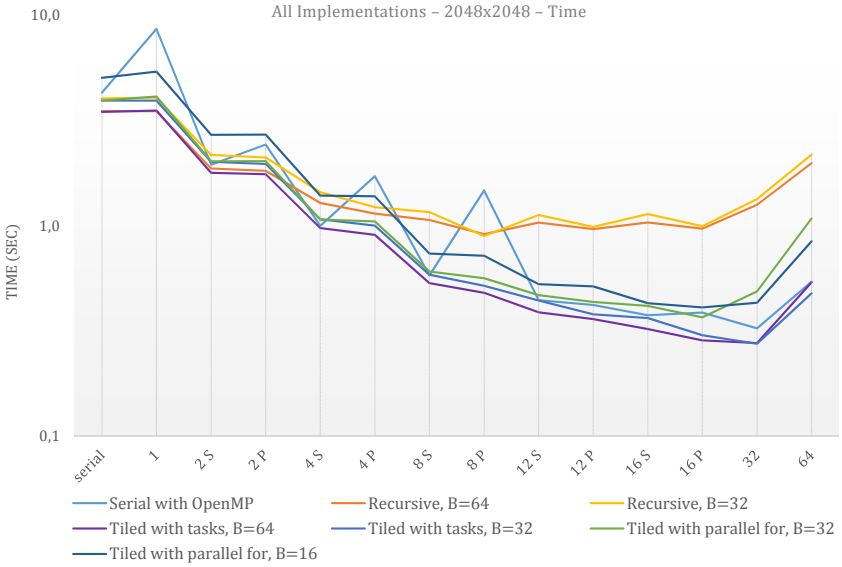
Πίν. 4.4: Καλύτεροι χρόνοι παράλληλων tiled υλοποιήσεων στο σύστημα Sandy Bridge

4.1.4. Συγκεντρωτικά αποτελέσματα

Στα *Σχήματα 4.15* και *4.16* φαίνονται συγκεντρωμένα οι μετρήσεις από όλες τις υλοποιήσεις για τους δύο γράφους που εξετάστηκαν στις προηγούμενες υποενότητες ανά σύστημα. Για όσες υλοποιήσεις περιέχουν blocking επιλέγονται τα μεγέθη block με τους καλύτερους χρόνους.

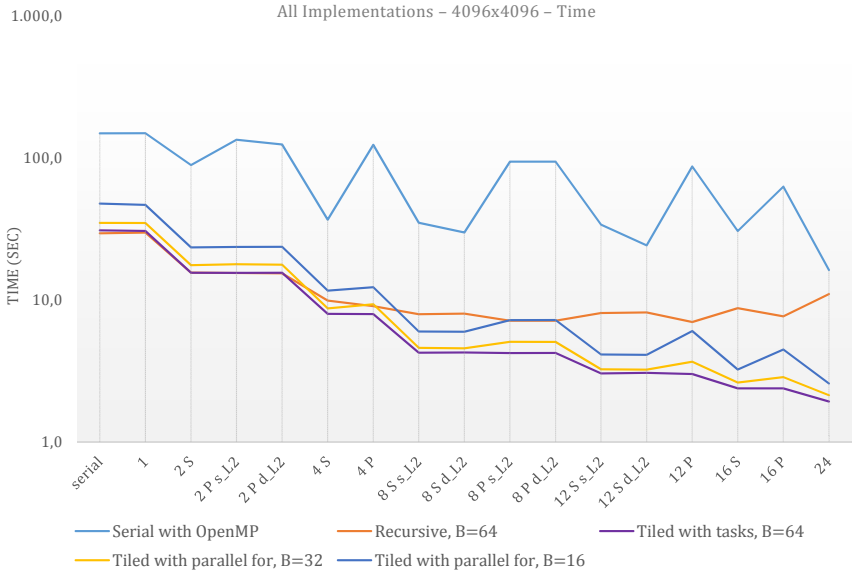


(a) Dunnington

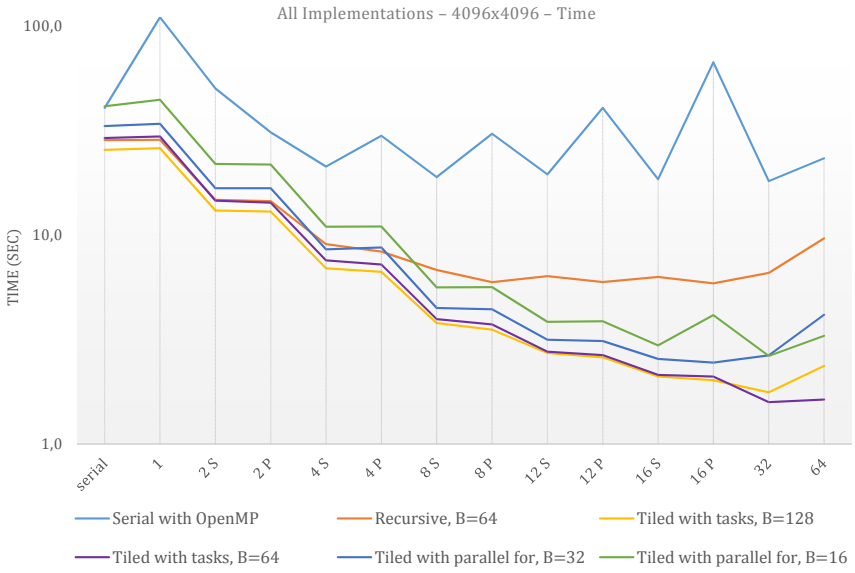


(b) Sandy Bridge

Σχ. 4.15: Συγκεντρωτικές μετρήσεις για το γράφο των 2048×2048 κόμβων από όλες τις υλοποιήσεις



(a) Dunnington

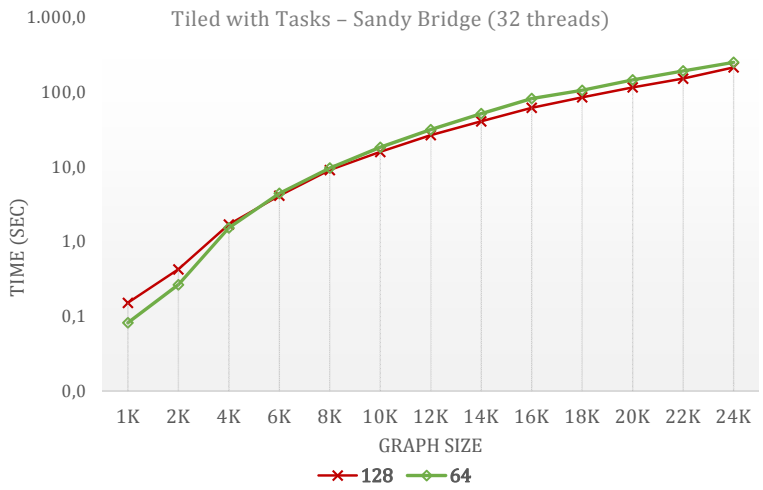


(b) Sandy Bridge

Σχ. 4.16: Συγκεντρωτικές μετρήσεις για το γράφο των 4096×4096 κόμβων από όλες τις υλοποιήσεις

Η tiled υλοποίηση εμφανίζει την καλύτερη κλιμάκωση από τις υπόλοιπες ενώ η παραλληλοποίησή της με OpenMP tasks δίνει τις γρηγορότερες επιδόσεις και στα δύο συστήματα που δοκιμάστηκαν. Καλύτερη τιμή μεγέθους block αποδεικνύεται η $B = 64\,(16\,KB)$, με την αμέσως προηγούμενη και την αμέσως επόμενη να εμφανίζουν ελάχιστα διαφοροποιημένους χρόνους. Επίσης, παρατηρείται ότι η τεχνολογία Hyper-Threading των επεξεργαστών του συστήματος Sandy Bridge. Στο σημείο αυτό επισημαίνεται ότι στα παραπάνω διαγράμματα ο κάθετος άξονας του χρόνου είναι σε λογαριθμική κλίμακα.

Τέλος, προκειμένου να διερευνηθεί το καλύτερο μέγεθος block της tiled υλοποίησης με tasks για μεγαλύτερους πίνακες στο σύστημα Sandy Bridge, δοκιμάζονται τα δύο καλύτερα μεγέθη block για διάφορα μεγέθη γράφων, όπως φαίνεται στο *Σχήμα 4.17*.



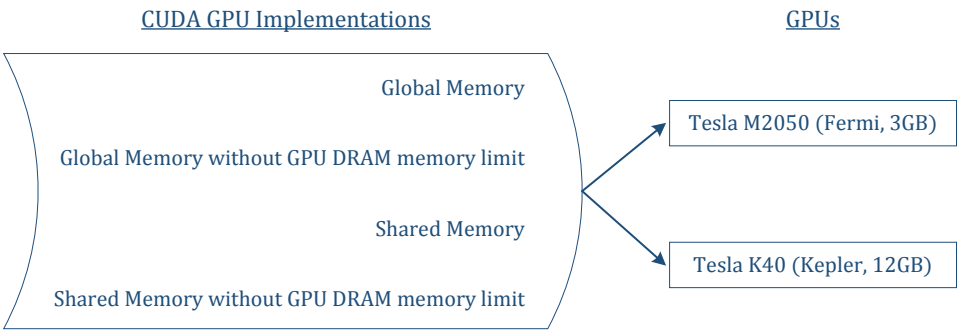
Σχ. 4.17: Χωρισμός πίνακα σε υποπίνακες και διαδοχικές αναδρομικές κλήσης σε κάθε επίπεδο της αναδρομής

Παρατηρείται ότι αυξάνεται το μέγεθος του γράφου, η χρήση block $B = 128$ εμφανίζει καλύτερα αποτελέσματα από το αντίστοιχο $B = 64$. Προφανώς, υπερισχύει το γεγονός των μειωμένων μεταφορών από και προς την κύρια μνήμη του συστήματος σε σχέση με το γεγονός ότι το tile $128 \times 128 \times 4B = 64\,KB$ δε χωράει ολόκληρο στην κρυφή μνήμη $L1$. Στα διαγράμματα που ακολουθούν, όπου συμπεριλαμβάνονται μετρήσεις από την tiled υλοποίηση στο σύστημα Sandy Bridge, υπονοείται μέγεθος block ίσο $B = 128$ και 32 threads.

4.2. Υλοποίηση σε κάρτες γραφικών

Στην ενότητα αυτή παρουσιάζονται και αναλύονται οι διαφορετικές υλοποιήσεις που έγιναν για κάρτες γραφικών. Αρχικά, η πρώτη υλοποίηση «Global Memory» είναι η πιο απλή και δεν κάνει χρήση της ενσωματωμένης shared memory της κάρτας γραφικών. Στη συνέχεια, τροποποιείται κατάλληλα ώστε το περιορισμένο μέγεθος της κύριας μνήμης DRAM της κάρτας γραφικών να μην αποτελεί εμπόδιο και να λειτουργεί για οσοδήποτε μεγάλους γράφους απαιτείται.

Έπειτα, αναλύεται η υλοποίηση «Shared Memory» όπου γίνεται χρήση της ενσωματωμένης μοιραζόμενης μνήμης και επιτυγχάνονται πολύ καλύτερες επιδόσεις από την πρώτη υλοποίηση. Τέλος, γίνεται επέκταση της υλοποίησης «Shared Memory» ώστε να ξεπεραστεί το εμπόδιο του περιορισμένου μεγέθους της κύριας μνήμης της κάρτας γραφικών.



Σχ. 4.18: Παράλληλες υλοποιήσεις αλγορίθμου Floyd-Warshall για κάρτες γραφικών

Μετρήσεις

Διεξάγονται μετρήσεις στις δύο κάρτες γραφικών αρχιτεκτονικών Fermi και Kepler της εταιρίας Nvidia που αναφέρθηκαν στην *Υποενότητα 2.4.2*. Χρησιμοποιείται η έκδοσης 5.0 της πλατφόρμας Nvidia CUDA. Στις μετρήσεις αναφέρονται οι χρόνοι που επιτυγχάνει κάθε υλοποίηση σε κάθε μία από τις δύο κάρτες γραφικών που εξετάζονται, για γράφο μεγέθους 8192×8192 κόμβων. Όμοια με προηγουμένως, ο γράφος αποθηκεύεται στη μνήμη με τη μορφή πίνακα γειτνίασης και αναπαρίστανται με τύπο `float`, δηλαδή με αριθμούς κινητής υποδιαστολής απλής ακρίβειας (32 bit). Συνεπώς, ο γράφος που εξετάζεται έχει μέγεθος $8192 \times 8192 \times 4B = 256\,MB$. Στους αναφερόμενους χρόνους συμπεριλαμβάνεται ο χρόνος μεταφοράς των δεδομένων από και προς την κάρτα γραφικών.

Ο υπολογισμός του speedup κάθε αποτελέσματος γίνεται έχοντας ως βάση το σειριακό χρόνο εκτέλεσης στο σύστημα που φιλοξενεί την κάρτα γραφικών Tesla M2050 του απλού αλγορίθμου με τα 3 loops που αναλύθηκε στην *Υποενότητα 4.1.1*. Πρόκειται για σύστημα με δύο επεξεργαστές Intel Xeon X5650 αρχιτεκτονικής Westmere χρονισμένους στα $2.67\,GHz$ και $48\,GB$ μνήμης RAM. Η μεταγλώττιση του προγράμματος του απλού αλγορίθμου γίνεται με τον compiler GCC έκδοσης 4.6.3 με επίπεδο βελτιστοποιήσεων -O3. Ο χρόνος που χρειάζεται για να επεξεργαστεί το γράφο των 8192 κόμβων είναι $1155,90\,sec$.

Όλες οι υλοποιήσεις που θα αναλυθούν σε αυτή την ενότητα αναπτύχθηκαν αρχικά στο σύστημα με την κάρτα γραφικών Tesla M2050 γενιάς Fermi και κριτήριο για την απόδοση και την βελτιστοποίησή τους ήταν τα αποτελέσματα σε αυτή την κάρτα. Αργότερα, όταν είχε

ολοκληρωθεί η ανάπτυξη όλων των υλοποιήσεων, αποκτήθηκε πρόσβαση και στη νεότερη κάρτα Tesla K40 γενιάς Kepler, οπότε συμπεριλαμβάνονται μετρήσεις και από αυτήν.

4.2.1. Global Memory υλοποίηση

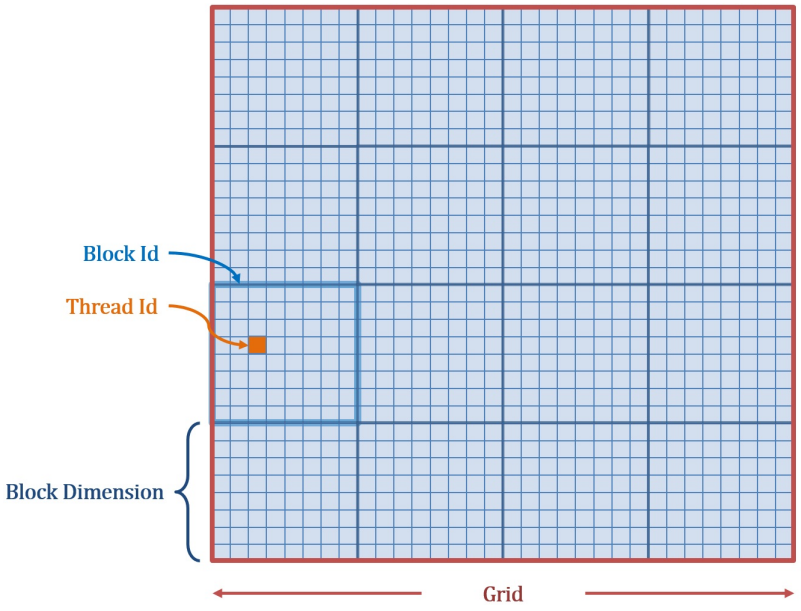
Αρχικά, εξετάζεται η πιο απλή υλοποίηση του αλγορίθμου Floyd-Warshall στην κάρτα γραφικών, η ονομαζόμενη Global Memory. Ο πίνακας γειτνίασης του γράφου μεγέθους $N \times N$ μεταφέρεται ολόκληρος στην κάρτα γραφικών και δημιουργείται ένας kernel ο οποίος υπολογίζει μία επανάληψη k του εξωτερικού loop του απλού αλγορίθμου με τα 3 loops. Ο kernel αυτός εκτελείται $k = N$ φορές και μετά την ολοκλήρωση της εκτέλεσης στην κάρτα γραφικών το αποτέλεσμα αντιγράφεται στην κύρια μνήμη του συστήματος.

Έγιναν τρεις υλοποιήσεις Global Memory, κάθε μια από τις οποίες αποτελεί βελτίωση της προηγούμενης. Η πρώτη έχει block 2 διαστάσεων (2D), η δεύτερη block μίας διάστασης (1D) και η τρίτη και γρηγορότερη block μίας διάστασης αλλά κάθε thread υπολογίζει 4 στοιχεία του πίνακα αντί για ένα όπως οι προηγούμενες (1D - 4 elements per thread). Σε όλες τις υλοποιήσεις οι διαστάσεις των block επιλέγονται έτσι ώστε τα block να περιέχουν 256 threads, διότι έτσι επιτυγχάνεται 100% χρήση των SM/SMX των καρτών γραφικών.

2D block

Σε αυτή την υλοποίηση, ο πίνακας χωρίζεται σε 2D blocks διαστάσεων 16×16 με συνολικά 256 threads το καθένα, όπως φαίνεται στο *Σχήμα 4.19*. Συνεπώς, το grid έχει μέγεθος $(N/16, N/16)$. Κάθε thread επεξεργάζεται ένα στοιχείο του πίνακα. Υπολογίζοντας τη θέση του μέσα στο grid βρίσκει ποιο στοιχείο του πίνακα πρέπει να επεξεργαστεί, όπως αναλύθηκε στην *Υποενότητα 2.2.4*.

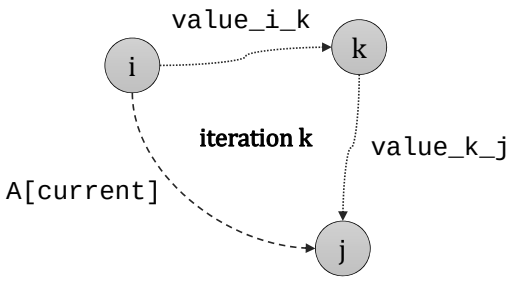
Στον *Κώδικα 4.6* φαίνεται η υλοποίηση του kenrel και η κλήση του από το κύριο πρόγραμμα που εκτελείται στον επεξεργαστή.



Σχ. 4.19: Κατανομή blocks της υλοποίησης Global Memory - 2D

Κώδικας 4.6 Υλοποίηση «CUDA Global Memory - 2D» του αλγορίθμου Floyd-Warshall

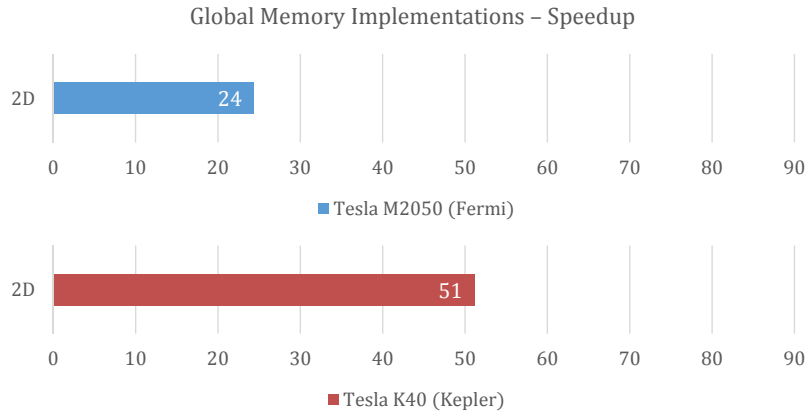
```
__global__ void _GPU_kernel_2D(const int k, float *A, const int N)
{
    int j = blockIdx.x * blockDim.x + threadIdx.x;
    int i = blockIdx.y * blockDim.y + threadIdx.y;

    int current = N * i + j;

    float value_i_k = A[i * N + k];
    float value_k_j = A[k * N + j];
    float temp = value_i_k + value_k_j;

    if (temp < A[current])
        A[current] = temp;
}

main()
{
    ...

    dim3 grid(N/16,N/16);
    dim3 block(16,16);

    for(int k=0; k<N; k++)
        _GPU_kernel_2D<<<grid,block>>>(k, A, N);

    ...
}
```



Στην υλοποίηση αυτή, οι προσβάσεις στη μνήμη για την ανάγνωση των τιμών `value_k_j` και `A[current]` είναι coalesced (συνενωμένες), αφού πρόκειται για συνεχόμενες θέσεις μνήμης. Αντίθετα, οι αναγνώσεις της τιμής `value_i_k` δεν είναι, αφού κάθε μία ανήκει σε διαφορετική γραμμή του πίνακα.

Όταν οι προσβάσεις των threads ενός warp στην global memory της κάρτας γραφικών είναι coalesced, τότε αυτές συνενώνονται σε όσο λιγότερες transactions γίνεται ώστε να μειωθεί ο χρόνος μεταφοράς. Όταν οι προσβάσεις δεν είναι coalesced, απαιτείται ένα transaction για κάθε πρόσβαση ξεχωριστά.

Στο *Σχήμα 4.20* φαίνεται το speedup που επιτυγχάνεται σε κάθε κάρτα γραφικών. Στην Tesla M2050 ο χρόνος εκτέλεσης είναι $47{,}37s$, ενώ στην Tesla K40 είναι $22{,}59s$.



Σχ. 4.20: Αποτελέσματα υλοποίησης Global Memory - 2D για γράφο μεγέθους 8192×8192

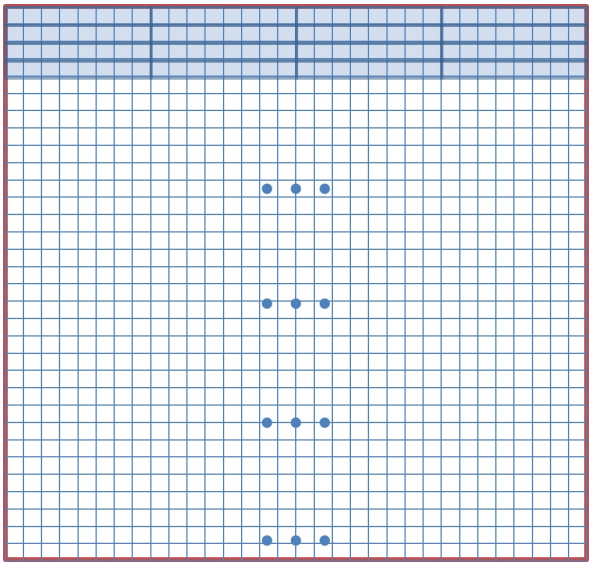
1D block

Η επόμενη υλοποίηση που δοκιμάστηκε είναι όμοια με την προηγούμενη με τη διαφορά ότι πλέον το block είναι μίας διάστασης και οριζόντιο. Εντούτοις, συνεχίζει να αποτελείται από 256 threads και κάθε thread να υπολογίζει ένα στοιχείο του πίνακα, όπως φαίνεται στο *Σχήμα 4.21*.

Ο λόγος που προτιμήθηκε η χρήση οριζόντιου block μίας διάστασης είναι ώστε όλα τα thread που ανήκουν στο ίδιο block να υπολογίζουν στοιχεία που ανήκουν στην ίδια γραμμή του πίνακα και συνεπώς έχουν την ίδια τιμή `value_i_k`. Έτσι:

- Μειώνονται οι (μη coalesced) αναγνώσεις από την global memory των τιμών `value_i_k` από 16 σε μόνο μία για κάθε block.
- Όταν ολοκληρωθεί η ανάγνωση από ένα thread κάθε block, τα υπόλοιπα δε χρειάζεται να καταφύγουν στη global memory για την ίδια τιμή αλλά την βρίσκουν στην L1 cache, αφού όλες οι προσβάσεις στη global memory είναι cached.

Στον *Κώδικα 4.7* φαίνεται η υλοποίηση του kenrel και η κλήση του από το κύριο πρόγραμμα.

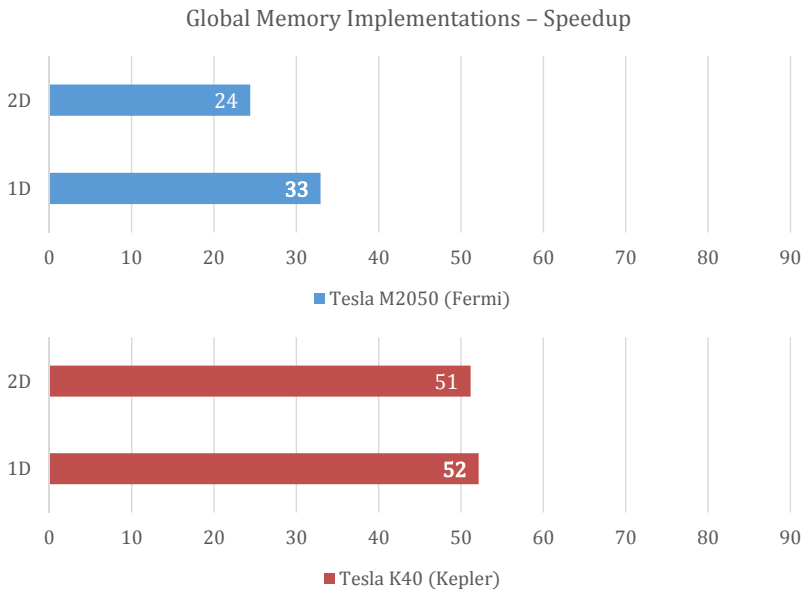


Σχ. 4.21: Κατανομή blocks της υλοποίησης Global Memory - 1D

Κώδικας 4.7 Υλοποίηση «CUDA Global Memory - 1D» του αλγορίθμου Floyd-Warshall

```
__global__ void _GPU_kernel_1D(const int k, float *A, const int N)
{
    int j = blockIdx.x * blockDim.x + threadIdx.x;
    int i = blockIdx.y;

    int current = N * i + j;
    float value_i_k = A[i * N + k]; float value_k_j = A[k * N + j];
    float temp = value_i_k + value_k_j;

    if (temp < A[current])
        A[current] = temp;
}

main()
{
    ...

    dim3 grid(N/256,N);
    dim3 block(256,1);

    for(int k=0; k<N; k++)
        _GPU_kernel_1D<<<grid,block>>>(k, A, N);

    ...
}
```

Στο *Σχήμα 4.22* φαίνεται το speedup που επιτυγχάνεται σε κάθε κάρτα γραφικών. Στην Tesla M2050 παρατηρείται μείωση του χρόνου εκτέλεσης σε $35{,}08\,s$, ενώ στην Tesla K40 μειώνεται οριακά σε $22{,}17\,s$.



Σχ. 4.22: Αποτελέσματα υλοποίησης Global Memory - 2D για γράφο μεγέθους 8192×8192

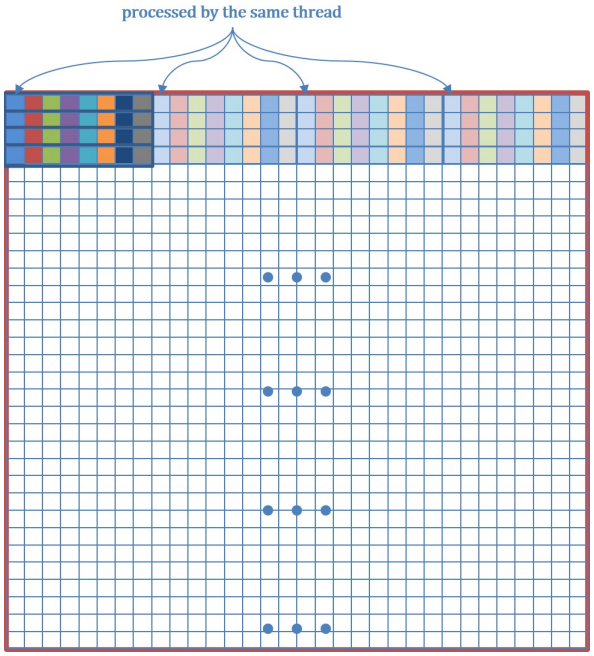
1D Block - 4 elements per thread

Η τρίτη και πιο αποδοτική υλοποίηση τροποποιεί κατάλληλα την προηγούμενη υλοποίηση ώστε κάθε thread να υπολογίζει 4 στοιχεία του πίνακα αντί για ένα. Το block παραμένει οριζόντιο μίας διάστασης και με 256 threads. Συνεπώς, ένα block δεν υπολογίζει 256 στοιχεία μιας γραμμής του πίνακα αλλά $4 \times 256 = 1024$ στοιχεία. Όπως φαίνεται στο *Σχήμα 4.23*, κάθε thread υπολογίζει το στοιχείο που προκύπτει κανονικά από τη θέση του στο grid και επιπλέον τα επόμενα 3 στοιχεία τα οποία απέχουν μεταξύ τους 256 θέσεις, όσο είναι το μέγεθος του block.

Για να γίνει περισσότερο κατανοητό, έστω για παράδειγμα πίνακας μεγέθους 1024×1024. Το thread$(0, 0)$ θα υπολογίσει τα στοιχεία $A[0][0]$, $A[0][256]$, $A[0][512]$, $A[0][768]$, ενώ το thread $(0, 255)$ θα υπολογίσει τα στοιχεία $A[0][255]$, $A[0][511]$, $A[0][767]$, $A[0][1023]$. Τελικά, με ένα μόνο block των 256 thread υπολογίζονται και τα 1024 στοιχεία μίας γραμμής, αντί για 4 blocks που απαιτούνται στην προηγούμενη υλοποίηση. Με τον τρόπο αυτό:

- Όλες οι προσβάσεις στη global memory παραμένουν coalesced, αφού τα στοιχεία που διαβάζει και επεξεργάζεται κάθε thread απέχουν τον ίδιο σταθερό αριθμό θέσεων και συνεπώς είναι συνεχόμενα.
- Γίνονται 4 φορές λιγότερες προσβάσεις στη global memory για την ανάγνωση του στοιχείου `value_i_k`.
- Ο δρομολογητής έχει μεγαλύτερη ευελιξία στη δρομολόγηση των warps και στην εξαγωγή ILP από τα εκτελούμενα warps.

Στο σημείο αυτό πρέπει να σημειωθεί ότι σε περίπτωση που το μέγεθος του πίνακα δεν είναι ακριβές πολλαπλάσιο του 1024 (ή εν γένει του $4 \times BLOCK\_SIZE$), υπάρχουν διάφοροι τρόποι αντιμετώπισης όπως το padding, ο υπολογισμός των στοιχείων που περισσεύουν με τον απλό 1D kernel κ.ά. (με κάποια μείωση των επιδόσεων σε κάθε περίπτωση).



Σχ. 4.23: Κατανομή blocks της υλοποίησης Global Memory - 1D - 4 elements per thread

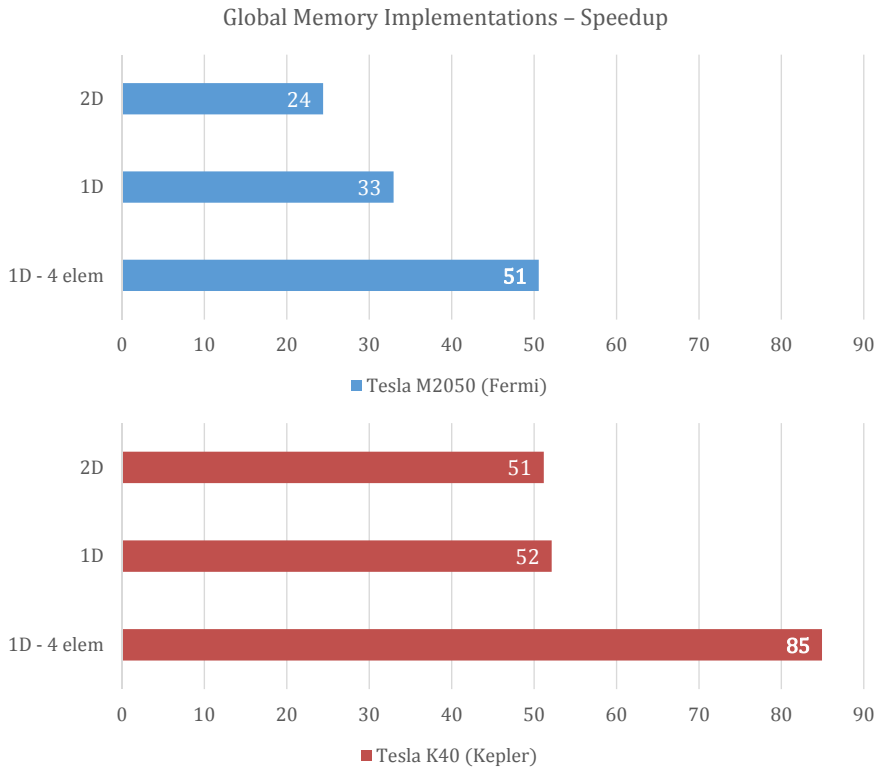
Στον *Κώδικα 4.8* φαίνεται η υλοποίηση του kenrel και η κλήση του από το κύριο πρόγραμμα που εκτελείται στον επεξεργαστή.

Κώδικας 4.8 Υλοποίηση «CUDA Global Memory - 1D - 4 elements per thread» του αλγορίθμου Floyd-Warshall

```
#define BLOCK_SIZE 256

__global__ void _GPU_kernel_1D_4elem(const int k, float *A, const int N)
{
    int i = N * blockIdx.y;
    int j = (4*blockIdx.x) * blockDim.x + threadIdx.x;

    float value_i_k = A[i + k];
    float value_k_j1 = A[k * N + j];
    float value_k_j2 = A[k * N + j + BLOCK_SIZE];
    float value_k_j3 = A[k * N + j + 2*BLOCK_SIZE];
    float value_k_j4 = A[k * N + j + 3*BLOCK_SIZE];

    int current1 = i + j;
    int current2 = current1 + BLOCK_SIZE;
    int current3 = current1 + 2*BLOCK_SIZE;
    int current4 = current1 + 3*BLOCK_SIZE;

    float temp1 = value_i_k + value_k_j1;
    float temp2 = value_i_k + value_k_j2;
    float temp3 = value_i_k + value_k_j3;
    float temp4 = value_i_k + value_k_j4;

    if (temp1 < A[current1])
        A[current1] = temp1;
    if (temp2 < A[current2])
        A[current2] = temp2;
    if (temp3 < A[current3])
        A[current3] = temp3;
    if (temp4 < A[current4])
        A[current4] = temp4;
}

main()
{
    ...

    dim3 grid(N/BLOCK_SIZE/4,N);
    dim3 block(BLOCK_SIZE,1);

    for(int k=0; k<N; k++)
        _GPU_kernel_1D_4elem<<<grid,block>>>(k, A, N);

    ...
}
```

Στο *Σχήμα 4.24* φαίνεται το speedup που επιτυγχάνεται σε κάθε κάρτα γραφικών. Παρατηρείται μεγάλη βελτίωση των επιδόσεων και στις δύο κάρτες. Στην Tesla M2050 παρατηρείται περαιτέρω μείωση του χρόνου εκτέλεσης στα *22,86 s* και στην Tesla K40 στα *13,61 s*.

Το τελικό speedup που επιτυγχάνεται για την Tesla M2050 είναι ίσο με 51, δηλαδή ο χρόνος εκτέλεσης είναι 51 φορές μικρότερος από τον αντίστοιχο σειριακό του απλού αλγορίθμου με τα 3 loops στο σύστημα με επεξεργαστές Intel Xeon X5650 που φιλοξενεί την κάρτα Tesla M2050. Για την Tesla K40, το τελικό speedup είναι ίσο με 85.



Σχ. 4.24: Αποτελέσματα υλοποίησης Global Memory - 1D - 4 elements per thread για γράφο μεγέθους 8192×8192

Στον *Πίνακα 4.5* φαίνονται συγκεντρωμένα οι χρόνοι όλων των υλοποιήσεων για κάθε κάρτα γραφικών.

Global Memory Impl.	**Tesla M2050**	**Tesla K40**
2D	47,37 *sec*	22,59 *sec*
1D	35,08 *sec*	22,17 *sec*
1D - 4 elem. per thread	22,86 *sec*	13,61 *sec*

Πίν. 4.5: Χρόνοι εκτέλεσης όλων των Global Memory υλοποιήσεων

Στα διαγράμματα που ακολουθούν στις επόμενες ενότητες, όπου αναφέρεται η υλοποίηση «Global Memory» υπονοείται η πιο αποδοτική υλοποίηση «1D - 4 elements per thread».

4.2.2. Global Memory υλοποίηση χωρίς περιορισμό μνήμης κάρτας γραφικών

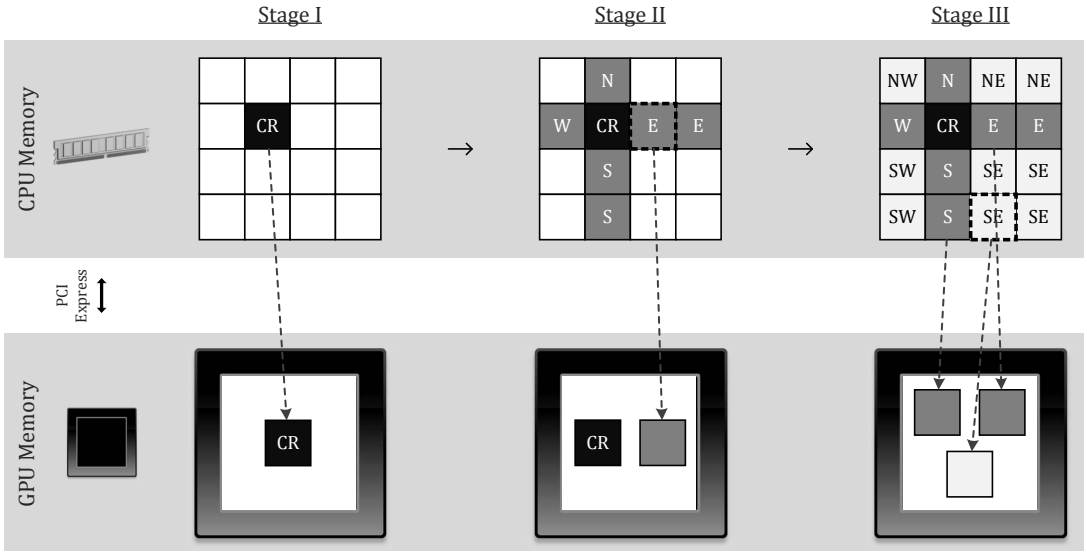
Η μνήμη DRAM της κάρτας γραφικών είναι περιορισμένη. Συνεπώς, το μέγιστο μέγεθος γράφου που μπορούν να υπολογίσουν όλες οι προηγούμενες υλοποιήσεις εξαρτάται άμεσα από τη μνήμη της εκάστοτε κάρτας γραφικών. Για παράδειγμα, η Tesla M2050 έχει συνολική μνήμη DRAM ίση με $3\,GB$, από τα οποία $2.6\,GB$ είναι διαθέσιμα στον προγραμματιστή όταν το ECC είναι ενεργοποιημένο, μέγεθος το οποίο δεν κρίνεται ιδιαίτερα μεγάλο.

Επομένως, είναι σημαντική η προσπάθεια τροποποίησης της υλοποίησης Global Memory ώστε το μέγιστο μέγεθος του γράφου να πάψει να εξαρτάται από τη μνήμη της κάρτας γραφικών. Στο σημείο αυτό έρχεται να βοηθήσει η τεχνική του tiling, η οποία αναλύθηκε διεξοδικά στην *Υποενότητα 4.1.3*. Ο πίνακας χωρίζεται σε μεγάλου μεγέθους tiles και ακολουθείται η γνωστή διαδικασία των τριών σταδίων για κάθε k επανάληψη, όπως φαίνεται στο *Σχήμα 4.25*:

- Αρχικά, αντιγράφεται από την κύρια μνήμη του συστήματος στη μνήμη της κάρτας γραφικών το διαγώνιο (k, k) tile (μαύρο, CR tile), επεξεργάζεται και αντιγράφεται πίσω το αποτέλεσμα.

- Στο δεύτερο στάδιο, αντιγράφονται διαδοχικά στην κάρτα γραφικών όλα τα tiles που βρίσκονται στην $k-$οστή γραμμή και στην $k-$οστή στήλη (σκούρο γκρι, $\{N, S, E, W\}$) και επεξεργάζονται. Σημειώνεται ότι το διαγώνιο tile από το οποίο εξαρτώνται υπάρχει ήδη υπολογισμένο στη μνήμη της κάρτας γραφικών.

- Τέλος, στο τρίτο στάδιο, αντιγράφονται διαδοχικά όλα τα υπόλοιπα tiles του πίνακα (ανοιχτό γκρι, $\{NW, SW, NE, SE\}$) καθώς και τα tiles της αντίστοιχης γραμμής και στήλης από τα οποία εξαρτώνται. Οι αντιγραφές γίνονται με κατάλληλο τρόπο ώστε να ελαχιστοποιηθεί ο συνολικός αριθμός αντιγραφών. Μετά την επεξεργασία κάθε tile, το αποτέλεσμα μεταφέρεται πίσω στην κύρια μνήμη του συστήματος.

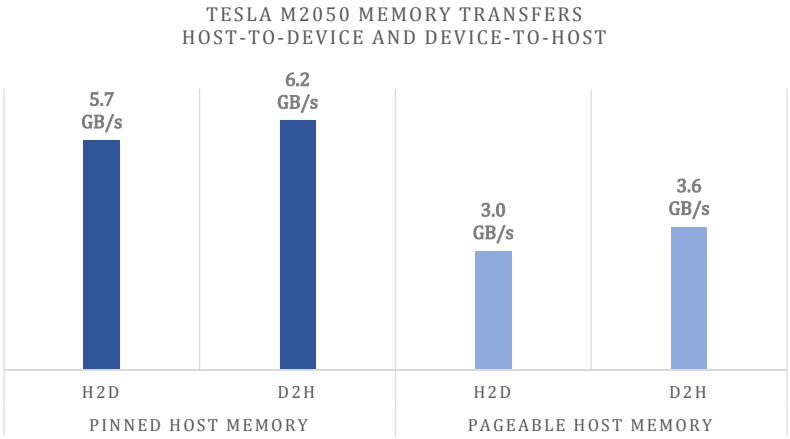


Σχ. 4.25: Εφαρμογή τεχνικής tiling στη Global Memory υλοποίηση για οποιοδήποτε μέγεθος μνήμης κάρτας γραφικών

Από τα παραπάνω γίνεται προφανές ότι το μέγιστο μέγεθος του tile είναι το 1/3 της συνολικής διαθέσιμης μνήμης της κάρτας γραφικών, αφού στο 3ο στάδιο θα πρέπει να αποθηκευτούν 3 tiles ταυτόχρονα στη μνήμη της κάρτας γραφικών. Με αυτό τον τρόπο μπορεί να υπολογιστεί οσοδήποτε μεγάλος γράφος ανεξάρτητα από το μέγεθος της μνήμης της εκάστοτε κάρτας, αρκεί να καθοριστεί σωστά το μέγεθος του tile.

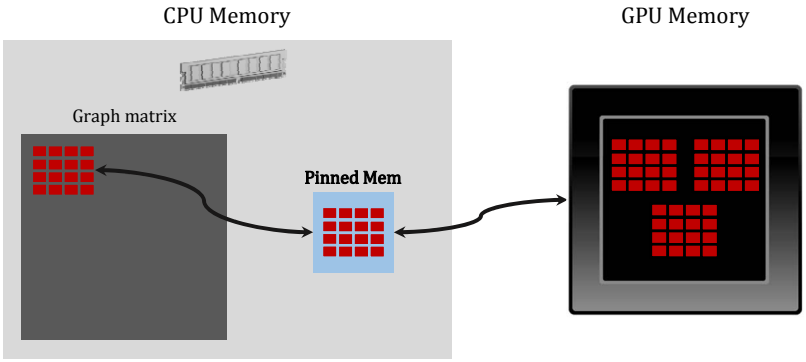
Μεταφορά δεδομένων από και προς την κάρτα γραφικών

Η σύνδεση των καρτών γραφικών στο σύστημα γίνεται μέσω του διαύλου PCI Express και μέσω αυτού γίνεται η μεταφορά των δεδομένων. Η Tesla M2050 υποστηρίζει την έκδοση 2.0 του διαύλου PCI Express και το μέγιστο θεωρητικό εύρος ζώνης ανέρχεται σε $8\,GB/s$ (χρησιμοποιεί 16 lanes). Για να επιτευχθεί η μέγιστη ταχύτητα αντιγραφής ανάμεσα στην κύρια μνήμη του συστήματος (host) και τη μνήμη της κάρτας γραφικών (device), θα πρέπει η μνήμη που έχει δεσμευτεί στο σύστημα να είναι pinned και όχι pageable. Όταν μία θέση μνήμης δηλώνεται ως pinned, το λειτουργικό σύστημα δεν μπορεί να την κάνει swap σε ένα δευτερεύον σύστημα αποθήκευσης (λ.χ. σκληρός δίσκος) και παραμένει μόνιμα στη φυσική μνήμη RAM. Στο *Σχήμα 4.26* φαίνονται οι ταχύτητες διαμεταγωγής προς ($H2D$) και από ($D2H$) την κάρτα γραφικών Tesla M2050, χρησιμοποιώντας και τα δύο είδη μνήμης pinned και pageable.



Σχ. 4.26: Εύρος ζώνης αντιγράφης προς και από την κάρτα γραφικών Tesla M2050

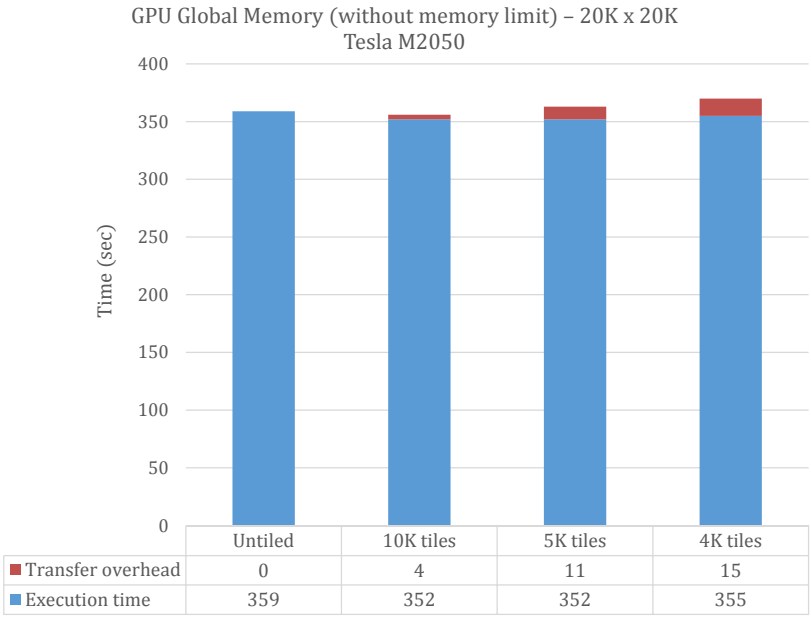
Παρατηρείται τεράστια διαφορά στο πραγματικό εύρος ζώνης ανάμεσα σε pinned και pageable είδος μνήμης, αν και το υπολείπεται από το θεωρητικό μέγιστο. Για το λόγο αυτό, το προτεινόμενο είναι να δεσμευτεί όλος ο πίνακας με pinned memory. Σε περίπτωση που κάτι τέτοιο δεν είναι εφικτό, η εναλλακτική είναι να δεσμευτεί με pinned μνήμη χώρος ίσος με το μέγεθος του tile και να χρησιμοποιηθεί ως buffer στις αντιγραφές, όπως φαίνεται στο *Σχήμα 4.27*. Η εναλλακτική αυτή ακολουθείται στην υλοποίηση της παρούσας διπλωματικής εργασίας ώστε να τονιστεί το αμελητέο overhead που έχει, όπως θα φανεί αργότερα στις μετρήσεις.



Σχ. 4.27: Αντιγραφή προς και από την κάρτα γραφικών μέσω pinned μνήμης ως buffer

Μετρήσεις

Για την παρουσίαση των αποτελεσμάτων θα χρησιμοποιηθεί η κάρτα Tesla M2050, της οποίας το μέγεθος μνήμης είναι αρκετά μικρότερο από αυτό της Tesla K40. Επιλέγεται γράφος μεγέθους $20480 \times 20480 \times 4B = 1600\,MB$, ο οποίος χωράει ολόκληρος στην Tesla M2050. Αρχικά υπολογίζεται ο χρόνος της κανονικής (untiled) υλοποίησης «Global Memory» της *Υποενότητας 4.2.1* και στη συνέχεια υπολογίζεται ο χρόνος της υλοποίησης χωρίς περιορισμό μνήμης για τρία διαφορετικά μεγέθη tile, ώστε να εκτιμηθεί το χρονικό overhead των αντιγραφών προς και από την κάρτα γραφικών. Επιλέγονται τα εξής μεγέθη tile: $10240 \times 10240 \times 4B = 400\,MB$, $5120 \times 5120 \times 4B = 100\,MB$, $4096 \times 4096 \times 4B = 64\,MB$. Στο *Σχήμα 4.28* φαίνεται το διάγραμμα των μετρήσεων.



Σχ. 4.28: Χρόνοι εκτέλεσης Global Memory υλοποίησης χωρίς περιορισμό μνήμης για γράφο μεγέθους 20480×20480 στην κάρτα Tesla M2050

Παρατηρείται ότι σε όλες τις περιπτώσεις το overhead που προκύπτει είναι ελάχιστο (4% στη χειρότερη περίπτωση). Επίσης, όσο μεγαλύτερο είναι το μέγεθος του tile, τόσο μικρότερο

είναι το overhead. Συνεπώς, η υλοποίηση Global Memory χωρίς περιορισμό μνήμης της κάρτας γραφικών είναι εξαιρετικά αποδοτική και πλέον μπορεί να επεξεργαστεί οποιοδήποτε μέγεθος πίνακα. Δοκιμάστηκαν μεγέθη πίνακα μέχρι και $52K \times 52K \times 4B = 10.56\,GB$ με απόλυτη επιτυχία.

Επιπλέον, η συγκεκριμένη υλοποίηση δοκιμάστηκε και στην κάρτα γραφικών Tesla K40, που διαθέτει τεράστια μνήμη μεγέθους $12\,GB$, και προέκυψε αντίστοιχη εικόνα αποτελεσμάτων με αυτή της Tesla M2050.

4.2.3. Shared Memory υλοποίηση

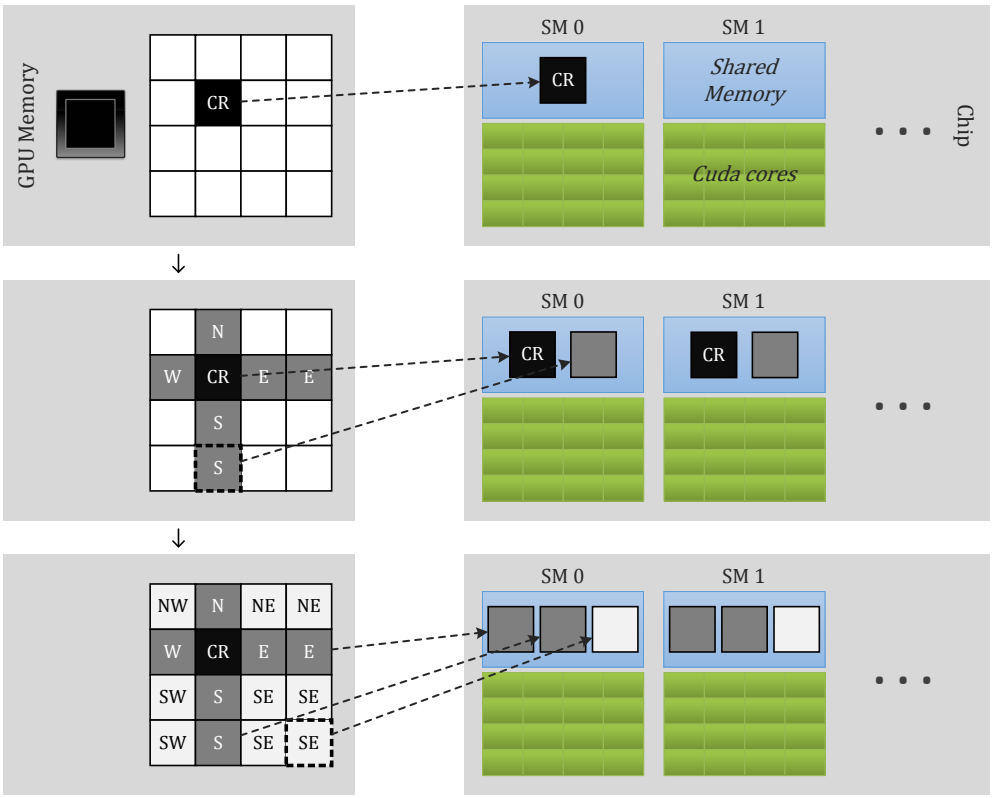
Παρά τις όποιες βελτιστοποιήσεις, η Global Memory υλοποίηση της *Υποενότητας 4.2.1* είναι memory bounded, αφού σε κάθε εκτέλεση του kernel πρέπει να μεταφέρεται ολόκληρος ο πίνακας από και προς την μνήμη DRAM της κάρτας γραφικών. Όπως αναφέρθηκε στον *Πίνακα 2.1*, το latency των προσβάσεων στη global memory κυμαίνεται ανάμεσα σε 400 και 600 κύκλους ρολογιού. Προκειμένου να αυξηθούν οι επιδόσεις πρέπει να μειωθούν όσο το δυνατόν περισσότερο οι μεταφορές δεδομένων ανάμεσα στο chip και την global memory. Για να επιτευχθεί αυτό είναι απαραίτητο να χρησιμοποιηθεί ένα από τα μεγάλα πλεονεκτήματα της κάρτας γραφικών, δηλαδή η ταχύτατη on chip shared memory η οποία είναι διαχειρίσιμη από τον προγραμματιστή.

Η τεχνική του tiling αποδεικνύεται για άλλη μια φορά εξαιρετικά χρήσιμη αφού πλέον θα εφαρμοστεί ανάμεσα στις global και shared memory. Ο πίνακας (ο οποίος βρίσκεται ολόκληρος αποθηκευμένος στη global memory της κάρτας) χωρίζεται σε μικρά tiles κατάλληλου μεγέθους ώστε να χωράνε στη shared memory και ακολουθείται η γνωστή διαδικασία, όπως φαίνεται στο *Σχήμα 4.29*:

- Αρχικά, αντιγράφεται από την global memory της κάρτας στη shared memory ενός SM το διαγώνιο (k, k) tile (μαύρο, CR tile), επεξεργάζεται και αντιγράφεται πίσω το αποτέλεσμα.
- Στο δεύτερο στάδιο, αντιγράφονται διαδοχικά από την global memory της κάρτας στις shared memory των SM όλα τα tiles που βρίσκονται στην $k-$ οστή γραμμή και στην $k-$ οστή στήλη (σκούρο γκρι, $\{N, S, E, W\}$) μαζί με το διαγώνιο tile από το οποίο εξαρτώνται και επεξεργάζονται. Ομοίως, το αποτέλεσμα αντιγράφεται πίσω στη global memory.
- Τέλος, στο τρίτο στάδιο, αντιγράφονται διαδοχικά στις shared memory των SM όλα τα υπόλοιπα tiles του πίνακα (ανοιχτό γκρι, $\{NW, SW, NE, SE\}$) καθώς και τα tiles της αντίστοιχης γραμμής και στήλης από τα οποία εξαρτώνται. Μετά την επεξεργασία τους, το αποτέλεσμα επιστρέφει πίσω στη global memory.

Χάρη στην τεχνική αυτή, μειώνεται ο αριθμός των προσβάσεων στη global memory τόσες φορές όσο είναι το μέγεθος του tile. Κατά το τρίτο στάδιο πρέπει να αποθηκευτούν στη shared memory κάθε SM 3 tiles. Επίσης, το μέγιστο μέγεθος της shared memory είναι $48KB$. Συνεπώς, το μέγιστο μέγεθος του tile είναι ίσο με:

$$\frac{48\,KB}{3} = 16\,KB \;\Rightarrow\; \frac{16384\,B}{4\,B/element} = 4096\,elements \;\Rightarrow\; 64 \times 64\,tile$$

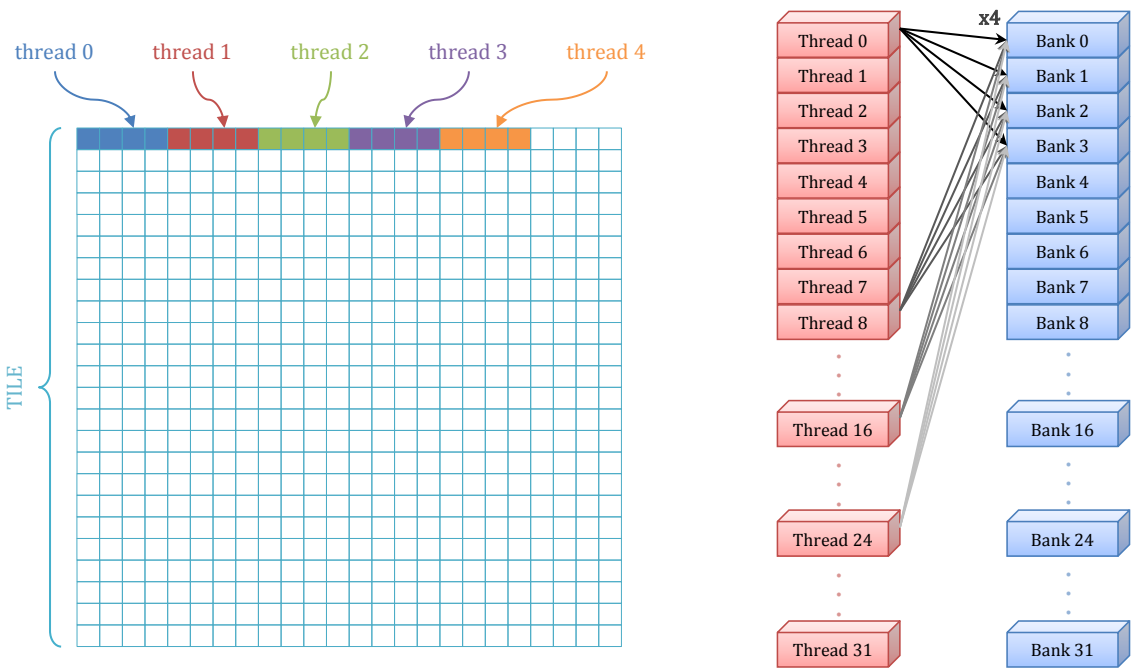


Σχ. 4.29: Εφαρμογή τεχνικής tiling στη Shared Memory υλοποίηση

Κατανομή των threads εντός των tiles

Με μέγιστες διαστάσεις των tiles ίσες με 64×64, κάθε tile αποτελείται από 4096 στοιχεία. Η shared memory είναι μοιραζόμενη ανάμεσα στα threads ενός block. Αφού με ένα block καταναλώνεται όλη η διαθέσιμη shared memory, για καλύτερες επιδόσεις πρέπει το block να έχει τον μέγιστο επιτρεπτό αριθμό από threads. Ο μέγιστος αριθμός threads ανά block που επιτρέπεται είναι 1024 threads. Συνεπώς, κάθε thread οφείλει να επεξεργαστεί 4 στοιχεία του πίνακα.

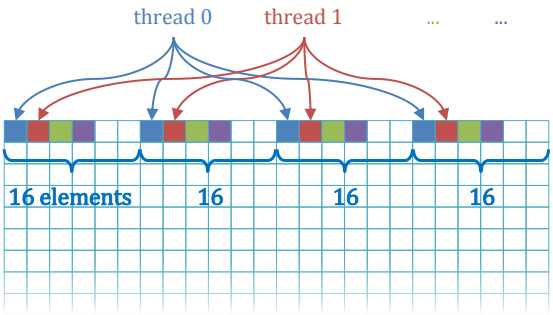
Η πρώτη κατανομή που δοκιμάστηκε είναι σε κάθε thread να ανατίθενται 4 συνεχόμενα στοιχεία του πίνακα προς επεξεργασία, όπως φαίνεται στο *Σχήμα 4.30*. Όμως, με αυτή την κατανομή δημιουργούνται 4-way bank conflicts ανάμεσα στα threads των warps αφού τα στοιχεία που πρέπει να επεξεργαστούν τα thread 0, 8, 16 και 24 βρίσκονται αποθηκευμένα στα ίδια memory banks. Τα bank conflicts αποτελούν σημαντική αιτία μείωσης των επιδόσεων αφού οι αιτήσεις πρόσβασης δεν εξυπηρετούνται ταυτόχρονα αλλά σειριακά.



Σχ. 4.30: Packed κατανομή των στοιχείων των threads εντός ενός tile

Αντίθετα, διατηρώντας τον ίδιο αριθμό threads ανά γραμμή του tile, στην spread κατανομή κάθε thread επεξεργάζεται στοιχεία της γραμμής ενός tile που απέχουν μεταξύ τους $^{64}/_{4} = 16$ θέσεις, όπως φαίνεται στο *Σχήμα 4.31*.

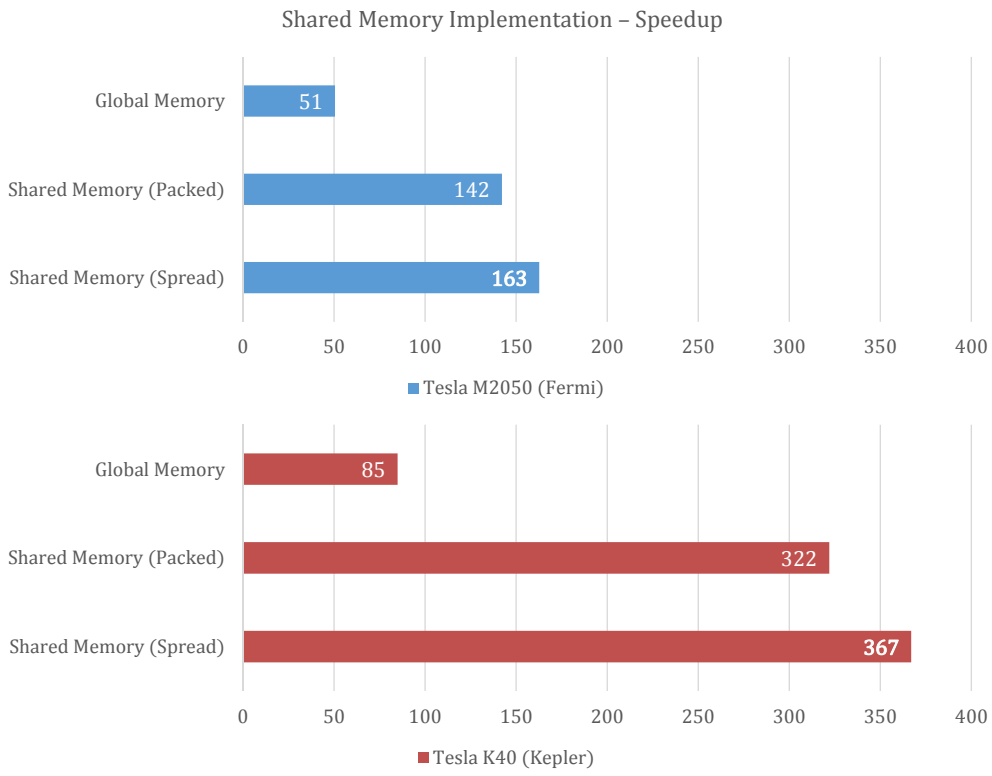
Με τον τρόπο αυτό, μειώνονται σε μεγάλο βαθμό τα bank conflicts, αυξάνοντας τις επιδόσεις όπως φαίνεται στις μετρήσεις που ακολουθούν στην επόμενη παράγραφο.



Σχ. 4.31: Spread κατανομή των στοιχείων των threads εντός ενός tile

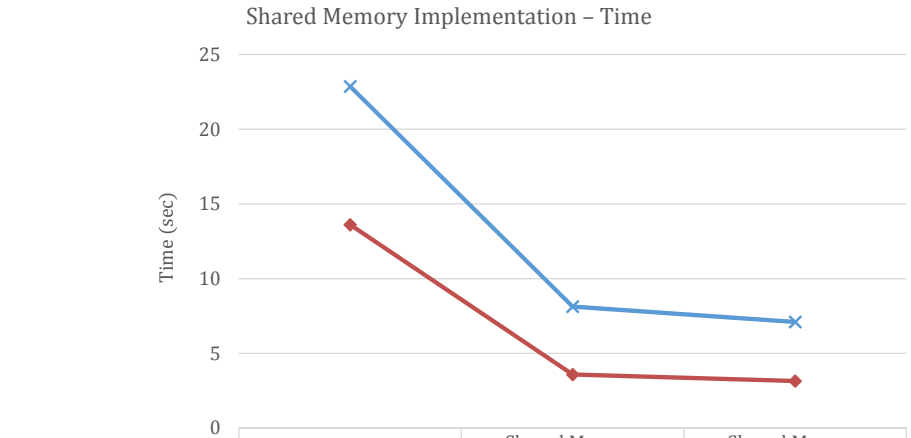
Μετρήσεις

Στο *Σχήμα 4.32* φαίνεται το speedup που επιτυγχάνεται σε κάθε κάρτα γραφικών για την υλοποίηση Shared Memory. Παρατηρείται θεαματική αύξηση των επιδόσεων σε σημείο που ξεπεράστηκαν ακόμα και οι αισιόδοξες προσδοκίες. Με αυτή την υλοποίηση η κάρτα Tesla K40 γενιάς Kepler επωφελείται περισσότερο αφού πλέον είναι πάνω από δύο φορές γρηγορότερη από την Tesla M2050 γενιάς Fermi. Επίσης, παρατηρείται και στις δύο κάρτες η επίπτωση που έχουν στις επιδόσεις τα bank conflicts στις προσβάσεις στη shared memory.



Σχ. 4.32: Αποτελέσματα speedup υλοποίησης Shared Memory για γράφο μεγέθους 8192×8192

Στο *Σχήμα 4.33* φαίνονται συγκεντρωμένοι οι χρόνοι των υλοποιήσεων Global Memory και Shared Memory για κάθε κάρτα γραφικών. Για να υπάρχει ένα μέτρο σύγκρισης, η καλύτερη υλοποίηση για επεξεργαστές (Tiled with OpenMP tasks, $B = 128$) στο σύστημα Sandy Bridge (με 32 threads) για τον ίδιο γράφο έχει χρόνο εκτέλεσης ίσο με *9,21 sec*.



	Global Memory	Shared Memory (Packed)	Shared Memory (Spread)
Tesla M2050 (Fermi)	22,86	8,13	7,11
Tesla K40 (Kepler)	13,61	3,59	3,15

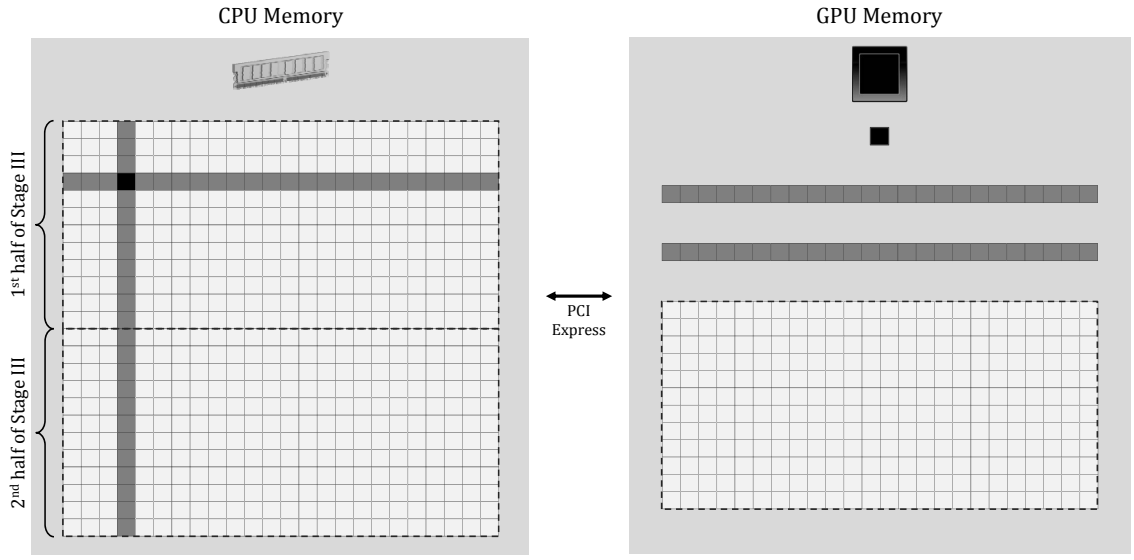
Σχ. 4.33: Χρόνοι εκτέλεσης υλοποίησης Shared Memory για γράφο μεγέθους 8192×8192

4.2.4. Shared Memory υλοποίηση χωρίς περιορισμό μνήμης κάρτας γραφικών

Λόγω της θεαματικής αύξησης στις επιδόσεις της Shared Memory υλοποίησης έναντι της Global Memory, κρίνεται σημαντική η επέκταση της Shared Memory υλοποίησης ώστε το μέγιστο μέγεθος του πίνακα να μην εξαρτάται από τη μνήμη DRAM της κάρτας γραφικών. Η τεχνική του tiling έχει ήδη χρησιμοποιηθεί ανάμεσα στην shared και στην global memory για να επιτευχθούν οι υψηλές επιδόσεις της Shared Memory υλοποίησης. Συνεπώς, δεν γίνεται να ακολουθηθεί ο ίδιος δρόμος που ακολουθήθηκε κατά την επέκταση της Global Memory υλοποίησης της *Υποενότητας 4.2.2*.

Η εναλλακτική που προκύπτει είναι σε κάθε επανάληψη k των τριών σταδίων της τεχνικής tiling να αντιγράφονται τα tiles προς επεξεργασία από την κύρια μνήμη στη μνήμη DRAM της κάρτας γραφικών και μετά να καλείται η υλοποίηση Shared Memory όπως περιγράφηκε στην προηγούμενη υποενότητα. Μετά το πέρας της εκτέλεσης, τα αποτελέσματα να αντιγράφονται πίσω στην κύρια μνήμη του συστήματος.

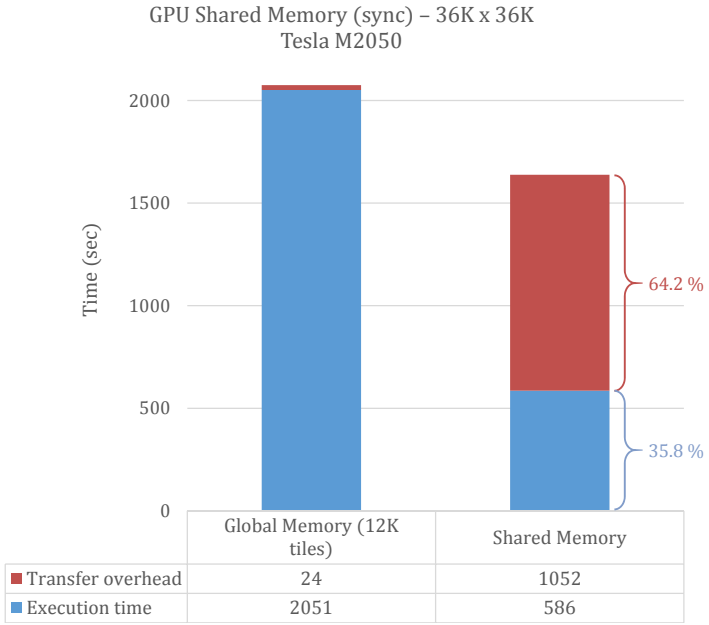
Αναλυτικότερα, αρχικά αντιγράφονται και επεξεργάζονται τα tiles του πρώτου και του δεύτερου σταδίου και επιστρέφονται τα αποτελέσματα στην κύρια μνήμη του επεξεργαστή. Στη συνέχεια, στο τρίτο στάδιο της τεχνικής, τα tiles που πρέπει να υπολογιστούν χωρίζονται σε όσα κομμάτια είναι αναγκαίο ώστε να χωράνε στη μνήμη της κάρτας γραφικών. Διαδοχικά καθένα από αυτά αντιγράφεται στην κάρτα γραφικών, γίνεται η επεξεργασία και επιστρέφεται πίσω το αποτέλεσμα. Η διαδικασία αυτή παρουσιάζεται σχηματικά στο *Σχήμα 4.34*.



Σχ. 4.34: Υλοποίηση Shared Memory χωρίς περιορισμό μνήμης κάρτας γραφικών

Το γεγονός ότι αφ' ενός η μεταφορά δεδομένων μέσω του διαύλου PCI Express είναι αργή και αφ' ετέρου το μέγεθος του tile είναι αρκετά μικρό, οδηγεί σε μεγάλο overhead κατά την ανταλλαγή των δεδομένων ανάμεσα στην κύρια μνήμη του συστήματος και στη μνήμη της κάρτας γραφικών. Στο *Σχήμα 4.35* φαίνεται ο χρόνος της συγκεκριμένης υλοποίησης στην κάρτα Tesla

M2050 για γράφο μεγέθους $36K \times 36K \times 4B = 5,1\,GB$ και για μέτρο σύγκρισης ο χρόνος της υλοποίησης Global Memory για τον ίδιο γράφο με tiles μεγέθους $12K \times 12K \times 4B = 576\,MB$.



Σχ. 4.35: Χρόνος εκτέλεσης υλοποίησης Shared Memory χωρίς περιορισμό μνήμης για γράφο μεγέθους $36K \times 36K$ στην κάρτα Tesla M2050

Όπως ήταν αναμενόμενο, το μεγαλύτερο τμήμα του χρόνου καταναλώνεται στις μεταφορές δεδομένων. Ο τελικός χρόνος εκτέλεσης να είναι σημαντικά αυξημένος, πλησιάζοντας εκείνον της αρκετά λιγότερο αποδοτικής υλοποίησης Global Memory. Συνεπώς, κρίνεται απαραίτητη η μείωση του υψηλού overhead για να εκμεταλλευτούμε στο έπακρον την ταχύτατη υλοποίηση Shared Memory.

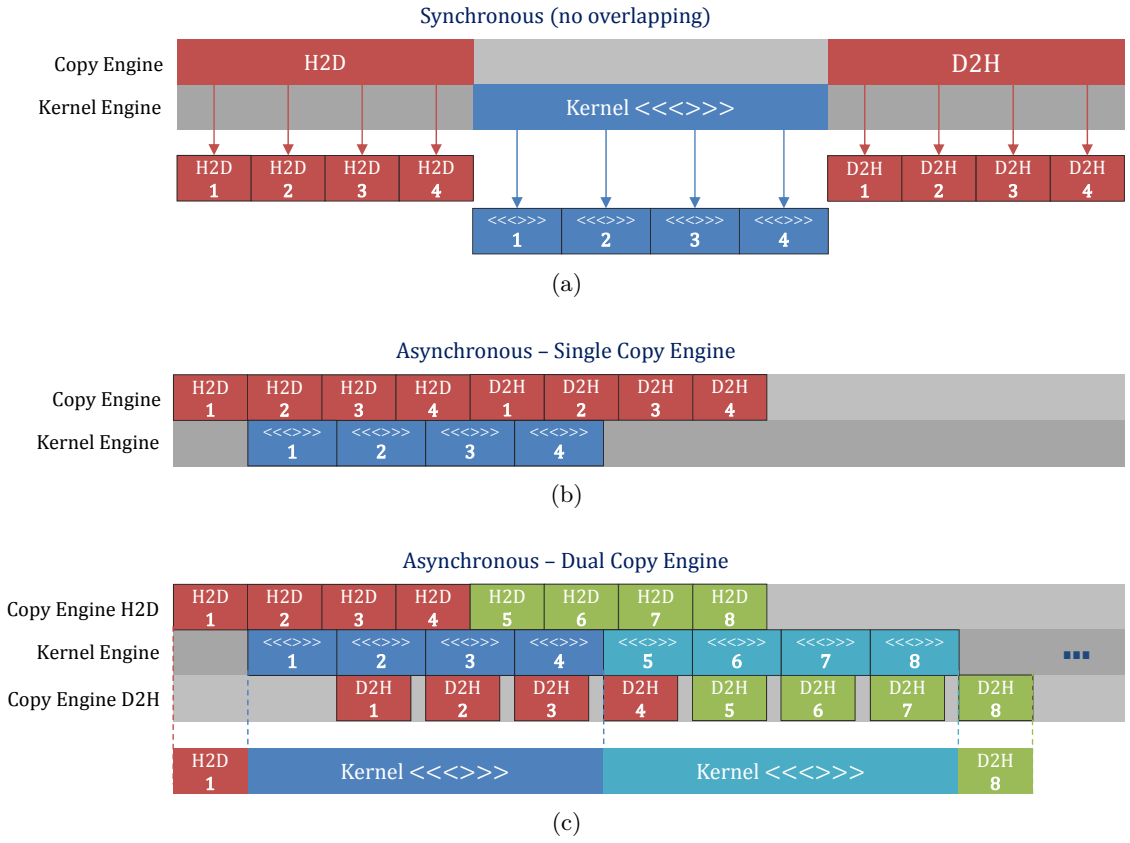
Ασύγχρονη μεταφορά δεδομένων και εκτέλεση

Η λύση στο πρόβλημα του overhead έρχεται με την ασύγχρονη εκτέλεση kernels και μεταφορά δεδομένων που υποστηρίζουν οι κάρτες γραφικών και η πλατφόρμα CUDA. Στο *Σχήμα 4.36* φαίνεται η διαδικασία που ακολουθείται σε κάθε στάδιο εκτέλεσης κάθε k επανάληψης της τεχνικής tiling, δηλαδή η αντιγραφή των δεδομένων από την κύρια μνήμη του συστήματος στην κάρτα γραφικών $(H2D)$, η επεξεργασία τους και η αντιγραφή του αποτελέσματος πίσω στην κύρια μνήμη $(D2H)$.

Σχ. 4.36: Διαδικασία εκτέλεσης κάθε σταδίου tiling της υλοποίησης Shared Memory χωρίς περιορισμό μνήμης

Για να είναι εφικτή η ασύγχρονη εκτέλεση και μεταφορά δεδομένων, πρέπει να χρησιμοποιηθούν τα λεγόμενα CUDA Streams. Ένα CUDA stream είναι μια σειρά από εντολές οι οποίες εκτελούνται στην κάρτα γραφικών. Η σειρά με την οποία γίνονται issue τα streams από τον κώδικα που εκτελείται στον επεξεργαστή είναι και η σειρά με την οποία θα ολοκληρωθεί η εκτέλεσή τους στην κάρτα γραφικών. Όμως, ενώ οι εντολές που βρίσκονται μέσα σε ένα stream είναι εγγυημένο ότι θα εκτελεστούν με τη σειρά στην οποία βρίσκονται, τα streams μπορούν να εκτελούνται ταυτόχρονα, αρκεί να γίνεται σεβαστή η σειρά προτεραιότητας με βάση την οποία έγιναν issue.

Η εκτέλεση ενός kernel ή η μεταφορά δεδομένων προς και από την κάρτα γραφικών μπορούν να αποτελέσουν ξεχωριστά streams. Συνεπώς, τα δεδομένα που πρέπει να επεξεργαστούν στην κάρτα γραφικών χωρίζονται σε μικρότερα διαδοχικά τμήματα. Έτσι, τόσο η διαδικασία μεταφοράς όσο και η εκτέλεση του kernel χωρίζονται σε αντίστοιχα τμήματα, όπως φαίνεται στο *Σχήμα 4.37a*. Κάθε τέτοιο τμήμα αποτελεί ξεχωριστό CUDA stream και γίνεται issue με τη σειρά.

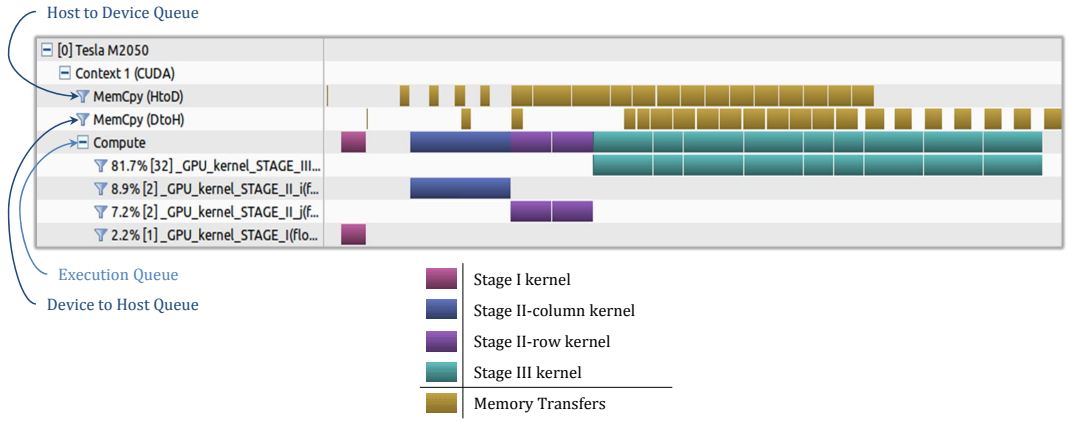


***Σχ.* 4.37**: Ασύγχρονη εκτέλεση kernels και μεταφορά δεδομένων

Το σωστό issue των streams εξασφαλίζει τη σωστή εκτέλεση με ακριβώς το ίδιο αποτέλεσμα σα να γίνονταν όλα σύγχρονα και όχι ασύγχρονα. Με δεδομένη τη σειρά με την οποία τα strems γίνονται issue, ο τρόπος με τον οποίο θα εκτελεστούν είναι ευθύνη του δρομολογητή της κάρτας γραφικών και εξαρτάται από τον αριθμό των μηχανών αντιγραφής δεδομένων (copy engine) που διαθέτει η εκάστοτε κάρτα γραφικών. Οι απλές κάρτες γραφικών της εταιρίας Nvidia διαθέτουν μόνο μία μηχανή αντιγραφής η οποία χρησιμοποιείται είτε για μεταφορά από την κύρια μνήμη προς

την κάρτα γραφικών είτε το ανάποδο, όπως φαίνεται στο *Σχήμα 4.37b*. Αντίθετα, οι επαγγελματικές κάρτες γραφικών Tesla και Quadro διαθέτουν δύο μηχανές αντιγραφής, ξεχωριστές για κάθε κατεύθυνση μεταφοράς δεδομένων, όπως φαίνεται στο *Σχήμα 4.37c*.

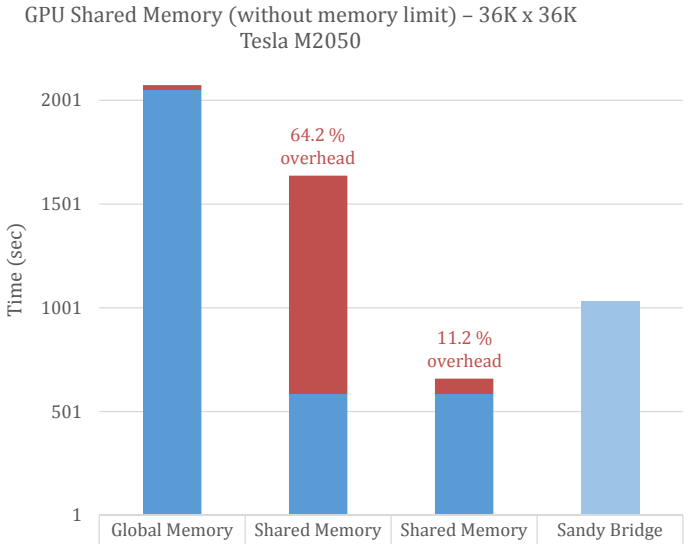
Στο *Σχήμα 4.38* φαίνεται εικόνα από το εργαλείο Nvidia Visual Profiler που αναλύει την εκτέλεση ενός προγράμματος στην κάρτα γραφικών. Φαίνεται ξεκάθαρα η ασύγχρονη εκτέλεση των kernels και των αντιγραφών δεδομένων.



Σχ. 4.38: Εικόνα ασύγχρονης εκτέλεσης από το εργαλείο Nvidia Visual Profiler

Μετρήσεις

Στο *Σχήμα 4.39* φαίνεται ο χρόνος της ασύγχρονης υλοποίησης Shared Memory στην κάρτα Tesla M2050 για τον ίδιο γράφο με προηγουμένως, διαστάσεων $36K \times 36K$. Για μέτρο σύγκρισης συμπεριλαμβάνεται ο χρόνος εκτέλεσης της καλύτερης υλοποίησης για επεξεργαστές (Tiled with OpenMP tasks, $B = 128$) στο σύστημα Sandy Bridge.

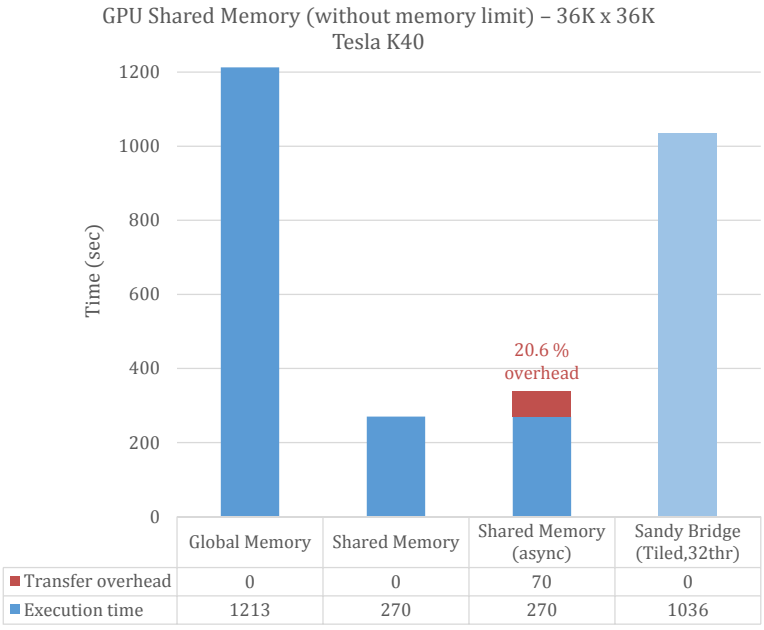


	Global Memory (12K tiles)	Shared Memory (sync)	Shared Memory (async)	Sandy Bridge (Tiled,32thr)
Transfer overhead	24	1052	74	0
Execution time	2051	586	586	1036

Σχ. 4.39: Χρόνος εκτέλεσης υλοποίησης Shared Memory χωρίς περιορισμό μνήμης για γράφο μεγέθους $36K \times 36K$ στην κάρτα Tesla M2050

Χάρη στην ασύγχρονη εκτέλεση παρατηρείται σημαντική μείωση του overhead των μεταφορών δεδομένων που πλέον ανέρχεται στο 11.2 % του συνολικού χρόνου εκτέλεσης, ποσοστό που κρίνεται αποδεκτό. Σημαντικό πλεονέκτημα της συγκεκριμένης υλοποίησης είναι ότι μπορεί εύκολα να υποστηρίξει πολλαπλές κάρτες γραφικών και με σωστό συγχρονισμό των αντιγραφών δεδομένων να παρουσιάζει πολύ καλή κλιμάκωση.

Η κάρτα Tesla K40 έχει μεγάλη μνήμη DRAM μεγέθους $12\,GB$, οπότε ο γράφος διαστάσεων $36K \times 36K \times 4B = 5,1\,GB$ χωράει να εκτελεστεί αυτούσιος. Στο *Σχήμα 4.40* φαίνονται οι χρόνοι εκτέλεσης των υλοποιήσεων Global και Shared Memory με τον πίνακα αποθηκευμένο ολόκληρο στη μνήμη της κάρτας και ο χρόνος της ασύγχρονης υλοποίησης Shared Memory, προσομοιώνοντας την περίπτωση που ο γράφος δεν χωρούσε αυτούσιος στη μνήμη.



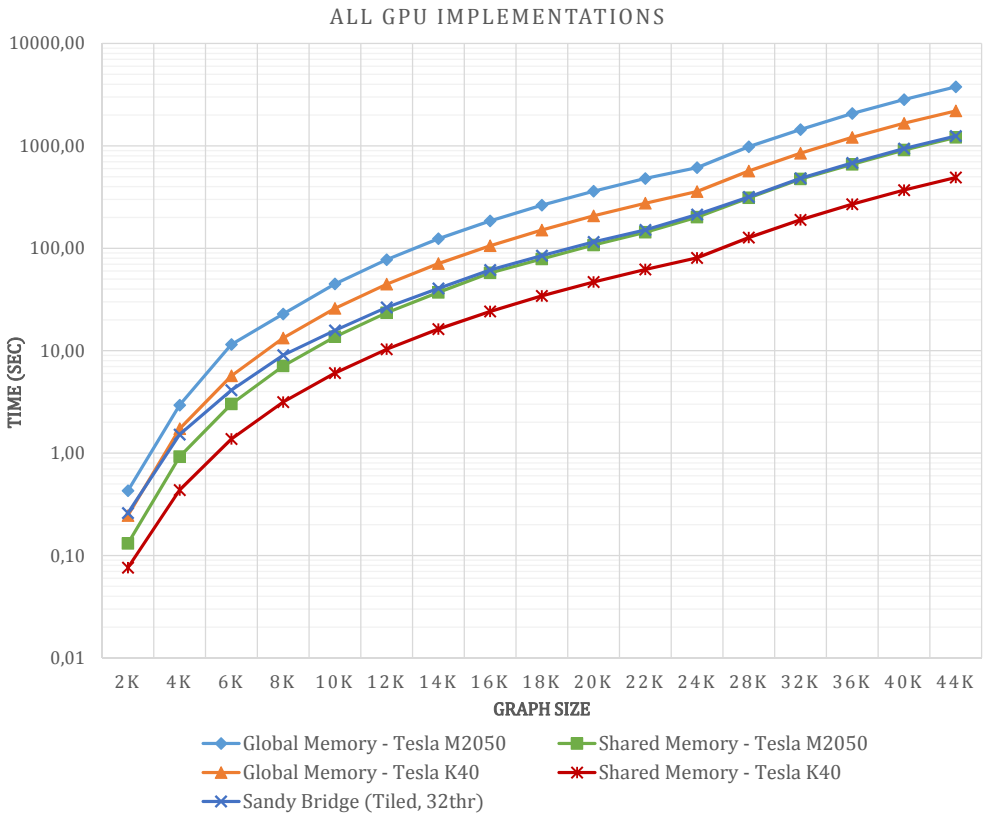
Σχ. 4.40: Χρόνος εκτέλεσης υλοποίησης Shared Memory χωρίς περιορισμό μνήμης για γράφο μεγέθους $36K \times 36K$ στην κάρτα Tesla K40

Παρατηρείται αυξημένο ποσοστό overhead σε σχέση με την Tesla M2050. Όταν η κάρτα γραφικών έχει δύο μηχανές αντιγραφής και η χρονική διάρκεια εκτέλεσης του kernel είναι μεγαλύτερη από αυτή της μίας αντιγραφής, τότε κατά τη διάρκεια συνεχόμενων εκτελέσεων όλες οι μεταφορές δεδομένων εκτός από την πρώτη και την τελευταία «κρύβονται» αφού πραγματοποιούνται ταυτόχρονα με την εκτέλεση του kernel, όπως φαίνεται στο *Σχήμα 4.37c*.

Κατά την εκτέλεση ενός σταδίου της τεχνικής tiling της υλοποίησης Shared Memory, η διάρκεια εκτέλεσης του kernel είναι μεγαλύτερη από αυτή μίας αντιγραφής στην κάρτα Tesla M2050. Αντίθετα, στην κάρτα Tesla K40 είναι μικρότερη, αφού η αύξηση της επεξεργαστικής ισχύος της είναι μεγαλύτερη από την αντίστοιχη αύξηση του εύρους ζώνης που προέκυψε από τη χρήση της έκδοσης 3.0 του διαύλου PCI Express. Για το λόγο αυτό προκύπτει το μεγαλύτερο ποσοστό του overhead.

4.2.5. Συγκεντρωτικά αποτελέσματα

Στο *Σχήμα 4.41* φαίνονται συγκεντρωμένοι οι χρόνοι εκτέλεσης για διάφορα μεγέθη γράφων των δύο υλοποιήσεων Global και Shared Memory στις δύο κάρτες γραφικών και της υλοποίησης Tiled with OpenMP tasks στο σύστημα Sandy Bridge. Επισημαίνεται ότι στο παρακάτω διάγραμμα ο κάθετος άξονας του χρόνου είναι σε λογαριθμική κλίμακα.



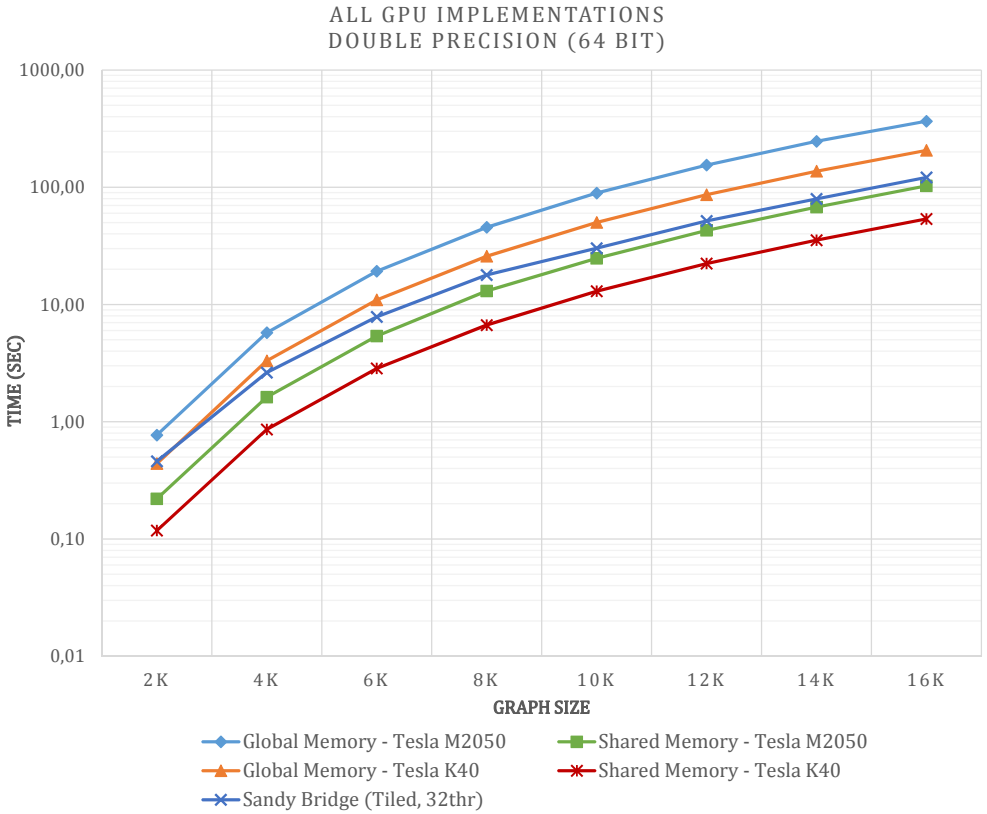
Σχ. 4.41: Χρόνοι εκτέλεσης υλοποιήσεων Global και Shared Memory

Παρατηρείται αμέσως η εξαιρετική απόδοση της Shared Memory υλοποίησης και η σημαντική υπεροχή της αρχιτεκτονικής Kepler έναντι της αρχιτεκτονικής Fermi. Για καλύτερη σύγκριση των υλοποιήσεων και των αρχιτεκτονικών, στην *Ενότητα 4.4* φαίνεται η αξιολόγηση των επιδόσεων σε σχέση με την κατανάλωση και το κόστος αγοράς.

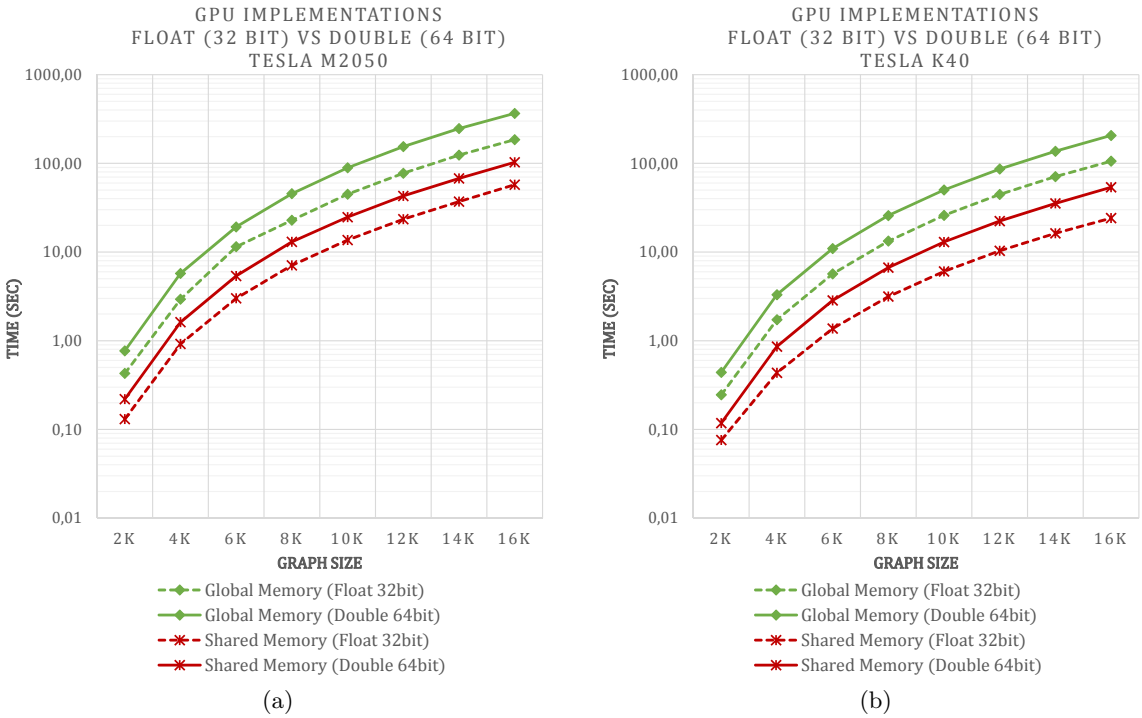
Μετρήσεις με χρήση αριθμών διπλής ακρίβειας 64 bit

Στο *Σχήμα 4.42* φαίνονται οι αντίστοιχοι χρόνοι εκτέλεσης του *Σχήματος 4.41* με χρήση αριθμών κινητής υποδιαστολής διπλής ακρίβειας (double precision 64 *bit*) για την αναπαράσταση του πίνακα γειτνίασης του γράφου.

Επίσης, για καλύτερη σύγκριση, στο *Σχήμα 4.43* φαίνονται οι χρόνοι εκτέλεσης των δύο υλοποιήσεων Global και Shared Memory με χρήση αριθμών απλής και διπλής ακρίβειας για κάθε μία από τις δύο κάρτες γραφικών.



Σχ. 4.42: Χρόνοι εκτέλεσης υλοποιήσεων Global και Shared Memory με χρήση αριθμών διπλής ακρίβειας 64 *bit*



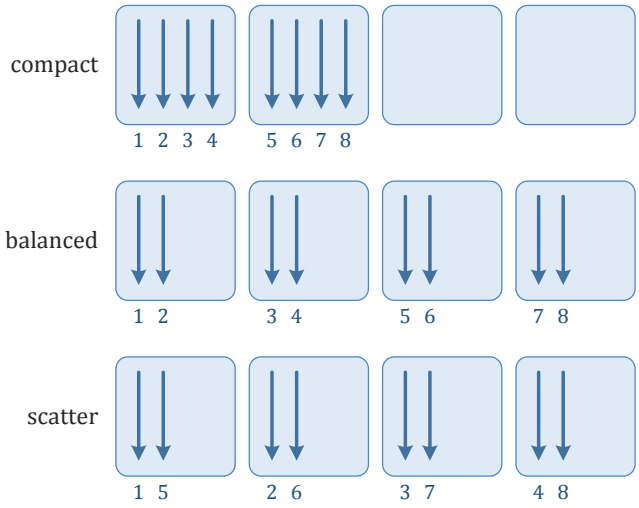
Σχ. 4.43: Σύγκριση χρόνων εκτέλεσης υλοποιήσεων Global και Shared Memory για αριθμούς απλής και διπλής ακρίβειας

4.3. Υλοποίηση στον Intel Xeon Phi

Η τρίτη και τελευταία αρχιτεκτονική η οποία εξετάζεται είναι η αρχιτεκτονική Intel MIC στην οποία βασίζεται ο συνεπεξεργαστής Intel Xeon Phi. Χρησιμοποιείται το Native Execution μοντέλο εκτέλεσης κατά το οποίο οι εφαρμογές μεταγλωττίζονται και εκτελούνται εγγενώς στη συσκευή.

Λόγω της περιορισμένης διάρκειας πρόσβασης στη συσκευή, σκοπός της παρούσας διπλωματικής εργασίας είναι η επιβεβαίωση ή μη ενός από τα βασικά επιχειρήματα προώθησης της συσκευής, δηλαδή η αποδοτική εκτέλεση στον Xeon Phi ήδη υλοποιημένων προγραμμάτων για επεξεργαστές. Θα αξιολογηθούν οι επιδόσεις των υλοποιήσεων του αλγορίθμου Floyd-Warshall που αναπτύχθηκαν για συστήματα επεξεργαστών στην *Ενότητα 4.1*, με μικρές αλλαγές ώστε να εκτελούνται πιο αποδοτικά στη συσκευή.

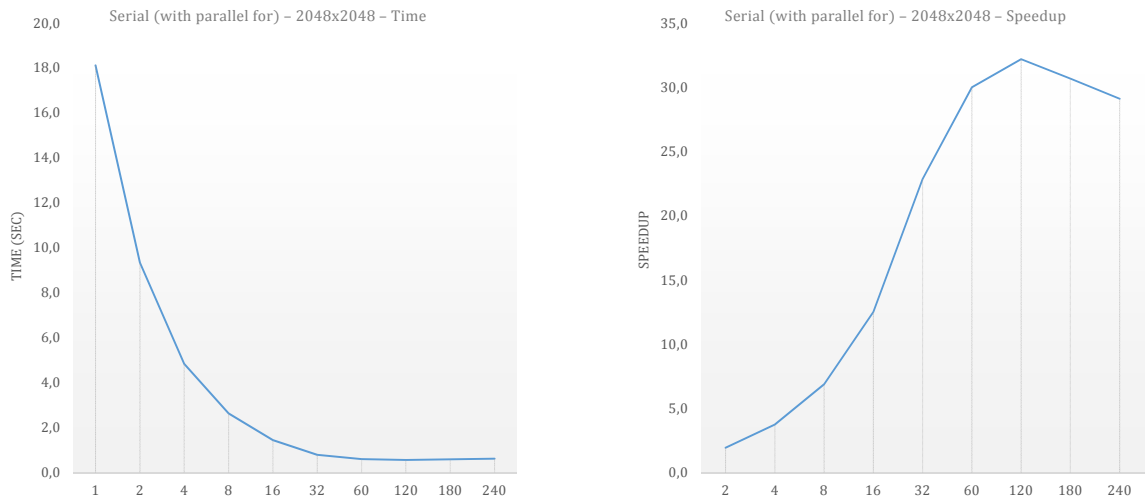
Το μέγεθος γράφου επιλέγεται ίσο με 2048×2048 κόμβους. Η συσκευή Intel Xeon Phi 5120D στην οποία υπήρχε πρόσβαση διαθέτει 60 x86 πυρήνες, οπότε ο μέγιστος αριθμός threads που υποστηρίζει είναι ίσος με 240 threads. Τέλος, σε όλες τις υλοποιήσεις εφαρμόζεται vectorization και επιλέγεται «scatter» κατανομή των threads μέσω της enviromental variable $KMP\_AFFINITY$ του μεταγλωττιστή της Intel, αφού με εκείνη προκύπτουν οι καλύτεροι χρόνοι εκτέλεσης. Οι διάφορες κατανομές που υποστηρίζονται φαίνονται στο *Σχήμα 4.44*.



Σχ. 4.44: Διαφορετικές κατανομές threads του Intel Compiler

Σειριακή υλοποίηση

Αρχικά, αξιολογείται η απλή σειριακή υλοποίηση των τριών loops της *Υποενότητας 4.1.1*, παραλληλοποιημένη ομοίως με χρήση OpenMP parallel for. Στο *Σχήμα 4.45* φαίνονται τα αποτελέσματα του χρόνου εκτέλεσης και του speedup.

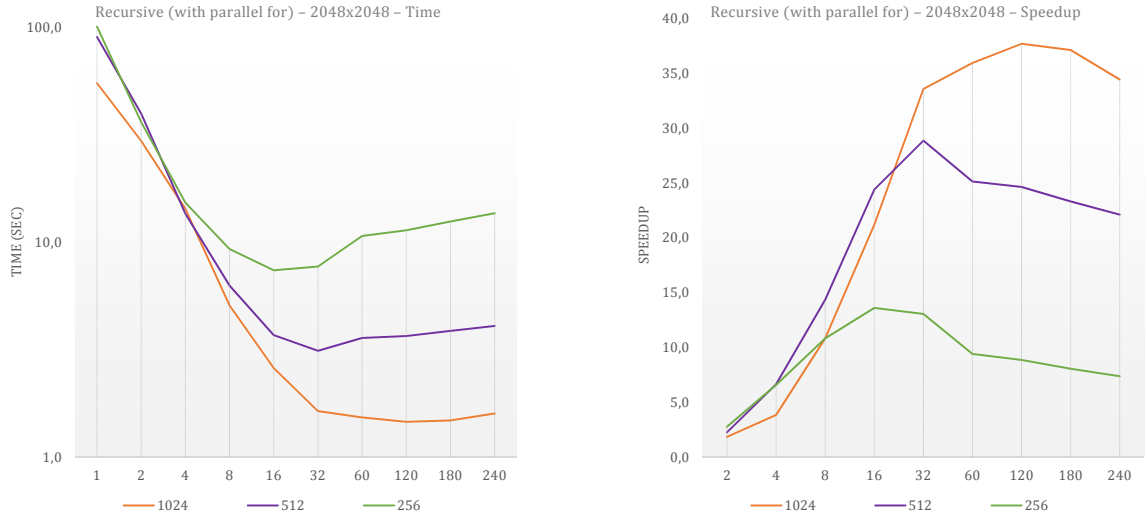


(a) (b)

Σχ. 4.45: Αποτελέσματα μετρήσεων σειριακής υλοποίησης στον Xeon Phi για γράφο μεγέθους 2048 × 2048

Αναδρομική υλοποίηση

Η αναδρομική υλοποίηση της *Υποενότητας 4.1.2* υλοποιημένη με OpenMP tasks εμφάνισε εξαιρετικά μεγάλους χρόνους εκτέλεσης, γεγονός που εξηγείται από το σημαντικό κομμάτι κώδικα που εκτελείται σειριακά. Για το λόγο αυτό, τροποποιείται η υλοποίηση με tasks και εφαρμόζεται παραλληλοποίηση με parallel for στη συνάρτηση `FWI` του *Κώδικα 4.2*. Η συγκεκριμένη υλοποίηση δοκιμάστηκε και εμφάνισε χειρότερο χρόνο εκτέλεσης στα συστήματα επεξεργαστών και συνεπώς για εκείνα προτιμήθηκε η υλοποίηση με tasks. Τα αποτελέσματα φαίνονται στο *Σχήμα 4.46*.

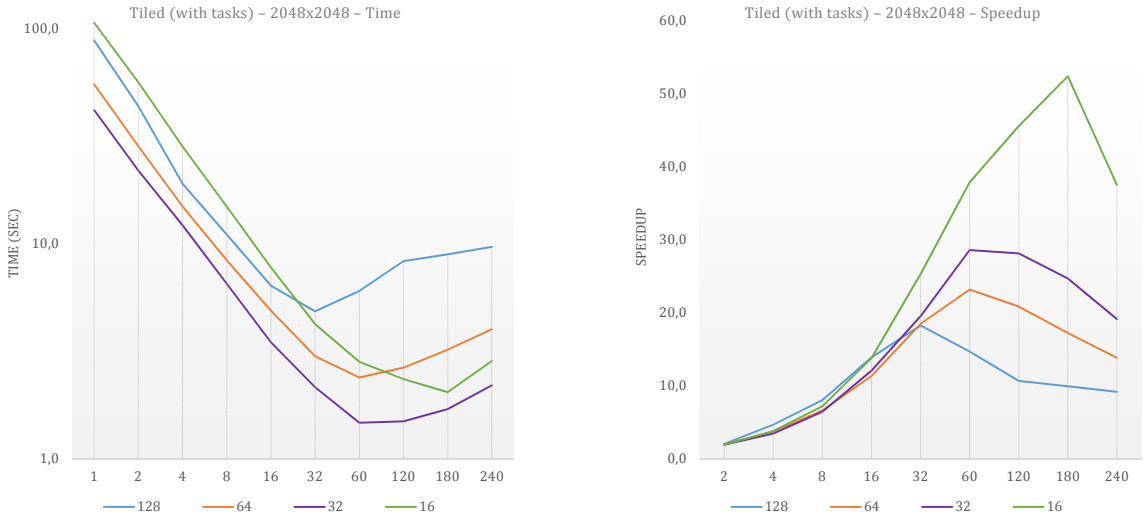


(a) (b)

Σχ. 4.46: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στον Xeon Phi για γράφο μεγέθους 2048 × 2048

Tiled υλοποίηση

Τέλος, δοκιμάστηκαν οι δύο tiled υλοποιήσεις της *Υποενότητας 4.1.3*. Τις καλύτερες επιδόσεις εμφάνισε η tiled υλοποίηση παραλληλοποιημένη με OpenMP tasks, τα αποτελέσματα της οποίας φαίνονται στο *Σχήμα 4.47*.



Σχ. 4.47: Αποτελέσματα μετρήσεων tiled υλοποίησης στον Xeon Phi για γράφο μεγέθους 2048×2048

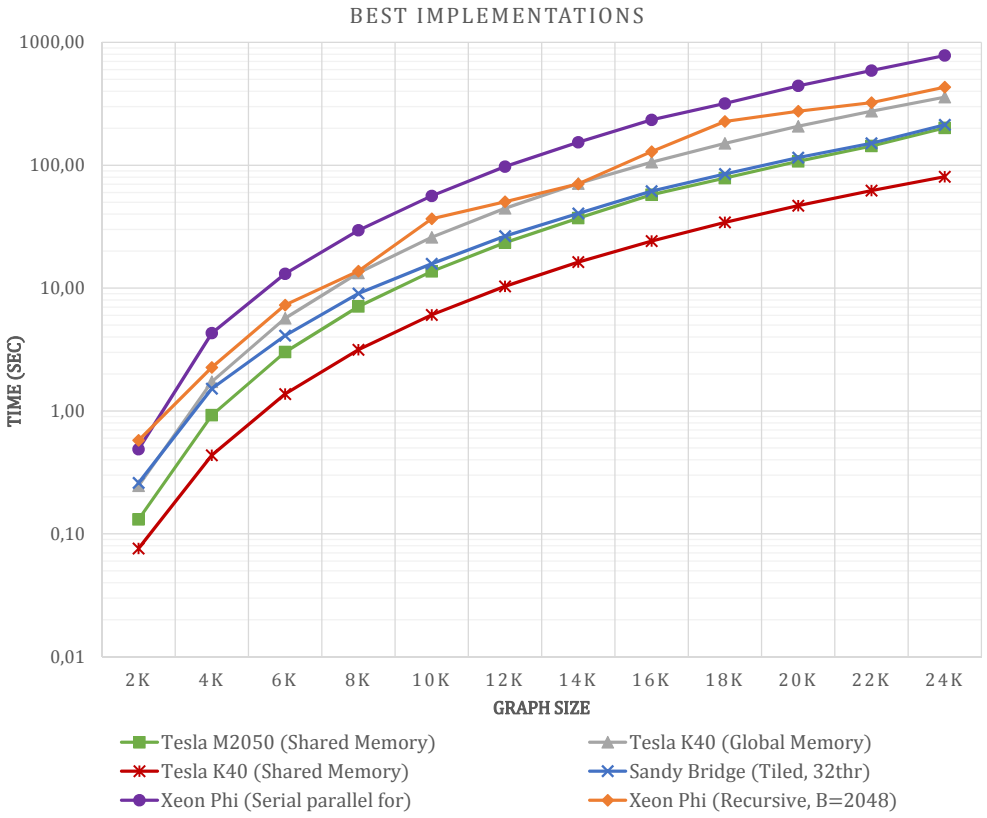
Συγκεντρωτικά αποτελέσματα

Στον *Πίνακα 4.6* φαίνονται οι καλύτεροι χρόνοι εκτέλεσης κάθε υλοποίησης που αναφέρθηκε παραπάνω.

(Graph size $2K \times 2K$)	Time
Global Memory (M2050)	0,43 *sec*
Shared Memory (M2050)	0,13 *sec*
Global Memory (K40)	0,25 *sec*
Shared Memory (K40)	0,08 *sec*
Sandy Bridge (Tiled, 32 thr)	0,26 *sec*
Xeon Phi (Serial, 120 thr)	0,56 *sec*
Xeon Phi (Recursive, 120 thr)	1,46 *sec*
Xeon Phi (Tiled, 60 thr)	1,47 *sec*

Πίν. 4.6: Χρόνοι εκτέλεσης όλων των υλοποιήσεων στον Xeon Phi για γράφο μεγέθους 2048×2048

Τέλος, στο *Σχήμα 4.48* φαίνονται οι χρόνοι εκτέλεσης των καλύτερων υλοποιήσεων από κάθε αρχιτεκτονική που έχει αναπτυχθεί στην παρούσα διπλωματική εργασία (CPU, GPU, Xeon Phi) για τον αλγόριθμο Floyd-Warshall και για διάφορα μεγέθη γράφων με χρήση αριθμών απλής ακρίβειας (32 *bit*).



Σχ. 4.48: Χρόνοι εκτέλεσης των καλύτερων υλοποιήσεων από κάθε αρχιτεκτονική του αλγορίθμου Floyd Warshall

Παρατηρείται ότι η αναδρομική υλοποίηση στον Xeon Phi με μέγεθος block $B = 2048$ εμφανίζεται εν τέλει καλύτερες επιδόσεις από την απλή υλοποίηση των 3 loops. Όμως, παραμένει σημαντικά πιο αργή από τις καλύτερες υλοποιήσεις τόσο για κάρτες γραφικών όσο και για συστήματα επεξεργαστών, με την Shared Memory υλοποίηση στην κάρτα Tesla K40 να είναι με μεγάλη διαφορά η ταχύτερη.

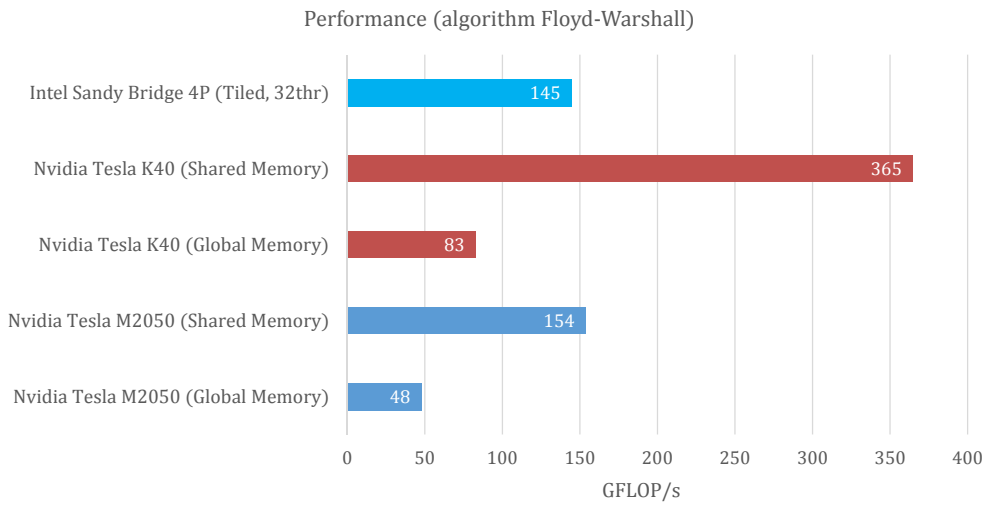
Έτσι, σύμφωνα με τα παραπάνω αποτελέσματα, δεν επιβεβαιώνεται το επιχείρημα της εταιρίας Intel για την αποδοτική εκτέλεση στον Xeon Phi ήδη γραμμένων προγραμμάτων. Φυσικά, αν αφιερωθεί ο χρόνος ώστε να γίνουν οι απαραίτητες βελτιστοποιήσεις τότε σίγουρα οι επιδόσεις στον Xeon Phi θα είναι πολύ καλύτερες. Όμως, αυτό ξεφεύγει από το σκοπό της παρούσας διπλωματικής εργασίας.

Τέλος, στο διάγραμμα συμπεριλαμβάνονται τα αποτελέσματα της Global Memory υλοποίησης στην κάρτα Tesla K40. Ο λόγος είναι η καλύτερη σύγκριση ανάμεσα στην κάρτα γραφικών και τον Xeon Phi αφού ο χρόνος ανάπτυξης της συγκεκριμένης υλοποίησης (Global Memory) βρίσκεται σε συγκρίσιμο επίπεδο με τις αντίστοιχες (serial, recursive) για τον Xeon Phi. Επίσης, τη στιγμή που γράφονται αυτές οι γραμμές, η αρχιτεκτονική Kepler είναι ο ισχυρότερος ανταγ-

ωνιστής του Xeon Phi στην αγορά του HPC. Υπό αυτή την οπτική γωνία, τα αποτελέσματα μπορούν να κριθούν (έως ένα βαθμό) συγκρίσιμα.

4.4. Συμπεράσματα

Ο αλγόριθμος Floyd-Warshall σε κάθε επανάληψη εκτελεί δύο πράξεις: μία σύγκριση και μία πρόσθεση. Συνεπώς, για μέγεθος γράφου ίσο με $N \times N$, στη χειρότερη περίπτωση εκτελούνται συνολικά $2 \cdot N^3$ πράξεις κινητής υποδιαστολής. Προκειμένου να αντληθούν χρήσιμα συμπεράσματα από τη σύγκριση των αρχιτεκτονικών επεξεργαστών και καρτών γραφικών, υπολογίζεται η επίδοση κάθε υλοποίησης σε FLOP/s (floating point operations per second) με βάση τα αποτελέσματα για γράφους μεγέθους από $8K \times 8K$ έως $24K \times 24K$ με χρήση αριθμών κινητής υποδιαστολής απλής ακρίβειας ($32\,bit$). Στο *Σχήμα 4.49* φαίνεται το διάγραμμα των επιδόσεων σε GFLOP/s των υλοποιήσεων στις δύο κάρτες γραφικών και στο καλύτερο σύστημα επεξεργαστών Intel Sandy Bridge.

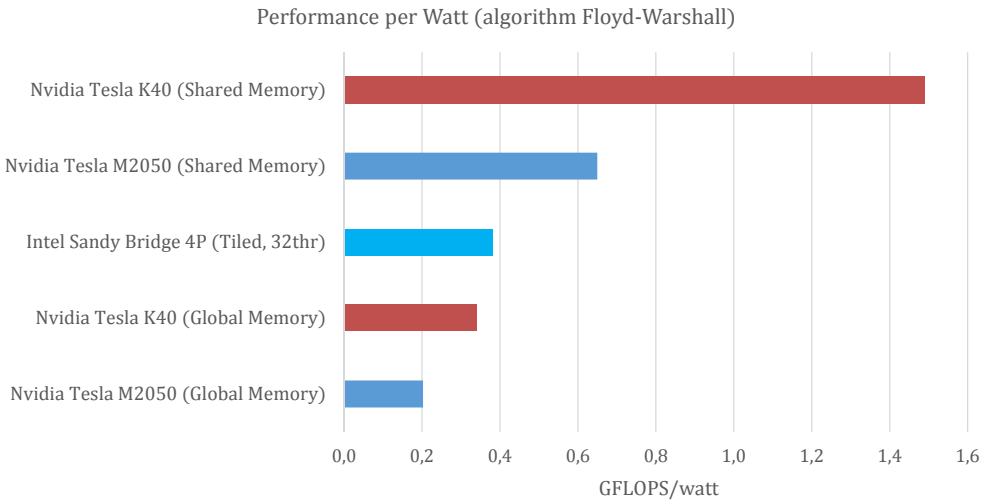


Σχ. 4.49: Επιδόσεις σε GFLOP/s (single precision $32\,bit$)

Στο παραπάνω διάγραμμα δεν συμπεριλαμβάνονται το σύστημα επεξεργαστών Dunnington λόγω παλαιότητας της πλατφόρμας και ο συνεπεξεργαστής Xeon Phi, καθώς κρίνεται άδικη η σύγκρισή του με τις υπόλοιπες αρχιτεκτονικές αφού δεν αφιερώθηκε χρόνος για τη βελτιστοποίηση των υλοποιήσεών του.

Παρατηρείται για ακόμα μια φορά η υπεροχή της Shared Memory υλοποίησης για κάρτες γραφικών. Ακόμα και η παλαιότερη γενιά καρτών Fermi κατάφερε να εμφανίσει καλύτερες επιδόσεις από το σύστημα επεξεργαστών Sandy Bridge. Όμως, η επίτευξη τέτοιων αποτελεσμάτων «κόστισε» αρκετά σε χρόνο. Για να αναπτυχθεί η συγκεκριμένη υλοποίηση απαιτήθηκε απόκτηση βαθιάς γνώσης και εξοικείωσης με το hardware και το software της πλατφόρμας Nvidia CUDA. Αντίθετα, η υλοποίηση Global Memory, αν και επίσης απαιτεί καλή γνώση της πλατφόρμας CUDA, χρειάστηκε λιγότερο χρόνο για την ανάπτυξή της. Όμως, οι επιδόσεις της είναι σημαντικά μειωμένες και αρκετά χαμηλότερες από το σύστημα Sandy Bridge.

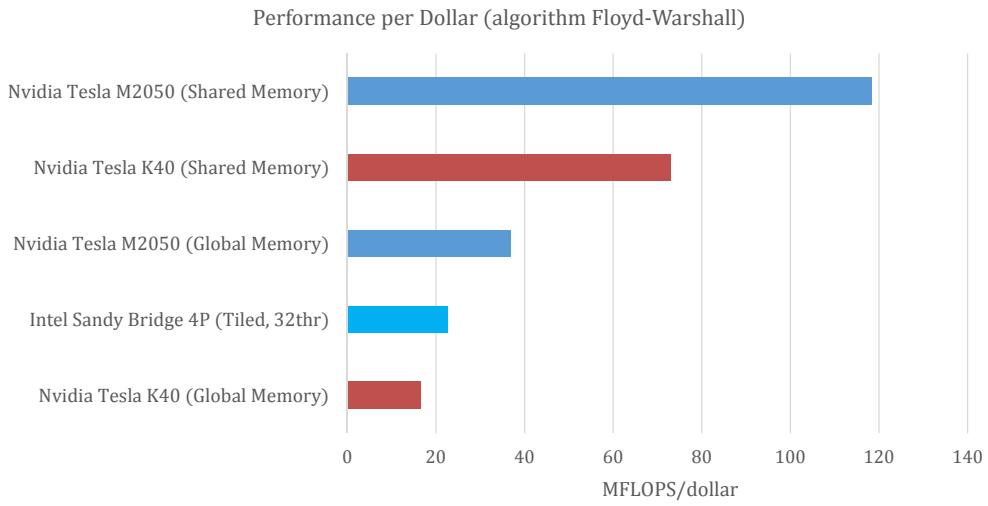
Για καλύτερη αξιολόγηση των διαφορετικών αρχιτεκτονικών, ακολουθεί στο *Σχήμα 4.50* διάγραμμα των επιδόσεων σε σχέση με την κατανάλωση. Η κατανάλωση σε Watt κάθε αρχιτεκτονικής λαμβάνεται από τους επίσημους πίνακες προδιαγραφών (όπως φαίνονται στην *Ενότητα 2.4*).



Σχ. 4.50: Επιδόσεις σε σχέση με την κατανάλωση

Τέλος, στο *Σχήμα 4.51* φαίνεται διάγραμμα των επιδόσεων σε σχέση με το κόστος αγοράς. Από το συγκεκριμένο διάγραμμα στόχος δεν είναι η ακριβής αποτύπωση της σύγκρισης ανάμεσα στις δύο αρχιτεκτονικές, αφού οι τιμές πώλησης δεν παραμένουν σταθερές και φυσικά υπάρχουν σημαντικές διαφορές ανάλογα με τον αριθμό τεμαχίων αγοράς.

Οι τιμές αγοράς των τεσσάρων επεξεργαστών Sandy Bridge λαμβάνονται από τον επίσημο τιμοκατάλογο της εταιρίας Intel. Για αγορά 1000 τεμαχίων, κάθε επεξεργαστής κοστίζει περίπου $1600, συνεπώς όλοι μαζί κοστίζουν $6400. Η κάρτα γραφικών Tesla M2050 είναι παλαιότερης γενιάς και δεν παράγεται πλέον από την εταιρία. Σαν τιμή αγοράς δεν λαμβάνεται η αυξημένη αρχική τιμή πώλησης αλλά η τωρινή τιμή πώλησης σε γνωστό κατάστημα της Αμερικής, η οποία είναι ίση με $1300. Τέλος, η τιμή αγοράς της Tesla K40 λαμβάνεται από τον μέσο όρο γνωστών καταστημάτων της Αμερικής, δηλαδή ίση με $5000.



Σχ. 4.51: Επιδόσεις σε σχέση με το κόστος αγοράς

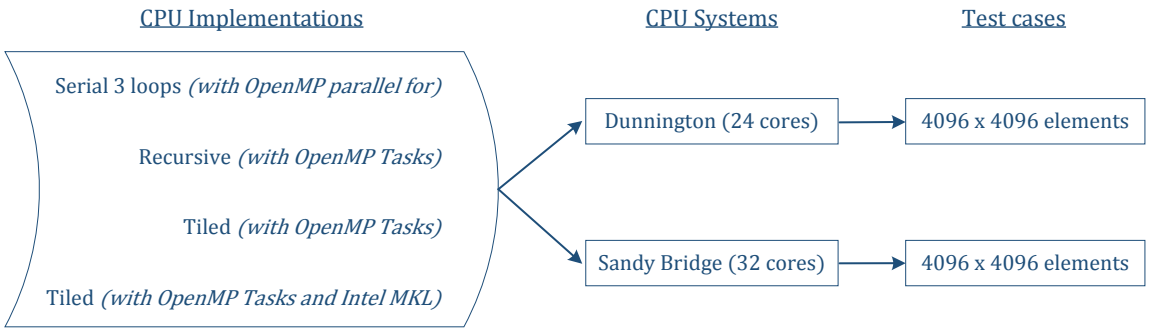
Κεφάλαιο 5

Υλοποίηση αλγορίθμου LU Decomposition

5.1. Υλοποίηση σε συστήματα πολλαπλών επεξεργαστών

Όμοια με τον αλγόριθμο Floyd-Warshall (FW), ακολουθούν τρεις διαφορετικές υλοποιήσεις για συστήματα πολλαπλών επεξεργαστών που έχουν παραλληλοποιηθεί με χρήση του OpenMP. Μεταγλωττίζονται με χρήση του Intel C Compiler 14.0 και με επίπεδο βελτιστοποιήσεων -O3. Χρησιμοποιούνται ειδικές εντολές οδηγιών προς τον μεταγλωττιστή της Intel για διανυσματοποίηση (vectorization) όπου είναι εφικτό.

Διεξάγονται μετρήσεις στα δύο συστήματα που αναφέρθηκαν στην *Υποενότητα 2.4.1* και χρησιμοποιείται πίνακας μεγέθους 4096×4096 στοιχείων (ώστε να μη χωράει ολόκληρος στη μνήμη cache), όπως φαίνεται στο *Σχήμα 5.1*. Ο πίνακας μεγέθους 2048×2048 στοιχείων εμφάνισε αντίστοιχα αποτελέσματα με αυτά του αλγορίθμου FW για γράφο μεγέθους 2048×2048 κόμβων και για τον λόγο αυτό δεν περιλαμβάνονται μετρήσεις. Τα στοιχεία του πίνακα αναπαρίστανται στη μνήμη με τύπο `double`, δηλαδή με αριθμούς κινητής υποδιαστολής διπλής ακρίβειας ($64\,bit$). Τα εκτελέσιμα που προορίζονται για το σύστημα με επεξεργαστές Dunnington μεταγλωττίζονται με υποστήριξη των SIMD εντολών SSE 4.1 ενώ εκείνα που προορίζονται για το σύστημα με επεξεργαστές Sandy Bridge με υποστήριξη των SIMD εντολών AVX.



Σχ. 5.1: Παράλληλες υλοποιήσεις αλγορίθμου LU Decomposition για συστήματα επεξεργαστών

5.1.1. Σειριακή υλοποίηση

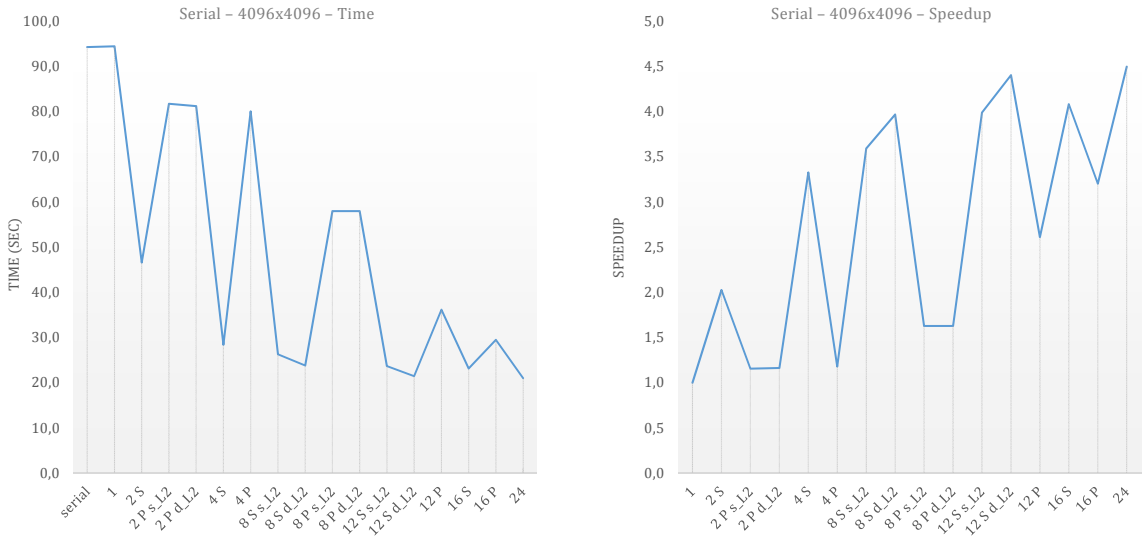
Αρχικά, παραλληλοποιείται η απλή σειριακή υλοποίηση του αλγορίθμου, όπως φαίνεται στον *Κώδικα 5.1*. Με κατάλληλη οδηγία προς τον μεταγλωττιστή γίνεται vectorization των εσωτερικών loop του κώδικα.

Κώδικας 5.1 Απλή σειριακή υλοποίηση αλγορίθμου LU Decomposition

```
#pragma omp parallel firstprivate(k)
for (k=0;k<N-1;k++)
{
    #pragma omp for
    for (i=k+1;i<N;i++)
        A[i][k] = A[i][k]/A[k][k];

    #pragma omp for private(j)
    for (i=k+1;i<N;i++)
        for (j=k+1;j<N;j++)
            A[i][j]-=A[i][k]*A[k][j];
}
```

Αποτελέσματα Dunnington

Στο *Σχήμα 5.2* φαίνονται τα διαγράμματα χρόνου και speedup των μετρήσεων στο σύστημα Dunnington για τον πίνακα διαστάσεων 4096×4096 για τις διάφορες κατανομές που περιγράφηκαν στην *Ενότητα 4.1*. Ο πίνακας καταλαμβάνει χώρο στη μνήμη ίσο με $4096 \times 4096 \times 8B = 128MB$.

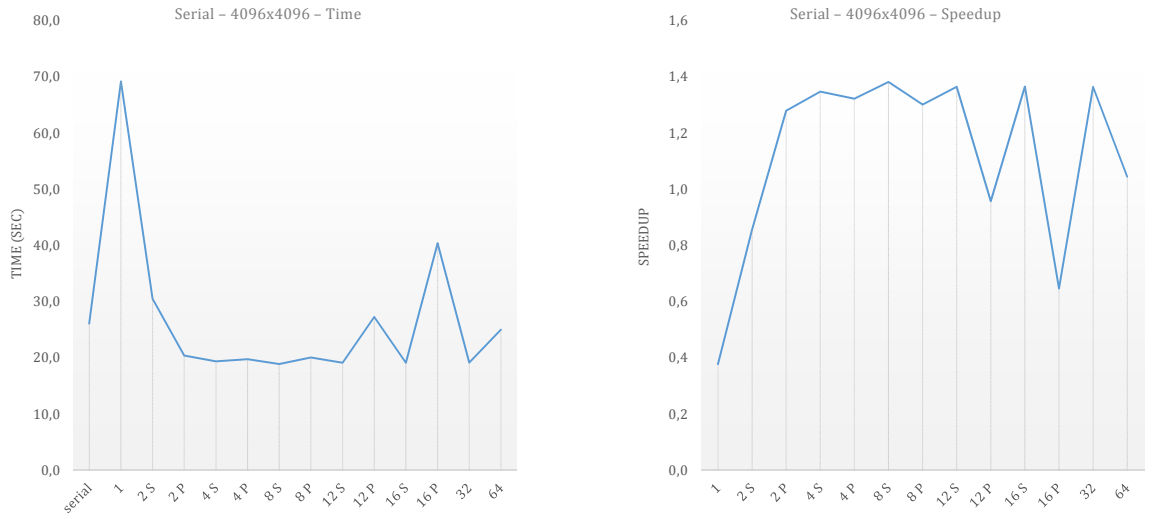


(a) (b)

Σχ. 5.2: Αποτελέσματα μετρήσεων απλής σειριακής υλοποίησης LU Decomposition στο σύστημα Dunnington

Παρατηρείται σχεδόν η ίδια εικόνα με εκείνη του αλγορίθμου FW λόγω της ίδιας φύσης του αλγορίθμου LU Decomposition, δηλαδή των χωρίς χωρική τοπικότητα προσβάσεων στη μνήμη. Αρχικά, κακή κλιμάκωση του αλγορίθμου όσο αυξάνεται ο αριθμός των threads. Στις packed κατανομές οι επιδόσεις είναι σημαντικά χειρότερες από τις αντίστοιχες spread επειδή η συνολικά διαθέσιμη κρυφή μνήμη είναι αρκετά μικρότερη. Τέλος, στις κατανομές different_L2 οι επιδόσεις είναι καλύτερες από τις αντίστοιχες same_L2 λόγω του ίδιου φαινομένου.

Αποτελέσματα Sandy Bridge



(a) (b)

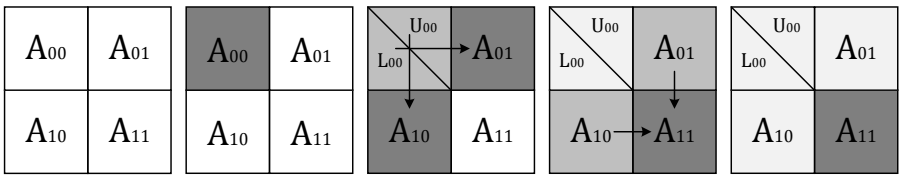
Σχ. 5.3: Αποτελέσματα μετρήσεων απλής σειριακής υλοποίησης LU Decomposition στο σύστημα Sandy Bridge

Όπως ήταν αναμενόμενο, η εικόνα που παρατηρείται στο σύστημα Sandy Bridge είναι η εξαιρετικά κακή κλιμάκωση του αλγορίθμου για τους λόγους που έχουν ήδη εξηγηθεί στην αντίστοιχη παράγραφο της *Υποενότητας 4.1.1*

5.1.2. Αναδρομική υλοποίηση

Η memory bounded φύση του αλγορίθμου και ο μη ευνοϊκός (για την επαναχρησιμοποίηση δεδομένων) τρόπος πρόσβασης στα στοιχεία του πίνακα σε κάθε επανάληψη εξηγεί το γεγονός της κακής κλιμάκωσης του αλγορίθμου σε αρχιτεκτονικές μοιραζόμενης μνήμης. Συνεπώς, ομοίως με τον αλγόριθμο Floyd-Warshall, προτάθηκαν οι ίδιες δύο εναλλακτικές μέθοδοι υλοποίησης για καλύτερη εκμετάλλευση της κρυφής μνήμης, δηλαδή η αναδρομική και η tiled.

Στο *Σχήμα 5.4* φαίνεται ο τρόπος με τον οποίο χωρίζεται ο πίνακας σε κάθε επίπεδο της αναδρομής (με βάση μία δεδομένη τιμή μεγέθους του block) και η σειρά με την οποία γίνονται οι αναδρομικές κλήσεις για την επεξεργασία κάθε υποπίνακα. Στον *Κώδικα 5.2* φαίνεται ο πηγαίος κώδικας της αναδρομικής υλοποίησης παραλληλοποιημένης με τη χρήση OpenMP tasks σε όλα τα σημεία όπου αυτό είναι εφικτό.



Σχ. 5.4: Αναδρομική υλοποίηση LU Decomposition

Αναλυτικότερα:

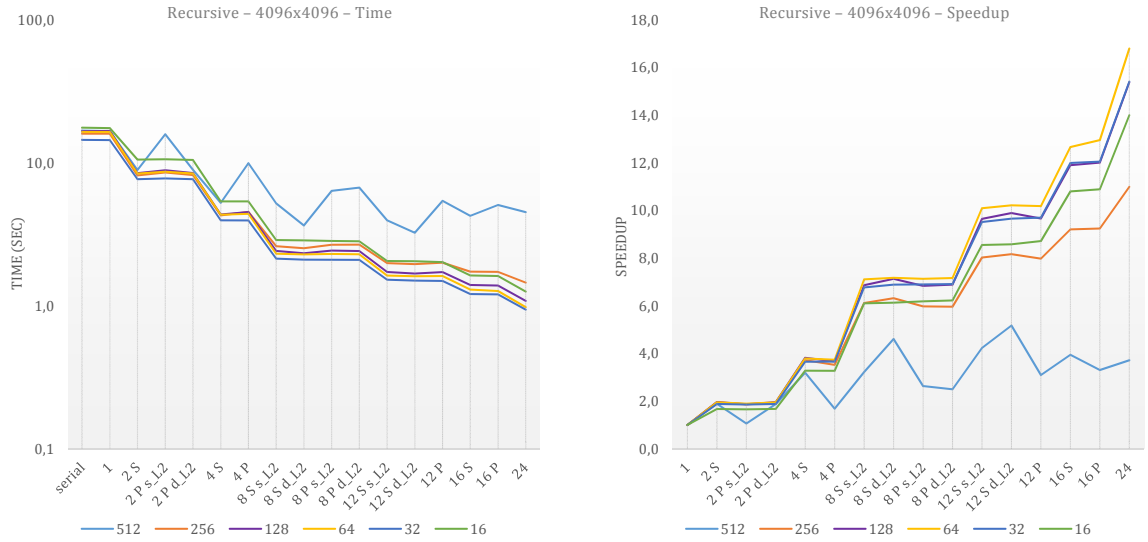
- Αναδρομικός υπολογισμός της παραγοντοποίησης LU του υποπίνακα A_{00}.
- Επίλυση του συστήματος $L_{00}A'_{01} = A_{01}$ ώστε να υπολογιστεί ο υποπίνακας A'_{01}. Η υλοποίηση της επίλυσης (`lower_solve()`) είναι υλοποιημένη αναδρομικά και παραλληλοποιημένη με χρήση tasks.
- Ομοίως, επίλυση του συστήματος $A'_{10}U_{00} = A_{10}$.
- Υπολογισμός του συμπληρώματος schur (schur complement) $A'_{11} = A_{11} - A'_{10}A'_{01}$. Η υλοποίηση της συνάρτησης υπολογισμού (`schur()`) είναι επίσης υλοποιημένη αναδρομικά και παραλληλοποιημένη με χρήση tasks.
- Τέλος, αναδρομικός υπολογισμός της παραγοντοποίησης LU του υποπίνακα A'_{11}.

Κώδικας 5.2 Αναδρομική (recursive) υλοποίηση αλγορίθμου LU Decomposition με OpenMP tasks

```
LU_recursive(A):
if (base case)
    LU_kernel(A);
else
{
    LU_recursive(A00);

    #pragma omp task
    lower_solve(A01,A00);
    #pragma omp task
    upper_solve(A10,A00);
    #pragma omp taskwait

    schur(A11,A10,A01);

    LU_recursive(A11);
}

schur(A,V,W):
if (base case)
    block_schur(A,V,W);
else
{
    #pragma omp task
    schur(A00,V00,W00);
    #pragma omp task
    schur(A01,V00,W01);
    #pragma omp task
    schur(A10,V10,W00);
    #pragma omp task
    schur(A11,V10,W01);
    #pragma omp taskwait

    #pragma omp task
    schur(A00,V01,W10);
    #pragma omp task
    schur(A01,V01,W11);
    #pragma omp task
    schur(A10,V11,W10);
    #pragma omp task
    schur(A11,V11,W11);
    #pragma omp taskwait
}
```

Αποτελέσματα Dunnington

Στο *Σχήμα 5.5* φαίνονται τα αποτελέσματα των μετρήσεων της αναδρομικής υλοποίησης στο σύστημα Dunnington για πίνακα διαστάσεων 4096×4096. Λαμβάνονται μετρήσεις για διάφορες τιμές μεγέθους του block που κυμαίνονται από 512×512 στοιχεία έως 16×16 στοιχεία.



(a) (b)

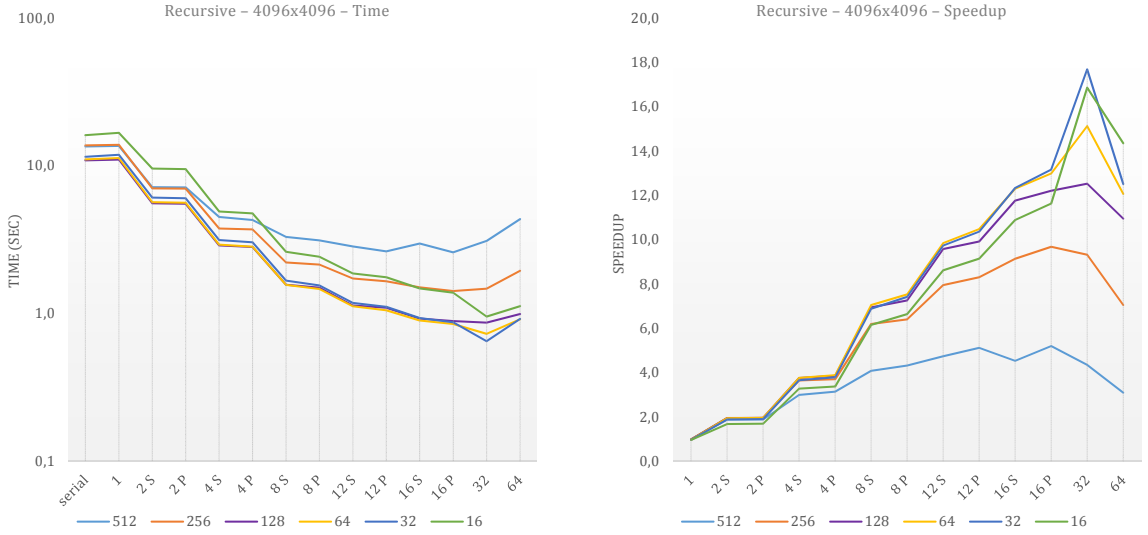
Σχ. 5.5: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στο σύστημα Dunnington

Όμοια με τον αλγόριθμο FW, παρατηρείται σημαντική βελτίωση των χρόνων σε σχέση με την απλή υλοποίηση των τριών loops, τόσο στη σειριακή όσο και στις παράλληλες εκτελέσεις. Όμως, λόγω του ότι είναι εφικτή η παραλληλοποίηση κάθε σταδίου εκτέλεσης των αναδρομικών κλήσεων, η αναδρομική υλοποίηση του αλγορίθμου LU Decomposition εμφανίζει πολύ καλή κλιμάκωση, σε αντίθεση με την αντίστοιχη υλοποίηση του FW.

Τα καλύτερα αποτελέσματα προκύπτουν για μεγέθη block ίσα με $32 \times 32 \times 8B = 8\,KB$ και $64 \times 64 \times 8B = 32\,KB$, γεγονός αναμενόμενο αν ληφθεί υπ' όψιν ότι η κρυφή μνήμη $L1$ έχει μέγεθος $32\,KB$.

Αποτελέσματα Sandy Bridge

Αντίστοιχα με προηγουμένως, στο *Σχήμα 5.6* φαίνονται τα αποτελέσματα των μετρήσεων της αναδρομικής υλοποίησης στο σύστημα Sandy Bridge.



(a) (b)

Σχ. 5.6: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στο σύστημα Sandy Bridge

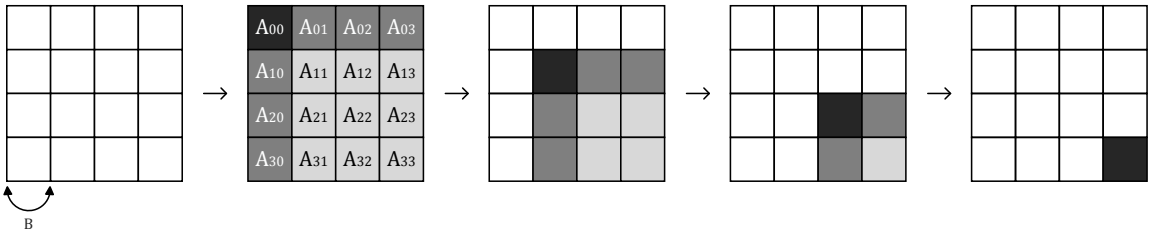
Και σε αυτά τα αποτελέσματα παρατηρείται η ίδια συμπεριφορά που αναλύθηκε προηγουμένως για το σύστημα Dunnington, με τα καλύτερα αποτελέσματα χρόνου να προκύπτουν για τα ίδια μεγέθη block, με το block μεγέθους 32×32 να εμφανίζει λίγο καλύτερο χρόνο για 32 threads.

5.1.3. Tiled υλοποίηση

Η δεύτερη εναλλακτική μέθοδος υλοποίησης για καλύτερη εκμετάλλευση της κρυφής μνήμης είναι γνωστή πλέον τεχνική του tiling.

Στο *Σχήμα 5.7* φαίνεται ο τρόπος με τον οποίο χωρίζεται ο πίνακας σε tiles μεγέθους $B \times B$ και οι διαδοχικές εκτελέσεις του αλγορίθμου για κάθε επανάληψη. Στην $k-$οστή επανάληψη:

- Αρχικά, υπολογίζεται η παραγοντοποίηση LU του διαγώνιου (k,k) tile (μαύρο του *Σχήματος 5.7*). Στη συνέχεια υπολογίζονται οι αντίστροφοι πίνακες L_{kk}^{-1} και U_{kk}^{-1}.
- Στη συνέχεια, ενημερώνονται τα tiles που βρίσκονται στην $k-$οστή γραμμή και στην $k-$οστή στήλη (σκούρο γκρι) ως εξής: $A_{ki} = L_{kk}^{-1} A_{ki}$ και $A_{ik} = A_{ik} U_{kk}^{-1}$. Δηλαδή, υπάρχει εξάρτηση δεδομένων από το αρχικό διαγώνιο tile που έχει ήδη ενημερωθεί. Οι ενημερώσεις μπορούν να γίνουν παράλληλα και παραλληλοποιούνται με χρήση OpenMP tasks.
- Τέλος, ενημερώνονται τα υπόλοιπα tiles του πίνακα (ανοιχτό γκρι) ως εξής: $A_{ij} = A_{ij} - A_{ik} A_{kj}$. Ομοίως με προηγουμένως, οι ενημερώσεις μπορούν να γίνουν παράλληλα μεταξύ τους και παραλληλοποιούνται με χρήση OpenMP tasks.



Σχ. 5.7: Εφαρμογή τεχνικής tiling στον αλγόριθμο LU Decomposition

Στον *Κώδικα 5.2* φαίνεται ο πηγαίος κώδικας της tiled παραλληλοποιημένης υλοποίησης του αλγορίθμου που περιγράφηκε παραπάνω.

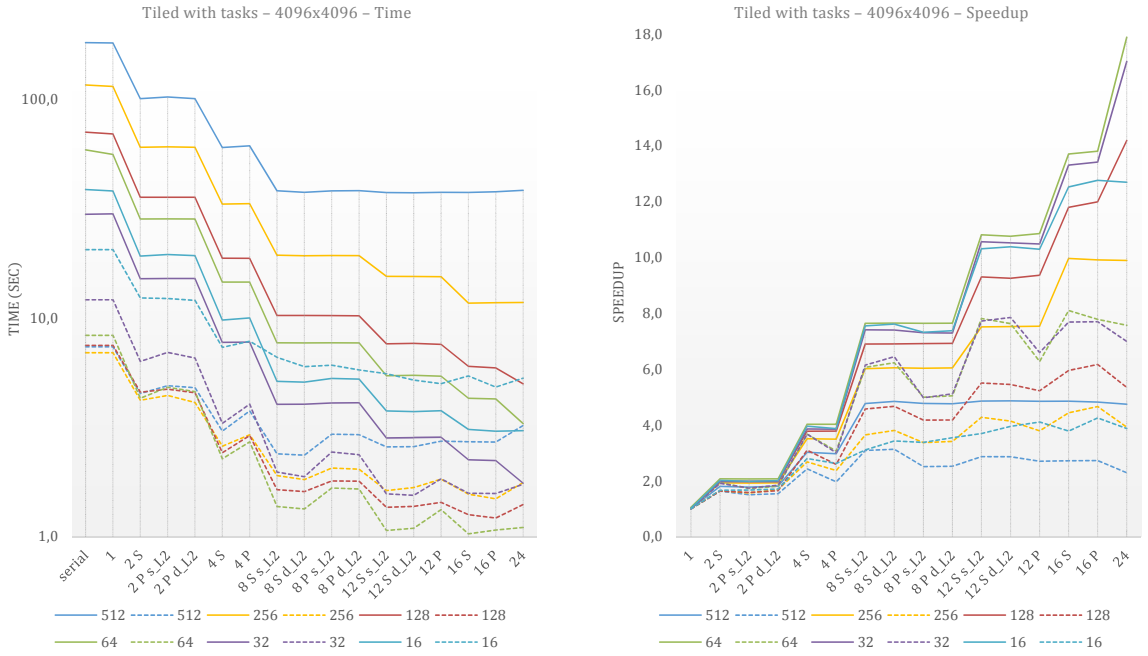
Κώδικας 5.3 Tiled υλοποίηση αλγορίθμου LU Decomposition παραλληλοποιημένη με OpenMP tasks

```
lu_tiled(A):

range=N/B;

for (k=0;k<range-1;k++)
{
    lu_kernel(A_kk);

    l_inv=get_inv_l(A_kk);
    u_inv=get_inv_u(A_kk);

    for (i=k+1;i<range;i++)
    {
        #pragma omp task firstprivate(i)
        mm_lower(l_inv, A_ki, A_ki);
        #pragma omp task firstprivate(i)
        mm_upper(A_ik, u_inv, A_ik);
        #pragma omp taskwait
    }

    for (i=k+1;i<range;i++)
        #pragma omp task firstprivate(i)
        for (j=k+1;j<range;j++)
            mm(A_ik, A_ki, A_ij);

    #pragma omp taskwait
}

lu_kernel(A_{range-1,range-1});
```

Αποτελέσματα Dunnington

Στο *Σχήμα 5.8* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης στο σύστημα Dunnington για πίνακα διαστάσεων 4096×4096. Λαμβάνονται μετρήσεις για διάφορες τιμές μεγέθους του block που κυμαίνονται από 512×512 στοιχεία έως 16×16 στοιχεία και για δύο διαφορετικές υλοποιήσεις ανάλογα με τον τρόπο που έχουν παραλληλοποιηθεί οι πολλαπλασιασμοί πινάκων. Οι συνεχόμενες γραμμές αφορούν την υλοποίηση όπου παραλληλοποιούνται με OpenMP tasks. Αντίθετα, οι διακεκομμένες αφορούν την υλοποίηση όπου οι πολλαπλασιασμοί

πινάκων γίνονται με την πολύ αποδοτική συνάρτηση `cblas_dgemm()` της βιβλιοθήκης Intel MKL (Math Kernel Library).



Σχ. 5.8: Αποτελέσματα μετρήσεων tiled υλοποίησης στο σύστημα Dunnington

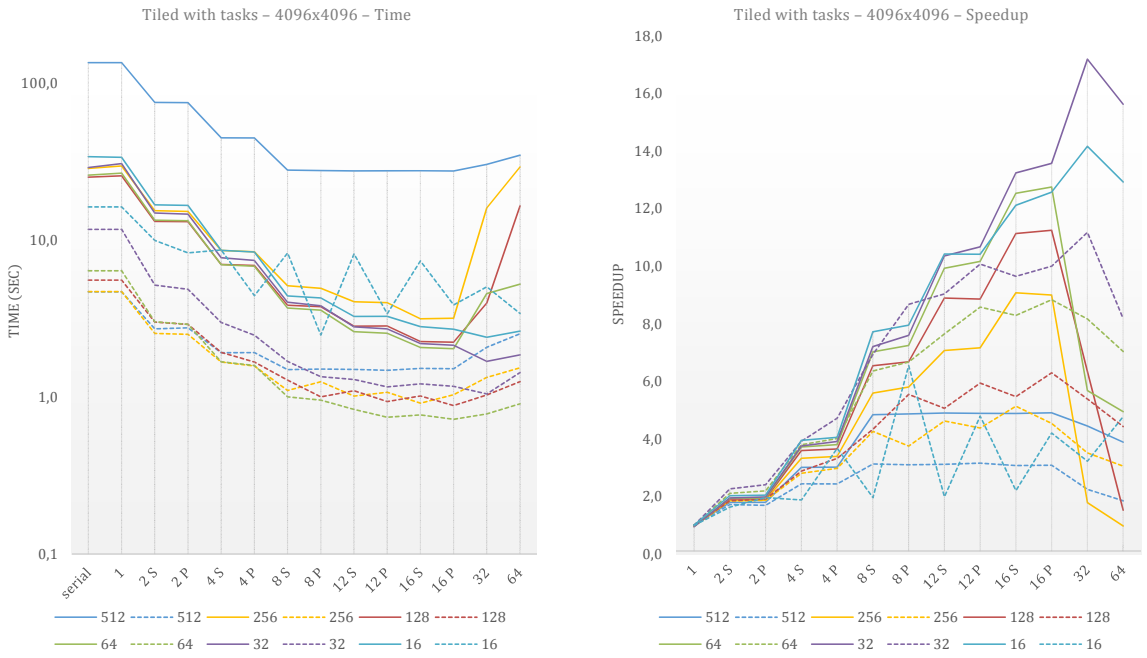
Παρατηρείται αμέσως ότι η υλοποίηση με χρήση της βιβλιοθήκης MKL είναι σημαντικά πιο γρήγορη από την αντίστοιχη όπου οι πολλαπλασιασμοί πινάκων παραλληλοποιούνται με χρήση tasks.

Οι χρόνοι εκτέλεσης των spread κατανομών είναι αρκετά γρηγορότεροι από τους χρόνους των αντίστοιχων packed, λόγω μεγαλύτερης συνολικής μνήμης cache και περισσότερου διαθέσιμου εύρους ζώνης προς τη μνήμη. Χαρακτηριστικό είναι ότι η spread κατανομή των 16 threads είναι γρηγορότερη των 24 threads.

Τέλος, λόγω του ήδη υψηλού επιπέδου βελτιστοποιήσεων των συναρτήσεων της συγκεκριμένης βιβλιοθήκης, η κλιμάκωση δεν είναι μεγάλη. Αντίθετα, στην υλοποίηση των πολλαπλασιασμών με tasks παρατηρείται σημαντικά καλύτερη κλιμάκωση, γεγονός αναμενόμενο.

Αποτελέσματα Sandy Bridge

Στο *Σχήμα 5.9* φαίνονται τα αποτελέσματα των μετρήσεων της tiled υλοποίησης στο σύστημα Sandy Bridge για τις δύο διαφορετικές παραλληλοποιήσεις των πολλαπλασιασμών πινάκων που αναφέρθηκαν προηγουμένως.

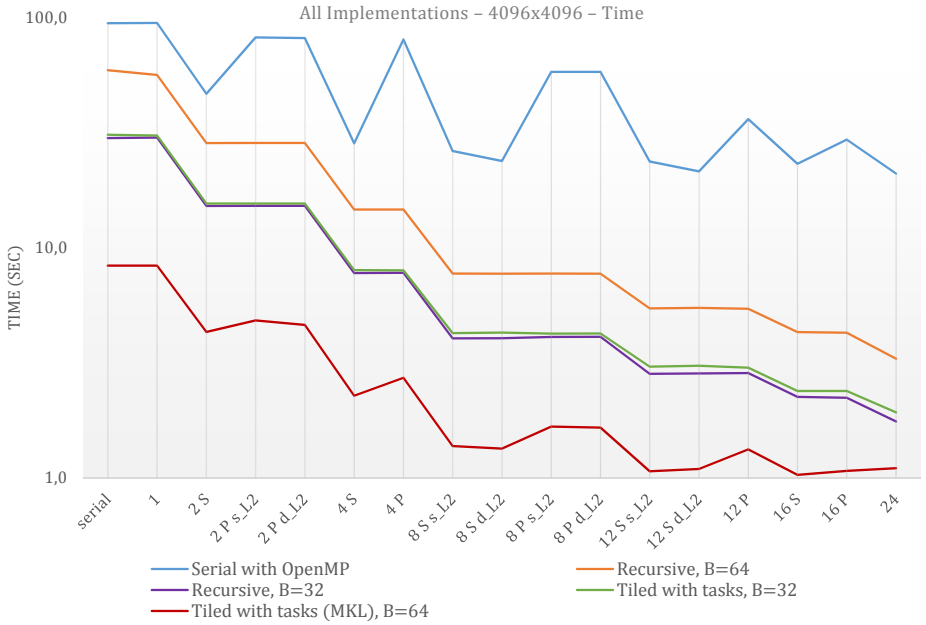


Σχ. 5.9: Αποτελέσματα μετρήσεων tiled υλοποίησης στο σύστημα Sandy Bridge

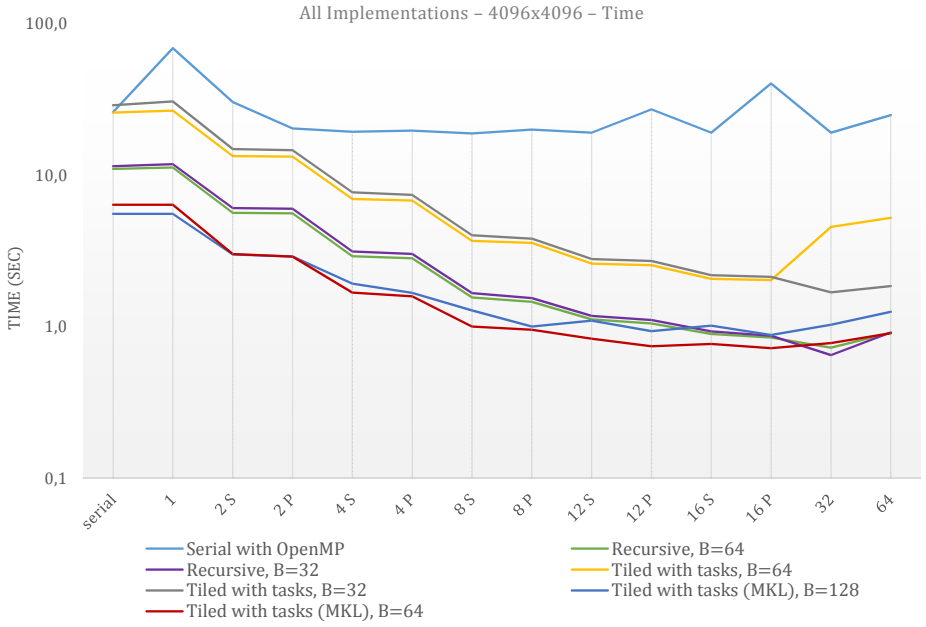
Αντίστοιχη εικόνα με προηγουμένως παρατηρείται και στα αποτελέσματα του συστήματος Sandy Bridge. Στην υλοποίηση με χρήση της βιβλιοθήκης MKL ανάλογα με το μέγεθος του block εμφανίζονται διαφορετικές προτιμήσεις ανάμεσα σε spread και packed κατανομές, ανάλογα με την προτίμηση σε μεγαλύτερο συνολικό μέγεθος cache και περισσότερο bandwidth ή μικρότερο συνολικό μέγεθος cache και επαναχρησιμοποίησης των tiles λόγω μοιραζόμενης μνήμης cache.

5.1.4. Συγκεντρωτικά αποτελέσματα

Στα *Σχήματα 5.10* και *5.11* φαίνονται συγκεντρωμένα οι μετρήσεις από όλες τις υλοποιήσεις που εξετάστηκαν στις προηγούμενες υποενότητες στα δύο συστήματα. Για όσες υλοποιήσεις περιέχουν blocking επιλέγονται τα μεγέθη block με τους καλύτερους χρόνους.

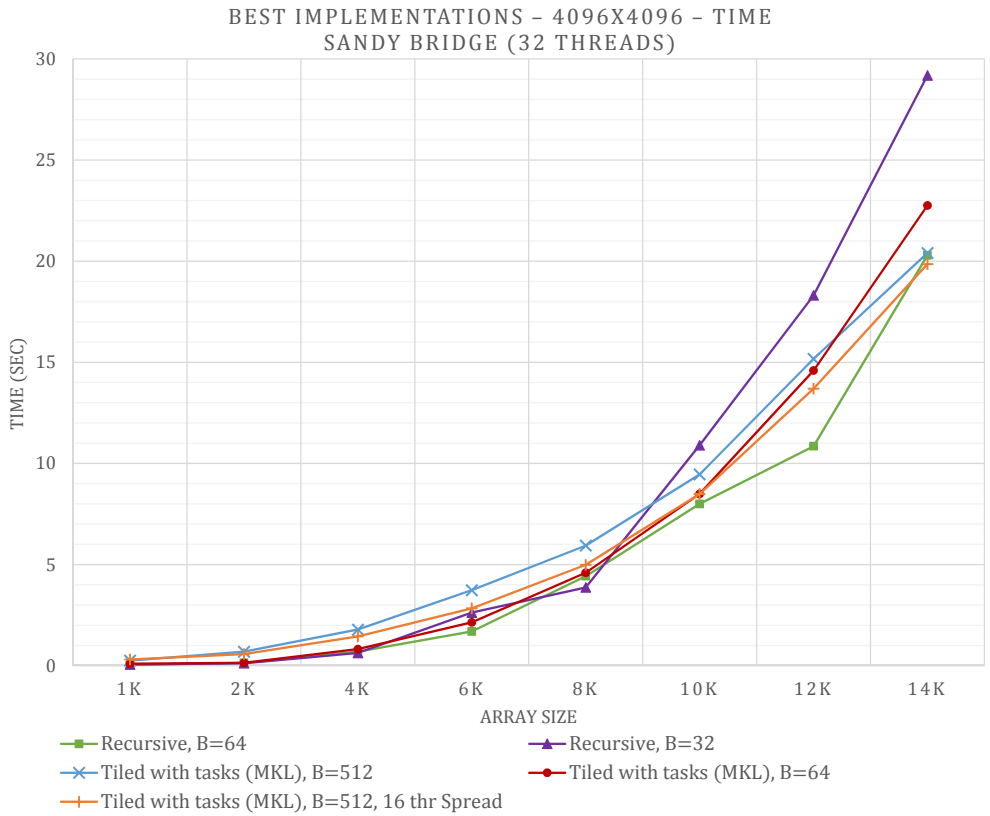


Σχ. 5.10: Συγκεντρωτικές μετρήσεις για τον πίνακα των 4096 × 4096 στοιχείων στο σύστημα Dunnington



Σχ. 5.11: Συγκεντρωτικές μετρήσεις για τον πίνακα των 4096 × 4096 στοιχείων στο σύστημα Sandy Bridge

Στο σύστημα Dunnington, η tiled υλοποίηση με χρήση της βιβλιοθήκης MKL εμφανίζει ξεκάθαρα τις καλύτερες επιδόσεις. Αντίθετα, στο σύστημα Sandy Bridge η εικόνα είναι πιο σύνθετη. Συνεπώς, προκειμένου να σχηματιστεί μια ολοκληρωμένη εικόνα και επίσης να αξιολογηθούν οι επιδόσεις για μεγαλύτερους πίνακες στο σύστημα Sandy Bridge, δοκιμάζονται οι δύο καλύτερες υλοποιήσεις για διάφορα μεγέθη πινάκων και για τα καλύτερα μεγέθη block. Τα αποτελέσματα φαίνονται στο *Σχήμα 5.12*.



Σχ. 5.12: Χρόνοι εκτέλεσης των καλύτερων υλοποιήσεων LU Decomposition στο σύστημα Sandy Bridge

Παρατηρείται ότι με μικρή διαφορά η καλύτερη υλοποίηση στο σύστημα Sandy Bridge είναι η αναδρομική με μέγεθος block $B = 64$.

5.2. Υλοποίηση σε κάρτες γραφικών

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση του αλγορίθμου LU Decomposition για κάρτες γραφικών. Σε αντίθεση με τον αλγόριθμο Floyd-Warshall, αναπτύχθηκε μόνο υλοποίηση όπου ο πίνακας αποθηκεύεται ολόκληρος στη μνήμη DRAM της κάρτας γραφικών και δεν γίνεται χρήση της on-chip shared memory, δηλαδή η απλή «Global Memory» υλοποίηση.

Φυσικά, αν και δεν κρίθηκε σκόπιμο για το σκοπό της παρούσας διπλωματικής εργασίας, είναι εφικτή η εφαρμογή της τεχνικής του tiling και στον αλγόριθμο LU Decomposition. Συνεπώς, μπορούν να αναπτυχθούν οι ίδιες υλοποιήσεις με αυτές που αναπτύχθηκαν για τον αλγόριθμο Floyd-Warshall, με τα αντίστοιχα θετικά αποτελέσματα.

Όμοια με την *Ενότητα 4.2*, για την αξιολόγηση της απόδοσης χρησιμοποιείται πίνακας μεγέθους 8192×8192 στοιχείων. Επίσης, χρησιμοποιούνται αριθμοί διπλής ακρίβειας ($64\,bit$). Συνεπώς, ο γράφος που εξετάζεται έχει μέγεθος $8192 \times 8192 \times 8B = 512\,MB$.

Ο υπολογισμός του speedup γίνεται έχοντας ως βάση το σειριακό χρόνο εκτέλεσης στο σύστημα που φιλοξενεί την κάρτα γραφικών Tesla M2050 του απλού αλγορίθμου με τα 3 loops που αναλύθηκε στην *Υποενότητα 5.1.1*. Η μεταγλώττιση του προγράμματος του απλού αλγο-

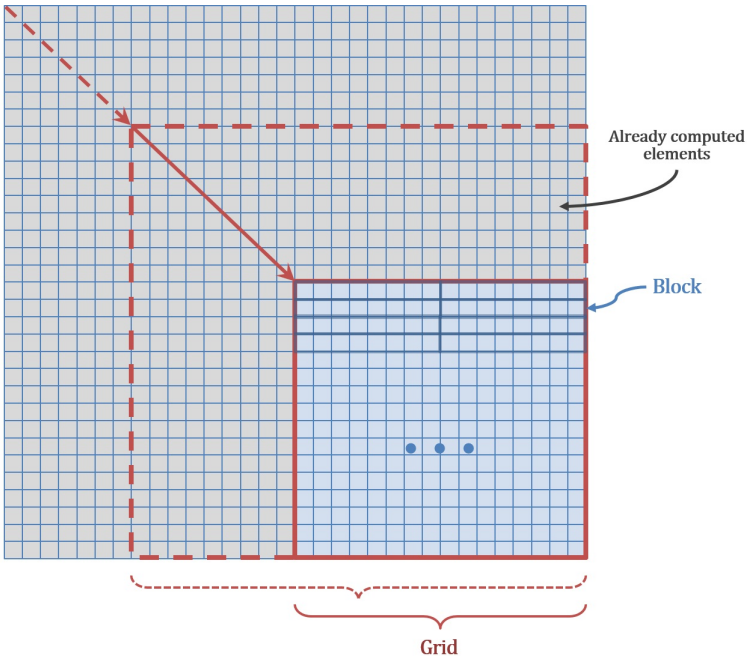
ρίθμου γίνεται με τον compiler GCC έκδοσης 4.6.3 με επίπεδο βελτιστοποιήσεων -O3 και Ο χρόνος που χρειάζεται για να επεξεργαστεί το γράφο των 8192 κόμβων είναι $350,05\,sec$.

5.2.1. Global Memory υλοποίηση

Ο πίνακας μεγέθους $N \times N$ μεταφέρεται ολόκληρος στη μνήμη DRAM της κάρτας γραφικών. Δημιουργούνται δύο kernels. Ο ένας υπολογίζει το πρώτο απλό for loop εντός του εξωτερικού k loop του αλγορίθμου με τα 3 loops της *Υποενότητας 5.1.1.* Ο δεύτερος υπολογίζει το διπλό for loop που ακολουθεί. Οι kernels εκτελούνται ο ένας μετά τον άλλον για κάθε k και μετά την ολοκλήρωση της εκτέλεσης το αποτέλεσμα αντιγράφεται στην κύρια μνήμη του συστήματος.

Όμοια με τις υλοποιήσεις του αλγορίθμου FW, το καλύτερο block είναι το οριζόντιο μίας διάστασης. Κάθε block περιέχει 256 threads, διότι έτσι επιτυγχάνεται 100% χρήση των SM/SMX των καρτών γραφικών. Αντίθετα όμως με τον αλγόριθμο FW, το μέγεθος του grid δεν είναι σταθερό για όλες τις k επαναλήψεις αλλά μειώνεται συνεχώς σε κάθε επανάληψη, όπως φαίνεται στο *Σχήμα 5.13.*

Λόγω του γεγονότος αυτού, δεν είναι αποδοτική η υλοποίηση όπου κάθε thread υπολογίζει παραπάνω από ένα στοιχεία του πίνακα που απέχουν απόσταση ίση με το μέγεθος του block, όπως ήταν εφικτό στην υλοποίηση 1D Block - 4 elements per thread του αλγορίθμου FW στην *Υποενότητα 4.2.1.*



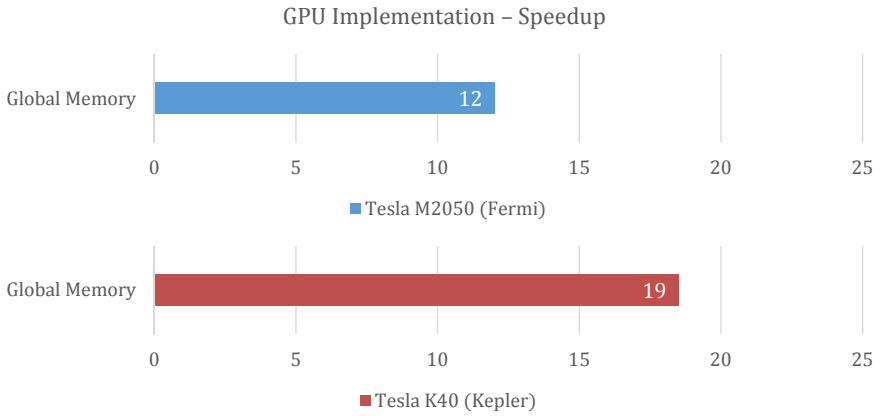
Σχ. 5.13: Κατανομή blocks της υλοποίησης Global Memory του αλγορίθμου LU Decomposition

Στον *Κώδικα 4.8* φαίνεται η υλοποίηση των δύο kenrels και οι κλήσεις τους από το κύριο πρόγραμμα που εκτελείται στον επεξεργαστή καθώς και η τροποποίηση του μεγέθους του grid σε κάθε k επανάληψη.

Κώδικας 5.4 Υλοποίηση «CUDA Global Memory» του αλγορίθμου LU Decomposition

```
#define BLOCK_SIZE 256

__global__ void _GPU_kernel_norm(const int k, double *A, const int N)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < k + 1)
        return;

    A[i * N + k] = A[i * N + k] / A[k * N + k];
}

__global__ void _GPU_kernel(const int k, double *A, const int N)
{
    int j = blockIdx.x * blockDim.x + threadIdx.x + k;
    int i = blockIdx.y + k;

    if ((i>=N) || (j >= N) || (i < k + 1) || (j < k + 1))
        return;
    double value_i_k = A[i * N + k];
    double value_k_j = A[k * N + j];
    value_k_j = value_i_k*value_k_j;
    __syncthreads();

    A[i * N + j] -= value_k_j;
}

main()
{
    ...
    dim3 grid_norm((N + BLOCK_SIZE - 1) / BLOCK_SIZE, 1);

    for(int k=0; k<N; k++)
    {
        grid = dim3(((N-k) + BLOCK_SIZE - 1) / BLOCK_SIZE, (N-k));
        _GPU_kernel_norm<<<grid_norm,BLOCK_SIZE>>>(k, A, N);
        _GPU_kernel<<<grid,BLOCK_SIZE>>>(k,A,N);
    }

    ...
}
```

Στο *Σχήμα 5.14* φαίνεται το speedup που επιτυγχάνεται σε κάθε κάρτα γραφικών. Στην Tesla M2050 ο χρόνος εκτέλεσης είναι *29,15 s*, ενώ στην Tesla K40 είναι *18,92 s*.



Σχ. 5.14: Αποτελέσματα υλοποίησης Global Memory για πίνακα μεγέθους 8192×8192

Το speedup που παρατηρείται είναι αρκετά μικρότερο και στις δύο κάρτες γραφικών σε σχέση με αυτό που παρατηρήθηκε στην αντίστοιχη υλοποίηση του αλγορίθμου FW. Αυτό οφείλεται σε αρκετούς παράγοντες. Αναλυτικότερα:

- Εντός του δεύτερου kernel υπάρχει σημαντικό divergence λόγω της αναγκαστικής ύπαρξης της πρότασης if ώστε να γνωρίζει κάθε thread αν βρίσκεται εκτός των ορίων του πίνακα. Το γεγονός αυτό μειώνει αρκετά τις επιδόσεις. Επίσης, το ίδιο προκαλεί και ο απαραίτητος συγχρονισμός των threads.

- Γίνεται κλήση σε δύο kernels ανά loop αντί σε έναν μόνο όπως στον αλγόριθμο FW, οπότε υπάρχει αυξημένο overhead κλήσεων και συγχρονισμών μεταξύ των kernels.

- Το μέγεθος του grid μειώνεται συνεχώς, όπως επίσης το επεξεργαστικό φορτίο κάθε κλήσης. Συνεπώς, η επίπτωση του αυξημένου overhead είναι μεγαλύτερη.

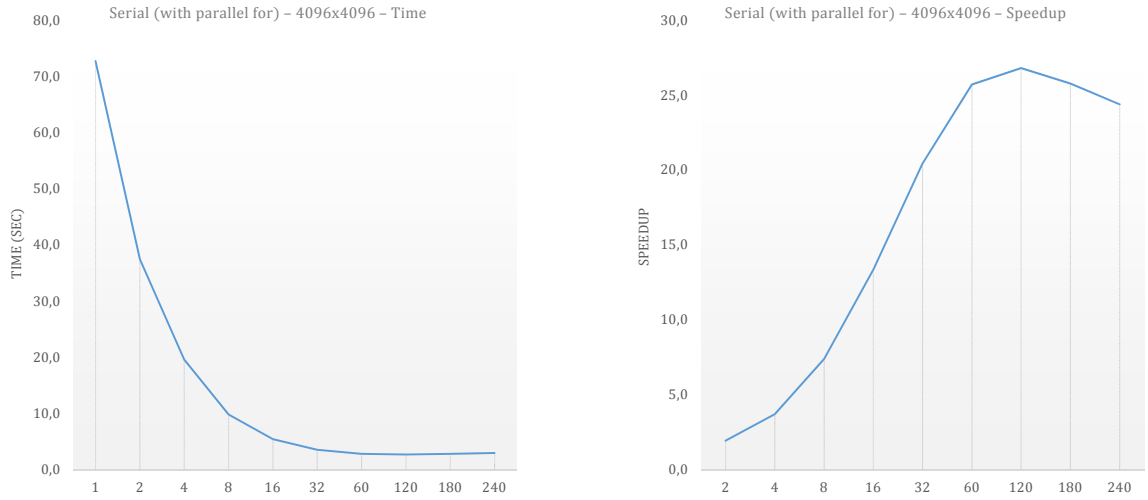
5.3. Υλοποίηση στον Intel Xeon Phi

Όπως και στην αντίστοιχη *Ενότητα 4.3*, η τρίτη αρχιτεκτονική η οποία εξετάζεται είναι η αρχιτεκτονική Intel MIC. Χρησιμοποιείται το Native Execution μοντέλο εκτέλεσης και αξιολογούνται οι επιδόσεις των υλοποιήσεων του αλγορίθμου LU Decomposition που αναπτύχθηκαν για συστήματα επεξεργαστών στην *Ενότητα 5.1*, με τις απαραίτητες αλλαγές ώστε να εκτελούνται πιο αποδοτικά στη συσκευή Xeon Phi.

Το μέγεθος του πίνακα επιλέγεται ίσο με 4096×4096 στοιχεία που αναπαρίστανται με αριθμούς διπλής ακρίβειας ($64\,bit$) και επιλέγεται «scatter» κατανομή των threads μέσω της enviromental variable $KMP\_AFFINITY$ του μεταγλωττιστή της Intel, αφού με εκείνη προκύπτουν οι καλύτεροι χρόνοι εκτέλεσης.

Σειριακή υλοποίηση

Αρχικά, αξιολογείται η απλή σειριακή υλοποίηση των τριών loops της *Υποενότητας 5.1.1*, παραλληλοποιημένη ομοίως με χρήση OpenMP parallel for. Στο *Σχήμα 5.15* φαίνονται τα αποτελέσματα του χρόνου εκτέλεσης και του speedup.

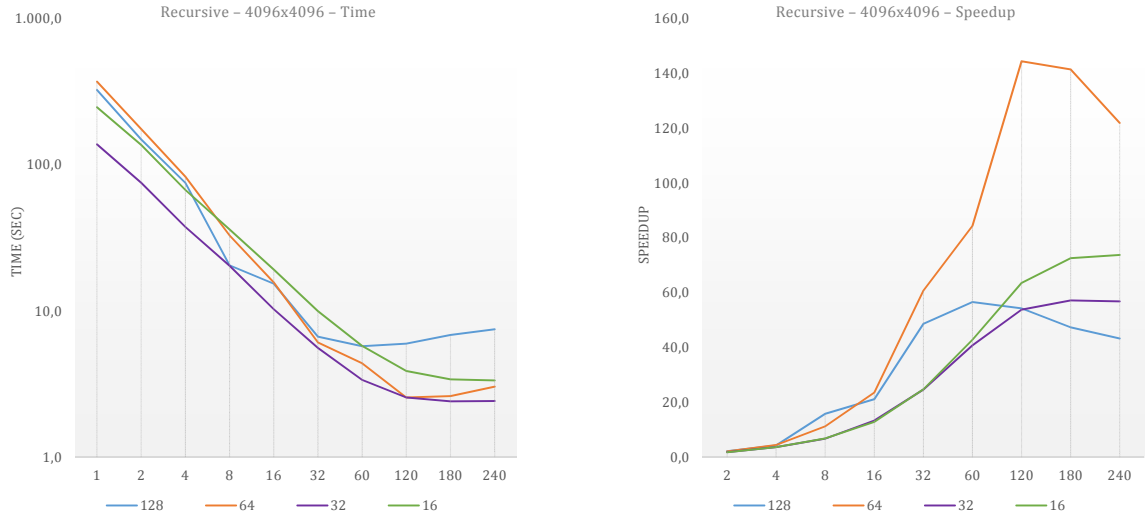


(a) (b)

Σχ. 5.15: Αποτελέσματα μετρήσεων σειριακής υλοποίησης στον Xeon Phi για πίνακα μεγέθους 4096×4096

Αναδρομική υλοποίηση

Στη συνέχεια, αξιολογείται η αναδρομική υλοποίηση της *Υποενότητας 5.1.2*, παραλληλοποιημένη ομοίως με χρήση OpenMP tasks. Τα αποτελέσματα φαίνονται στο *Σχήμα 5.16*.



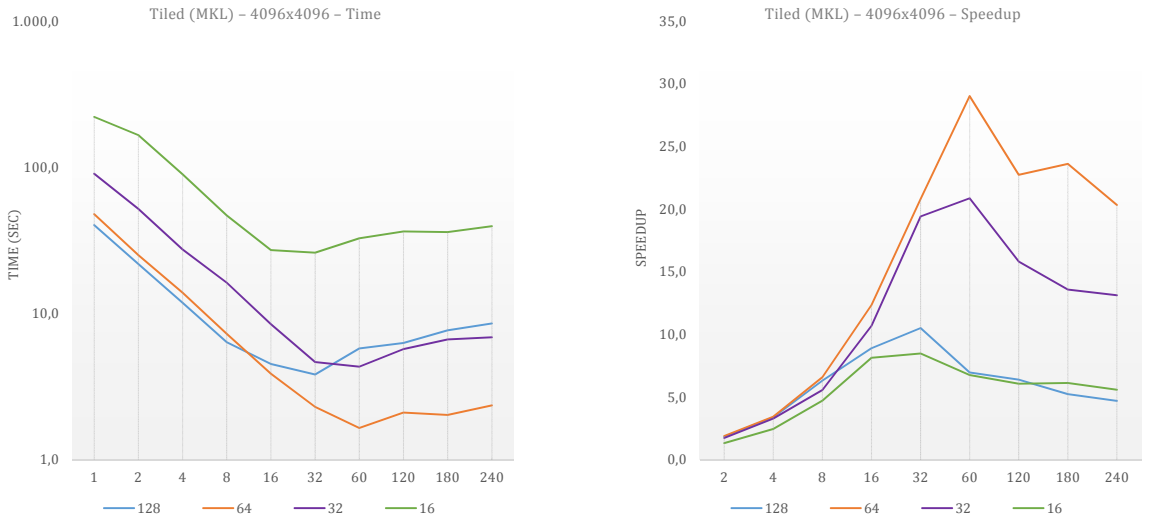
(a) (b)

Σχ. 5.16: Αποτελέσματα μετρήσεων αναδρομικής υλοποίησης στον Xeon Phi για πίνακα μεγέθους 4096×4096

Λόγω του υψηλού overhead των πολλών tasks που δημιουργούνται στην αναδρομική υλοποίηση, οι χρόνοι εκτέλεσης για μικρό αριθμό από threads είναι σημαντικά αυξημένοι. Όμως, για τον ίδιο λόγο, υπάρχει πολύ καλή κλιμάκωση όσο αυξάνεται ο αριθμός των threads.

Tiled υλοποίηση

Τέλος, αξιολογείται η tiled υλοποίηση της *Υποενότητας 5.1.3*, παραλληλοποιημένη με OpenMP tasks και χρήση της βιβλιοθήκης Intel MKL για τον πολλαπλασιασμό πινάκων. Η συγκεκριμένη βιβλιοθήκη είναι βελτιστοποιημένη από την εταιρία Intel όχι μόνο για συστήματα επεξεργαστών αλλά και για τον Xeon Phi. Τα αποτελέσματα φαίνονται στο *Σχήμα 5.17*.

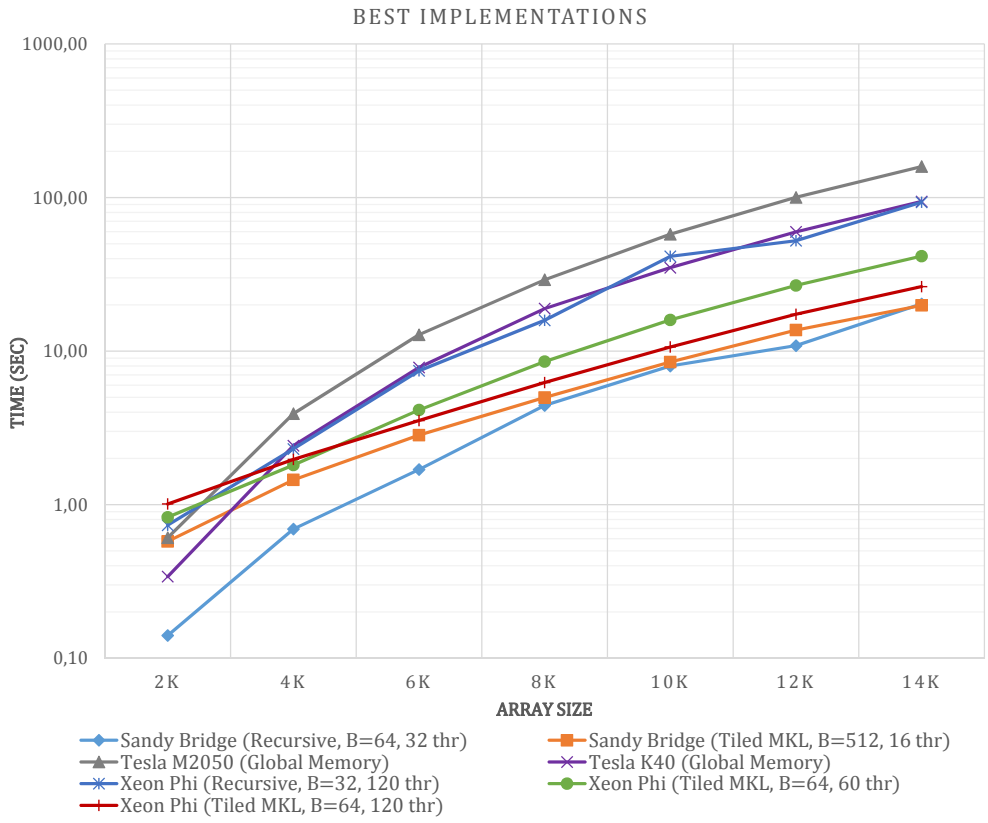


Σχ. 5.17: Αποτελέσματα μετρήσεων tiled υλοποίησης στον Xeon Phi για πίνακα μεγέθους 4096×4096

Λόγω της χρήσης της βελτιστοποιημένης βιβλιοθήκης MKL, η tiled υλοποίηση εμφανίζει τον καλύτερο χρόνο εκτέλεσης στον Xeon Phi.

Συγκεντρωτικά αποτελέσματα

Στο *Σχήμα 5.18* φαίνονται οι χρόνοι εκτέλεσης των καλύτερων υλοποιήσεων από κάθε αρχιτεκτονική (CPU, GPU, Xeon Phi) για τον αλγόριθμο LU Decomposition και για διάφορα μεγέθη πινάκων με χρήση αριθμών διπλής ακρίβειας (64 *bit*).



Σχ. 5.18: Χρόνοι εκτέλεσης των καλύτερων υλοποιήσεων από κάθε αρχιτεκτονική του αλγορίθμου LU Decomposition

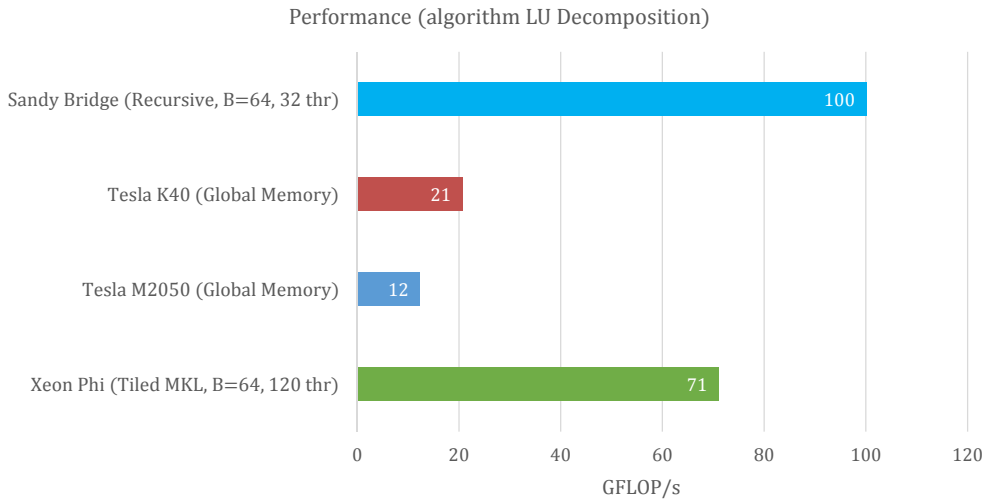
Παρατηρείται ότι η tiled MKL υλοποίηση με 120 threads είναι η γρηγορότερη υλοποίηση στον Xeon Phi. Σε σύγκριση με τις υπόλοιπες αρχιτεκτονικές, ο Xeon Phi εμφανίζει ανταγωνιστικά αποτελέσματα λόγω της χρήσης της βελτιστοποιημένης βιβλιοθήκης Intel MKL. Η αντίστοιχη tiled υλοποίηση με χρήση OpenMP tasks για την παραλληλοποίηση των πολλαπλασιασμών πινάκων όταν δοκιμάστηκε εμφάνισε τα χειρότερα αποτελέσματα από όλες και συνεπώς δεν συμπεριλαμβάνονται μετρήσεις. Για το λόγο αυτό, στον αλγόριθμο LU Decomposition (ομοίως με τον αλγόριθμο Floyd-Warshall) δεν επιβεβαιώνεται το επιχείρημα της εταιρίας Intel για την αποδοτική εκτέλεση στον Xeon Phi ήδη γραμμένων προγραμμάτων για επεξεργαστές.

Η καλύτερη υλοποίηση από όλες είναι η αναδρομική υλοποίηση στο σύστημα επεξεργαστών Sandy Bridge, με την tiled MKL να ακολουθεί με ελάχιστη διαφορά.

5.4. Συμπεράσματα

Κατά την εκτέλεση του αλγορίθμου LU Decomposition για πίνακα μεγέθους $N \times N$ εκτελούνται συνολικά (κατά προσέγγιση) $^2/_3 \cdot N^3$ πράξεις κινητής υποδιαστολής. Προκειμένου να αντληθούν χρήσιμα συμπεράσματα από τη σύγκριση των τριών αρχιτεκτονικών, υπολογίζεται η επίδοση κάθε υλοποίησης σε FLOP/s (floating point operations per second) με βάση τα αποτελέσματα για πίνακες μεγέθους από $8K \times 8K$ έως $14K \times 14K$ με χρήση αριθμών κινητής υποδιαστολής διπλής ακρίβειας ($64\,bit$). Στο *Σχήμα 5.19* φαίνεται το διάγραμμα των επιδόσεων σε GFLOP/s των υλοποιήσεων στις τρεις αρχιτεκτονικές. Σε αντίθεση με τα αποτελέσματα του

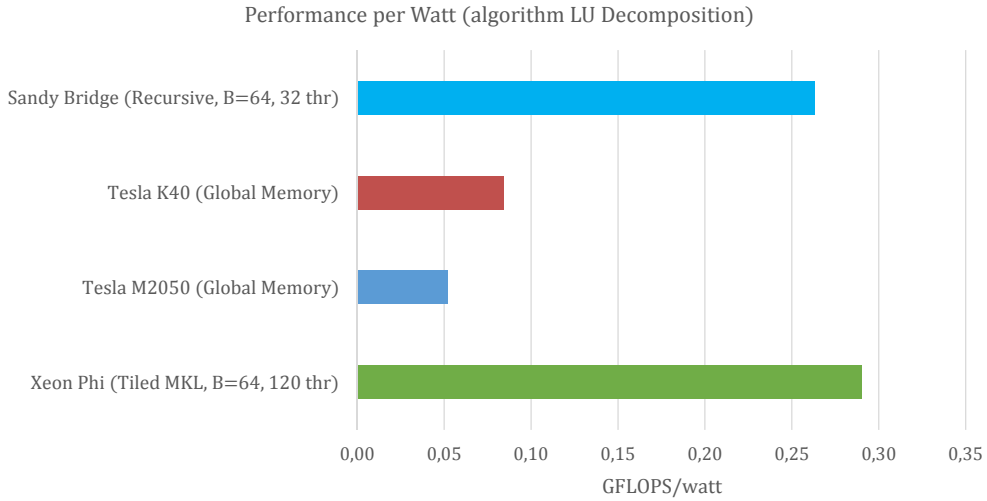
αλγορίθμου Floyd-Warshall, συμπεριλαμβάνεται στο διάγραμμα ο συνεπεξεργαστής Xeon Phi, αφού παρά το γεγονός ότι δεν αφιερώθηκε χρόνος για τη βελτιστοποίηση των υλοποιήσεών του, η χρήση της αποδοτικής βιβλιοθήκης Intel MKL οδήγησε σε ικανοποιητικά αποτελέσματα.



Σχ. 5.19: Επιδόσεις σε GFLOP/s (double precision 64 *bit*)

Παρατηρείται ότι τις καλύτερες επιδόσεις εμφανίζει η αναδρομική υλοποίηση στο σύστημα επεξεργαστών Sandy Bridge και ακολουθεί ο συνεπεξεργαστής Xeon Phi (χάρη στη χρήση της βιβλιοθήκης Intel MKL). Εντύπωση προκαλούν τα αρκετά άσχημα αποτελέσματα στις κάρτες γραφικών για τους λόγους που έχουν ήδη αναφερθεί. Είναι βέβαιο ότι η Shared Memory υλοποίηση για τον αλγόριθμο LU Decomposition (όπως εκείνη που αναπτύχθηκε για τον αλγόριθμο Floyd-Warshall) θα εμφάνιζε αντίστοιχα πολύ καλύτερες επιδόσεις.

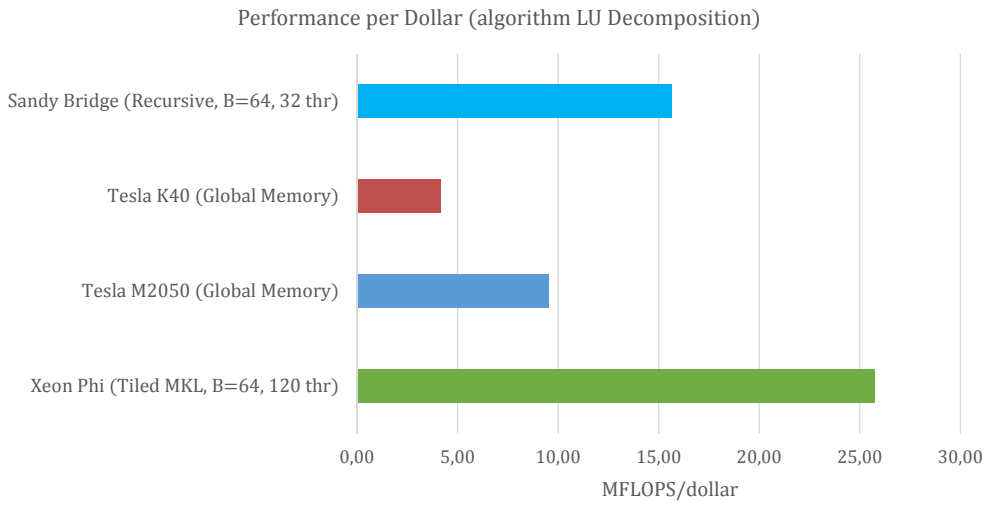
Για την περαιτέρω αξιολόγηση των διαφορετικών αρχιτεκτονικών, ακολουθεί στο *Σχήμα 5.20* διάγραμμα των επιδόσεων σε σχέση με την κατανάλωση. Η κατανάλωση σε Watt κάθε αρχιτεκτονικής λαμβάνεται από τους επίσημους πίνακες προδιαγραφών (όπως φαίνονται στην *Ενότητα 2.4*).



Σχ. 5.20: Επιδόσεις σε σχέση με την κατανάλωση

Τέλος, στο *Σχήμα 5.21* φαίνεται διάγραμμα των επιδόσεων σε σχέση με το κόστος αγοράς. Από το συγκεκριμένο διάγραμμα στόχος δεν είναι η ακριβής αποτύπωση της σύγκρισης ανάμεσα στις δύο αρχιτεκτονικές, αφού οι τιμές πώλησης δεν παραμένουν σταθερές και φυσικά υπάρχουν σημαντικές διαφορές ανάλογα με τον αριθμό τεμαχίων αγοράς.

Οι τιμές αγοράς των επεξεργαστών Sandy Bridge και των καρτών γραφικών είναι ίδιες με εκείνες που αναφέρονται στην Ενότητα 4.4, ενώ η τιμή του Xeon Phi 5120D λαμβάνεται από τον επίσημο τιμοκατάλογο της εταιρίας Intel και είναι ίση με $2759 (για αγορά 1000 τεμαχίων).



Σχ. 5.21: Επιδόσεις σε σχέση με το κόστος αγοράς

Κεφάλαιο 6

Συμπεράσματα

Στην παρούσα διπλωματική εργασία υλοποιήθηκαν και βελτιστοποιήθηκαν οι αλγόριθμοι Floyd Warshall και LU Decomposition σε συστήματα πολλαπλών επεξεργαστών, κάρτες γραφικών και στην manycore αρχιτεκτονική MIC της εταιρίας Intel. Εξηγήθηκε η στροφή των κατασκευαστών επεξεργαστών προς τους επεξεργαστές πολλαπλών πυρήνων (multicore) ως μέσο αύξησης των επιδόσεων και αναλύθηκαν οι αιτίες που οδήγησαν σε αυτό. Αναλύθηκαν και αξιολογήθηκαν όλες οι βελτιστοποιήσεις κατά την υλοποίηση των αλγορίθμων για συστήματα πολλαπλών επεξεργαστών.

Αναπτύχθηκαν σε βάθος η αρχιτεκτονική των καρτών γραφικών και το προγραμματιστικό μοντέλο που ακολουθούν και αναλύθηκαν οι μεγάλες δυνατότητες που παρέχουν. Παρουσιάστηκαν οι προκλήσεις που συναντώνται από τους προγραμματιστές κατά την υλοποίηση των αλγορίθμων για κάρτες γραφικών. Ο προγραμματισμός manycore αρχιτεκτονικών, όπως οι κάρτες γραφικών, απαιτεί αρκετή εξοικείωση. Τα εργαλεία ανάπτυξης που υπάρχουν διαθέσιμα και τα χαρακτηριστικά των σύγχρονων καρτών γραφικών διευκολύνουν αρκετά την κατάσταση. Φυσικά, για να ξεδιπλωθούν στο μέγιστο βαθμό οι δυνατότητες των καρτών γραφικών και να αναπτυχθούν πολύ γρήγορες και αποδοτικές υλοποιήσεις, πρέπει η φύση του προβλήματος να το επιτρέπει. Επιπλέον, απαιτείται μεγάλη προσπάθεια από την πλευρά του προγραμματιστή και εις βάθος κατανόησης του τρόπου λειτουργίας της εκάστοτε αρχιτεκτονικής. Όμως, όπως φάνηκε στον αλγόριθμο Floyd Warshall, όταν αυτές οι δύο προϋποθέσεις πληρούνται, τότε το αποτέλεσμα που προκύπτει είναι πολύ καλό και αποζημιώνει για το χρόνο που δαπανήθηκε. Επιπρόσθετα, εξετάστηκε και αντιμετωπίστηκε με επιτυχία το πρόβλημα της περιορισμένης μνήμης DRAM των καρτών γραφικών.

Τέλος, αναλύθηκε η αρχιτεκτονική του συνεπεξεργαστή Intel Xeon Phi, ο οποίος βασίζεται στην αρχιτεκτονική Many Integrated Core της εταιρίας Intel. Η διάρκεια πρόσβασης στη συσκευή ήταν περιορισμένη, συνεπώς στόχος ήταν η επιβεβαίωση ή μη ενός από τα βασικά επιχειρήματα προώθησης της συσκευής, δηλαδή η αποδοτική εκτέλεση στον Xeon Phi ήδη υλοποιημένων προγραμμάτων για επεξεργαστές. Το συγκεκριμένο επιχείρημα σε γενικές δεν επιβεβαιώνεται. Παρά ταύτα, η αρχιτεκτονική του Xeon Phi παρουσιάζει μεγάλο ενδιαφέρον και αν εφαρμοστούν οι απαραίτητες βελτιστοποιήσεις τότε σίγουρα οι επιδόσεις θα είναι πολύ καλύτερες. Επίσης, αν και δεν διερευνήθηκαν στην παρούσα διπλωματική εργασία, σημαντικό πλεονέκτημα του Xeon Phi αποτελούν τα υπόλοιπα δύο μοντέλα εκτέλεσης που υποστηρίζει και οι δυνατότητες υβριδικού παράλληλου προγραμματισμού που προσφέρουν.

Ολοκληρώνοντας, σαν τελικό συμπέρασμα φαίνεται να προκύπτει ότι, αν και δεν αποτελούν πανάκεια, οι manycore αρχιτεκτονικές προσφέρουν σημαντικά πλεονεκτήματα και αν συνδυ-

αστούν κατάλληλα με τα υπάρχοντα συστήματα πολυπύρηνων επεξεργαστών (multicore), τότε είναι εφικτή η επίτευξη σημαντικών επεξεργαστικών επιδόσεων με ικανοποιητική ενεργειακή αποδοτικότητα.

Λίστα Σχημάτων

Λίστα Πινάκων

Συντομογραφία

MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
AVX	Advanced Vector Extensions
CUDA	Compute Unified Device Architecture
EMU	Extended Math Unit
FMA	Fused Multiply-Add
FSB	Front Side Bus
GPGPU	General Purpose computing on Graphics Processing Units
GPU	Graphics Processing Unit
HPC	High Performance Computing
HTT	Hyper-Threading Technology
ILP	Instruction Level Parallelism
IMC	Intergrated Memory Controller
Intel MIC	Intel Many Integrated Core
Intel MKL	Intel Math Kernel Library
MAD	Multiply-Add
MCH	Memory Controller Hub
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
OpenCL	Open Computing Language
OpenMP	Open Specifications for Multi Processing
QPI	Intel QuickPath Interconnect
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Threads
SISD	Single Instruction Single Data
SM	Streaming Multiprocessor
SMT	Simultaneous Multithreading
SSE	Streaming SIMD Extensions
TDP	Thermal Design Power
TDP	Thermal Design Power
TLP	Thread Level Parallelism
UMA	Uniform Memory Access
VPU	Vector Processing Unit

Βιβλιογραφία

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* MIT Press and McGraw-Hill. ISBN 0-262-03384-4.

[2] David E. Culler, Jaswinder Pal Singh, Anoop Gupta. *Parallel Computer Architecture - A Hardware/Software Approach.* Morgan Kaufmann Publishers, 1999. ISBN 1-55860-343-3

[3] Gottlieb, Allan; Almasi, George S. (1989). *Highly parallel computing.* Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1

[4] David A. Patterson, John L. Hennessy. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 2013. ISBN-10: 0124077269, ISBN-13: 978-0124077263

[5] Intel® Hyper-Threading Technology, *Technical User's Guide*, January 2003

[6] *A View of the Parallel Computing Landscape*, Communications of the acm, October 2009, vol. 52, no. 10

[7] International Technology Roadmap for Semiconductors. *Executive Summary, 2005 and 2007*, http://public.itrs.net/

[8] Samuel H. Fuller, Lynette I. Millett. *Computing Performance: Game Over or Next Level?*, Published by the IEEE Computer Society 0018-9162/11/$26.00, January 2011.

[9] Sylvain Collange. *Design challenges in many-core architectures*, Journée GDR SOC-SIP, April 2013.

[10] David A. Patterson, John L. Hennessy. *Computer Architecture: A Quantitative Approach Fourth Edition*, 2007. ISBN 13: 978-0-12-370490-0, ISBN 10: 0-12-370490-1

[11] Flynn, M. J. *Some Computer Organizations and Their Effectiveness*, September 1972, IEEE Trans. Comput.

[12] Intel Automated Relational Knowledge Base, http://ark.intel.com/

[13] Advanced Micro Devices Official Webpage, http://www.amd.com/

[14] NVIDIA Fermi Compute Architecture Whitepaper, v1.1, 2009

[15] NVIDIA Kepler GK110 Compute Architecture Whitepaper, v1.0, 2012

[16] NVIDIA CUDA C Programming Guide, v5.5, July 2013

[17] OpenMP 4.0 Complete Specifications, July 2013

[18] Blaise Barney, *OpenMP.* Lawrence Livermore National Laboratory, 2013, https://computing.llnl.gov/tutorials/openMP/

[19] Nvidia Official Webpage, http://www.nvidia.com/

[20] PCI Express® 3.0 Frequently Asked Questions, http://www.pcisig.com/news_room/faqs/pcie3.0_faq

[21] Seneca, Intel® Xeon® Phi™ Coproccessor, http://www.senecadata.com/products/vendor-partners/Intel/intel-xeon-phi.aspx

[22] Xing Liu, Edmond Chow, Mikhail Smelyanskiy, Pradeep Dubey. *Efficient Sparse Matrix-Vector Multiplication on x86-Based Many-Core Processors.*

[23] Dirk Schmidl, Tim Cramer, Sandra Wienke, Christian Terboven, and Matthias S. Muller. *Assessing the Performance of OpenMP Programs on the Intel Xeon Phi.*

[24] http://www.qdpma.com/systemarchitecture/StrategyShift.html
[25] STREAM: Sustainable Memory Bandwidth in High Performance Computers, http://www.cs.virginia.edu/stream/
[26] Intel Math Kernel Library Reference Manual, https://software.intel.com/sites/products/documentation/hpc/mkl/mklman/
[27] Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms, Second Edition*