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

Μελέτη και αξιολόγηση των σύγχρονων Μονάδων Επεξεργασίας Γραφικών για χρήση σε εφαρμογές γενικού σκοπού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Πανταζή Ν. Τζαμπούρα

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

Αθήνα, Σεπτέμβριος 2008

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

Μελέτη και αξιολόγηση των σύγχρονων Μονάδων Επεξεργασίας Γραφικών για χρήση σε εφαρμογές γενικού σκοπού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Πανταζή Ν. Τζαμπούρα

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2η Σεπτεμβρίου 2008.

.....	.....	.....
Ν. Κοζύρης	Ν. Παπασπύρου	Κ. Πεκμεστζή
Αν.Καθηγητής Ε.Μ.Π.	Επ.Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2008

.....
Πανταζής Ν. Τζαμπούρας

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

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η μελέτη μιας από τις αρχιτεκτονικές των σύγχρονων Μονάδων Επεξεργασίας Γραφικών (GPUs) για χρήση της σε υπολογισμούς γενικού σκοπού (GPGPU), και ιδιαίτερα σε επιστημονικές εφαρμογές. Αυτές οι αρχιτεκτονικές προσφέρουν πολύ υψηλές υπολογιστικές δυνατότητες και ειδικά η επίδοση τους σε πράξεις κινητής υποδιαστολής, είναι σημαντικά καλύτερη από τις σύγχρονες CPUs του εμπορίου. Κατά καιρούς, έχουν χρησιμοποιηθεί κάρτες γραφικών για επιτάχυνση των υπολογισμών σε πληθώρα εφαρμογών αλλά οι διαδικασίες που έπρεπε να ακολουθηθούν ήταν ιδιαίτερα σύνθετες και πολύπλοκες. Αυτό συνέβαινε γιατί, το εκάστοτε πρόβλημα προς επίλυση έπρεπε πρώτα να απεικονιστεί σε πρόβλημα επεξεργασίας γραφικών. Με τις νέες γενιές γραφικών πολλά από τα προβλήματα ξεπεράστηκαν, ενώ παρουσιάστηκαν νέες προοπτικές.

Μελετήθηκε εδώ η αρχιτεκτονική G80 της εταιρείας nVidia, ένας από τους σημαντικότερους αντιπροσώπους των νέων γενιών GPUs. Αυτή η αρχιτεκτονική περιλαμβάνει ενοποιημένους υπολογιστικούς πυρήνες και αρκετά άλλα πολύ υποσχόμενα χαρακτηριστικά. Μαζί με αυτήν εξετάσθηκαν και οι δυνατότητες του CUDA, του πακέτου ανάπτυξης λογισμικού που η εταιρεία προσφέρει, και θα είναι συμβατό και με όλες τις επόμενες γενιές GPUs της εταιρείας. Μέσω αυτού αποφεύγεται πλέον η ανάγκη χρησιμοποίησης των προγραμματιστών διεπαφών γραφικών για προγραμματισμό στις νέες GPUs.

Αρχικά, μελετήθηκαν τα χαρακτηριστικά της αρχιτεκτονικής και ο τρόπος που η προγραμματιστική διεπαφή τα χρησιμοποιεί. Δόθηκε ιδιαίτερη έμφαση σε τεχνικές βελτιστοποίησης που θα πρέπει να ακολουθούνται από τους προγραμματιστές με σκοπό την πλήρη εκμετάλλευση της υποκείμενης αρχιτεκτονικής. Στη συνέχεια, εκτελέστηκε ένας αριθμός πειραμάτων, για να αξιολογηθεί η δυνατότητα της μελετούμενης αρχιτεκτονικής να αντεπεξέλθει στις ανάγκες υπολογισμών γενικού σκοπού. Τα αποτελέσματα ήταν ενθαρρυντικά, και ειδικά αν ακολουθούνται οι σημαντικότερες από τις στρατηγικές βελτιστοποίησης της επίδοσης, μπορούν να επιτευχθούν μεγάλες επιταχύνσεις σε ποικίλες εφαρμογές.

Οι προοπτικές της προσπάθειας για GPGPU φαίνονται πολλά υποσχόμενες, ενώ ήδη έχουν σχεδιαστεί ακόμα πιο εξελιγμένες γενιές GPUs για εκμετάλλευση στο άμεσο μέλλον. Είναι γεγονός ότι το ενδιαφέρον της έρευνας έχει στραφεί πλέον στον σχεδιασμό υπολογιστικών συστημάτων με μαζικά πολλούς πυρήνες, δανειζόμενη από τις GPUs πολλές ιδέες και χαρακτηριστικά. Διαφαίνεται, τελικά, ένα είδος σύγκλισης ως προς κάποια χαρακτηριστικά που θα υλοποιούν οι μελλοντικές CPUs και GPUs, όπως για παράδειγμα ο μεγάλος αριθμός υπολογιστικών πυρήνων και η ταυτόχρονη υποστήριξη πολλών νημάτων σε επίπεδο υλικού.

Λέξεις Κλειδιά: Πολυπύρηνοι επεξεργαστές, επεξεργαστές πολλών πυρήνων, Μονάδα Επεξεργασίας Γραφικών (GPU), υπολογισμοί Γενικού Σκοπού σε GPUs (GPGPU), αρχιτεκτονική nVidia G80, μοντέλο προγραμματισμού σε ροές, CUDA

Abstract

The purpose of this diploma thesis is the study of one of the architectures of the modern Graphics Processing Units (GPUs) for use in general purpose computations (GPGPU), and especially in scientific applications. These architectures offer very high computational power, and their performance in floating point computations is substantially better than the performance of modern CPUs. During the past, GPUs have been used for acceleration of computations in a number of applications, but it was a particularly difficult task. This happened because each problem should be initially mapped into a graphics problem in order to use the GPU's hardware efficiently. After the appearance of the new generations of GPUs many of these problems no longer exist, and new prospects have opened up.

The nVidia G80 architecture, one the most powerful representatives of the new generations of the GPUs, was studied. This architecture implements unified processing cores and many other new promising characteristics. Along with this architecture we studied the capabilities of CUDA, the Software Development Kit nVidia presented, and which will be compatible with future GPUs, as well. After the coming of CUDA, along with the streaming programming model it offers, there is no longer a need for use of graphics APIs in general purpose computations.

Initially, we studied the main characteristics of the G80 architecture and the way it is used by the CUDA API. We emphasized in a number of optimization techniques that CUDA programmers should apply, in order to achieve maximum performance. Then, a number of experiments were executed, in order to evaluate the performance of this new platform in solving general purpose problems. The results were more than satisfactory, and especially if the most important of the performance guidelines are followed, substantial speed-ups can be attained.

The prospects of the GPGPU effort seem really promising, whereas new, even more powerful, architectures have been designed and are soon to be exploited. It is a fact that a great amount of researchers' interest has been turned to the design of manycore computing systems, borrowing many characteristics from the GPUs. There seems to be a kind of convergence in the horizon, as far as certain aspects of the future GPUs and CPUs are concerned, such as the large number of processing cores and the support of many hardware threads.

Keywords: Multicore processors, manycore processors, Graphics Processing Unit (GPU), General Purpose computations on GPUs (GPGPU), nVidia G80 architecture, , streaming programming model, CUDA.

Περιεχόμενα

Κεφάλαιο 1

Επεξεργαστικά Συστήματα Πολλαπλών Πυρήνων

Η παράλληλη επεξεργασία είναι ο τρόπος επεξεργασίας ή υπολογισμού κατά τον οποίο την ίδια στιγμή συμβαίνουν ταυτόχρονα πολλαπλές λειτουργίες. Το όλο σκεπτικό βασίζεται στην αρχή, ότι τα περισσότερα μεγάλα προβλήματα προς λύση μπορούν να διαιρεθούν σε μικρότερα προβλήματα-συνιστώσες, τα οποία είναι δυνατόν να λύνονται «παράλληλα». Από τους Almasi και Gottlieb [1] έχει αποδοθεί ο παρακάτω ορισμός για την έννοια του παράλληλου υπολογιστή, του εργαλείου δηλαδή της παράλληλης επεξεργασίας, που συνοψίζει την παραπάνω ιδέα:

«Ένας παράλληλος υπολογιστής είναι μια συλλογή επεξεργαστικών στοιχείων τα οποία επικοινωνούν και συνεργάζονται μεταξύ τους, με απώτερο σκοπό την επίλυση μεγάλων προβλημάτων γρήγορα.»

Η παραπάνω ρήση όμως εγείρει αρκετά ερωτήματα. Αυτά έχουν να κάνουν με το πλήθος των επεξεργαστικών στοιχείων που θα συνεργάζονται, με το πώς θα επικοινωνούν, με τον ακριβή τρόπο συνεργασίας τους, με το πόσο γρηγορότερα λύνουν το ίδιο πρόβλημα σε σύγκριση με έναν μη παράλληλο επεξεργαστή, με το πόσο σύνθετα θα είναι τα δομικά στοιχεία του επεξεργαστή αυτού, με το πώς θα αντιμετωπίζει ένα τέτοιο μηχάνημα ένας επίδοξος προγραμματιστής/χρήστης του, και με άλλα ζητήματα.

Η Αρχιτεκτονική παράλληλων υπολογιστικών συστημάτων σε όλα τα επίπεδα τροφοδοτήθηκε από την ανάγκη να επιλύονται πιο γρήγορα τα προβλήματα ή να λύνονται μεγαλύτερα σε έκταση προβλήματα και από τη δίψα των σχεδιαστών υπολογιστών για ολοένα και καλύτερη επίδοση των μηχανημάτων τους. Μέσω της παραλληλίας, σε όλα τα επίπεδα που θα εξερευνηθεί, κατάφεραν να πετύχουν βελτίωση των δυνατοτήτων των υπολογιστών με ραγδαίους ρυθμούς και να ξεπερνούν κάθε φορά όποιο εμπόδιο παρουσιαζόταν και δυσχέραινε την ανάπτυξη αυτού του τομέα της τεχνολογίας. Αποτέλεσμα ήταν η συνεχής τροφοδότηση λύσεων σε μορφή υλικού (hardware) στη συνεχή ζήτηση για πιο σύνθετες, «βαρύτερες», μεγαλύτερες και δυσκολότερες εφαρμογές.

Η Αρχιτεκτονική παράλληλων υπολογιστών έχει να κάνει με την οργάνωση των πολλών πόρων που θα έχει ένα παράλληλο σύστημα επεξεργασίας, ώστε να εκμεταλλευτεί στο έπακρο τις δυνατότητές του και να μας προσφέρει αυξημένες δυνατότητες σε σχέση με τα μονοεπεξεργαστικά συστήματα. Πρέπει δηλαδή το σχεδιαζόμενο σύστημα να είναι σε θέση να αντλεί το 100% των δυνατοτήτων όλων των δομικών μερών που το αποτελούν, κάτι που απαιτεί καλή οργάνωση, μηχανισμούς συνεργασίας και συγχρονισμού και αποδοτικούς τρόπους και δρόμους επικοινωνίας μεταξύ των στοιχείων που το αποτελούν. Στις μέρες μας πλέον, οι παράλληλες αρχιτεκτονικές κατέχουν εξέχοντα ρόλο στην επεξεργασία της πληροφορίας, εφόσον όλα τα βλέμματα έχουν στραφεί επάνω τους μετά τον κορεσμό των άλλων τρόπων αύξησης της επίδοσης των υπολογιστικών συστημάτων.

Η ιδέα της παραλληλίας δεν είναι καινούργια στον τομέα της Αρχιτεκτονικής Υπολογιστών. Αντίθετα, έχει ήδη χρησιμοποιηθεί, και συνεχίζει να χρησιμοποιείται, σε πολλά διαφορετικά επίπεδα σχεδίασης των υπολογιστικών συστημάτων, όπως θα παρουσιαστεί πιο αναλυτικά στην ενότητα 1.1. Το τελευταίο επίπεδο στο οποίο χρησιμοποιήθηκε παραλληλία είναι οι πολλαπλοί επεξεργαστές στην ίδια ψηφίδα που περιλαμβάνουν τα υπολογιστικά μηχανήματα πλέον, είτε πρόκειται για αυτά που κυκλοφορούν στην αγορά είτε για ιδιαίτερα εξελιγμένα και εξειδικευμένα παράλληλα συστήματα (Supercomputers – Υπερυπολογιστές).

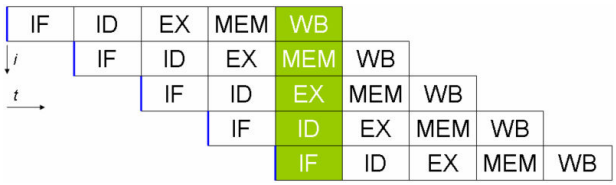
1.1 Είδη Παραλληλίας πάνω σε μια ψηφίδα

Παραλληλισμός σε επίπεδο bit

Πρώτα από όλα, η ιδέα της παραλληλίας χρησιμοποιήθηκε σε επίπεδο bit. Αυξήθηκε με τον καιρό το μέγεθος της λέξης των υπολογιστών, δηλαδή το μέγεθος της πληροφορίας που μπορούσε ένα μηχάνημα να επεξεργάζεται σε κάθε κύκλο. Ο διαδοχικός διπλασιασμός της λέξης είχε ως συνέπεια να μειωθούν οι εντολές που χρειάζονταν για να εκτελεστεί μια λειτουργία πάνω σε δεδομένα με μέγεθος μεγαλύτερο του μεγέθους της λέξης του υπολογιστή, κάτι που είχε ως αποτέλεσμα μείωση του χρόνου που θα απαιτούνταν για να εκτελεστεί η συγκεκριμένη λειτουργία, και επομένως, αύξηση της ταχύτητας του υπολογιστή. Αρχικά υπήρχαν 4-bit επεξεργαστές οι οποίοι αντικαταστάθηκαν πρώτα με 8-bit, έπειτα με 16-bit επεξεργαστές και αργότερα με 32-bit επεξεργαστές. Τότε ήταν και η στιγμή που αυτή η τάση παραλληλίας σε επίπεδο bit έδειχνε να φτάνει σε ένα τέλμα μέχρι πρόσφατα (2003) με τον ερχομό της αρχιτεκτονικής x86-64 πρώτος εκπρόσωπος της οποίας ήταν ο επεξεργαστής Opteron της AMD [2].

Παραλληλισμός σε επίπεδο εντολής

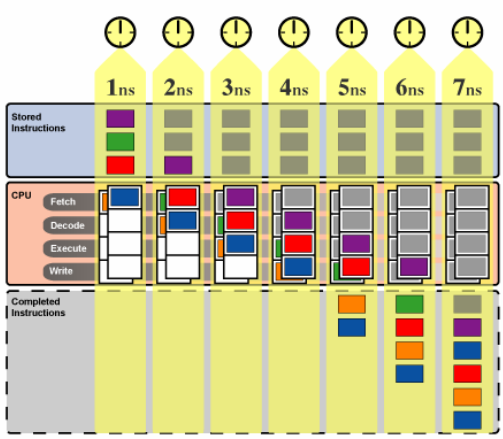
Ένας δεύτερος τρόπος εφαρμογής της ιδέας της παραλληλίας στην Αρχιτεκτονική Υπολογιστών, είναι να εκτελούνται παράλληλα διαφορετικές εντολές, διαδοχικές ή μη στον κώδικα του προγράμματος. Αυτό επιτυγχάνεται με τη χρήση της τεχνικής της σωλήνωσης (pipelining). Αυτού του είδους η παραλληλία είναι γνωστή σαν Instruction-Level Parallelism (Παραλληλισμός σε επίπεδο εντολής). Την ίδια χρονική στιγμή μπορεί ο επεξεργαστής να έχει μια διαφορετική εντολή σε κάθε ένα από τα στάδια αυτά. Για παράδειγμα, αν ένας επεξεργαστής έχει 10 στάδια στο pipeline μπορεί να επεξεργάζεται 10 διαφορετικές εντολές, την κάθε μια σε διαφορετικό στάδιο. Αυτό, βέβαια, είναι εφικτό, εφόσον δεν υπάρχει κάποιου είδους εξάρτηση μεταξύ κάποιων από αυτές τις εντολές, οπότε και πρέπει να αποφευχθούν τυχόν λάθη στα αποτελέσματα με τη χρήση των κατάλληλων τεχνικών. Η πρόοδος σε αυτό το είδος παραλληλίας κυρίαρχησε στον τομέα της Αρχιτεκτονικής Υπολογιστών από τα μέσα της δεκαετίας του 1980 μέχρι και τα μέσα της δεκαετίας του 1990. Μέχρι τότε, συνήθως, κάθε ένα από τα βασικά στάδια εκτέλεσης της εντολής (αποκωδικοποίηση εντολής, υπολογισμός διεύθυνσης, αριθμητική πράξη κτλ.) απαιτούσε έναν κύκλο ρολογιού για να ολοκληρωθεί. Κατά συνέπεια, για να εκτελεστεί η εντολή χρειαζόταν πολλούς κύκλους. Αυτό είχε ως αποτέλεσμα, η διεκπεραιωτική ικανότητα των επεξεργαστών να είναι πολύ μικρότερη της μιας εντολής ανά κύκλο ρολογιού. Με τη μέθοδο της σωλήνωσης στόχος είναι σε κάθε κύκλο, να ολοκληρώνεται μια διαφορετική εντολή. Οι σύγχρονοι επεξεργαστές έχουν πολλά στάδια στη σωλήνωσή τους. Το πιο συνηθισμένο παράδειγμα που χρησιμοποιείται για την πλήρη κατανόηση του παραλληλισμού σε επίπεδο εντολής, είναι ένας επεξεργαστής RISC (Reduced Instruction Set Computer – Υπολογιστής Μειωμένου Συνόλου Εντολών) με πέντε στάδια σωλήνωσης (βλ. Σχήμα 1-1): instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), register write back (WB), όπου την ίδια χρονική στιγμή 5 διαφορετικές εντολές βρίσκονται σε διαφορετικό στάδιο της σωλήνωσης.



Σχήμα 1-1 Παραλληλισμός σε επίπεδο εντολής με σωλήνωση. Σε αυτό το παράδειγμα έχουμε πέντε στάδια σωλήνωσης από τα οποία περνάει κάθε εντολή. Την ίδια χρονική στιγμή λοιπόν, πέντε διαφορετικές εντολές βρίσκονται σε διαφορετικά στάδια της σωλήνωσης.

Με τον καιρό και καθώς η επιφάνεια των δομικών μονάδων του επεξεργαστή (οι Αριθμητικές και Λογικές Μονάδες για ακέραιους αριθμούς και για αριθμούς κινητής υποδιαστολής, ο ελεγκτής της Κρυφής μνήμης κτλ.) μειωνόταν, κατέστη δυνατό να ενσωματωθούν όλα τα στοιχεία από τα οποία αποτελούνταν ο επεξεργαστής σε ένα τσιπ. Έγινε σιγά-σιγά αρκετά ελκυστική η ιδέα να επεξεργάζονται ταυτόχρονα πολλές εντολές, οι οποίες σε κάποια στάδια τους να χρησιμοποιούν ξεχωριστό υλικό η κάθε μία. Αυτό ήταν και το επόμενο βήμα του Παραλληλισμού σε επίπεδο εντολής. Αυτοί, οι επεξεργαστές ονομάστηκαν υπέρ-βαθμωτοί επεξεργαστές (superscalar processors) και είχαν τη δυνατότητα να εκμεταλλεύονται τους ολοένα αυξανόμενους πόρους που εμφανίζονταν στο τσιπ του επεξεργαστή. Αυτό κατέστη δυνατό, καθώς σταδιακά προστέθηκαν και άλλες μονάδες διαφόρων λειτουργιών που συνήθως είχαν να κάνουν με το στάδιο εκτέλεσης των εντολών. Είχαμε δηλαδή μια πιο ευρεία σωλήνωση

(wide pipeline), καθώς πλέον, σε κάθε κύκλο ρολογιού κομίζονταν πολλαπλές εντολές οι οποίες χρησιμοποιούν τις πολλές διαφορετικές μονάδες υλικού που είναι διαθέσιμες, όπως, για παράδειγμα, στο Σχήμα 1-2 όπου είναι υπάρχει το διπλάσιο υλικό (hardware). Έτσι, δυο ανεξάρτητες μεταξύ τους εντολές μπορούν να εκτελούνται παράλληλα, βρισκόμενες στο ίδιο στάδιο της σωλήνωσης, στον ίδιο κύκλο μηχανής. Πρόκειται, δηλαδή, για ένα άλλον τρόπο παραλληλίας που έδωσε ακόμα μεγαλύτερη ώθηση στην Αρχιτεκτονική Υπολογιστών και στην ανάπτυξη ολοένα ταχύτερων και αποδοτικότερων μηχανημάτων. Οι υπερ-βαθμωτές αρχιτεκτονικές επίσης υποστηρίζουν την εκτέλεση εντολών εκτός σειράς (out-of-order). Αυτό σημαίνει ότι κάποιες εντολές που ίσως καθυστερούν σε κάποια από τις σωληνώσεις του υπερ-βαθμωτού επεξεργαστή (όπως όταν γίνεται μια αναφορά στην κύρια μνήμη), παρακάμπτονται και συνεχίζεται η εκτέλεση κάποιων από τις επόμενες εντολές. Βέβαια, τελικά, θα φαίνεται ότι οι εντολές ολοκληρώνονται με τη σειρά που επιτάσσει το αρχικό πρόγραμμα (in-order retirement). Έτσι, διατηρείται η ορθότητα της λογικής σειράς των εντολών και τα αποτελέσματά είναι τα ίδια με την περίπτωση που οι εντολές θα εκτελούνταν αυστηρά σειριακά. Το Σχήμα 1-2 κάνει ακόμα πιο ξεκάθαρα τα παραπάνω. Φαίνεται ότι υπάρχουν δυο ξεχωριστές σωληνώσεις, η κάθε μια από τις οποίες έχει όλα τα στάδια του pipeline (εδώ τέσσερα). Σε κάθε στάδιο, κάθε σωλήνωσης μπορεί να εκτελείται μια εντολή σε κάθε κύκλο. Έχει υποτεθεί κύκλος διάρκειας 1 ns, ενδεικτικά. Παρατηρείται ότι τελικά οι εντολές ακολουθούν δυο διαφορετικούς δρόμους αλλά τελικά φαίνεται στο χρήση ότι η σειρά τους διατηρήθηκε.



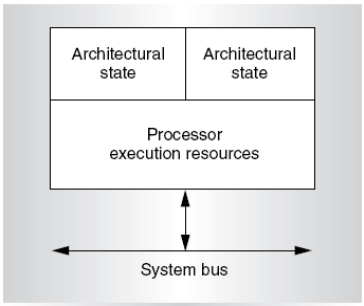
Σχήμα 1-2 Παράδειγμα υπέρ-βαθμωτού επεξεργαστή, ο οποίος περιλαμβάνει διπλό υλικό, ώστε να μπορεί να επεξεργάζεται την κάθε στιγμή δυο διαφορετικές εντολές.

Παραλληλισμός σε επίπεδο νημάτων

Επόμενος τρόπος παραλληλίας που εξερευνήθηκε και εξακολουθεί να παρουσιάζει ιδιαίτερο ενδιαφέρον και σήμερα είναι ο παραλληλισμός σε επίπεδο νημάτων (Thread Level Parallelism - TLP). Πολλές φορές σε έναν υπέρ-βαθμωτό επεξεργαστή ο οποίος εκτελεί ένα μόνο νήμα, είναι δυνατόν να μην μπορεί να γίνει χρήση όλων των διαθέσιμων πόρων τους συστήματος, γιατί δε θα είναι εφικτό να υπάρξει αρκετή παραλληλία σε επίπεδο εντολών (π.χ. όταν κάποια εντολή χρειάζεται να περιμένει δεδομένα από την κύρια μνήμη, όταν έγινε κάποια λανθασμένη πρόβλεψη διακλάδωσης, κ.α.). Αυτό οδηγεί σε σπατάλη υπολογιστικής δύναμης και δεν επιτρέπει την πλήρη εκμετάλλευση του υπολογιστικού συστήματος. Με την τεχνική SMT (Simultaneous MultiThreading – Ταυτόχρονος Πολυνηματισμός) ο επεξεργαστής μπορεί να εκτελεί την

ίδια στιγμή εντολές από περισσότερα του ενός διαφορετικά νήματα εκτέλεσης [3]. Αυτά τα νήματα θα ανταγωνίζονται μεταξύ τους για τους πόρους του επεξεργαστή, οι οποίοι θα μοιράζονται στα εν λόγω νήματα με στόχο να είναι συνεχώς «κατειλημμένες» όλες οι λειτουργικές μονάδες του. Με άλλα λόγια, αν ένα από τα νήματα που είναι προς εκτέλεση στον επεξεργαστή δεν μπορεί να εμφανίσει αρκετό παραλληλισμό σε επίπεδο εντολών ώστε να κρατά απασχολημένα συνεχώς όλα τα κυκλώματα του επεξεργαστή, παράλληλα με αυτό το νήμα θα μπορεί να εκτελείται και κάποιο άλλο, το οποίο θα μοιράζεται με το πρώτο τους πόρους του επεξεργαστή. Σε περίπτωση που υπάρχει ένα μοναδικό νήμα προς εκτέλεση, εάν δε μπορεί να απασχολεί συνέχεια τον επεξεργαστή, θα είναι αναπόφευκτη η σπατάλη πόρων, αλλά σε τέτοια περίπτωση δεν θα καθυστερεί κάποια άλλη εργασία.

Η ιδέα του SMT υλοποιήθηκε από την Intel στην οικογένεια επεξεργαστών Pentium 4 και Xeon με την ονομασία Hyperthreading Technology [4]. Με αυτήν την τεχνολογία ο επεξεργαστής παρουσιάζεται στο υπερκείμενο λογισμικό ως ένα σύνολο επεξεργαστών. Αυτό επιτυγχάνεται διατηρώντας περισσότερα του ενός αντίγραφα της αρχιτεκτονικής κατάστασης (architectural state) του επεξεργαστή. Η αρχιτεκτονική κατάσταση του επεξεργαστή ορίζεται ως το σύνολο των καταχωρητών δεδομένων/ελέγχου, διακοπών, κ.λπ. Για παράδειγμα, στην μικροαρχιτεκτονική Netburst, κάθε πραγματικός επεξεργαστής παρουσιάζεται σαν δυο επεξεργαστές όπως φαίνεται στο Σχήμα 1-3. Όπως και όλοι οι τρόποι βελτίωσης της επίδοσης ενός επεξεργαστή υπάρχει και κάποιο κόστος στην επιφάνεια της ψηφίδας του τσιπ του επεξεργαστή. Σε αυτήν την υλοποίηση της Intel παρατηρήθηκε αύξηση επιφάνειας της τάξης του 5%, ενώ η αύξηση επίδοσης ήταν κατά μέσο όρο κοντά στο 25% [4].



Σχήμα 1-3 Η αρχιτεκτονική Netburst ουσιαστικά κάνει το σύστημά μας να βλέπει τον επεξεργαστή «διπλό», μοιράζοντας δυνητικά στους δύο λογικούς επεξεργαστές τους πόρους του τσιπ.

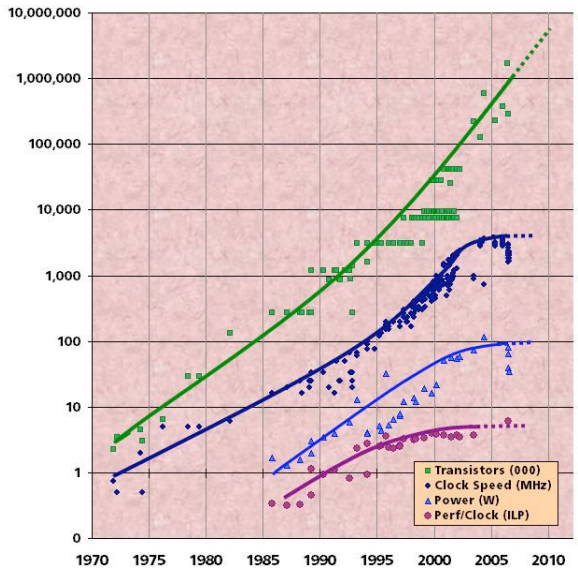
Άλλη μια υλοποίηση παραλληλισμού νημάτων σε επίπεδο υλικού είναι ο επεξεργαστής Niagara της Sun [5]. Αυτός ο επεξεργαστής στοχεύει τη χρησιμοποίηση του σε συστήματα εξυπηρετητών (servers) για διαδικτυακές εφαρμογές. Σε τέτοια συστήματα υπάρχει μεγάλος αριθμός αιτήσεων εξυπηρέτησης, κάθε μια από τις οποίες αποτελεί ξεχωριστό νήμα εκτέλεσης προς επεξεργασία. Η έλλειψη παραλληλισμού σε επίπεδο εντολών και τοπικότητας στις αναφορές μνήμης, καθώς και η συνήθης αποτυχία στην πρόβλεψη διακλαδώσεων (branch prediction) εμποδίζουν την επίτευξη υψηλών επιδόσεων. Ο Niagara αποτελείται από οκτώ πυρήνες (βλ. ενότητα 1.4.1.3), κάθε ένας εκ των οποίων μπορεί να εκτελεί τέσσερα νήματα εκτέλεσης χρησιμοποιώντας την τεχνολογία του ταυτόχρονου πολυνηματισμού. Έτσι, συνολικά, ο επεξεργαστής Niagara μπορεί να εκτελέσει 32 νήματα εκτέλεσης ταυτόχρονα.

Παραλληλισμός σε επίπεδο πυρήνων

Το τελευταίο είδος παραλληλίας που παρουσιάστηκε στον τομέα της αρχιτεκτονικής υπολογιστών είναι η παραλληλία σε επίπεδο πυρήνων, με την συνύπαρξη πολλών επεξεργαστικών πυρήνων στην ίδια ψηφίδα. Οι αρχιτεκτονικές αυτές υλοποιούν περισσότερους από έναν πυρήνες στο ίδιο κομμάτι ολοκλήρωσης (τσιπ), με συνήθεις περιπτώσεις αυτές των δύο και τεσσάρων πυρήνων. Πλέον, έχοντας στο σύστημά παραπάνω από έναν πυρήνα μπορεί να γίνει ανάθεση σε κάθε έναν από αυτούς κάποια διαφορετική λειτουργία την οποία θα αναλαμβάνει εξ'ολοκλήρου αυτός ή να διαμοιράζουμε στους πολλαπλούς πυρήνες κάποιο σύνθετο πρόβλημα προς επίλυση. Με αυτό το είδος παραλληλίας θα ασχοληθούμε από εδώ και στο εξής αφού, όπως θα εξηγηθεί σε επόμενες παραγράφους, τα άλλα είδη παραλληλίας (εκτός της περίπτωσης του SMT) έχουν εξαντλήσει τα περιθώρια βελτίωσής τους.

1.2 Στροφή προς τους Πολυπύρηνους Επεξεργαστές (Multicore Processors)

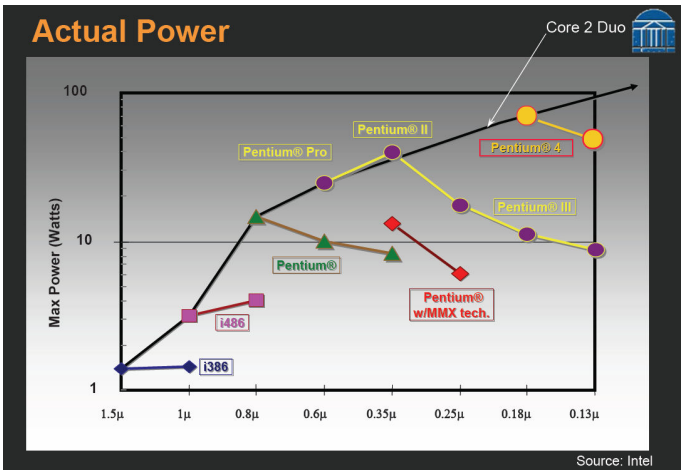
Για να κατανοηθεί καλύτερα γιατί έγινε η στροφή προς τους πολλαπλούς πυρήνες (multicore επεξεργαστές) πρέπει να γίνει ξεκάθαρο γιατί δεν μπορούσαν οι άλλοι τρόποι αύξησης της επίδοσης να χρησιμοποιούνται και να αυξάνονται επ' αόριστο [6]. Σε αυτό το σημείο κρίνεται σκόπιμο να υπενθυμιστεί ο νόμος του Moore. Αυτός ορίζει ότι, μπορούμε να διπλασιάζουμε τον αριθμό των τρανζίστορ που «χωράνε» σε ένα ολοκληρωμένο κύκλωμα κάθε 18 μήνες, κάτι που ενώ είχε ειπωθεί πολλά χρόνια πριν, ισχύει μέχρι και σήμερα και είναι κάτι που το εκμεταλλεύεται συνεχώς η Αρχιτεκτονική Υπολογιστών ώστε να πετυχαίνει συνεχόμενη αύξηση της επίδοσης των επεξεργαστών. Αυτό φαίνεται και στο Σχήμα 1-4 (πράσινη καμπύλη).



Σχήμα 1-4 Ο νόμος του Moore και η αύξηση της επίδοσης των υπολογιστικών συστημάτων του εμπορίου ανά συνιστώσα.

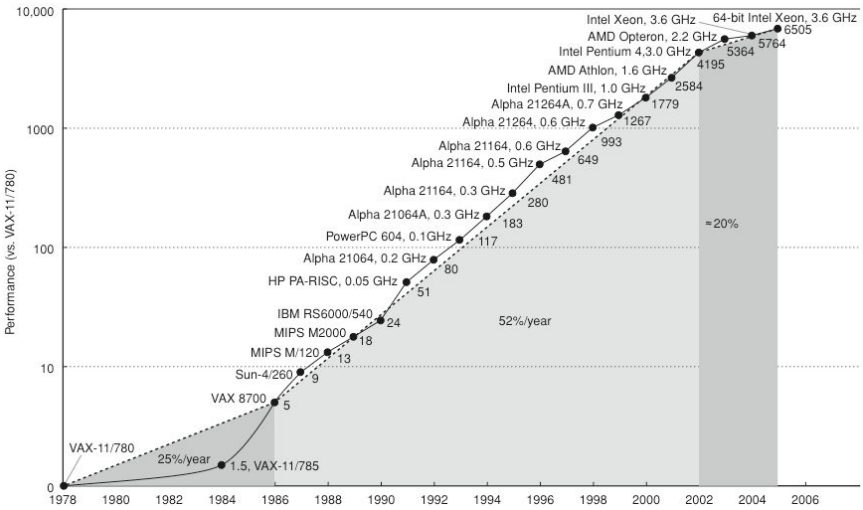
Οι άλλες τρεις καμπύλες του Σχήματος 1-4 δείχνουν την αύξηση της επίδοσης των επεξεργαστών ανάλογα με άλλους παράγοντες όπως ο παραλληλισμός σε επίπεδο εντολών, η συχνότητα του ρολογιού και η ισχύς. Είναι εμφανής ο κορεσμός των τεχνικών αυτών όσον αφορά την αύξηση της επίδοσης των υπολογιστικών συστημάτων.

Το φαινόμενο αυτό παρουσιάστηκε όταν το μέγεθος της ψηφίδας έγινε μικρότερο των 90nm, οπότε και άρχισαν να παρουσιάζονται προβλήματα και περιορισμοί που είχαν να κάνουν με την έκλυση ισχύος. Αυτό οδήγησε στο να μην είναι δυνατή η περαιτέρω αύξηση της συχνότητας του ρολογιού των επεξεργαστών [7]. Στο Σχήμα 1-5 φαίνονται οι απαιτήσεις σε ισχύ πολλών οικογενειών επεξεργαστών μέχρι και τις μέρες μας.



Σχήμα 1-5 Απαιτήσεις σε ισχύ διαφόρων επεξεργαστών με την πάροδο του χρόνου και την τεχνολογία που χρησιμοποιούνταν για την κατασκευή των τσιπ τους.

Έτσι, παρατηρήθηκε μια στασιμότητα ως προς τις ταχύτητες των επεξεργαστών μετά το 2002 και οι σχεδιαστές άρχισαν να ψάχνουν αλλού για αύξηση της επίδοσης των συστημάτων τους. Όμως, όπως γίνεται φανερό και από το Σχήμα 1-6, όλες οι ως τότε γνωστές οδοί βελτίωσης της επίδοσης είχαν σχεδόν εξαντληθεί καθώς το κόστος θα ήταν απαγορευτικό για μαζική παραγωγή με περισσότερες βελτιώσεις σε αυτούς τους τομείς, με συνέπεια ο ρυθμός αύξησης της επίδοσης να μειωθεί από το 55%/χρόνο, όπου ήταν σχεδόν σταθερός για παραπάνω από δυο δεκαετίες, σε λιγότερο από 25%.



Σχήμα 1-6 Αύξηση της επίδοσης επεξεργαστών από το 1978 και μετά όπως μετράται από την SpecINT.

Αυτό, οδήγησε τους σχεδιαστές στην υιοθέτηση της ιδέας των πολλαπλών πυρήνων. Τα οφέλη από μια τέτοια στροφή ήταν αρκετά και κατέστησαν δυνατό να ξεφύγει

από τη στασιμότητα η επίδοση των επεξεργαστών. Κάποια από τα πλεονεκτήματα των πολλαπλών, πιο απλών πυρήνων, που αποτελούν από εδώ και στο εξής τους σύχρονους επεξεργαστές, ανήκουν στους εξής τομείς:

- *Ισχύς:* Με τον παραλληλισμό μπορούμε να έχουμε καλύτερη χρησιμοποίηση της διαθέσιμης ισχύος, καθώς πολλοί απλοί πυρήνες έχουν μεγαλύτερη επίδοση για το ίδιο εμβαδόν σε σχέση με έναν πιο σύνθετο και πολύπλοκο πυρήνα.
- *Κόστος Σχεδίασης:* Μπορούμε πολύ πιο εύκολα να σχεδιάσουμε και να προβλέψουμε τη συμπεριφορά απλών σχετικά πυρήνων που θα αποτελούν τα δομικά στοιχεία με τα οποία θα χτίσουμε τους νέους πολυπύρηνους (multicore) επεξεργαστές. Η μεγαλύτερη απλότητα κάνει τα νέα σχέδια επεξεργαστών πιο οικονομικά στο σχεδιασμό και στην παραγωγή, κάτι πολύ σημαντικό για τις εταιρείες παραγωγής επεξεργαστών.
- *Ανοχή σε ελαττώματα:* Έχοντας στον ίδιο επεξεργαστή μεγάλο αριθμό πυρήνων οι οποίοι κάνουν την ίδια δουλειά, μάς επιτρέπει εάν συμβεί κάτι σε κάποιον από αυτούς να μπορέσουμε να τον θέσουμε εκτός λειτουργίας χωρίς να μειωθεί δραματικά η απόδοση του. Κάτι τέτοιο είναι δυνατόν να εφαρμοστεί για παράδειγμα στον επεξεργαστή STI Cell, ο οποίος έχει οκτώ πυρήνες από καταστκεύης, αλλά στο Playstation 3 όπου χρησιμοποιείται είναι ενεργοποιημένοι μόνο οι έξι. Εάν συμβεί κάτι σε έναν από αυτούς που λειτουργούν θα μπορεί να τον αντικαταστήσει κάποιος από τους δυο που είναι αδρανείς [7].

1.3 Manycore εναντίον Multicore

Η λέξη «multicore» χαρακτηρίζει την τάση, η οποία υπάρχει σήμερα στο εμπόριο να διπλασιάζεται ο αριθμός των πυρήνων που περιλαμβάνει ο κάθε επεξεργαστής για γενική χρήση σε κάθε νέα γενιά. Πριν από περίπου 18 μήνες έγινε μια προσπάθεια στο πανεπιστήμιο του Berkeley στην California των Ηνωμένων Πολιτειών να προβλεφθεί το μέλλον της αρχιτεκτονικής και της αγοράς των επεξεργαστών. Ειδικοί από πολλούς τομείς της Επιστήμης των Υπολογιστών, της Ηλεκτρονικής και πολλών άλλων κλάδων έφτασαν μετά πό μια σειρά συναντήσεων στο συμπέρασμα ότι το μέλλον δε βρίσκεται στους «multicore» επεξεργαστές αλλά έκαναν μια περισσότερο τολμηρή πρόταση-πρόβλεψη: Τους επεξεργαστές «manycore» [6,7]. Η λέξη «manycore» αναφέρεται σε επεξεργαστές οι οποίοι θα ξεκινούν με δεκάδες ή εκατοντάδες πυρήνων και ο αριθμός των πυρήνων σε αυτούς θα αυξάνεται με γεωμετρικούς ρυθμούς κάθε επόμενη γενιά. Το κύριο επιχείρημα όσον αφορά το υλικό είναι ότι, πλέον, τα πλεονεκτήματα που προσφέρουν πολλοί απλοί πυρήνες στους τομείς της κατανάλωσης ισχύος και της επιφάνειας των ψηφίδων είναι πολύ σημαντικά και ίσως ο πιο προσιτός δρόμος για αύξηση της επίδοσης των υπολογισικών συστημάτων του παρόντος και του μέλλοντος.

Ούτε όμως η ιδέα του «manycore» είναι κάτι απόλυτα καινούργιο καθώς manycore επεξεργαστές χρησιμοποιούνται σε routers (π.χ. το Cisco CRS-1 περιλαμβάνει 188 πυρήνες της Tensilica) και διάφορες μικροηλεκτρονικές συσκευές όπως σύγχρονα κινητά τηλέφωνα (π.χ. το Razor της Motorola που περιλαμβάνει οκτώ πυρήνες της εταιρείας Tensilica [7]). Ακόμη, εδώ και αρκετό καιρό υπάρχουν στα περισσότερα σπίτια του κόσμου όπου υπάρχουν προσωπικοί υπολογιστές, κάρτες γραφικών με μεγάλο αριθμό πυρήνων η κάθε μια, ικανές για πολύ μεγάλες ταχύτητες επεξεργασίας γραφικών. Αυτό, είναι αποτέλεσμα της ιδιαίτερης άνθισης της βιομηχανία ηλεκτρονι-

κών παιχνιδιών τα τελευταία 20 χρόνια και, το κυριότερο, της μεγάλης κερδοφορίας της, ώστε να ωθήσει τις εταιρείες να επενδύσουν στη συνεχή βελτίωση των γραφικών μονάδων επεξεργασίας που εισάγουν στην αγορά. Αυτού του είδους οι επεξεργαστες είναι αυτό με το οποίο θα ασχοληθεί η παρούσα εργασία και ειδικότερα η χρήση τέτοιων επεξεργαστών σε γενικές εφαρμογές. Όμως πριν οδηγηθεί η συζήτηση εκεί, θα αναφερθούν κάποιοι παράγοντες οι οποίοι είναι ευνοϊκοί προς την κατεύθυνση των πολλών πυρήνων.

Η χρήση απλών πυρήνων χαμηλής συχνότητας πραγματοποιείται από τη βιομηχανία των ενσωματωμένων συστημάτων (embedded-computing industry), με σκοπό τη μείωση κατά το δυνατόν των απαιτήσεων σε ισχύ και τα κόστη σχεδίασης και κατασκευής. Οι σχεδιαστικοί στόχοι αυτής της βιομηχανίας ήταν αντίθετοι με τους στόχους της βιομηχανίας των υπολογιστικών συστημάτων αιχμής μέχρι πρόσφατα. Ωστόσο, κυρίως οι ανάγκες μείωσης των απαιτήσεων ισχύος άλλαξαν το σκηνικό και πλέον οι σχεδιαστές υπολογιστικών συστημάτων αιχμής στράφηκαν στη βιομηχανία ενσωματωμένων συστημάτων, η οποία είχε ιδιαίτερη ειδίκευση και εμπειρία στο σχεδιασμό απλών, μη σπάταλων σε ισχύ πυρήνων, προκειμένου να δανειστούν κάποια στοιχεία. Παρατηρείται δηλαδή μια σύγκλιση των δυο βιομηχανιών πλέον προς τον σχεδιασμό και κατασκευή ιδιαίτερα απλών πυρήνων με ολοένα αυξανόμενες δυνατότητες [7]. Βέβαια, πρέπει να τονίσουμε ότι ένα υπολογιστικό σύστημα θα αποτελείται από μεγάλο αριθμό τέτοιων πυρήνων ώστε να μπορεί να ξεπεράσει σε δυνατότητες τον μονοπύρηνο ή «ολιγοπύρηνο» (για την ακρίβεια της σημερινές γενιές multicore επεξεργαστών) ανταγωνισμό.

Ωστόσο, υπάρχουν κάποια σημεία που χρήζουν ιδιαίτερης προσοχής, τα σημαντικότερα από τα οποία είναι τα εξής [7]:

- *Δυσκολίες στα δίκτυα διασύνδεσης των πυρήνων:* Η ισορροπία των συστημάτων επεξεργασίας θα διαταραχθεί, και μια άλλη κύρια πηγή κόστους πλέον θα είναι το πως θα σχεδιαστούν και θα υλοποιηθούν τα δίκτυα διασύνδεσης των πολλών απλών πυρήνων. Αυτό θα συμβεί, επειδή διαφαίνεται αναγκαία η χρήση κάποιων πλήρως συνδεδεμένων τοπολογιών, οι οποίες είναι και πιο δύσκολες να υλοποιηθούν.
- *Διατάραξη ισορροπίας της αρχιτεκτονικής και ειδικά των συστημάτων μνήμης:* Το μέγεθος της κρυφής μνήμης κοντά σε κάθε πυρήνα θα πρέπει να μειωθεί αισθητά για λόγους κόστους και κατανάλωσης ισχύος καθώς θα είναι πλέον πολύ πιο ακριβή.
- *Δυσκολίες συγχρονισμού:* Αναμένεται να υπάρξουν προβλήματα συγχρονισμού, συνάφειας και συνέπειας μνήμης με δεδομένο τον πολύ μεγάλο αριθμό πυρήνων που θα είναι διαθέσιμοι σε κάθε μονάδα επεξεργασίας.
- *Ανάγκη για νέο προγραμματιστικό μοντέλο:* Για να μπορέσει κάποιος να εκμεταλλευτεί όλες τις δυνατότητες αυτών των διαφαινόμενων μελλοντικών υπολογιστικών συστημάτων πρέπει να ξέρει να τις προγραμματίζει. Ο τρόπος που προγραμματίζει η συντριπτική πλειοψηφία των προγραμματιστών αυτή τη στιγμή είναι βασισμένος σε ένα σειρικό μοντέλο, το οποίο δεν θα μπορεί να επωφεληθεί από τις δυνατότητες των manycore μηχανημάτων. Έτσι, προκύπτει η ανάγκη να αναπτυχθεί και να γίνει γνωστό και ευρέως αποδεκτό κάποιο κατάλληλο νέο προγραμματιστικό μοντέλο, κάτι που δεν είναι τόσο απλό.

1.4 Ευρέως διαδεδομένες Αρχιτεκτονικές Παράλληλων Επεξεργαστών σήμερα

Σε αυτήν την ενότητα θα αναφερθούν με συντομία τα συστήματα, εκτός της περιοχής του High Performance Computing, στα οποία βρίσκει εφαρμογή στις μέρες μας, η ιδέα των επεξεργαστών πολλαπλών πυρήνων κοιτάζοντας τρεις χαρακτηριστικές κατηγορίες μηχανημάτων που έχουν κυριαρχούν στην αγορά.

1.4.1 Πολυπύρηνες CPUs

Οι περισσότεροι πολυπύρηνοι επεξεργαστές του εμπορίου, βασίζονται στην ιδέα της συμμετρικής πολυεπεξεργασίας (SMP-Symmetric Multiprocessors). Σύμφωνα με αυτήν, τα επεξεργαστικά στοιχεία-πυρήνες του όλου συστήματος είναι ακριβώς όμοια μεταξύ τους και συνδέονται μέσω ενός κοινού διαύλου δεδομένων με την κύρια μνήμη του συστήματος. Τα τελευταία τσιπ επεξεργαστών περιέχουν στο ίδιο ολοκληρωμένο κύκλωμα τους επεξεργαστές, την κρυφή μνήμη και τον ελεγκτή πρόσβασης στον κοινό δίαυλο. Σε τέτοια συστήματα συνήθως ο κάθε πυρήνας έχει πολύ «κοντά» του μια κρυφή μνήμη επιπέδου 1, ενώ την κρυφή μνήμη επιπέδου 2 τη μοιράζονται πάνω από ένας επεξεργαστές. Υπάρχουν συστήματα όπου ο κάθε πυρήνας έχει και επιπέδου 2 αποκλειστική κρυφή μνήμη «κοντά» του, ενώ μοιράζεται με τους υπόλοιπους πυρήνες μια κρυφή μνήμη επιπέδου 3. Ο κάθε ένας από τους πυρήνες που αποτελούν το σύστημα υλοποιεί πολλές από τις αρχές που αναφέρθηκαν πιο πάνω (παραλληλισμός σε επίπεδο bit, παραλληλισμός σε επίπεδο εντολής, όσο το δυνατόν μεγαλύτερη συχνότητα ρολογιού, σωλήνωση με αρκετά στάδια, out-of-order εκτέλεση εντολών σε κάποιες περιπτώσεις και παράλληλη εκτέλεση νημάτων) με σκοπό την καλύτερη δυνατή επίδοση.

Αυτή τη στιγμή το μεγαλύτερο μερίδιο της αγοράς στους επεξεργαστές για προσωπικούς υπολογιστές κατέχουν οι εταιρείες Intel και AMD. Σε αυτήν την ενότητα θα παρουσιαστούν τα χαρακτηριστικά δυο αντιπροσώπων των αρχιτεκτονικών που λανσάρουν στο εμπόριο αυτές οι δυο εταιρείες. Επίσης, θα αναφερθεί και μία διαφορετική, πιο εξειδικευμένη πρόταση στην σύγχρονη αγορά, ο επεξεργαστή Niagara της Sun.

1.4.1.1 Οικογένεια επεξεργαστών Core2 της Intel

Η ονομασία Core 2 αναφέρεται σε μια σειρά επεξεργαστών της εταιρείας Intel με το σετ εντολών x86-64, που απευθύνονται σε κάθε είδους χρήστες. Προσφέρονται και μονοπύρηνες και διπύρηνες και τετραπύρηνες εκδόσεις από αυτήν την οικογένεια. Οι συχνότητες ρολογιού είναι χαμηλότερες από ότι οι τελευταίοι μονοπύρηνοι επεξεργαστές της εταιρείας, καθώς δόθηκε βάρος κατά τον σχεδιασμό πιο αποδοτικών δομικών στοιχείων καθώς και στην μείωση της κατανάλωσης ισχύος του κάθε πυρήνα. Οι επεξεργαστές αυτοί παρουσιάστηκαν τον Ιούλιο του 2006 στην αγορά και από τότε έχουν κυκλοφορήσει διάφορες γενιές οι οποίες παρουσιάζουν βελτιώσεις ή προσφέρουν ειδικές δυνατότητες [15]. Θα αναφερθούμε με συντομία στα χαρακτηριστικά της αρχιτεκτονικής, κρατώντας όμως στο μυαλό μας ότι οι ποικιλία των εκδόσεων που υπάρχει διαθέσιμη στο εμπόριο είναι εξαιρετικά μεγάλη και είναι δυνατόν να υπάρχουν αποκλίσεις από γενιά σε γενιά.

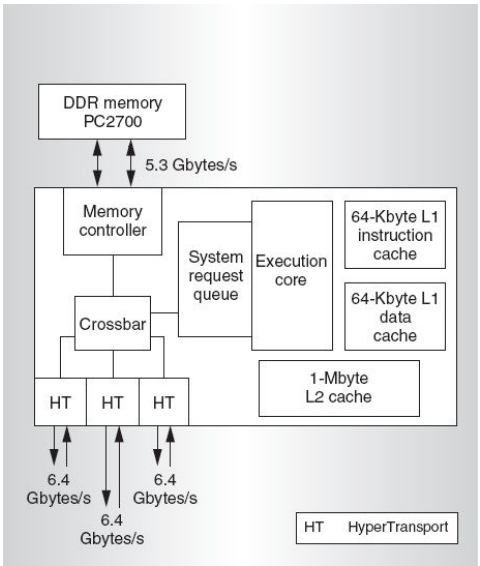
Η σωλήνωση αυτής της αρχιτεκτονικής περιλαμβάνει 14 στάδια. Σε κάθε πυρήνα υπάρχουν τρεις ΑΛΜ (Αριθμητικές και Λογικές Μονάδες) και τέσσερις αποκωδικοποιητές οι οποίοι μπορούν να αποκωδικοποιούν μέχρι και πέντε εντολές σε κάθε κύ-

κλο. Οι εντολές αναλύονται σε μικρολειτουργίες (όπως και σε όλες τις αρχιτεκτονικές x86-64, οι οποίες μπορούν να αναδιαταχθούν για καλύτερη επίδοση όπως είδαμε και στην περίπτωση του AMD Opteron (υπάρχει δηλαδή, δυνατότητα εκτέλεσης των μικρολειτουργιών εκτός σειράς). Σε κάθε κύκλο μπορούν να ολοκληρωθούν μέχρι και έξι μικρολειτουργίες ή να τερματίσουν τις λειτουργίες τους μέχρι και τέσσερις εντολές. Ο κάθε πυρήνας υλοποιεί εξελιγμένες τεχνικές πρόβλεψης διακλαδώσεων, ενώ υποστηρίζεται η τεχνολογία Hyperthreading Technology που αναλύθηκε στην ενότητα 1.1. Το σύστημα κρυφών μνημών περιλαμβάνει δυο επίπεδα ενώ ο δρόμος δεδομένων μεταξύ των δύο επιπέδων είναι πιο ευρύς (256 bit), με αποτέλεσμα την ταχύτερη επικοινωνία τους. Η κρυφή μνήμη επιπέδου 2 είναι διαμοιραζόμενη μεταξύ των δύο πυρήνων και μπορεί να έχει μέγεθος 4 MB ή 2 MB, ενώ υπάρχουν ξεχωριστές κρυφές μνήμες επιπέδου 1 για τις εντολές και τα δεδομένα (32 KB η κάθε μια συνήθως), οι οποίες είναι ξεχωριστές για κάθε πυρήνα. Η συχνότητα του ρολογιού ανάλογα με τη γενιά και το μοντέλο κυμαίνεται μεταξύ 1.06 και 3.2 GHz [14,15].

1.4.1.2 Επεξεργαστής Opteron της AMD

Ο επεξεργαστής της AMD με την ονομασία Opteron αποτελεί την πρόταση της εταιρείας για πολυπύρηνα υπολογιστικά συστήματα εξυπηρετητών, αλλά αποτελεί και τη βάση των επεξεργαστών που χρησιμοποιούνται σε προσωπικούς ή φορητούς υπολογιστές. Είναι ένας από τους πρώτους 64-bit επεξεργαστές του εμπορίου, βασιζόμενος στην αρχιτεκτονική x86-64 της AMD, όντας παράλληλα συμβατός με εφαρμογές που έχουν γραφτεί για τις 32-bit αρχιτεκτονικές, χωρίς να χρειάζεται να γίνει κάποια αλλαγή ή επαναμεταγλώττιση σε αυτές [2,10].

Στο Σχήμα 1-7 διακρίνονται τα βασικότερα στοιχεία αυτής της αρχιτεκτονικής.

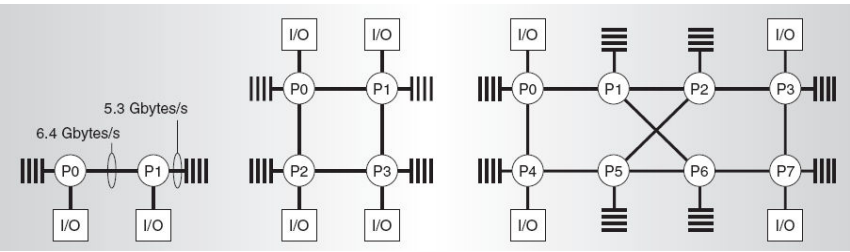


Σχήμα 1-7 Η αρχιτεκτονική του επεξεργαστή Opteron.

Παρατηρείται ότι υπάρχουν δυο ξεχωριστές κρυφές μνήμες επιπέδου 1 (μεγέθους 64 KB), η μια για εντολές και η άλλη για δεδομένα, που υποστηρίζονται από μια μεγάλη σχετικά κρυφή μνήμη επιπέδου 2. Σημαντικό στοιχείο αυτού του επεξεργαστή είναι ότι ο ελεγκτής μνήμης βρίσκεται πάνω στο τσιπ (αντί για την παραδοσιακή θέση του στο τσιπ Northbridge), κάτι που έχει αποτέλεσμα μια μείωση της καθυστέρησης πρόσβασης μνήμης (latency) της τάξης του 20% με 25% [10]. Σε αυτήν την αρχιτεκτονι-

κή (όπως και όλες οι αρχιτεκτονικές τύπου x86 και x86-64) οι εντολές μεταβλητού μήκους του σετ x86-64, μετατρέπονται σε μικρολειτουργίες οι οποίες με τη σειρά τους εκτελούνται. Ο Opteron είναι ένας υπέρ-βαθμωτός επεξεργαστής ο οποίος υποστηρίζει και εκτέλεση εντολών εκτός σειράς. Έτσι μπορεί να χρονοδρομολογήσει μέχρι και 72 μικρολειτουργίες, ενώ παράλληλα να στέλνει και εντολές φόρτωσης/αποθήκευσης ώστε δεδομένα που θα χρειαστούν στο κοντινό μέλλον να αναλάβει το σύστημα μνήμης να τα φέρει στην κρυφή μνήμη επιπέδου 1 όσο εκτελούνται άλλες μικρολειτουργίες. Περιέχει επίσης δυο ξεχωριστούς χρονοδρομολογητές μικρολειτουργιών έναν για πράξεις ακεραίων και έναν για πράξεις κινητής υποδιαστολής ή εντολών πολυμέσων. Σε κάθε κύκλο τελικά είναι δυνατόν ένας πυρήνας Opteron να εκτελεί μέχρι και 11 διαφορετικές μικρολειτουργίες αρκεί να είναι μίγμα μικρολειτουργιών που έχουν να κάνουν με ακεραίους (μέχρι τρεις), με υπολογισμούς διεύθυνσης (μέχρι τρεις), με αριθμούς κινητής υποδιαστολής (μέχρι τρεις) και με αιτήσεις αποθήκευσης/φόρτωσης από τη μνήμη (μέχρι δυο). Η σωλήνωση του Opteron έχει από 12 στάδια (για ακεραίους), μέχρι 17 στάδια (για πράξεις με αριθμούς κινητής υποδιαστολής), αρκετά μεγάλο δηλαδή για την υψηλή συχνότητα στην οποία λειτουργεί, αλλά και αρκετά μικρό για να γίνεται καλή χρήση του παραλληλισμού σε επίπεδο εντολών.

Όλα τα παραπάνω όμως αφορούν την αρχιτεκτονική όχι μόνο ενός πυρήνα αλλά και ενός πολυπύρηνου υπολογιστικού συστήματος. Ο σύνδεσμος HT (HyperTransport) που διακρίνονται στο Σχήμα 1-7 αποτελεί τη δίοδο επικοινωνίας με άλλους πυρήνες Opteron, χωρίς να χρειάζεται κάποιο επιπλέον κύκλωμα για τη σύνδεσή τους, με ταχύτητες που φτάνουν τα 6.4 GB/s. Στο Σχήμα 1-8 φαίνονται κάποιες προτεινόμενες τοπολογίες για διπύρηνα, τετραπύρηνα ή οκταπύρηνα υπολογιστικά συστήματα εξυπηρετητών.



Σχήμα 1-8 Τοπολογίες πολυπύρηνων συστημάτων με βάση τον Opteron. Διακρίνουμε τις συνδέσεις με άλλους πυρήνες (σύνδεσμοι ταχύτητας 6.4 GB/s), με μνήμες (σύνδεσμοι ταχύτητας 5.3 GB/s) και με συσκευές εισόδου/εξόδου (I/O).

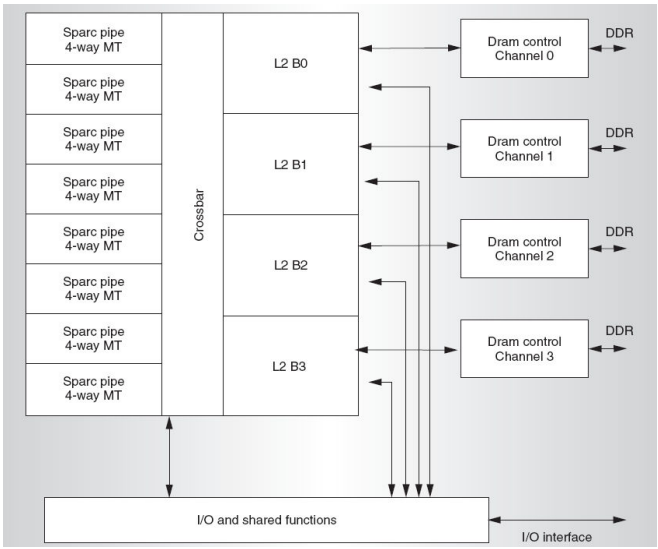
Τέλος, θα πρέπει να αναφερθεί ότι επιτρέπεται στο κάθε επεξεργαστή να έχει την δική του κύρια μνήμη και ότι δεν υπάρχει ενιαία διευθυνσιοδότηση του συστήματος μνήμης, αν και κάθε επεξεργαστής μπορεί να έχει πρόσβαση σε οποιαδήποτε κύρια μνήμη. Όσον αφορά στον προγραμματιστή, όλοι οι χώροι μνήμης φαίνονται ενιαίοι, δηλαδή ο Opteron αποτελεί παράδειγμα επεξεργαστή αρχιτεκτονικής NUMA (Non–Unified Memory Access – Μη Ενοποιημένης Πρόσβασης στη Μνήμη) [11,12].

1.4.1.3 Επεξεργαστής Niagara της Sun Microsystems

Η πρόταση της εταιρείας Sun στον τομέα της παράλληλης επεξεργασίας είναι ο επεξεργαστής Niagara (κωδική ονομασία του επεξεργαστή UltraSPARC T1), ο οποίος παρουσιάστηκε το 2005 [13]. Στόχος της αρχιτεκτονικής αυτού του επεξεργαστή ή-

ταν η μέγιστη δυνατή επίδοση σε πολυνηματικές εφαρμογές (σε εξυπηρετητές του Web, στον τομέα των βάσεων δεδομένων και αλλού). Αυτός ο επεξεργαστής έχει τη δυνατότητα να εκτελεί παράλληλα 32 διαφορετικά νήματα (σε επίπεδο υλικού) με ιδιαίτερα υψηλές επιδόσεις και όχι πολύ μεγάλες αιτήσεις σε ισχύ (λειτουργεί στα 1.4 GHz με 72 W ισχύος) [13]. Οι εφαρμογές βλέπουν τον επεξεργαστή αυτόν σαν να αποτελείται από 32 διαφορετικοί πυρήνες επεξεργασίας, κάθε ένας από τους οποίους μπορεί να αναλάβει την εκτέλεση ενός νήματος.

Η αρχιτεκτονική του συγκεκριμένου επεξεργαστή μπορεί να εξηγηθεί με τη βοήθεια του Σχήματος 1-9.



Σχήμα 1-9 Μπλοκ διάγραμμα του επεξεργαστή Niagara της Sun Microsystems.

Ο Niagara, ουσιαστικά, αποτελείται από οκτώ πυρήνες (Sparc pipe στο Σχήμα 1-9), καθένας εκ των οποίων μπορεί να εκτελέσει μέχρι τέσσερα νήματα υλικού με την τεχνολογία του ταυτόχρονου πολυνηματισμού. Κάθε τέτοιος πυρήνας περιλαμβάνει δυο κρυφές μνήμες επιπέδου 1 για εντολές (μεγέθους 16 KB) και δεδομένα (μεγέθους 8 KB), τις οποίες μοιράζονται τα τέσσερα νήματα. Η σωλήνωση εδώ αποτελείται από έξι βασικά στάδια: προσκόμιση εντολής, επιλογή νήματος, αποκωδικοποίηση, εκτέλεση, στάδιο προσπέλασης μνήμης και εγγραφή-πίσω/write back. Με την ομαδοποίηση των νημάτων ανά τετράδες (και της ανάθεσης κάθε μίας από τις τετράδες σε έναν πυρήνα), μπορούν να κρυφτούν οι καθυστερήσεις κατά την πρόσβαση της μνήμης ή στη σωλήνωση του ενός νήματος, καθώς πάντα θα υπάρχουν εντολές προς εκτέλεση ενός τουλάχιστον από τα τέσσερα νήματα. Η χρονοδρομολόγηση των νημάτων σε κάθε πυρήνα γίνεται με μηδενικό κόστος [5].

Όλα τα νήματα μοιράζονται μια κρυφή μνήμη επιπέδου 2 μεγέθους 3 MB, η οποία είναι οργανωμένη σε τέσσερις τράπεζες/banks κάθε μια από τις οποίες συνδέεται με έναν κανάλι επικοινωνίας με τη μνήμη ή με κάποια περιφερειακή συσκευή. Η επικοινωνία των πυρήνων με την κρυφή μνήμη επιπέδου 2, τα περιφερειακά και τους άλλους διαμοιραζόμενους πόρους του επεξεργαστή γίνονται με ένα δίκτυο διασύνδεσης Crossbar, το οποίο προσφέρει μέχρι και 200 GB/s εύρος ζώνης [5].

Ο Niagara είναι ο πρώτος μιας σειράς επεξεργαστών της Sun που στοχεύουν στην παράλληλη επεξεργασία νημάτων σε επίπεδο υλικού σε εμπορικά συστήματα. Οι

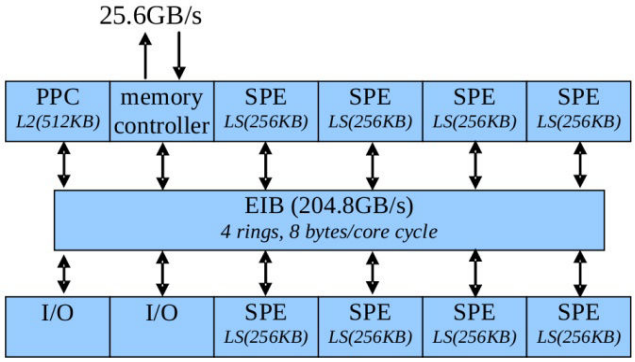
σύγχρονες εφαρμογές του Διαδικτύου αποτελούν την αγορά αυτών των επεξεργαστών, καθώς βελτιώνουν σημαντικά την ρυθμοαπόδοση τέτοιου είδους εφαρμογών με αρκετά χαμηλή κατανάλωση σε ισχύ.

1.4.2 STI Cell Processor

Ο επεξεργαστής Cell (Cell Broadband Engine Architecture – Cell BE ή CBEA) αποτελεί δημιούργημα της σύμπραξης τριών εταιριών, των Sony, Toshiba και IBM (εξ'ου και η συντομογραφία STI) στο πλαίσιο ενός προγράμματος που ξεκίνησε το 2000 [8]. Είναι ο επεξεργαστής ο οποίος αποτελεί την καρδιά εκατομμυρίων συσκευών Playstation 3 ανά τον κόσμο. Δημιουργήθηκε με αρχικό σκοπό να γεφυρώσει τις διαφορές και να συνδυάσει τα πλεονεκτήματα των –πιο εξεζητημένων και με μεγάλη επίδοση σε θέματα επεξεργασίας γραφικών ή μεγάλων πακέτων δεδομένων– Γραφικών Μονάδων Επεξεργασίας (Graphics processing Units – GPUs) με τις –πιο γενικού σκοπού– πολυπύρηνες σύγχρονες CPUs και με στόχο την καλύτερη επίδοση έναντι των προσωπικών υπολογιστών που θα έβγαιναν στην αγορά μέσα στο 2005. Θα γίνει μια σύντομη αναφορά στα κυριότερα σημεία της αρχιτεκτονικής του.

Ο Cell είναι ένας ετερογενής πολυπύρηνος επεξεργαστής, δηλαδή περιέχει πολλούς πυρήνες (συνολικά εννέα) δυο διαφορετικών ειδών: Έναν επεξεργαστή 64-bit γενικής χρήσης με υψηλές επιδόσεις (POWER Processing Element – PPE) και οκτώ Single Instruction Multiple Data (SIMD) επεξεργαστικά στοιχεία-πυρήνες (Synergistic Processing Elements – SPE). Ο PPE συνήθως αφιερώνεται στο να εκτελεί το λειτουργικό σύστημα, ενώ τα SPEs αναλαμβάνουν να εκτελέσουν τους απαιτητικούς υπολογισμούς.

Στο Σχήμα 1-10 διακρίνεται ένα απλοϊκά σχεδιασμένο μπλοκ διάγραμμα του επεξεργαστή Cell.



Σχήμα 1-10 Μπλοκ διάγραμμα του επεξεργαστή Cell.

Ο PPE περιλαμβάνει μια ιεραρχία κρυφών μνημών δυο επιπέδων, η οποία περιλαμβάνει μια κρυφή μνήμη επιπέδου 1 μικρού μεγέθους και μια κρυφή μνήμη επιπέδου 2 μεγέθους 512 KB, η οποία επικοινωνεί μέσω του EIB (Element Interconnect Bus) με τα υπόλοιπα υποσυστήματα μνήμης με ταχύτητα που φτάνει τα 51.2 GB/s. Με την ίδια ταχύτητα επικοινωνούν και οι Τοπικές Μνήμες (Local Storage – LS) των SPEs (μεγέθους 256 KB η κάθε μια) μέσω του EIB. Η επικοινωνία με την κύρια μνήμη γίνεται μέσω ενός ελεγκτή μνήμης (memory controller) με ταχύτητα που φτάνει τα 25.6 GB/s.

Ο PPE είναι ένας dual-issue και in-order επεξεργαστής ο οποίος υποστηρίζει και dual-threading. Σκοπός του είναι η βελτιστοποίηση των λόγων επίδοσης/εμβαδό chip και επίδοσης/απαιτούμενη ισχύ. Εκτός από την εκτέλεση των λειτουργιών του λειτουργικού συστήματος του όλου συστήματος, καθήκον του PPE είναι ο συγχρονισμός και ο έλεγχος της λειτουργίας των SPEs καθώς και ο διαμοιρασμός στα SPEs των πόρων και των δεδομένων πάνω στα οποία θα επενεργήσουν. Η συχνότητα ρολογιού του είναι 3.2 GHz [8,9].

Οι SPEs, όπως αναφέραμε, αναλαμβάνουν το φορτίο της εκτέλεσης των απαιτητικών σε υπολογισμούς καθηκόντων. Ο κάθε SPE είναι dual-issue και in-order πυρήνας και περιλαμβάνει ένα αρχείο καταχωρητών 128-bit μεγέθους 128 θέσεων. Περιλαμβάνει μια μονάδα επεξεργασίας (Synergistic Processing Unit – SPU) και έναν ελεγκτή μνήμης (Memory Flow Controller – MFC) για την επικοινωνία με τα υπόλοιπα υποσυστήματα μνήμης. Η συχνότητα ρολογιού είναι 3.2 GHz, ενώ το θεωρητικό μέγιστο σε ταχύτητα πράξεων κινητής υποδιαστολής μονής ακρίβειας είναι τα 25.6 GFlop/s [8,9].

Όλοι οι επεξεργαστές του τσιπ του Cell έχουν πρόσβαση σε όλο το σύστημα μνήμης, το οποίο περιλαμβάνει την κύρια μνήμη, τα αρχεία καταχωρητών όλων των SPEs, του καταχωρητές ελέγχου και τις συσκευές εισόδου/εξόδου [8]. Η επικοινωνία αυτή γίνεται μέσω μεταφορών Άμεσης Πρόσβασης στη Μνήμη (Direct Memory Access – DMA).

Τέλος, ο δίαυλος διασυνδεσης EIB παίζει πολύ σημαντικό ρόλο στη λειτουργία του Cell καθώς είναι το μέσο επικοινωνίας μεταξύ των πυρήνων, της κεντρικής μνήμης και των συσκευών εισόδου/εξόδου. Περιλαμβάνει, όπως είναι φανερό και από το Σχήμα 1-10, 4 δακτύλιους για τη μεταφορά δεδομένων με θεωρητικό μεγιστο όσον αφορά τη μεταφορά δεδομένων τα 204.8 GB/s [9].

Σχεδιαστικά, στόχος του Cell ήταν σε ένα συγκεκριμένο εμβαδό τσιπ, να «χωρέσουν» όσο το δυνατόν περισσότεροι πυρήνες ώστε να παρουσιάζει ιδιαίτερα υψηλές επιδόσεις σε σύγχρονες εφαρμογές απαιτητικές σε υπολογισμούς δεδομένων καθώς και να υποστηρίζεται ο παραλληλισμός σε επίπεδο νημάτων. Αν οι αρχιτέκτονες των τριών εταιρειών έδιναν περισσότερη έμφαση στην αύξηση της συχνότητας, στην υποστήριξη περισσότερων σταδίων σωλήνωσης ή σε άλλες τεχνικές βελτιστοποίησης της απόδοσης των πυρήνων, θα μπορούσαν να καταλήξουν σε ισχυρότερους πυρήνες. Τότε όμως, ο αριθμός τους πάνω στο τσιπ θα ήταν αρκετά μικρότερος, κάτι μη επιθυμητό για τις εφαρμογές που στοχεύει η αρχιτεκτονική του Cell [8].

1.4.3 Επεξεργαστές Γραφικών (GPUs)

Μια από της αρχιτεκτονικές που έχει πλέον συγκεντρώσει μεγάλο εμπορικό, ερευνητικό και επιστημονικό ενδιαφέρον είναι οι Μονάδες Επεξεργασίας Γραφικών (Graphic Processing Units – GPUs). Τα τελευταία χρόνια έχουν εισαχθεί στην αγορά, από τις δύο κυρίαρχες σε αυτόν τον τομέα εταιρείες nVidia και ATI (πρόσφατα αγορασμένη από την AMD), μοντέλα καρτών με εξαιρετικές δυνατότητες, ενώ έχει καταστεί δυνατό ένας προσωπικός υπολογιστής να μπορεί να φιλοξενήσει παραπάνω της μιας κάρτες γραφικών, εισάγοντας έτσι άλλο ένα επίπεδο παραλληλισμού. Στο επόμενο κεφάλαιο θα εξεταστούν αυτές οι αρχιτεκτονικές και στη χρήση τους και σε

εφαρμογές γενικού σκοπού, εκτός από την απεικόνιση εικόνων υψηλής ευκρίνειας. Αυτή η νέα χρήση των καρτών γραφικών αποτελεί έναν πρόσφατα και ταχέως αναπτυσσόμενο κλάδο της Επιστήμης Υπολογιστών που έχει την ευρέως αποδεκτή ονομασία GPGPU (General Purpose computations on GPUs). Με την τελευταία γενιά καρτών της nVidia (σειρά 8) και την αρχιτεκτονική που αυτές υποστηρίζουν (G80), με την οποία και θα ασχοληθεί ενδελεχώς αυτή η εργασία, αυτός ο κλάδος έχει καταστεί πολύ πιο προσιτός προς ευρεία χρήση και εκμετάλλευση.

1.5 Κάποια πρώτα συμπεράσματα

Ολοκληρώνοντας τη σύντομη αυτή περιγραφή στην Αρχιτεκτονική των Υπολογιστών σήμερα, εξήχθη το συμπέρασμα ότι το μέλλον βρίσκεται στους πολυπύρηνους επεξεργαστές, καθώς αποτελούν τον πλέον ενδεδειγμένο τρόπο για να συνεχιστεί με γοργούς ρυθμούς η αύξηση της απόδοσης στα υπολογιστικά συστήματα του μέλλοντος. Οι τρόποι αύξησης της επίδοσης ενός μοναδικού επεξεργαστικού πυρήνα έχουν πλέον εξαντληθεί, και επομένως η βιομηχανία επεξεργαστών έχει ήδη στραφεί σε πολυπύρηνες λύσεις. Ενδεικτικό αυτής της ισχυρής τάσης προς τους επεξεργαστές πολλών πυρήνων είναι η άποψη του Chris Rowen της εταιρείας Tensilica, η οποία συνοψίζεται στο εξής [7]:

«Ο επεξεργαστής(πυρήνας) είναι το νέο transistor.»

Ακόμη, έγινε αναφορά της σύγκλισης των αγορών μικροηλεκτρονικών ή ενσωματωμένων συσκευών με αυτή των υπολογιστών αιχμής που βασίζονται στη λειτουργία μικροεπεξεργαστών, οι οποίοι μπορούν να γίνουν οι δομικές μονάδες που θα απαρτίζουν ισχυρά συστήματα πολυεπεξεργασίας. Είναι, πλέον, φανερό ότι τα περισσότερα πεδία όπου χρησιμοποιούνται επεξεργαστές, συμπεριλαμβανομένων των φορητών ηλεκτρονικών συσκευών και των μηχανημάτων του κλάδου του High Performance Computing, οδηγούνται πλέον σε σύγκλιση με κοινό παρονομαστή την τάση προς τις αρχιτεκτονικές πολλών πυρήνων [7].

1.6 Σκοπός και δομή της διπλωματικής εργασίας

Πολλές από τις αρχές που θα διέπουν τους «manycore» επεξεργαστές του μέλλοντος βρίσκουν εφαρμογή στις κάρτες γραφικών τελευταίων γενεών κάτι που τις καθιστά μια οικονομική και θελκτική λύση για τη χρήση τους σαν πιθανές πλατφόρμες επίλυσης διαφόρων προβλημάτων επιστημονικής φύσης. Η παρούσα διπλωματική εργασία, πραγματεύεται αυτό το ζήτημα και σκοπός της είναι να αποφανθεί το κατά πόσο αποδοτική είναι η χρήση της τελευταίας γενιάς GPUs στην επίλυση προβλημάτων γενικού σκοπού. Επίσης, θα γίνει προσπάθεια να παρουσιαστούν τα οφέλη της στροφής προς μαζικά πολυπύρηνους επεξεργαστές με αρκετά γενικά χαρακτηριστικά, όπως είναι αυτοί της αρχιτεκτονικής G80 της εταιρείας nVidia.

Αρχικά, στο Κεφάλαιο 2, θα παρουσιαστεί η ιστορία των καρτών γραφικών και του τρόπου προγραμματισμού τους μέχρι σήμερα, ενώ θα γίνει αναφορά και στις προσπάθειες που έχουν γίνει για προγραμματισμό τους σε εφαρμογές γενικού σκοπού. Στο Κεφάλαιο 3, θα παρουσιαστεί ενδελεχώς η αρχιτεκτονική G80 των καρτών γραφικών και η διεπαφή προγραμματισμού τους (CUDA), η τεχνολογία δηλαδή που υ-

ποστηρίζουν οι κάρτες γραφικών της σειράς 8 της εταιρείας nVidia για προγραμματισμό σε εφαρμογές πέρα από τα γραφικά. Στο Κεφάλαιο 4, θα διερευνηθούν οι στρατηγικές βελτιστοποίησης των προγραμμάτων που αναπτύσσονται πάνω σε αυτήν τη νέα πλατφόρμα. Στο Κεφάλαιο 5, θα χρησιμοποιηθούν οι γνώσεις των προηγούμενων κεφαλαίων για να υλοποιήσουμε με τον πλέον αποδοτικό τρόπο κάποιοι στοιχειώδεις πυρήνες, που αποτελούν μέρος πλήθους πραγματικών εφαρμογών, με σκοπό την αξιολόγηση της αρχιτεκτονικής G80 για υπολογισμούς γενικού σκοπού. Τέλος, στο Κεφάλαιο 6, θα γίνει μια σύνοψη των συμπερασμάτων και θα εξερευνηθούν μελλοντικές επεκτάσεις και προοπτικές.

Κεφάλαιο 2

Υπολογισμοί γενικού σκοπού σε Μονάδες Επεξεργασίας Γραφικών (GPGPU)

Ο όρος «GPGPU» (General Purpose computing on Graphics Processing Units) αναφέρεται σε μια προσπάθεια εκμετάλλευσης των δυνατοτήτων των μαζικά παράλληλων πολυπύρηνων καρτών γραφικών, που κυκλοφορούν ευρέως στο εμπόριο, για γενικούς σκοπούς και υπολογισμούς. Με την πάροδο του χρόνου και τις εξελίξεις στον τρόπο προγραμματισμού και χειρισμού των καρτών γραφικών, κατέστη δυνατή η χρήση αυτών των ισχυρών πολυεπεξεργαστών σε κάτι πέρα από τους στόχους για τους οποίους είχαν αρχικά σχεδιαστεί. Αποτελούν πανίσχυρους συνεπεξεργαστές της CPU στην επίλυση ποικίλων προβλημάτων, ιδιαίτερα απαιτητικών σε αριθμητικούς υπολογισμούς. Το επιστημονικό ενδιαφέρον εδώ και σχεδόν μια δεκαετία έχει στραφεί προς αυτήν την κατεύθυνση, λόγω κάποιων αλλαγών στον τρόπο που προγραμματίζονται, πλέον, οι GPUs. Οι σύγχρονες GPUs παρουσιάζουν σημαντικά μεγάλη επίδοση όσον αφορά τις ταχύτητες επεξεργασίας και εύρους ζώνης μνήμης που προσφέρουν, σε σχέση με τις CPUs που κυκλοφορούν στην αγορά σήμερα.

Στο παρόν κεφάλαιο αρχικά θα γίνει αναφορά στον τρόπο χειρισμού των GPUs για τον πρωταρχικό σκοπό για τον οποίο σχεδιάστηκαν, την επεξεργασία γραφικών. Έπειτα, θα παρουσιαστεί μια αναδρομή των σημαντικότερων γενεών των καρτών γραφικών, των χαρακτηριστικών του υλικού τους και του τρόπου προγραμματισμού τους

μέχρι την έλευση της σειράς 8 της nVidia. Στη συνέχεια, θα γίνει αναφορά στους λόγους για τους οποίους οι Μονάδες Επεξεργασίας Γραφικών χρησιμοποιήθηκαν για υπολογισμούς γενικού σκοπού, ενώ θα αναφερθούν και οι τρόποι με τους οποίους έγινε κάτι τέτοιο. Αφού δοθούν παραδείγματα εφαρμογών και επιστημονικών τομέων στις οποίες έγινε ευρεία και αποτελεσματική χρήση αυτού του τρόπου προγραμματισμού, θα εξαχθούν συμπεράσματα για τα προβλήματα πού έχουν οι κάρτες γραφικών σαν εναλλακτικές πλατφόρμες μέχρι σήμερα, ώστε να έχουμε ένα μέτρο σύγκρισης με την τελευταία γενιά γραφικών, έναν αντιπρόσωπο της οποίας θα μελετήσουμε στα επόμενα κεφάλαια.

2.1 Μονάδες Επεξεργασίας Γραφικών (GPUs): Το Πού

Σε αυτήν την ενότητα θα παρουσιαστεί η εξέλιξη του υλικού που έχει να κάνει με την επεξεργασία ή την επιτάχυνση των γραφικών στους προσωπικούς υπολογιστές. Πρώτα θα πρέπει να ξεκαθαριστεί η διαφορά μεταξύ των Επιταχυντών Γραφικών (Graphics Accelerators) και των Μονάδων Επεξεργασίας Γραφικών (Graphics Processing Units). Οι πρώτοι ήταν απλά επιταχυντές των υπολογισμών που είχαν να κάνουν με την απεικόνιση γραφικών στην οθόνη του υπολογιστή, όπως δηλώνει ξεκάθαρα και η ονομασία τους. Δηλαδή, αν δεν υπήρχαν καθόλου, το τελικό αποτέλεσμα στο εκάστοτε μέσο απεικόνισης θα ήταν ακριβώς το ίδιο, απλά χωρίς αυτούς η ταχύτητα των υπολογισμών θα ήταν μικρότερη μιας και η CPU θα επωμιζόταν πολύ μεγαλύτερο βάρος υπολογισμών. Όμως, με την πάροδο του χρόνου και την έλευση των GPUs ο τρόπος χειρισμού των γραφικών άλλαξε άρδην. Οι GPUs, πλέον, δεν επιτάχυναν απλά τους υπολογισμούς αλλά επεξεργάζονταν εξ' ολοκλήρου ό,τι είχε σχέση με την απεικόνιση σκηνών στην οθόνη. Τα στάδια και η διαδοχή της σωλήνωσης των γραφικών (graphics pipeline) είχαν μείνει ανέγγιχτα για περίπου 20 χρόνια μέχρι τη διαδοχή των Επιταχυντών Γραφικών από τις Μονάδες Επεξεργασίας Γραφικών. Τώρα πλέον η σωλήνωση αυτή έχει αλλάξει, και κάποια από τα στάδια της μπορούν πλέον να προγραμματιστούν.

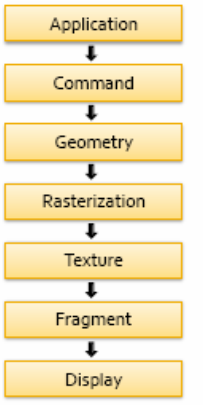
2.1.1 Εξέλιξη της σωλήνωσης των γραφικών (Graphics Pipeline)

Η σωλήνωση γραφικών είναι κάτι ανάλογο με τη σωλήνωση στις CPUs. Οι λειτουργίες δηλαδή που πρέπει να εκτελεστούν προκειμένου να γίνει η επεξεργασία των γραφικών χωρίζονται σε στάδια (στάδια της σωλήνωσης) και ανά πάσα στιγμή σε κάθε στάδιο μπορεί να βρίσκεται μια διαφορετική σειρά λειτουργιών (ακριβώς όπως διαφορετικές εντολές στις CPUs βρίσκονται σε διαφορετικό στάδιο της εκτέλεσής τους). Η είσοδος αυτής της σωλήνωσης είναι οι κορυφές και τα πρωτογενή γεωμετρικά χαρακτηριστικά των αντικειμένων προς απεικόνιση, οι συνθήκες φωτισμού της κάθε σκηνής, επιθυμητοί μετασχηματισμοί και άλλα χαρακτηριστικά που επηρεάζουν την σκηνή προς απεικόνιση και είναι απαραίτητα για το ρεαλιστικότερο δυνατό αποτέλεσμα . Η έξοδος της σωλήνωσης είναι μια δισδιάστατη εικόνα προς απεικόνιση σε κάποιο μέσο εξόδου, συνήθως μια οθόνη.

Τα δυο κυρίαρχα πρότυπα σωλήνωσης γραφικών είναι τα Direct3D και OpenGL. Το Direct3D αποτελεί μέρος του API (Application Programming Interface-Περιβάλλοντος Προγραμματισμού Εφαρμογών) του DirectX (συλλογή APIs σχετικών με επεξεργασία και αναπαραγωγή πολυμέσων της εταιρείας Microsoft. Περιλαμβάνει, δηλαδή, εντολές για την πραγματοποίηση διαφόρων λειτουργιών σχετικών με

την επεξεργασία τρισδιάστατων εικόνων. Είναι συμβατό και διαθέσιμο με τα διάφορα λειτουργικά συστήματα της οικογένειας Windows της εταιρείας Microsoft. Το Direct3D χρησιμοποιείται για την επεξεργασία γραφικών σε εφαρμογές με μεγάλες απαιτήσεις (παιχνίδια, επαγγελματική επεξεργασία εικόνας και βίντεο) [18,19]. Το OpenGL είναι ένα πρότυπο που ορίζει μια, ανεξάρτητη πλατφόρμας, προγραμματιστική διεπαφή για τη δημιουργία δισδιάστατων και τρισδιάστατων εικόνων. Αποτελείται από περισσότερες από 250 κλήσεις συναρτήσεων δημιουργίας και σχεδίασης απλών μέχρι και ιδιαίτερα σύνθετων αντικειμένων και σκηνών. Η κάθε εταιρεία υλοποιεί αυτές τις συναρτήσεις διαφορετικά, ακολουθώντας όμως τις αρχές τις οποίες ορίζουν οι προδιαγραφές του OpenGL. Δημιουργήθηκε από την εταιρεία Silicon Graphics Inc. (SGI) το 1992 και από τότε χρησιμοποιείται ευρέως σε πληθώρα εφαρμογών (από επιστημονικές εφαρμογές και εξομοιωτές πτήσεων μέχρι και παιχνίδια) [16,17].

Στο Σχήμα 2-1 φαίνεται το κλασικό μοντέλο σωλήνωσης γραφικών. Θα αναφερθεί με συντομία ο ρόλος κάθε σταδίου στην διαδικασία της επεξεργασίας των γραφικών και στη συνέχεια θα παρουσιαστεί η εξέλιξή της σωλήνωσης με την πάροδο του χρόνου.



Σχήμα 2-1 Κλασική σωλήνωση γραφικών.

Το κάθε στάδιο είναι υπεύθυνο για τα εξής [20]:

- *Εφαρμογή (Application):* Η εφαρμογή προετοιμάζει και φορτώνει στην GPU τα δεδομένα προς επεξεργασία και, μέσω του υποστηριζόμενου προγραμματιστικού μοντέλου (OpenGL, Direct3D), εκδίδει τις εντολές.
- *Εντολές Command ():* Οι ενέργειες που λαμβάνουν χώρα κατά αυτό το στάδιο έχουν να κάνουν με την τριγωνοποίηση των πολυγώνων και την προετοιμασία της ροής των δεδομένων των κορυφών (vertex data streams).
- *Γεωμετρία (Geometry):* Στο στάδιο αυτό γίνεται η επεξεργασία των κορυφών πραγματοποιώντας μετασχηματισμούς προοπτικής και οπτικής (perspective and viewing transformation) και κάποιους υπολογισμούς για τον φωτισμό.
- *Μετατροπή εικόνας σε εικονοστοιχεία (Rasterization):* Εδώ η εικόνα προς προβολή μετατρέπεται σε εικονοστοιχεία (pixels) μετά την επεξεργασία των τριγώνων και των κορυφών που έχουν προκύψει από προγενέστερους υπολογισμούς.
- *Απεικόνιση υφής (Texture mapping):* Σε αυτό το στάδιο επιτυγχάνεται η προσθήκη ρεαλισμού και λεπτομερειών στηνεικόνα προς απεικόνιση με κατάλληλο φιλτράρισμα και χρήση της μνήμης. Αποτέλεσμα αυτού του σταδίου είναι

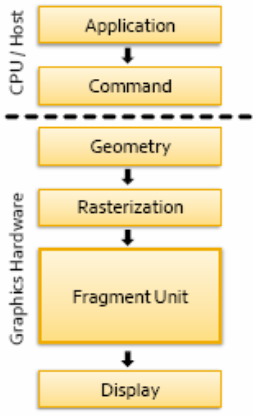
τα τμήματα(fragments). Τα τμήματα είναι όλα εκείνα τα στοιχεία και χαρακτηριστικά που είναι απαραίτητα για τον πλήρη καθορισμό ενός εικονοστοιχείου.

- *Επεξεργασία τμημάτων (Fragment Processing):* Τα τμήματα επιδέχονται περαιτέρω επεξεργασία που μπορέι να έχει να κάνει με τον φωτισμό, ενώ εφαρμόζονται διάφορες άλλες τεχνικές για την προσθήκη περαιτέρω ρεαλισμού στην τελική σκηνή.
- *Απεικόνιση (Display):* Σε αυτό το τελευταίο στάδιο τα τμήματα μετατρέπονται σε εικονοστοιχεία και επιτελούνται κάποια επιπλέον λειτουργίες. Αυτές έχουν να κάνουν με την απόφαση για το ποια αντικείμενα θα εμφανιστούν και σε ποιό βάθος. Τέλος, οι εικόνες προς έξοδο γράφονται στο frame buffer (απομονωτής στον οποίο αποθηκεύονται οι έτοιμες σκηνές προς απεικόνιση).

Όλα τα παραπάνω εκτελούνταν παραδοσιακά από τις CPUs με τη βοήθεια των επιταχυντών γραφικών, όποτε ήταν αυτοί διαθέσιμοι. Με στόχο τη βελτίωση της επίδοσης, τα περισσότερα στάδια από τα παραπάνω μεταφέρθηκαν σε ολοκληρωτικά αφοριωμένο υλικό, τις GPUs. Με χρήση της παραλληλίας που αυτές οι μονάδες είχαν τη δυνατότητα να προσφέρουν κατέστη δυνατό να επεξεργάζονται ταυτόχρονα πολλά διαφορετικά τμήματα. Το υλικό υλοποιούσε αρχικά όλα τα στάδια της παραπάνω σωλήνωσης, όμως τελικά κάποια στάδια έγιναν πιο προγραμματίσιμα, γεγονός που προσέδωσε ευελιξία στο χειρισμό των γραφικών και έδωσε το έναυσμα στην κοινότητα του GPGPU να αναπτυχθεί ακόμα περισσότερο.

Αρχικά, το *1998* το στάδια της Απεικόνιση υφής και της Επεξεργασίας αποσπασμάτων έγιναν τα πρώτα προγραμματιζόμενα στάδια, χωρίς όμως να προσφέρουν πολλές δυνατότητες προγραμματισμού και ευελιξίας. Οι πρώτης γενιάς κάρτες γραφικών προσέφεραν κάποιες συγκεκριμένες συναρτήσεις, οι οποίες μπορούσαν να μετατρέπουν προ-μετασχηματισμένα τρίγωνα σε εικονοστοιχεία, καθώς και να υποστηρίζουν την τεχνική του multi-texturing. Αυτή επέτρεπε τη μίξη δυο διαφορετικών υφών κατά το στάδιο της Μετατροπής της εικόνας σε εικονοστοιχεία.

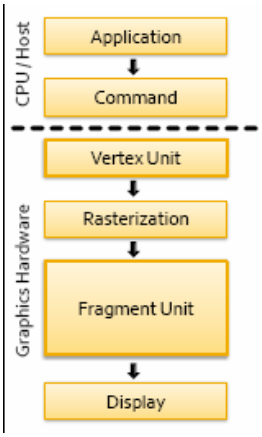
Το επόμενο βήμα έγινε γύρω στο *2000,* οπότε και τα στάδια του της Απεικόνιση υφής και της Επεξεργασίας αποσπασμάτων έγιναν περισσότερο προγραμματιζόμενα, υλοποιούνταν, πλέον, από μια ξεχωριστή προγραμματιζόμενη μονάδα, τη Μονάδα Τμημάτων (Fragment Unit), όπως φαίνεται στο Σχήμα 2-2.



Σχήμα 2-2 Η μορφή της σωλήνωση γραφικών το 2000.

Η μονάδα αυτή ήταν προγραμματιζόμενη μέσω γλώσσας μηχανής (assembly language). Έγινε, επίσης, εφικτή η ανάγνωση από τη μνήμη των υφών μέσω διαδικασίας που ονομάζεται αναζήτηση υφών (texture lookups). Έτσι, όλο το βάρος των υπολογισμών για τους Μετασχηματισμούς και τον Φωτισμό (Transformation and Lighting – T&L) το επωμιζόταν πλέον η κάρτα γραφικών. Το γεγονός αυτό, έδωσε ιδιαίτερη ώθηση στην ποιότητα των γραφικών αφού τέτοιου είδους υπολογισμοί είναι συνεχώς απαραίτητοι κατά την επεξεργασία μιας σκηνής. Ωστόσο, ούτε σ'αυτή τη γενιά μπορεί κανείς να πει ότι τα γραφικά ήταν αρκετά προγραμματιζόμενα.

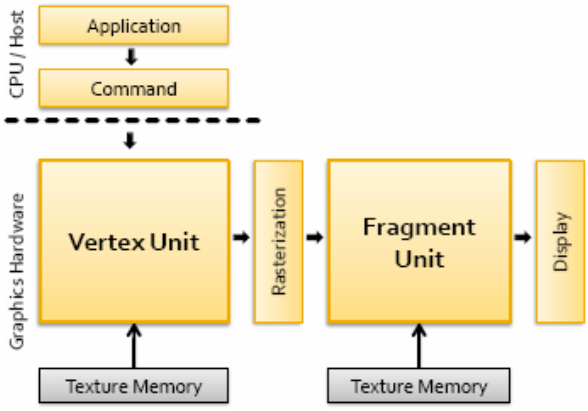
Η επόμενη γενιά των γραφικών παρουσιάστηκε το *2001*, οπότε και το στάδιο της Γεωμετρίας (Geometry) έγινε προγραμματιζόμενο και το ανέλαβε η Μονάδα Κόμβων (Vertex Unit), όπως φαίνεται και στο Σχήμα 2-3.



Σχήμα 2-3 Η μορφή της σωλήνωση γραφικών το 2001.

Αυτό το στάδιο προγραμματιζόταν μέσω γλώσσας μηχανής, αλλά δεν επιτρέπονταν αναγνώσεις από τη μνήμη, το μέγεθος του προγράμματος ήταν περιορισμένο (μέχρι 128 εντολές) και δεν επιτρέπονταν διακλαδώσεις και βρόχοι. Κατά το στάδιο αυτό, άρχισαν να πραγματοποιούνται και οι πρώτες σημαντικές προσπάθειες χρήσης των καρτών γραφικών για υπολογισμούς γενικού σκοπού.

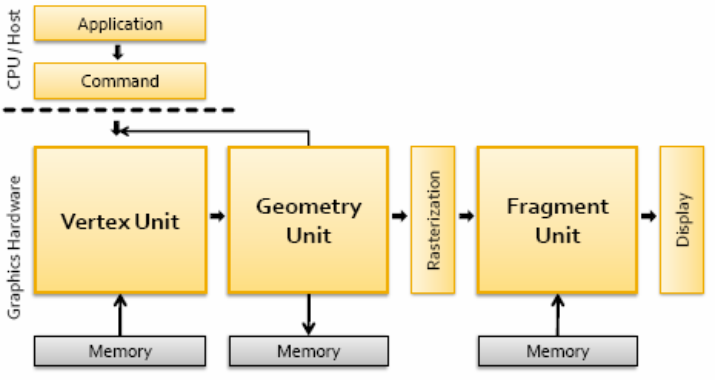
Κατά το επόμενο στάδιο, το *2003* έγιναν κάποιες σημαντικές αλλαγές που βελτίωσαν πολύ την κατάσταση. Πλέον, η Μονάδα Κόμβων μπορούσε να προβεί σε αναγνώσεις μνήμης, το μέγιστο μέγεθος προγράμματος αυξήθηκε, έγιναν εφικτές οι διακλαδώσεις, ενώ εμφανίστηκαν και γλώσσες υψηλού επιπέδου για προγραμματισμό στους επεξεργαστές γραφικών (High Level Shading Language – HLSL, C for graphics – Cg). Βέβαια η Μονάδα Κόμβων και η Μονάδα Τμημάτων δεν μπορούσαν να γράψουν στη μνήμη, παρά μόνο στο Frame Buffer, δεν υποστηρίζονταν πράξεις μεταξύ ακέραιων αριθμών και τελεστές πράξεων μεταξύ bits (bitwise operators). Η σωλήνωση γραφικών μετά από τις παραπάνω αλλαγές είναι φανερή στο Σχήμα 2-4



Σχήμα 2-4 Σωλήνωση γραφικών μετά το 2003.

Παρ' όλους, βέβαια, τους περιορισμούς η κοινότητα του GPGPU έκανε αρκετά βήματα προόδου μετά και τις παραπάνω αλλαγές που συντελέστηκαν στον τρόπο που επεξεργάζονταν τα γραφικά.

Το *2007* με την εμφάνιση της τελευταίας γενιάς καρτών τα πράγματα εξελίχθηκαν περαιτέρω. Ενδεικτικό είναι το Σχήμα 2-5.



Σχήμα 2-5 Σύγχρονη σωλήνωση γραφικών.

Σε αυτή τη γενιά επανήλθε το στάδιο της Γεωμετρίας μέσω της εμφάνισης της Μονάδας Γεωμετρίας (Geometry Unit), η οποία μπορεί να επιτελεί λειτουργίες πάνω σε πρωτογενή γεωμετικά στοιχεία και να γραφει στη μνήμη, κάτι που δεν είχε ξαναγίνει στην ιστορία της επεξεργασίας γραφικών. Ακόμη είχαμε την εμφάνιση ενοποιημένων μονάδων επεξεργασίας (unified processing units), το οποίο ουσιαστικά άλλαξε τον τρόπο που θα εργαζόταν πλέον η κοινότητα του GPGPU και αποτελεί τη βάση της αρχιτεκτονικής G80 της nVidia.

2.1.2 Εξέλιξη της ιστορίας του υλικού επεξεργασίας γραφικών (Graphics Hardware)

Είναι γενικώς αποδεκτό ότι το υλικό που είχε να κάνει με την επεξεργασία γραφικών μπορεί να διακριθεί στην εποχή πριν την εμφάνιση των GPUs και μετά. Η εποχή των GPUs μπορεί με τη σειρά της να διακριθεί σε πέντε γενιές κάθε μια από τις οποίες σχετίζεται άμεσα με τις εξελίξεις και τις διαδοχικές αλλαγές που συντελέστηκαν στη σωλήνωση των γραφικών.

Προ-GPUs Επιταχυντές Γραφικών: Εταιρείες όπως η Silicon Graphics και η Evans & Sutherland (E&S) ήταν από τις πρώτες και σημαντικότερες που κατασκεύασν επιταχυντές γραφικών αλλά το μεγάλο κόστο τους απέτρεψε την αγορά και τους χρήστες να υποδεχτεί μαζικά αυτά τα προιόντα.

GPUs Πρώτης Γενιάς: Το 1998 εμφανίστηκαν οι πρώτες GPUs για οικιακούς χρήστες που γνώρισαν αρκετά μεγάλη επιτυχία που οφειλόταν σε μεγάλο βαθμό στην εμφάνιση των πρώτων ηλεκτονικών παιχνιδιών με εντυπωσιακά, για την εποχή, γραφικά. Ενδεικτικά παραδείγματα GPUs εκείνης της εποχής ήταν οι κάρτες TNT2 της nVidia, Rage της ATI και Voodoo της 3Dfx. Οι κάρτες αυτής της γενιάς πραγματοποιούσαν επεξεργασία Μετασχηματισμών και Φωτισμού με χρήση υλικού για πρώτη φορά.

GPUs Δεύτερης Γενιάς: Η δεύτερη γενιά καρτών γραφικών εμφανίστηκε το χειμώνα του 1999/2000 με σημαντικότερα και πιο επιτυχημένα, εμπορικά και από πλευράς επιδόσεων, μοντέλα τις κάρτες GeForce256 και GeForce2 της nVidia, Radeon 7500 της ATI και Savage3D της S3. Σε αυτήν την γενιά εμφανίστηκε η προγραμματιζόμενη Μονάδα Τμημάτων.

GPUs Τρίτης Γενιάς: Το 2001 εμφανίστηκε η επόμενη γενιά GPUs με πιο αντιπροσωπευτικά μοντέλα τις κάρτες GeForce3 και GeForce4 Ti της nVidia και την Radeon 8500 της ATI. Αυτή η γενιά εισήγαγε τη δυνατότητα προγραμματισμού στην Μονάδα Κορυφών.

GPUs Τέταρτης Γενιάς: Τα σημαντικότερα μοντέλα της γενιάς μετά το 2003 ήταν τα GeForce FX και GeForce 6800 της nVidia και οι σειρές 9700, 9800 και X800 της ATI. Εδώ, προστέθηκαν επιπλέον δυνατότητες στις Μονάδες Τμημάτων και Κορυφών, με αποτέλεσμα να προσδοθεί ευελιξία στον προγραμματισμό των GPUs.

GPUs Πέμπτης Γενιάς: Η πιο σύγχρονη γενιά γραφικών αυτή τη στιγμή είναι εκείνη η οποία άλλαξε άρδην το σκηνικό στον προγραμματισμό των GPUs, με την ενοποιημένη αρχιτεκτονική στην οποία στηρίχτηκε. Σημαντικότερος εκπρόσωπος αυτής της γενιάς αποτελεί η $8^{η}$ σειρά καρτών γραφικών της nVidia. Ο ισχυρότερος εμπορικά διαθέσιμος για οικιακή χρήση αντιπρόσωπος αυτή της σειράς είναι το μοντέλο GeForce 8800 Ultra, το οποίο και θα χρησιμοποιηθεί σε επόμενα κεφάλαια, όπου θα αναλυθεί η αρχιτεκτονική στην οποία στηρίζεται (G80) και η πλατφόρμα ανάπτυξης προγραμμάτων που την εκμεταλλευτεί (CUDA).

Ενδεικτικά, και σαν μέτρο σύγκρισης, παρατίθεται ένας πίνακας (Σχήμα 2.6) με την εξέλιξη των τεχνικών χαρακτηριστικών και των επιδόσεων διαφόρων καρτών από τις παραπάνω γενιές.

Γενιά	Έτος	Προϊόν	Μέγεθος Ψηφίδας	Αριθμός transistors	Ρυθμός "γεμίσματος" εικονοστοιχείων
Πρώτη	Τέλη 1998	RIVA TNT	0,25 μ	7 M	50 M
Πρώτη	Αρχές 1999	RIVA TNT2	0,22 μ	9 M	75 M
Δεύτερη	Τέλη 1999	GeForce256	0,22 μ	23 M	120 M
Δεύτερη	Αρχές 2000	GeForce2	0,18 μ	25 M	200 M
Τρίτη	Αρχές 2001	GeForce3	0,15 μ	57 M	800 M
Τρίτη	Αρχές 2002	GeForce4 Ti	0,15 μ	63 M	1200 M
Τέταρτη	Αρχές 2003	GeForce FX	0,13 μ	125 M	2000 M
Τέταρτη	Αρχές 2004	GeForce6	0,13 μ	220 M	4000 M
Πέμπτη	Αρχές 2007	GeForce 8600 GT	0,08 μ	289 M	4320 M
Πέμπτη	Μέσα 2007	GeForce 8800 Ultra	0,09 μ	681 M	14688 M

Σχήμα 2-6 Σε αυτόν τον πίνακα φαίνονται χαρακτηριστικά επιδόσεων των παλαιότερων γενιών καρτών γραφικών.

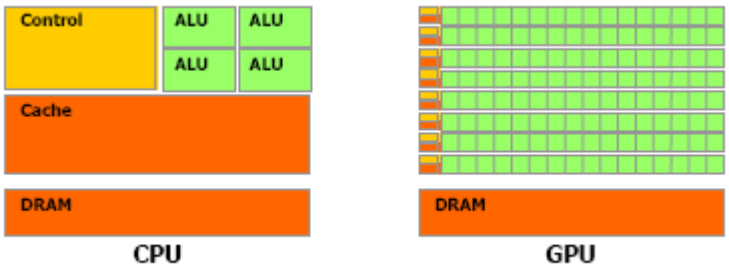
2.2 Λόγοι στροφής προς τη χρήση των GPUs σε υπολογισμούς γενικού σκοπού: Το Γιατί

Σε αυτήν την ενότητα θα γίνει αναφορά στους λόγους για τους οποίους στράφηκε το ενδιαφέρον στη χρήση των GPUs σε υπολογισμούς γενικού σκοπού, τι προσπάθησαν οι προγραμματιστές της κοινότητας GPGPU να εκμεταλλευτούν, καθώς και οι διάφοροι μέχρι στιγμής διαθέσιμοι τρόποι για εκμετάλλευση των δυνατοτήτων των GPUs. Η στροφή του ενδιαφέροντος τόσο της έρευνας όσο και της βιομηχανίας στις GPUs μπορεί να διαφανεί καλύτερα, εάν παρουσιαστούν πρώτα κάποιες σημαντικές διαφορές τους με τις CPUs.

Οι GPUs είναι οι πιο διαδεδομένοι παράλληλοι επεξεργαστές στο εμπόριο και ίσως οι πιο επιτυχημένοι από αυτήν την άποψη. Σχεδόν κάθε προσωπικός υπολογιστής στις μέρες μας έχει μια κάρτα γραφικών από τις τελευταίες γενιές και οι GPUs αποτελούν αρκετά σημαντικό μέρος του προϋπολογισμού για την αγορά ενός προσωπικού υπολογιστή. Το γεγονός ότι οι επεξεργαστές γραφικών παρέχουν σοβαρή υπολογιστική ισχύ με πολύ μικρό συγκριτικά κόστος, αποτέλεσε σημαντικό πόλο έλξης τόσο για την ερευνητική κοινότητα, όσο και για την βιομηχανία επεξεργαστών.

Οι GPUs παρέχουν έναν διαφορετικό τρόπο επεξεργασίας σε σχέση με τις CPUs, πράγμα που οφείλεται στις σχεδιαστικές διαφορές τους. Μεγάλο μέρος της επιφάνειας του τσιπ των CPUs αφιερώνεται σε ταχείες κρυφές μνήμες, δίνεται ιδιαίτερη σημασία στις διακλαδώσεις και τον χερισμό τους, προσανατολίζονται στο να επεξεργάζονται παράλληλα ή ψευδοπαράλληλα πολλά διαφορετικά νήματα/διεργασίες και έχουν κύριο στόχο την πολύ καλή επίδοση για ένα νήμα εκτέλεσης. Με άλλα λόγια,

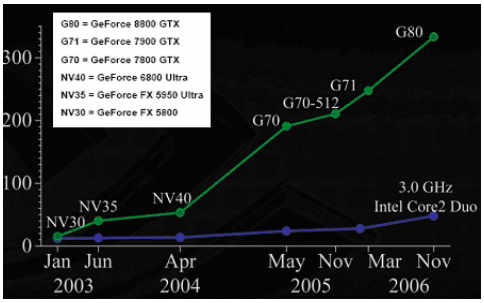
σκοπός είναι ο παραλληλισμός σε επίπεδο καθηκόντων (task parallelism), δηλαδή ταυτόχρονη εκτέλεση διεργασιών με μικρές ή καθόλου ανάγκες επικοινωνίας μεταξύ τους. Αντίθετα, οι GPUs είναι προσανατολισμένες στο να επιτυγχάνουν υψηλές επιδόσεις σε περιπτώσεις που είναι δυνατόν να έχουμε παραλληλισμό σε επίπεδο δεδομένων (data parallelism), δηλαδή όταν έχουμε ένα μεγάλο σύνολο δεδομένων στο οποίο είναι επιθυμητό να εκτελεστούν οι ίδιες λειτουργίες σε όλα τα στοιχεία του, ενώ ταυτόχρονα δεν υπάρχουν εξαρτήσεις δεδομένων μεταξύ των στοιχείων σε κάθε βήμα των υπολογισμών. Αυτό δίνει τη δυνατότητα στις GPUs να έχουν περισσότερες Μονάδες Αριθμητικών και Λογικών Πράξεων (Arithmetic and Logic Units - ALUs) οι οποίες μπορούν να επεξεργάζονται ταυτόχρονα, και ανεξάρτητα μεταξύ τους, διαφορετικά στοιχεία του αρχικού συνόλου δεδομένων. Τα παραπάνω φαίνονται σχηματικά στο Σχήμα 2.7.



Σχήμα 2-7 Διαφορές επιφάνειας τσιπ CPU και GPU ως προς την κατάληψη από τα διάφορα υποσυστήματα.

Επιπλέον, το σύστημα μνήμης των GPUs επιτρέπει πολύ γρήγορη πρόσβαση στην κύρια μνήμη της κάρτας γραφικών. Τέλος, παρά την έλλειψη κρυφών μνημών, οι GPUs μπορούν και καλύπτουν την καθυστέρηση προσπέλασης δεδομένων (latency) επικαλύπτοντας τον χρόνο αυτό με χρήσιμους υπολογισμούς, λόγω της αυξημένης παραλληλίας που παρέχουν. Στην ουσία, οι επεξεργαστές γραφικών ακολουθούν τον τρόπο επεξεργασίας SIMD (Single Instruction Multiple Data), όπως έχει οριστεί από τον Flynn [21], δηλαδή σε κάθε χρονική στιγμή εφαρμόζουν την ίδια λειτουργία/εντολή σε πολλά δεδομένα.

Αποτέλεσμα των παραπάνω είναι οι GPUs να αποτελούν ιδανική λύση για καθήκοντα υψηλών απαιτήσεων σε πράξεις και με μεγάλες δυνατότητες παραλληλισμού, καθώς η μεγαλύτερη επιφάνεια του τσιπ τους είναι αφιερωμένη σε κυκλώματα εκτέλεσης πράξεων. Γι αυτό το λόγο, όπως φαίνεται και στο Σχήμα 2-8, υπερτερούν κατά κράτος όσον αφορά την παρεχόμενη υπολογιστική δύναμη συγκριτικά με τις CPUs.



Σχήμα 2-8 Σύγκριση παρεχόμενης υπολογιστικής δύναμης μεταξύ GPUs και CPUs (σε εκατομμύρια πράξεις αριθμών κινητής υποδιαστολής μονής ακρίβειας).

Στο Σχήμα 2-8, παρατηρείται ότι το χάσμα στις επιδόσεις μεταξύ των GPUs και των CPUs είναι ήδη πολύ μεγάλο και διαφαίνεται η τάση να συνεχίσει να διευρύνεται. Επομένως, η εκμετάλλευση αυτής της υπολογιστικής δύναμης και του αυξημένου παραλληλισμού που προσφέρεται από τις σύγχρονες GPUs για εφαρμογές και αλγορίθμους γενικού σκοπού, αποτελεί μια πολύ ελκυστική και ενδιαφέρουσα πρόκληση.

2.3 Προγραμματισμός των GPUs για γενικές εφαρμογές: Το Πώς

Τα χαρακτηριστικά των αρχιτεκτονικών των GPUs φαίνεται να υπόσχονται πολλά στη θεωρία, αλλά θα ήταν ανώφελα αν δεν υπήρχε τρόπος να τα εκμεταλλευτεί κανείς στο έπακρο. Για αυτόν τον σκοπό έχουν αναπτυχθεί διάφορα προγραμματιστικά μοντέλα που εμφανίστηκαν με την πάροδο του χρόνου και τον προγραμματισμό των GPUs πραγματικά χρήσιμο και αποδοτικό.

Όπως αναφέρθηκε νωρίτερα, τα προγραμματιζόμενα στάδια της σωλήνωσης γραφικών είναι η Επεξεργασία Κορυφών και η Επεξεργασία Τμημάτων. Οι GPUs μέχρι και την τέταρτη γενιά είχαν ξεχωριστούς επεξεργαστές για κάθε ένα από τα δυο αυτά στάδια. Χαρακτηριστικό παράδειγμα είναι η κάρτα γραφικών ATI Radeon X1900 XTX, η οποία περιλάμβανε 8 επεξεργαστές κορυφών και 48 επεξεργαστές τμημάτων. Οι πρώτοι ήταν υπεύθυνοι για να συλλέξουν (gather) από την μνήμη υφών τα δεδομένα. Οι δεύτεροι ήταν υπεύθυνοι για την πραγματοποίηση του μεγαλύτερου όγκου των απαιτούμενων υπολογισμών, αλλά οι θέσεις στις οποίες έγραφαν τα αποτελέσματά ήταν προκαθορισμένες καθώς τα εικονοστοιχεία των οποίων τις τιμές θα καθόριζαν ήταν σε συγκεκριμένες θέσεις. Έτσι, δεν μπορούσαν να επιτελέσουν την λειτουργία διαμοιρασμού (scatter), σε αντίθεση με τους επεξεργαστές κορυφών. Εκείνοι μπορούσαν να κάνουν κάτι τέτοιο, αν και σε κάποιες περιπτώσεις ήταν δυνατόν να δημιουργηθούν προβλήματα και ανωμαλίες στα στάδια της σωλήνωσης γραφικών μετά την επεξεργασία κορυφών. Αυτή η διάκριση σε δυο διαφορετικά είδη επεξεργαστών εξαλείφθηκε στην τελευταία γενιά GPUs. Χαρακτηριστικό παράδειγμα είναι οι κάρτες της σειράς 8 της nVidia που βασίζονται στην Αρχιτεκτονική Συσκευής Ενιαίων Υπολογισμών (Compute Unified Device Architecture), κάτι που σημαίνει ότι όλοι οι επεξεργαστές μπορούν να λειτουργήσουν είτε σαν επεξεργαστές κορυφών και είτε σαν επεξεργαστές τμημάτων. Η αρχιτεκτονική αυτή θα αναλυθεί περισσότερο στο Κεφάλαιο 3.

2.3.1 Το μοντέλο προγραμματισμού σε ροές (Streaming programming model)

Για να ληφθεί η μέγιστη δυνατή επίδοση των προγραμμάτων που εκτελούνται σε GPUs, πρέπει να αλλάξει ο τρόπος σκέψης και προγραμματισμού σε σχέση με τους ευρέως διαδεδομένους σειριακούς τρόπους προγραμματισμού σε CPUs. Ίσως το πιο κατάλληλο μοντέλο που πρέπει να ακολουθηθεί είναι το μοντέλο προγραμματισμού σε ροές, καθώς είναι δυνατόν να εκμεταλλευτεί τον παραλληλισμό σε επίπεδο δεδομένων. Αυτό που επιτάσσει αυτός ο τρόπος προγραμματισμού ελιναι η οργάνωση των δεδομένων σε ροές δεδομένων (streams) και η συγκέντρωση των υπολογισμών που πρέπει να συντελεστούν σε πυρήνες υπολογισμού (kernels). Επειδή οι περισσότερες σκηνές γραφικών περιλαμβάνουν περισσότερα τμήματα παρά κορυφές χρησιμοποιήθηκαν για γενικούς υπολογισμούς οι επεξεργαστές τμημάτων καθώς υπήρχαν σε μεγαλύτερο πλήθος και με αυξημένες υπολογιστικές δυνατότητες σε σχέση με τους επε-

ξεργαστές κορυφών. Τα βήματα που ακολουθούνται ώστε να επιλυθεί ένα γενικού σκοπού πρόβλημα με χρήση του μοντέλου προγραμματισμού σε ροές σε μια GPU είναι τα εξής:

Βήμα 1: Αφού εντοπιστούν τα σημεία της εφαρμογής που μπορούν να παραλληλοποιηθούν, ο προγραμματιστής δημιουργεί τους πυρήνες υπολογισμού που χρειάζονται και τους υλοποιεί σαν ένα πρόβλημα για επεξεργασία τμημάτων, δηλαδή σαν να είχε να επεξεργαστεί τμήματα με σκοπό να πάρει σαν έξοδο εικονοστοιχεία στην οθόνη. Οι είσοδοι και έξοδοι αυτών των πυρηνών υπολογισμού θα είναι κάποιοι πίνακες, οι οποίοι αποθηκεύονται σαν υφές στη μνήμη της GPU. Οι αποθηκευμένοι πίνακες στην μνήμη των υφών αποτελούν τις ροές δεδομένων που θα επεξεργαστούν οι πυρήνες υπολογισμού.

Βήμα 2: Για να κληθεί ένας πυρήνας υπολογισμού πρέπει πρώτα να καθοριστεί το εύρος υπολογισμού του, δηλαδή τι μέγεθος θα έχει η ροή δεδομένων εξόδου. Αυτό γίνεται με το περασμα κορυφών στην GPU, σχηματίζοντας ένα ορθογώνιο στο εσωτερικό της επιφάνειας των εικονοστοιχείων που θα κάλυπταν ολόκληρη την οθόνη. Αφότου επιστρέψει η κλήση του συγκεκριμένου πυρήνα, στο προαναφερθέν ορθογώνιο θα περιέχονται τα αποτελέσματα του υπολογισμού του πυρήνα.

Βήμα 3: Ο μετατροπέας εικόνας σε εικονοστοιχεία (rasterizer) δημιουργεί ένα τμήμα για κάθε εικονοστοιχείο που θα περιλαμβάνεται τελικά στο απεικονιστικό μέσο (οθόνη). Έτσι, προετοιμάζονται οι θέσεις στις οποίες θα γίνει εγγραφή των αποτελεσμάτων πάνω στα εικονοστοιχεία της εξόδου, και μετά από αυτό το βήμα, κάθε τμήμα που θα επεξεργαστεί έχει αντιστοιχηθεί σε κάποιο εικονοστοιχείο, στο οποίο θα γράψει το αποτέλεσμα της επεξεργασίας του.

Βήμα 4: Αυτό το βήμα αποτελεί ουσιαστικά το στάδιο επεξεργασίας, καθώς το κάθε τμήμα αποτελεί ένα στοιχείο δεδομένων και κάθε τέτοιο θα επεξεργαστεί από τον πυρήνα που έχει συνταχθεί στο Βήμα 1. Σημειώνεται ότι το κάθε στοιχείο επεξεργάζεται από τον ίδιο πυρήνα. Ανάλογα με τις συντεταγμένες πάνω στην οθόνη που θα γραφούν τα αποτελέσματα του πυρήνα υπολογισμού (οι θέσεις είναι προκαθορισμένες) μπορούν να διαφοροποιούνται οι ακριβείς πράξεις που συντελούνται σε κάθε στοιχείο δεδομένων

Βήμα 5: Η έξοδος του σταδίου επεξεργασίας τμημάτων έχει σαν αποτέλεσμα σε κάθε εικονοστοιχείο να αντιστοιχεί μια τιμή (ή ένας πίνακας τιμών). Ο πίνακας όλων των τιμών εξόδου μπορεί να είναι το τελικό μας αποτέλεσμα ή να αποτελεί κάποιο ενδιάμεσο βήμα της εφαρμογής και πιθανώς να χρησιμοποιηθεί σαν input ροή δεδομένων σε κάποια μεταγενέστερη κλήση άλλου πυρήνα.

Το παραπάνω μοντέλο έχει χρησιμοποιηθεί κατά κόρον μέχρι και σήμερα σε πολλές εφαρμογές GPGPU με μικρές ή μεγαλύτερες παρεκκλίσεις [22].

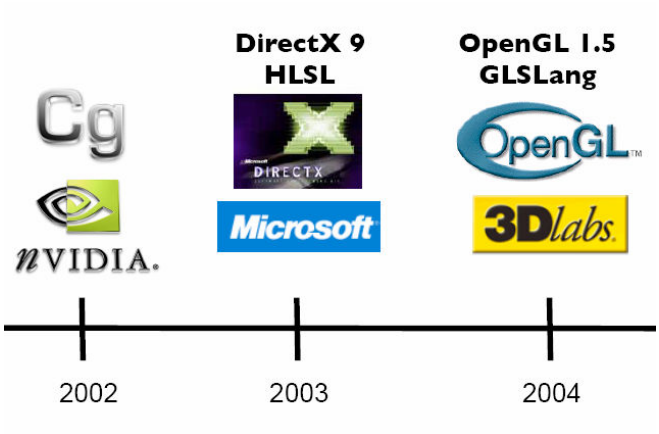
2.3.2 Γλώσσες προγραμματισμού για GPGPU

Παλαιότερα τα προγραμματιζόμενα στάδια της σωλήνωσης γραφικών προγραμματίζονταν μέσω γλώσσας μηχανής από την προγραμματιστική διεπαφή των γραφικών (OpenGL ή Direct3D). Τα τελευταία χρόνια όμως έχουν γίνει προσπάθειες από εταιρείες αλλά και ακαδημαϊκά ιδρύματα να δημιουργηθούν γλώσσες υψηλού αλλά και

μεσαίου επιπέδου που να δώσουν μια επιπλέον ώθηση, ευελιξία, ευχρησία και ευκολίες στην κοινότητα του GPGPU. Θα αναφερθεί παρακάτω με συντομία ένα σημαντικό μέρος αυτών των προσπαθειών, με έμφαση στα πλεονεκτήματα, στα μειονεκτήματα και στις ιδιαιτερότητες της κάθε προσπάθειας.

Γλώσσες σκίασης-Shaders

Αυτός ο τρόπος προγραμματισμού είναι ο πλησιέστερος στον τρόπο που προγραμματίζονται τα γραφικά. Οι γλώσσες προγραμματισμού οι οποίες χρησιμοποιούνται για προγραμματισμό γραφικών στα στάδια της επεξεργασίας κορυφών και της επεξεργασίας τμημάτων, ονομάζονται γλώσσες σκίασης. Αυτές μπορούν να χρησιμοποιηθούν για γενικές εφαρμογές με την κατάλληλη απεικόνιση της εφαρμογής που μας ενδιαφέρει σε πρόβλημα επεξεργασίας γραφικών.



Σχήμα 2-9 Χρονοδιάγραμμα εμφάνισης των σημαντικότερος γλωσσών σκίασης.

Οι σημαντικότερες και πιο διαδεδομένες γλώσσες σκίασης που έχουν χρησιμοποιηθεί για προγραμματισμό GPUs σε γενικές εφαρμογές είναι οι εξής:

C for Graphics (Cg): Η Cg [30] είναι μια γλώσσα σκίασης που αναπτύχθηκε από την nVidia. Έχει παρόμοια σύνταξη με την γλώσσα προγραμματισμού C, περιλαμβάνοντας παράλληλα κάποια στοιχεία από τις C++ και Java και από κάποιες πιο παλιές γλώσσες σκίασης. Είναι ανεξάρτητη πλατφόρμας χάρη στη συμβατότητά της με το πρότυπο OpenGL, ενώ είναι συμβατή και με Direct3D. Για να αντιμετωπίσει διαφορές και αποκλίσεις που υπάρχουν μεταξύ διαφόρων προγραμματιστικών διεπαφών γραφικών περιλαμβάνει έναν αριθμό προτύπων, τα οποία αν ακολουθηθούν εγγυώνται ότι τα προγράμματα που θα γράψουμε σε αυτήν την γλώσσα σκίασης θα τρέξουν στις κάρτες γραφικών που υποστηρίζουν το επιλεγμένο πρότυπο.

High Level Shading language (HLSL): Αυτή η υψηλού επιπέδου γλώσσα σκίασης εμφανίστηκε μαζί με το DirectX 9. Γλώσσες σκίασης υποστηρίζονταν και από την προηγούμενη έκδοση του DirectX (την 8), αλλά ο μόνος τρόπος να γραφτούν ήταν μέσω γλώσσας μηχανής. Η Cg και η HLSL [31] είναι σχεδόν η ίδια γλώσσα και αναπτύχθηκαν ύστερα από συνεργασία της nVidia με την Microsoft, αλλά έχουν διαφορετικά ονόματα για εμπορικούς λόγους. Διαφορά τους είναι ότι η HLSL μεταγλωττίζεται μόνο σύμφωνα μ το πρότυπο Direct3D και όχι με το OpenGL.

OpenGL Shading Language (GLSlang): Και αυτή η γλώσσα σκίασης είναι βασισμένη στην γλώσσα προγραμματισμού C και έχει διατηρήσει πολλά στοιχεία της παρότι αποτελούν στενωπό στην επίδοση κάποιες φορές. Εμπλουτίζει την C με νέους τύπους διανυσμάτων και πίνακα, δομές που χρησιμοποιούνται κατά κόρον στην επεξεργασία γραφικών, ενώ δανείζεται και κάποια στοιχεία από την C++, όπως για παράδειγμα ο πολυμορφισμός συναρτήσεων. Η GLSlang [32] είναι διαθέσιμη από την έκδοση 1.4 του OpenGL και αποτελεί τον πυρήνα της έκδοσης 2.0 του OpenGL.

Brook

Το BrookGPU [25] είναι ένα ερευνητικό πρόγραμμα του πανεπιστημίου του Stanford με σκοπό να καταστήσει δυνατό σε προγραμματιστές που δεν είχαν άμεση σχέση με τα γραφικά να μπορέσουν να γράψουν αποδοτικές εφαρμογές για GPUs. Είναι μια προσπάθεια που βρήκε μεγάλη απήχηση στην επιστημονική κοινότητα, καθώς με τη χρήση του έχουν γραφτεί πολλές εφαρμογές. Αποτελεί μια γλώσσα προγραμματισμού, η οποία είναι στην ουσία μια επέκταση της γλώσσας C, με επεκτάσεις που έχουν να κάνουν με την χρήση και αξιοποίηση του προγραμματιστικού μοντέλου σε ροές. Για το BrookGPU, οι GPUs είναι μια πλατφόρμα εκτέλεσης κώδικα που μπορεί να παραληλοποιηθεί. Η κάρτα γραφικών, δηλαδή, αποτελεί έναν συνεπεξεργαστή εκτέλεσης παράλληλου κώδικα. Ο μεταγλωττιστής του Brook ονομάζεται brcc. Οι ροές που χρησιμοποιεί το Brook είναι όμοιες με τους πίνακες της C, με τη διαφορά ότι τα στοιχεία τους μπορούν να επεξεργαστούν την ίδια χρονική στιγμή από διαφορετικά επεξεργαστικά στοιχεία/πυρήνες. Οι πυρήνες υπολογισμού του Brook είναι σαν τις συναρτήσεις της C μόνο που εκτελούνται παράλληλα από πολλούς διαφορετικούς πυρήνες. Ουσιαστικά, αυτό που κάνει το Brook είναι να απεικονίζει ένα πραγματικό πρόβλημα σε πρόβλημα γραφικών. Οι πυρήνες υπολογισμού αποτελούν προγράμματα επεξεργασίας τμημάτων, και οι ροές, είναι δεδομένα της texture υφών. Σε τελικό στάδιο, ένα πρόγραμμα Brook μετατρέπεται σε κλήσεις κάποιου API γραφικών.

Sh

Η Sh [26] είναι μια γλώσσα ανάπτυξης προγραμμάτων σκίασης, η οποία αναπτύχθηκε και αυτή ακαδημαικά στο πανεπιστήμιο του Waterloo. Είναι υλοποιημένη με βάση την C++ και μπορεί να απεικονίσει κώδικα σε ένα API γραφικών εκμεταλλευόμενη κάποιες από τις δυνατότητές της C++.

RapidMind

Η RapidMind [27] αποτελεί μια εφαρμογή με σκοπό να αξιοποιεί αποδοτικά πολυπύρηνους επεξεργαστές. Επιτρέπει σε κώδικα που έχει γραφεί εξ'ολοκλήρου σε C++ μέσω των εργαλείων του RapidMind να παραλληλοποιηθεί για πολυπύρηνα συστήματα, ενώ είναι εφικτό να παράγει κώδικα για τον επεξεργαστή Cell για διάφορες GPUs ή και για πολυπύρηνες CPUs.

Microsoft Accelerator

Ο Microsoft Accelerator [28] είναι ένα σύστημα που δημιουργήθηκε από ερευνητές της Microsoft με στόχο να απλοποιήσει την προσπάθεια του GPGPU προσφέροντας μια υψηλού επιπέδου γλώσσα. Σκοπός ήταν αυτή η γλώσσα να είναι κατάλληλη για εφαρμογές με μεγάλο παραλληλισμό σε επίπεδο δεδομένων μέσω μιας βιβλιοθήκης η

οποία μπορεί να χρησιμοποιηθεί και από άλλες γλώσσες προγραμματισμού. Ο Accelerator μεταφράζει λειτουργίες που εφαρμόζονται παράλληλα σε σύνολα δεδομένων σε προγράμματα σκίασης εικονοστοιχείων, επιτυγχάνοντας σημαντική επιτάχυνση σε σχέση με κώδικα που τρέχει εξ' ολοκλήρου σε CPUs.

Peakstream

Το προϊόν Peakstream [29] προσφέρει κάποια εργαλεία, τα οποία παρέχουν δυνατότητες χρησιμοποίησης του προγραμματιστικού μοντέλου σε ροές σε πολυνηματικές εφαρμογές που προορίζονται για εκτέλεση σε πολυπύρηνες πλατφόρμες (πολυπύρηνοι x86 επεξεργαστές, GPUs, συνδυασμός CPU και GPU, επεξεργαστής Cell κλπ.). Τα εργαλεία αυτά περιλαμβάνουν προγραμματιστικές διεπαφές, μια εικονική μηχανή (virtual machine), ένα just-in-time μεταγλωττιστή, ένα πρόγραμμα καταγραφής του προφίλ εκτέλεσης (profiler) κλπ. Η εικονική μηχανή αναλαμβάνει τη χρονοδρομολόγηση (scheduling) και την κατανομή πόρων ανεξάρτητα από την υποδομή σε υλικό της κάθε πλατφόρμας. Απευθύνεται στην κοινότητα του High Performance Computing και μπορεί να χρησιμοποιηθεί σε ποικίλες εφαρμογές.

MATLAB

Το MATLAB [33] ή MATrix LABoratory (Εργαστήριο Πινάκων) χρησιμοποιείται ευρέως από ερευνητές για αριθμητικούς υπολογισμούς και σαν μια γλώσσα προγραμματισμού για υπολογιστικές εφαρμογές. Παρ'όλο που χρησιμοποιεί κάποιες βιβλιοθήκες με καλή επίδοση, είναι δυνατόν να επιτευχθούν υψηλές επιδόσεις σε εφαρμογές που τρέχουν σε αυτό εάν γίνει χρήση GPUs. Αυτό μπορεί να επιτευχθεί με χρήση αρχείων με την επέκταση MEX (Matlab EXecutable) τα οποία καλούν υπάρχοντα κομμάτια κώδικα γραμμένα σε C ή FORTRAN. Επίσης το MATLAB μπορεί να επιταχυνθεί και μέσω των βιβλιοθηκών που προσφέρει η τεχνολογία CUDA.

CUDA

Η τεχνολογία CUDA [23] (Compute Unified Device Architecture), η οποία θα αναλυθεί περισσότερο σε επόμενο κεφάλαιο, αποτελεί την τελευταία προσπάθεια στον τομέα του προγραμματισμού καρτών γραφικών για γενικού σκοπού εφαρμογές. Η CUDA προσφέρει μια επέκταση της γλώσσας προγραμματισμού C και παρέχει μια αφαίρεση του υποκείμενου υλικού προσφέροντας τρόπους πρόσβασης σε μια γενική μνήμη DRAM για αποθήκευση αποτελεσμάτων ή δεδομένων και σε μια τοπική μνήμη πιο κοντά στο σύνολο των μονάδων επεξεργασίας. Οι παραπάνω μνήμες είναι πλήρως εκμεταλλεύσιμες από τον προγραμματιστή και ο τρόπος αξιοποίησης του προγραμματιστικού μοντέλου σε ροές είναι λίγο διαφορετικός από αυτόν που αναλύθηκε στην ενότητα 2.3.1. Το SDK (Software Development Kit) έγινε διαθέσιμο δωρεάν στις αρχές του 2007 σε δοκιμαστική μορφή, ενώ τώρα είναι διαθέσιμη η έκδοση 1.1. Περιλαμβάνει API, οδηγό υλικού (hardware driver), δυο Μαθηματικές βιβλιοθήκες και άλλα χαρακτηριστικά.

CTM

Η προσανατολισμένη προς το GPGPU τεχνολογία της AMD που υποστηρίζει τη σειρά των ATI Radeon καρτών γραφικών ονομάζεται Close To Metal (CTM) [24] και όπως φαίνεται από την ονομασία της είναι μια πιο χαμηλού επιπέδου προσέγγιση σε

σχέση με το CUDA. Αποτελεί μια προσπάθεια να αξιοποιηθούν οι υψηλές επιδόσεις σε υπολογισμούς κινητής υποδιαστολής των μαζικά παραλλήλων καρτών γραφικών της AMD. Το SDK του CTM δίνει πρόσβαση χαμηλού επιπέδου στους προγραμματιστές εφαρμογών ενώ περιλαμβάνει και κάποιες υλοποιήσεις σε πιο υψηλό επίπεδο. Η προγραμματιστική διεπαφή του CTM έχει εξελιχθεί σε 2 διαφορετικά επίπεδα: Το Επίπεδο Αφαίρεσης Υλικού (Hardware Abstraction Layer-HAL) και το Επίπεδο Αφαίρεσης Υπολογισμών (Compute Abstraction Layer-CAL). Το πρώτο είναι κάτι σαν μια διεπαφή οδηγού και εξαρτάται από το μοντέλο της κάρτας γραφικών και αφορά το πιο χαμηλό επίπεδο προγραμματισμού. Αντίθετα, το CAL αποτελεί μια διεπαφή υψηλότερου επιπέδου, ανεξάρτητου από το μοντέλο της κάρτας γραφικών και μπορεί να προσφέρει υλοποιήσεις για πολυπύρηνους επεξεργαστές, για GPUs καθώς και για ετερογενή υπολογιστικά συστήματα. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν την βιβλιοθήκη ACML (AMD Core Math Library) για να γράψουν κώδικα υψηλού επιπέδου ή εργαλεία όπως το RapidMind ή το Peakstream.

2.4 Εφαρμογές GPGPU

Σε αυτήν την ενότητα θα αναφερθούν κάποιες από τις εφαρμογές που έχει βρει η ιδέα της χρήσης Μονάδων Επεξεργασίας Γραφικών σε ποικίλους επιστημονικούς ή άλλης φύσεως τομείς.

Οι GPUs χρησιμοποιήθηκαν σε ένα πλήθος λειτουργιών που έχουν να κάνουν με *Βάσεις Δεδομένων* [34] κυρίως σαν συνεπεξεργαστές της CPU, ως τρόπος να επιταχυνθούν κάποιοι υπολογισμοί ή χωρικές λειτουργίες βάσεων δεδομένων. Ένας άλλος τομέας όπου χρησιμοποιήθηκαν οι κάρτες γραφικών είναι ο τομέας της *Επεξεργασίας Ήχου* [35, 36] και ιδιαίτερα στη δημιουργία ηχητικών εφέ και στον υπολογισμό της ακουστικής. Άλλη εφαρμογή του GPGPU είχε να κάνει με *Προχωρημένες Τεχνικές Επεξεργασίας Εικόνας* και ειδικότερα σε αλγορίθμους απεικόνισης φωτονίων [37, 42, 43, 44] (photon mapping algorithms), Υποεπιφανειακή Σκέδαση [38] (Subsurface Scattering), Μοντελοποίηση Εικόνων (image modeling) και ανάπτυξη περαιτέρω τεχνικών για ρεαλιστικότερες εικόνες (για παράδειγμα global illumination, ray tracing) [39,40, 41]. Ακόμη, χρησιμοποιήθηκαν οι GPUs σε *Εφαρμογές Επεξεργασίας Σήματος και Εικόνας*, όπως για παράδειγμα στην Όραση Υπολογιστών [45, 46] (Computer Vision), στην ανάλυση ιατρικών εικόνων, στη σύγκριση εικόνων ενώ αναπτύχθηκαν κατάλληλοι αλγόριθμοι για να συντελούνται σε GPUs μετασχηματισμοί όπως ο FFT (Fast Fourier Transformation), ο DCT (Discrete Cosine Transform) [47, 48, 49, 50]. Τέλος χρησιμοποιήθηκαν στη δημιουργία τεχνικών φιλτραρίσματος σημάτων ή εικόνων. Επίσης, η ιδέα του GPGPU βρήκε εφαρμογή και στην *Υπολογιστική Γεωμετρία* όπου κατέστη δυνατόν να ανλάβουν οι GPUs απαιτητικά, σε υπολογισμούς, καθήκοντα, όπως η Ανίχνευση Συγκρούσεων (Collision Detection), η Κατασκευαστική Στερεά Γεωμετρία (Constructive Solid Geometry), ο Υπολογισμός Διαφάνειας (Transparency Computation), τα Πεδία Απόστασης (Distance Fields)[51, 52, 53, 54]. Επιπροσθέτως, ευρείες ήταν και οι χρήσεις σε *Επιστημονικούς Υπολογισμούς* κυρίως με σκοπό ρεαλιστικές προσομοιώσεις πραγματικού χρόνου, βαριές σε αριθμητικές πράξεις, αλλά και σε εφαρμογές που είχαν ανάγκη συστήματα υψηλών επιδόσεων, με χαρακτηριστικό παράδειγμα την προσομοίωση ροής υγρών (Fluid Flow Simulation) [55]. Άλλα επιστημονικά πεδία, στα οποία έχουν χρησιμοποιηθεί οι GPUs για καλύτερες επιδόσεις είναι πράξεις γραμμικής άλγεβρας μεταξύ πινάκων και διανυσμάτων μεγάλου μεγέθους, προσομοιώσεις φυσικής, αναγνώριση φωνής, σε μοντελοποίηση

οικονομικών συστημάτων ή συστημάτων πρόβλεψης, προσομοιώσεις βιολογικών φαινομένων (π.χ. αναδίπλωση πρωτεϊνών)[56]. Επίσης, εφαρμογές έχουν υπάρξει στους τομείς της Υπολογιστικής Χημείας, στην Υπολογιστική Γεωεπιστήμη, στην Κρυπτογραφία, στα Νευρωνικά Δίκτυα και αλλού. Ο αναγνώστης μπορεί να αναζητήσει περισσότερες πληροφορίες για τις παραπάνω εφαρμογές στο [57].

2.5 Περιορισμοί – Προβλήματα – Μειονεκτήματα παλαιών γενιών GPUs

Σε αυτήν την ενότητα θα αναφερθούν οι περιορισμοί και τα προβλήματα που παρουσιάζονται σε γενικές υπολογιστικές εφαρμογές με χρήση GPUs. Κάποια από αυτά τα προβλήματα με την έλευση της τελευταίας γενιάς καρτών γραφικών και τεχνολογιών, όπως είναι το CUDA, ξεπερνιούνται και καθίσταται πιο εύκολος ο προγραμματισμός υψηλών επιδόσεων. Κάποια από τα προβλήματα φαίνεται αδύνατον στην παρούσα φάση να ξεπεραστούν, ενώ όσον αφορά άλλους περιορισμούς γίνονται ενθαρρυντικά βήματα προς την άρση τους, και είναι πιθανόν στο εγγύς μέλλον να αποτελέσουν παρελθόν.

Οι GPUs δεν είναι πανάκεια. Μπορεί οι επιδόσεις τους συγκριτικά με αυτές των CPUs να είναι πολύ καλύτερες και κάποιος να σκεφτεί ότι θα μπορούσαν ίσως να τις αντικαταστήσουν προσφέροντάς πανίσχυρα μηχανήματα για πάσης φύσεως εφαρμογές, αλλά αυτή η εντύπωση απέχει πολύ από την πραγματικότητα. Οι GPUs είναι ιδανικές στο να επιταχύνουν εφαρμογές που μπορούν να παραλληλοποιηθούν σε επίπεδο δεδομένων και όχι οποιεσδήποτε εφαρμογές. Δεν υπάρχουν αντίστοιχοι αποδοτικοί παράλληλοι αλγόριθμοι επίλυσης κάθε είδους προβλήματος. Χαρακτηριστικά παραδείγματα είναι οι εφαρμογές που έχουν να κάνουν με ευρεία χρήση δεικτών (όπως η επεξεργασία κειμένου), ή άλλες εφαρμογές οι οποίες δεν είναι δυνατόν να παραλληλοποιηθούν, λόγω της φύσης τους.

Ακόμη, το εύρος ζώνης επικοινωνίας μεταξύ GPU και CPU είναι περιορισμένο, κάτι που αποτελεί τροχοπέδη σε περιπτώσεις που θέλουμε να έχουμε συχνές ή μεγάλες σε ποσότητα δεδομένων μεταφορές. Ο δίαυλος AGP που χρησιμοποιούνταν μέχρι πρόσφατα έφτανε σε ταχύτητες μόλις τα 256MB/sec όταν πρόκειται για ανάγνωση.

Σημαντικό μειονέκτημα ήταν ότι, για να προγραμματίσει κάποιος στοχεύοντας σε υψηλές επιδόσεις, έπρεπε να γνωρίζει πολύ καλά πώς επεξεργάζονται τα γραφικά και πού πρέπει να επέμβει, ώστε να πετύχει υψηλές επιδόσεις. Κάτι τέτοιο απαιτούσε πολύ μεγάλη εξοικείωση, καθώς κανείς έπρεπε να μελετήσει ενδελεχώς το προγραμματιστικό μοντέλο γραφικών και την εκάστοτε αρχιτεκτονική.

Επίσης, οι GPUs μέχρι και την τέταρτη γενιά δεν υποστήριζαν τον μηχανισμό διαμοιρασμού, κάτι που περιόριζε σημαντικά την ευελιξια των προγραμμάτων. Ακόμη, δεν υποστηρίζονταν πράξεις μεταξύ ακεραίων αλλά και λειτουργίες σε επίπεδο bit. Επίσης, επιτρέπεται η αποκλειστική χρήση αριθμητικών πράξεων μονής ακρίβειας (32 bit) μεταξύ αριθμών κινητής υποδιαστολής, και μάλιστα χωρίς αυτές οι πράξεις να ακολουθούν κατά γράμμα το πρότυπο της IEEE (πρότυπο IEEE-754).

Επιπροσθέτως, οι εταιρείες παραγωγής καρτών γραφικών, για λόγους ανταγωνισμού και προστασίας των πρωτότυπων τεχνολογιών τους, δεν έδιναν στη ελεύθερη πρόσβαση σε όλες τις πτυχές των τεχνολογιών που χρησιμοποιούσαν τα προϊόντα τους.

Αυτό το γεγονός μπορεί να οδηγήσει σε λάθος υποθέσεις για το πώς μπορεί κάποιος προγραμματιστής να χρησιμοποιήσει την GPU για εφαρμογές γενικού σκοπού και κατ'επέκταση σε μη βέλτιστα αποτελέσματα.

Τέλος, η ύπαρξη δυο διαφορετικών ειδών επεξεργαστών (Κορυφών και Τμημάτων) στο υλικό των GPUs, άφηνε μέρος του αχρησιμοποίητο και σε λανθάνουσα κατάσταση κατά τη διάρκεια της εκτέλεσης εφαρμογών γενικού σκοπού. Αυτό αποτελούσε σπατάλη πόρων, και για αυτόν τον λόγο, δεν ήταν δυνατό να επιτευχθεί η καλύτερη δυνατή επίδοση.

Κεφάλαιο 3

Η ενοποιημένη αρχιτεκτονική πολλών πυρήνων nVidia G80

Η αρχιτεκτονική nVidia G80 αποτελεί την αρχιτεκτονική στην οποία βασίστηκαν οι κάρτες γραφικών της σειράς 8 της εταιρείας nVidia. Προκειμένου να γίνει εφικτός ο προγραμματισμός εφαρμογών γενικού σκοπού με χρήση αυτής της αρχιτεκτονικής, παρουσιάστηκε μέσα στο 2007 μια νέα πλατφόρμα ανάπτυξης εφαρμογών. Αυτή η πλατφόρμα ονομάζεται CUDA (Compute Unified Device Architecture – Ενοποιημένη Αρχιτεκτονική Συσκευής Υπολογισμών) και προσφέρει στον προγραμματιστή μια αφαίρεση του υλικού των καρτών γραφικών που ακολουθούν την αρχιτεκτονική G80. Εκτός από τις κάρτες γραφικών της σειράς 8, μπορεί να χρησιμοποιηθεί για να προγραμματίσει και κάποιες πολυπύρηνες εξειδικευμένες πλατφόρμες που απευθύνονται σε επαγγελματίες, με την επωνυμία Tesla, και οι οποίες βασίζονται και αυτές στην αρχιτεκτονική G80. Σε αυτό το κεφάλαιο θα εξερευνηθεί με λεπτομέρεια κάθε πτυχή της αρχιτεκτονικής G80 και της τεχνολογίας του CUDA που την εκμεταλλεύεται. Κάνοντας χρήση αυτής της γνώσης, θα καταστεί δυνατή στη συνέχεια η εκτέλεση κάποιων πειραμάτων προκειμένου να αποφανθούμε για το κατά πόσο αρμόζει η νέα αρχιτεκτονική σε σύγχρονες επιστημονικές εφαρμογές.

Όταν προγραμματίζεται η GPU μέσω του CUDA, αυτή θα φαίνεται σαν ένας ισχυρός συνεπεξεργαστής, ο οποίος μπορεί να εκτελέσει πολύ μεγάλο αριθμό νημάτων (threads) παράλληλα. Για να αξιοποιήσουμε δηλαδή την τεχνολογία αυτή, θα πρέπει να μετατρέψουμε έτσι τους αλγόριθμούς, ώστε τα τμήματά τους που περιλαμβάνουν πολλούς υπολογισμούς επί συνόλων δεδομένων και που μπορούν να παραλληλοποιηθούν, να «μεταφέρονται» στην GPU. Θα αναφέρεται από εδώ και στο εξής σαν «host» («οικοδεσπότης») η CPU και η κύρια μνήμη του υπολογιστή του συστήματος, και η κάρτα γραφικών/GPU και η μνήμη που αυτή περιλαμβάνει ως «device» ή «συσκευή».

3.1 Σύγκριση νέου και παλαιών τρόπων ανάπτυξης εφαρμογών γενικού σκοπού σε GPUs

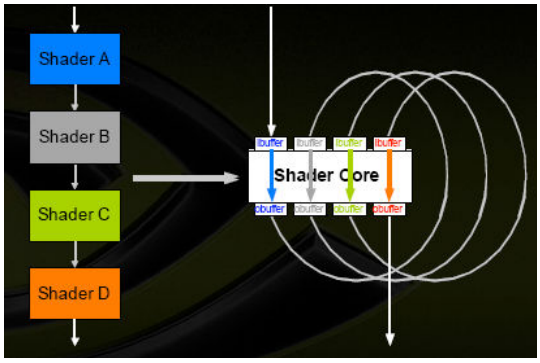
Αρχικά, σε αυτήν την ενότητα θα παρουσιαστούν οι διαφορές της αρχιτεκτονικής G80 και της πλατφόρμας χρήσης της (CUDA) με τους προγενέστερους τρόπους ανάπτυξης εφαρμογών γενικού σκοπού σε GPUs. Θα γίνει αναφορά των προβλημάτων που λύνονται σε αυτή τη νέα γενιά GPUs, ενώ θα γίνει μνεία και στους περιορισμούς που δεν αίρονται με την έλευση της αρχιτεκτονικής G80. Έτσι, θα γίνουν κατανοητοί στον αναγνώστη, οι λόγοι εμφάνισης αυτής της νέας πλατφόρμας και των δυνατοτήτων που αυτή προσφέρει σε σχέση με τις παλαιότερες. Επίσης, θα γίνει φανερό το τι έδωσε την ώθηση για αυτήν τη διπλωματική εργασία, καθώς οι προοπτικές της νέας αυτής αρχιτεκτονικής είναι πολλά υποσχόμενες.

3.1.1 Τι βελτιώνει η αρχιτεκτονική G80 και η τεχνολογία CUDA

Η νέα αυτή τεχνολογία περιλαμβάνει έναν ξεχωριστό οδηγό (driver) ο οποίος μεταφέρει και προσαρμόζει στη GPU βέλτιστα τα υπολογιστικά προγράμματα που γράφονται χωρίς να γίνεται χρήση της διεπαφής προγραμματισμού γραφικών. Έτσι απαλλάσσονται οι εφαρμογές από πολλούς από τους περιορισμούς που δημιουργούνταν μέχρι πρότινος, καθότι οι υπολογισμοί έπρεπε να προσαρμοστούν σε κατάλληλα προβλήματα επεξεργασίας γραφικών. Πολύ σημαντικό αποτέλεσμα αυτού του γεγονότος είναι ότι, πλέον, για να εκμεταλλευτεί κάποιος τα πανίσχυρα μηχανήματα επεξεργασίας γραφικών, δεν χρειάζεται να είναι εξοικειωμένος με την επεξεργασία και τον προγραμματισμό γραφικών. Αρκεί να έχει γνώση της γλώσσας C και των επεκτάσεων που προσφέρει η προγραμματιστική διεπαφή του CUDA καθώς και τα χαρακτηριστικά της αρχιτεκτονικής G80 που έχει στη διάθεση του, ώστε να πετύχει υψηλές επιδόσεις στους επιθυμητούς αλγόριθμους. Η διεπαφή CUDA αποτελεί επιπλέον ένα πολύ καλό εργαλείο για εκμάθηση των αρχών των μαζικά παράλληλων αρχιτεκτονικών και του προγραμματισμού τους, καθώς ήδη σε διάφορα πανεπιστήμια προσφέρονται μαθήματα σχετικά με συστήματα παράλληλης επεξεργασίας στηριζόμενα στη χρήση του CUDA ως πλατφόρμας πειραματισμού και εφαρμογής της θεωρίας.

Οι κάρτες της σειράς 8 της nVidia, οι οποίες βασίζονται στην αρχιτεκτονική G80, περιλαμβάνουν ενοποιημένα επεξεργαστικά στοιχεία. Αυτό σημαίνει ότι δεν υπάρχουν πλέον δυο ειδών επεξεργαστές (Τμημάτων και Κορυφών) όπως στις παλαιότερες γενιές GPUs, αλλά μόνο ένα είδος επεξεργαστικών στοιχείων. Αυτά τα επεξεργαστικά στοιχεία μπορούν να λειτουργήσουν είτε ως επεξεργαστές Τμημάτων. είτε ως επεξεργαστές Κορυφών, ανάλογα με τις απαιτήσεις της εφαρμογής που εκτελείται ανά πάσα στιγμή. Το γεγονός αυτό οδηγεί σε ελαχιστοποίηση της σπατάλης πόρων και στην

καλύτερη εκμετάλλευση του υλικού των καρτών γραφικών από τον προγραμματιστή και στην περίπτωση των εφαρμογών γραφικών αλλά και σε αυτήν των γενικών υπολογισμών. Προηγουμένως αναφέρθηκε ότι μέχρι τώρα χρησιμοποιούνταν μόνο οι επεξεργαστές Τμημάτων για τον προγραμματισμό εφαρμογών γενικού σκοπού. Πλέον, επειδή όλοι οι πυρήνες μπορούν να λειτουργήσουν σαν οποιοδήποτε είδος επεξεργαστή, η χρησιμοποίηση του υλικού είναι πιο αποδοτική. Στο Σχήμα 3-1 φαίνεται η σύγκριση του πώς εφαρμόζονταν διαδοχικά κάποια προγράμματα σκίασης μέχρι και τις προηγούμενες γενιές καρτών γραφικών, και του πώς συμβαίνει αυτό με τη χρήση των καρτών γραφικών με ενοποιημένα στοιχεία, όπως είναι αυτές που στηρίζονται στην αρχιτεκτονική G80.



Σχήμα 3-1 Σύγκριση παλαιού και νέου τρόπου χειρισμού προγραμμάτων σκίασης από το υλικό των GPUs.

Τα διαφορετικά χρώματα του Σχήματος 3-1 αναφέρονται σε διαφορετικά προγράμματα σκίασης. Μέχρι και την έλευση της τελευταίας γενιάς GPUs, τα προγράμματα εκτελούνταν σειριακά το ένα μετά το άλλο, ενώ τώρα πια υπάρχει ένα επιπλέον είδος σωλήνωσης: Μετά από κάθε στάδιο του προγράμματος σκίασης, αυτό είναι δυνατόν να γράψει το αποτέλεσμά του σε κάποιους καταχωρητές και να ανατροφοδοτήσει με δεδομένα ξανά το ενοποιημένο επεξεργαστικό στοιχείο (unified shader) και αυτό να συνεχίσει για όλα τα στάδια όλων των εκτελούμενων προγραμμάτων σκίασης.

Ένα ακόμα νέο στοιχείο που εισάγει η αρχιτεκτονική G80, όσον αφορά τις GPGPU εφαρμογές, είναι ότι, πλέον, υπάρχει η δυνατότητα πιο ευέλικτης χρήσης της μνήμης της GPU. Επιτρέπονται λειτουργίες και σκέδασης (scatter) και συλλογής (gather), σε αντίθεση με τις προηγούμενες γενιές. Γενικώς, το CUDA προσφέρει ένα αρκετά ευέλικτο μοντέλο μνήμης με πολλές δυνατότητες και πολυσύνθετη ιεραρχία μνήμης, το οποίο εκμεταλλεύεται την υποκείμενη αρχιτεκτονική με αποδοτικό τρόπο.

Η αρχιτεκτονική G80 πρωτοπορεί και σε έναν άλλον πολύ σημαντικό τομέα. Για πρώτη φορά παρέχεται, μέσω του CUDA, η δυνατότητα σε μια GPU να εκτελεί υπολογισμούς και πράξεις με ακέραιους αριθμούς και με bits. Επίσης, υποστηρίζεται δυνατότητα για διακλαδώσεις (branching), κάτι που δεν ήταν εφικτό σε παλαιότερες γενιές.

Οι επεξεργαστές γραφικών που στηρίζονται στη αρχιτεκτονική G80 δεν είναι βασισμένοι σε διανύσματα υπολογισμών (vector-based) όπως ήταν οι παλαιότεροι επεξεργαστές, οι οποίοι επιτελούσαν τις λειτουργίες τους πάνω σε μικρά διανύσματα (συνήθως τεσσάρων στοιχείων). Αυτό συνέβαινε γιατί τα γραφικά επεξεργάζονται τέτοια σύνολα δεδομένων (σε κάθε διάνυσμα συνήθως η πρώτη συντεταγμένη είχε να κάνει

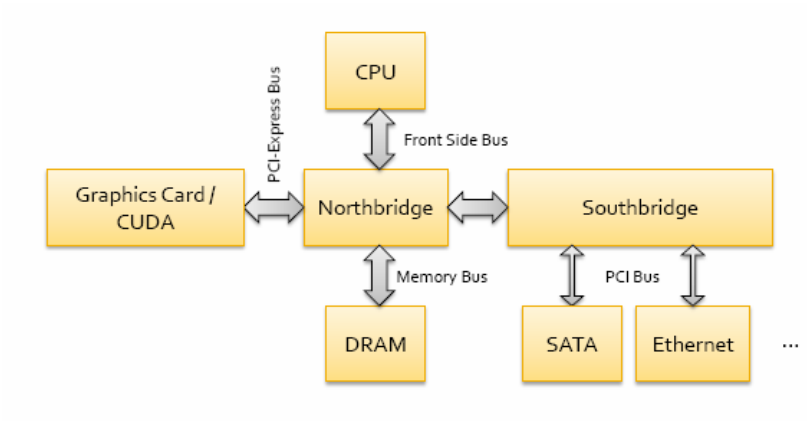
με το ποσοστό του κόκκινου χρώματος στο εικονοστοιχείο, το δεύτερο για το πράσινο, το τρίτο για το μπλε κτλ.), οπότε μία τέτοια οργάνωση των δεδομένων ευνοούσε την γρήγορη επεξεργασία τους. Αντίθετα, επενεργούν πάνω σε βαθμωτά (scalar) δεδομένα, κάτι που, εκτός των άλλων, βοηθάει στην αντίληψη της λειτουργίας της αρχιτεκτονικής από κάποιον ο οποίος δεν είναι εξοικειωμένος με τα γραφικά. Ο λόγος για τον οποίον έγινε αυτή η στροφή είναι η παρατήρηση αλλαγών στη δομή και τις ανάγκες των σύγχρονων προγραμμάτων σκίασης που συνθέτουν τις σύγχρονες εφαρμογές γραφικών. Εκείνα, για να έχουν καλή επίδοση, χρησιμοποιούσαν πολλές φορές βαθμωτά δεδομένα και είχαν ανάγκη από βαθμωτές πράξεις/λειτουργίες. Έτσι, ενσωμάτωσαν τη δυνατότητα να επιτελείται μίγμα διαφορετικών ειδών λειτουργιών (είτε σε βαθμωτά είτε σε διανυσματικά δεδομένα) στην αρχιτεκτονική G80.

Όσον αφορά τη χρήση αριθμών κινητής υποδιαστολής, συνεχίζονται να υποστηρίζονται μόνο αριθμοί μονής ακρίβειας, αλλά υπήρξε βελτίωση αναφορικά με την παρέκκλισή τους από το πρότυπο IEEE-754. Στο Σχήμα 3-2 παρουσιάζονται σε πίνακα οι αποκλίσεις και οι συμφωνίες με το εν λόγω πρότυπο αναλυτικά, και σε σύγκριση με άλλα σύγχρονα πολυεπεξεργαστικά μηχανήματα.

	G80	**SSE**	**IBM Altivec**	**Cell SPE**
Πρότυπο ακρίβειας	IEEE 754	IEEE 754	IEEE 754	IEEE 754
Στρογγυλοποιήσεις για πράξεις πολλαπλασιασμού και πρόσθεσης μεταξύ αριθμών κινητής υποδιαστολής	Στρογγυλοποίηση στο πλησιέστερο και στρογγυλοποίηση στο μηδέν	Όλες οι στρογγυλοποιήσεις του προτύπου (στο πλησιέστερο, στο μηδέν, στο άπειρο και το μείον άπειρο)	Στρογγυλοποίηση μόνο στο πλησιέστερο	Στρογγυλοποίηση στο μηδέν/αποκοπή μόνο
Χειρισμός υποκανονικών(denormal) αριθμών	Υποβιβασμός στο μηδέν	Υποστηρίζεται (χιλιάδες κύκλοι)	Υποστηρίζεται (χιλιάδες κύκλοι)	Υποβιβασμός στο μηδέν
Υποστήριξη μη αριθμών (NaN - Not a Number)	Ναι	Ναι	Ναι	Όχι
Υποστήριξη υπερχείλισης και άπειρου	Ναι	Ναι	Ναι	Μόνο του απείρου
Σημαίες	Όχι	Ναι	Ναι	Μερικές
Τετραγωνική Ρίζα	Μέσω λογισμικού μόνο	Μέσω υλικού	Μέσω λογισμικού μόνο	Μέσω λογισμικού μόνο
Διαίρεση	Μέσω λογισμικού μόνο	Μέσω υλικού	Μέσω λογισμικού μόνο	Μέσω λογισμικού μόνο
Ακρίβεια τετραγώνου	24 bit	12 bit	12 bit	12 bit
Ακρίβεια τετραγωνικής ρίζας	23 bit	12 bit	12 bit	12 bit
Ακρίβεια πράξεων log2(x) και 2^x	23 bit	Όχι	12 bit	Όχι

Σχήμα 3-2 Πίνακας αποκλίσεων/συμφωνιών από το πρότυπο IEEE 754 σύγχρονων πολυεπεξεργαστικών υπολογιστικών συστημάτων.

Οι κάρτες γραφικών των τελευταίων χρόνων είναι συνδεδεμένες με τη CPU μέσω του διαύλου PCIe (PCI Express), κάτι που επιτρέπει πολύ μεγαλύτερες ταχύτητες (θεωρητικά μέχρι 4 GB/s) μεταφοράς δεδομένων από και προς τον host. Στο Σχήμα 3-3, βλέπουμε σχηματικά το εσωτερικό ενός σύγχρονου προσωπικού υπολογιστή. Παρατηρούμε το πως συνδέονται ο host με την κάρτα γραφικών και τις ταχύτητες επικοινωνίας όλων των συσκευών/περιφερειακών.



Σχήμα 3-3 Θέση της κάρτας γραφικών σε ένα σύγχρονο προσωπικό υπολογιστή.

3.1.2 Προβλήματα-Περιορισμοί που συνεχίζουν να υφίσταται στην προσπάθεια ανάπτυξης εφαρμογών γενικού σκοπού με GPUs

Όμως, υπάρχουν κάποια εμπόδια και περιορισμοί, τους οποίους η αρχιτεκτονική G80 δεν έχει καταφέρει να ξεπεράσει. Για παράδειγμα, αν κοιτάξει κανείς το θέμα των αριθμών κινητής υποδιαστολής, θα παρατηρήσει ότι υποστηρίζονται μόνο αριθμοί μονής ακρίβειας και όχι διπλής[1]. Αυτό σημαίνει ότι τα αποτελέσματα και οι πράξεις παρουσιάζουν μειωμένη ακρίβεια. Συνεπώς, κάποιες επιστημονικές εφαρμογές οι οποίες χρειάζονται τη μέγιστη δυνατή ακρίβεια στους υπολογισμούς δεν μπορούν να εμπιστευτούν τις GPUs σαν αξιόπιστες πλατφόρμες εκτέλεσης. Επίσης, οι πράξεις μεταξύ αριθμών μονής ακρίβειας δεν εναρμονίζονται πλήρως ως προς την ακρίβεια τους με το πρότυπο 754 της IEEE. Οι αριθμοί διπλής ακρίβειας υποστηρίζονται από τη σειρά 9 της nVidia, η οποία κυκλοφόρησε τα μέσα του 2008, και η χρήση τους είναι εφικτή με την επόμενη έκδοση του CUDA (το CUDA 2.0, το οποίο βρίσκεται σε δοκιμαστική έκδοση).

Επίσης, είναι δύσκολο, έως και αδύνατο σε πολλές περιπτώσεις, κάποιοι αλγόριθμοι να απεικονιστούν κατάλληλα σε GPUs λόγω διαφόρων ειδών περιορισμών και παρά τις διευκολύνσεις που προσφέρει στους προγραμματιστές το CUDA δεν διαφαίνεται στον ορίζοντα κάποια βελτίωση σε αυτόν τον τομέα. Για παράδειγμα, συνεχίζει να μην υποστηρίζεται η αναδρομή (recursion) και η χρήση δεικτών κάτι που δυσχεραίνει την επίλυση συγκεκριμένων προβλημάτων με χρήση των νέων GPUs.

[1] Για την ακρίβεια, υποστηρίζεται ο τύπος double/διπλής ακρίβειας αλλά γίνεται αυτόματη μετατροπή των δηλωμένων με αυτόν τον τρόπο δεδομένων σε float/μονής ακρίβειας

Οι μεταφορές μεταξύ CPU και GPU είναι ακόμα πολύ αργές συγκριτικά με τις μεταφορές από μνήμη προς επεξεργαστή και για αυτό, όπως θα αναλυθεί σε επόμενο κεφάλαιο, θα πρέπει να αποφεύγονται όσο είναι αυτό εφικτό.

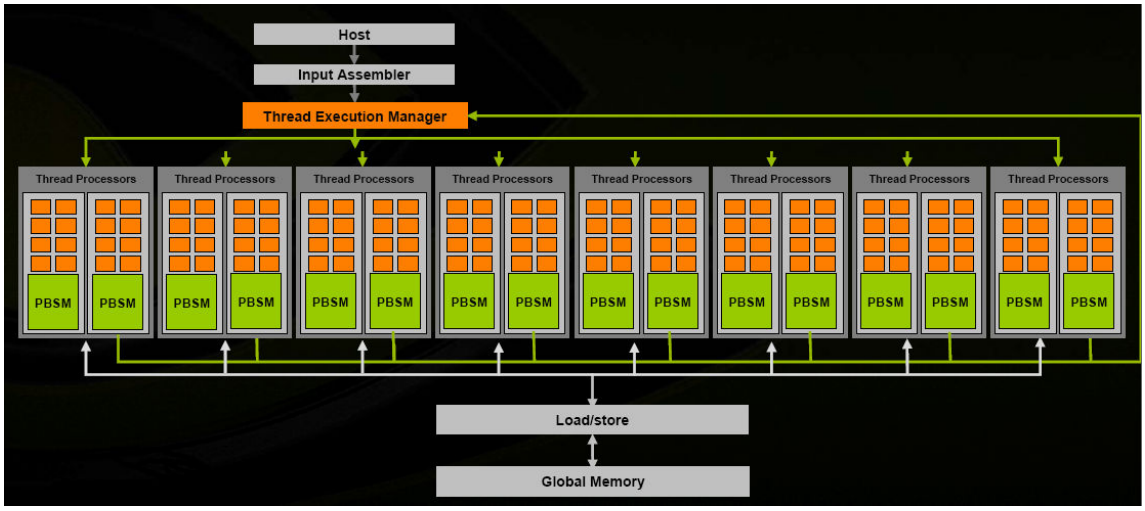
Η τεχνολογία CUDA λειτουργεί μόνο σε συνεργασία με τις κάρτες γραφικών της nVidia και, συνεπώς, τα προγράμματα που γράφονται με χρήση της προγραμματιστικής διεπαφής που αυτή προσφέρει, δεν είναι μεταφέρσιμα σε άλλα αντίστοιχα μηχανήματα ή πλατφόρμες. Επίσης, για να χρησιμοποιηθεί αποδοτικά η αρχιτεκτονική G80 με τη βοήθεια του CUDA, ο προγραμματιστής πρέπει να μάθει σε βάθος την αρχιτεκτονική αυτή, ώστε να έρθει σε θέση να γράφει τα βέλτιστα δυνατά προγράμματα. Επομένως, τελικά, δεν αποφεύγεται ένα στάδιο προσαρμογής και εκμάθησης των ατόμων που θα συντάξουν κώδικα για εφαρμογές γενικού σκοπού στην αρχιτεκτονική G80. Ωστόσο, ο χρόνος που απαιτείται είναι πολύ μικρότερος με τον χρόνο που θα χρειαζόταν για να εξοικειωθεί κάποιος με τον προγραμματισμό γραφικών, κάτι απαραίτητο στις παλαιότερες προσπάθειες των μελών της κοινότητας του GPGPU.

3.2 Ανάλυση Υλικού και Αρχιτεκτονικής

Σε αυτήν την ενότητα θα παρουσιαστεί ότι έχει να κάνει με την αρχιτεκτονική G80 και γενικώς το υλικό (hardware) που εκμεταλλεύεται η τεχνολογία CUDA για χρήση των GPUs σε εφαρμογές γενικού σκοπού. Θα γίνει, έτσι, ορατό στον αναγνώστη το πώς επιτυγχάνονται οι διαφοροποιήσεις που αναφέρθηκαν στην προηγούμενη ενότητα όσον αφορά την προσπάθεια για ανάπτυξη εφαρμογών γενικού σκοπού στη νέα αυτή πλατφόρμα. Η αρχιτεκτονική G80 βρίσκει εφαρμογή σε όλες τις κάρτες γραφικών της σειράς 8 της εταιρείας nVidia. Το κάθε μοντέλο αυτής της σειράς παρουσιάζει κάποιες διαφοροποιήσεις ως προς τα μεγέθη μνήμης, τον αριθμό επεξεργαστικών πυρήνων και κάποιων επιμέρους δυνατοτήτων, αλλά οι αρχές της αρχιτεκτονικής, όπως αυτή θα παρουσιαστεί σε αυτή την ενότητα, διατηρούνται σε όλα τα μοντέλα.

3.2.1 Γενική επισκόπηση – Ομαδοποιήσεις επεξεργαστικών στοιχείων

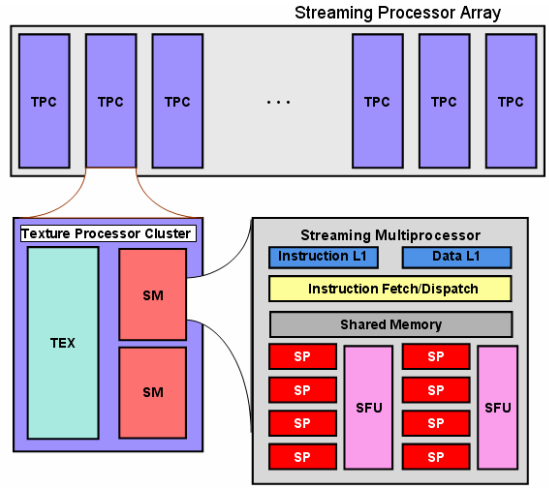
Η αρχιτεκτονική G80 περιλαμβάνει ένα σύνολο πολυεπεξεργαστών, οι οποίοι είναι τύπου SIMD (Single Instruction Multiple Data), δηλαδή σε έναν δεδομένο κύκλο, σε κάθε πολυπεξεργαστική μονάδα εκτελούνται οι ίδιες εντολές πάνω σε διαφορετικά δεδομένα. Αρκετά διαφωτιστικό ως προς την οργάνωση των επεξεργαστικών μονάδων, είναι το Σχήμα 3-4, στο οποίο φαίνεται ένα μπλοκ διάγραμμα της κάρτας γραφικών. Το κάθε στοιχείο που αυτό περιλαμβάνει, τα χαρακτηριστικά του και τα δομικά του στοιχεία θα αναλυθούν στις επόμενες ενότητες. Στο Σχήμα 3-4 διακρίνονται δύο επίπεδα ομαδοποιήσεων: ομαδοποιήσεις των επεξεργαστικών πυρήνων σε οκτάδες, και ομαδοποιήσεις ανά ζευγάρι γειτονικών οκτάδων. Κάθε ζευγάρι οκτάδων ονομάζεται Συστοιχία Επεξεργαστών Υφών (Texture Processor Cluster – TPC), ενώ κάθε οκτάδα ονομάζεται Πολυεπεξεργαστικό Στοιχείο Υπολογισμών σε Ροές (Streaming Multiprocessor).



Σχήμα 3-4 Μπλοκ διάγραμμα της κάρτας γραφικών GeForce 8800 Ultra.

3.2.2 Συστοιχία Επεξεργαστών Υφών – Texture Processor Clusters (TPC)

Τα δομικά στοιχεία στην αρχιτεκτονική G80 είναι τα TPCs. Αυτά, όσον αφορά την επεξεργασία γραφικών, αναλαμβάνουν ό,τι έχει να κάνει με την επεξεργασία υφών. Το κάθε TPC έχει τη δική του κρυφή μνήμη υφών και οι επεξεργαστικοί πυρήνες που περιλαμβάνει έχουν πρόσβαση σε αυτή την κοινή κρυφή μνήμη, όσον αφορά τις υφές, και μπορούν να συνεργάζονται. Στο Σχήμα 3-5 φαίνονται τα δομικά τμήματα των TPCs.



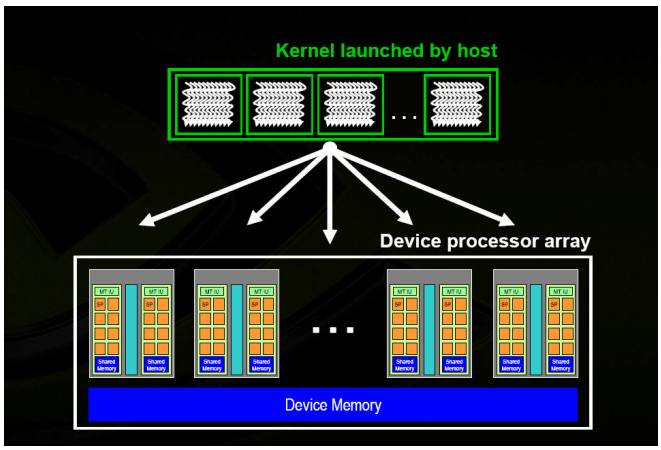
Σχήμα 3-5 Περιεχόμενα μιας Επεξεργαστικής Συστοιχίας Textures.

Το κάθε TPC περιλαμβάνει δυο Πολυεπεξεργαστές Υπολογισμών σε Ροές (Streaming Multiprocessors – SMs) και μια Μονάδα Υφών (TEX). Η Μονάδα Υφών περιλαμβάνει την κρυφή μνήμη υφών και υλικό που χρησιμοποιείται κυρίως για επεξεργασία γραφικών. Συνεπώς, δε θα μας απασχολήσει ιδιαίτερα για την ανάπτυξη εφαρμογών γενικού σκοπού. Αντίθετα, οι SMs είναι αυτά που ουσιαστικά χρησιμοποιούνται για γενικές εφαρμογές.

3.2.3 Πολυεπεξεργαστής Υπολογισμών σε Ροές - Streaming Multiprocessor (SM)

Υπάρχουν δυο SMs σε κάθε TPC, ο καθένας από τους οποίους περιλαμβάνει οκτώ Επεξεργαστές Υπολογισμών σε Ροές (Streaming Processors – SPs), δυο Μονάδες Ειδικών Λειτουργιών (Special Function Units – SFUs), την διαμοιραζόμενη μνήμη μεγέθους 16 KB, το αρχείο καταχωρητών (Register File- RF), μεγέθους 8192 32-bit καταχωρητών καθώς και τη Μονάδα Προσκόμισης και Διάδοσης Εντολών (Instruction Fetch/Dispatch Unit).

Κάθε συσκευή ικανή να υποστηρίξει το CUDA (CUDA-capable device) μπορεί να περιλαμβάνει διαφορετικό αριθμό TPCs, και κατά συνέπεια SMs και SPs, χωρίς να επηρεάζεται ο τρόπος με το οποίον προγραμματίζουμε. Με άλλα λόγια, το CUDA μπορεί και εκμεταλλεύεται οποιαδήποτε συσκευή έχουμε στη διάθεσή μας και σκοπός του είναι να κλιμακώνεται ικανοποιητικά χωρίς να χρειάζονται τα προγράμματά μας εκ νέου μεταγλώττιση. Ένας υπολογιστικός πυρήνας μπορεί να εκτελεστεί σε οσαδήποτε SMs έχει το μηχάνημα μας ανεξάρτητα με την ποσότητα των SMs του μηχανήματος όπως βλέπουμε και από το Σχήμα 3-6.



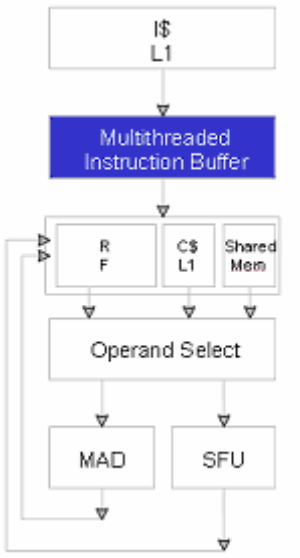
Σχήμα 3-6 Διαμοιρασμός εργασιών ενός υπολογιστικού πυρήνα σε μεταβλητό και αρχικά άγνωστο αριθμό SMs.

Στο Σχήμα 3-7 φαίνεται ένας πίνακας, ο οποίος περιλαμβάνει τον αριθμό SMs σε όλες τις κάρτες γραφικών του εμπορίου που υποστηρίζουν το CUDA.

Συσκευή	Αριθμός SMs στη συσκευή
GeForce 800 Ultra, 8800 GTX	16
GeForce 8800 GT	14
GeForce 8800M GTX	12
GeForce 8800 GTS	12
GeForce 8800M GTS	8
GeForce 8600 GTS, 8600 GT, 8700M GT, 8600M GT, 8600M GS	4
GeForce 8500 GT, 8400 GS, 8400M GT, 8400M GS	2
GeForce 8400M G	1
Tesla S870	4x16
Tesla D870	2x16
Tesla C870	16
Quadro Plex 1000 Model S4	4x16
Quadro Plex 1000 Model IV	2x16
Quadro FX 5600	16
Quadro FX 4600	12
Quadro FX 1700, FX 570, NVS 320M, FX 1600M, FX 570M	4
Quadro FX 370, NVS 290, NVS 140M, NVS 135M, FX 360M	2
Quadro NVS 130M	1

Σχήμα 3-7 Αριθμός SMs στα προϊόντα της nVidia που βασίζονται στη αρχιτεκτονική G80.

Πολύ βοηθητικό σχήμα για την κατανόηση των σταδίων που περνά μια οποιαδήποτε εντολή κατά την εκτέλεσή της μέσα σε ένα SM είναι το Σχήμα 3-8.



Σχήμα 3-8 Δρόμος που ακολουθεί κάθε εντολή σε ένα SM.

Στο Σχήμα 3-8 βλέπουμε ότι αρχικά αποφασίζεται ποια εντολή που υπάρχει στην κρυφή μνήμη εντολών επιπέδου 1 (Level 1 Instruction Cache – I$ L1 στο Σχήμα 3-8) θα εκτελεστεί. Αυτή η εντολή αποκωδικοποιείται και επιλέγεται κάποιος τελεστής (στάδιο Operand Select) ο οποίος θα επιδράσει πάνω στα δεδομένα εισόδου της εντολής. Αυτά τα δεδομένα προσκομίζονται από το αρχείο καταχωρητών, την διαμοιραζόμενη μνήμη (shared memory) ή/και τις κρυφές μνήμες των υφών και των σταθερών μνημών (C$ L1). Ανάλογα με τη φύση της εντολής χρησιμοποιείται είτε ο MAD (ικανός να εκτελέσει πράξεις πολλαπλασιασμού και πρόσθεσης με μια εντολή) επεξεργαστικός πυρήνας (δηλαδή, ο κάθε SP) είτε η Μονάδα Ειδικών Συναρτήσεων (SFU). Τέλος, τα αποτελέσματα γράφονται σε κάποιον καταχωρητή ή σε κάποια θέση της καθολικής μνήμης.

3.2.4 Επεξεργαστές Υπολογισμών σε Ροές - Streaming Processor (SP)

Οι SPs αποτελούν τους επεξεργαστικούς πυρήνες στους οποίους εκτελούνται όλες οι εντολές είτε πρόκειται για επεξεργασία γραφικών είτε υπολογισμούς για γενικού σκοπού. Η συχνότητα λειτουργίας τους διαφέρει από μοντέλο σε μοντέλο και κυμαίνεται μεταξύ 800 MHz και 1625 MHz. Επίσης, ο συνολικός τους αριθμός στο τσιπ της κάρτας γραφικών εξαρτάται άμεσα από τον αριθμό των SMs στο τσιπ, καθώς σε κάθε SM περιλαμβάνονται οκτώ SPs. Οι SPs είναι ικανοί να εκτελέσουν εντολές που περιλαμβάνουν αναγνώσεις/εγγραφές μνήμης, προσκόμιση δεδομένων υφών, λειτουργίες συγχρονισμού, αποτίμηση/αποφάσεις συνθηκών/διακλαδώσεων, πράξεις μεταξύ 32-bit ακεραίων (integers) και πράξεις μεταξύ αριθμών κινητής υποδιαστολής μονής ακρίβειας (32-bit floats) με μικρές αποκλίσεις από το πρότυπο IEEE-754 (βλ. Σχήμα 3-2) και μετατροπές τύπου μεταξύ integer και float δεδομένων. Πρέπει, επίσης, να τονιστεί ότι είναι εφικτό να γίνει διπλή έκδοση (dual-issue) μιας εντολής MAD με μια εντολή MUL, κάτι που επιτρέπει σημαντικά μεγάλες ταχύτητες εκτέλεσης πράξεων. Αυτή η τελευταία δυνατότητα θα εξερευνηθεί στο Κεφάλαιο 5 στο αντίστοιχο πείραμα.

Ενδεικτικά, θα αναφέρουμε πόσους κύκλους ρολογιού χρειάζονται για να εκτελεστούν κάποιες πράξεις. Οι προσθέσεις (ADD), αφαιρέσεις (SUB), πολλαπλασιασμοί (MUL) και οι πολλαπλασιασμοί/προσθέσεις (MAD–Multiply ADd) μεταξύ αριθμών κινητής υποδιαστολής και οι προσθέσεις μεταξύ integers χρειάζονται 4 κύκλους. Οι 32-bit πολλαπλασιασμοί μεταξύ integers διαρκούν 16 κύκλους ρολογιού ενώ οι 24-bit πολλαπλασιασμοί (MUL_24) μεταξύ integers διαρκούν μόλις 4 κύκλους. Η διαίρεση (DIV) μεταξύ floats διαρκεί 36 κύκλους, ενώ η διαίρεση (DIV) και υπόλοιπο (MOD) μεταξύ integers είναι πολύ πιο χρονοβόρες, διαρκώντας μεγάλο αριθμό κύκλων ρολογιού.

Στην τελευταία έκδοση του CUDA (1.1) υποστηρίζονται επίσης κάποιες ατομικές συναρτήσεις (atomic functions) αλλά αυτή η έκδοση δεν υποστηρίζεται από όλες τις κάρτες γραφικών της σειράς 8 (βλ. ενότητα 3.4.2.1.3). Στο Σχήμα 3-9 φαίνονται όλες οι κάρτες με την τελευταία έκδοση του CUDA που αυτές υποστηρίζουν:

Συσκευή	Υποστηριζόμενη έκδοση CUDA
GeForce 8800 Ultra, 8800GTX	1.0
GeForce 8800GT	1.1
GeForce 8800M GTX	1.1
GeForce 8800 GTS	1.0
GeForce 8600 GTS, 8600GT, 8700M GT, 8600M GT, 8600M GS	1.1
GeForce 8500 GT, 8400 GS, 8400M GT	1.1
Tesla S870	1.0
Tesla D870	1.0
Tesla C870	1.0

Σχήμα 3-9 Έκδοση του CUDA που υποστηρίζουν τα διαθέσιμα στο εμπόριο προϊόντα της nVidia.

3.2.5 Μονάδα Ειδικών Συναρτήσεων - Special Function Unit (SFU)

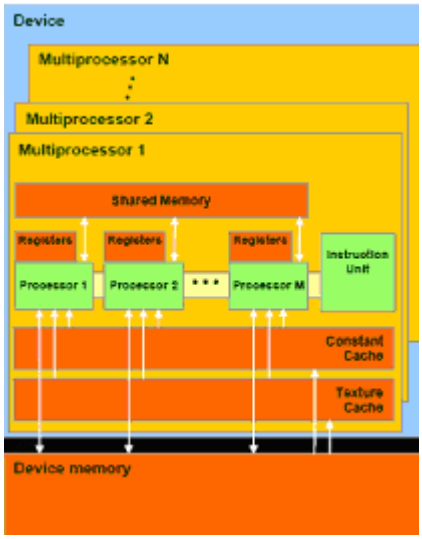
Η SFU (Special Function Unit) είναι υπεύθυνη για αποτιμήσεις των αποτελεσμάτων κάποιων ειδικών συναρτήσεων καθώς και για την παρεμβολή εικονοστοιχείων, όταν χρησιμοποιείται για υπολογισμούς γραφικών. Οι συναρτήσεις τις οποίες αυτή η Ειδική Μονάδα προσεγγίζει είναι το τετράγωνο αριθμού (RCP), η τετραγωνική ρίζα (RSQRT), ο λογάριθμος με βάση 2 (LOG2), το εκθετικό (EXP2) καθώς και οι τριγωνομετρικές (SIN, COS).

3.2.6 Ιεραρχία Μνήμης

Κάθε SM έχει πρόσβαση σε διαφόρων ειδών μνήμες:

- *Αρχείο καταχωρητών (Register file):* Κάθε SM περιλαμβάνει 8192 καταχωρητές μεγέθους 32 bit, στους οποίους έχουν πρόσβαση όλα τα SPs του SM.
- *Διαμοιραζόμενη μνήμη (Shared memory)*: Κάθε SM περιλαμβάνει μια μνήμη η οποία μοιράζεται από τα SPs που περιλαμβάνει και έχει μέγεθος 16 KB.
- *Καθολική μνήμη (Global memory)*: Κάθε επεξεργαστικός πυρήνας έχει πρόσβαση ανάγνωσης/εγγραφής στην καθολική μνήμη της κάρτας γραφικών. Το μέγεθος αυτής της μνήμης κυμαίνεται, ανάλογα με το μοντέλο της κάρτας γραφικών από 128 MB μέχρι και 1,5 GB.
- *Κρυφή μνήμη σταθερών (Constant cache memory)*: Όλοι οι επεξεργαστές ενός SM μοιράζονται μια κοινή κρυφή μνήμη, η οποία επιταχύνει τις αναγνώσεις μνήμης από την μνήμη σταθερών (Constant memory) της κάρτας γραφικών. Στην μνήμη αυτή επιτρέπονται μόνο αναγνώσεις. Η μνήμη σταθερών αποτελεί μέρος της καθολικής μνήμης.
- *Κρυφή μνήμη υφών (Texture cache memory)*: Οι επεξεργαστές ενός SM μοιράζονται επίσης μια κρυφή μνήμη, η οποία επιταχύνει τις αναγνώσεις από την μνήμη υφών της κάρτας γραφικών, η οποία επίσης είναι διαθέσιμη μόνο για ανάγνωση. Όπως και η μνήμη σταθερών, έτσι και η μνήμη υφών (Texture memory) αποτελεί μέρος της καθολικής μνήμης.

Τα προηγούμενα γίνονται πιο ξεκάθαρα με τη βοήθεια του Σχήματος 3-10.



Σχήμα 3-10 Ιεραρχία μνήμης και γενική εποπτεία της device.

Παρατηρώντας το Σχήμα 3-10, μπορεί κανείς να διακρίνει το αρχείο καταχωρητών που αντιστοιχεί σε κάθε SM και διαμοιράζεται στα SPs του. Επίσης, διακρίνεται η διαμοιραζόμενη μνήμη, η κρυφή μνήμη σταθερών, η κρυφή μνήμη υφών που μοιράζονται οι SPs ενός SM, καθώς και η καθολική μνήμη (device memory στο Σχήμα 3-10), στην οποία έχουν πρόσβαση όλοι οι επεξεργαστικοί πυρήνες, ανεξάρτητα του σε ποιό SM ανήκουν. Βλέπει κανείς, επίσης, ότι οι κρυφές μνήμης σταθερών και υφών επικοινωνούν με την καθολική μνήμη, καθώς τα δεδομένα που περιέχονται σε αυτές προέρχονται από κάποια τμήματα της καθολικής μνήμης τα οποία είναι αφιερωμένα στις μνήμες σταθερών και υφών, αντίστοιχα. Είναι σαφές, πλέον, ότι η καθολική μνήμη, η μνήμη υφών και η μνήμη σταθερών προέρχονται από το ίδιο κομμάτι φυσικής μνήμης της συσκευής.

3.2.7 Καθολική Μνήμη Συσκευής - Global device memory

Όπως αναφέρθηκε και προηγουμένως, το μέγεθος της καθολικής μνήμης εξαρτάται από το μοντέλο της κάρτας γραφικών και κυμαίνεται μεταξύ 128 MB και 1,5 GB. Το εύρος ζώνης της, όσον αφορά την επικοινωνία της με τους SPs, κυμαίνεται θεωρητικά στο κορυφαίο μοντέλο της σειράς μέχρι και τα 86,4 GB/s. Αν και το εύρος ζώνης αυτό μοιάζει εξαιρετικά μεγάλο, είναι σχετικά εύκολο να κορεστεί, καθώς υπάρχουν διαθέσιμα πολλά SMs. Αν αυτά χρειάζονται συχνά προσβάσεις στην καθολική μνήμη, το εύρος ζώνης αποδεικνύεται ανεπαρκές και αποτελεί την κύρια στενωπό της επίδοσης των εφαρμογών μας σε αυτήν την πλατφόρμα, όπως θα αποδειχτεί και στα επόμενα κεφάλαια. Αυτή η μνήμη είναι ορατή (είτε για εγγραφές είτε για αναγνώσεις) από όλα τα νήματα που τρέχουν ανά πάσα στιγμή στην κάρτα γραφικών, αποτελεί δηλαδή κοινό σημείο αναφοράς για κάθε εφαρμογή που εκτελείται στην GPU.

Η καθολική μνήμη είναι η μνήμη στην οποία μπορεί κανείς να επέμβει άμεσα στο σύστημα μνήμης των GPUs, μέσω της επικοινωνίας αυτής με τη CPU και την κύρια μνήμη. Η επικοινωνία επιτυγχάνεται μέσω του διαύλου PCIe και η θεωρητικά μέγιστη ταχύτητα επικοινωνίας, μέσω αυτού του διαύλου, αγγίζει τα 4 GB/s. Ουσιαστικά,

δηλαδή, μέσω του παραπάνω διαύλου επικοινωνίας, εισάγονται τα δεδομένα εισόδου της εκάστοτε εφαρμογής στην καθολική μνήμη και όταν ολοκληρωθεί η εκτέλεση των πυρήνων υπολογισμού, από εκεί θα συλλεχθούν τα τελικά αποτελέσματά.

Το κύριο πλεονέκτημα της καθολικής μνήμης είναι το αρκετά μεγάλο μέγεθός της για ενσωματωμένη μνήμη σε τσιπ γραφικών, αλλά σημαντικό μειονέκτημα αποτελεί η ταχύτητα προσπέλασής της από τα νήματα εκτέλεσης (εκατοντάδες κύκλοι ρολογιού). Πρέπει να σημειωθεί, επίσης, πως δεν υπάρχει κάποια κρυφή μνήμη προκειμένου να «κρύβεται» κατά ένα ποσοστό ο μεγάλος χρόνος πρώτης προσπέλασης (latency). Αυτό έγινε σκόπιμα, προκειμένου να μην προκύπτουν προβλήματα συνέπειας/συνάφειας μνήμης, καθώς για να υπάρχει πρόβλεψη αποφυγής τέτοιων προβλημάτων, θα έπρεπε να εφαρμοστεί σε υλικό ο κατάλληλος μηχανισμός. Κάτι τέτοιο θα αύξανε πολύ το κόστος και θα είχε σίγουρα επιπτώσεις στην προσπάθεια επίτευξης υψηλών επιδόσεων.

Περισσότερες πληροφορίες για το πώς επιτυγχάνεται η καλύτερη δυνατή επίδοση στις προσπελάσεις στην καθολική μνήμη, θα παρουσιαστούν στο Κεφάλαιο 4.

3.2.8 Αρχείο Καταχωρητών - Register File

Σε κάθε SM αναφέρθηκε ότι υπάρχει ένα αρχείο καταχωρητών, μεγέθους 8192 32-bit καταχωρητών (32 KB). Ο χρόνος προσπέλασής οποιουδήποτε καταχωρητή είναι μηδαμινός. Το αρχείο καταχωρητών διαμοιράζεται εξ' ίσου από όλα τα νήματα εκτέλεσης που εκτελούνται ταυτόχρονα στο συγκεκριμένο SM. Έτσι, κάθε νήμα μπορεί να έχει πρόσβαση μόνο στους καταχωρητές που του έχουν ανατεθεί και σε κανέναν άλλον. Σε περιπτώσεις που το αρχείο καταχωρητών δεν είναι αρκετό για τις ανάγκες καθενός νήματος της εφαρμογής μας τότε γίνεται χρήση της τοπικής μνήμης.

3.2.9 Τοπική Μνήμη - Local Memory

Αυτή η μνήμη αποτελεί ένα μέρος των φυσικών διευθύνσεων της καθολικής μνήμης και η χρήση της είναι αποκλειστικά ως συμπλήρωμα του αρχείου καταχωρητών σε περιπτώσεις που οι καταχωρητές του SM δεν επαρκούν για τις ανάγκες των νημάτων εκτέλεσης που έχουν ανατεθεί στον συγκεκριμένο SM. Αποτελεί μια μνήμη τόσο αργή όσο και η καθολική μνήμη, πράγμα αναμενόμενο, αφού αυτές οι μνήμες βρίσκονται στην ίδια φυσική μνήμη. Η τοπική μνήμη χρησιμοποιείται μόνο σε περιπτώσεις «ανάγκης», υπερχείλισης δηλαδή του αρχείου καταχωρητών, χωρίς να μπορεί ο προγραμματιστής να τη δεσμεύσει για οποιαδήποτε χρήση ή να τη χρησιμοποιήσει κατ' επιλογή.

3.2.10 Διαμοιραζόμενη Μνήμη - Shared Memory

Κάθε SM περιλαμβάνει μία διαμοιραζόμενη μνήμη μεγέθους 16 KB. Αυτή η μνήμη χρησιμοποιείται σαν κοινό σημείο αναφοράς μεταξύ των SPs που περιλαμβάνει το SM. Ουσιαστικά, οκτώ διαφορετικοί επεξεργαστικοί πυρήνες έχουν πρόσβαση στη μνήμη αυτή. Ο χρόνος προσπέλασης σε κάποια λέξη που είναι αποθηκευμένη σε αυτή τη μνήμη είναι τόσο μικρός όσο και ο χρόνος προσπέλασης μιας λέξης που βρίσκεται στο αρχείο καταχωρητών.

Η διαμοιραζόμενη μνήμη είναι οργανωμένη σε 16 διαφορετικές τράπεζες (banks), οι οποίες μπορούν να προσπελαστούν την ίδια στιγμή ανεξάρτητα η μία από την άλλη χωρίς καμία επιπλέον επιβάρυνση. Με άλλα λόγια, την ίδια στιγμή, 16 διαφορετικά νήματα εκτέλεσης μπορούν να προσπελάσουν 16 διαφορετικές 32-bit λέξεις, αν αυτές ανήκουν σε διαφορετικές τράπεζες. Οι λέξεις είναι οργανωμένες σε τράπεζες με τέτοιο τρόπο, ώστε διαδοχικές λέξεις αποθηκευμένες σε διαδοχικές θέσεις μνήμης να αντιστοιχούν σε διαδοχικές τράπεζες.

Λεπτομέρειες σε σχέση με την καλύτερη επίδοση όσον αφορά την προσπέλαση αυτής της μνήμης από πολλαπλά νήματα, ώστε οι αναγνώσεις και οι εγγραφές να είναι πιο αποδοτικές θα αναλυθούν στο Κεφάλαιο 4.

3.2.11 Constant Memory –Μνήμη Σταθερών

Η μνήμη σταθερών έχει μέγεθος 64 KB και είναι κοινή για ολόκληρη την κάρτα γραφικών. Από τη μνήμη αυτή επιτρέπονται μόνο αναγνώσεις από τους SPs.. Αναφέρθηκε και προηγουμένως πως σε κάθε SM υπάρχει μια κρυφή μνήμη σταθερών στην οποία μεταφέρονται αντίγραφα των δεδομένων που ζητούν οι SPs να αναγνώσουν. Η κάθε κρυφή μνήμη σταθερών έχει μέγεθος 8 KB. Ο χρόνος προσπέλασης για μια λέξη η οποία βρίσκεται στην κρυφή μνήμη αυτήν είναι ουσιαστικά ίδιος με τον χρόνο που χρειάζεται να διαβαστεί ένας καταχωρητής από το αρχείο καταχωρητών. Αντίθετα, στην περίπτωση που συμβαίνει αστοχία κρυφής μνήμης (cache miss), ο χρόνος που απαιτείται είναι ίσος με αυτόν που χρειάζεται για να μεταφερθεί από την μνήμη σταθερών στην αντίστοιχη κρυφή μνήμη. Δεδομένου ότι πρόκειται για μια προσπέλαση της καθολικής μνήμης, ο χρόνος ισούται με μια ανάγνωση από την καθολική μνήμη.

3.2.12 Μνήμη Υφών - Texture Memory

Αυτή η μνήμη λειτουργεί παρόμοια με την μνήμη σταθερών. Προσπελάζεται από τα SPs μέσω μιας κρυφής μνήμης υφών μεγέθους 8 KB ανά SM. Είναι και αυτή διαθέσιμη μόνο για ανάγνωση. Για τους χρόνους προσπέλασης ισχύει ότι αν η λέξη που θέλουμε είναι στην κρυφή μνήμη, ο χρόνος είναι ίδιος με την περίπτωση που προσπελάζουμε έναν καταχωρητή από το αρχείο καταχωρητών. Σε αντίθετη περίπτωση, αυτός ο χρόνος επιβαρύνεται με τη μεταφορά της λέξης από την μνήμη υφών στην κρυφή μνήμη, χρόνος που μπορεί να είναι εκατοντάδες κύκλοι μηχανής, όπως και σε οποιαδήποτε περίπτωση προσπέλασης οποιουδήποτε μέρους της καθολικής μνήμης. Η διαφορά είναι ότι η κρυφή μνήμη υφών είναι βελτιστοποιημένη ώστε να εκμεταλλεύεται την τοπικότητα σε δυο διαστάσεις (2D πεδίο). Αυτό οφείλεται στη χρήση της μνήμης υφών για εφαρμογές γραφικών που «γράφουν» τα αποτελέσματά τους σε μια δισδιάστατη έξοδο, την οθόνη ή οποιοδήποτε άλλο δισδιάστατο μέσο απεικόνισης.

3.3 Το Μοντέλο Εκτέλεσης

Σε αυτήν την ενότητα θα παρουσιαστεί ο τρόπος με τον οποίο η προγραμματιστική διεπαφή του CUDA χρησιμοποιεί την αρχιτεκτονική G80, η οποία παρουσιάστηκε με αρκετή λεπτομέρεια στην προηγούμενη ενότητα. Ουσιαστικά, θα αναλυθεί το πώς οργανώνονται τα νήματα εκτέλεσης και πώς ανατίθενται στα διαθέσιμα SPs, ώστε να γίνει κατανοητό το μοντέλο εκτέλεσης της πολυπύρηνης πλατφόρμας.

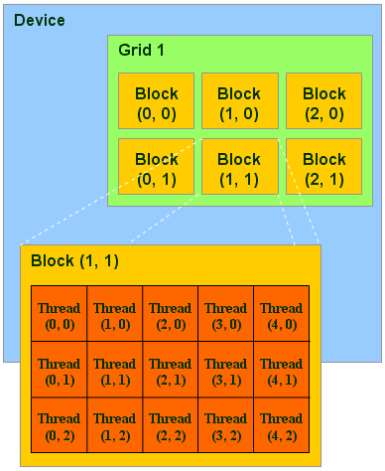
Έχει ειπωθεί ότι η GPU (ή αλλιώς συσκευή όπως θα αναφέρεται από εδώ και στο εξής) αποτελεί έναν πολυνηματικό συνεπεξεργαστή της CPU (ή αλλιώς host). Εκτελεί, δηλαδή, παράλληλα και μαζικά πολλά νήματα στα SMs που διαθέτει. Ουσιαστικά, αυτά τα SMs είναι SIMT (Single Instruction Multiple Thread) επεξεργαστές (κατά τον όρο SIMD), δηλαδή εκτελούν την ίδια, κοινή, εντολή πολλών νημάτων «ταυτόχρονα»-παράλληλα.

Τα προγράμματα που εκτελούνται έχουν τη μορφή πυρήνων υπολογισμών. Κάθε νήμα εκτέλεσης που δημιουργείται εκτελεί τον παραπάνω κώδικα παρράληλα με όλα τα άλλα. Τα νήματα αυτά είναι πολύ «ελαφριά», με την έννοια ότι δεν επιβαρύνουν καθόλου το σύστημά κατά τη δημιουργία τους. Επίσης, υποστηρίζονται χιλιάδες νημάτων και για να φτάσει η συσκευή στα όρια των δυνατών επιδόσεων της, απαιτείται μεγάλος αριθμός από αυτά. Για να γίνει καλύτερη αξιοποίηση της αρχιτεκτονικής G80, καθώς και για να προσδοθεί ευελιξία και ευκολία στον προγραμματιστή, υπάρχει ένας συγκεκριμένος τρόπος οργάνωσης και ανάθεσης των νημάτων εκτέλεσης ανά ομάδες στα SMs.

Ένας πυρήνας εκτελείται ως ένα πλέγμα (grid) από μπλοκ νημάτων. Δηλαδή, όλα τα νήματα στο πλέγμα θα εκτελέσουν τις εντολές που περιλαμβάνει ο αντίστοιχος πυρήνας. Σε κάθε πυρήνα που επιχειρείται να εκτελεστεί αντιστοιχεί ένα και μόνο πλέγμα. Αντίστροφα, κάθε πλέγμα που ξεκινά την εκτέλεσή του μπορεί να εκτελέσει μόνο έναν πυρήνα. Υπάρχει, δηλαδή, μια ένα προς ένα αντιστοιχία πλέγματος-υπολογιστικού πυρήνα. Βέβαια, είναι εφικτό να εκτελεστούν διαδοχικά δύο πυρήνες σε ίδιων διαστάσεων και οργάνωσης πλέγματα, αλλά οι κλήσεις θα είναι τελείως ανεξάρτητες μεταξύ τους. Το πλέγμα αποτελείται από μπλοκ νημάτων εκτέλεσης, τα οποία μπορεί να έχει οργανωμένα σε μία ή δύο διαστάσεις. Το μέγιστο μέγεθος της κάθε διάστασης του πλέγματος είναι 65535 μπλοκ. Το κάθε μπλοκ χαρακτηρίζεται από τις συντεταγμένες του μέσα στο πλέγμα και αυτό βοηθάει στο να διαφοροποιείται η λειτουργία των μπλοκ ανάλογα με τη θέση τους μέσα στο πλέγμα.

Κάθε μπλοκ αποτελείται από έναν αριθμό νημάτων εκτέλεσης, τα οποία μπορεί να είναι οργανωμένα σε ένα μονοδιάστατο, δισδιάστατο ή τρισδιάστατο πεδίο. Ανάλογα με την περίπτωση, έχουν συγκεκριμένο αριθμό συντεταγμένων, οι οποίες τα διακρίνουν από όλα τα άλλα νήματα, ώστε να μπορεί να διαφοροποιηθεί η ακριβής λειτουργία του κάθε νήματος από όλα τα υπόλοιπα. Για παράδειγμα, αν είναι ανάγκη να επεξεργαστούμε ένα πολύ μεγάλο διάνυσμα, χρησιμοποιώντας μονοδιάστατο πλέγμα και μονοδιάστατα μπλοκ, είναι εφικτό να ανατεθούν σε κάθε νήμα η επεξεργασία ενός στοιχείο του μεγάλου διανύσματος, ανάλογα με τον αύξοντα αριθμό του νήματος ως προς το συνολικό πλέγμα. Τα μπλοκ ονομάζονται επίσης και Συνεργαζόμενοι Πίνακες Νημάτων (Cooperative Thread Arrays – CTA). Το μέγιστο μέγεθος ενός μπλοκ είναι τα 512 νήματα, ανεξάρτητα με το πλήθος των διαστάσεων του. Συγκεκριμένα, στη διάσταση *x* η μέγιστη τιμή είναι 512, στη διάσταση *y* η μέγιστη τιμή είναι 512, ενώ στη διάσταση *z* η μέγιστη τιμή είναι 64.

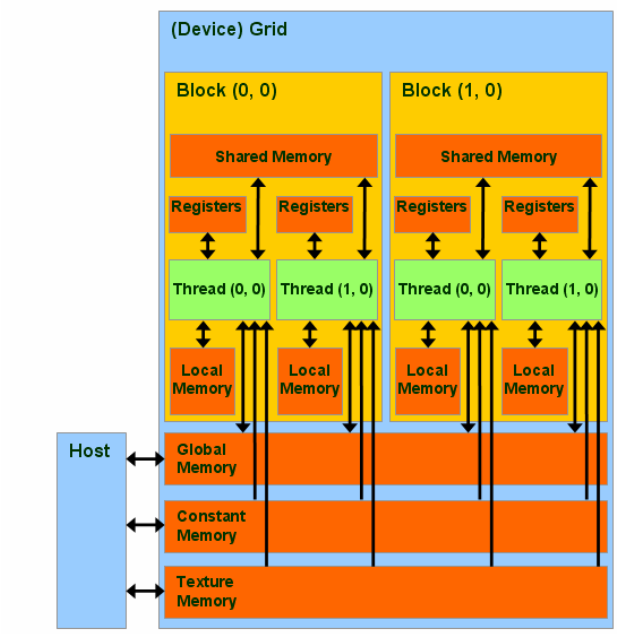
Στο Σχήμα 3-11 μπορεί κανείς να παρατηρήσει μια ενδεικτική οργάνωση ενός πλέγματος σε μπλοκ με τις συντεταγμένες των μπλοκ μέσα στο πλέγμα, καθώς και την οργάνωση ενός από τα μπλοκ (όλα τα άλλα θα είναι όμοια) και τις συντεταγμένες των νημάτων που ανήκουν σε αυτό.



Σχήμα 3-11 Οργάνωση ενός πλέγματος σε μπλοκ και ενός μπλοκ σε νήματα.

Τα νήματα που ανήκουν στο ίδιο μπλοκ μπορούν και συνεργάζονται κυρίως με δύο τρόπους. Μπορούν να συγχρονιστούν μέσω ενός barrier, δηλαδή, όλα τα νήματα ενός μπλοκ σταματούν την εκτέλεσή τους σε ένα σημείο, στο οποίο καλείται από όλα τα νήματα μια συγκεκριμένη εντολή (`__syncthreads()`, βλ. ενότητα 3.4.2.1.2) και συνεχίζουν αφού φτάσουν σε εκείνο το σημείο του κώδικα του πυρήνα. Επίσης όλα τα νήματα ενός μπλοκ έχουν πρόσβαση στη διαμοιραζόμενη μνήμη ενός SM και μπορούν μέσω αυτής να δουλέψουν συνεργατικά σε κάποιο σύνολο δεδομένων. Ένα μπλοκ ανατίθεται, σε επίπεδο υλικού, εξ'ολοκλήρου σε κάποιο από τα SMs για όλη τη διάρκεια της ζωής του, δεσμεύοντας κάποιους από τους πόρους του συγκεκριμένου SM. Η διάρκεια της ζωής του μπλοκ είναι ουσιαστικά ο απαιτούμενος χρόνος ώστε όλα τα νήματα που ανήκουν σε αυτό να εκτελέσουν όλες τις εντολές του υπολογιστικού πυρήνα. Ο αριθμός των μπλοκ που μπορούν να ανατεθούν την ίδια χρονική στιγμή σε ένα SM εξαρτάται από τους διαθέσιμους πόρους του SM και τις αντίστοιχες απαιτήσεις του μπλοκ. Ένας περιορισμός είναι ότι κάθε SM περιλαμβάνει ένα αρχείο καταχωρητών με 8192 καταχωρητές και 16 KB διαμοιραζόμενης μνήμης. Άλλος ένας περιορισμός ως προς το πόσα μπλοκ μπορούν να φιλοξενηθούν την ίδια στιγμή από το ίδιο SM είναι ότι σε μια δεδομένη στιγμή ένα SM μπορεί να περιλαμβάνει μέχρι 768 ενεργά νήματα εκτέλεσης, ανεξάρτητα από τον αριθμό των μπλοκ .Τέλος, κάθε SM μπορεί να έχει αναλάβει σε μια δεδομένη στιγμή το πολύ 8 μπλοκ. Έτσι αν κάποιο μπλοκ έχει μέγεθος 256 νήματα και οι απαιτήσεις του σε διαμοιραζόμενη μνήμη είναι 4 KB, κάθε SM μπορεί να «φιλοξενεί» κάθε δεδομένη στιγμή 3 μπλοκ. Αν το μέγεθος του μπλοκ είναι 128 και οι απαιτήσεις του σε ποσό διαμοιραζόμενης μνήμης είναι 4 KB, μπορούν να «φιλοξενηθούν» 4 μπλοκ σε κάθε SM. Αν οι απαιτήσεις σε διαμοιραζόμενη μνήμη είναι 16 KB, σε κάθε SM δε μπορεί να «χωρέσει» πάνω από ένα μπλοκ σε μια δεδομένη χρονική στιγμή, ανεξάρτητα από το μέγεθος του μπλοκ. Εάν οι απαιτήσεις ενός μπλοκ είναι τέτοιες που δε μπορεί να το φιλοξενήσει κανένα SM μόνο του, τότε ο υπολογιστικός πυρήνας δε θα μπορέσει να εκκινήσει την εκτέλεσή του.

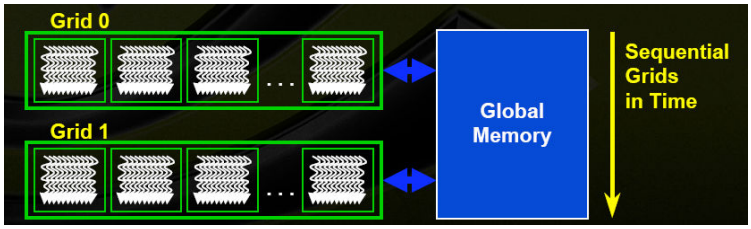
Στο Σχήμα 3-12 φαίνονται σχηματικά σε ποιές από τις μνήμες που περιλαμβάνει η συσκευή έχουν πρόσβαση τα νήματα ενός υποθετικού πλέγματος :



Σχήμα 3-12 Πρόσβαση σε όλα επίπεδα ιεραρχίας μνήμης από νήματα που ανήκουν στο ίδιο ή σε διαφορετικό μπλοκ ενός πλέγματος.

Στο Σχήμα 3-12 υπάρχουν δύο μπλοκ μεγέθους δύο νημάτων το καθένα[2]. Παρατηρούμε ότι τα νήματα κάθε μπλοκ έχουν πρόσβαση στον κοινό χώρο διαμοιραζόμενης μνήμης. Επίσης, το κάθε νήμα, ανεξάρτητα από το μπλοκ στο οποίο ανήκει, έχει πρόσβαση στους δικούς του «ιδιωτικούς» καταχωρητές και δυνητικά (εάν δηλαδή το αρχείο καταχωρητών δεν επαρκεί για την κάλυψη των αναγκών των νημάτων) στο δικό του κομμάτι της τοπικής μνήμης. Εκτός των παραπάνω, όλα τα νήματα εκτέλεσης έχουν πρόσβαση στην καθολική μνήμη, καθώς και στις μνήμες σταθερών και υφών.

Νήματα που ανήκουν σε διαφορετικά μπλοκ δε μπορούν να συνεργαστούν ή να συγχρονιστούν με κάποιον τρόπο παρεχόμενο από την προγραμματιστική διεπαφή του CUDA. Μόνη λύση για κάτι τέτοιο είναι να μοιραστούν σε πολλαπλούς και διαδοχικούς υπολογιστικούς πυρήνες οι λειτουργίες του αλγορίθμου που πρόκειται να εκτελεστούν και έτσι να υπάρξει ένα είδος εξωτερικού συγχρονισμού μεταξύ των υπολογιστικών πυρήνων. Αυτό θα συμβεί, γιατί καθώς οι υπολογιστικοί πυρήνες εκτελούνται ακολουθιακά ο ένας μετά τον άλλον, τερματισμός ενός πυρήνα σημαίνει ότι όλα τα νήματα που τον εκτέλεσαν έχουν ολοκληρωθεί, οπότε έμμεσα δημιουργείται ένα καθολικό barrier συγχρονισμού. Ενδεικτικό του παραπάνω είναι το Σχήμα 3-13.



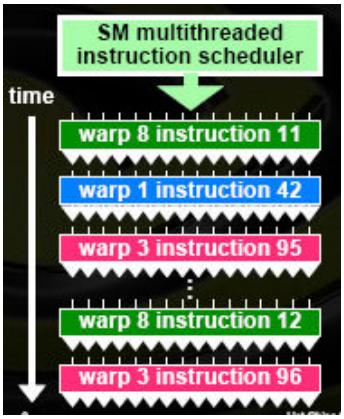
Σχήμα 3-13 Συγχρονισμός μεταξύ της εκτέλεσης δυο διαφορετικών υπολογιστικών πυρήνων. Αυτό το σχήμα συγχρονισμού είναι και το μοναδικό καθολικό σχήμα συγχρονισμού.

[2] Το παράδειγμα είναι τελείως ενδεικτικό και εκτός πραγματικότητας, καθώς με μια τέτοια οργάνωση δεν μπορούμε να πετύχουμε αποδοτική χρησιμοποίηση της συσκευής.

Είναι σαφές από τα παραπάνω ότι δεν εκτελούνται όλα τα νήματα του πλέγματος πραγματικά παράλληλα αλλά ουσιαστικά παράλληλα. Δηλαδή, κάθε SM αναλαμβάνει τον αριθμό μπλοκ που μπορεί να εκτελέσει, και αφού ολοκληρώσει τις λειτουργίες που επιτάσσουν οι εντολές του πυρήνα τότε προχωρά στην εκτέλεση της επόμενης «φουρνιάς» (batch) από μπλοκ. Τα μπλοκ που εκτελούνται σε μια δεδομένη χρονική στιγμή από όλα τα SMs ονομάζονται ενεργά (active).

Κάθε ενεργό μπλοκ που έχει δεσμεύσει πόρους σε ένα SM χρονοδρομολογεί την εκτέλεση των νημάτων του ως εξής:

Τα νήματά του χωρίζονται σε ομάδες SIMD νημάτων που ονομάζονται στημόνια (warps). Το ποια νήματα θα ανήκουν σε κάθε στημόνι αποφασίζεται ανάλογα με τον αύξοντα αριθμό των νημάτων (thread ID) και ανατίθενται σε αυτά νήματα με συνεχόμενους αύξοντες αριθμούς. Τα νήματα εκτέλεσης που ανήκουν σε ένα στημόνι εκτελούν ακριβώς τις ίδιες εντολές του υπολογιστικού πυρήνα. Το μέγεθος του στημονιού ονομάζεται warp size και η μέγιστη και συνήθης τιμή που μπορεί να έχει είναι 32 νήματα. Τα στημόνια ενός ενεργού μπλοκ ονομάζονται ενεργά (active) στημόνια. Ο χρονοπρογραμματισμός της εκτέλεσης των στημονιών γίνεται από το SM ανάλογα με την προτεραιότητά τους, με το ποια είναι έτοιμα προς εκτέλεση και ανάλογα με το ποιών στημονιών τα δεδομένα πάνω στα οποία θα εκτελέσουν τις λειτουργίες τους είναι διαθέσιμα. Κάθε απλή εντολή διαρκεί 4 κυκλους (όπως για παράδειγμα η ADD μεταξύ αριθμών κινητής υποδιαστολής). Αυτό συμβαίνει γιατί σε ένα SM το οποίο περιλαμβάνει ένα ενεργό στημόνι μεγέθους 32, ο κάθε ένας από τους SPs εκτελεί σε κάθε κύκλο μια εντολή για κάθε νήμα του στημονιού. Έτσι, σε 4 κύκλους οι 8 επεξεργαστές έχουν εκτελέσει την ίδια εντολή σε κάθε ένα από τα 32 (4 επί 8) νήματα του ενεργού στημονιού Από τον επόμενο κύκλο, το SM είναι έτοιμο να εκτελέσει την επόμενη εντολή ενός άλλου στημονιού ή ακόμα και του ίδιου ανάλογα με τον χρονοπρογραμματισμό που έχει γίνει. Στο Σχήμα 3-14 διακρίνεται η διαδοχική εκτέλεση εντολών που ανήκουν σε διάφορα στημόνια από ένα συγκεκριμένο SM.



Σχήμα 3-14 Χρονοδρομολόγηση εντολών διαφόρων στημονιών εντός ενός SM.

Τονίζεται ότι η χρονοδρομολόγηση γίνεται δυναμικά και ένα SM θα «κολλάει» μόνο όταν δεν υπάρχουν εντολές προς εκτέλεση σε οποιοδήποτε από τα ενεργά στημόνια της δεδομένης χρονικής στιγμής. Αυτό είναι πολύ σημαντικό και για να «κρύβονται» οι καθυστερήσεις εντολών με μεγάλη διάρκεια σε κύκλους μηχανής, όπως οι προσβάσεις στην καθολική μνήμη. Επίσης, η εναλλαγή μεταξύ διαφορετικών στημονιών

γίνεται με μηδενικό κόστος σε χρόνο, προκειμένου να μην σταματά τη λειτουργία του το SM όταν κάποιο στημόνι «κολλάει» περιμένοντας την ολοκλήρωση μιας λειτουργίας μνήμης, ενώ την ίδια στιγμή υπάρχει σε κάποιο άλλο στημόνι μια έτοιμη προς εκτέλεση εντολή.

Το κάθε στημόνι διαιρείται σε δυο ημι-στημόνια (half-warps), δηλαδή σε δυο μισά, μεγέθους 16 νημάτων, (το ένα μέρος περιλαμβάνει τα 16 πρώτα νήματα και το δεύτερο τα 16 τελευταία). Αυτή η διαίρεση έχει άμεση σχέση με τις 16 διαφορετικές τράπεζες (banks) της διαμοιραζόμενης μνήμης και του πώς γίνεται η πρόσβαση σε αυτή.

Σε κάποιες ειδικές περιπτώσεις παρατηρούνται παρεκκλίσεις σε σχέση με τα όσα αναφέρθηκαν παραπάνω. Όταν στον υπολογιστικό πυρήνα υπάρχει κάποια συνθήκη με περισσότερα από ένα πιθανά μονοπάτια, το κάθε στημόνι εκτελεί όλες τις εντολές των πιθανών μονοπατιών και, στο τέλος, αποφασίζεται ποια αποτελέσματα θα κρατηθούν σε κάθε νήμα. Για παράδειγμα, έστω ότι υπάρχει σε ένα σημείο του κώδικα μια συνθήκη που απαιτεί μια ιδιωτική μεταβλητή να αυξηθεί κατά 5, εάν ο αύξων αριθμός του νήματος είναι μεγαλύτερος από 100, αλλιώς να μειωθεί κατά 5. Σε ένα δεδομένο στημόνι θα εκτελεστεί και η πρόσθεση και η αφαίρεση για όλα τα νήματα που ανήκουν σε αυτό, αλλά τελικά θα γραφτούν μόνο τα σωστά αποτελέσματα στις αντίστοιχες ιδιωτικές μεταβλητές των νημάτων, σύμφωνα με την τελική αποτίμηση της συνθήκης. Περισσότερα για αυτά τα θέματα θα αναπτυχθούν στο Κεφάλαιο 4.

3.4 Προγραμματιστικό Περιβάλλον Ανάπτυξης Εφαρμογών του CUDA

Σε αυτή την ενότητα θα παρουσιαστεί ο τρόπος με τον οποίον μπορούν οι GPUs να χρησιμοποιηθούν ώστε να προγραμματιστούν γενικές εφαρμογές. Θα γίνει κατανοητό πώς φαίνονται όλα όσα αναλύθηκαν προηγουμένως από την πλευρά του προγραμματιστή, τι δυνατότητες προσφέρονται και πώς εκμεταλλεύεται το Πακέτο Ανάπτυξης Λογισμικού (Software Development Kit-SDK) του CUDA την αρχιτεκτονική G80 για υπολογισμούς γενικού σκοπού.

3.4.1 Γλώσσα Προγραμματισμού του CUDA

Η γλώσσα που προσφέρει το CUDA είναι μια επέκταση της γλώσσας προγραμματισμού C, με κάποια στοιχεία από τη C++. Οι επεκτάσεις και οι προσθήκες δεν είναι πολύ μεγάλες, απλά παρέχουν ένα συμπλήρωμα, ώστε να δίνουν τη δυνατότητα σε επίδοξους προγραμματιστές να εκμεταλλευτούν όσα προσφέρει η παρουσιαζόμενη τεχνολογία. Ο λόγος για τον οποίο επιλέχθηκε να γίνει κάτι τέτοιο είναι το γεγονός, ότι πρωταρχικός στόχος ήταν να γίνει πιο προσιτό σε όλους να προγραμματίσουν κάρτες γραφικών για γενικές εφαρμογές, και η γλώσσα C είναι ίσως η πιο διαδεδομένη και εύχρηστη γλώσσα ανά τον κόσμο. Θα παρουσιαστούν κατά σειρά οι σημαντικότερες επεκτάσεις που προσφέρονται και χρησιμοποιήθηκαν στο Κεφάλαιο 5. Περισσότερες πληροφορίες και λεπτομέρειες μπορούν να αναζητηθούν στο [23].

3.4.1.1 Χαρακτηριστές Μεταβλητών/Συναρτήσεων – Variable/Function Qualifiers

Υπάρχουν κάποιοι συγκεκριμένοι χαρακτηριστές οι οποίοι χρησιμοποιούνται για να δηλώνουν από πού θα καλούνται και πού θα εκτελούνται οι χαρακτηριζόμενες συ-

ναρτήσεις. Υπενθυμίζεται ότι το CUDA χρησιμοποιεί την GPU σαν συνεπεξεργαστή της CPU και άρα υπάρχει μια συνεχής συνεργασία αυτών των δύο επεξεργαστικών πλατφόρμων. Υπάρχουν, δηλαδή, δυο πλευρές στις οποίες μπορεί να εκτελείται μια συνάρτηση ή από τις οποίες μπορεί να καλείται: η πλευρά του host και η πλευρά της συσκευής.

Υπάρχουν τρεις διαφορετικοί χαρακτηριστές συναρτήσεων:

Α) `__host__`: Αυτός ο χαρακτηριστής χρησιμοποιείται σαν πρόθεμα στις συναρτήσεις που μεταγλωττίζονται και εκτελούνται στον host. Αυτές οι συναρτήσεις μπορούν να κληθούν από άλλες συναρτήσεις του host και μόνο. Όλες οι συναρτήσεις που γράφουμε σε προγράμματα για το CUDA και δεν περιλαμβάνουν κάποιον χαρακτηρίστη, αυτόματα θεωρούνται host συναρτήσεις.

Β) `__device__`: Χρησιμοποιείται για συναρτήσεις οι οποίες μεταγλωττίζονται και εκτελούνται στην συσκευή, αλλά καλούνται μόνο από άλλες συναρτήσεις που εκτελούνται στη συσκευή. Αυτός ο χαρακτηρισμός μπορεί επίσης να χρησιμοποιηθεί και σε συνδυασμό με τον `__host__`, κάτι που θα έχει αποτέλεσμα η εν λόγω διπλά χαρακτηριζόμενη συνάρτηση να μπορεί κληθεί είτε από συναρτήσεις του host είτε από άλλες συναρτήσεις της συσκευής.

Γ) `__global__`: Αυτός ο χαρακτηριστής χρησιμοποιείται ουσιαστικά για τις συναρτήσεις που θα αποτελούν τον υπολογιστικό πυρήνα των εφαρμογών. Καλούνται από τον host μόνο, ενώ μεταγλωττίζονται και εκτελούνται αποκλειστικά στη συσκευή. Πρέπει να επιστρέφουν υποχρεωτικά `void`, σε αντίθεση με τις άλλες δυο κατηγορίες, οι οποίες μπορούν να επιστρέφουν οτιδήποτε.

Το Σχήμα 3-15 συνοψίζει τις κατηγορίες των συναρτήσεων που υποστηρίζονται με τρία παραδείγματα δηλώσεων συναρτήσεων, ένα για κάθε περίπτωση:

	Εκτελείται σε:	**Καλείται μόνο από:**
__device__ float DeviceFunc()	συσκευή	συσκευή
__global__ void KernelFunc()	συσκευή	host
__host__ float HostFunc()	host	host

Σχήμα 3-15 Κατηγορίες Χαρακτηριστών Συναρτήσεων.

Τέλος, αναφέρεται ότι για όσες συναρτήσεις εκτελούνται στην συσκευή (δηλαδή αυτές με χαρακτηρισμούς `__device__` και `__global__`) δεν υποστηρίζεται αναδρομή, απαγορεύονται δηλώσεις `static` μεταβλητών εντός του κώδικα τους και ο αριθμός των παραμέτρων τους δε μπορεί να είναι μεταβλητού μεγέθους.

Όσον αφορά τις μεταβλητές, υπάρχουν κάποιοι χαρακτηριστές οι οποίοι επιτρέπουν να δηλώνεται πού ακριβώς είναι επιθυμητό να «στεγάζεται» η κάθε μία, πράγμα που δίνει τη δυνατότητα να αξιοποιήσουμε κατ'επιλογή όλη την ιεραρχία της μνήμης της αρχιτεκτονικής G80.

Υπάρχουν και εδώ τρεις κατηγορίες χαρακτηριστών:

Α) `__device__`: Αυτός ο χαρακτηριστής δηλώνει ότι η μεταβλητή την οποία χαρακτηρίζει ανήκει στο χώρο μνήμης της συσκευής. Μπορεί να χρησιμοποιηθεί σε συνδυασμό με τους άλλους χαρακτηρισμούς, αλλά εάν είναι ο μοναδικός, σηματοδοτεί ότι αυτή η μεταβλητή θα στεγάζεται στην καθολική μνήμη. Η διάρκεια ζωής της μεταβλητής που χαρακτηρίζει είναι όση και η διάρκεια της εφαρμογής. Αυτές οι μεταβλητές μπορούν αν προσπελαστούν από όλα τα νήματα της εκάστοτε εφαρμογής, ενώ μέσω της προγραμματιστικής διεπαφής που προσφέρει το CUDA, είναι εφικτό να προσπελαστούν και από τον host.

Β) `__constant__`: Αυτός ο χαρακτηριστής μπορεί να χρησιμοποιηθεί είτε μόνος είτε σε συνδυασμό με τον `__device__`, δηλώνοντας ότι η μεταβλητή η οποία χαρακτηρίζει βρίσκεται στη μνήμη σταθερών. Η διάρκεια ζωής αυτών των μεταβλητών είναι ίση με το χρόνο ζωής της εφαρμογής στην οποία χρησιμοποιείται. Μπορούν να προσπελαστούν από όλα τα νήματα της εφαρμογής, μόνο για αναγνώσεις βέβαια, ενώ έχει πρόσβαση σε αυτή και ο host ο οποίος μπορεί να την αλλάξει, διαβάσει και γράψει μέσω της προγραμματιστικής διεπαφής.

Γ) `__shared__`: Αυτός ο χαρακτηριστής μπορεί να χρησιμοποιηθεί είτε μόνος, είτε σε συνδυασμό με τον `__device__`, δηλώνοντας ακριβώς το ίδιο πράγμα. Χρησιμοποιείται για μεταβλητές οι οποίες «στεγάζονται» στις διαμοιραζόμενες μνήμες των SMs. Οι μεταβλητές αυτές μπορούν να προσπελαστούν από όλα τα νήματα ενός μπλοκ, ενώ η διάρκεια ζωής τους είναι αυτή του συγκεκριμένου μπλοκ. Ο host δεν μπορεί να έχει καμία αλληλεπίδραση ή πρόσβαση σε αυτές τις μεταβλητές. Συνήθως ο χαρακτηριστής `__shared__` χρησιμοποιείται για πίνακες στους οποίους είναι επιθυμητό να έχουν πρόσβαση όλα τα νήματα του μπλοκ και το μέγεθος των πινάκων πρέπει να δηλώνεται επακριβώς. Εναλλακτικά, είναι εφικτό να έχουμε μέγεθος πίνακα που θα αποφασιστεί τη στιγμή που θα εκτελεστεί η εφαρμογή. Για να καταστεί κάτι τέτοιο δυνατό, πρέπει να δηλωθεί ο συγκεκριμένος πίνακας ως `extern` και το μέγεθός του θα περαστεί κατά την κλήση του υπολογιστικού πυρήνα από τον host.

Το Σχήμα 3-16 συνοψίζει τα παραπάνω:

	Μνήμη όπου φιλοξενείται:	**Ορατή στα νήματα του:**	**Διάρκεια ζωής:**
`__device__ __shared__ int SharedVar;`	διαμοιραζόμενη	μπλοκ	του μπλοκ
`__device__ int GlobalVar;`	καθολική	πλέγματος	της εφαρμογής
`__device__ __constant__ int ConstantVar;`	σταθερών	πλέγματος	της εφαρμογής

Σχήμα 3-16 Κατηγορίες Χαρακτηριστών Μεταβλητών.

Τέλος, αναφέρονται κάποιοι περιορισμοί και σημεία που χρήζουν ιδιαίτερης προσοχής. Πρώτον, δεν επιτρέπονται οι παραπάνω χαρακτηριστές για δομές όπως οι τύποι δεδομένων `struct` και οι `union`. Καμία μεταβλητή με οποιονδήποτε από τους παραπάνω χαρακτηριστές δεν μπορεί να δηλωθεί σαν «εξωτερική» (external) με χρήση της ειδικής λέξης `extern`, εκτός των `__shared__`. Ακόμα, οι `__shared__` μεταβλητές δεν μπορούν να αρχικοποιούνται κατά την δήλωσή τους. Τέλος, δεικτοδότηση μπορούμε να έχουμε μόνο σε μεταβλητές που έχουν δηλωθεί στη καθολική μνήμη.

Ένα άλλο σημαντικό στοιχείο που πρέπει να αναφερθεί είναι το τι γίνεται στην περίπτωση που σε μια μεταβλητή της συσκευής δεν υπάρχει κανένας από τους παραπάνω χαρακτηριστές. Σε αυτήν την περίπτωση οι συγκεκριμένες μεταβλητές θα αποθηκεύονται σε καταχωρητές και θα έχουν διάρκεια ζωής τη διάρκεια ζωής του νήματος στο οποίο ανήκουν. Σε περιπτώσεις μεγάλων πινάκων ή μεγάλου αριθμού μεταβλητών, αντί για τους καταχωρητές θα χρησιμοποιείται η τοπική μνήμη αντί του αρχείου καταχωρητών.

3.4.1.2 Ενσωματωμένοι Τύποι Δεδομένων

Το CUDA προσφέρει έναν αριθμό τύπων δεδομένων διαφόρων ειδών που είναι ουσιαστικά μικρά διανύσματα τεσσάρων διαστάσεων το πολύ. Παρακάτω, παρατίθεται η λίστα αυτών των τύπων και ακολουθεί η εξήγηση του τί αντιπροσωπεύει ο καθένας:

```
char1, uchar1, char2, uchar2, char3, uchar3, char4, uchar4, short1, ushort1, short2, ushort2, short3, ushort3, short4, ushort4, int1, uint1, int2, uint2, int3, uint3, int4, uint4, long1, ulong1, long2, ulong2, long3, ulong3, long4, ulong4, float1, float2, float3, float4
```

Ο κάθε ένας από τους παραπάνω τύπους αποτελεί ένα διάνυσμα με τόσες διαστάσεις όσες δηλώνει ο αριθμός στο τέλος του ονόματος του τύπου (1, 2, 3 ή 4), ενώ στην κάθε διάστασή του περιλαμβάνει δεδομένα του τύπου που δηλώνει το υπόλοιπο του ονόματός του (`char`, `short`, `int`, `long`, `float`). Τέλος, σε όσους από τους παραπάνω τύπους υπάρχει ένα u στην αρχή σημαίνει ότι στις διαστάσεις τα δεδομένα είναι χωρίς πρόσημο (`unsigned`). Έτσι, για παράδειγμα, ο τύπος `int3` αποτελεί ένα διάνυσμα τριών διαστάσεων, σε κάθε μια από τις οποίες περιλαμβάνει έναν ακέραιο αριθμό (integer- `int`).

Για να προσπελαστούν τα δεδομένα του καθενός από αυτους τους τύπους, θεωρούνται σαν δομές `struct` ,με πεδία `.x`, `.y`, `.z` και `.w` για την πρώτη, δεύτερη, τρίτη και τέταρτη διάσταση, αντίστοιχα. Έτσι, για παράδειγμα, αν η μεταβήτη `var` είναι τύπου `float4`, με αναφορά στην `var.y` προσπελάζεται το δεδομένο κινητής υποδιαστολής που θα υπάρχει στη δεύτερη διάσταση της. Κάθε ένας από τους παραπάνω τύπους δεδομένων συνοδεύεται από έναν κατασκευαστή (constructor), ο οποίος διευκολύνει την αρχικοποίησή του. Αυτός είναι της μορφής:

```
make_<όνομα τύπου> (αρχική_τιμή_διάστασης_x[, αρχική_τιμή_διάστασης_y], …)
```

Για παράδειγμα, η `var`, που αναφέρθηκε πριν, θα μπορούσε κάποιος να την αρχικοποιήσει ως εξής:

```
float4 make_float4 (float x, float y, float z, float w);
```

Έτσι, θα είχε αρχική τιμή την `(x, y, z, w)`.

Ιδιαίτερη μνεία αξίζει να γίνει και σε έναν ακόμα τύπο δεδομένων που προσφέρει το CUDA, ο οποίος χρησιμοποιείται σε κάθε εφαρμογή που αναπτύσσεται με χρήση του

CUDA. Πρόκειται για τον τύπο dim3. Ουσιαστικά αποτελεί μια μεταβλητή τύπου uint3, στην οποία κάθε διάσταση που δεν αρχικοποιείται παίρνει αυτόματα τιμή 1.

3.4.1.3 Ενσωματωμένες Μεταβλητές

Υπάρχουν τέσσερις μεταβλητές στις οποίες έχει πρόσβαση κάθε νήμα της εφαρμογής και αποτιμώνται κατά την εκκίνηση εκτέλεσης του εκάστοτε υπολογιστικού πυρήνα. Οποιαδήποτε συνάρτηση με τους χαρακτηριστές __global__ ή/και __device__ έχουν πρόσβαση στις παρακάτω μεταβλητές. Αυτές είναι οι εξής:

Α) gridDim : Αυτή η μεταβλητή είναι τύπου dim3 και περιλαμβάνει το μέγεθος των διαστάσεων του πλέγματος που εκτελείται τη δεδομένη στιγμή. Τα πλέγματα μπορούν να είναι μονοδιάστατα ή διασδιάστατα και άρα η τρίτη διάσταση (πεδίο .z) θα είναι πάντα ίση με 1, ενώ οι τιμές των δύο πρώτων εξαρτώνται από το τις επιθυμητές διστάσεις του πλέγματος, τις οποίες καθορίζει ο προγραμματιστής. Η gridDim έχει την ίδια τιμή για όλα τα νήματα της εφαρμογής.

Β) blockIdx : Η μεταβλητή είναι τύπου uint3 και περιλαμβάνει τις συντεταγμένες της θέσης του μπλοκ στο οποίο ανήκει το συγκεκριμένο νήμα εκτέλεσης. Η τρίτη διάσταση θα είναι πάντα 1, ενώ τα νήματα του ίδιου μπλοκ θα έχουν την ίδια τιμή blockIdx.

Γ) blockDim : Η μεταβλητή αυτή είναι τύπου dim3 και περιλαμβάνει τα μεγέθη των διαστάσεων του κάθε μπλοκ της εφαρμογής μας. Εφόσον όλα τα μπλοκ της εφαρμογής θα έχουν την ίδια διάσταση, αυτή η μεταβλητή θα έχει την ίδια τιμή σε όλα τα νήματα της εφαρμογής ανεξάρτητα σε ποιό μπλοκ θα ανήκουν.

Δ) threadIdx : Αυτή η μεταβλητή είναι τύπου uint3 και περιλαμβάνει τις συντεταγμένες της θέσης του συγκεκριμένου νήματος εκτέλεσης στο μπλοκ στο οποίο ανήκει. Όλα τα νήματα ενός μπλοκ έχουν διαφορετικές τιμές στη μεταβλητή threadIdx τους.

Απαγορεύεται οποιαδήποτε αλλαγή στην τιμή των παραπάνω μεταβλητών ενώ η διεύθυνσή τους στη μνήμη δεν μπορεί να παρθεί. Τονίζεται ότι η χρησιμότητα των παραπάνω μεταβλητών είναι πολύ μεγάλη, καθώς ουσιαστικά είναι αυτές που εκμεταλλεύεται κανείς, ώστε να διαφοροποιεί την ροή εκτέλεσης και τα δεδομένα προς επεξεργασίας κάθε νήματος.

3.4.1.4 Καθορισμός Παραμέτρων Εκτέλεσης

Κάθε κλήση υπολογιστικού πυρήνα που γίνεται από τον host πρέπει να συμπεριλαμβάνει και τις παραμέτρους εκτέλεσης του συγκεκριμένου πυρήνα. Αυτοί οι παράμετροι εκτέλεσης περιλαμβάνουν τη διάταξη και την οργάνωση του πλέγματος και των μπλοκ που το αποτελούν. Ουσιαστικά, μέσω αυτών των παραμέτρων γνωστοποιείται στη συσκευή το μέγεθος των διαστάσεων του πλέγματος, καθώς και το μέγεθος των διαστάσεων του κάθε μπλοκ. Για να γίνει κάτι τέτοιο, χρησιμοποιούνται συνήθως δύο μεταβλητές τύπου dim3. Επίσης, υπάρχει και άλλη μια προαιρετική παράμετρος, η οποία μπορεί να περαστεί στη συσκευή κατά την κλήση οποιουδηποτε πυρήνα. Αυτή είναι το ποσό διαμοιραζόμενης μνήμης που είναι επιθυμητό να δεσμευτεί για κάθε

μπλοκ, πέρα από το ποσό μνήμης που δεσμεύεται στατικά εντός του κώδικα της `__global__` συνάρτησης. Η παραπάνω παράμετρος περνιέται σαν επιπρόσθετη παράμετρος εντός των <<< >>>, τα οποία τοποθετούνται μεταξύ του ονόματος τη συνάρτησης και της λίστας των παραμέτρων της κατά την κλήση της συνάρτησης του υπολογιστικού πυρήνα. Ένα παράδειγμα είναι το εξής:

```
kernel_function<<< Gr, Bl, Sh>>>(…);
```

Στο παράδειγμα αυτό υποθέτουμε ότι:

α) `Gr`: Μια μεταβλητή τύπου dim3 που καθορίζει την οργάνωση του πλέγματος, αρχικοποιημένη στο επιθυμητό μέγεθος διαστάσεων σε μπλοκ. Θα μπορούσε ενδεικτικά πριν την κλήση της συνάρτησης να υπάρχει η παρακάτω γραμμή κώδικα:

```
dim3 Gr (100, 50);
```

Αυτό θα σήμαινε ότι ο πυρήνας θα εκτελεστεί ως ένα πλέγμα 100×50 μπλοκ.

β) `Bl`: Ομοίως με την `Gr` και αυτή η μεταβλητή είναι τύπου `dim3` και καθορίζει το μέγεθος και την διάταξη του κάθε μπλοκ μετρούμενα σε αριθμό νημάτων εκτέλεσης. Για παράδειγμα, θα μπορούσε πριν την κλήση, να υπάρχει η εξής γραμμή κώδικα:

```
dim3 Bl (4, 8, 8);
```

Αυτό θα σήμαινε ότι το κάθε μπλοκ του πυρήνα θα ήταν ένα $4 \times 8 \times 8$ ορθογώνιο παραλληλεπίπεδο αποτελούμενο από τα νήματα εκτέλεσης. Δηλαδή, το κάθε μπλοκ σε μια τέτοια σύνθεση θα αποτελούνταν από 256 νήματα.

γ) `Sh`: Αυτή είναι μια μεταβλητή ακεραίου, η οποία περιλαμβάνει το ποσό μνήμης σε bytes που θέλουμε να δεσμευτεί στη διαμοιραζόμενη μνήμη του κάθε μπλοκ, πέρα από το ποσό που έχει δεσμευτεί στατικά από τον κώδικα του υπολογιστικού πυρήνα. Αυτή η παράμετρος είναι προαιρετική με προεπιλεγμένη τιμή τα 0 bytes διαμοιραζόμενης μνήμης.

Όσον αφορά τις δύο πρώτες παραμέτρους, εκτός από το πέρασμά τους μέσω μεταβλητών τύπου `dim3`, μπορούν να χρησιμοιηθούν και αριθμοί uint απ' ευθείας στην κλήση της συνάρτησης όπως στο παρακάτω παράδειγμα:

```
kernel_function<<< 5000, 128 >>>(…);
```

Ο παραπάνω πυρήνας θα εκτελεστεί ως ένα πλέγμα 5000 μπλοκ με 128 νήματα το καθένα. Το πλέγμα και όλα τα μπλοκ θα είναι μονοδιάστατα σε αυτήν την περίπτωση.

Όλες αυτές οι παράμετροι εκτέλεσης αποτιμούνται πριν τις παραμέτρους της συνάρτησης του πυρήνα και περνάνε στη συσκευή μέσω της διαμοιραζόμενης μνήμης, όπως άλλωστε και οι πραγματικές παράμετροι της συνάρτησης. Σε περίπτωση που οι δύο πρώτες παράμετροι δεν είναι εντός των επιτρεπόμενων ορίων (κάποια διάσταση έχει μεγαλύτερο μέγεθος από το μέγιστο επιτρεπτό ή το σύνολο των μπλοκ ή των νημάτων/μπλοκ έχει ξεπεράσει το μέγιστο), ο πυρήνας αποτυγχάνει κατά την προσπάθεια εκτέλεσής του. Επίσης, αν το ποσό διαμοιραζόμενης μνήμης που είναι επιθυμητό

να δεσμευτεί, ξεπερνά το μέγεθος της διαμοιραζόμενης μνήμης, ο πυρήνας επίσης αποτυγχάνει.

Τέλος, θα πρέπει να αναφερθεί ότι η κλήση ενός υπολογιστικού πυρήνα γίνεται ασύγχρονα, δηλαδή η `__global__` συνάρτηση που βρίσκεται στην πλευρά του host επιστρέφει αμέσως και ο κώδικας του host συνεχίζει με την επόμενη εντολή. Υπάρχει, όμως, μια συνάρτηση του CUDA, η οποία μπορεί να μπλοκάρει την εκτέλεση του νήματος του host μέχρι να ολοκληρωθεί οποιαδήποτε λειτουργία του CUDA, επομένως, στην συγκεκριμένη περίπτωση μέχρι να ολοκληρωθούν οι λειτουργίες του εν λόγω υπολογιστικού πυρήνα.

3.4.2 Συχνά χρησιμοποιούμενες συναρτήσεις του CUDA

Σε αυτή την ενότητα θα γίνει μια σύντομη περιγραφή των συναρτήσεων που χρησιμοποιούνται στις περισσότερες των εφαρμογών που «χτίζονται» με χρήση του CUDA. Αυτές οι συναρτήσεις έχουν να κάνουν με το πώς ακριβώς μπορεί ο προγραμματιστής να χειριστεί όλα αυτα που του προσφέρει η αρχιτεκτονική G80, είτε πρόκειται για μεταφορές μνήμης, για μαθηματικούς υπολογισμούς ή για συγχρονισμό. Για περισσότερες λεπτομέρειες σχετικά με τις παρακάτω συναρτήσεις, ο αναγνώστης παραπέμπεται στο [23].

3.4.2.1 Συναρτήσεις καλούμενες από την πλευρά της συσκευής

Οι συναρτήσεις που θα παρουσιαστούν σε αυτήν την υποενότητα μπορούν να κληθούν μόνο από συναρτήσεις της συσκευής και όχι από τον host.

3.4.2.1.1 Μαθηματικές συναρτήσεις

Εκτός από τις περισσότερες μαθηματικές συναρτήσεις της C, οι οποίες υποστηρίζονται και μπορούν να εκτελεστούν και από την συσκευή με μικρές αποκλίσεις στα αποτελέσματα, η συσκευή υποστηρίζει και κάποιες γρηγορότερες αλλά λιγότερο ακριβείς συναρτήσεις όμοιες με αυτές της C. Αυτές φαίνονται στο Σχήμα 3-17 μαζί με τα σφάλματά τους και διακρίνονται από τις πρότυπες χάρη στα __ (δυο underscores) στην αρχή του ονόματός τους.

Συνάρτηση	Όρια σφαλμάτων
__fadd_rs(x,y)	Εναρμονισμένα με IEEE
__fmul_rs(x,y)	Εναρμονισμένα με IEEE
__fdividef(x,y)	Για y μεταξύ 2^{-126} έως 2^{126}, το μέγιστο ulp σφάλμα είναι 2
__expf(x)	Μέγιστο ulp σφάλμα είναι ίσο με 2 + floor(abs(1.16*x)).
__exp10f(x)	Μέγιστο ulp σφάλμα είναι ίσο με 2 + floor(abs(2.95*x)).
__logf(x)	Για x μεταξύ 0.5 και 2, το μέγιστο απόλυτο σφάλμα είναι $2^{-21.41}$, αλλιώς το μέγιστο ulp σφάλμα είναι 3.
__log2f(x)	Για x μεταξύ 0.5 και 2, το μέγιστο απόλυτο σφάλμα είναι 2^{-22}, αλλιώς το μέγιστο ulp σφάλμα είναι 2.
__log10f(x)	Για x μεταξύ 0.5 και 2, το μέγιστο απόλυτο σφάλμα είναι 2^{-24}, αλλιώς το μέγιστο ulp σφάλμα είναι 3
__sinf(x)	Για x μεταξύ -π και π,τ ο μέγιστο απόλυτο σφάλμα είναι $2^{-21.41}$, και μεγαλύτερο σε άλλες περιπτώσεις.
__cosf(x)	Για x μεταξύ -π και π,τ ο μέγιστο απόλυτο σφάλμα είναι $2^{-21.19}$, και μεγαλύτερο σε άλλες περιπτώσεις.
__sincosf(x,sptr,cptr)	Όπως στις περιπτώσεις των __sinf(x) και _cosf(x)
__tanf(x)	Προέρχεται από την υλοποίησή της ως __sinf(x)*(1/cosf(x)).
__powf(x,y)	Προέρχεται από την υλοποίησή της ως exp2f(y * __log2f(x))
__mul24(x,y), __umul24(x,y)	Μη διαθέσιμα
__lumhi(x,y_, __umulhi(x,y)	Μη διαθέσιμα
__int_as_float(x)	Μη διαθέσιμα
__float_as_int(x)	Μη διαθέσιμα
__saturate(x)	Μη διαθέσιμα
__sad(x,y,z), usad(x,y,z)	Μη διαθέσιμα
__cls(x)	Μη διαθέσιμα
__ffs(x)	Μη διαθέσιμα

Σχήμα 3-17 Σύνοψη ταχέων εκδόσεων μαθηματικών συναρτήσεων και αποκλίσεις από τα πρότυπα της IEEE.

3.4.2.1.2 Συνάρτηση συγχρονισμού

Έγινε νωρίτερα λόγος για τη δυνατότητα συγχρονισμού όλων των νημάτων που ανήκουν στο ίδιο μπλοκ. Αυτό υλοποιείται με κλήση της συνάρτησης __syncthreads() από τα νήματα της κάθε εφαρμογής. Όλα τα νήματα πρέπει να φτάσουν στο σημείο που καλείται αυτή η συνάρτηση για να μπορέσουν να συνεχίσουν με τις υπόλοιπες λειτουργίες που ακολουθούν την κλήσης αυτής. Κύρια χρησιμότητα αυτής της συνάρτησης είναι το γεγονός, ότι με τη χρήση της μπορούν να αποφευχθούν κίνδυνοι τύπου ανάγνωση-μετά-από-εγγραφή (read-after-write), εγγραφή-μετά-από-ανάγνωση

(write-after-read) ή εγγραφή-μετά-από-εγγραφή στην καθολική ή την διαμοιραζόμενη μνήμη. Μεταξύ όλων των προσβάσεων που εγκυμονούν κινδύνους δεδομένων σε μια από τις παραπάνω δυο μνήμες επιβάλλεται η προσθήκη της συνάρτησης συγχρονισμού.

Ένα σημείο που χρίζει ιδιαίτερης προσοχής είναι η χρήση της συνάρτησης συγχρονισμού μέσα σε κώδικα μετά από κάποια συνθήκη. Η χρήση της συνάρτησης συγχρονισμού θα πρέπει να γίνεται μόνο όταν είναι εγγυημένο ότι όλα τα νήματα του κάθε μπλοκ θα αποτιμούν τη συνθήκη στο ίδιο αποτέλεσμα και κατά συνέπεια είτε όλα τα νήματα είτε κανένα θα φτάνουν στο σημείο κλήσης της. Αν δε γίνει κάτι τέτοιο και μόνο μερικά νήματα την καλέσουν, ενώ κάποια άλλα συνεχίσουν προσπερνώντας την, η εφαρμογή θα «κολλήσει» γιατί τα νήματα που θα την καλέσουν θα περιμένουν επ᾽ αόριστον τα υπόλοιπα να φτάσουν εκεί, κάτι που δε θα γίνει ποτέ (deadlock).

3.4.2.1.3 Ατομικές συναρτήσεις

Οι ατομικές συναρτήσεις είναι διαθέσιμες μόνο σε συσκευές που υποστηρίζουν το CUDA 1.1 και δίνουν τη δυνατότητα στους προγραμματιστές μέσω των προγραμμάτων τους να πραγματοποιούν ατομικές λειτουργίες ανάγνωσης-τροποποίησης-εγγραφής σε κάποια θέση της καθολικής μνήμης, η οποία περιέχει ένα δεδομένο μεγέθους 32 bit. Αυτό σημαίνει ότι αν ένα νήμα πραγματοποιεί μια ατομική λειτουργία σε μιά θέση μνήμης, κανένα άλλο νήμα δεν μπορεί να προσπελάσει τη συγκεκριμένη θέση μέχρι να ολοκληρωθεί αυτή η λειτουργία και να γραφτεί το τελικό αποτέλεσμα της συγκεκριμένης λειτουργίας στην καθολική μνήμη. Οι ατομικές λειτουργίες πρέπει να σημειωθεί ότι είναι εφικτές μόνο για ακεραίους (προσημασμένους ή μη).

3.4.2.2 Συναρτήσεις καλούμενες από την πλευρά του host

Οι συναρτήσεις αυτής της υποενότητας μπορούν να κληθούν μόνο από τον host και δίνουν τη δυνατότητα ελέγχου και χειρισμού της συσκευής, καθώς και τη δυνατότητα πραγματοποίησης ποικίλων λειτουργιών.

3.4.2.2.1 Αρχικοποίηση

Δεν υπάρχει κάποια συγκεκριμένη συνάρτηση η οποία πρέπει να κληθεί ώστε να αρχικοποιηθεί με κάποιο τρόπο το προγραμματιστικό περιβάλλον του CUDA. Αντίθετα, κατά την πρώτη κλήση οποιασδήποτε συνάρτησης του CUDA, συμβαίνει αυτόματα και η αρχικοποίηση.

3.4.2.2.2 Διαχείριση Συσκευών

Σε ένα σύστημα είναι δυνατόν να υπάρχουν παραπάνω από μια κάρτες γραφικών ενεργοποιημένες, οι οποίες μπορεί να είναι επιθυμητό να χρησιμοποιηθούν συνεργατικά σε κάποια εφαρμογή. Όμως, χρειάζεται ένα ξεχωριστό νήμα εκτέλεσης του host για να εκτελεστεί κώδικας σε κάθε μια από τις διαθέσιμες συσκευές. Πρέπει, δηλαδή, να υπάρχει μια ένα-προς-ένα αντιστοιχία μεταξύ νημάτων του host και συσκευών που χρησιμοποιούνται. Το CUDA προσφέρει έναν αριθμό συναρτήσεων με τις οποίες μπορεί κανείς να διαχειριστεί τις διαθέσιμες στο σύστημά συσκευές. Οι σημαντικότερες από αυτές περιγράφονται στην συνέχεια:

α) `cudaGetDeviceCount()`: Αυτή η συνάρτηση επιστρέφει σε μια μεταβλητή του host τον αριθμό των ενεργοποιημένων συσκευών που υπάρχουν στο σύστημα και υποστηρίζουν το CUDA (CUDA-capable devices).

β) `cudaGetDeviceProperties()`: Αυτή η συνάρτηση είναι πιο εξεζητημένη σε σχέση με την προηγούμενη και εκτός από απλή απαρίθμηση των διαθέσιμων συσκευών, επιστρέφει τα χαρακτηριστικά τους (αριθμό SMs, μέγεθος καθολικής και διαμοιραζόμενης μνήμης, μέγεθος αρχείου καταχωρητών, μέγιστο μέγεθος και διαστάσεις πλέγματος και μπλοκ κ.ά.) σε μια δομή του CUDA που ονομάζεται `cudaDeviceProp`.

γ) `cudaSetDevice()` : Με αυτήν την συνάρτηση επιλέγεται σε ποια συσκευή θα τρέξει ο – ανεπτυγμένος με το CUDA – κώδικας που θα παράγει το συγκεκριμένο νήμα εκτέλεσης του host, το οποίο την καλεί. Αυτό γίνεται με το πέρασμα του αύξοντος αριθμού της επιθυμητής συσκευής σαν παράμετρο αυτής της συνάρτησης.

3.4.2.2.3 Διαχείριση Μνήμης

Το CUDA προσφέρει έναν αριθμό από συναρτήσεις για δέσμευση και απελευθέρωση μνήμης, καθώς και για μεταφορές μεταξύ host και συσκευών. Οι κυριότερες από αυτές είναι οι εξής:

α) `cudaMalloc()`: Αυτή η συνάρτηση χρησιμοποιείται για τη δέσμευση μνήμης στην καθολική μνήμη της συσκευής ,με την οποία αλληλεπιδρά το συγκεκριμένο νήμα του host. Η σημασιολογία της είναι αντίστοιχη της γνωστής `malloc` της πρότυπης βιβλιοθήκης της C.

β) `cudaMallocHost()`: Αυτή η συνάρτηση μπορεί να χρησιμοποιηθεί για δέσμευση μνήμης στον host αντί για την κλασική `malloc()` της C. Εάν συμβεί κάτι τέτοιο, όταν συμβεί μεταφορά δεδομένων που οι θέσεις στην κύρια μνήμη που κατέχουν έχουν δεσμευτεί με την `cudaMallocHost()`, σε θέσεις της καθολικής μνήμης της συσκευής που έχουν δεσμευτεί με την `cudaMalloc()`, τότε επιτυγχάνονται μεγαλύτερες ταχύτητες μεταφοράς σε σχέση με κάθε άλλον τρόπο. Περισσότερα για αυτό το ζήτημα θα αναφερθούν στο Κεφάλαιο 4.

γ) `cudaFree()`: Αυτή η συνάρτηση εκτελεί ακριβώς την αντίθετη λειτουργία σε σχέση με την `cudaMalloc()`, καθώς απελευθερώνει το ποσό μνήμης το οποίο είχε δεσμευτεί νωρίτερα στην καθολική μνήμη της συσκευής.

δ) `cudaFreeHost()`: Αυτή η συνάρτηση απελευθερώνει το ποσό της μνήμης που είχε δεσμευτεί προηγουμένως με αντίστοιχη κλήση της `cudaMallocHost()`.

ε) `cudaMemcpy()`: Αυτή είναι η βασική συνάρτηση που θα χρησιμοποιηθεί για μεταφορές δεδομένων/αποτελεσμάτων από host στη συσκευή και το αντίστροφο. Μέσα στις παραμέτρους αυτής της συνάρτησης καθορίζεται η φορά της μεταφοράς, η οποία μπορεί να είναι `cudaMemcpyHostToHost` (από μια μεταβλητή του host σε μια άλλη), `cudaMemcpyHostToDevice` (από μεταβλητή του host σε δεσμευμένο χώρο στην συσκευή), `cudaMemcpyDeviceToHost` (από μια μεταβλητή στην συσκευή σε μια του host) ή `cudaMemcpyDeviceToDevice` (από μεταβλητή σε μια

συσκευή σε μεταβλητή μιας άλλης συσκευής). Η συνάρτηση επιστρέφει όταν ολοκληρωθεί η μεταφορά, και δεν αρχίζει την μεταφορά μέχρι την ολοκλήρωση όλων των προηγούμενων κλήσεων συναρτήσεων του CUDA.

3.4.2.2.4 Διαχείριση Σφαλμάτων

Υπάρχουν δυο συναρτήσεις οι οποίες βοηθούν πολύ στο να αντιληφθεί κανείς αν κάτι δεν πάει καλά κατά την εκτέλεση μιας εφαρμογής βασισμένης στο CUDA καθώς και τι ακριβώς είναι αυτό που συνέβη όταν γίνει κάτι τέτοιο. Υπάρχει ο τύπος `cudaError_t` ο οποίος παίρνει σαν τιμή έναν κωδικό σφάλματος κάθε φορά. Αυτές οι συναρτήσεις είναι οι εξής :

α) `cudaGetLastError()`: Αυτή η συνάρτηση επιστρέφει τον κωδικό του σφάλματος της τελευταία κλήσης μιας συνάρτησης του CUDA αν υπήρξε τέτοιο, ενώ αν δεν συνέβη κάποιο σφάλμα, επιστρέφει τον κωδικό `cudaSuccess`.

β) `cudaGetErrorString()`: Αυτή η συνάρτηση παίρνει σαν είσοδο έναν κωδικό σφάλματος και επιστρέφει ένα μήνυμα σαν συμβολοσειρά, το οποίο περιγράφει το σφάλμα στο οποίο αντιστοιχεί ο προαναφερθείς κωδικός.

Παράδειγμα χρήσης των δυο παραπάνω συναρτήσεων σε συνδυασμό με σκοπό την εύρεση του τελευταίου σφάλματος που συνέβη είναι το εξής:

```
printf("%s\n", cudaGetErrorString(cudaGetLastError()) );
```

Η παραπάνω γραμμή κώδικα επιστρέφει τη συμβολοσειρά που αντιστοιχεί στο τελευταίο σφάλμα που έχει σχέση με το CUDA.

3.4.2.2.5 Διαχείριση νημάτων

Στην πλευρά του host υπάρχουν άλλες δυο πολύ σημαντικές συναρτήσεις οι οποίες έχουν να κάνουν με την διαχείριση του νήματος του host. Αυτές είναι οι εξής:

α) `cudaThreadSynchronize()`: Αυτή η συνάρτηση σταματά τη συνέχιση εκτέλεσης του νήματος του host μέχρι να ολοκληρωθούν όλες οι συναρτήσεις του CUDA, των οποίων η εκτέλεση είναι σε εξέλιξη. Η χρήση αυτής της συνάρτησης είναι πολύ χρήσιμη. Εφόσον όλοι οι υπολογιστικοί πυρήνες εκτελούντα ασύγχρονα, πρέπει να υπάρχει ένας τρόπος ελέγχου της ολοκλήρωσής τους. Αυτό χρειάζεται γιατί, όταν έρθει η ώρα να μεταφερθούν από την συσκευή στον host κάποια αποτελέσματα ή είναι επιθυμητό να επιτελεστεί μια άλλη λειτουργία, πρέπει να είναι σίγουρο ότι έχουν ολοκληρωθεί οι λειτουργίες του προαναφερθέντος πυρήνα, προκειμένου να μη αποκλίνει η εφαρμογή από την επιθυμητή συμπεριφορά.

β) `cudaThreadExit()`: Αυτή η συνάρτηση «καθαρίζει» όλους τους πόρους που έχει δεσμεύσει μέχρι τώρα το νήμα του host και έχουν σχέση με το περιβάλλον του CUDA. Οποιαδήποτε ενδεχόμενη επόμενη κλήση συνάρτησης του CUDA επαναρχικοποιεί το περιβάλλον ξεκινώντας από την αρχή σαν να μην είχε προηγηθεί τίποτε.

3.4.3 Πρόσθετες βιβλιοθήκες του CUDA

Μαζί με το CUDA η nVidia προσφέρει και δυο βιβλιοθήκες με ευρέως χρησιμοποιούμενες συναρτήσεις σε επιστημονικές εφαρμογές. Αυτές έχουν να κάνουν με υλοποίηση διαφόρων συναρτήσεων για πράξεις Γραμμικής Άλγεβρας και για την υλοποίηση του Ταχέως Μετασχηματισμός Φουριέ (Fast Fourier Transform).

3.4.3.1 CUBLAS

Το CUBLAS είναι η υλοποίηση των Βασικών Υποπρογραμμάτων Γραμμικής Άλγεβρας (Basic Linear Algebra Subprograms - BLAS) πάνω στον οδηγό του CUDA. Ουσιαστικά δίνει πρόσβαση στον προγραμματιστή στους υπολογιστικούς πόρους της κάρτας γραφικών με στόχο την επιτάχυνση πολλών λειτουργιών που έχουν να κάνουν με πράξεις Γραμμικής Άλγεβρας. Η βιβλιοθήκη αυτή είναι ενσωματωμένη στο Προγραμματιστικό Περιβάλλον Ανάπτυξης Εφαρμογών του CUDA και δεν χρειάζεται κάποιος να έρθει σε επαφή με τον οδηγό του CUDA.

Το CUBLAS εκτός των συναρτήσεων που περιλαμβάνει και έχουν να κάνουν με τις πράξεις Γραμμικής Άλγεβρας, περιλαμβάνει και έναν αριθμό συναρτήσεων που δημιουργούν ή καταστρέφουν αντικείμενα πινάκων ή διανυσμάτων, και αναλαμβάνουν τη μεταφορά δεδομένων από τη συσκευή προς τον host και το αντίστροφο.

Ο βασικός τρόπος χρήσης αυτής της βιβλιοθήκης συνοψίζεται ως εξής:

Αρχικά, ο προγραμματιστής δημιουργεί τα αντικείμενα πινάκων ή διανυσμάτων στην μνήμη της κάρτας γραφικών και τα γεμίζει με δεδομένα (αρχικές τιμές των αντικειμένων αυτών). Έπειτα, καλεί ένα αριθμό συναρτήσεων του CUBLAS ώστε να επεξεργαστεί τα αρχικά δεδομένα που εισήχθησαν στο προηγούμενο βήμα. Τέλος, μεταφέρει στον host τα αποτελέσματα ή τις τελικές δομές που προέκυψαν μετά την εκτέλεση των συναρτήσεων που είχαν κληθεί στο προηγούμενο βήμα.

Για να γίνει χρήση αυτής της βιβλιοθήκης αρκεί να συμπεριλάβουμε το αρχείο επικεφαλίδας cublas.h στα προγράμματα που θα συνταχθούν. Για περισσότερες πληροφορίες σχετικά με αυτήν την δυνατότητα μπορεί κάποιος να προμηθευτεί δωρεάν μέσω του site της nVidia το έγγραφο που περιγράφει αναλυτικά όλες τις συναρτήσεις που περιλαμβάνει αυτή η βιβλιοθήκη [23].

3.4.3.2 CUFFT

Ο Ταχύς Μετασχηματισμός Φουριέ (Fast Fourier Transform–FFT) αποτελεί έναν αλγόριθμο «διαίρει-και-βασίλευε» για αποδοτικό υπολογισμό των διακριτών μετασχηματισμών Φουριέ διαφόρων πραγματικών ή σύνθετων συνόλων δεδομένων. Αποτελεί έναν από τους πιο ευρέως χρησιμοποιούμενους αριθμητικούς αλγόριθμους σε επιστημονικές εφαρμογές στα πεδία κυρίως της Υπολογιστικής Φυσικής και της Επεξεργασίας Σήματος.

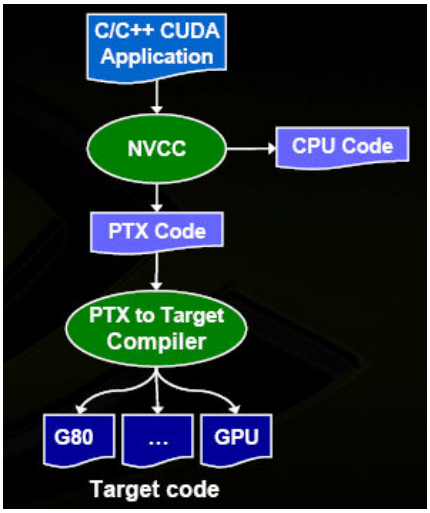
Η CUFFT είναι μια βιβλιοθήκη η οποία επιτρέπει στον προγραμματιστή να χρησιμοποιήσει μια σχετικά απλή διεπαφή ώστε να εκμεταλλευτεί τις πολύ υψηλές δυνατότητες που προσφέρουν οι κάρτες γραφικών στον τομέα της ταχύτητας εκτέλεσης πράξεων κίνηση υποδιαστολής. Υποστηρίζει μονοδιάστατους, δισδιάστατους και τρισδιάστατους μετασχηματισμούς Φουριέ σε διάφορα μεγέθη. Για περισσότερες πληρο-

φορίες όσον αφορά τις δυνατότητες αυτή της βιβλιοθήκης υπάρχει διαθέσιμο ένα σχετικό της nVidia στο site της στο Διαδίκτυο [23].

3.4.4 Μεταγλώττιση προγραμμάτων του CUDA

Σε αυτήν την ενότητα θα αναλυθεί το πώς μεταγλωττίζονται τα προγράμματα που συντάσσονται με χρήση του περιβάλλοντος του CUDA. Καταρχήν, πρέπει να αναφερθεί ότι όλα τα αρχεία κειμένου που περιλαμβάνουν κώδικα ο οποίος έχει να κάνει με το CUDA (είτε πρόκειται για κώδικα από την πλευρά του host είτε για κώδικα που έχει να κάνει με την συσκευή) πρέπει να έχουν την επέκταση .cu.

Ο μεταγλωττιστής που προσφέρει η nVidia μαζί με το CUDA ονομάζεται nvcc και είναι ένα περιτύλιγμα (wrapper) για μια αρκετά πιο σύνθετη διαδικασία μεταγλώττισης. Ουσιαστικά είναι ένας οδηγός-μεταγλωττιστής, ο οποίος επικαλείται έναν αριθμό εργαλείων σε κάθε στάδιο της μεταγλώττισης. Σαν είσοδο αυτή η διαδικασία μεταγλώττισης έχει κάποια αρχεία κώδικα με επεκτάσεις .c, .cpp, .cu, ανάλογα με το αν τα αρχεία κώδικα αυτά έχουν CUDA κώδικα ή όχι και η έξοδος είναι εκτελέσιμα ή αρχεία αντικειμενικού κώδικα (object files) στη μεριά του host και .cubin εκτελέσιμα αρχεία για την συσκευή. Στο Σχήμα 3-22 μπορούμε να διακρίνουμε τα βασικά στάδια της διαδικασίας μεταγλώττισης.

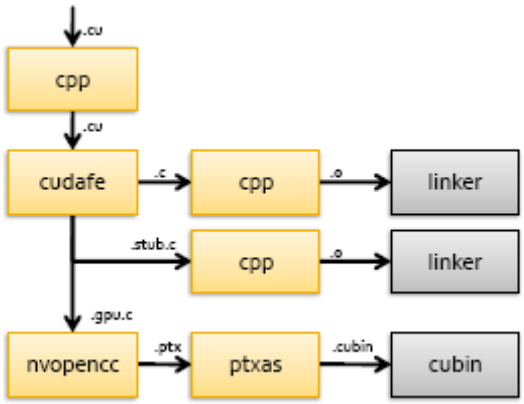


Σχήμα 3-18 Σύνοψη σταδίων μεταγλώττισης εφαρμογής ανεπτυγμένης με χρήση του CUDA.

Φαίνεται από το Σχήμα 3-18 ότι ο nvcc χωρίζει τον κώδικα που προορίζεται να εκτελεστεί στον host αποκλειστικά (αρχεία .c και .cpp) από τον κώδικα που έχει να κάνει με το CUDA (αρχεία .cu). Για τα πρώτα γίνεται χρήση του τοπικού μεταγλωττιστή της C του συστήματος, π.χ., του gcc, ενώ για τα υπόλοιπα η διαδικασία είναι πιο πολύπλοκη. Οι εφαρμογές, ανάλογα με την περίπτωση, μπορούν είτε να αγνοήσουν τον host κώδικα και είτε να φορτώσουν το .cubin εκτελέσιμο αρχείο στη συσκευή είτε να συνδεθεί (link) το αρχείο .cubin με τον κώδικα του host και έπειτα να φορτωθεί στη συσκευή. Όλα τα εκτελέσιμα αρχεία με κώδικα CUDA απαιτούν την ύπαρξη δυο δυναμικών βιβλιοθηκών για τη φάση της σύνδεσης (linking) : της cudart (CUDA runtime βιβλιοθήκη) και της cuda (κύρια βιβλιοθήκη του CUDA).

Όπως φαίνεται από το Σχήμα 3-18, πρώτα γεννάται κώδικας σε μία ενδιάμεση μορφή, που ονομάζεται PTX (Εκτέλεση Παράλληλων Νημάτων – Parallel Thread eXecution)

και μετά, ανάλογα με την ακριβή πλατφόρμα όπου θα εκτελεστεί ο κώδικάς, γίνεται ένα επιπλέον στάδιο μεταγλώττισης με στόχο την παραγωγή του τελικού εκτελέσιμου κώδικα της συσκευής. Στο Σχήμα 3-19 είναι φανερή αυτή η ροή μεταγλώττισης για κώδικα προοριζόμενο για τη συσκευή, με τη βοήθεια του cpp[3] και του ptxas. Ο πρώτος χρησιμοποιείται για την μεταγλώττιση των .c αρχείων με αποτέλεσμα την παραγωγή .o αντικειμένων, καθώς και του ptxas (PTX Assembler), ο οποίος συγκεντρώνει τα .ptx αρχεία και παράγει το .cubin εκτελέσιμο.



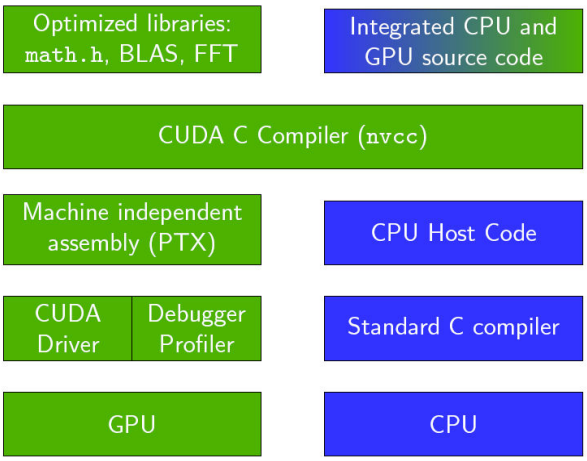
Σχήμα 3-19 Ροή μεταγλώττισης του κώδικα που προορίζεται για εκτέλεση στη συσκευή.

Τα .ptx αρχεία είναι μια συλλογή προγραμμάτων τα οποία περιλαμβάνουν κώδικα σε μια ψευδογλώσσα μηχανής με το δικό της σύνολο εντολών και τους δικούς της τελεστές. Περισσότερες λεπτομέρειες για αυτήν την γλώσσα μηχανής μπορούν να αναζητηθούν στο σχετικό έγγραφο της nVidia [23].

Να σημειωθεί ότι όσον αφορά το τελευταίο στάδιο της μεταγλώττισης, το σε ποια ακριβώς αρχιτεκτονική θα μεταγλωττιστεί τελικά ο κώδικάς γίνεται σαφές μέσω μίας οδηγίας (directive) .target μέσα στα .ptx αρχεία, ενώ η έκδοση του Συνόλου Αρχιτεκτονικής Εντολών (Instruction Set Architecture – ISA) του PTX που χρησιμοποιήθηκε για τη δημιουργία αυτού του .ptx αρχείου δηλώνεται με την οδηγία .version στην αρχή κάθε τέτοιου αρχείου.

Στο Σχήμα 3-20 παρουσιάζεται μια σύνοψη του Πακέτου Ανάπτυξης Λογισμικού του CUDA με έμφαση στους δρόμους μεταγλώττισης που συνοψίζει όσα αναφέρθηκαν παραπάνω.

[3] Εδώ ο cpp αναφέρεται σε μεταγλωττιστή που αναγνωρίζει την σύνταξη της C++ και όχι στον προεπεξεργαστή της C στο Linux, που επίσης συνήθως φέρει το όνομα cpp.



Σχήμα 3-20 Επίπεδα του Πακέτου Ανάπτυξης Λογισμικού του CUDA (CUDA SDK).

Αναλυτική περιγραφή του nvcc driver compiler μπορεί να αναζητηθεί στο σχετικό έγγραφο της nVidia [23].

3.4.5 Πρόσθετα εργαλεία που προσφέρει το CUDA

Το CUDA προσφέρει κάποια επιπλέον εργαλεία τα οποία έχουν να κάνουν με τη μέτρηση του ποσοστού χρησιμοποίησης του υλικού από τα εκάστοτε προγράμματα, με τη βοήθεια στον εντοπισμό σφαλμάτων, καθώς και την παρατήρηση διαφόρων στατιστικών κατά τη εκτέλεση των υπολογιστικών πυρήνων.

3.4.5.1 Εντοπισμός σφαλμάτων μέσω του Εξομοιωτή (Emulator)

Το προγραμματιστικό περιβάλλον του CUDA δεν προσφέρει κάποιον εντοπιστή σφαλμάτων (debugger), αλλά για αυτόν τον σκοπό υποστηρίζει έναν τρόπο προσομοίωσης της λειτουργίας της συσκευής. Αυτή η δυνατότητα είναι δυνατόν να χρησιμοποιηθεί με την επιλογή `--deviceemu` κατά τη μεταγλώττιση. Αν γίνει κάτι τέτοιο, ο κώδικάς μεταγλωττίζεται για να εκτελεστεί στον host και όχι στην συσκευή, σαν δηλαδή να πρόκειται για μια εφαρμογή του επεξεργαστή του συστήματος. Έτσι, είναι δυνατόν να ανιχνευτούν σφάλματα με τον ίδιο τρόπο που γίνεται όταν πρόκειται για εφαρμογές που εκτελούνται στη CPU. Γενικά, υπάρχουν δυο δυνατότητες κατά τη μεταγλώττιση: είτε θα χρησιμοποιηθεί ο nvcc για παραγωγή κώδικα που θα εκτελεστεί στη συσκευή είτε θα χρησιμοποιηθεί η προσομοίωση λειτουργίας της συσκευής με αποτέλεσμα την παραγωγή μεταγλωττισμένου κώδικα για εκτέλεση στη.

Όταν προσομοιώνεται η εκτέλεση μιας εφαρμογής, πρέπει να φροντίζεται ώστε:

a) Να έχει τη δυνατότητα ο host να υποστηρίζει αριθμό ενεργών νημάτων εκτέλεσης όσο το μέγεθος του μπλοκ συν ένα (το νήμα που θα αναλάβει τις λειτουργίες από την πλευρά του host).
b) Να υπάρχει αρκετή μνήμη στο μηχάνημα ώστε να μπορέσει να ολοκληρωθεί ομαλά η προσομοίωση.

Μέσω αυτού του τρόπου λειτουργίας μπορεί κανείς να προβεί σε ανίχνευση σφαλμάτων του αλγορίθμου με έναν από τους παρακάτω τρόπους:

a) Με χρήση των ανιχνευτών σφαλμάτων που είναι εγκατεστημένα στο σύστημα (π.χ., gdb, Visual Studio κτλ.) και με εκμετάλλευση των δυνατοτήτων που το περιβάλλον αυτών των εργαλείων προσφέρουν (π.χ., breakpoints).
b) Είναι εφικτό, πλέον, να χρησιμοποιηθούν στον κώδικα που προορίζεται για τρέξιμο στη συσκευή εντολές ή συναρτήσεις που δεν θα μπορούσαν, αν επρόκειτο να τρέξει πραγματικά στη συσκευή τα προγράμματά μας. Χαρακτηριστικό παράδειγμα η εντολή printf() η οποία μπορεί να βοηθήσει στον εντοπισμό διαφόρων σφαλμάτων
c) Εφόσον όλα τα δεδομένα και οι μεταβλητές βρίσκονται στην κύρια μνήμη του συστήματός, είναι δυνατό όλα τα νήματα εκτέλεσης να έχουν πρόσβαση σε οποιαδήποτε από αυτές είτε πρόκειται για το νήμα του host είτε για τα νήματα των μπλοκ. Εκτός αυτού, οποιαδήποτε συνάρτηση του host ή της συσκευής μπορεί να κληθεί είτε από τον host είτε από συσκευή, με αποτέλεσμα να είναι δυνατόν, πλέον, να αναμειχθούν κατά βούληση τα προγράμματα των δυο πλευρών.
d) Τέλος, ανιχνεύονται αυτόματα περιπτώσεις αδιεξόδων (deadlocks) που είναι πιθανόν να προκύψουν από λανθασμένη ή μη προσεκτική χρήση της συνάρτησης συγχρονισμού των νημάτων ενός μπλοκ.

Η χρήση αυτής της δυνατότητας πρέπει να γίνεται προσέχοντας ότι αυτός ο τρόπος εκτέλεσης των αλγορίθμων δεν αναπαριστά πιστά την πραγματική εκτέλεση των εφαρμογών στη συσκευή. Υπάρχει ένας αριθμός λαθών τα οποία δεν είναι δυνατόν να ανιχνευτούν με αυτό τον τρόπο, καθώς και κάποια σημεία που δεν μπορούν να οδηγήσουν στην άντληση συμπερασμάτων κατά την αποσφαλμάτωση:

a) Τα αποτελέσματα εκτέλεσης με τους δυο τρόπους μπορεί να διαφέρουν όταν τα νήματα των μπλοκ έχουν πρόσβαση στην ίδια διεύθυνση μνήμης, καθώς κατά την προσομοίωση, τα νήματα εκτελούνται ακολουθιακά το ένα μετά το άλλο σε αντίθεση με την πραγματική εκτέλεση στη συσκευή, όπου τα νήματα εκτελούνται ταυτόχρονα.
b) Όταν κάποια στιγμή γίνει αποαναφερθεί (dereference) κάποιος δείκτης που δείχνει στην καθολική μνήμη, η προσομοίωση μπορεί να δώσει σωστά αποτελέσματα, αλλά στην πραγματικότητα θα συνέβαινε κάποιο απροσδιόριστο σφάλμα και θα τερματιζόταν η εφαρμογή σε πραγματικές συνθήκες εκτέλεσης.
c) Τέλος, τα ακριβή αποτελέσματα πράξεων μεταξύ αριθμών κινητής υποδιαστολής μπορούν να διαφέρουν μεταξύ των εκτελέσεων στον host με προσομοίωση και στη συσκευή. Αυτό είναι γενικά αναμενόμενο, καθώς για τους υπολογισμούς μεταξύ αριθμών κινητής υποδιαστολής διαφέρουν οι αρχιτεκτονικές, τα σύνολα εντολών, οι μεταγλωττιστές. Επίσης, οι host, συνήθως, χρησιμοποιούν για ενδιάμεσα αποτελέσματα πράξεων κινητής υποδιαστολής καταχωρητές αυξημένης ακρίβειας, κάτι αδύνατον να γίνει λόγω μη διαθέσιμης υποδομής στις συσκευές.

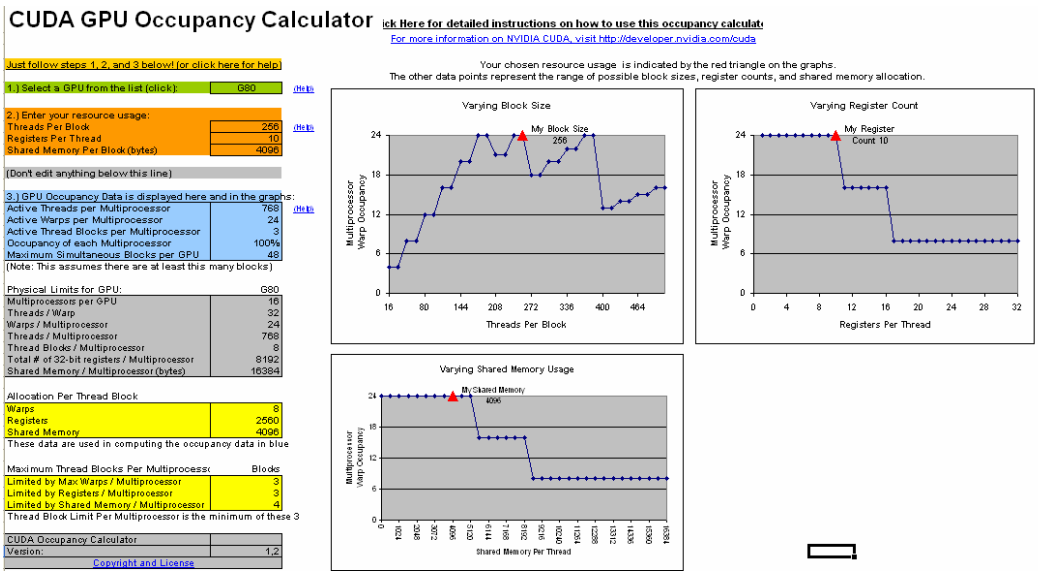
3.4.5.2 Υπολογιστής Χρησιμοποίησης – Occupancy Calculator

Το Πακέτο Ανάπτυξης Λογισμικού του CUDA προσφέρει ένα λογιστικό φύλλο το οποίο επιτρέπει δει κανείς το ποσοστό κατάληψης πόρων των πολυεπεξεργαστών SMs της συσκευής κατά την εκτέλεση ενός συγκεκριμένου πυρήνα (βλ. Σχήμα 3-21). Το ποσοστό αυτό, ουσιαστικά, είναι ο λόγος των ενεργών στημονιών ως προς τον αριθμό των στημονιών που υποστηρίζεται από ένα SM. Γενικώς, ο μεταγλωττιστής

ριθμό των στημονιών που υποστηρίζεται από ένα SM. Γενικώς, ο μεταγλωττιστής του CUDA προσπαθεί να ελαχιστοποιήσει τον αριθμό των καταχωρητών που θα χρησιμοποιούνται ώστε να μεγιστοποιήσει τον αριθμό των μπλοκ που θα είναι την ίδια στιγμή ενεργά σε ένα SM. Όσο μεγαλύτερο είναι το ποσοστό χρησιμοποίησης, τόσο θα μειώνεται ο χρόνος πρόσβασης στην καθολική μνήμη. Ο σκοπός αυτού του εργαλείου είναι να βοηθήσει να επιτευχθεί η καλύτερη επιλογή στην οργάνωση του πλέγματος που θα εκτελέσει τον επιθυμητό υπολογιστικό πυρήνα, και πιο συγκεκριμένα στην καλύτερη επιλογή του μεγέθους του μπλοκ βασισμένη στον αριθμό των απαιτούμενων καταχωρητών/νημάτων και του αιτούμενου ποσού διαμοιραζόμενης μνήμης.

Η χρήση του εργαλείου αυτού είναι πολύ απλή και συνοψίζεται στα εξής βήματα:

a) Καταρχήν, η εφαρμογή θα πρέπει να μεταγλωττιστεί χρησιμοποιώντας την επιλογή `--cubin` αντί για την `-c` στον nvcc ώστε σαν αποτέλεσμα να προκύψει ένα αρχείο με επέκταση .cubin. Αυτό θα περιέχει, εκτός από κώδικα σε δεκαεξαδική μορφή, ένα κομμάτι με το όνομα `code`. Αυτό το μέρος του αρχείου περιλαμβάνει κάποια στατιστικά στοιχεία για τον υπολογιστικό πυρήνα και, πιο συγκεκριμένα, το ποσό διαμοιραζόμενης μνήμης που χρειάζεται κάθε μπλοκ, τον αριθμό των καταχωρητών που απαιτεί κάθε νήμα και, τέλος, το ποσό της τοπικής μνήμης που χρειάζεται κάθε νήμα.
b) Έπειτα, αφού ανοίξει κανείς το λογιστικό φύλλο του Υπολογιστή Χρησιμοποίησης και αφού επιλέξει σε ποια οικογένεια καρτών γραφικών ανήκει η διαθέσιμη συσκευή, εισάγει τα δεδομένα που συλλέχθηκαν στο αμέσως προηγούμενο βήμα στα κατάλληλα πεδία.
c) Τέλος, στα διαγράμματα στα δεξιά του λογιστικού φύλλου τα οποία παρουσιάζουν το ποσοστό χρησιμοποίησης με τα εισαχθέντα δεδομένα, τον αριθμό των ενεργών νημάτων σε κάθε δεδομένη στιγμή, τον αριθμό των ενεργών στημονιών, των μπλοκ που αναλαμβάνει το κάθε SM και τον μέγιστο αριθμό των ενεργών μπλοκ που μπορεί να υποστηρίξει η συσκευή. Με ένα κόκκινο τρίγωνο υποδεικνύεται σε κάθε διάγραμμα η τρέχουσα χρησιμοποίηση πόρων της συσκευής, ενώ στα διαγράμματα είναι εμφανή τα ποσοστά χρησιμοποίησης για όλα τα πιθανά μεγέθη μπλοκ.



Σχήμα 3-21 Ο Υπολογιστής Χρησιμοποίησης.

3.4.5.3 Τομογράφος του CUDA – CUDA Visual Profiler

Το CUDA προσφέρει και έναν τομογράφο, με σκοπό την παρατήρηση διαφόρων στατιστικών, σχετικών με την εκτέλεση οποιουδήποτε πυρήνα στη διαθέσιμη συσκευή. Δηλαδή, είναι εφικτό μέσω αυτού του τομογράφου, να μετρηθεί ο ακριβής χρόνος που χρειάζεται για να εκτελεστεί μια εφαρμογή σε κάθε στάδιο της, είτε πρόκειται για μεταφορά δεδομένων μεταξύ host και συσκευής είτε για χρόνο εκτέλεσης στον host ή στη συσκευή.

Εάν εκτελεστεί ένα πρόγραμμα με τον τομογράφο ενεργοποιημένο συγκεντρώνονται σε έναν πίνακα τα στατιστικά στοιχεία της εκτέλεσης. Αυτά περιλαμβάνουν ότι έχει να κάνει με προσβάσεις των νημάτων στην καθολική μνήμη και σε ιδιωτικούς καταχωρητές. Μετρώνται, επίσης, ο αριθμός των εντολών που εκτελούνται, πόσες φορές έχουμε σειριοποίηση κατά την πρόσβαση των νημάτων του ίδιου στημονιού σε ίδιες τράπεζες της διαμοιραζόμενης μνήμης, ο αριθμός των μπλοκ που εκτελέστηκαν συνολικά καθώς και πόσες φορές συνέβησαν διακλαδώσεις στα νήματα της εφαρμογής μας. Για όλα τα παραπάνω ο τομογράφος παρουσιάζει έναν πίνακα με τα αποτελέσματα της εκτέλεσης οργανωμένα ανά συνάρτηση που εκτελείται στη συσκευή, ενώ είναι εφικτή η σύγκριση με παλαιότερα αποθηκευμένες εκτελέσεις και η εξαγωγή συμπερασμάτων για τις ενδεχόμενες διαφορές στους αλγόριθμους ή στις επιλεγμένες διατάξεις της κάθε περίπτωσης.

Πρέπει, όμως, να είναι κανείς ιδιαίτερα προσεκτικός κατά τη χρήση αυτού του εργαλείου όσον αφορά την εξαγωγή συμπερασμάτων, λόγω των παρακάτω παραγόντων:

a) Οι αριθμητικές τιμές που επιστρέφονται από τον τομογράφο αφορούν γεγονότα που συμβαίνουν σε κάθε στημόνι και όχι σε κάθε ξεχωριστό νήμα.
b) Ο τομογράφος στοχεύει και ανιχνεύει το τι συμβαίνει σε ένα μόνο SM και τα στατιστικά που συλλέγει αντιστοιχούν στα στημόνια που εκτελούνται σε αυτό και μόνο. Γι'αυτό, για να συλλεχθούν στατιστικά με κάποιο νόημα, πρέπει να εκτελείται μεγάλος σχετικά αριθμός μπλοκ, ώστε να είναι σίγουρο ότι το συγκεκριμένο SM θα αναλάβει ένα σημαντικό ποσοστό της συνολικής δουλειάς της συσκευής.
c) Τα αποτελέσματα που προσφέρει ο τομογράφος είναι ιδανικά για τη σύγκριση βελτιστοποιημένου κώδικα με μη βελτιστοποιημένο, προσπαθώντας κάθε φορά όλα τα στατιστικά που επιβαρύνουν χρονικά την εκτέλεση να μειώνονται (για παράδειγμα, οι σειριοποιήσεις πρόσβασης στη διαμοιραζόμενη μνήμη, οι διακλαδώσεις στα στημόνια κτλ.).

3.5 Ανακεφαλαίωση

Σε αυτό το κεφάλαιο παρουσιάστηκε αναλυτικά η αρχιτεκτονική G80, το Πακέτο Ανάπτυξης Λογισμικού CUDA που τη συνοδεύει και οι σημαντικότερες πτυχές της προσπάθειας ανάπτυξης γενικών εφαρμογών μέσω της χρήσης των παραπάνω. Μετά από την παρουσίαση αυτήν, γίνονται εμφανείς οι διαφορές με τους παλαιούς τρόπους χρήσης των GPUs για υπολογισμούς γενικού σκοπού. Η πλατφόρμα της nVidia αποτελεί κάτι εντελώς καινούργιο στο χώρο και οι δυνατότητες που παρέχει φαίνονται υποσχόμενες. Όπως έχει προαναφερθεί, η παρούσα διπλωματική εργασία έχει στόχο να κάνει μια προσπάθεια να εξερευνήσει το κατά πόσο ταιριάζει μια τέτοια πλατφόρμα στις ανάγκες σύγχρονων επιστημονικών και υπολογιστικών εφαρμογών. Για να

συμβεί κάτι τέτοιο, μετά την παρουσίαση του τρόπου αποδοτικής χρησιμοποίησης της αρχιτεκτονικής που θα γίνει στο Κεφάλαιο 4, στο Κεφάλαιο 5 θα επιχειρηθούν πειράματα, τα αποτελέσματα των οποίων θα χρησιμοποιηθούν για να εξαχθούν τα ανάλογα συμπεράσματα.

Κεφάλαιο 4

Τεχνικές βελτιστοποίησης εφαρμογών γενικού σκοπού στην αρχιτεκτονική G80

Σε αυτό το κεφάλαιο, θα αναλυθεί το πώς είναι δυνατόν να χρησιμοποιηθεί κάθε πτυχή της αρχιτεκτονικής G80, όπως αυτή παρουσιάστηκε ενδελεχώς στο προηγούμενο κεφάλαιο, με σκοπό τη βελτιστοποίηση των προγραμμάτων που προορίζονται για κάρτες γραφικών που στηρίζονται σε αυτήν. Θα γίνει σαφές ότι είναι εφικτή η εκμετάλλευση πολλών λεπτομερειών της αρχιτεκτονικής των καρτών γραφικών, κάποιων προφανών και άλλων λιγότερο, με πολύ σημαντικά αποτελέσματα ως προς την ταχύτητα διεκπεραίωσης των λειτουργιών των εφαρμογών.

Αφού γίνει αναφορά στα σημεία που είναι δυνατό να δώσουν κάποια επιτάχυνση σε οποιαδήποτε εφαρμογή, θα προετοιμαστεί το έδαφος ώστε στο επόμενο κεφάλαιο να εφαρμοστούν παρόμοιες τεχνικές σε πραγματικές συνθήκες, δηλαδή σε κομμάτια πραγματικών επιστημονικών εφαρμογών.

Είναι πολύ σημαντικό να ακολουθούνται από τους επίδοξους προγραμματιστές πολλές από τις στρατηγικές που θα αναλυθούν σε κάθε εφαρμογή στη συγκεκριμένη πλατφόρμα, προκειμένου τα αποτελέσματα ως προς την επίδοση και την ταχύτητα εκτέλεσης να είναι κοντά στις μέγιστες δυνατότητες των καρτών γραφικών. Όπως θα αποδειχθεί και στη συνέχεια, πολλές από τις τεχνικές έχουν τεράστια επίπτωση στο συνολικό χρόνο εκτέλεσης

4.1 Στρατηγικές Βελτιστοποίησης Επίδοσης

Σε αυτή την ενότητα θα παρουσιαστούν οι περισσότερες τεχνικές που μπορούν να εφαρμοστούν σε σχεδόν κάθε εφαρμογή ανεπτυγμένη με χρήση του CUDA, ώστε να επιτευχθούν μεγάλες ταχύτητες και να εκμεταλλευθούν όσο το δυνατόν περισσότερο οι μεγάλες υπολογιστικές δυνατότητες των τσιπ των καρτών γραφικών που στηρίζονται στην αρχιτεκτονική G80.

4.1.1 Κατάλληλη προσαρμογή αλγορίθμου

Το πρώτο βήμα που πρέπει να γίνεται πάντα είναι η όσο το δυνατόν πιο σωστή απεικόνιση του αλγορίθμου στην πλατφόρμα που είναι διαθέσιμη. Πρέπει, δηλαδή, να γίνει διερεύνηση για το αν μπορεί να παραλληλοποιηθεί αρκετά, ώστε να εκμεταλλευτεί τον μεγάλο αριθμό επεξεργαστικών πυρήνων, και αν περιλαμβάνει μεγάλο αριθμό πράξεων και υπολογισμών σε σχέση με λειτουργίες μνήμης – οι οποίες κοστίζουν περισσότερο σε χρόνο – ώστε να γίνεται χρήση όλων των διαθέσιμων πόρων της κάρτας γραφικών. Αφού γίνει η καλύτερη δυνατή επιλογή, πρέπει να γίνει προσπάθεια αναδιάταξης των εντολών, ώστε να επιτευχθεί μεγάλος λόγος υπολογισμών προς προσβάσεις μνήμης. Πολλές φορές είναι προτιμότερο κάτι να υπολογιστεί εκ νέου παρά να επανακομιστεί από τη μνήμη του host ή την καθολική μνήμη του τσιπ γραφικών, καθώς το δυνατό σημείο της αρχιτεκτονικής G80 είναι οι μεγάλες δυνατότητες σε λειτουργίες υπολογισμών.

4.1.2 Βέλτιστη επιλογή παραμέτρων εκτέλεσης

Πολύ σημαντικό σημείο είναι η καλύτερη δυνατή επιλογή των παραμέτρων εκτέλεσης του κάθε υπολογιστικού πυρήνα προς εκτέλεση. Δηλαδή, πρέπει να προσεχθεί πόσα μπλοκ θα περιλαμβάνει το πλέγμα, καθώς και ποιό θα είναι το μέγεθος αυτών των μπλοκ, ώστε να γίνεται η καλύτερη δυνατή χρήση του υλικού της συσκευής.

Αναφέρθηκε ότι από τους πιο σημαντικούς παράγοντες καθυστέρησης είναι οι προσβάσεις στην καθολική μνήμη. Όμως, αυτό το εμπόδιο δεν είναι αξεπέραστο, καθώς, αν εγγυηθεί κανείς ότι θα υπάρχουν πάντα νήματα προς εκτέλεση, η καθυστέρηση πρόσβασης μπορεί να «κρυφτεί». Αυτό θα συμβεί γιατί όσο έρχονται τα δεδομένα από την μνήμη, οι πυρήνες μπορούν να είναι απασχολημένοι κάνοντας άλλους υπολογισμούς κάποιων άλλων νημάτων, χωρίς να σπαταλάται χρόνος αναμονής για τα ζητούμενα δεδομένα. Από αυτά συμπεραίνει κανείς πως ο αριθμός των μπλοκ πρέπει να είναι σημαντικά μεγάλος, αν υπάρχουν συχνές αναφορές στη μνήμη, ώστε η καθυστέρηση που θα εισήγαγαν αυτές να κρύβεται κατά το δυνατόν.

Όσον αφορά την επιλογή των παραμέτρων εκτέλεσης, ο επίδοξος προγραμματιστής θα πρέπει να έχει υπ’ όψη του τα παρακάτω:

- Ο αριθμός των μπλοκ θα πρέπει να είναι τουλάχιστον μεγαλύτερος ή ίσος με τον αριθμό των SMs της κάρτας γραφικών, στην οποία πρόκειται να τρέξει η εφαρμογή, ώστε κάθε SM να έχει διαθέσιμες λειτουργίες προς εκτέλεση ανά πάσα στιγμή και να μην μένουν αχρησιμοποίητοι πόροι του διαθέσιμου υλικού.
- Επίσης, είναι καλό ο αριθμός των μπλοκ να είναι μεγαλύτερος από το διπλάσιο του αριθμού των SMs ώστε να καταστεί δυνατό το «κρύψιμο» των καθυστερήσεων πρόσβασης στην καθολική μνήμη.

- Ο αριθμός των μπλοκ θα πρέπει να είναι όσο το δυνατόν μεγαλύτερος για λόγους κλιμάκωσης της επίδοσής και σε μελλοντικά μηχανήματα επεξεργασίας γραφικών, που θα υποστηρίζουν το CUDA. Εκείνα πιθανότατα θα έχουν πολλαπλάσιο αριθμό SMs σε σχέση με την σειρά 8 της nVidia. Υπολογιστικοί πυρήνες που τρέχουν σε εκατοντάδες ή χιλιάδες μπλοκ θα κλιμακώνουν για πολλές επόμενες γενιές καρτών γραφικών, χωρίς να χρειαστούν αλλαγές στον κώδικά τους.
- Οι πόροι που θα χρειάζεται κάθε μπλοκ θα πρέπει να είναι λιγότεροι από τους μέγιστους πόρους που διαθέτει κάθε SM, ώστε αυτό να μπορεί να φιλοξενήσει παραπάνω από ένα μπλοκ κάθε στιγμή.
- Το μέγεθος του μπλοκ θα πρέπει να είναι πολλαπλάσιο του μεγέθους του στημονιού ώστε να μην υπάρχουν μισογεμισμένα στημόνια τα οποία θα σπαταλούν πόρους κατά τους υπολογισμούς τους.
- Ο διαμοιρασμός των εργασιών του πυρήνα, θα πρέπει να γίνεται ισομερώς σε όλα τα μπλοκ, και η δουλειά που αναλαμβάνει το κάθε μπλοκ να είναι η ίδια για λόγους συμμετρικότητας, δηλαδή όλα τα SMs να ξεκινούν ταυτόχρονα τις εργασίες τους και να δουλεύουν παράλληλα μέχρι και την ολοκλήρωση των ενεργειών τους την ίδια, περίπου, στιγμή.

Όλες οι παραπάνω ενέργειες έχουν επίπτωση στο ποσοστό χρησιμοποίησης του υλικού. Σκοπός θα πρέπει να είναι το όσο το δυνατόν μεγαλύτερο ποσοστό χρησιμοποίησης της συσκευής. Βέβαια, δε σημαίνει πάντα ότι όσο μεγαλύτερο ποσοστό χρησιμοποίησης έχει επιτευχθεί, τόσο καλύτερη επίδοση θα επιτυγχάνεται, αλλά είναι σίγουρο ότι οι υπολογιστικοί πυρήνες που εκτελούνται με παραμέτρους που δεν εξασφαλίζουν μεγάλο ποσοστό χρησιμοποίησης, δεν θα μπορούν να «κρύψουν» ικανοποιητικά καθυστερήσεις πρόσβασης στη μνήμη.

Ολοκληρώνοντας, πρέπει να αναφερθεί ότι το έργο της επιλογής των καλύτερων δυνατών παραμέτρων εκτέλεσης είναι αρκετά δύσκολο. Αν και οι παραπάνω αρχές ισχύουν σχεδόν πάντα, πολλές φορές πρέπει κάποιος να πειραματιστεί χρησιμοποιώντας όλους τους δυνατούς συνδυασμούς για να βρει τη βέλτιστη οργάνωση του πλέγματος σε μπλοκ και των μπλοκ σε νήματα. Αυτό συμβαίνει γιατί κάθε εφαρμογή έχει τις δικές της ιδιαιτερότητες, και δεν υπάρχει μια συνταγή επιτυχίας για όλες τις περιπτώσεις [58].

4.1.3 Προσεκτική επιλογή μείγματος εντολών

Στο προηγούμενο κεφάλαιο, και πιο συγκεκριμένα στην ενότητα 3.2.4, έγινε μια σύντομη αναφορά στο ρεπερτόριο εντολών που μπορεί να εκτελέσει κάθε SP, είτε έχει να κάνει με αριθμητικές/υπολογιστικές εντολές είτε με εγγραφές/αναγνώσεις σε/από κάποια από τις μνήμες. Κάθε εντολή έχει διαφορετικό κόστος σε χρόνο, και στην αναζήτησή για την καλύτερη επίδοση θα πρέπει να είναι κανείς προσεκτικός κατά την επιλογή των εντολών που θα χρησιμοποιήσει για τους υπολογισμούς, επιλέγοντας εντολές που κοστίζουν το δυνατόν λιγότερο, όταν βεβαίως η εφαρμογή επιτρέπει εναλλακτικές λύσεις. Στόχος είναι η μεγάλη διεκπεραιωτική ικανότητα εντολών (instruction throughput), δηλαδή στη μονάδα του χρόνου να εκτελούνται όσο το δυνατόν περισσότερες εντολές ώστε να οδεύει όσο το δυνατόν γρηγορότερα ο κώδικάς προς την ολοκλήρωσή του.

Οι εντολές των νημάτων εκτελούνται στο «πλαίσιο» ενός στημονιού. Ουσιαστικά, τα 32 νήματα του ίδιου στημονιού, που βεβαίως εκτελείται σε ένα SM, εκτελούν την ίδια εντολή την ίδια χρονική στιγμή. Επειδή υπάρχουν οκτώ πυρήνες σε ένα SM και πρέπει κάθε εντολή του στημονιού να εκτελεστεί 32 φορές (μια για κάθε νήμα), το λιγότερο που μπορεί να διαρκέσει μια εντολή είναι 4 κύκλοι μηχανής (32 εντολές / 8 πυρήνες = 4 διαδοχικοί κύκλοι). Στις επόμενες υποενότητες θα αναφερθούν τα κόστη κάποιων σημαντικών οικογενειών εντολών.

4.1.3.1 Αριθμητικές εντολές

Αναφέρεται ότι οι εντολές στοιχειωδών πράξεων μεταξύ αριθμών κινητής υποδιαστολής (πρόσθεση, αφαίρεση, πολλαπλασιασμός, πολλαπλασιασμός-πρόσθεση), η πρόσθεση ακεραίων, οι λογικές πράξεις μεταξύ bits, οι μετατροπές μεταξύ διαφορετικών τύπων, η πράξεις μεγίστου/ελαχίστου και οι συγκρίσεις, καθώς επίσης και ο ειδικός πολλαπλασιασμός μεταξύ ακεραίων 24 bit διαρκούν τέσσερις κύκλους ρολογιού. Ο πολλαπλασιασμός ακεραίων, η τετραγωνική ρίζα ακεραίων, το τετράγωνο ακεραίων, η `__log()` – η ταχεία έκδοση του λογαρίθμου που προσφέρει το CUDA – απαιτούν 16 κύκλους. Εδώ θα πρέπει να αναφερθεί ότι θα πρέπει να αποφεύγονται οι πράξεις διαίρεσης και εύρεσης υπολοίπου μεταξύ ακεραίων (div και mod αντίστοιχα) γιατί είναι ιδιαίτερα χρονοβόρες σε κύκλους ρολογιού. Αντί αυτών και εφόσον οι διαιρέτες είναι δυνάμεις του 2, μπορούν να χρησιμοποιηθούν οι παρακάτω εναλλακτικές λύσεις (όπου i ο διαιρετέος, n ο διαιρέτης ο οποίος είναι δύναμη του 2, >> το σύμβολο ολίσθησης προς τα δεξιά και & το σύμβολο της λογικής πράξης AND) :

$$i >> \log_2 n \text{ αντί για } \frac{i}{n}$$

$$i \,\&\, (n-1) \text{ αντί για i\%n}$$

Τέλος, η τετραγωνική ρίζα αριθμών κινητής υποδιαστολής διαρκεί 32 κύκλους, όπως και οι ταχείες συναρτήσεις `__sin()`, `__cos()`, `__exp()`, ενώ η διαίρεση μεταξύ αριθμών κινητής υποδιαστολής διαρκεί 36 κύκλους.

4.1.3.2 Εντολές πρόσβασης στη μνήμη

Οι εντολές πρόσβασης σε οποιαδήποτε μνήμη διαρκούν τουλάχιστον τέσσερις κύκλους, και ανάλογα με την μνήμη στην οποία γίνεται αναφορά, προστίθενται και κάποιοι κύκλοι αναμονής μέχρι να μεταφερθούν από/προς αυτήν τα δεδομένα. Στην περίπτωση της καθολικής μνήμης, η διάρκεια της εντολής μπορεί να φτάσει τους 400 με 600 κύκλους ρολογιού. Όπως αναφέρθηκε και σε προηγούμενες παραγράφους, ένα μεγάλο μέρος αυτής της καθυστέρησης μπορεί να «κρυφτεί» από τον χρονοδρομολογητή του SM, αν υπάρχουν αρκετά ενεργά στημόνια. Συνεπώς, θα πρέπει να περιορίζονται οι προσβάσεις στην καθολική μνήμη στις απολύτως απαραίτητες. Για παράδειγμα, στην αρχή του υπολογιστικού πυρήνα μπορούν να μεταφερθούν σε καταχωρητές ή στην διαμοιραζόμενη μνήμη τα δεδομένα εισόδου και στο τέλος να γραφτούν σε αυτήν τα τελικά αποτελέσματά, αντί να συμβαίνουν συνεχώς προσβάσεις στην καθολική μνήμη.

4.1.3.3 Συγχρονισμός νημάτων

Η εντολή συγχρονισμού μεταξύ των νημάτων του ίδιου μπλοκ διαρκεί τέσσερις κύκλους συν τον χρόνο που κάνουν όλα τα νήματα του ίδιου μπλοκ μέχρι να φτάσουν

στο σημείο να την εκτελέσουν. Αυτή η εντολή εισάγει σημαντική καθυστέρηση και πρέπει να περιορίζεται στις απολύτως απαραίτητες κλήσεις. Αυτές είναι αφού μεταφερθούν δεδομένα από την καθολική μνήμη στη διαμοιραζόμενη και αμέσως μετά την παραπάνω μεταφορά όταν επιχειρείται πρόσβαση από κάποια νήματα σε αυτά τα δεδομένα. Αυτό είναι απαραίτητο γιατί τα νήματα πρέπει να συγχρονιστούν ώστε να είναι σίγουρο ότι θα επεξεργαστούν τα νέα δεδομένα της διαμοιραζόμενης μνήμης και ότι δε θα προλάβει κάποιο από αυτά να αναγνώσει πριν ολοκληρωθούν οι εγγραφές των νέων τιμών.

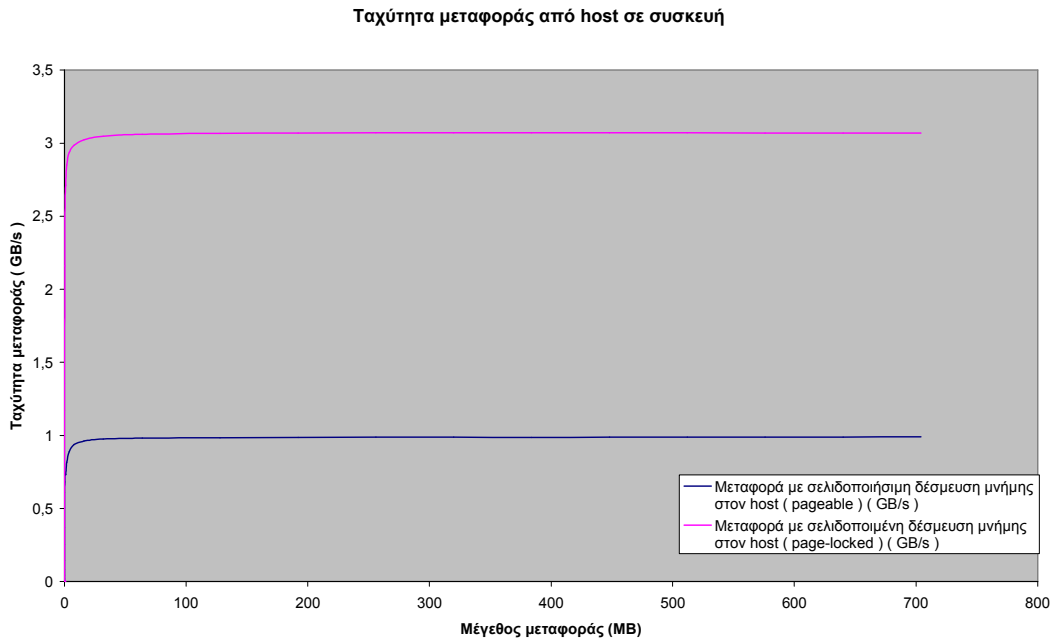
4.1.4 Ελαχιστοποίηση μεταφορών δεδομένων host <-> device

Το εύρος ζώνης μεταξύ της κύριας μνήμης του host και της καθολικής μνήμης της συσκευής είναι πολύ μικρότερο από το εύρος ζώνης μέσω του οποίου επικοινωνούν οι διάφορες μνήμες της συσκευής μεταξύ τους. Αποτέλεσμα είναι οι σχετικά μικρές ταχύτητες μεταφοράς δεδομένων που έχουμε όταν η κάρτα γραφικών επικοινωνεί με την κύρια μνήμη. Έτσι, πρέπει ο κώδικάς να μην περιλαμβάνει πολλές τέτοιου είδους μεταφορές για να αποφεύγονται όσο το δυνατόν περισσότερες καθυστερήσεις. Σκόπιμη είναι η στρατηγική μιας αρχικής μεταφοράς όλων των δεδομένων που θα χρειαστούν από τον υπολογιστικό πυρήνα και μιας τελική μεταφοράς των τελικών αποτελεσμάτων από την συσκευή στον host, ώστε να μην παρουσιαστεί η ανάγκη να γίνει νέα μεταφορά δεδομένων από τον host. Επίσης, πρέπει να αναφερθεί ότι η κάθε μεταφορά κατά την προετοιμασία και τη διεκπεραίωσή της, εισάγει μια επιπλέον επιβάρυνση (overhead), οπότε είναι προς το συμφέρον του προγραμματιστή να επιχειρεί μια μεγάλη μεταφορά αντί για πολλές μικρές, η οποία να περιλαμβάνει όλα τα δεδομένα προς μετακίνηση.

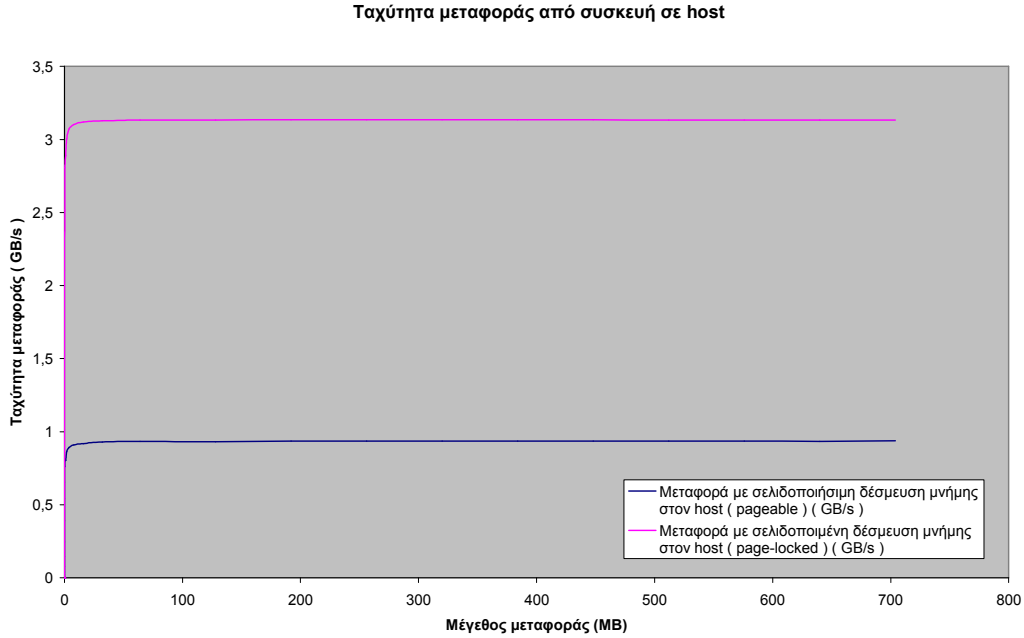
Αναφέρθηκε στην ενότητα 3.4.2.2.3 ότι υπάρχουν δυο διαφορετικοί τρόποι δέσμευσης μνήμης στον host, μέσω δυο διαφορετικών εντολών: της κλασικής `malloc()` της C και της `cudaMallocHost()`. Στην πρώτη περίπτωση, έχουμε σελιδοποιήσιμο (pageable) χώρο μνήμης, ενώ στην άλλη περίπτωση έχουμε σελιδοποιημένο[4] (page-locked) χώρο μνήμης. Με το παρακάτω πείραμα, θα συγκριθούν οι δυο τρόποι.

Επιχειρήθηκε να μεταφερθούν και με τους δυο τρόπους, διάφορες ποσότητες δεδομένων. Σε κάθε απόπειρα, μετρήθηκαν οι ταχύτητες που επετεύχθησαν σε κάθε περίπτωση. Τα αποτελέσματά φαίνονται στα Σχήματα 4-1 και 4-2:

[4] Εναλλακτικά, μπορεί να ονομαστεί μη σελιδοποιημένος αυτός ο χώρος μνήμης.



Σχήμα 4.1 Ταχύτητες μεταφοράς από τον host στην συσκευή για διάφορα μεγέθη μεταφοράς με δυο διαφορετικούς τρόπους (σελιδοποιήσιμη ή σελιδοποιημένη δέσμευση μνήμης στον host) .



Σχήμα 4.2 Ταχύτητες μεταφοράς από την συσκευή στον host για διάφορα μεγέθη μεταφοράς με δυο διαφορετικούς τρόπους (σελιδοποιήσιμη ή σελιδοποιημένη δέσμευση μνήμης στον host) .

Παρατηρείται ότι ο τρόπος μεταφοράς με την σελιδοποιημένη δέσμευση μνήμης είναι πάνω από τρεις φορές πιο γρήγορη σε σχέση με την σελιδοποιήσιμη, πράγμα που σημαίνει ότι θα πρέπει να προτιμάται προκειμένου να μειωθεί ο χρόνος που αντιστοιχεί στις μεταφορές. Αυτή η διαφορά στην ταχύτητα υπάρχει επειδή κατά τη μεταφορά κάποιου κομματιού σελιδοποιήσιμης μνήμης από τον host στη συσκευή, γίνεται αρχικά μια μεταφορά σε σελιδοποιημένη μνήμη (πράγμα που διαρκεί σημαντικό κομμάτι χρόνου της συνολικής μεταφοράς) και στην συνέχεια με Απ'ευθείας Πρόσβαση

Μνήμης χωρίς την εμπλοκή της CPU (Direct Memory Access – DMA) μεταφέρονται τα δεδομένα στη συσκευή. Αντίθετα, στην περίπτωση της σελιδοποιημένης μνήμης δεν χρειάζεται να γίνει το πρώτο στάδιο και έτσι απλά συμβαίνει μια μεταφορά με DMA από τον host στη συσκευή. Κάτι αντίστοιχο συμβαίνει και στην αντίστροφη πορεία μεταφοράς δεδομένων.

Επίσης αναφέρεται ότι στην περίπτωση της σελιδοποιημένης μνήμης, οι μεταφορές από τον host στη συσκευή είναι ελαφρώς πιο αργές από τις μεταφορές της αντίστροφης κατεύθυνσης, ενώ για την περίπτωση της σελιδοποιήσιμης μνήμης οι μεταφορές από τον host στη συσκευή είναι λίγο πιο γρήγορες σε σχέση με τις μεταφορές της αντίστροφης πορείας.

Τέλος θα πρέπει να σημειωθεί ότι δεν ενδείκνυνται μεταφορές από σελιδοποιημένους χώρους μνήμης όταν το σύστημα είναι πολύ, γιατί υπάρχει κίνδυνος να επιβαρυνθεί σημαντικά η επίδοση του.

4.1.5 Σωστή αξιοποίηση της Ιεραρχίας Μνήμης της συσκευής

Το εύρος ζώνης του κάθε είδους μνήμης που υπάρχει στην ιεραρχία μνήμης της συσκευής είναι διαφορετικό και χρειάζεται ιδιαίτερο χειρισμό. Γενικά, όμως, είναι εφικτό να ακολουθείται ένας ενιαίος τρόπος διάρθρωσης των εφαρμογών, τουλάχιστον ως προς τις προσβάσεις των πυρήνων της συσκευής στα δεδομένα εισόδου. Έτσι, ακολουθώντας τα παρακάτω πέντε βήματα, καθώς και τα συμπεράσματα στα οποία θα οδηγηθούμε αφού μελετήσουμε τον τρόπο πρόσβασης σε κάθε επίπεδο της ιεραρχίας μνήμης, καταλήγει κανείς σε έναν οδηγό στην προσπάθειά για την αποδοτική υλοποίηση οποιουδήποτε αλγορίθμου στην μελετώμενη πλατφόρμα:

- Κάθε μπλοκ φορτώνει τα δεδομένα εισόδου που θα χρειαστεί από την καθολική μνήμη στη διαμοιραζόμενη, μοιράζοντας τις φορτώσεις σε όλα τα νήματά του.
- Επιτυγχάνεται συγχρονισμός με τη βοήθεια της συνάρτησης συγχρονισμού, ώστε όταν κάποιο νήμα επιχειρήσει να αναγνώσει κάτι από τη διαμοιραζόμενη μνήμη, να αναγνώσει τα αναβαθμισμένα (updated) δεδομένα.
- Γίνεται επεξεργασία των δεδομένων που υπάρχουν στην διαμοιραζόμενη μνήμη από όλα τα νήματα του μπλοκ και παράγονται τα τελικά αποτελέσματα του υπολογιστικού πυρήνα.
- Γίνεται εκ νέου συγχρονισμός των νημάτων του μπλοκ, ώστε μετά το πέρας αυτού να είναι σίγουρο ότι όλα τα νήματα έχουν γράψει τα τελικά τους αποτελέσματα στην διαμοιραζόμενη μνήμη.
- Τέλος, μεταφέρονται στην καθολική μνήμη τα αποτελέσματα και ακολουθεί πιθανότατα μια μεταφορά από την συσκευή στον host.

4.1.6 Προσεκτική πρόσβαση στην καθολική μνήμη της συσκευής

Για να βρεθεί ο τρόπος να παίρνουν όσο το δυνατόν λιγότερο χρόνο οι προσβάσεις στην καθολική μνήμη, πρέπει να μελετηθεί ο τρόπος με τον οποίο γίνεται η πρόσβαση σε αυτήν και να γίνει εκμετάλλευση του διαθέσιμου εύρους ζώνης στο έπακρο. Όπως θα φανεί και στο επόμενο κεφάλαιο, αυτές οι προσβάσεις αποτελούν από τις πιο χρονοβόρες λειτουργίες των εκάστοτε υπολογιστικών πυρήνων. Επομένως, αν

επιτευχθεί ο περιορισμός του χρόνου που χρειάζονται, μειώνεται σημαντικά ο χρόνος εκτέλεσης των εκάστοτε εφαρμογών.

4.1.6.1 Μείωση εντολών φόρτωσης με χρήση προσδιοριστών στοίχισης

Καταρχήν θα πρέπει να αναφερθεί ότι η συσκευή μπορεί να μεταφέρει με μια εντολή, λέξεις μεγέθους 32, 64 ή 128 bits από την καθολική μνήμη σε έναν καταχωρητή, οπότε είναι προτιμότερο να οργανώνονται έτσι τα δεδομένα ώστε να χρειάζονται όσο το δυνατόν λιγότερες εντολές φόρτωσης. Σε αυτό βοηθούν οι προσδιοριστές στοίχισης προς τον μεταγλωττιστή: `__align__(8)` και `__align__(16)`, με τους οποίους μπορεί κανείς να δημιουργήσει δομές που να στοιχίζονται σε μια λέξη 8 ή 16 bytes (δηλαδή 64 ή 128 bits), αντίστοιχα. Για παράδειγμα, έστω ότι είναι επιθυμητό να φορτωθούν πέντε αριθμοί κινητής υποδιαστολής μονής ακρίβειας, 32 bits ο καθένας. Αν επιχειρηθεί να φορτωθούν ένας-ένας, ο μεταγλωττιστής θα εκδώσει 5 διαφορετικές εντολές. Αν όμως είχε δηλωθεί μια `struct` ως εξής:

```
struct __align__(16) {
     float a;
     float b;
     float c;
     float d;
     float e;
}
```

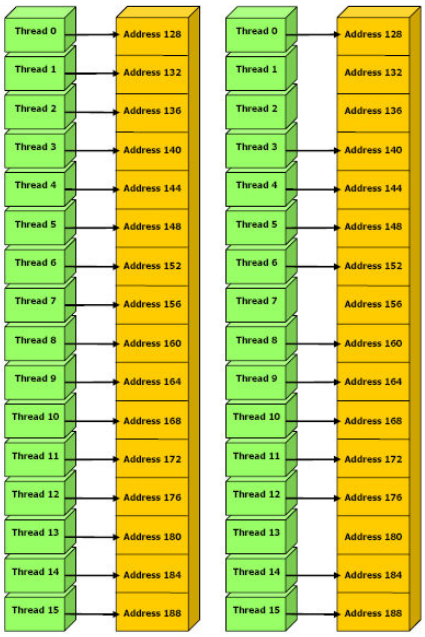
η μεταγλώττιση θα είχε αποτέλεσμα την έκδοση 2 εντολών φόρτωσης 128 bits καθώς οι floats θα ήταν συνεχόμενοι στη μνήμη στο πλαίσιο της struct και όταν γινόταν μεταφορά της παραπάνω δομής, θα είχαμε αρχικά τη μετακίνηση των τεσσάρων πρώτων floats με μια εντολή φόρτωσης 128 bits και στη συνέχεια μια δεύτερη εντολή φόρτωσης 128 bits. Με τους προσδιοριστές στοίχισης, δηλαδή, μπορεί να επιτευχθεί η επιθυμητή οργάνωση των δεδομένων στην καθολική μνήμη ώστε να μειωθούν οι εντολές φόρτωσης και κατά συνέπεια οι κύκλοι ρολογιού που χρειάζονται για τη μεταφορά των δεδομένων.

4.1.6.2 Συγχώνευση προσβάσεων στην καθολική μνήμη

Οι προσβάσεις στην καθολική μνήμη από τα νήματα της εφαρμογής γίνονται στο πλαίσιο ενός ημι-στημονιού, δηλαδή τα νήματα ενός ημι-στημονιού προσπαθούν να αποκτήσουν πρόσβαση ταυτόχρονα στις θέσεις μνήμης που επιτάσσει ο υπολογιστικός πυρήνας. Αυτές οι αναγνώσεις/εγγραφές θα πρέπει να είναι σε συνεχόμενες και ευθυγραμμισμένες θέσεις μνήμης, για να επιτυγχάνεται η ταχύτερη δυνατή εκτέλεση της απαιτούμενης κάθε φορά λειτουργίας. Όταν συμβαίνει κάτι τέτοιο, επιτυγχάνεται συγχωνευμένη πρόσβαση στη μνήμη.

Πιο συγκεκριμένα, πρέπει οι θέσεις μνήμης που θα αναγνωστούν/εγγραφούν από νήματα με διαδοχικούς αύξοντες αριθμούς μέσα στο ημι-στημόνι να είναι και αυτές διαδοχικές στην καθολική μνήμη. Επίσης, πρέπει οι διευθύνσεις αυτές να είναι ευθυγραμμισμένες σε `16*sizeof(type)` bytes, όπου type είναι ένας τύπος δεδομένων 32, 64 ή 128 bit. Δηλαδή, το πρώτο νήμα του ημι-στημονιού θα πρέπει να προσπαθεί να αποκτήσει πρόσβαση σε μια διεύθυνση, η οποία θα είναι πολλαπλάσιο του `16*sizeof(type)`, όπου `type` ο τύπος δεδομένων τον οποίο θα αναγνώ-

σει/εγγράψει το νήμα και τα επόμενα 15 νήματα θα πρέπει να προσπελάσουν τις επόμενες 15 διευθύνσεις που θα περιέχουν δεδομένα του τύπου `type`. Τα παραπάνω θα ξεκαθαρίσουν στα Σχήματα 4-3 έως 4-5 στα αντίστοιχα ενδεικτικά παραδείγματα.



(α) **(β)**

Σχήμα 4-3 Παραδείγματα Συγχωνευμένων Προσβάσεων στην καθολική μνήμη: (α) Προσπέλαση δεδομένων τύπου `float` με συνεχόμενο και ευθυγραμμισμένο τρόπο από όλα τα νήματα του ημι-στημονιού. (β) Προσπέλαση δεδομένων `float` με συνεχόμενο και ευθυγραμμισμένο τρόπο, αλλά όχι από όλα τα νήματα του ημι-στημονιού.

(α) **(β)**

Σχήμα 4-4 Παραδείγματα Μη Συγχωνευμένων Προσβάσεων στην καθολική μνήμη: (α) Προσπέλαση δεδομένων τύπου `float` με μη συνεχόμενο τρόπο από όλα τα νήματα του ημι-στημονιού (περίπτωση μη αντιστοιχίας αύξοντος αριθμού νήματος εντός ημι-στημονιού με αντίστοιχη θέση μνήμης). (β) Προσπέλαση δεδομένων `float` με μη ευθυγραμμισμένο τρόπο από όλα τα νήματα του ημι-στημονιού (η αρχική διεύθυνση δεν είναι πολλαπλάσιο του `16 * sizeof(float)`).

Σχήμα 4-5 Επιπλέον Παραδείγματα Μη Συγχωνευμένων Προσβάσεων στην καθολική μνήμη: (α) Προσπέλαση δεδομένων τύπου `float` με μη συνεχόμενο τρόπο από όλα τα νήματα του ημιστημονιού (περίπτωση παράκαμψης μιας διεύθυνσης). (β) Προσπέλαση δεδομένων `float3`, τα οποία είναι μεγέθους 12 byte το καθένα, καθιστώντας αδύνατη τη συγχωνευμένη πρόσβαση αφού δεν μπορούν να ευθυγραμμιστούν.

Συνήθως, όταν είναι επιθυμητό να προσπελάσει κάθε νήμα της εφαρμογής ένα στοιχείο ενός μεγάλου πίνακα ή διανύσματος, το οποίο, για παράδειγμα, έχει διεύθυνση βάσης `Arraybase` τύπου `type *` είναι αρκετά εύκολο να επιτευχθούν συγχωνευμένες προσβάσεις. Αυτό γίνεται αν το κάθε νήμα (με διακριτικό `tID` ως προς το συνολικό πλέγμα) αναγνώσει/εγγράψει τη διεύθυνση `Arraybase+tID`. Αυτός είναι και ο πιο συνηθισμένος και αποδοτικός τρόπος χρήσης της καθολικής μνήμης αν πρόκειται δεδομένα 32, 64 ή 128 bit.

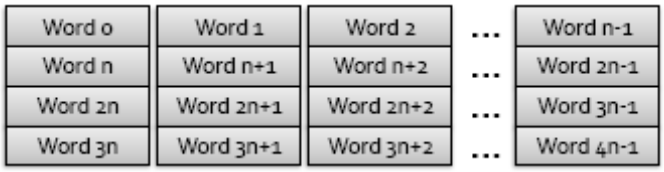
Επιπροσθέτως, είναι προτιμώμενο να οργανώνονται οι προσβάσεις στην καθολική μνήμη με τέτοιο τρόπο, ώστε όλα τα νήματα του στημονιού να στοχεύουν συνεχόμενες και ευθυγραμμισμένες διευθύνσεις για λόγους συμβατότητας του κώδικά με μελλοντικές συσκευές (ώστε οι προσβάσεις να είναι συγχωνευμένες και σε αυτές).

Έχει παρατηρηθεί ότι οι συγχωνευμένες προσπελάσεις σε δεδομένα 64 ή 128 bit επιτυγχάνουν χρήση μικρότερου εύρους ζώνης σε σχέση με αυτές των 32 bit. Οι συγχωνευμένες προσβάσεις σε δεδομένα 32 bit χρησιμοποιούν μια τάξη μεγέθους περισσότερο εύρος ζώνης σε σχέση με τις μη συγχωνευμένες, οι συγχωνευμένες προσβάσεις δεδομένων 64 και 128 bit κάνουν χρήση τετραπλάσιου και διπλάσιου εύρους ζώνης από τις αντίστοιχες μη συγχωνευμένες.

4.1.7 Προσεκτική πρόσβαση στις διαμοιραζόμενες μνήμες της συσκευής

Στο προηγούμενο κεφάλαιο, μιλώντας για τις διαμοιραζόμενες μνήμες (ενότητα 3.2.10), αναφέρθηκε ότι οι προσβάσεις σε αυτές κοστίζουν σε κύκλους ρολογιού όσο και η πρόσβαση στους καταχωρητές του αρχείου καταχωρητών. Θα ερευνηθεί σε αυτήν την ενότητα τι πρέπει να προσεχθεί ώστε να ισχύει το παραπάνω και ποιες περιπτώσεις πρέπει να αποφεύγονται, για να μην υπεισέρχονται πρόσθετες καθυστερήσεις.

Για να επιτευχθεί χρήση μεγάλου εύρος ζώνης κατά την προσπέλαση δεδομένων που βρίσκονται στην διαμοιραζόμενη μνήμη, αυτή είναι διαιρεμένη σε 16 ισόποσα μέρη, τις τράπεζες. Είναι δυνατόν να επιτυγχάνεται πρόσβαση ταυτόχρονα σε κάθε μια από αυτές την ίδια χρονική στιγμή από διαφορετικά νήματα. Η οργάνωση αυτή είναι φανερή στο Σχήμα 4-6. Συνεπώς, 16 εγγραφές/αναγνώσεις που αρχίζουν στον ίδιο κύκλο μπορούν να ικανοποιηθούν ταυτόχρονα αρκεί κάθε μια από αυτές τις λειτουργίες να απευθύνεται σε διεύθυνση μνήμης που ανήκει σε ξεχωριστή τράπεζα. Τελικά, το συνολικό εύρος ζώνης θα είναι ίσο με το 16-πλάσιο της προσπέλασης μιας τράπεζας.



Σχήμα 4-6 Οργάνωση των 32-bit λέξεων (words) σε n (n=16 για τις συσκευές που υποστηρίζουν CUDA 1.0 και 1.1) τράπεζες. Κάθε στήλη αντιστοιχεί σε μια τράπεζα. Παρατηρούμε ότι διαδοχικές λέξεις στεγάζονται σε διαδοχικές θέσεις μνήμης ώστε η τράπεζα στην οποία αντιστοιχίζεται να είναι εκείνη με αύξων αριθμό ίσο με word# % 16.

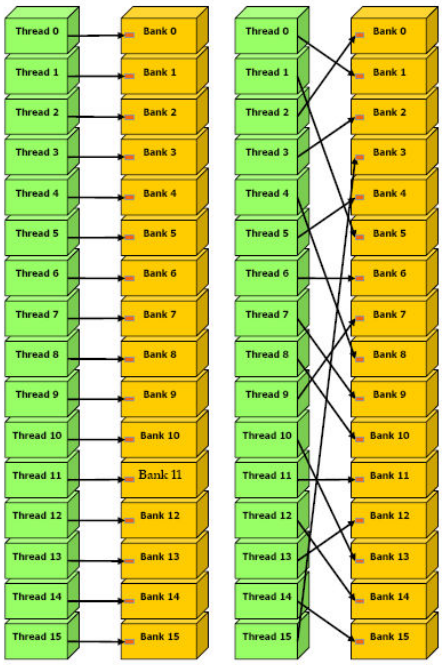
Σε αντίθετη περίπτωση, αν κάποιες από τις 16 εγγραφές/αναγνώσεις απευθύνονται στην ίδια τράπεζα, οι προσπελάσεις θα πρέπει να σειριοποιηθούν καθώς θα υπάρξει διαμάχη πρόσβασης στην τράπεζα (bank-conflict). Τότε η συσκευή αναλύει τις προσβάσεις σε δύο ή περισσότερα γκρουπ προσβάσεων, στο κάθε ένα από τα οποία δεν θα υπάρχει διαμάχη. Αποτέλεσμα είναι η μείωση του εύρους ζώνης μεταφορών από και προς την διαμοιραζόμενη μνήμη. Αν υπάρχουν *n* αιτήσεις εγγραφής/ανάγνωσης προς την ίδια τράπεζα θα αναλυθούν οι προσβάσεις σε *n* γκρουπ, σε καθένα από τα οποία δεν θα υπάρχει διαμάχη, με στόχο την επίλυση της διαμάχης *n*-δρόμων (*n*-way bank-conflict). Διαμάχη μπορεί να υπάρξει μεταξύ διαφορετικών νημάτων του ίδιου ημι-στημονιού και όχι μεταξύ ενός νήματος του πρώτου ημι-στημονιού με ένα νήμα του δεύτερου ημι-στημονιού του ίδιου στημονιού, καθώς, οι προσβάσεις στη διαμοιραζόμενη μνήμη γίνονται για κάθε ημι-στημόνι ξεχωριστά.

Μια σημαντική εξαίρεση αποτελεί η περίπτωση που όλα τα νήματα του ημι-στημονιού θέλουν να διαβάσουν την ίδια διεύθυνση η οποία ανήκει σε μια τράπεζα. Τότε, αντί για διαμάχη 16-δρόμων γίνεται εκπομπή της 32-bit λέξης σε όλα τα νήματα. Αυτό το γεγονός εξαλείφει και τις διαμάχες που εμφανίζονται όταν νήματα προσπαθούν να διαβάσουν δεδομένα μεγέθους μικρότερου των 32 bit, και που αν τα νήματα προσπέλαζαν διαδοχικά αυτά τα δεδομένα, θα εμφανιζόταν διαμάχη πολλαπλών δρόμων. Πιο συγκεκριμένα, και για να γίνει πιο σαφές το τελευταίο, πρέπει να αναφερθεί ότι οι αιτήσεις αναγνώσεων ικανοποιούνται σε περισσότερα του ενός βήματα όταν υπάρχουν διαμάχες (ένα βήμα κάθε δυο κύκλους). Σε κάθε βήμα γίνεται ικανο-

ποίηση ενός υποσυνόλου των αιτήσεων ανάγνωσης, ώστε σε αυτό το υποσύνολο να μην υπάρχει καμία απολύτως διαμάχη, ενώ η απόφαση για το ποιό ακριβώς θα είναι αυτό το υποσύνολο λαμβάνεται ως εξής:

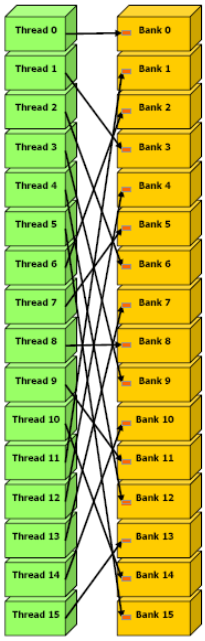
- Επιλογή με ακαθόριστο τρόπο μιας από τις λέξεις των εναπομεινάντων διευθύνσεων ως λέξη εκπομπής.
- Όλες οι διευθύνσεις δεδομένων που περιλαμβάνονται στην λέξη εκπομπής προστίθεται στο σχηματιζόμενο υποσύνολο.
- Προστίθεται στο υποσύνολο αυτό, μια διεύθυνση για κάθε τράπεζα στην οποία δείχνουν οι εναπομείνασες διευθύνσεις.

Ανακεφαλαιώνοντας, η κάθε διαμοιραζόμενη μνήμη περιέχει 16 τράπεζες ίσου μεγέθους (1 KB η κάθε μια) και είναι οργανωμένη έτσι, ώστε διαδοχικές λέξεις μεγέθους 32 bit να ανήκουν σε διαδοχικές τράπεζες με κάθε μια από αυτές να προσφέρει εύρος ζώνης ίσο με 32 bit ανά δύο κύκλους μηχανής. Στα Σχήματα 4-7 έως 4-10 φαίνονται διάφορες περιπτώσεις προσπελάσεις λέξεων μεγέθους 32 bit που βρίσκονται στην διαμοιραζόμενη μνήμη, και σε κάθε περίπτωση μελετάται αν υπάρχει ή όχι διαμάχη πρόσβασης στην ίδια τράπεζα.

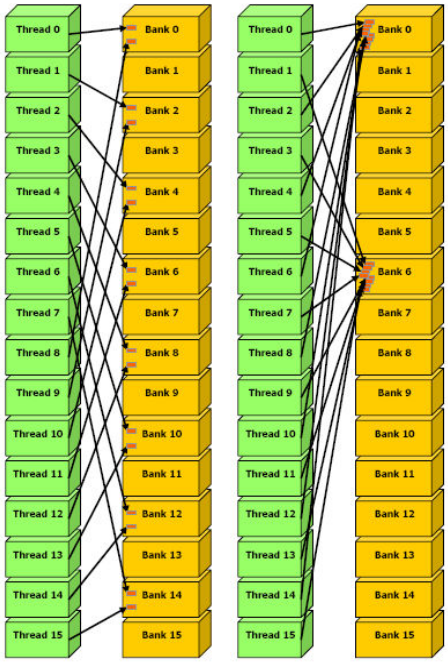


(α) **(β)**

Σχήμα 4-7 Παραδείγματα Πρόσβασης στην διαμοιραζόμενη μνήμη με αποφυγή κάθε διαμάχης: (α) Προσπέλαση δεδομένων μεγέθους 32 bit χωρίς την δημιουργία διαμάχης, καθώς κάθε νήμα του ημι-στημονιού προσπελάζει διαφορετική τράπεζα αντίστοιχη με τον αύξοντα αριθμό του μέσα στο ημι-στημόνι. (β) Προσπέλαση δεδομένων μεγέθους 32 bit χωρίς την δημιουργία διαμάχης, καθώς κάθε νήμα του ημι-στημονιού προσπελάζει διαφορετική τράπεζα με κάπως τυχαίο, όμως, τρόπο.

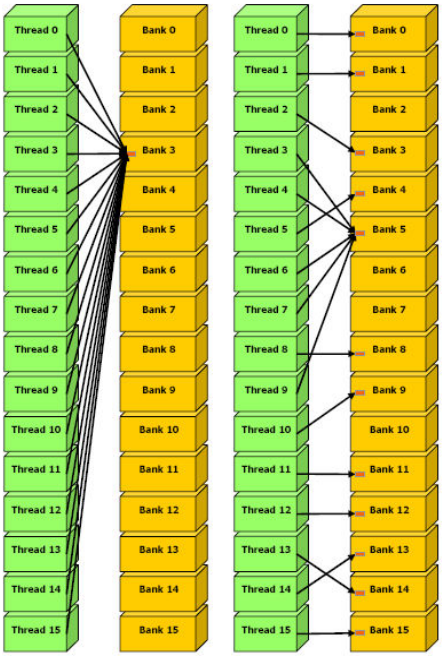


Σχήμα 4-8 Επιπλέον Παράδειγμα Πρόσβασης στην διαμοιραζόμενη μνήμη με αποφυγή κάθε διαμάχης: Εδώ κάθε νήμα προσπελάζει λέξη μεγέθους 32 bit της οποίου η διεύθυνση είναι 3 θέσεις μετά το δεδομένο που προσπελάζει το νήμα με τον αμέσως προηγούμενο αύξοντα αριθμό. Ούτε εδώ υπάρχει δημιουργία διαμάχης καθώς κάθε τράπεζα προσπελάζεται από ένα μοναδικό νήμα.



(α) **(β)**

Σχήμα 4-9 Παραδείγματα Πρόσβασης στην διαμοιραζόμενη μνήμη με δημιουργία διαμάχης: (α) Προσπέλαση δεδομένων μεγέθους 32 bit με την δημιουργία διαμάχης 2-δρόμων, καθώς κάθε τράπεζα προσπελάζεται από δυο διαφορετικά νήματα του ημι-στημονιού. (β) Προσπέλαση δεδομένων μεγέθους 32 bit με την δημιουργία διαμάχης 8-δρόμων, καθώς κάθε τράπεζα προσπελάζεται από οκτώ διαφορετικά νήματα του ημι-στημονιού.



Σχήμα 4-10 Πρόσθετα Παραδείγματα Πρόσβασης στην διαμοιραζόμενη μνήμη: (α) Προσπέλαση διευθύνσεων που ανήκουν στην ίδια 32-bit λέξη από όλα τα νήματα του ημι-στημονιού, με αποτέλεσμα την ενεργοποίηση του μηχανισμού εκπομπής ώστε να μην καθυστερήσουν οι προσπελάσεις λόγω διαμαχών. (β) Ένας τρόπος προσπέλασης ο οποίος μπορεί να δημιουργήσει διαμάχη 2-δρόμων (στο πρώτο βήμα θα ικανοποιηθούν οι αιτήσεις ανάγνωσης σε όλες τις τράπεζες εκτός της #5 και μια από τις αιτήσεις προς την τράπεζα #5, και στο δεύτερο βήμα θα ικανοποιηθούν όλες οι υπόλοιπες) ή καμία διαμάχη αν επιλεγεί η τράπεζα #5 σαν η λέξη εκπομπής στο πρώτο βήμα.

Συνοψίζοντας, καταλήγει κανείς στο συμπέρασμα πως η χρήση της διαμοιραζόμενης μνήμης, εφόσον φροντίζεται να μην συμβαίνουν διαμάχες στις τράπεζές της κατά την πρόσβαση, αποτελεί σημαντικό δρόμο για τη βελτιστοποίηση των αλγορίθμων, καθώς το κόστος προσπέλασης μιας λέξης της είναι όσο το κόστος προσπέλασης ενός καταχωρητή. Συνεπώς, η σωστή αξιοποίηση της Ιεραρχίας Μνήμης με μεταφορά των δεδομένων εισόδου που θα επεξεργαστεί το κάθε μπλοκ στην διαμοιραζόμενη μνήμη είναι ιδιαίτερα κερδοφόρα στρατηγική, αν ληφθούν υπόψη όσα αναλύθηκαν σε αυτήν την ενότητα.

4.1.8 Αποφυγή καθυστερήσεων πρόσβασης στους καταχωρητές

Η καθυστέρηση πρόσβασης σε δεδομένα που υπάρχουν σε καταχωρητές είναι μηδενική αφού το αρχείο καταχωρητών βρίσκεται σε μηδενική χρονική απόσταση από τους SPs. Όμως, υπάρχουν δυο περιπτώσεις στις οποίες μπορεί να εμφανιστούν επιπλέον καθυστερήσεις.

Η πρώτη από αυτές είναι σε περιπτώσεις όπου συμβαίνει διάβασμα-μετά-από-εγγραφή σε έναν καταχωρητή. Η νέα τιμή δε θα μπορεί να διαβαστεί από την εντολή ανάγνωσης μέχρι να εγγραφεί από την προηγούμενη εντολή εγγραφής, όπως δηλαδή συμβαίνει και σε άλλα υπολογιστικά συστήματα με στάδια σωλήνωσης. Πάντως αυτές οι καθυστερήσεις δεν εμφανίζονται αν εξασφαλιστεί ότι κάθε στιγμή υπάρχουν τουλάχιστον 192 ενεργά νήματα σε κάθε SM (ανεξάρτητα από το αν θα ανήκουν στο ίδιο ή σε διαφορετικά μπλοκ), ώστε αυτή η καθυστέρηση να «κρύβεται».

Η δεύτερη περίπτωση έχει να κάνει με διαμάχες κατά την πρόσβαση στις τράπεζες μνήμης του αρχείου καταχωρητών. Ο μεταγλωττιστής, κατά τη διαδικασία μεταγλώττισης, και ο χρονοδρομολογητής των νημάτων που υπάρχει σε κάθε SM, κατά τη δυναμική χρονοδρομολόγηση τη στιγμή της εκτέλεσης, φροντίζουν να υπάρχει κατάλληλη διαδοχή εντολών ώστε να αποφεύγονται κατά το δυνατόν οι προαναφερθείσες διαμάχες. Έτσι, τελικά, επιλέγεται ο βέλτιστος τρόπος πρόσβασης στο αρχείο καταχωρητών. Από την πλευρά του προγραμματιστή, αυτό που μπορεί να κάνει αν παρουσιαστεί τέτοιο πρόβλημα είναι να επιλέξει σαν μέγεθος του μπλοκ ένα πολλαπλάσιο του 64.

4.1.9 Αποφυγή αποκλινόντων στημονιών

Η χρήση εντολών ελέγχου της ροής ενός προγράμματός (π.χ. εντολές `if`, `switch`, `do`, `for`, `while` κλπ.) μπορεί να έχει σημαντικό αντίκτυπο στον ρυθμό περάτωσης εντολών του προγράμματος, εάν η αποτίμηση της συνθήκης μιας τέτοιας εντολής γίνει διαφορετικά σε κάθε νήμα. Εάν γίνει κάτι τέτοιο, θα ακολουθηθεί διαφορετικό μονοπάτι εντολών για κάποια από τα νήματα του στημονιού. Σε τέτοια περίπτωση, και αφού το στημόνι δε μπορεί να «σπάσει» και να χωριστεί σε μικρότερες ομάδες νημάτων εκτέλεσης, όλα τα νήματα υποχρεώνονται να εκτελέσουν σειριακά όλες τις εντολές όλων των πιθανών ροών που μπορούν να ακολουθηθούν. Όμως, στο τέλος κάθε νήμα γράφει τα αποτελέσματα μόνο των εντολών που ανήκουν στο μονοπάτι εκτέλεσης του, ενώ τα αποτελέσματα των εντολών των άλλων μονοπατιών αγνοούνται. Μόλις τα διαφορετικά μονοπάτια συγκλίνουν η εκτέλεση συνεχίζεται όπως και πριν την συνθήκη ελέγχου με όμοιες εντολές για όλα τα νήματα του στημονιού. Συνέπεια του παραπάνω είναι η αύξηση των εντολών που πρέπει να εκτελέσει κάθε στημονιού, με πολλές από αυτές να είναι άχρηστες, καθώς τα αντίστοιχα νήματα θα έχουν ακολουθήσει ένα μοναδικό μονοπάτι και τελικά θα τις αγνοήσουν. Αποτέλεσμα είναι η μείωση της επίδοσης καθώς το υλικό απασχολείται με άχρηστες εντολές, καθυστερώντας την ολοκλήρωση των λειτουργιών του υπολογιστικού πυρήνα της εφαρμογής.

Ο προγραμματιστής μπορεί να αποφύγει σε πολλές περιπτώσεις τα αποκλίνοντα στημόνια αν εκφράσει προσεκτικά τη συνθήκη, ώστε νήματα που αναμένεται από τον αλγόριθμό να αποκλίνουν να μην είναι διαδοχικά, με αποτέλεσμα να ανήκουν σε διαφορετικά στημόνια. Με τον περιορισμό των αποκλινόντων στημονιών αυξάνεται η επίδοση των προγραμμάτων και ο ρυθμός περάτωσης εντολών σε όλες τις πιθανές εφαρμογές.

Κάποιες φορές όμως, ο μεταγλωττιστής μπορεί να προσπαθήσει να προβλέψει ποιο μονοπάτι θα ακολουθηθεί μετά από μια συνθήκη ελέγχου και σε αυτήν την περίπτωση κανένα στημόνι δεν μπορεί να αποκλίνει. Αυτή η διαδικασία ονομάζεται πρόβλεψη εντολών (instruction prediction). Κάθε εντολή ενός νήματος του στημονιού συσχετίζεται με έναν κωδικό ο οποίος μπορεί να είναι αληθής ή ψευδής (true ή false) και παρόλο που όλες οι εντολές χρονοδρομολογούνται ανεξάρτητα με την τιμή του κωδικού, στην πραγματικότητα θα εκτελεστούν μόνο οι εντολές με κωδικό true. Οι υπόλοιπες δεν γράφουν τα αποτελέσματά τους, δεν αποτιμούν διευθύνσεις ούτε διαβάζουν τους όρους των πράξεων, αν πρόκειται για αριθμητικές εντολές. Ο μεταγλωττιστής θα αντικαταστήσει τις εντολές των διαφορετικών μονοπατιών που καθορίζονται από την αποτίμηση της συνθήκης ελέγχου με πρόβλεψη εντολών με τον τρόπο που εξηγήθηκε παραπάνω, αν ο αριθμός των εντολών δεν ξεπερνούν ένα συγκεκριμέ-

νο κατώφλι. Αυτό το κατώφλι είναι επτά εντολές, αν ο μεταγλωττιστής κρίνει ότι είναι πιθανή η δημιουργία πολλών αποκλινόντων στημονιών, ενώ αν δεν προβλέπεται να συμβεί κάτι τέτοιο το κατώφλι είναι τέσσερις εντολές.

4.1.10 Τελικά στάδια βελτιστοποίησης

Αφού ληφθούν υπόψη όλα τα παραπάνω κατά την εγγραφή του κώδικά μιας εφαρμογής και αφού δοκιμαστούν τα αποτελέσματα των προαναφερθέντων στρατηγικών, μπορεί να επιχειρηθεί ένα τελευταίο στάδιο βελτιστοποίησης. Μπορούν πλέον να ξετυλιχτούν οι βρόχοι που περιλαμβάνει ο κώδικας για να αποφευχθούν, για παράδειγμα, πράξεις για την αποτίμηση της συνθήκης συνέχισης ή τερματισμού του βρόχου, καθώς και ο υπολογισμός διευθύνσεων ή όρων πράξεων που εξαρτώνται από τον αύξοντα αριθμό της επανάληψης του βρόχου. Τονίζεται ότι αυτό πρέπει να γίνει στο τέλος της προσπάθειας βελτιστοποίησης για να είναι εγγυημένο ότι έχουν εξαντληθεί όλοι οι άλλοι τρόποι βελτιστοποίησης που συνήθως είναι πιο κερδοφόροι.

Μια ακόμα ενέργεια στην οποία μπορεί κανείς να προβεί για να διασφαλίσει την βέλτιστη επίδοση των προγραμμάτων σε οποιαδήποτε συσκευή που υποστηρίζει το CUDA είναι η παραμετροποίηση των υπολογιστικών πυρήνων ανάλογα με τον αριθμό των SMs, του μεγέθους της διαμοιραζόμενης μνήμης, του αρχείου καταχωρητών και του αριθμού των νημάτων που χωράνε σε ένα μπλοκ. Με αυτόν τον τόπο εξασφαλίζεται ότι η εφαρμογή θα μπορεί να εκτελεστεί χωρίς καμία αλλαγή σε οποιαδήποτε μελλοντική κάρτα γραφικών που θα υποστηρίζει το CUDA. Κάτι παρόμοιο μπορεί να γίνει και για περιπτώσεις που ο αλγόριθμός μεταβάλλεται ανάλογα με την είσοδό του. Σε τέτοια περίπτωση, αντί να αποφασίζονται την στιγμή εκτέλεσης διάφοροι παράμετροι, μπορεί να εξοικονομηθεί χρόνος έχοντας φροντίσει να υπάρχουν όλες οι πιθανές εκδόσεις υπολογιστικών πυρήνων και να καλείται σε κάθε περίπτωση ο πιο ταιριαστός. Αρωγός στην προσπάθεια για παραμετροποίηση με το CUDA, αποτελεί η δυνατότητα χρησιμοποίησης των templates της C++, οι συνθήκες των οποίων αποτιμώνται κατά τη μεταγλώττιση, χωρίς να επιβαρύνουν την εφαρμογή κατά την εκτέλεση με περαιτέρω υπολογισμούς.

4.2 Ανακεφαλαίωση Στρατηγικών Βελτιστοποίησης Επίδοσης

Μετά από την παρουσίαση όλων των πιθανών στρατηγικών που μπορούν να εφαρμοστούν για τη βελτιστοποίηση της επίδοσης των προγραμμάτων που απευθύνονται στην μελετώμενη πλατφόρμα, καταλήγει κανείς στο ότι κινούνται σε τρεις κυρίως άξονες:

- Μεγιστοποίηση παραλληλίας του κώδικα.
- Βελτιστοποίηση χρήση μνήμης με στόχο τη μέγιστη χρησιμοποίηση του διαθέσιμου εύρους ζώνης (memory bandwidth).
- Βελτιστοποίηση χρησιμοποιούμενων εντολών με στόχο την μέγιστη δυνατή ταχύτητα περάτωσης εντολών (instruction throughput).

Αρχικά διατυπώθηκε η ανάγκη προσεκτικής μελέτης του αλγορίθμου της υποψήφιας εφαρμογής, ώστε να αποφανθεί κανείς σε ποια σημεία επιδέχεται παραλληλοποίησης, δηλαδή ο εντοπισμός των περιοχών που μπορούν να επιταχυνθούν με χρήση της τε-

χνολογίας του CUDA. Αμέσως μετά πρέπει να ελεγχθούν όλοι οι πιθανοί τρόποι με τους οποίους μπορεί ο αλγόριθμός να απεικονιστεί στη διαθέσιμη συσκευή, προσέχοντας πολύ τις παραμέτρους εκτέλεσης, κάτι που σε μερικές περιπτώσεις επιτυγχάνεται μόνο με εξαντλητικές δοκιμές [58].

Έπειτα, όσον αφορά τη χρησιμοποίηση των διαθέσιμων μνημών, εξήχθη το συμπέρασμα ότι υπάρχει μια ιεραρχία αρκετών επιπέδων που μας προσφέρει αρκετές επιλογές ως προς την αξιοποίηση της. Για να επιτευχθεί το καλύτερο δυνατό αποτέλεσμα πρέπει να ακολουθηθούν κάποιες βασικές αρχές. Θα πρέπει να περιορίζονται οι μεταφορές μεταξύ host και συσκευής, και όταν αυτές γίνονται, να είναι μαζικές και να μην «σπάνε» σε μικρότερες. Ακόμη, οι προσβάσεις στην καθολική μνήμη πρέπει να είναι συγχωνευμένες, ενώ είναι δυνατόν να «κρύβεται» μέρος της καθυστέρησης που αυτές εισάγουν, με την επίτευξη μεγάλου ποσοστού χρησιμοποίησης στα SMs. Σημαντικό είναι επίσης, να γίνεται χρήση της διαμοιραζόμενης μνήμης, καθώς το εύρος ζώνης που προσφέρει όσον αφορά την επικοινωνία με τους πυρήνες είναι πολύ μεγαλύτερο σε σχέση με αυτό της καθολικής μνήμης, όταν αποφεύγονται οι διαμάχες πρόσβασης στις τράπεζές της.

Επιπροσθέτως, προτείνεται η χρησιμοποίηση των πιο «γρήγορων» (αυτών με μικρότερο αριθμό απαιτούμενων κύκλων ρολογιού) εντολών και ο περιορισμός αυτών με την χαμηλή ταχύτητα περάτωσης. Όπου είναι εφικτό να βρεθεί κάποιο τρικ για την αντικατάσταση μια πιο «αργής» εντολής, προτείνεται να γίνεται η αντικατάσταση, προσέχοντας, πάντως, η τελική ακρίβεια των υπολογιστών να είναι αποδεκτή. Αναφέρθηκε ότι υπάρχει και ένας αριθμός ταχέων συναρτήσεων που αντικαθιστούν τις κλασικές που προσφέρει η γλώσσα C και μπορούμε κάθε φορά να τις αντικαθιστούμε, αν είναι επιθυμητό να ανταλλαχθεί ταχύτητα (που προσφέρουν οι ταχείες συναρτήσεις του CUDA) με ακρίβεια (που προσφέρουν οι συναρτήσεις της C). Ιδιαίτερα προσεκτικοί χειρισμοί θα πρέπει να γίνονται όπου υπάρχει έλεγχος ροής με σκοπό την αποφυγή της εμφάνιση αποκλινόντων στημονιών.

Τέλος, είναι ανάγκη να αναφερθεί ότι ο χώρος βελτιστοποίησης όσον αφορά τις παραμέτρους εκτέλεσης δεν είναι γραμμικά μεταβαλλόμενος και παρουσιάζει πολλές και απότομες ασυνέχειες και διακυμάνσεις [58,60], εξ' ου και η ανάγκη εξαντλητικών μετρήσεων ορισμένες φορές. Ο λόγος είναι ότι οποιαδήποτε αλλαγή σε κάποιον τομέα της εφαρμογής δεν είναι ανεξάρτητη από όλες τις υπόλοιπες πτυχές του προγράμματος και μπορεί να επηρεάσει τελικά την συνολική επίδοση με τρόπο, πολλές φορές, μη ευδιάκριτα προβλέψιμο. Προσπάθειες έχουν γίνει ώστε να βρεθεί ένας πιο αυτοματοποιημένος τρόπος αναγνώρισης των ιδανικών παραμέτρων εκτέλεσης για οποιαδήποτε εφαρμογή, ώστε να μειωθεί ο χώρος στον οποίο αναγκάζονται οι προγραμματιστές να πραγματοποιούν εξονυχιστικές μετρήσεις, αλλά είναι ακόμα σε πρώιμο στάδιο και είναι αποτελεσματικός κάτω από προϋποθέσεις [60].

Ολοκληρώνοντας και συνοψίζοντας όσα αναλύθηκαν σε αυτό το κεφάλαιο συμπεραίνεται ότι προκειμένου η εκάστοτε εφαρμογή μας να είναι βέλτιστα δομημένη και να παρουσιάζει όσο το δυνατόν μεγαλύτερη επίδοση, πρέπει να ελεγχθούν έξι διαφορετικοί, μα όχι ανεξάρτητοι μεταξύ τους, τομείς:

I. *Χρήση του συστήματος μνήμης με τον αποδοτικότερο δυνατό τρόπο*, όπως αναλύθηκε στην ενότητα 4.1.

II. *Ύπαρξη αρκετών ενεργών στημονιών,* με τις ευεργετικές συνέπειες που έχει αυτό στη χρησιμοποίηση των SMs, όπως έγινε φανερό από την παραπάνω ανάλυση.
III. *Κατανομή της δουλειάς σε νήματα και μπλοκ νημάτων με τον αποδοτικότερο τρόπο*, κάτι που εξαρτάται από τον εκάστοτε αλγόριθμό. Δηλαδή, καθώς οι συνολικές λειτουργίες μπορούν να διαμοιραστούν με πολλούς διαφορετικούς τρόπους στα νήματα της εφαρμογής, στόχος πρέπει να είναι η εύρεση του τρόπου που ταιριάζει περισσότερο στην αρχιτεκτονική G80.
IV. *Μείωση του δυναμικού αριθμού εντολών καθενός νήματος και των εντολών με μικρή ρυθμοαπόδοση*, δηλαδή εφαρμογή κάποιων γνωστών τεχνικών από άλλες πλατφόρμες, όπως το ξετύλιγμα βρόχων, ώστε να μειωθούν εντολές που θα έχουν να κάνουν με ελέγχους σχετικά με τον βρόχο αυτό, και επιλογή όσο το δυνατόν ταχύτερων και ελαφρύτερων εντολών με αποφυγή κατά το δυνατόν των ιδιαίτερα χρονοβόρων.
V. *Προσπάθεια για παραλληλισμό εντός του νήματος*, με διαδοχικές εντολές να είναι όσο το δυνατόν ανεξάρτητες μεταξύ τους ώστε να μην εμφανίζονται εξαρτήσεις και να μην αναμένουν κάποιες εντολές τελεστές που θα τους παρέχουν οι αμέσως προηγούμενες εντολές. Αυτή η προσπάθεια θα έχει ως αποτέλεσμα την τροφοδότηση περισσότερων ανεξάρτητων μεταξύ τους εντολών με απώτερο σκοπό την συνεχή λειτουργία των SMs και την αποφυγή κάθε είδους μη επιθυμητής καθυστέρησης.
VI. *Προσεκτική ανάθεση πόρων στα νήματα,* ώστε να επωφελείται η εφαρμογή από την αύξηση της χρησιμοποίησης και να παρατηρούνται οι ελάχιστες δυνατές «φουρνιές» από μπλοκ για κάθε σύνολο παραμέτρων εκτέλεσης.

Κεφάλαιο 5

Πειραματική αξιολόγηση της αρχιτεκτονικής G80

Σε αυτό το κεφάλαιο, μέσα από μια σειρά πειραμάτων θα γίνει προσπάθεια να αποτιμηθεί το κατά πόσο η αρχιτεκτονική G80 είναι κατάλληλη για χρησιμοποίηση σε εφαρμογές γενικού σκοπού. Μέσω της προγραμματιστικής πλατφόρμας που προσφέρει η εταιρεία nVidia, το CUDA, θα παρουσιαστούν κάποιοι στοιχειώδεις πυρήνες που χρησιμοποιούνται ιδιαίτερα συχνά σε επιστημονικές εφαρμογές σε πολλούς διαφορετικούς τομείς. Κάνοντας χρήση των στρατηγικών βελτιστοποίησης που παρουσιάστηκαν σε προηγούμενο κεφάλαιο, θα γίνει απόπειρα να βρεθούν οι βέλτιστες δυνατές υλοποιήσεις των παραπάνω πυρήνων, με σκοπό την πλήρη χρησιμοποίηση της αρχιτεκτονικής G80. Μέσα από αυτήν την διαδικασία, θα γίνουν πιο ξεκάθαρες πολλές από τις πτυχές αυτής της νέας αρχιτεκτονικής καθώς και τα αποτελέσματα της εφαρμογής των διαφόρων στρατηγικών βελτιστοποίησης που έχουν αναφερθεί.

Αρχικά, θα εξερευνηθεί η ταχύτητα εκτέλεσης αριθμητικών εντολών της διαθέσιμης κάρτας γραφικών. Η αρχιτεκτονική G80 αποτελεί μια ιδανική λύση για αριθμητικά απαιτητικές εφαρμογές, λόγω του μεγάλου αριθμού επεξεργαστικών πυρήνων. Για να αξιολογηθούν οι διαφαινόμενες δυνατότητές της, θα πραγματοποιηθούν μετρήσεις κάποιων μετροπρογραμμάτων μέτρησης της ρυθμοαπόδοσης αριθμητικών εντολών. Στη συνέχεια, θα παρουσιαστούν υλοποιήσεις ενός παράλληλου αλγορίθμου αναγω-

γής, ενός πυρήνα αρκετά συχνά εμφανιζόμενου σε επιστημονικές εφαρμογές. Θα παρουσιαστούν αρκετές διαφορετικές εκδόσεις, οι οποίες θα εφαρμόζουν διάφορες στρατηγικές βελτιστοποίησης, για να αξιολογηθούν οι επιπτώσεις των τελευταίων στην επίδοση ενός πραγματικού πυρήνα. Τέλος, θα υλοποιηθεί ένας αλγόριθμος πολλαπλασιασμού πυκνού πίνακα επί διάνυσμα, μια από τις πλέον διαδεδομένες πράξεις γραμμικής άλγεβρας σε πραγματικές υπολογιστικές εφαρμογές. Και για αυτήν την εφαρμογή, θα παρουσιαστεί ένας αριθμός διαφορετικών εκδόσεων, οι οποίες θα εφαρμόζουν διαδοχικά αρκετές από τις διαθέσιμες στρατηγικές βελτιστοποίησης. Στο τέλος αυτού του κεφαλαίου θα μπορούν να εξαχθούν συμπεράσματα για τις δυνατότητες χρησιμοποίησης της αρχιτεκτονικής G80 σε πραγματικούς υπολογισμούς γενικού σκοπού.

5.1 Πειραματική πλατφόρμα

Στα επόμενα πειράματα χρησιμοποιήθηκε το μοντέλο nVidia GeForce Ultra 8800, το οποίο περιλαμβάνει 16 SMs (128 SPs συνολικά), 768 MB καθολικής μνήμης, υποστηρίζει την έκδοση 1.0 του CUDA και κατά τα άλλα ακολουθεί την αρχιτεκτονική G80. Ο επεξεργαστής του υπολογιστικού συστήματος ήταν ένας Intel Core2 Quad, στα 2,4 GHz ο κάθε ένας από τους τέσσερις πυρήνες. Αυτός περιλαμβάνει δυο κρυφές μνήμες επιπέδου 1, μια για εντολές και μια για δεδομένα, μεγέθους 32KB η κάθε μια. Επίσης, περιλαμβάνεται μια κρυφή μνήμη επιπέδου 2 για δεδομένα, μεγέθους 4MB. Η κύρια μνήμη του συστήματος είχε μέγεθος 2GB και ήταν χρονισμένη στα 667 MHz. Η μητρική μονάδα που χρησιμοποιήθηκε ήταν βασισμένη στο τσιπ nVidia n680i, καθώς οι μητρικές αυτής της οικογένειας συνεργάζονται καλύτερα από οποιαδήποτε άλλη με τις κάρτες γραφικών της nVidia. Η επικοινωνία μεταξύ GPU και CPU έγινε μέσω του διαύλου PCIe. Το εγκατεστημένο λειτουργικό σύστημα ήταν το GNU/Linux 2.6.18, το οποίο είναι ένα 64-bit λειτουργικό. Για τον κώδικα των υλοποιήσεων της CPU χρησιμοποιήθηκε ο μεταγλωττιστής gcc v4.1.2, ενώ για τον κώδικα που περιελάμβανε κλήσεις του CUDA έγινε χρήση του nvcc v0.2.1221.

5.2 Μετροπρογράμματα

Σε αυτήν την ενότητα θα επιχειρηθεί να φτάσει η κάρτα γραφικών GeForce 8800 Ultra της nVidia στα όρια της, όσον αφορά την ταχύτητα εκτέλεσης πράξεων μεταξύ αριθμών κινητής υποδιαστολής. Θα ερευνήσουμε τις περιπτώσεις πρόσθεσης (πράξη ADD), πολλαπλασιασμού (πράξη MUL), πολλαπλασιασμού-πρόσθεσης ταυτόχρονα (πράξη MAD), καθώς και την περίπτωση πράξεων πολλαπλασιασμού-πρόσθεσης και πολλαπλασιασμού (πράξεις MAD-MUL). Από τις προδιαγραφές πο της συγκεκριμένης αρχιτεκτονικής, αναμένεται η μεγαλύτερη ταχύτητα να επιτυγχάνεται στην περίπτωση των πράξεων MAD και στην περίπτωση MAD-MUL, καθώς σε αυτήν την αρχιτεκτονική είναι δυνατή η διπλή-έκδοση (dual-issue) εντολών MAD και MUL ταυτόχρονα. Θα εξεταστούν ποιοι παράγοντες επιτρέπουν να προσεγγιστούν οι θεωρητικά μέγιστες δυνατές ταχύτητες εκτέλεσης, καθώς και ποια στοιχεία είναι περιοριστικά για να συμβεί κάτι τέτοιο.

Στα επόμενα πειράματα, θα χρησιμοποιηθεί ένα μετροπρόγραμμα το οποίο γενικά δεν θα είχε κάποια πρακτική χρησιμότητα σε γενικές εφαρμογές, αλλά με τις μετρήσεις που θα πραγματοποιηθούν, θα καταστεί δυνατό να προβεί κανείς σε κάποια συμπε-

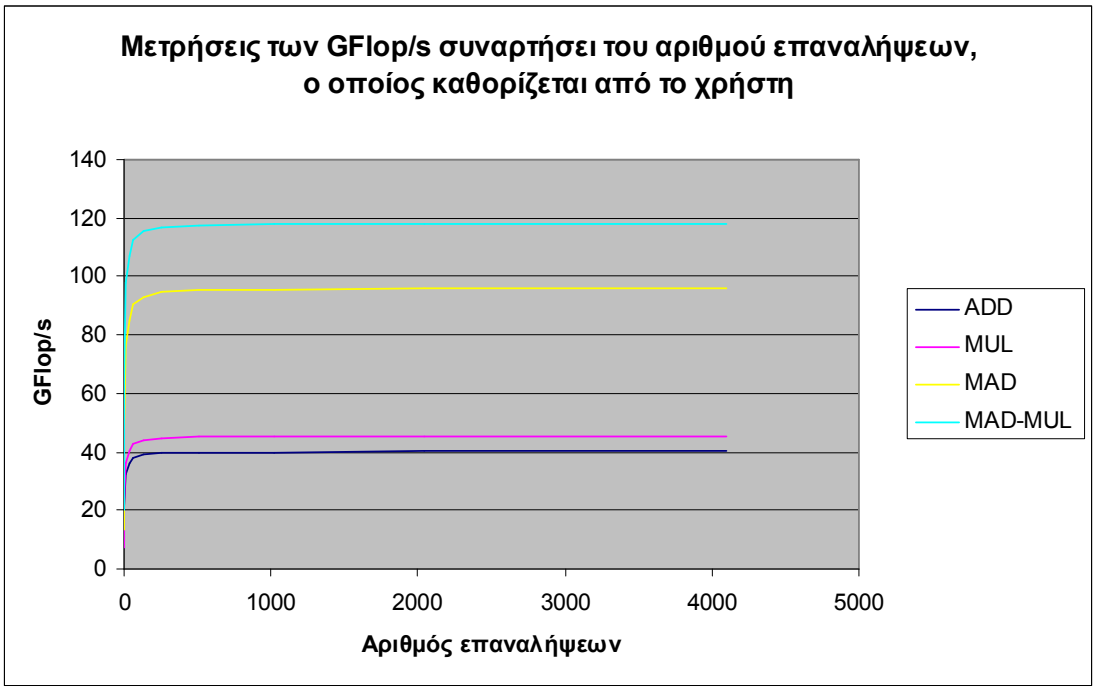
ράσματα για την ρυθμοαπόδοση των παραπάνω εντολών και τις δυνατότητες της κάρτας γραφικών σε ταχύτητα εκτέλεσης πράξεων. Επίσης, θα εξαχθούν και κάποια συμπεράσματα που θα βοηθήσουν στην προσπάθεια επίτευξης της μέγιστης δυνατής επίδοσης σε προγράμματα πραγματικών εφαρμογών με μεγάλες απαιτήσεις σε υπολογισμούς.

Το μετροπρόγραμμα το οποίο θα χρησιμοποιηθεί στα επόμενα πειράματα, αρχικά θα φορτώνει από τον host στην κάρτα γραφικών δυο σχετικά μεγάλα διανύσματα με στοιχεία `float`. Έπειτα, κάθε νήμα θα αποθηκεύει σε δύο καταχωρητές ένα από τα στοιχεία του κάθε διανύσματος και μετά, θα εκτελεί έναν αριθμό πράξεων (τον οποίο είτε θα εισάγει ο χρήστης στη γραμμή εντολών είτε θα είναι προκαθορισμένη) μεταξύ αυτών των δύο αριθμών. Το τελικό αποτέλεσμα θα αποθηκεύεται σε ένα καινούργιο διάνυσμα (κάθε νήμα θα συνεισφέρει ένα στοιχείο του τελικού διανύσματος) και αυτό θα αντιγράφεται πίσω στον host για έλεγχο αποτελεσμάτων και σύγκριση με αυτών της CPU του συστήματός. Τονίζεται ότι ο αριθμός νημάτων που θα δημιουργηθούν σε κάθε εκτέλεση του μετροπρογράμματος θα είναι ίσος με τον αριθμό των στοιχείων του κάθε διανύσματος. Σε όλες τις περιπτώσεις διεξήχθη το ίδιο πείραμα 128 φορές, και οι τιμές με βάση τις οποίες έγιναν τα διαγράμματα που θα παρουσιαστούν παρακάτω, είναι οι μέσοι όροι αυτών των μετρήσεων.

Καθώς βέβαια οι πυρήνες δεν θα περιλαμβάνουν μόνο τις αριθμητικές εντολές, οι οποίες μελετώνται, αλλά και μεταφορές από και προς την καθολική μνήμη (οι οποίες είναι ιδιαίτερα χρονοβόρες σε σχέση με άλλες εντολές), κάποιους ελέγχους (για παράδειγμα στους βρόχους εκτέλεσης των πράξεων) και λοιπές άλλες λειτουργίες, δεν αναμένεται να μετρηθεί η μέγιστη διαφημιζόμενη ταχύτητα αλλά κάποια λίγο μικρότερη, παρόλο που η μεγάλη πλειοψηφία των εντολών των πυρηνών θα είναι οι «ωφέλιμες» πράξεις.

5.2.1 Πρώτο πείραμα: Εισαγωγή αριθμού επαναλήψεων από το χρήστη κατά την εκτέλεση

Σε αυτό το πείραμα, ο αριθμός των πράξεων που θα εκτελεστούν ανά δυάδα στοιχείων που θα φέρει το κάθε νήμα από την καθολική μνήμη, καθορίζεται από το χρήστη ακριβώς πριν την εκτέλεση του πυρήνα. Χρησιμοποιούνται διανύσματα μεγέθους 32 MB το καθένα (δηλαδή μεγέθους 8 M `float`) και τα αποτελέσματα για διάφορους αριθμούς επαναλήψεων φαίνονται στο Σχήμα 5-1.



Σχήμα 5-1 Μέτρηση ταχύτητας εκτέλεσης πράξεων (πρόσθεσης, πολλαπλασιασμού, πρόσθεσης-πολλαπλασιασμού και πρόσθεσης-πολλαπλασιασμού σε συνδυασμό με πολλαπλασιασμό) συναρτήσει του αριθμού των επαναλήψεων, ο οποίος εισάγεται από το χρήστη κατά την εκτέλεση του μικροπρογράμματος.

Σε αυτό το πρώτο πείραμα, είναι δυνατή η εξαγωγή των πρώτων συμπερασμάτων για την συγκριτική ρυθμοαπόδοση των εντολών. Βλέπουμε ότι στη MAD-MUL περίπτωση παρατηρείται η μέγιστη ταχύτητα, ακολουθούμενη από την MAD, ενώ σε χαμηλότερες ταχύτητες κυμαίνονται οι MUL και ADD, με την MUL να είναι λίγο πιο γρήγορη. Κάτι τέτοιο είναι αναμενόμενο, καθώς η εντολή MAD έχει την ίδια διάρκεια (σε κύκλους μηχανής) με τις ADD και MUL, αλλά μετράει ως δυο εντολές καθώς πραγματοποιεί και πολλαπλασιασμό και πρόσθεση στους ίδιους κύκλους μηχανής. Επίσης, σύμφωνα με τις τεχνικές εκθέσεις της nVidia για τις κάρτες γραφικών που ακολουθούν την αρχιτεκτονική G80, είναι εφικτή η διπλή-έκδοση μιας εντολής MAD με μια MUL, οπότε είναι επόμενο αυτό το μείγμα εντολών να είναι το καλύτερο δυνατό. Αναμένεται σε όλες οι υπόλοιπες μετρήσεις να επιβεβαιωθεί η παραπάνω ανάλυση σχετικά με τη συγκριτική ταχύτητα των προαναφερθείσων εντολών.

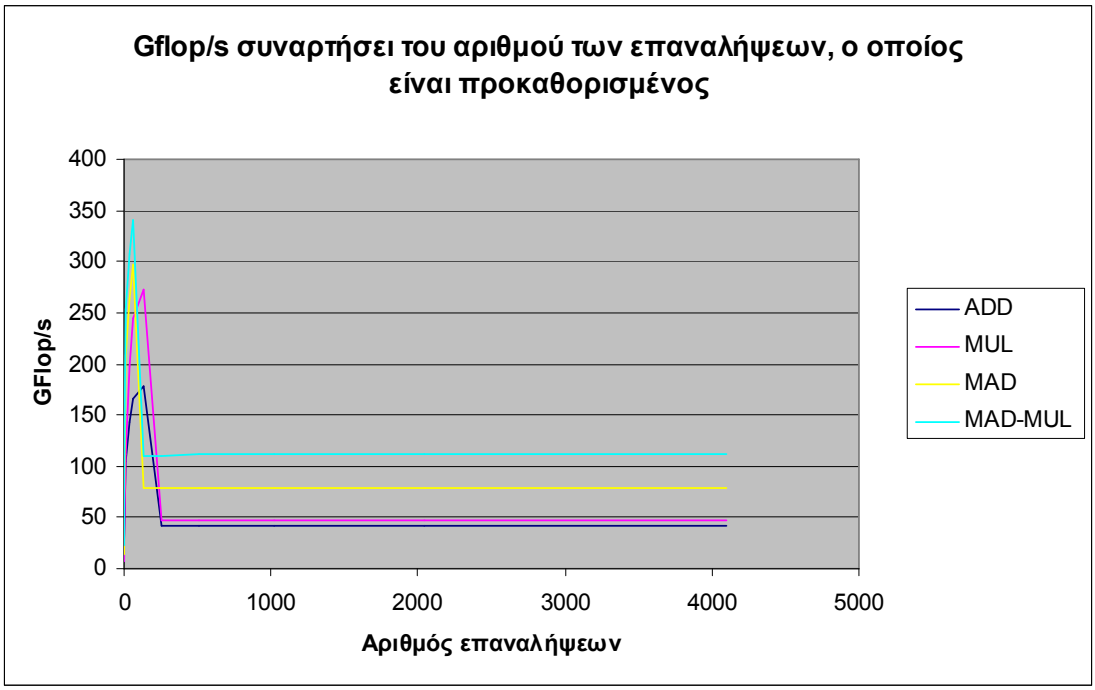
Όσον αφορά την ταχύτητα σαν απόλυτο αριθμό, οι μετρήσεις δείχνουν ότι βρίσκεται ακόμα αρκετά μακριά το θεωρητικό μέγιστο των 550 GFlop/s. Βλέποντας τον κώδικα σε γλώσσα μηχανής αυτής της περίπτωσης για όλες τις εντολές, παρατηρήθηκε ότι πολύ μεγάλο ποσοστό των εντολών γλώσσας μηχανής ήταν αφιερωμένο στις συγκρίσεις για το βρόχο επαναλήψεων της πράξης, και τελικά οι «ωφέλιμες» πράξεις ήταν ένα μικρό κλάσμα των συνολικών, ενώ δεν έγινε κανένα ξετύλιγμα του βρόχου από τον μεταγλωττιστή αυτόματα (κάτι που θα είχε σα συνέπεια τη μείωση των εντολών ελέγχου ροής του μικροπρογράμματος), καθώς όταν γίνεται η μεταγλώττιση, δεν είναι γνωστός ο αριθμός των επαναλήψεων.

Παρατηρείται ότι, μετά τις 32 επαναλήψεις, η ταχύτητα εκτέλεσης των πράξεων φτάνει σε ένα σημείο κορεσμού. Από εκεί και πέρα, διατηρείται πρακτικά σταθερή. Μέχρι εκείνο το σημείο οι αριθμητικές εντολές του πυρήνα ήταν ένα πολύ μικρότερο

κλάσμα των συνολικών εντολών και ειδικότερα ο χρόνος εκτέλεσής τους ήταν πολύ μικρότερος από το χρόνο των μεταφορών από και προς την καθολική μνήμη. Οι τελευταίες κυριαρχούσαν μέχρι τότε ως προς το ποσοστό του συνολικού χρόνου εκτέλεσης που καταλάμβαναν. Μετά από κάποιο σημείο όμως, η ταχύτητα σταθεροποιείται καθώς πλέον κυριαρχούν οι αριθμητικές πράξεις.

5.2.2 Δεύτερο πείραμα: Προκαθορισμένος αριθμός επαναλήψεων των πράξεων

Σε αυτό το πείραμα, θα γίνει ότι ακριβώς και στο πρώτο με τη διαφορά ότι ο αριθμός των επαναλήψεων των πράξεων που θα εκτελεί ο κάθε πυρήνας, θα είναι προκαθορισμένος. Στο Σχήμα 5-2 υπάρχουν τα αποτελέσματα των εν λόγω πειραμάτων.



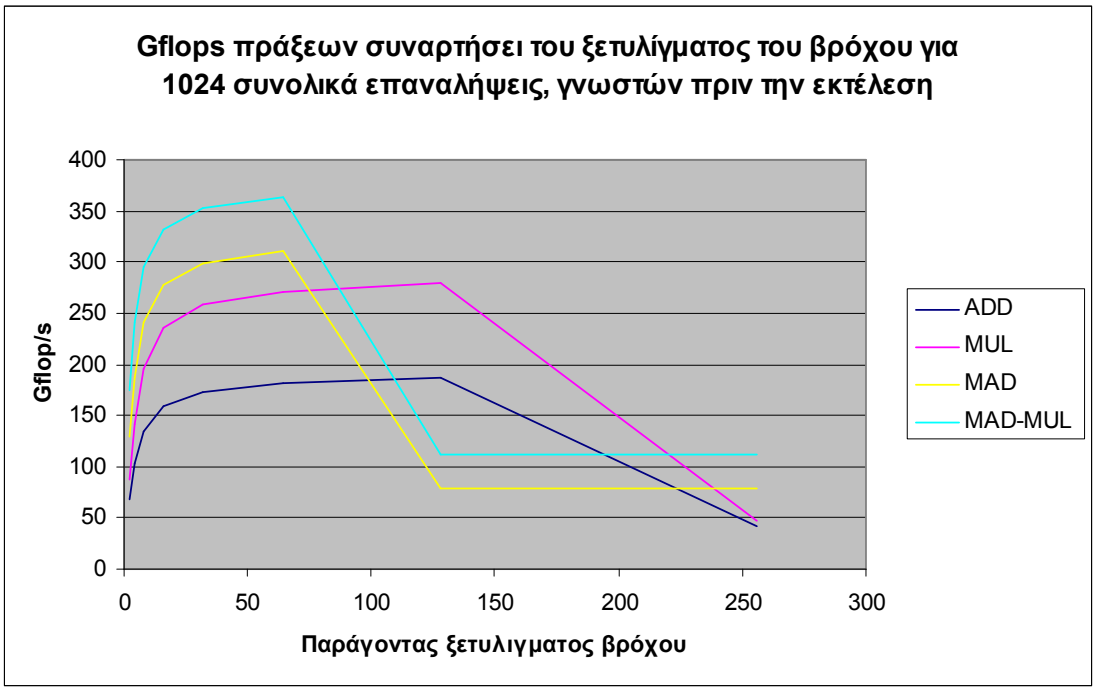
Σχήμα 5-2 Μέτρηση ταχύτητας εκτέλεσης πράξεων (πρόσθεσης, πολλαπλασιασμού, πρόσθεσης-πολλαπλασιασμού και πρόσθεσης-πολλαπλασιασμού σε συνδυασμό με πολλαπλασιασμό) συναρτήσει του αριθμού των επαναλήψεων, ο οποίος είναι προκαθορισμένος και γνωστός πριν τη μεταγλώττιση του μικροπρογράμματος.

Στο διάγραμμα του Σχήματος 5-2 επιβεβαιώνονται τα συμπεράσματα για τη συγκριτική ρυθμοαπόδοση των διαφορετικών εντολών, καθώς η MAD-MUL περίπτωση παραμένει η πιο γρήγορη, ακολουθούμενη κατά σειρά από τις MAD, MUL και ADD. Σε αυτήν την περίπτωση όμως παρατηρείται ένα μέγιστο στις 64 επαναλήψεις για τις περιπτώσεις των MAD και MAD-MUL και στις 128 επαναλήψεις για τις άλλες δύο περιπτώσεις. Βλέποντας τον κώδικα σε γλώσσα μηχανής, παρατηρήθηκε ότι μέχρι τα σημεία που επιτυγχάνεται αυτό το μέγιστο, ο μεταγλωττιστής ξετυλίγει ολόκληρο το βρόχο των επαναλήψεων, οπότε εξαλείφονται όλες οι εντολές ελέγχου που είχαμε στο προηγούμενο πείραμα. Όμως, φαίνεται να υπάρχει κάποιο όριο στο πόσες εντολές μπορεί να περιέχει ένας βρόχος για να τον ξετυλίξει ολοκληρωτικά ο μεταγλωττιστής. Αυτό το όριο είναι οι 64 επαναλήψεις για τις περιπτώσεις των MAD-MUL και MAD και οι 128 για τις άλλες δύο περιπτώσεις. Αυτός είναι και ο λόγος που η ταχύτητα εκτέλεσης των πυρήνων επιστρέφει στα χαμηλά επίπεδα που είχαν παρατηρηθεί και κατά το πρώτο πείραμα.

Τέλος, αν απομονωθεί η ταχύτητα ως απόλυτη τιμή, παρατηρείται ότι στην περίπτωση MAD-MUL (όπου έχουμε και την μεγαλύτερη ταχύτητα εκτέλεσης), φτάνει σχεδόν τα 340 GFlop/s, πλησιάζει, δηλαδή, αρκετά στο θεωρητικό μέγιστο. Σε επόμενο πείραμα θα διερευνηθεί κατά πόσο τελικά είναι δυνατόν να προσεγγιστεί το θεωρητικό μέγιστο.

5.2.3 Τρίτο πείραμα: Εύρεση μέγιστου επιτρεπτού ξετυλίγματος βρόχου

Κατά τη διάρκεια του προηγούμενου πειράματος υπήρξαν ενδείξεις ότι ο μεταγλωττιστής επιτρέπει μέχρι ένα συγκεκριμένο ξετύλιγμα του βρόχου όταν ο συνολικός αριθμός επαναλήψεων είναι εκ των προτέρων γνωστός. Διεξήχθη ένα τρίτο πείραμα για να επιβεβαιωθεί κάτι τέτοιο. Χρησιμοποιήθηκαν, όπως και στα προηγούμενα πειράματα, μεγέθη διανυσμάτων ίσα με 32 MB (8 M `float`) και καθορισμένο αριθμό επαναλήψεων αρκετά μεγάλο και ίσο με 1024. Αυτό έγινε ώστε να είναι σίγουρο ότι το μεγαλύτερο μέρος του συνολικού χρόνου είναι η εκτέλεση των «ωφέλιμων»-μετρούμενων πράξεων. Δοκιμάστηκε για κάθε μια από τις τέσσερις περιπτώσεις πράξεων, διάφοροι παράγοντες ξετυλίγματος του βρόχου, οι οποίοι κυμαίνονται από δυο (υποδιπλασιασμός τελικά δηλαδή των εντολών ελέγχου και δυο πράξεις ανά βήμα με το ξετύλιγμα) μέχρι 256 (δηλαδή τέσσερις φορές εκτέλεση εντολών ελέγχου, αφού ο βρόχος θα έχει τελικά τέσσερα βήματα από 256 πράξεις στο κάθε ένα από αυτά). Στο Σχήμα 5-3 φαίνονται τα αποτελέσματά της παραπάνω διαδικασίας.



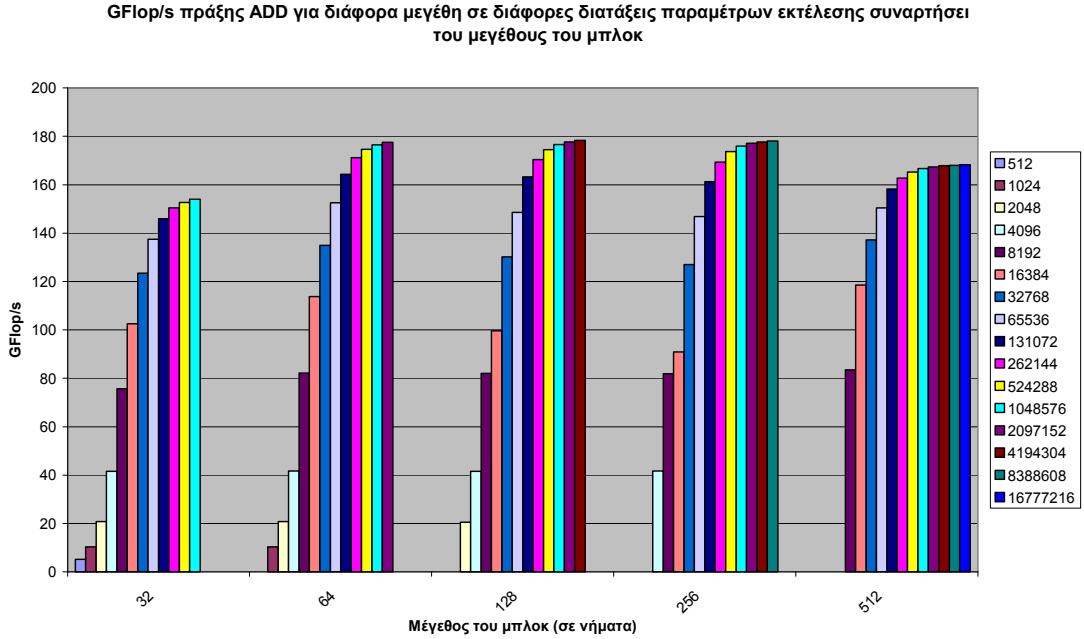
Σχήμα 5-3 Ταχύτητα εκτέλεσης για 1024 πράξεις σε κάθε πυρήνα (αριθμός προκαθορισμένος πριν τη μεταγλώττιση του μικροπρογράμματος) συναρτήσει του παράγοντα ξετυλίγματος βρόχου που επιχειρήθηκε.

Αυτό το πείραμα επιβεβαιώνει τις ενδείξεις ότι για βρόχους με πράξεις ADD ή MUL, είναι δυνατόν να ξετυλιχτούν μέχρι και 128 φορές, ενώ για τις περιπτώσεις των MAD και MAD-MUL ο μέγιστος παράγοντας ξετυλίγματος βρόχου που επιτρέπει ο μεταγλωττιστής είναι 64 φορές. Η διαφορά στον μέγιστο παράγοντα ξετυλίγματος βρόχου οφείλεται στη διαφορά όγκου λειτουργιών σε κάθε περίπτωση, με τις πιο απλές περι-

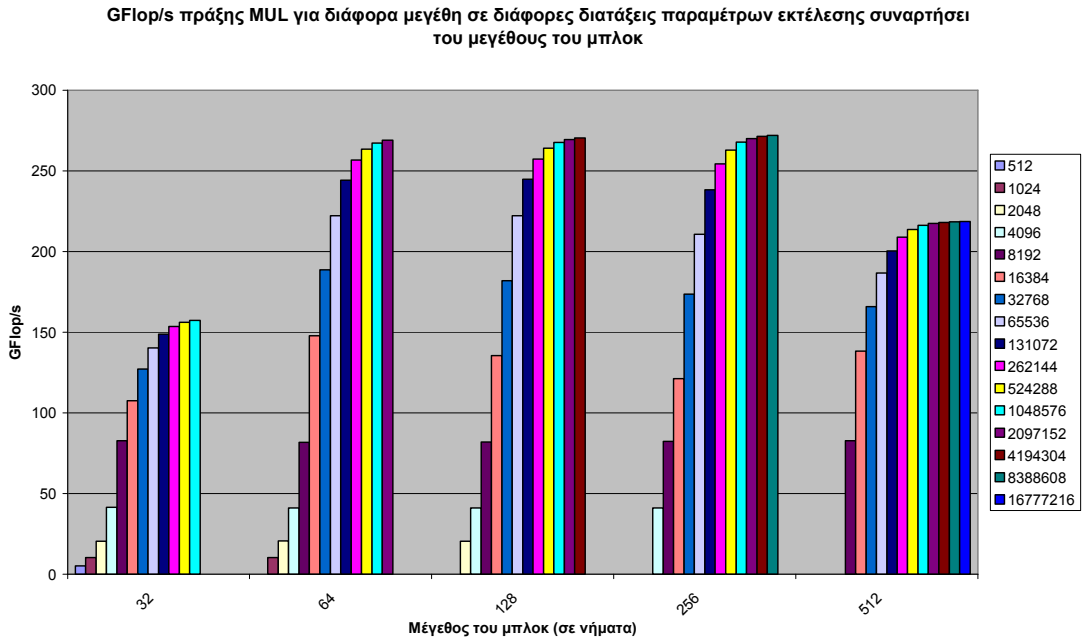
πτώσεις (MUL και ADD) να επιτρέπουν μεγαλύτερο ξετύλιγμα, και με τις πιο σύνθετες (MAD και MAD-MUL) να επιτρέπουν πιο μικρό. Όταν ο παράγοντας ξετυλίγματος ξεπερνά τα όρια που αναφέρθηκαν παραπάνω, ο μεταγλωττιστής αγνοεί την εντολή για περαιτέρω ξετύλιγμα, και τα αποτελέσματα είναι όμοια με εκείνα του πρώτου πειράματος (στ οποίο δεν συνέβαινε ξετύλιγμα). Τέλος, παρατηρήθηκε ότι σε αυτήν την περίπτωση και στην πράξη MAD-MUL η μέση ταχύτητα έφτασε τα 360 GFlop/s, αφού σε σχέση με το δεύτερο πείραμα που μελετήθηκε παραπάνω, αυξήθηκαν οι αριθμητικές εντολές του πυρήνα. Συνέπεια αυτού του γεγονότος ήταν το ποσοστό του χρόνου των «ωφέλιμων» εντολών ως προς τον συνολικό χρόνο εκτέλεσης της εφαρμογής (ο οποίος περιλαμβάνει και τις μεταφορές μεταξύ καθολικής μνήμης και πυρήνων επεξεργασίας και άλλες λειτουργίες αρχικοποιήσεων κάθε νήματος κτλ.) να αυξηθεί, και άρα η μέση ταχύτητα εκτέλεσης να αυξηθεί περαιτέρω. Ορμώμενοι από τα αποτελέσματά σε αυτό το πείραμα, και κρατώντας σαν πιο χρήσιμα συμπεράσματα τους μέγιστους παράγοντες ξετυλίγματος βρόχου, θα γίνει προσπάθεια στα επόμενα πειράματα να ξεκαθαρίσουν και άλλοι τομείς που είναι δυνατόν να επηρεάζουν την επίδοση αυτού του μετροπρογράμματος.

5.2.4 Τέταρτο πείραμα: Επίδραση των διαφορετικών οργανώσεων των νημάτων σε μπλοκ

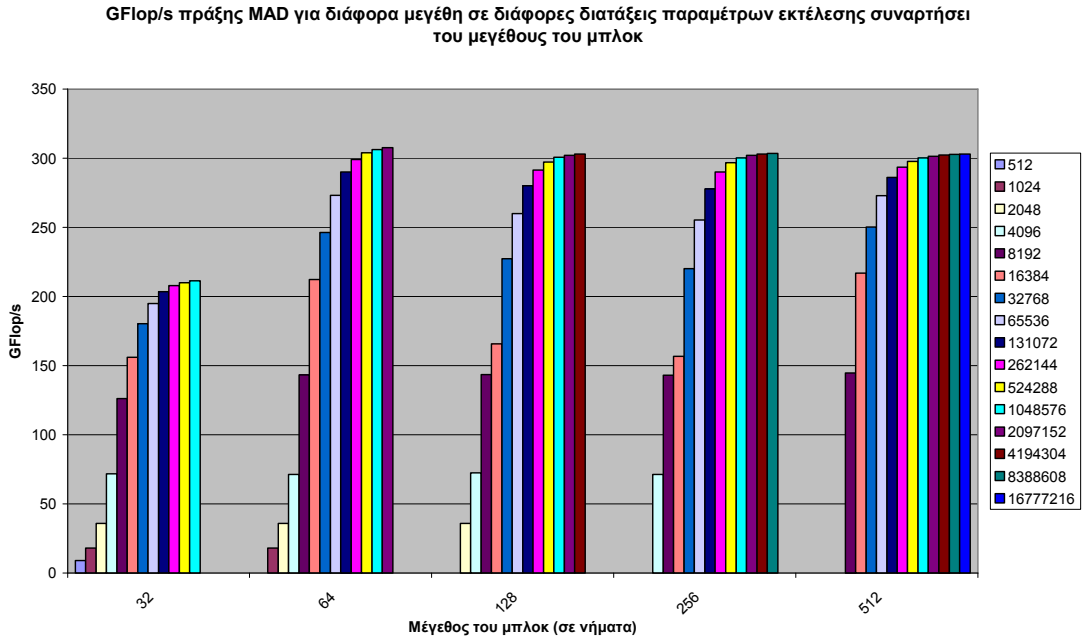
Σε αυτό το πείραμα δοκιμάστηκαν όλοι οι πιθανοί συνδυασμοί και οργανώσεις παραμέτρων εκτέλεσης, για πολλά διαφορετικά μεγέθη διανυσμάτων, με σκοπό την εύρεση των πιο ευνοϊκών για την επίδοση του μετροπρογράμματος. Όλες οι μετρήσεις αφορούν 128 επαναλήψεις (128 πράξεις ανά νήμα) και έχουν ξετυλιχτεί κατά το δυνατόν οι βρόχοι των επαναλήψεων (128 ο παράγοντας ξετυλίγματος στις περιπτώσεις των ADD και MUL και 64 στις άλλες δύο περιπτώσεις). Τα αντίστοιχα διαγράμματα για κάθε μια από τις πράξεις που μελετώνται σε αυτή την ενότητα παρουσιάζονται στα Σχήματα 5-4 έως 5-7:



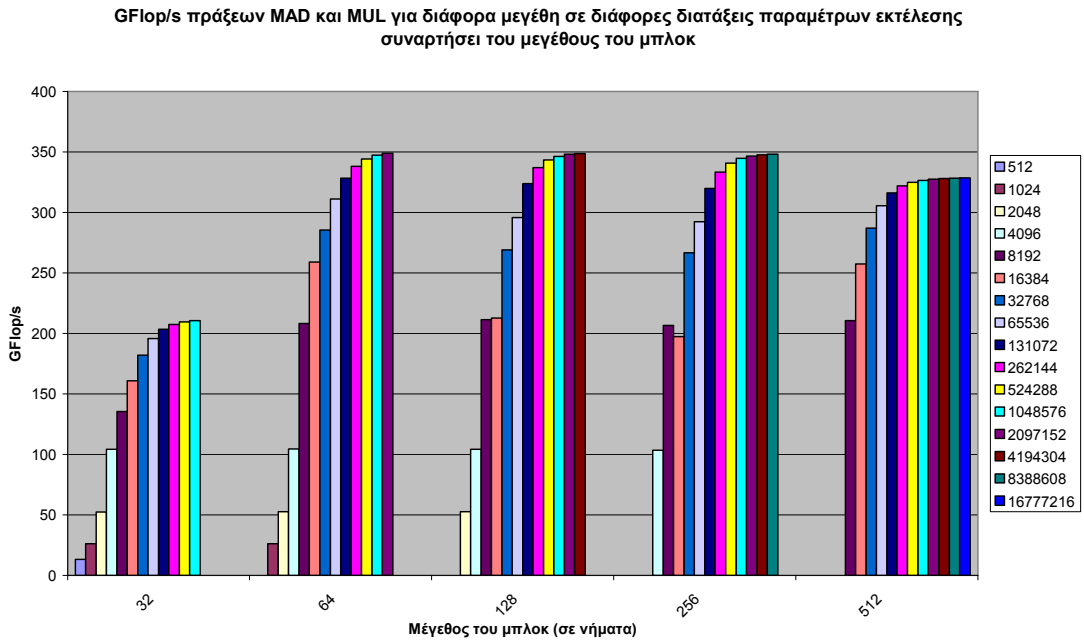
Σχήμα 5-4 Ταχύτητα εκτέλεσης πράξης ADD για διάφορα μεγέθη διανύσματος, σε όλες τις πιθανές οργανώσεις των παραμέτρων εκτέλεσης, συναρτήσει του μεγέθους του μπλοκ της κάθε οργάνωσης.



Σχήμα 5-5 Ταχύτητα εκτέλεσης πράξης MUL για διάφορα μεγέθη διανύσματος, σε όλες τις πιθανές οργανώσεις των παραμέτρων εκτέλεσης, συναρτήσει του μεγέθους του μπλοκ της κάθε οργάνωσης.



Σχήμα 5-6 Ταχύτητα εκτέλεσης πράξης MAD για διάφορα μεγέθη διανύσματος, σε όλες τις πιθανές οργανώσεις των παραμέτρων εκτέλεσης, συναρτήσει του μεγέθους του μπλοκ της κάθε οργάνωσης.



Σχήμα 5-7 Ταχύτητα εκτέλεσης πράξεων MAD-MUL για διάφορα μεγέθη διανύσματος, σε όλες τις πιθανές οργανώσεις των παραμέτρων εκτέλεσης, συναρτήσει του μεγέθους του μπλοκ της κάθε οργάνωσης.

Θα γίνουν αρχικά ορισμένες παρατηρήσεις οι οποίες θα κάνουν πιο εύκολη την ανάγνωση των Σχημάτων 5-4 έως 5-7, πριν επιχειρηθεί η ανάλυση τους και η εξαγωγή συμπερασμάτων. Καταρχήν, αναφέρεται ότι κάθε διαφορετικό χρώμα μπάρας αντιστοιχεί σε διαφορετικό μέγεθος του προβλήματος (διαφορετικό αριθμός `float` ανά διάνυσμα δηλαδή). Σε κάθε ομάδα που καθορίζεται από το μέγεθος του μπλοκ, υπάρχει για (σχεδόν) κάθε διαφορετικό πρόβλημα η μέση ταχύτητα που επιτυγχάνεται κατά την εκτέλεσή του, για μια συγκεκριμένη οργάνωση των παραμέτρων εκτέλεσης. Το μέγεθος του μπλοκ φαίνεται από την ομάδα στην οποία ανήκει η συγκεκριμένη μπάρα, ενώ για να βρει κανείς τον συνολικό αριθμό των μπλοκ του πλέγματος της συγκεκριμένης οργάνωσης αυτού του πειράματος πρέπει να διαιρέσει το μέγεθος του προβλήματος με το μέγεθος του μπλοκ. Για παράδειγμα, για το πρόβλημα των 1 M `float`, και για μέγεθος μπλοκ 128, χρειάστηκαν συνολικά 8192 μπλοκ, για να αντιστοιχεί ένα νήμα σε κάθε στοιχείο του διανύσματος.

Καταρχήν, παρατηρείται μια τάση, όσο μεγαλώνει το μέγεθος του προβλήματος, τόσο να αυξάνεται και η μέση ταχύτητα εκτέλεσης. Αυτό οφείλεται στο γεγονός ότι, όσο μεγαλώνει το μέγεθος του προβλήματός (άρα και ο αριθμός των συνολικών πράξεων που πρέπει να εκτελεστούν), τόσο μικρότερο ποσοστό του συνολικού χρόνου καταλαμβάνουν οι αρχικοποιήσεις της εκτέλεσής του προγράμματός, και όλες οι άλλες λειτουργίες που συμβαίνουν ούτως ή άλλως κατά την έναρξη της εκτέλεσης ενός πυρήνα. Αποτέλεσμα είναι, όσο μεγαλώνει το μέγεθος του προβλήματος (αλλά και ο αριθμός των επαναλήψεων σε αντίστοιχη περίπτωση), τόσο μεγαλώνει το ποσοστό του συνολικού χρόνου που καταλαμβάνουν οι «ωφέλιμες» πράξεις, άρα και η μέση ταχύτητα. Τονίζουμε ότι οι «ωφέλιμες» εντολές είναι αυτές των πράξεων και κατά τον υπολογισμό της ταχύτητας προσμετρώνται αυτές μόνο και διαιρούνται δια τον συνολικό χρόνο εκτέλεσης του πυρήνα. Ο σημαντικότερος λόγος για τον οποίο όμως παρατηρείται αυτή η αύξηση της ταχύτητας με αύξηση των νημάτων εκτέλεσης, είναι

ο εξής: Διαθέτοντας πολλά νήματα εκτέλεσης, μπορούν να κρυφτούν καλύτερα οι καθυστερήσεις προσβάσεις στην καθολική μνήμη κατά την ανάγνωση των αριθμών που θα συμμετάσχουν στις πράξεις. Επιπλέον, έχει αναφερθεί και στην παρουσίαση της αρχιτεκτονικής G80, η ανάγκη ύπαρξης μεγάλου αριθμού νημάτων για την πλήρη εκμετάλλευσή της, ενώ η συγκεκριμένη αρχιτεκτονική δείχνει να κλιμακώνει πολύ καλά σε αυξανόμενα μεγέθη υπολογιστικών προβλημάτων.

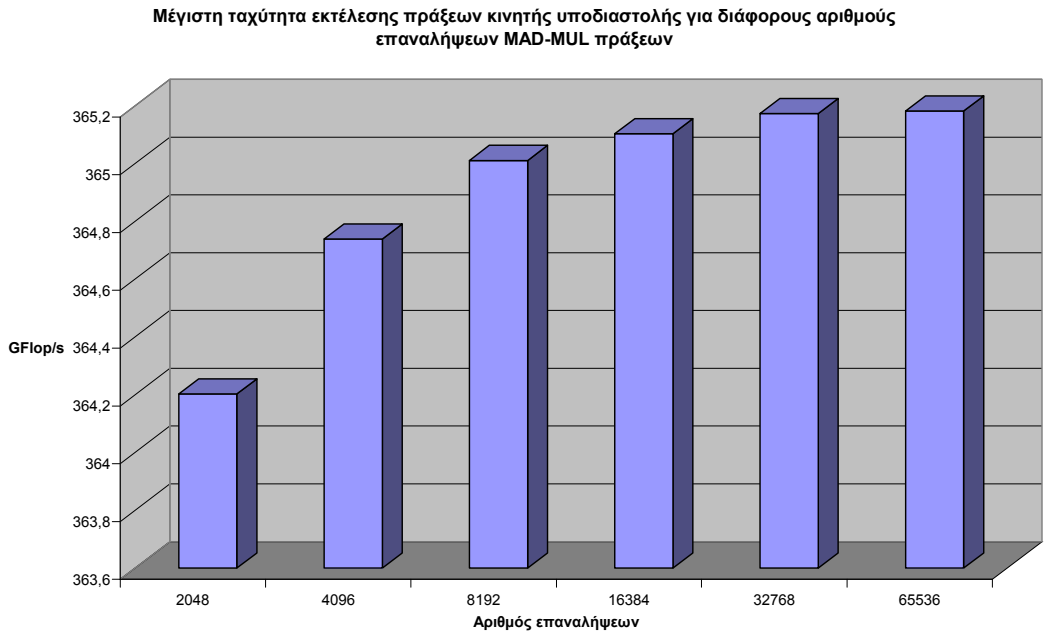
Παρατηρώντας τις μετρήσεις για μεγάλα μεγέθη (άρα και αρκετά μεγάλο αριθμό μπλοκ, ώστε η χρησιμοποίηση να παίζει ρόλο), βλέπουμε ότι οι παράμετροι εκτέλεσης που έχουν σα μέγεθος μπλοκ 64, 128 ή 256 νήματα, παρουσιάζουν πρακτικά τις ίδιες ταχύτητες, ενώ οι περιπτώσεις των 32 και των 512 νημάτων ανά μπλοκ εμφανίζουν μειωμένες επιδόσεις. Αυτό εξηγείται αν λάβουμε υπ'όψιν τους περιορισμούς που θέτει κάθε SM όσον αφορά τον αριθμό μπλοκ και νημάτων που μπορεί να «φιλοξενεί» ανά πάσα στιγμή. Αυτά ανέρχονται σε 8 διαφορετικά μπλοκ και στα 768 νήματα (ανεξάρτητα σε πόσα διαφορετικά μπλοκ ανήκουν) συνολικά. Σημειώνεται ότι, σε αυτόν τον πολύ απλό πυρήνα, για οποιαδήποτε από τις πράξεις που μελετώνται εδώ, η χρησιμοποίηση εξαρτάται μόνο από το αν «χωράει» το SM το μέγιστο αριθμό νημάτων που μπορεί να υποδεχτεί, καθώς οι απαιτήσεις του κάθε νήματος είναι πολύ μικρές για να αποβούν περιοριστικές, ενώ επιπλέον δε γίνεται καθόλου χρήση της διαμοιραζόμενης μνήμης. Δεδομένου ότι υπάρχουν πολλά μπλοκ προς δρομολόγηση στα SMs(καθώς μιλάμε για μεγάλα σε μέγεθος προβλήματα με μεγάλο αριθμό συνολικών μπλοκ στο πλέγμα εκτέλεσής τους), μια οργάνωση με 32 νήματα ανά μπλοκ, περιορίζει τα συνολικά νήματα σε 256 έναντι των 768 που επιτρέπονται συνολικά, πράγμα που σημαίνει ότι η μέγιστη χρησιμοποίηση περιορίζεται στο 33,3%. Αποτέλεσμα αυτού, η αύξηση των «φουρνιών» που θα χρειαστούν συνολικά για την εκτέλεση του μικροπρογράμματός και άρα η μικρή σχετική ταχύτητα καθώς μένουν αναξιοποίητοι πόροι που σε άλλες περιπτώσεις οργανώσεων των παραμέτρων εκτέλεσης θα χρησιμοποιούνταν. Επίσης, στην περίπτωση που το μέγεθος του μπλοκ είναι ίσο με 512, το ποσοστό χρησιμοποίησης φτάνει το 66,7%, κάτι που σημαίνει ότι μένουν πόροι αναξιοποίητη. Αυτό εξηγεί τον λόγο που υπερισχύουν οι εκδόσεις με μεγέθη μπλοκ τέτοια ώστε να χωρούν 768 νήματα ταυτόχρονα σε ένα SM μια οποιαδήποτε στιγμή. Επιπροσθέτως, η έλλειψη ανάγκης για συγχρονισμό μεταξύ νημάτων που ανήκουν σε διαφορετικά μπλοκ, σε συνδυασμό με την απλότητα του κώδικα του κάθε νήματος και την έλλειψη οποιασδήποτε σχέσης μεταξύ διαφορετικών νημάτων, κάνει όλους τους συνδυασμούς παραμέτρων εκτέλεσης που πετυχαίνουν ποσοστό χρησιμοποίηση 100% να έχουν την ίδια σχεδόν ταχύτητα. Συμπεραίνουμε λοιπόν ότι θα πρέπει να αποφεύγονται συνδυασμοί παραμέτρων εκτέλεσης που δεν εξασφαλίζουν τη μέγιστη δυνατή χρησιμοποίηση των πόρων των SMs. Αυτό μπορεί να αποφευχθεί με τη χρήση του εργαλείου που προσφέρει το περιβάλλον του CUDA, τον Υπολογιστή Χρησιμοποίησης.

Κοιτώντας τις μετρήσεις που λάβαμε για μέσα προς μικρά προβλήματα (και πιο συγκεκριμένα για τις περιπτώσεις που το μέγεθος του διανύσματος είναι 16K και 32K `float`), παρατηρείται κάτι που, χωρίς προσεκτική παρατήρηση, ίσως ξενίσει τον αναγνώστη. Εδώ, οι μέγιστες ταχύτητες παρατηρούνται για μέγεθος μπλοκ ίσο με 64 και 512, παρουσιάζεται δηλαδή ένα αποτέλεσμα που έρχεται σε αντίθεση με όσα ειπώθηκαν για μεγαλύτερα προβλήματα, καθώς η χρησιμοποίηση σε αυτές τις περιπτώσεις δεν είναι 100% αλλά 66,7%. Αυτό παρουσιάστηκε και στην περίπτωση της αναγωγής και έχει να κάνει με τον αριθμό των «φουρνιών» στις οποίες χωρίζονται τα συνολικά μπλοκ της εφαρμογής (βλ. ενότητα 5.3.3).

5.2.5 Πέμπτο πείραμα: Προσπάθεια προσέγγισης θεωρητικά μέγιστης ταχύτητας εκτέλεσης πράξεων

Με χρήση των συμπερασμάτων από τα προηγούμενα πειράματα θα επιχειρηθεί να βρεθεί το όριο στην ταχύτητα εκτέλεσης πράξεων στην διαθέσιμη κάρτα γραφικών. Θα χρησιμοποιηθεί το μετροπρόγραμμα του συνδυασμού MAD-MUL, πράξη στην οποία παρατηρήσαμε τις μεγαλύτερες ταχύτητες. Οι μετρήσεις θα αφορούν όσο το δυνατόν μεγαλύτερο μέγεθος προβλήματος (8M `float`), και όσο το δυνατόν μεγαλύτερο αριθμό επαναλήψεων, με το μέγιστο επιτρεπτό ξετύλιγμα βρόχων (που όπως είδαμε σε παραπάνω πείραμα είναι το 64). Πρέπει πάντοτε να είναι υπ'όψη του αναγνώστη, πως ένα μέρος του χρόνου που μετράμε αποτελεί η έναρξη της εκτέλεσης του πυρήνα (κάτι αναπόφευκτο σε οποιαδήποτε εκτέλεση πυρήνα, καθώς περιλαμβάνει τη μεταφορά των εντολών που θα εκτελεστούν στην συσκευή και λοιπές αρχικοποιήσεις, ενώ πάντα υπάρχει ένα overhead κατά την έναρξη και τερματισμό του πυρήνα). Ακόμα, ο πυρήνας περιλαμβάνει έναν αριθμό εντολών, που δεν αποτελούν «ωφέλιμο» (δηλαδή μετρούμενο) φορτίο, προκειμένου να γίνεται ομαλά η εκτέλεση των υπόλοιπων εντολών του πυρήνα[5] ενώ σημαντικό ρόλο παίζουν οι «αργές» εντολές μεταφοράς από και προς την καθολική μνήμη. Περιλαμβάνοντας όμως στις μετρήσεις μεγάλο αριθμό νημάτων, μπορεί να «κρυφτεί» μέρος της καθυστέρησης προσπελάσεων της καθολικής μνήμης, ενώ οι λοιπές αναπόφευκτες αλλά σταθερές καθυστερήσεις (overhead) καταλαμβάνουν πολύ μικρότερο ποσοστό του συνολικού χρόνου. Συνέπεια αυτού του γεγονότος είναι ότι, μπορεί να ειπωθεί με βεβαιότητα ότι τελικά θα ψηλαφιστούν οριακά οι μέγιστες δυνατότητες της κάρτας γραφικών σε υπολογιστικές δυνατότητες, χωρίς όμως να φτάσουν στο απόλυτο μέγιστο για λόγους που εξηγήθηκαν παραπάνω. Στο Σχήμα 5-8 φαίνονται οι μετρήσεις αυτού του πειράματος.

[5] Στην εξεταζόμενη περίπτωση υπάρχουν εντολές υπολογισμού των διευθύνσεων της καθολικής μνήμης τις οποίες θα προσπελάσει το κάθε νήμα προκειμένου να λάβει τα δεδομένα εισόδου του, και εντολές ελέγχου ροής καθώς ο αριθμός των συνολικών επαναλήψεων θα είναι πολύ μεγαλύτερος του μέγιστου επιτρεπτού παράγοντα ξετυλίγματος του βρόχου



Σχήμα 5-8 Μέτρηση ταχύτητας μικροπρογράμματος για μέγεθος διανυσμάτων ίσο με 8 M `float` και για διάφορους αριθμούς πράξεων ανά νήμα. Η διάταξη παραμέτρων εκτέλεσης που χρησιμοποιήθηκε ήταν 32K μπλοκ επί 256 νήματα/μπλοκ και ο παράγοντας ξετυλίγματος των βρόχων ήταν 64.

Η τελική μέγιστη ταχύτητα που μετρήθηκε ήταν κάτι παραπάνω από 365 GFlop/s, ταχύτητα μικρότερη από την θεωρητική μέγιστη αλλά τεράστια για μια πλατφόρμα διαθέσιμη στο ευρύ κοινό σε σχετικά χαμηλή τιμή για υπολογιστικό σύστημα με τέτοιες δυνατότητες. Τα αποτελέσματα κρίνονται ιδιαίτερα εντυπωσιακά, αν ληφθεί υπ'όψη ότι προτεραιότητα της αρχιτεκτονικής G80 είναι οι εφαρμογές γραφικών και όχι οι εφαρμογές γενικού σκοπού.

5.2.6 Συμπεράσματα από τη μελέτη του μετροπρογράμματος μέτρησης ταχύτητας εκτέλεσης πράξεων κινητής υποδιαστολής

Σε αυτήν την ενότητα διευρευνήθη το κατά πόσο μπορεί να αντεπεξέλθει η αρχιτεκτονική G80 σε προβλήματα ιδιαίτερα απαιτητικά σε υπολογισμούς αριθμών κινητής υποδιαστολής. Τέτοια προβλήματα είναι διαδεδομένα σε πάρα πολλούς τομείς, και κυρίως στον επιστημονικό και ερευνητικό. Από τα παραπάνω πειράματα μπορεί να εξαχθεί το συμπέρασμα ότι είναι δυνατό να επιτευχθούν πολύ μεγαλύτερες ταχύτητες εκτέλεσης πράξεων συγκριτικά με επεξεργαστές του εμπορίου ή πιο παραδοσιακές πλατφόρμες ανάπτυξης εφαρμογών. Η χρησιμότητα των καρτών γραφικών που υλοποιούν την αρχιτεκτονική G80 μπορεί να αποδειχθεί εξαιρετικά μεγάλη, αν οι προγραμματιστές επενδύσουν στο να μελετήσουν τις δυνατότητες της και, κυρίως, κάποιες σημαντικές λεπτομέρειες που μπορούν να έχουν σημαντικό αντίκτυπο στην επίδοση. Ενδεικτικό αυτού του γεγονότος είναι η χρησιμοποίηση της μελετούμενης αρχιτεκτονικής σε απαιτητικές, σε αριθμητικούς υπολογισμούς, εφαρμογές από την δοκιμαστική της έκδοση ακόμα.

Τα αρχικά αποτελέσματα ήταν απογοητευτικά όταν χρησιμοποιήθηκαν μεγάλοι βρόχοι αριθμητικών πράξεων με εισαγωγή του μεγέθους του βρόχου από το χρήστη κατά

τη στιγμή της εκτέλεσης. Όμως, ότι αν είναι γνωστό εκ των προτέρων το μέγεθος των βρόχων, μπορεί να ακολουθηθεί μια στρατηγική ξετυλίγματος των βρόχων (που μπορεί να διαφέρει ως προς τον παράγοντα ξετυλίγματος ανάλογα με το μέγεθος σε εντολές και υπολογιστικές απαιτήσεις του κάθε βήματος του βρόχου), πετυχαίνοντας πολύ μεγάλη επιτάχυνση στους απαιτούμενους αριθμητικούς υπολογισμούς.

Ακόμη, είναι σημαντικό να επιλεγούν οι καλύτερες δυνατές παράμετροι εκτέλεσης, ακόμα και σε έναν τόσο απλό αλγόριθμο, ώστε να μην αφήνονται αναξιοποίητοι πόροι. Επιβεβαιώθηκε, επίσης, ότι η αρχιτεκτονική G80 επιτρέπει την πολύ καλή κλιμάκωση σε μεγάλα μεγέθη προβλημάτων, ενώ η χρήση πολλών νημάτων είναι πάντα ευεργετική ως προς την μέγιστη μέση ταχύτητα αφού καθίσταται εφικτό το «κρύψιμο» των αργών προσπελάσεων στην καθολική μνήμη.

Τέλος, οι μεγαλύτερες ταχύτητες εμφανίστηκαν στο μετροπρόγραμμα που εκτελούσε συνεχώς πράξεις MAD και MUL, κάτι το οποίο οφείλεται στη δυνατότητα διπλής-έκδοσης των δύο αυτών εντολών από τους πυρήνες των SMs της συγκεκριμένης αρχιτεκτονικής. Συμπερασματικά, σε πιο γενικές και πρακτικές εφαρμογές θα παίζει σημαντικό ρόλο το μίγμα εντολών που χρησιμοποιείται κάθε φορά και καλό θα ήταν, όπου είναι αυτό δυνατόν, να τοποθετούνται εντολές MAD και MUL κοντινά ώστε να γίνεται χρήση αυτού του χαρακτηριστικού της αρχιτεκτονικής G80.

5.3 Αναγωγή

Σε αυτήν την ενότητα θα εφαρμοστούν κάποιες από τις στρατηγικές που αναφέρθηκαν στο προηγούμενο κεφάλαιο σε έναν στοιχειώδη αλγόριθμο που μπορεί να παραλληλοποιηθεί, με στόχο την πλήρη εκμετάλλευση της κάρτας γραφικών που υποστηρίζει την τεχνολογία του CUDA. Η στοιχειώδης εφαρμογή που θα μελετηθεί εδώ είναι η αναγωγή. Αρχικά υπάρχει ένα σύνολο από δεδομένα (εδώ αριθμοί κινητής υποδιαστολής μονής ακρίβειας) και είναι επιθυμητό να εφαρμοστεί μεταξύ των στοιχείων του ένας τελεστής (ουσιαστικά δηλαδή, να συντελεστεί μια πράξη μεταξύ όλων των στοιχείων του συνόλου). Επιλέχθηκε αυτή η πράξη να είναι η πρόσθεση στα πειράματα που θα παρουσιαστούν στη συνέχεια. Ουσιαστικά στόχος είναι να προστεθούν όλα τα στοιχεία του συνόλου και το τελικό αποτέλεσμα να είναι διαθέσιμο το γρηγορότερο δυνατό.

Είναι εύκολο να διακρίνει κανείς ότι έχοντας στη διάθεση του πολλούς επεξεργαστικούς πυρήνες, μπορεί να μοιράσει τη δουλειά που πρέπει να επιτελεστεί σε αυτούς τους πυρήνες ώστε να πραγματοποιήσει ο καθένας εξ αυτών ένα κλάσμα της συνολικής εργασίας. Με αυτόν τον τρόπο χρησιμοποιείται η παραλληλία σε επίπεδο πυρήνων με σχετικά απλό τρόπο. Στην πλατφόρμα του CUDA αυτό θα επιτευχθεί καθορίζοντας στους παραμέτρους εκτέλεσης της εφαρμογής μεγάλο αριθμό μπλοκ, τα οποία θα περιλαμβάνουν μεγάλο αριθμό νημάτων το καθένα. Με τη σειρά τους αυτά τα μπλοκ θα ανατεθούν σε όλα τα διαθέσιμα SMs όσο το δυνατόν ισομερώς.

Όσον αφορά τις υλοποιήσεις της αναγωγής, ακολουθήθηκαν δυο διαφορετικές προσεγγίσεις με αρκετές εκδόσεις η κάθε μια ώστε να φανούν όλα τα στάδια βελτιστοποίησης και να γίνουν ευδιάκριτες οι πιθανές στρατηγικές βελτίωσης της επίδοσης. Σε γενικές γραμμές, στόχος ήταν η έξοδος κάθε σταδίου της εφαρμογής να αποτελεί είσοδο του επόμενου σταδίου δηλαδή με κλήση του ίδιου υπολογιστικού πυρήνα, με

διαφορετική είσοδο κάθε φορά, να μειώνεται το μέγεθος της αρχικής εισόδου κατά το μισό (στα περισσότερα στάδια, καθώς για πολύ μεγάλα διανύσματα το πρώτο στάδιο ίσως αναγκαζόταν να εκτελεί πρόσθεση μεταξύ περισσότερων δεδομένων).

Στην πρώτη προσέγγιση (πυρήνες R1 και R2), χρησιμοποιήθηκε μόνο η καθολική μνήμη και ο συγχρονισμός μεταξύ των διαδοχικών σταδίων της εφαρμογής έγινε «εξωτερικά» δηλαδή ο πυρήνας κλήθηκε $\log_2$SIZE φορές, όπου SIZE το μέγεθος του αρχικού συνόλου δεδομένων, ώστε κατά την τελευταία κλήση του να επιστρεφόταν ένας αριθμός, το τελικό άθροισμα. Σε κάθε κλήση ο αριθμός των συνολικών νημάτων των παραμέτρων εκτέλεσης υποδιπλασιαζόταν αφού υποδιπλασιαζόταν και οι ανάγκες σε ενεργά νήματα, λόγω της μείωσης στο μισό του συνόλου εισόδου σε σχέση με το προηγούμενο στάδιο. Ο πυρήνας R1 δεν πραγματοποιούσε συγχωνευμένες προσπελάσεις στη μνήμη σε αντίθεση με τον πυρήνα R2.

Η δεύτερη προσέγγιση (πυρήνες R3, R3.5, R4, R4.5, R5, R6) , έκανε χρήση της διαμοιραζόμενης μνήμης στην οποία μεταφέρθηκε στην αρχή το κομμάτι δεδομένων στο οποίο θα δούλευαν τα νήματα του κάθε μπλοκ. Σε αυτές τις εκδόσεις χρειαζόταν δυο κλήσεις του πυρήνα συνολικά σε σύγκριση με τις $\log_2$SIZE της άλλης προσέγγισης, και ο συγχρονισμός ο οποίος είναι απαραίτητος μεταξύ των διαδοχικών σταδίων εντός του κάθε μπλοκ επιτυγχανόταν με τη χρήση της συνάρτησης συγχρονισμού `__syncthreads()`. Εντός του κάθε μπλοκ είχαμε $\log_2$Block_Size στάδια, όπου Block_Size το μέγεθος του μπλοκ σε αριθμό νημάτων. Σε κάθε στάδιο δε, πραγματοποιούνταν οι προσθέσεις μεταξύ δυο στοιχείων από κάθε νήμα, εκτός ίσως από το πρώτο στάδιο στην περίπτωση που ο αριθμός των στοιχείων του συνόλου εισόδου ήταν μεγαλύτερο του διπλάσιου του αριθμού των νημάτων με τον οποίο ξεκινούσε η εφαρμογή. Σε αυτήν την περίπτωση στο πρώτο στάδιο πραγματοποιούνταν παραπάνω από μια πρόσθεση ώστε, στο τέλος του σταδίου, το μέγεθος του συνόλου εξόδου να είναι ίσο με τον συνολικό αριθμό των νημάτων της εφαρμογής (σε κάθε μπλοκ δηλαδή το μέγεθος του συνόλου εξόδου του πρώτου σταδίου θα ήταν ίσο με Block_Size). Στο τέλος, κάθε μπλοκ έγραφε το άθροισμα των στοιχείων τα οποία είχε αναλάβει, στην καθολική μνήμη. Στο τέλος της πρώτης κλήσης του πυρήνα η έξοδος αποτελούνταν από Grid_Size στοιχεία, όπου Grid_Size το μέγεθος του πλέγματος σε μπλοκ. Η τελευταία κλήση του πυρήνα είχε σκοπό να προσφέρει το τελικό άθροισμα όλων των στοιχείων του αρχικού συνόλου και για να γίνει αυτό, χρησιμοποιώντας τον ίδιο πυρήνα με την πρώτη κλήση, έπρεπε να έχει στις παραμέτρους εκτέλεσης της ένα μόνο μπλοκ με όσα νήματα ήταν απαραίτητο. Αυτό συνέβαινε γιατί δεν είναι εφικτή η συνεργασία και ο συγχρονισμός νημάτων που ανήκουν σε διαφορετικά μπλοκ Ο πυρήνας R3 είναι ο πιο απλοϊκός και τον συμπεριλάβαμε για λόγους σύγκρισης, ενώ κάθε ένας από τους επόμενους εισάγει και μια βελτίωση σε σχέση με τον προηγούμενο του (Σχήμα 5-9).

Κάνοντας ενδελεχείς μετρήσεις θα καταστεί δυνατό να εξαχθούν κάποια συμπεράσματα γα την αποδοτικότητα των στρατηγικών βελτιστοποίησης ώστε να αποφανθεί κανείς για το ποιες αξίζουν τον κόπο να χρησιμοποιηθούν κατά κόρον. Στο Σχήμα 5-9 συμπεριλαμβάνονται συνοπτικά οι στρατηγικές που ακολουθεί κάθε μια από τις εκδόσεις του αλγορίθμου που δοκιμάστηκαν.

Στρατηγική Βελτίωσης	Έκδοση αλγορίθμου							
	R1	***R2***	***R3***	***R3.5***	***R4***	***R4.5***	***R5***	***R6***
Χρήση της διαμοιραζόμενης μνήμης	όχι	όχι	ναι	ναι	ναι	ναι	ναι	ναι
Συγχωνευμένες προσπελάσεις στην καθολική μνήμη.	όχι	ναι	όχι	όχι	ναι	ναι	ναι	ναι
Αποφυγή αποκλινόντων στημονιών	-	-	όχι	ναι	ναι	ναι	ναι	ναι
Αποφυγή διαμαχών στις τράπεζες της διαμοιραζόμενης μνήμης	-	-	όχι	όχι	όχι	ναι	ναι	ναι
Ξετύλιγμα του βρόχου τελευταίου ενεργού στημονιού	-	-	όχι	όχι	όχι	όχι	ναι	ναι
Παραμετροποίηση και χρήση του πιο ταιριαστού πυρήνα	όχι	όχι	όχι	όχι	όχι	όχι	όχι	ναι

Σχήμα 5-9 Στρατηγικές βελτιστοποίησης που ακολουθούν οι διαφορετικές εκδόσεις της εφαρμογής της αναγωγής

Παρακάτω παρατίθενται οι κώδικες όλων των εκδόσεων των υπολογιστικών πυρήνων των προγραμμάτων, που χρησιμοποιήθηκαν στα πειράματα της παρούσας ενότητας. Η προσεκτική μελέτη τους σε συνδυασμό με την προηγούμενη ανάλυση

R1

```
__global__ void R1(float *in, float *out, unsigned long
num_of_elems_sumed_per_thread)
{
      unsigned long id = (blockIdx.x*blockDim.x +
threadIdx.x);
      unsigned long j;
      float temp=in[id*num_of_elems_sumed_per_thread];

      for(j = 1; j < num_of_elems_sumed_per_thread; j++)
            temp+= in[id*num_of_elems_sumed_per_thread+j];

      out[id]=temp;

      return;
}
```

R2

```
__global__ void R2(float *array, unsigned long
num_of_elems_sumed_per_thread)
{
      unsigned long id = (blockIdx.x*blockDim.x +
threadIdx.x) ;
      unsigned long j;
      float temp;
      temp= ar-
ray[blockIdx.x*num_of_elems_sumed_per_thread*blockDim.x +
threadIdx.x];

      for(j=1;j<num_of_elems_sumed_per_thread;j++)
            temp+= ar-
```

```
ray[blockIdx.x*num_of_elems_sumed_per_thread*blockDim.x      +
j*blockDim.x + threadIdx.x];

      array[id]=temp;

      return;
}
```

R3

```
__global__    void    R3(float    *array,    unsigned    long
num_els_sum_per_th)
{
      unsigned    long    id    =    (blockIdx.x*blockDim.x    +
threadIdx.x);
      unsigned long i;
      int stride;
      extern __shared__ float shared_array[];

      shared_array[threadIdx.x]= array[id*num_els_sum_per_th];


      for(i=1;i<num_els_sum_per_th;i++)
      shared_array[threadIdx.x]+=                                ar-
ray[id*num_els_sum_per_th+i];
      __syncthreads();

      for(stride=1;stride<blockDim.x;stride*=2)
      {
            if(threadIdx.x%(2*stride)==0)
shared_array[threadIdx.x]    +=    shared_array[threadIdx.x    +
stride];
            __syncthreads();
      }

      if(threadIdx.x==0)                                         ar-
ray[blockIdx.x]=shared_array[threadIdx.x];
}
```

R3.5

```
__global__    void    R3_5(float    *array,    unsigned    long
num_of_els_sum_per_thread
{
      unsigned    long    id    =    (blockIdx.x*blockDim.x    +
threadIdx.x);
      unsigned long i;
      int stride;
      int index;
      extern __shared__ float shared_array[];

      shared_array[threadIdx.x]=                                 ar-
ray[id*num_of_els_sum_per_thread];
      for(i=1;i<num_of_els_sum_per_thread;i++)
            shared_array[threadIdx.x]+=                          ar-
ray[id*num_of_els_sum_per_thread +i];
```

```
      __syncthreads();

      for(stride=1;stride<blockDim.x;stride*=2)
      {
                  index= 2 * stride * threadIdx.x;
                  if(index<blockDim.x)
shared_array[index]+=shared_array[index + stride];
                  __syncthreads();
      }

      if(threadIdx.x==0)array[blockIdx.x]=
shared_array[threadIdx.x];
}
```

R4

```
__global__      void      R4(float*array,      unsigned      long
num_of_elems_sumed_per_thread)
{
      unsigned long i;
      int stride;
      int index;
      extern __shared__ float shared_array[];

      shared_array[threadIdx.x]=                                ar-
ray[blockIdx.x*blockDim.x*num_of_elems_sumed_per_thread      +
threadIdx.x];

      for(i=1;i<num_of_elems_sumed_per_thread;i++)
            shared_array[threadIdx.x]+=                         ar-
ray[blockIdx.x*blockDim.x*num_of_elems_sumed_per_thread+
threadIdx.x +i*blockDim.x];

      __syncthreads();

      for(stride=1;stride<blockDim.x;stride*=2)
      {
            index= 2 * stride * threadIdx.x;
            if(index<blockDim.x)
shared_array[index]+=shared_array[index + stride];
            __syncthreads();
      }

      if(threadIdx.x==0)array[blockIdx.x]=
shared_array[threadIdx.x];
}
```

R4.5

```
__global__    void    R4_5(float    *array,    unsigned    long
num_of_elems_sumed_per_thread)
{
      unsigned long i;
      int stride;
      extern __shared__ float shared_array[];

      shared_array[threadIdx.x]=array[blockIdx.x  *  blockDim.x
* num_of_elems_sumed_per_thread + threadIdx.x];

      for(i=1;i<num_of_elems_sumed_per_thread;i++)
            shared_array[threadIdx.x]+=                       ar-
ray[blockIdx.x*blockDim.x*num_of_elems_sumed_per_thread      +
threadIdx.x +i*blockDim.x];

      __syncthreads();

      for(stride=blockDim.x/2;stride>0;stride/=2)
      {
            if(threadIdx.x<stride)
shared_array[threadIdx.x]+=shared_array[threadIdx.x + stride];
            __syncthreads();
      }

      if(threadIdx.x==0)array[blockIdx.x]=
shared_array[threadIdx.x];
}
```

R5

```
__global__    void    R5(float    *array,    unsigned    long
num_of_elems_sumed_per_thread)
{
      unsigned long i;
      int stride;
      extern __shared__ float shared_array[];

      shared_array[threadIdx.x]=array[blockIdx.x*blockDim.x*
num_of_elems_sumed_per_thread + threadIdx.x];

      for(i=1;i<num_of_elems_sumed_per_thread;i++)
            shared_array[threadIdx.x]+=   array[blockIdx.x   *
blockDim.x                                                  *
num_of_elems_sumed_per_thread+threadIdx.x+i*blockDim.x];

      __syncthreads();

      for(stride=blockDim.x/2;stride>32;stride/=2)
      {
            if(threadIdx.x<stride)
shared_array[threadIdx.x]+=shared_array[threadIdx.x + stride];
            __syncthreads();
      }

      if (threadIdx.x < 32)
      {
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+32];
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+16];
```

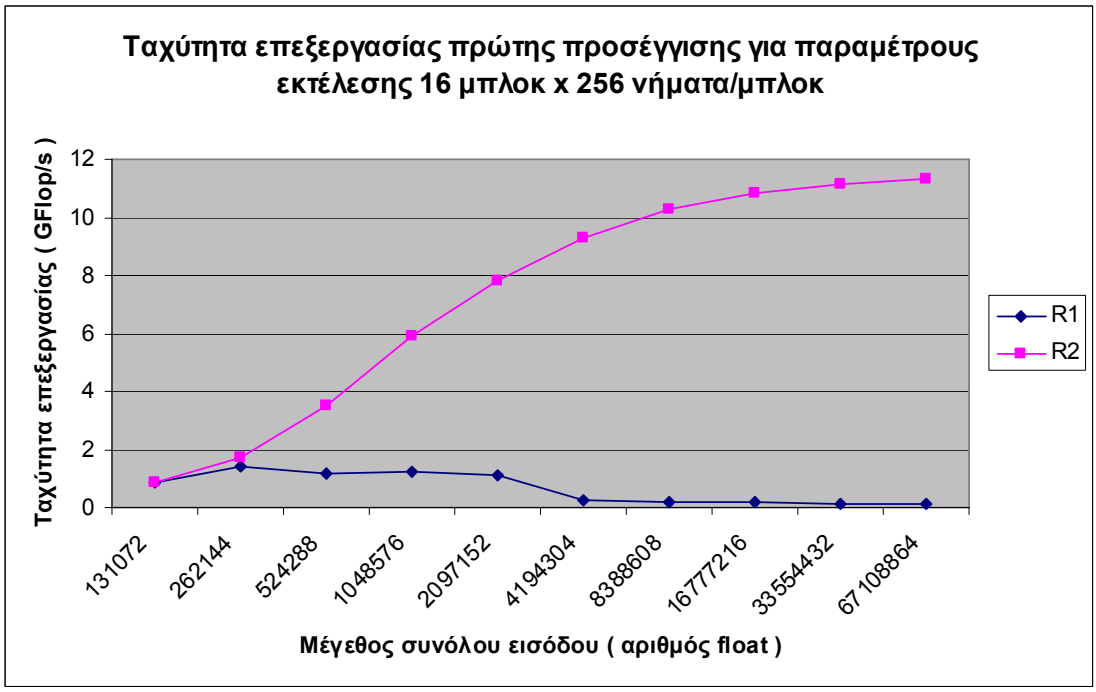
```
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+8];
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+4];
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+2];
      shared_array[threadIdx.x]+=shared_array[threadIdx.x+1];
      }

      if(threadIdx.x==0)array[blockIdx.x]=
shared_array[threadIdx.x];
}
```

Σημειώνεται ότι ο R6 περιλαμβάνει πολλούς διαφορετικούς πυρήνες, ανάλογα με το μέγεθος των μπλοκ, οι οποίοι εργάζονται όπως ο R5 αλλά έχουν πλήρως ξετυλιγμένους όλους τους βρόχους τους (δηλαδή και το πρώτο for υπολογισμών του R5).

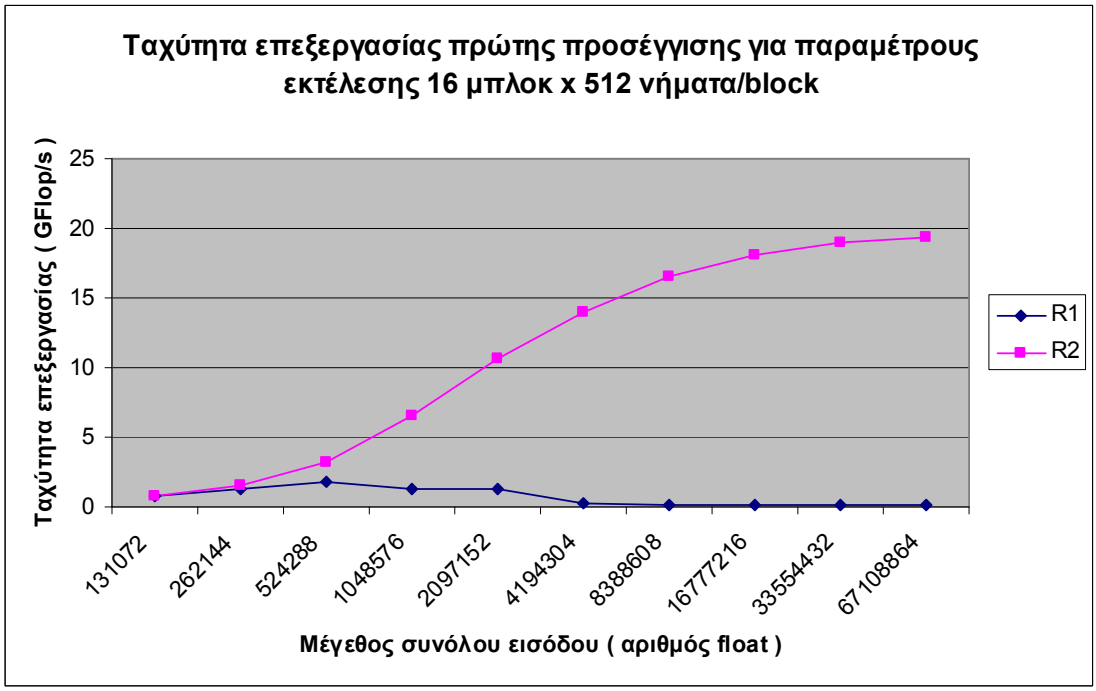
Για λόγους σύγκρισης, αναφέρεται ότι η CPU του συστήματος που χρησιμοποιήθηκε διεκπεραίωνε τον αλγόριθμο της αναγωγής με μέγιστη ταχύτητα που δεν ξεπερνούσε τα 353 MFlop/s για μεγάλα μεγέθη του προβλήματος. Χρησιμοποιήθηκε η σημαία –O3 κατά τη μεταγλώττιση, για τη βελτιστοποίηση του παραγόμενου κώδικα.

5.3.1 Ανάλυση και σύγκριση εκδόσεων πρώτου τρόπου υλοποίησης του αλγορίθμου αναγωγής: Σημασία συγχωνευμένων προσπελάσεων στην καθολική μνήμη

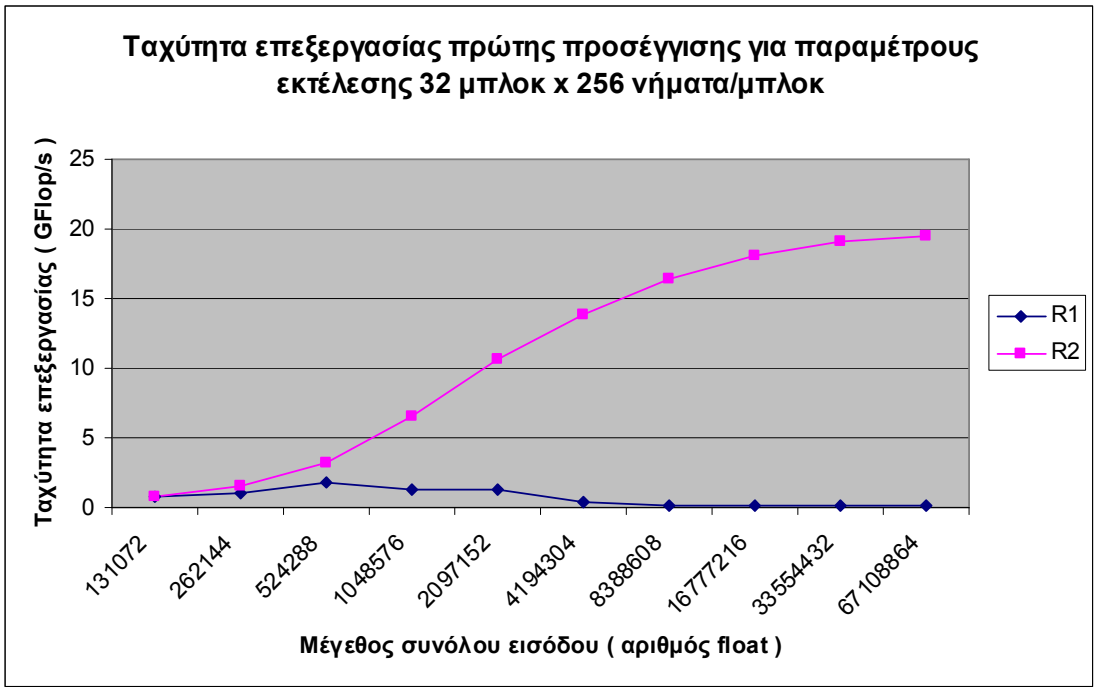
Σε αυτήν την ενότητα θα συγκριθούν οι ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής μεταξύ των εκδόσεων R1 και R2, δηλαδή των δύο τρόπων της πρώτης υλοποίησης του αλγορίθμου. Η διαφορά αυτών των δύο πυρήνων είναι ο τρόπος προσπέλασης μνήμης. Στην R2 εξασφαλίζεται ότι διαδοχικά νήματα (προσπελάζουν την ίδια στιγμή (όταν εκτελούν δηλαδή την ίδια εντολή) συνεχόμενες και διαδοχικές θέσεις της καθολικής μνήμης, ενώ στην περίπτωση του R1 κάτι τέτοιο δε συμβαίνει. Πραγματοποιήθηκαν μετρήσεις σε πολλά διαφορετικά μεγέθη συνόλου εισόδου (από 128K `float` μέχρι και 64M `float`) και σε τέσσερις διαφορετικές διατάξεις των παραμέτρων εκτέλεσης (16x256, 16x512, 32x256 και 32x512 μπλοκ x νήματα/μπλοκ). Στα Σχήματα 5-10 έως 5-13, παρουσιάζονται, σε μορφή διαγραμμάτων, οι μέσες ταχύτητες εκτέλεσης σε κάθε περίπτωση σε GFlop/s δηλαδή δισεκατομμύρια πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο :



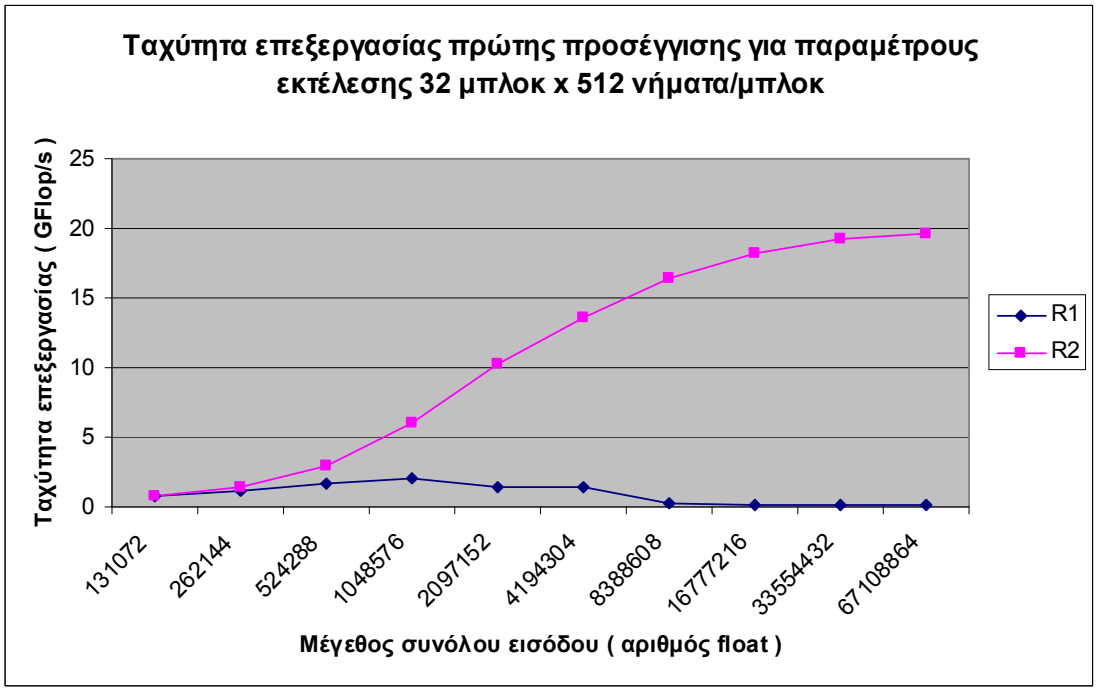
Σχήμα 5-10 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 16 μπλοκ x 256 νήματα/μπλοκ (εκδόσεις R1 και R2) .



Σχήμα 5-11 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 16 μπλοκ x 512 νήματα/μπλοκ (εκδόσεις R1 και R2) .



Σχήμα 5-12 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 32 μπλοκ x 256 νήματα/μπλοκ (εκδόσεις R1 και R2) .



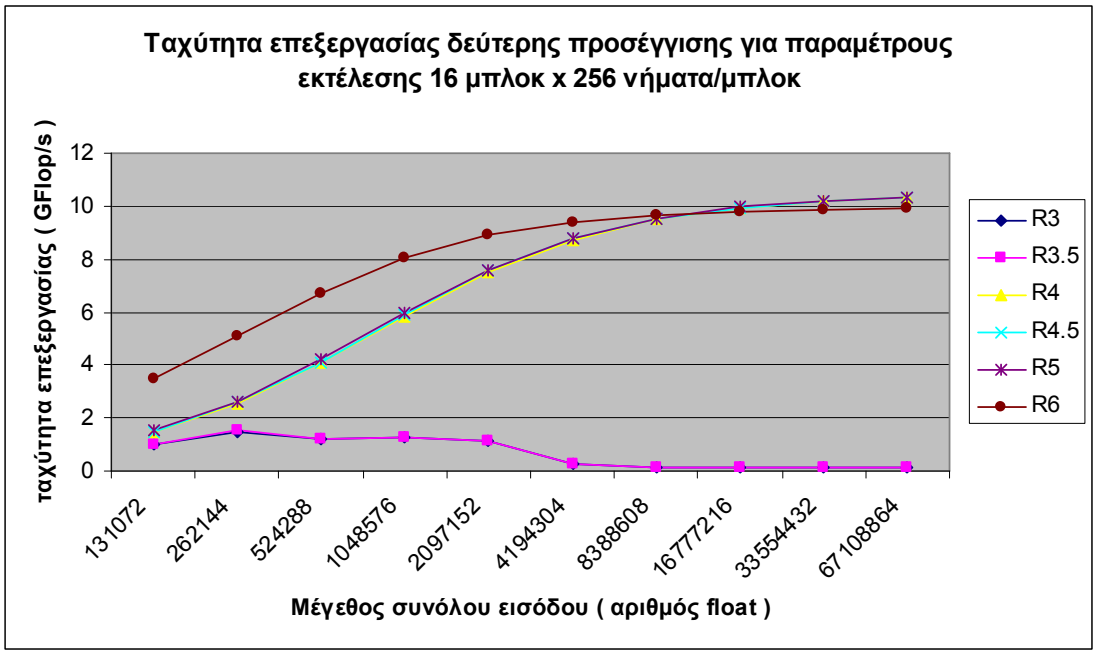
Σχήμα 5-13 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 32 μπλοκ x 512 νήματα/μπλοκ (εκδόσεις R1 και R2) .

Γίνεται σαφές από τα διαγράμματα αυτά, τα οποία ανεξαρτήτως παραμέτρων εκτέλεσης έχουν όλα την ίδια μορφή, του πόσο σημαντικό είναι κατά τη χρήση της τεχνολογίας CUDA οι προσπελάσεις στην καθολική μνήμη να είναι πάντοτε συγχωνευμένες.

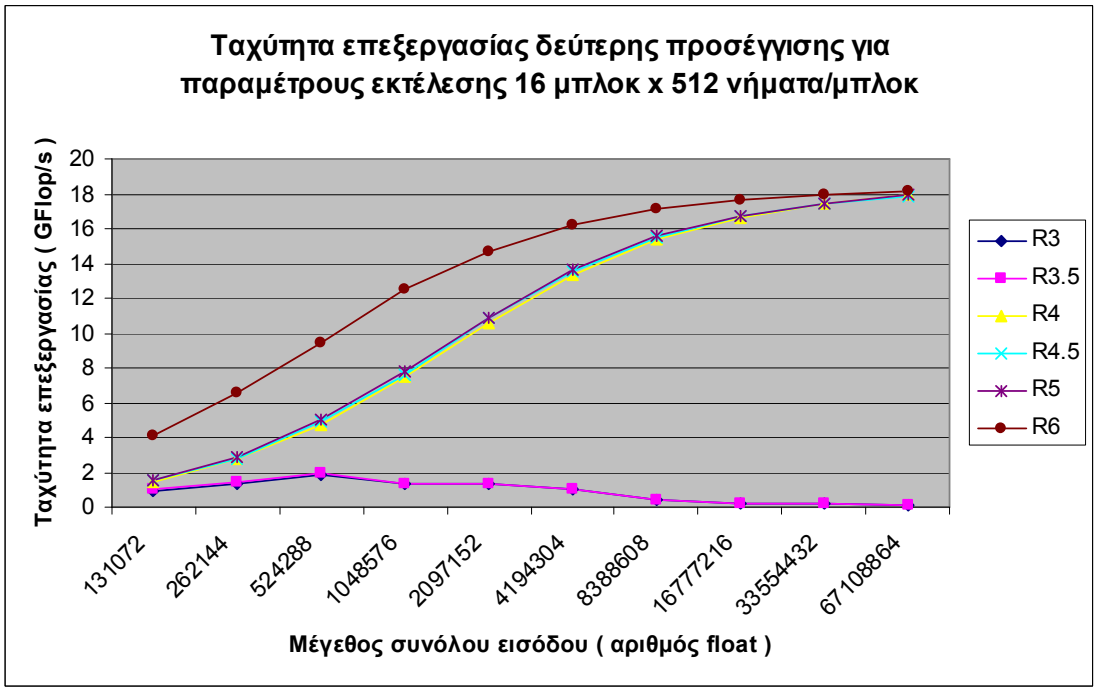
Παρατηρείται ότι, ειδικά σε μεγάλα σύνολα εισόδου, και συνεπώς αυξανόμενες προσπελάσεις στην καθολική μνήμη από κάθε νήμα, οι διαφορές είναι πολύ μεγάλες στην τελική μέση τιμή της ταχύτητας επεξεργασίας. Είναι ξεκάθαρο ότι ο πυρήνας R2 για μεγάλα σύνολα εισόδου είναι μέχρι και 190 φορές πιο γρήγορος, κάτι που αποκτά ιδιαίτερη σημασία αν ληφθεί υπ' όψη ότι μελετάται ένας περιορισμένος-από-μνήμη (memory-bound) αλγόριθμος, δηλαδή ένας αλγόριθμος του οποίου το μεγαλύτερο μέρος των λειτουργιών του είναι μεταφορές μεταξύ μνημών. Ένα ακόμα γεγονός που προκαλεί εντύπωση είναι οι αντίθετες πορείες που ακολουθούν οι καμπύλες της ταχύτητας. Δηλαδή, ο R2 πυρήνας ακολουθεί ανοδική πορεία συνεχώς σε συνάρτηση με το μέγεθος της εισόδου, κάτι που σημαίνει ότι κλιμακώνει καλά μέχρι να φτάσει στα όριά του και να μικρύνει ο ρυθμός αύξησης του. Αντίθετα, ο R2 πυρήνας, για μεγάλα σύνολα εισόδου μειώνει σταδιακά την απόδοσή του, δείγμα των μεγάλων καθυστερήσεων που προκαλούν οι μη συγχωνευμένες προσβάσεις. Συμπεραίνει κανείς, λοιπόν, ότι, όταν πολλαπλασιάζονται οι μη συγχωνευμένες προσβάσεις στην καθολική μνήμη από κάθε νήμα, η απόδοση των εφαρμογών δεν κλιμακώνει καθόλου καλά και συνεχώς μειώνεται. Επιβεβαιώνεται, από τα όσα παρουσιάστηκαν σε αυτήν την ενότητα, ότι οι προσβάσεις στην καθολική μνήμη συμφέρει ιδιαίτερα να είναι συγχωνευμένες, αλλιώς υπάρχει κίνδυνος σημαντικού περιορισμού της ταχύτητα εκτέλεσης των εκάστοτε προγραμμάτων.

5.3.2 Ανάλυση και σύγκριση εκδόσεων δεύτερου τρόπου υλοποίησης του αλγορίθμου αναγωγής: Σημασία βελτιστοποιήσεων αλγορίθμου στον πυρήνα κάθε έκδοσης

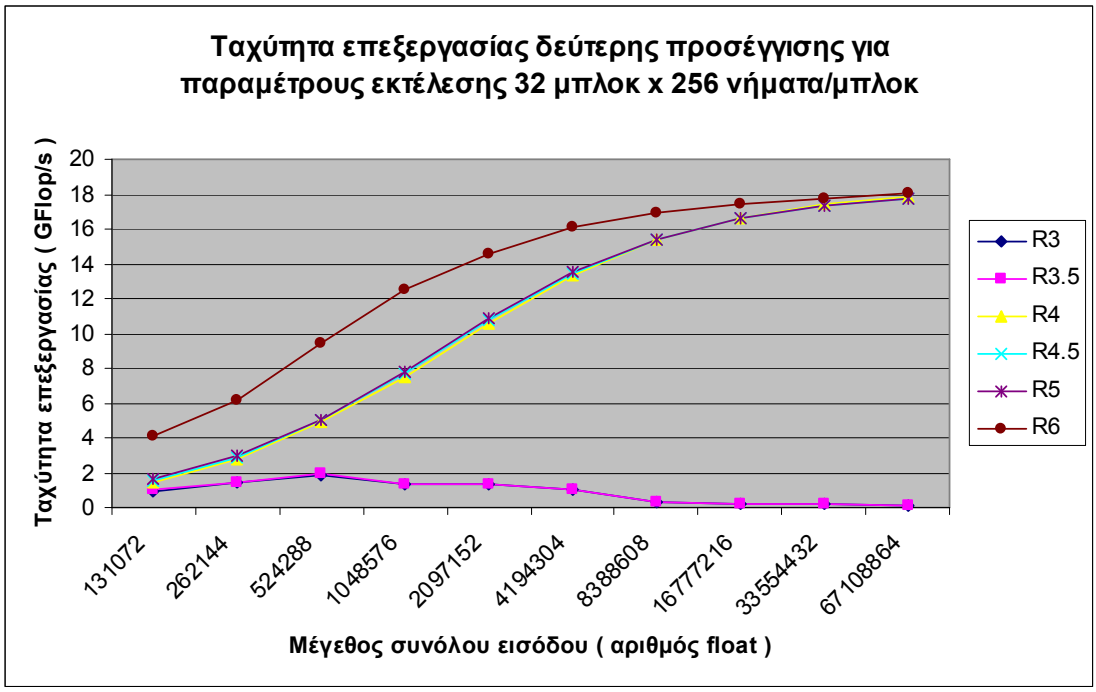
Σε αυτήν την ενότητα θα συγκριθούν οι έξι διαφορετικοί τρόποι με τους οποίους υλοποιήθηκε η αναγωγή με χρήση της διαμοιραζόμενης μνήμης. Θα γίνει μια βήμα-βήμα εξήγηση του τί προσφέρει κάθε μια από τις βελτιώσεις που αναφέρθηκαν στο Σχήμα 5-9. Πραγματοποιήθηκαν, μετρήσεις της ταχύτητας εκτέλεσης των πράξεων σε συνάρτηση με το μέγεθος του συνόλου εισόδου της εφαρμογής για τέσσερα διαφορετικά σετ παραμέτρων εκτέλεσης. Τα αποτελέσματα φαίνονται στα Σχήματα 5-14 έως 5-17, ενώ τα συμπεράσματα θα συμπληρωθούν και θα ενισχυθούν κατά την παράθεση επιπλέον μετρήσεων στην ενότητα 5.3.3 :



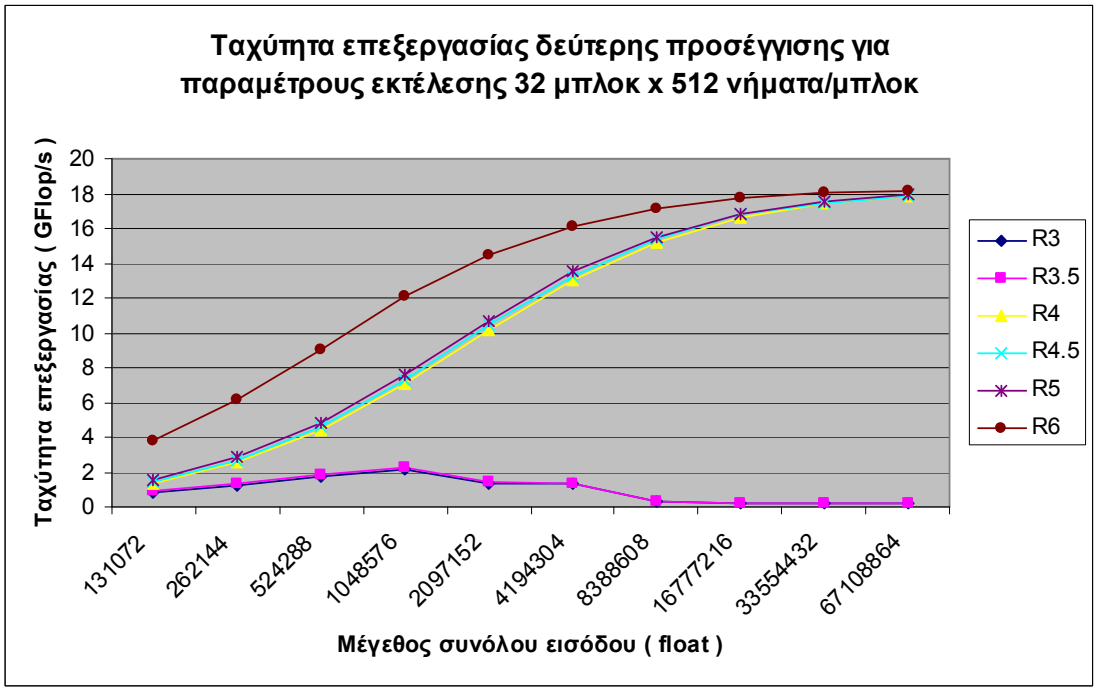
Σχήμα 5-14 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 16 μπλοκ x 256 νήματα/μπλοκ (εκδόσεις R3, R3.5, R4, R4.5 ,R5 και R6) .



Σχήμα 5-15 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 16 μπλοκ x 512 νήματα/μπλοκ (εκδόσεις R3, R3.5, R4, R4.5 ,R5 και R6) .



Σχήμα 5-16 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 32 μπλοκ x 256 νήματα/μπλοκ (εκδόσεις R3, R3.5, R4, R4.5 ,R5 και R6) .



Σχήμα 5-17 Ταχύτητες επεξεργασίας κατά την εκτέλεση της αναγωγής για διάφορα μεγέθη συνόλου εισόδου και για διάταξη παραμέτρων εκτέλεσης 32 μπλοκ x 512 νήματα/μπλοκ (εκδόσεις R3, R3.5, R4, R4.5 ,R5 και R6) .

Αρχικά, αναφέρεται ότι η R3 έκδοση (όπως και όλες οι μεταγενέστερές της) χρησιμοποιεί την διαμοιραζόμενη μνήμη, μεταφέροντας εκεί το κομμάτι δεδομένων το οποίο θα επεξεργαστεί. Κάθε μπλοκ καταλαμβάνει ένα κομμάτι της διαμοιραζόμενης μνήμης και στο τέλος της εκτέλεσης του πυρήνα θα γράψει στην καθολική μνήμη το αποτέλεσμα της αναγωγής του συνόλου δεδομένων που ανέλαβε στην αρχή. Ουσιαστικά, στο πρώτο στάδιο της δεύτερης υλοποίησης πραγματοποιείται κατανεμημένη αναγωγή σε υποσύνολα του αρχικού συνόλου δεδομένων, και τα αποτελέσματα των επιμέρους αυτών αναγωγών θα αναχθούν μέσω της επανάκλήσης του υπολογιστικού, με ένα μοναδικό μπλοκ αυτή τη δεύτερη φορά, με σκοπό την εξαγωγή του τελικού αποτελέσματος. Σε αυτήν την έκδοση οι προσπελάσεις στην καθολική μνήμη κατά τις αρχικές μεταφορές δεν είναι συγχωνευμένες, ενώ παρουσιάζονται διαμάχες κατά την πρόσβαση στη διαμοιραζόμενη μνήμη από τα νήματα του ίδιου μπλοκ καθώς και αποκλίνοντα στημόνια. Αποτελεί τον πιο απλοϊκό και πρόχειρο στη σκέψη υπολογιστικό πυρήνα, ο οποίος θα χρησιμοποιηθεί για λόγους σύγκρισης στη συνέχεια.

Στην έκδοση R3.5 επετεύχθη η εξάλειψη των αποκλινόντων στημονιών. Ο αλγόριθμος, από τη φύση του, χρειάζεται σε κάθε βήμα τον μισό αριθμό ενεργών νημάτων. Στην έκδοση R3 τα εναπομείναντα ενεργά νήματα (αν υποτεθεί ότι διατάσσονται ως προς την εφαρμογή, με συνεχόμενα 32 νήματα να ανήκουν στο ίδιο στημόνι) απείχαν 2^{step} βήματα, όπου step ο αριθμός των βημάτων που έχουν ολοκληρωθεί. Αυτό είχε σα συνέπεια από το δεύτερο βήμα και μετά να αποκλίνουν τα στημόνια καθώς κάποια από τα νήματα που ανήκαν σε αυτά εμφάνιζαν διαφορετική συμπεριφορά (άλλα συνέχιζαν να είναι «ενεργά» κάνοντας μια πρόσθεση στο επόμενο βήμα και άλλα είχαν ολοκληρώσει τις λειτουργίες τους, αλλά δεν μπορούσαν να τερματίσουν καθώς ανήκαν σε στημόνι που περιελάμβανε νήματα με εναπομείναντες εντολές προς εκτέλεση). Τέτοια συμπεριφορά δεν είναι επιθυμητή και δυνητικά μπορεί να καθυστερήσει αρκετά την εκάστοτε εφαρμογή. Στη συγκεκριμένη περίπτωση, όπως φαίνεται και στα Σχήματα 5-14 έως 5-17, όταν επανασυντάχθηκε ο κώδικα, φροντίζοντας να μην υπάρχουν, πλέον, αποκλίνοντα στημόνια, επετεύχθη ένα μικρό κέρδος σε ταχύτητα, το οποίο όμως δεν είναι τόσο σημαντικό και ευδιάκριτο, λόγω του ότι δεν είναι συγχωνευμένες οι προσπελάσεις στην καθολική μνήμη (που όπως θα αποδειχτεί είναι ο κύριος παράγοντας καθυστέρησης). Πάντως δεν αναμενόταν να είναι πολύ μεγάλη η επιτάχυνση μετά από αυτό το στάδιο, αν ληφθεί υπόψη ότι, οι εντολές που προστίθενται σε κάθε αποκλίνον στημόνι από κάποιο βήμα και μετά, είναι κάποιες παραπάνω προσθέσεις, κάθε μια από τις οποίες προσθέτει στον συνολικό χρόνο εκτέλεσης 4 κύκλους μηχανής ανά στημόνι. Αυτός ο χρόνος, σε σύγκριση με τον χρόνο που παίρνει το σύνολο των λειτουργιών της αναγωγής, είναι ένα πολύ μικρό ποσοστό.

Η σημαντικότερη επιτάχυνση του αλγορίθμου επετεύχθη κατά τη μετάβαση από τις μη συγχωνευμένες προσβάσεις στην καθολική μνήμη, στις συγχωνευμένες. Αυτό που άλλαξε ο πυρήνας R4 ήταν τρόπος πρόσβασης στην καθολική μνήμη (και πιο συγκεκριμένα οι αναγνώσεις κατά τη μεταφορά του υποσυνόλου που θα αναλάμβανε το κάθε μπλοκ στις θέσεις της διαμοιραζόμενης μνήμης) ο οποίος έγινε με συγχωνευμένο τρόπο. Στα Σχήματα 5-14 έως 5-17 είναι εμφανές το μεγάλο άλμα στην ταχύτητα εκτέλεσης σε όλες τις περιπτώσεις, ανεξαρτήτως παραμέτρων εκτέλεσης και μεγέθους αρχικού συνόλου. Στην πλειονότητα των περιπτώσεων των σχετικά μεγάλων συνόλων εισόδου, η επιτάχυνση ξεπερνά το 120x σε σχέση με τον πυρήνα R3.5.

Σε όλες τις παραπάνω εκδόσεις της δεύτερης υλοποίησης της αναγωγής οι προσβάσεις στην διαμοιραζόμενη μνήμη σε κάθε βήμα, δεν ήταν προσεγμένες ώστε να απο-

φεύγονται οι διαμάχες. Αυτό ξεπεράστηκε στον πυρήνα R4.5, όπου νήματα με διαδοχικά `threadIdx.x` επιχειρούν προσβάσεις σε διαδοχικές διευθύνσεις της διαμοιραζόμενης μνήμης. Αυτό επετεύχθη με μια μικρή αλλαγή στις διευθύνσεις τις οποίες αναλαμβάνει το κάθε νήμα να προσθέσει. Η επιτάχυνση που παρατηρήθηκε ήταν σχετικά μικρή, της τάξης του 0.1 έως 0.2 GFlop/s, αλλά καλοδεχούμενη.

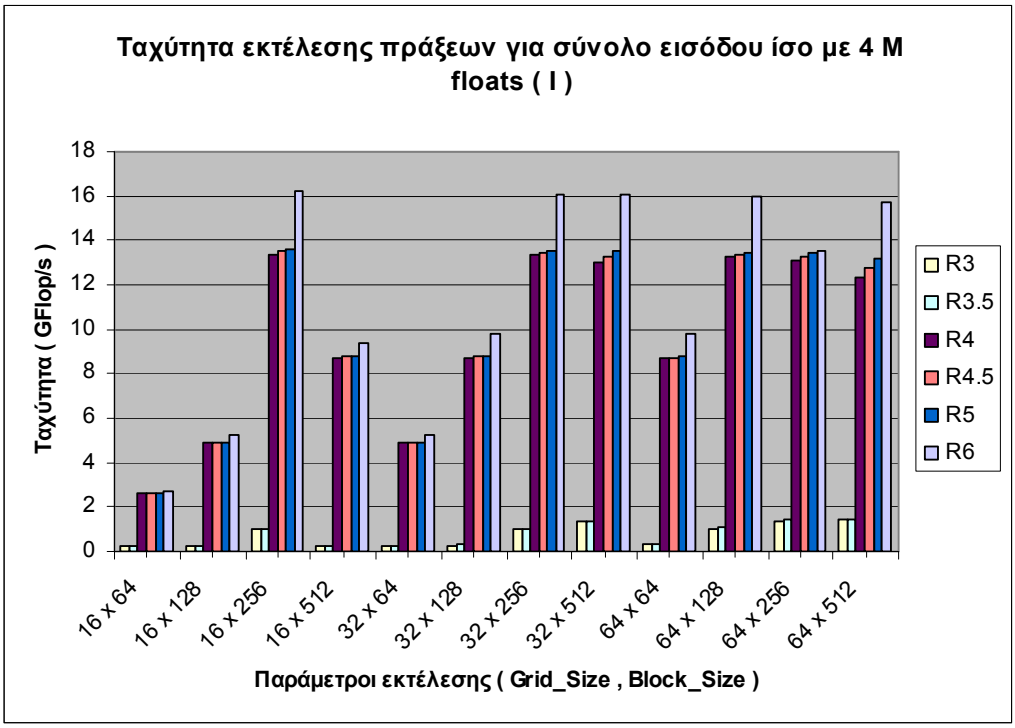
Άλλο ένα σημείο, το οποίο συνεισφέρει σημαντικά στον συνολικό χρόνο εκτέλεσης είναι ο συγχρονισμός μεταξύ των νημάτων του ίδιου μπλοκ. Στον πυρήνα R5 ξετυλίχτηκε πλήρως το τελευταίο ενεργό στημόνι, το οποίο θα είναι το μόνο υπάρχον από τη στιγμή που θα μείνουν 64 στοιχεία στην διαμοιραζόμενη μνήμη, δηλαδή θα προκύψει ανάγκη για 32 ενεργά νήματα. Το ξετύλιγμα του βρόχου για 64 εναπομείναντα στοιχεία και κάτω επιτρέπει σε αυτά τα τελευταία 6 από τα συνολικά 10 (το πολύ) βήματα, να αφαιρεθεί η εντολή συγχρονισμού, καθώς δε θα είναι, πλέον, απαραίτητη. Αυτό συμβαίνει γιατί τα νήματα που έχουν απομείνει, από τη στιγμή που είναι μόνο 32 και κάτω, θα ανήκουν αναγκαστικά το ίδιο στημόνι, οπότε εκ των πραγμάτων τα νήματα θα συγχρονίζονται. Επιπλέον, με αυτή την ενέργεια γλιτώνονται καθυστερήσεις στους ελέγχους και τις πράξεις που γίνονται κατά την απόφαση συνέχισης ή μη του βρόχου for που περιλαμβάνει τις αριθμητικές πράξεις του υπολογιστικού πυρήνα. Κατά τη μετάβασή από την έκδοση R4.5 στην R5, παρατηρήθηκε αύξηση περί τα 0.2 και 0.3 GFlop/s στις μετρήσεις στις οποίες αντιστοιχούν τα Σχήματα 5-14 έως 5-17.

Στο τελευταίο βήμα της προσπάθειάς για βελτιστοποίηση του αλγορίθμου της αναγωγής παραμετροποιήθηκαν οι υπολογιστικοί πυρήνες ώστε ανάλογα με τις παραμέτρους εκτέλεσης της εκάστοτε κλήσης, να καλείται ο ιδανικότερος. Αυτή η προσπάθεια αποτελεί κάτι παρόμοιο με τη χρήση των templates στη γλώσσα C++. Στόχος ήταν να υπάρχουν ανά περίπτωση οι ελάχιστοι παράπλευροι υπολογισμοί (για παράδειγμα πράξεις σύγκρισης κατά την απόφαση για συνέχιση των βρόχων και διάφοροι έλεγχοι απαραίτητοι αν δε γνωρίζουμε εξ'αρχής το μέγεθος των μπλοκ και του πλέγματος) ώστε η ταχύτητά να αυξηθεί κατά ένα ποσοστό. Από τα αντίστοιχα διαγράμματα γίνεται φανερό ότι κάτι τέτοιο επετεύχθη με συνέπεια αύξηση της ταχύτητας μέχρι και 4 GFlop/s.

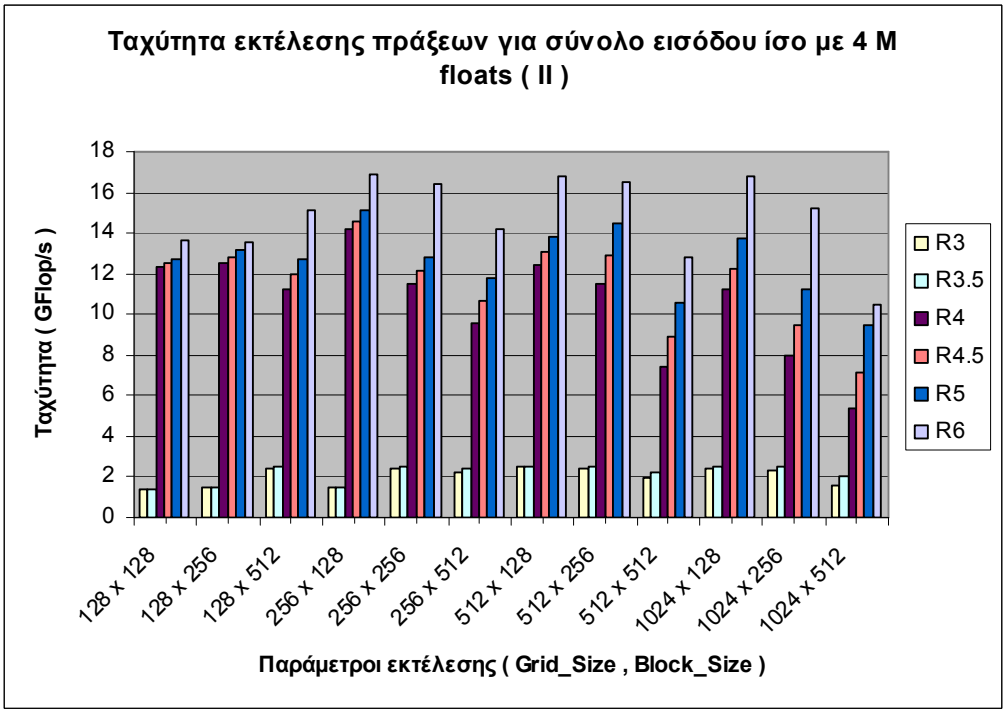
Τα παραπάνω συμπεράσματα αφορούν τέσσερα σύνολα παραμέτρων εκτέλεσης τα οποία, όπως θα αποδειχτεί στην επόμενη ενότητα, είναι τα λιγότερο αποδοτικά. Στις μέχρι στιγμής μετρήσεις δόθηκε βάρος στον αλγόριθμο της εφαρμογής και σε βελτιστοποιήσεις εντός του κώδικα του υπολογιστικού πυρήνα. Οι παραπάνω βελτιστοποιήσεις και τα συμπεράσματα από μόνα τους δεν μπορούν να οδηγήσουν στη βέλτιστη λύση των προβλημάτων σε αυτήν την πλατφόρμα. Θα αναλυθεί αργότερα ότι, οι επιλογές που θα γίνουν στις παραμέτρους εκτέλεσης και στις ανάγκες σε πόρους του κάθε νήματος, έχουν πολύ μεγάλη σημασία και χρειάζεται ιδιαίτερη προσπάθεια και προσοχή για την επιλογή των βέλτιστων.

5.3.3 Πρόσθετες μετρήσεις δεύτερης υλοποίησης - Αναζήτηση των ιδανικών παραμέτρων εκτέλεσης: Σημασία βελτιστοποιήσεων εκτός του κώδικα του πυρήνα

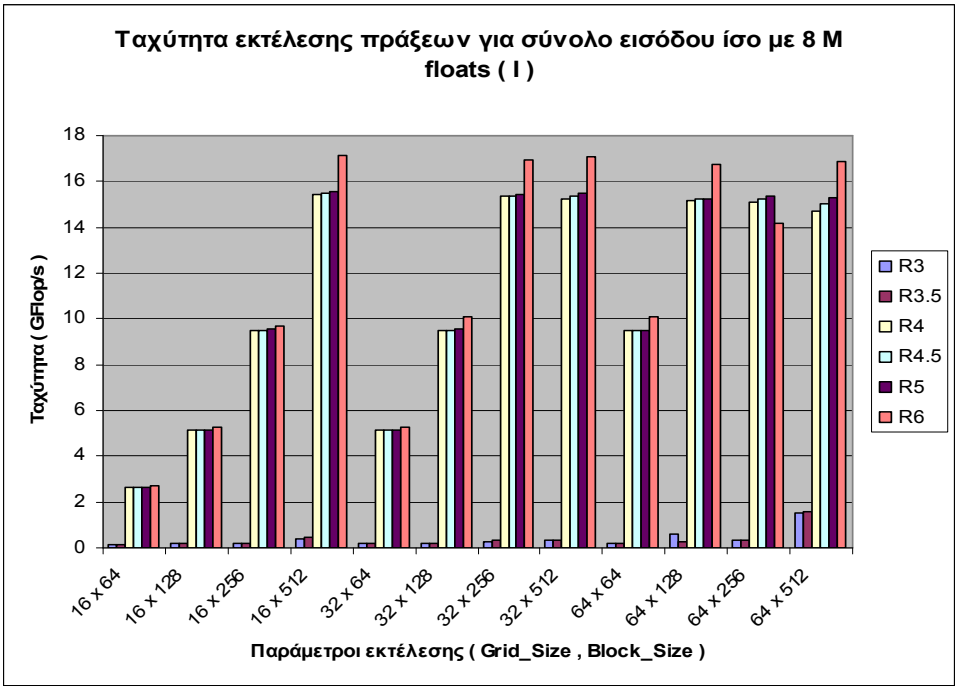
Σε αυτή την ενότητα θα παρουσιαστούν μετρήσεις που έγιναν για μεγέθη συνόλου εισόδου 4M `float` (16 MB), 8M `float` (32 MB) και 16M `float` (64 MB) για πολλές διαφορετικές παραμέτρους εκτέλεσης, Στη συνέχεια, θα σχολιαστούν τα αντίστοιχα διαγράμματα και η επίπτωση της επιλογής παραμέτρων εκτέλεσης στην ταχύτητα. Οι εν λόγω μετρήσεις παρουσιάζονται στα Σχήματα 5-18 έως 5-23 :



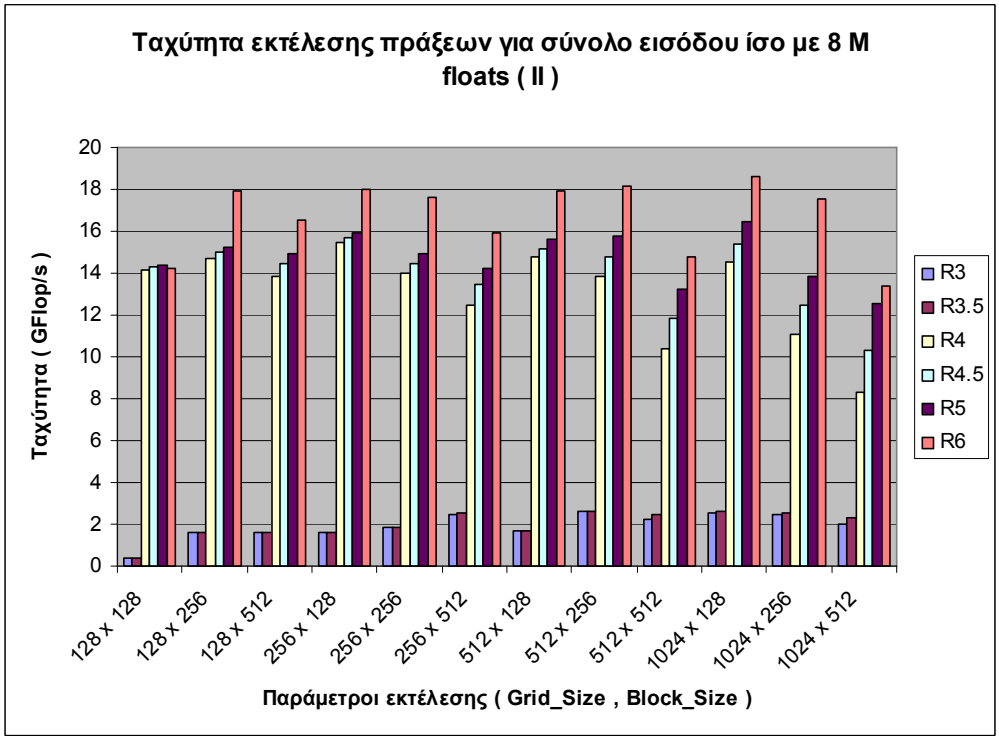
Σχήμα 5-18 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 4 M `float` (πρώτο μέρος).



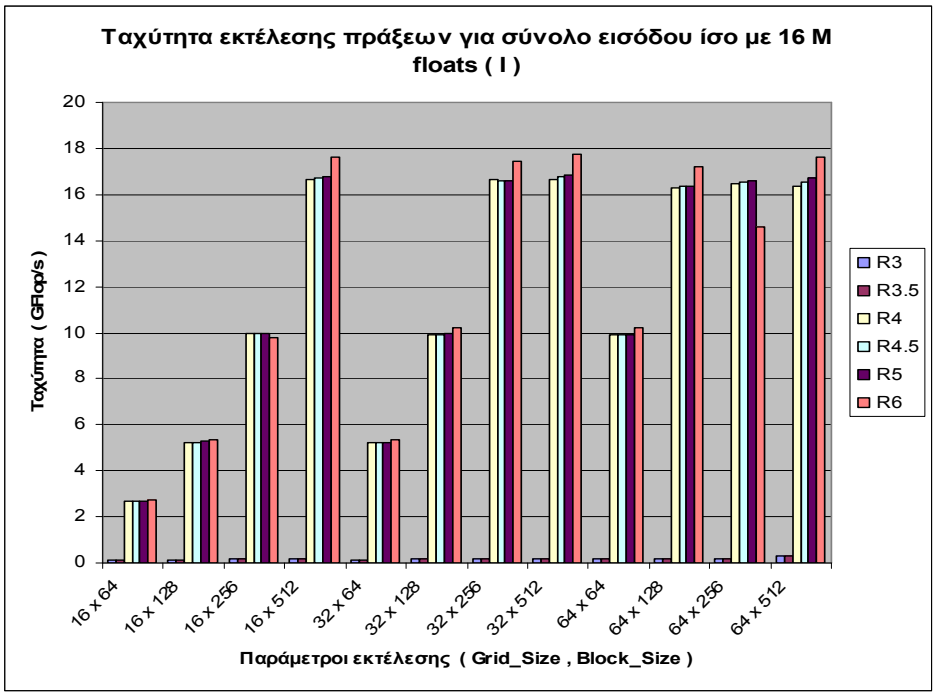
Σχήμα 5-19 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 4 M `float` (δεύτερο μέρος).



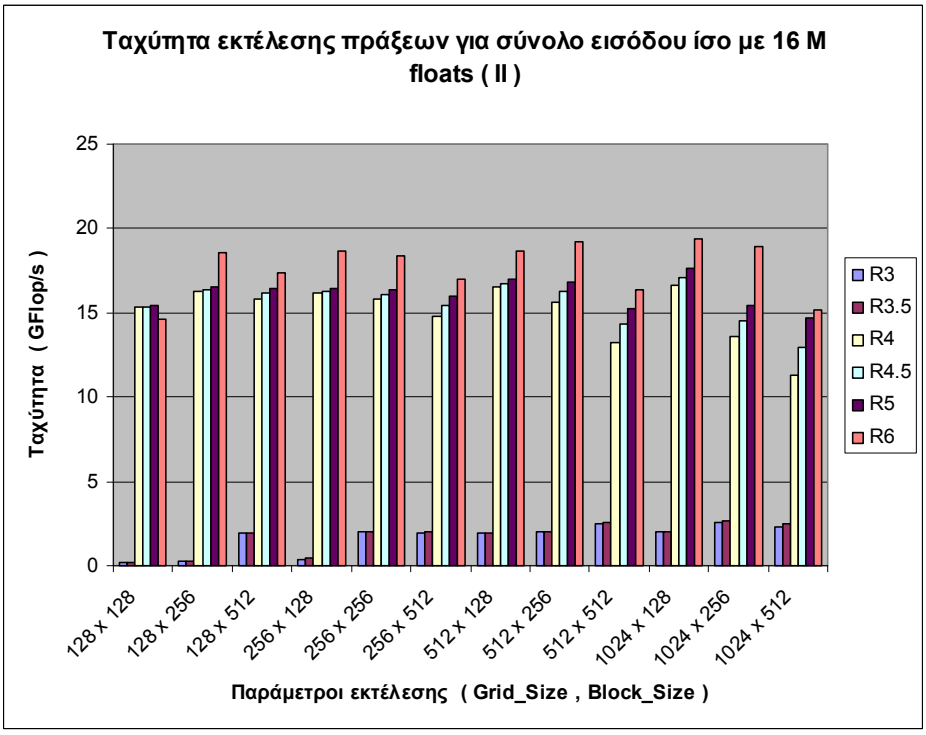
Σχήμα 5-20 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 8 M `float` (πρώτο μέρος).



Σχήμα 5-21 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 8 M `float` (δεύτερο μέρος).



Σχήμα 5-22 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 16 M `float` (πρώτο μέρος).



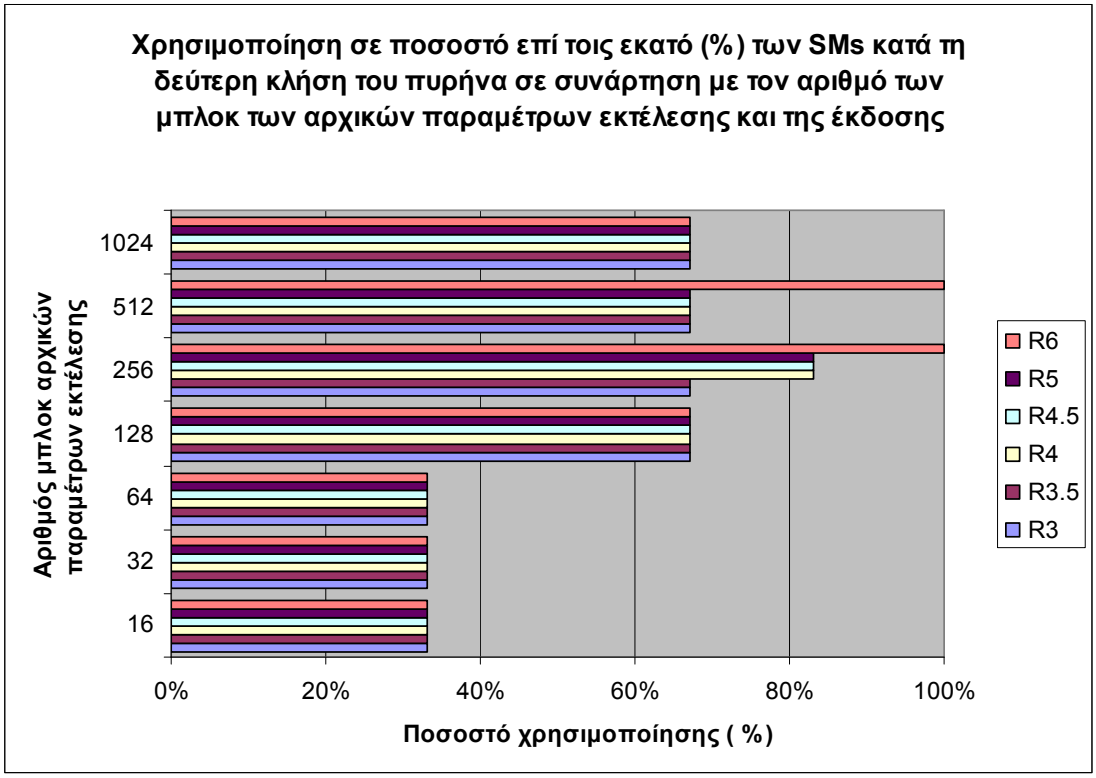
Σχήμα 5-23 Ταχύτητα εκτέλεσης πράξεων σε συνάρτηση με τις παραμέτρους εκτέλεσης και την έκδοση της δεύτερης υλοποίησης για σύνολο εισόδου 16 M `float` (δεύτερο μέρος).

Πρώτα από όλα αναφέρεται ένα γενικό συμπέρασμα που εξάγεται από την παρατήρηση των διαγραμμάτων αυτών. Αυτό είναι το γεγονός αύξησης της επίδοσης όσο αυξάνονται μέχρι ενός σημείου τα νήματα τα οποία εκτελούνται. Σε όλες τις περιπτώσεις μέχρι και τη χρήση διατάξεων με 128 μπλοκ και πάνω, είχαμε σημαντική αύξηση στην ταχύτητα. Από εκεί και έπειτα ήδη υπάρχουν αρκετά μπλοκ ώστε να είναι απασχολημένοι όλοι οι επεξεργαστικοί πυρήνες, και η επίδοση εξαρτάται από άλλους παράγοντες, οι οποίοι θα ερευνηθούν αμέσως παρακάτω.

Συγκρίνοντας τις έξι διαφορετικές εκδόσεις, για οποιεσδήποτε παραμέτρους εκτέλεσης και για κάθε σύνολο παραμέτρων εκτέλεσης ξεχωριστά, παρατηρεί κανείς ότι υπάρχει μια συνεχής βελτίωση όσο η έκδοση του υπολογιστικού πυρήνα είναι μεταγενέστερη. Αυτή μεταβάλλεται ανάλογα με την βελτίωση που εισάγεται κάθε φορά με σημαντικότερη, όπως αναμενόταν τη βελτίωση κατά τη μετάβαση από τον R3.5 στον R4. Πολύ σημαντική, επίσης, ήταν η επιτάχυνση από τον πυρήνα R5 στον R6 κάτι που επιβεβαιώνει ότι είναι δυνατόν να υπάρξει αξιόλογη επιτάχυνση αν είναι από πριν γνωστές οι παράμετροι εκτέλεσης.

Στην αναζήτηση των ιδανικών παραμέτρων εκτέλεσης και στην προσπάθεια εξήγησης των Σχημάτων 5-18 έως 5-23 βοήθησε η χρήση του Υπολογιστή Χρησιμοποίησης που προσφέρει το CUDA (για περιγραφή βλ. και 3.4.5.2). Υπενθυμίζεται ότι ο δεύτερος τρόπος υλοποίησης του αλγορίθμου της αναγωγής γίνεται σε δυο στάδια. Στο πρώτο γίνεται η κατανεμημένη αναγωγή σε όσα μπλοκ καθορίζουν οι αρχικές παράμετροι εκτέλεσης, και στο δεύτερο και τελικό στάδιο έχουμε κλήση του πυρήνα με ένα μπλοκ και τόσο νήματα όσα ο αριθμός των μπλοκ του πρώτου σταδίου. Έτσι υπολογίστηκε, για κάθε περίπτωση αρχικών παραμέτρων εκτέλεσης, η χρησιμοποίηση των SMs. Στα Σχήματα 5-24 και 5-25 διακρίνεται τη χρησιμοποίηση στο πρώτο και στο δεύτερο στάδιο της εφαρμογής ανάλογα με τις αρχικές παραμέτρους εκτέλεσης.

Σχήμα 5-24 Χρησιμοποίηση σε ποσοστό επί τοις εκατό (%) των SMs κατά την πρώτη κλήση του πυρήνα σε συνάρτηση με το μέγεθος του μπλοκ των αρχικών παραμέτρων εκτέλεσης και της έκδοσης .



Σχήμα 5-25 Χρησιμοποίηση σε ποσοστό επί τοις εκατό (%) των SMs κατά τη δεύτερη κλήση του πυρήνα σε συνάρτηση με τον αριθμό των μπλοκ των αρχικών παραμέτρων εκτέλεσης και της έκδοσης.

Πριν εξηγηθεί ο λόγος που ο αλγόριθμός «τρέχει» πιο αποδοτικά για κάποιες συγκεκριμένες παραμέτρους εκτέλεσης, πρέπει να αναφερθεί η δυσκολία που έχει παρουσιαστεί από ακαδημαϊκούς ερευνητές-χρήστες της τεχνολογίας CUDA κατά την αναζήτηση των ιδανικών παραμέτρων εκτέλεσης. Σε πολλές περιπτώσεις αυτοί ωθήθηκαν στο να προβούν σε εξαντλητικές δοκιμές όλων των πιθανών παραμέτρων εκτέλεσης, προκειμένου να επιλέξουν τον καλύτερο κατά την κρίση τους [58, 59]. Πολλές φορές, αλλαγές στις παραμέτρους εκτέλεσης μπορεί να προκαλέσουν μεγάλες αλλαγές στην επίδοση και στην ταχύτητα εκτέλεσης των αλγορίθμων, και αυτό έχει να κάνει συνήθως με το ποσοστό χρησιμοποίησης των SMs, δηλαδή ουσιαστικά με το πόσο αποδοτικά χρησιμοποιούνται οι πόροι που διαθέτει η κάρτα γραφικών σε κάθε χρονική στιγμή. Για να γίνει το παραπάνω πιο κατανοητό μπορεί κανείς να σκεφτεί το εξής: Έστω ότι υπάρχει μια εφαρμογή, όπως η αναγωγή για παράδειγμά. Η εφαρμογή αυτή περιλαμβάνει έναν αριθμό ενεργειών που πρέπει να ολοκληρωθούν προκειμένου να τερματιστεί με σωστά και πλήρη αποτελέσματα. Διαθέτοντας 128 SPs οργανωμένους σε 16 SMs και έχοντας τη δυνατότητα επιλογής των παραμέτρων εκτέλεσης, στην ουσία υπάρχουν πολλές επιλογές διαμοιρασμού και οργάνωσης των παραπάνω λειτουργιών στο διαθέσιμο υλικό. Κάθε στιγμή η πληρότητα του κάθε SM περιορίζεται από τέσσερις παράγοντες :

- Μέγιστος αριθμός των νημάτων που υποδέχεται το SM είναι τα 768, ανεξάρτητα σε πόσα διαφορετικά μπλοκ ανήκουν. Εναλλακτικά μπορούμε να πούμε ότι ο μέγιστος αριθμός στημονιών που μπορεί να υποδεχτεί ένα SM είναι 24.
- Μέγιστος αριθμός μπλοκ ανεξαρτήτως μεγέθους που δύναται να φιλοξενήσει το SM είναι 8.

- Οι ανάγκες των καταχωρητών όλων των νημάτων που τρέχουν κάποια στιγμή σε ένα SM μπορούν να είναι 8192, όσο δηλαδή και το μέγεθος του αρχείου καταχωρητή.
- Οι ανάγκες των μπλοκ που είναι ενεργά σε ένα SM δεν πρέπει να ξεπερνούν το μέγεθος της διαθέσιμης διαμοιραζόμενης μνήμης που ανέρχεται στα 16 KB.

Όταν επιλεχθούν παράμετροι εκτέλεσης με αρκετά μεγάλο αριθμό μπλοκ, ώστε να μη χωρούν να εκτελεστούν ταυτόχρονα σε όλα τα SMs, κάποια παραμένουν σε αδράνεια μέχρι να φτάσει η σειρά τους και τα υπόλοιπα γίνονται ενεργά και καταλαμβάνουν πόρους στα SMs αμέσως πριν ξεκινήσει η εκτέλεσή τους. Για παράδειγμα, αν υπάρχουν 256 μπλοκ προς δρομολόγηση και, λόγω των αναγκών της εφαρμογής και των περιορισμών που αναφέρθηκαν παραπάνω, μπορούν να είναι ενεργά 2 μπλοκ σε κάθε SM ανά πάσα στιγμή, τελικά μπορούν να είναι ενεργά σε ολόκληρη την κάρτα γραφικών 32 μπλοκ. Τα υπόλοιπα 96 θα περιμένουν τη σειρά τους. Συνολικά θα χρειαστούν 4 «φουρνιές» από 32άδες μπλοκ για να ολοκληρωθούν οι λειτουργίες της εφαρμογής.

«Πειράζοντας» το μέγεθος του πλέγματος και το μέγεθος των μπλοκ, επηρεάζεται ο αριθμός των «φουρνιών» που θα χρειαστούν συνολικά. Δηλαδή, συμφέρει να υπάρχουν όσο το δυνατόν λιγότερες «φουρνιές», με όσο το δυνατόν μεγαλύτερη χρησιμοποίηση του υλικού από κάθε μια από αυτές. Ο αριθμός των συνολικών λειτουργιών που πρέπει να εκτελεστούν δεν αλλάζει τελικά, αφού γίνεται λόγος πάντα για την ίδια εφαρμογή. Αλλάζει, όμως, το σε πόσα νήματα θα διαμοιραστούν αυτές και το πόσα νήματα θα εκτελούνται παράλληλα κάθε χρονική στιγμή. Για παράδειγμα,, αν μπορούν να χωρέσουν στο SM μέχρι 256 νήματα (λόγω απαιτήσεων σε καταχωρητές), και μέχρι 4 μπλοκ (λόγω απαιτήσεων σε διαμοιραζόμενη μνήμη), δε συμφέρει να καθοριστεί μέγεθος μπλοκ το 32, καθώς τότε θα υπάρχουν 4 μπλοκ μεν ενεργά, αλλά συνολικά μόνο 128 ενεργά νήματα. Σε αντίθεση με αυτήν την περίπτωση αν επιλεγόταν μέγεθος μπλοκ ίσο με 64, θα επιτυγχάνονταν μέγιστη χρησιμοποίηση αφού θα υπήρχαν 256 ενεργά νήματα κάθε στιγμή σε κάθε SM. Το ίδιο θα συνέβαινε και για επιλογή μεγέθους μπλοκ ίσο με 128 ή 256. Σε διαφορετική περίπτωση αν ενδιαφέρει ο αριθμός των μπλοκ να είναι ο μέγιστος δυνατός, γιατί, για παράδειγμα, το κάθε μπλοκ είναι το «δομικό στοιχείο» του αλγορίθμου (δεν λειτουργεί, δηλαδή, απλά σαν ομαδοποίηση των νημάτων), πρέπει να προσεχθεί ώστε να επιλεγεί όσο μικρό μέγεθος μπλοκ είναι δυνατόν ώστε να είναι ενεργά τόσα μπλοκ όσα επιτρέπει ο περιορισμός των αναγκών σε διαμοιραζόμενη μνήμη. Τέλος, έστω ότι δεν τίθεται περιορισμός από τις ανάγκες σε αριθμό καταχωρητών και σε bytes διαμοιραζόμενης μνήμης. Τότε, συμφέρει οι παράμετροι εκτέλεσης να περιλαμβάνουν μέγεθος μπλοκ ίσο με 256 (ή 128) αντί για 512, ώστε να είναι ενεργά 3 (ή αντίστοιχα 6) μπλοκ την ίδια στιγμή, με 768 συνολικά ενεργά νήματα αντί, για απλά 1 μπλοκ με 512 ενεργά νήματα, καθώς σε αυτήν την περίπτωση θα εκτελούνται παράλληλα περισσότερα νήματα, επιτυγχάνοντας καλύτερη αξιοποίηση πόρων.

Θα αναλυθεί τώρα, πώς, όλα όσα εξηγήθηκαν στην προηγούμενη παράγραφο, έρχονται να εφαρμόσουν και στην πράξη και συγκεκριμένα στην αναγωγή. Το ποσοστό χρησιμοποίησης της πρώτης από τις δύο κλήσεις του πυρήνα εξαρτάται αποκλειστικά και μόνο από το μέγεθος του μπλοκ με το οποίο εκκινεί η εφαρμογή, όπως φαίνεται από το Σχήμα 5-24. Το ποσοστό χρησιμοποίησης της δεύτερης κλήσης του πυρήνα, εξαρτάται αποκλειστικά από τον αριθμό των μπλοκ των αρχικών παραμέτρων εκτέ-

λεσης. Πρέπει να σημειωθεί ότι η χρησιμοποίηση δεν έχει αντίκτυπο στην δεύτερη κλήση του πυρήνα και περιλαμβάνεται στο Σχήμα 5-25 για λόγους πληρότητας. Αυτό συμβαίνει γιατί, η δεύτερη κλήση περιλαμβάνει μόνο ένα μπλοκ και άρα δεν τίθεται θέμα αύξησης των απαιτούμενων «φουρνιών» και της μη καλής χρησιμοποίησης πόρων. Οι μόνοι περιορισμοί οι οποίοι ισχύουν είναι να χωράει το μπλοκ από άποψης μεγέθους σε νήματα, να μην έχει ανάγκες σε bytes διαμοιραζόμενης μνήμης πάνω από το μέγιστο επιτρεπτό (16 KB) και να μην έχει απαιτήσεις σε καταχωρητές πάνω από το όριο των 8192. Γενικά, στην δεύτερη κλήση χρησιμοποιείται πολύ μικρό ποσοστό του διαθέσιμου υλικού της κάρτας γραφικών καθώς μόνο ένα από τα 16 SMs θα έχει να επιτελέσει κάποιες λειτουργίες. Γενικά, το ποσοστό χρησιμοποίησης σαν μετρική είναι αξιόπιστο για να προβεί κανείς σε ασφαλή συμπεράσματα σε περιπτώσεις που έχουν σχετικά μεγάλο αριθμό μπλοκ. Μια τέτοια περίπτωση είναι αυτή της πρώτης κλήσης του πυρήνα στο παράδειγμα της αναγωγής, σε αντίθεση με τη δεύτερη όπως εξηγήθηκε παραπάνω. Η ανάλυση θα επικεντρωθεί κυρίως στις εκδόσεις R5 και R6, οι οποίες είναι και αυτές με την μεγαλύτερη επίδοση, αλλά το ίδια πράγματα ισχύουν και για τις άλλες εκδόσεις οι οποίες επηρεάζονται εξ'ίσου από τις παραμέτρους εκτέλεσης. Παρατηρείται ότι, για οποιοδήποτε δεδομένο αριθμό μπλοκ όταν το μέγεθος του μπλοκ είναι 128 επιτυγχάνεται η καλύτερη επίδοση, με κορυφαίες τις περιπτώσεις των 1024x128, 512x128 και 256x256, μαζί με τις περιπτώσεις του μεγέθους μπλοκ 256, και πιο συγκεκριμένα τις εκδόσεις 256x256 και 512x256. Επίσης, παρατηρείται ότι, όσα περισσότερα συνολικά νήματα χρησιμοποιούνται, τόσο υπάρχει μια τάση αύξησης της ταχύτητας που σε συνδυασμό με τα καλύτερα μεγέθη μπλοκ, που αποδεικνύεται ότι είναι τα 128 και 256, οδηγούν στις καλύτερες επιδόσεις. Όμως, δεν αρκεί να είναι απλώς ο μεγάλος αριθμός νημάτων αλλά και ο τρόπος οργάνωσής τους να είναι τέτοιος, ώστε η χρησιμοποίηση να κυμαίνεται πάντα σε υψηλά επίπεδα.

Με τη βοήθεια του Σχήματος 5-24 μπορεί κανείς να πει ότι αυτή ήταν και η αναμενόμενη συμπεριφορά, καθώς αυτά τα μεγέθη μπλοκ εξασφαλίζουν χρησιμοποίηση 100% για τον R6. Θα γίνει αναφορά, επίσης, σε κάποια λεπτά σημεία τα οποία χρίζουν προσοχής κα επιβεβαιώνουν την παραπάνω ανάλυση:

- Στις περιπτώσεις των 8M `float` και των 16M `float` είναι αρκετά ευδιάκριτο αυτό που αναφέρθηκε για την αξιοπιστία του ποσοστού χρησιμοποίησης σαν μετρική σε συνδυασμό με τον αριθμό διαθέσιμων μπλοκ προς επεξεργασία. Παρατηρείται ότι για μέγεθος μπλοκ 128 είναι καλύτερες οι περιπτώσεις με τα 64, 256 και 1024 μπλοκ σε σχέση με τις αντίστοιχες περιπτώσεις με μέγεθος μπλοκ ίσο με 256. Στις περιπτώσεις που υπάρχουν αρχικά 128 και 512 μπλοκ, υπερέχουν αυτές με μέγεθος μπλοκ ίσο με 256 έναντι εκείνων με μέγεθος μπλοκ 128. Αυτό μπορεί με μια πρώτη ματιά να ξενίσει καθώς θα αναμενόταν για ίδια χρησιμοποίηση και για ολοένα αυξανόμενο αριθμό νημάτων να αυξάνεται η επίδοση. Όμως, αν κοιταχτούν τα αποτελέσματα πιο προσεκτικά, αντιλαμβάνεται κανείς ότι προκύπτει θέμα μιας επιπλέον φουρνιάς στις περιπτώσεις που καθυστερούν. Για παράδειγμα, όταν ο αριθμό μπλοκ είναι 256, θα περίμενε κανείς η περίπτωση με μέγεθος μπλοκ 256 να είναι πιο γρήγορη από την περίπτωση του μεγέθους μπλοκ 128, λόγω περισσότερων νημάτων, αλλά κάτι τέτοιο δε συμβαίνει. Διαιρώντας τον αριθμό μπλοκ με τον αριθμό των SMs προκύπτει ότι σε κάθε SM αντιστοιχούν 16 μπλοκ προς επεξεργασία σε πολλαπλές «φουρνιές». Με μέγεθος μπλοκ 256, 100% χρησιμοποίηση (Σχήμα 5-24) σημαίνει 3 μπλοκ ανά «φουρνιά», δηλαδή συνολικά θα

χρειαστούν 6 «φουρνιές» με το υλικό να απασχολείται πλήρως, εκτός από την τελευταία «φουρνιά» η οποία θα έχει χρησιμοποίηση 33% καθώς θα περιλαμβάνει ένα μπλοκ από τα 3 που συνολικά μπορούν να φιλοξενηθούν. Αντίθετα, για μέγεθος μπλοκ 128, 100% χρησιμοποίηση (Σχήμα 5-24) σημαίνει 6 μπλοκ ανά «φουρνιά», δηλαδή συνολικά χρειάζονται 3 «φουρνιές», οι μισές. Βέβαια, οι «φουρνιές» της τελευταίας περίπτωσης είναι πιο «βαριές» σε υπολογισμούς, αλλά και σε αριθμό προσβάσεων στην καθολική μνήμη. Στην περίπτωση που ο αριθμός μπλοκ είναι 128, με τον ίδιο συλλογισμό καταλήγει κανείς ότι χρειάζονται 3 «φουρνιές» για την περίπτωση του μεγέθους μπλοκ 256 και 2 «φουρνιές» για την περίπτωση των 128 νημάτων ανά μπλοκ. Δηλαδή σε αυτήν την περίπτωση απαιτούνται λιγότερες από τις μισές και εκεί οφείλεται το πλεονέκτημα του μεγέθους μπλοκ 256. Όταν, δηλαδή, ο αριθμός των «φουρνιών» των 256 νημάτων/μπλοκ σε σχέση με αυτές των 128 νημάτων/μπλοκ είναι διπλάσιος ή παραπάνω, πλεονέκτημα παρουσιάζει η περίπτωση των 128 νημάτων/μπλοκ, ενώ σε άλλη περίπτωση προτιμότερη είναι η έκδοση με μέγεθος μπλοκ 256 νήματα.

- Πολύ σημαντικό ρόλο παίζει πάντα η ύπαρξη πολλών νημάτων εκτέλεσης, ώστε να υπάρχει η μέγιστη δυνατή παραλληλία. Αυτό επιτυγχάνεται με την σταδιακή αύξηση των μπλοκ της εφαρμογής και έχει ιδιαίτερα σημαντικό αντίκτυπο, καθώς όπως είναι φανερό από τα Σχήματα 5-18 έως 5-23, από τα 16 μπλοκ μέχρι και τα 64 παρατηρείται πολύ μεγάλη επιτάχυνση. Από αυτόν τον αριθμό μπλοκ και πάνω φαίνεται ότι υπάρχουν αρκετά μπλοκ προς επεξεργασία, και κατά συνέπεια έχουν μοιραστεί ικανοποιητικά οι απαιτούμενες λειτουργίες της εφαρμογής. Έτσι, μετά από εκείνο το σημείο, μπορεί κανείς να αναλωθεί σε πιο λεπτομερειακές ρυθμίσεις αύξησης της επίδοσης.
- Τα παραπάνω συμπεράσματα για το ποσοστό χρησιμοποίησης αφορούν περισσότερο την έκδοση R6. Αν στραφεί η προσοχή στον R5 τώρα, βλέπει κανείς ότι, από τη στιγμή που υπάρχει ικανός συνολικός αριθμός νημάτων, πάντα η επίδοση των περιπτώσεων με μέγεθος μπλοκ ίσο με 128, είναι πάντοτε καλύτερη αυτής των 64 νημάτων/μπλοκ. Αυτό εξηγείται από τη διαφορά του ποσοστού χρησιμοποίησης το οποίο είναι καλύτερο στην περίπτωση των 128 νημάτων/μπλοκ (στο Σχήμα 5-24, 83% για τα 128 και 67% για τα 256) ανεξάρτητα με τις διαφορές σε αριθμό «φουρνιών», καθώς στη συγκεκριμένη περίπτωση, το πλεονέκτημα στη χρησιμοποίηση ξεκαθαρίζει το τοπίο όσον αφορά την καλύτερη επιλογή μεγέθους μπλοκ.
- Τέλος, αν παρατηρηθεί το Σχήμα 5-26, στο οποίο υπάρχουν οι απαιτήσεις κάθε νήματος σε καταχωρητές και σε διαμοιραζόμενη μνήμη, σε συνδυασμό με το Σχήμα 5-24 των χρησιμοποιήσεων στις διάφορες εκδόσεις της αναγωγής μπορεί αν εξαχθεί ένα συμπέρασμα για ένα επιπλέον λόγο αύξησης της επίδοσης. Αυτός ήταν η μείωση σε απαιτήσεις σε καταχωρητές ανά νήμα, κάτι που επέτρεψε καλύτερη αξιοποίηση του υλικού. Δηλαδή, πέρα από τις βελτιώσεις στο μίγμα των εντολών και τις βελτιστοποιήσεις σε προσβάσεις των διαφόρων μνημών της κάρτας γραφικών, με το να μειώνονται οι απαιτήσεις σε καταχωρητές, εξασφαλίζεται ότι θα γίνονταν εμφανείς οι παραπάνω βελτιστοποιήσεις και δεν θα «κρύβονταν» λόγω αντικτύπου των αλλαγών σε απαιτήσεις πόρων. Επιπλέον, σε κάποιες περιπτώσεις δόθηκε η ευκαιρία για επίτευξη καλύτερων ποσοστών χρησιμοποίησης με συνέπεια οι βελτιώσεις να ενισχυθούν περαιτέρω.

Απαιτήσεις σε πόρους	Έκδοση αλγορίθμου							
	R1	R2	R3	R3.5	R4	R4.5	R5	R6
Shared Μνήμη (bytes)	40	32	32 + (4 x BLOCK_SIZE)	32 + (4 x BLOCK_SIZE)	32 + (4 x BLOCK_SIZE)	32 + (4 x BLOCK_SIZE)	32 + (4 x BLOCK_SIZE)	32 + (4 x BLOCK_SIZE)
Αριθμός καταχωρητών	13	11	14	14	11	11	11	10

Σχήμα 5-26 Απαιτήσεις όλων των εκδόσεων του αλγορίθμου της αναγωγής σε καταχωρητές ανά νήμα και σε bytes διαμοιραζόμενης μνήμης ανά μπλοκ. BLOCK_SIZE είναι το μέγεθος του μπλοκ στις αρχικές παραμέτρους εκτέλεσης της εφαρμογής.

5.4 Πολλαπλασιασμός Πυκνού Πίνακα με Διάνυσμα

Στην παρούσα ενότητα θα επιχειρηθεί η υλοποίηση αποδοτικού αλγορίθμου για τον πολλαπλασιασμό πυκνού πίνακα επί διάνυσμα (DMV – Dense Matrix Vector multiplication), με χρήση του περιβάλλοντος προγραμματισμού CUDA πάνω σε μια κάρτα γραφικών της αρχιτεκτονικής G80 (GeForce 8800 Ultra). Ο ΠΠΠΔ (Πολλαπλασιασμός Πυκνού Πίνακα επί Διάνυσμα) αποτελεί έναν ευρέως χρησιμοποιούμενο πυρήνα σε επιστημονικές και ερευνητικές εφαρμογές. Η ενδεχόμενη επιτάχυνσή του με χρήση της πλατφόρμας που μελετάται, μπορεί να έχει ως αποτέλεσμα σημαντική μείωση του χρόνου περάτωσης των εφαρμογών αυτών.

Ο ΠΠΠΔ αποτελεί έναν αλγόριθμο ο οποίος μπορεί να παραλληλοποιηθεί, δηλαδή οι απαραίτητες για την ολοκλήρωσή του λειτουργίες, μπορούν να μοιραστούν σε πολλούς διαφορετικούς πυρήνες. Οι κάρτες γραφικών της αρχιτεκτονικής G80 με τους πολλούς πυρήνες και τα χιλιάδες νήματα εκτέλεσης που υποστηρίζουν, μοιάζουν σαν μια πολύ ταιριαστή πλατφόρμα για την αξιοποίηση του παραλληλισμού του ΠΠΠΔ.

Ο ΠΠΠΔ ενός NxM πίνακα με ένα Mx1 διάνυσμα έχει ως αποτέλεσμα ένα Nx1 διάνυσμα. Απαιτούνται συνολικά 2xNxM πράξεις, εκ των οποίων οι μισές είναι πολλαπλασιασμοί και οι άλλες μισές προσθέσεις. Ουσιαστικά δηλαδή χρειάζονται NxM πράξεις MAD. Κάθε στοιχείο του αρχικού πίνακα συμμετέχει σε μια και μόνο πράξη πολλαπλασιασμού, ενώ κάθε στοιχείο του αρχικού διανύσματος χρησιμοποιείται N φορές, συμμετέχει δηλαδή σε N πράξεις πολλαπλασιασμού. Οι τρόποι διαμοιρασμού των λειτουργιών σε πολλά νήματα μοιάζουν αρκετοί. Είναι δυνατόν σε κάθε νήμα να ανατεθεί ένα στοιχείο (έστω το i-οστό) του αρχικού διανύσματος και αυτό να πολλαπλασιαστεί με κάθε στοιχείο της i-οστής στήλης του πυκνού πίνακα. Εναλλακτικά, μπορούν k διαφορετικά νήματα εκτέλεσης να αναλάβουν τον υπολογισμό τού κάθε στοιχείο του τελικού διανύσματος (έχοντας έτσι συνολικά kxN νήματα εκτέλεσης, με κάθε νήμα να αναλαμβάνει M/k πράξεις MAD). Διαφορετικά, είναι δυνατόν να ανατίθεται σε κάθε νήμα εκτέλεσης ο υπολογισμός p στοιχείων του τελικού διανύσματος (έχοντας συνολικά N/p νήματα εκτέλεσης, με κάθε νήμα να εκτελεί pxM πράξεις MAD). Ωστόσο, μετά από πειραματισμούς, εξήχθη το συμπέρασμα ότι ο καλύτερος τρόπος εκμετάλλευσης του παραλληλισμού του ΠΠΠΔ είναι η ανάθεση των (M) πράξεων MAD που αντιστοιχούν σε κάθε στοιχείο του τελικού διανύσματος σε ξεχωριστό νήμα. Οι δυο κυριότεροι λόγοι ήταν οι εξής: α)η ανάγκη για μεγάλο αριθμό νημάτων ώστε να γίνεται χρήση μεγάλου, κατά το δυνατόν, ποσοστού των δυνατοτήτων της αρχιτεκτονικής της διαθέσιμης κάρτας γραφικών, και β)η αποφυγή των προβληματικών, όπως αποδείχθηκαν, ταυτόχρονων προσβάσεων στη ίδια θέση της καθολικής μνήμης, πράγμα αναπόφευκτό όταν διαφορετικά νήματα είχαν να συνεισφέρουν κομμάτια του συνολικού αποτελέσματος (σε αυτό το θέμα λύση δίνουν οι ατομικές

λειτουργίες που υποστηρίζει η έκδοση 1.1 του CUDA, αλλά η GeForce 8800 Ultra υποστηρίζει μέχρι και την έκδοση 1.0).

5.4.1 Παρουσίαση και πειραματική σύγκριση χρησιμοποιούμενων εκδόσεων αλγορίθμου του ΠΠΠΔ

Στην παρούσα υποενότητα, θα παρουσιαστούν τα χαρακτηριστικά των εκδόσεων που χρησιμοποιήθηκαν σε παρακάτω πειράματα για την υλοποίηση του ΠΠΠΔ με το περιβάλλον προγραμματισμού του CUDA. Χρησιμοποιήθηκαν συνολικά οκτώ διαφορετικοί πυρήνες με κλιμακούμενες βελτιώσεις, με σκοπό την παρατήρηση των επιπτώσεων της εφαρμογής της κάθε στρατηγικής βελτιστοποίησης, ξεχωριστά.

Ο αρχικός πυρήνας *DMV_1*, ο οποίος ήταν και ο πιο απλοϊκός και αφελής, αποτελεί ένα μέτρο σύγκρισης για τους επόμενους. Έκανε χρήση μόνο της καθολικής μνήμης, με κάθε νήμα να αναλαμβάνει όλους τους υπολογισμούς που σχετίζονται με ένα στοιχείο του τελικού διανύσματος (συνολικά χρήση N νημάτων εκτέλεσης). Σε κάθε κλήση του, απαιτούνταν M πράξεις MAD ανά νήμα, με το αποτέλεσμα της κάθε πράξης να προστίθεται στη θέση της καθολικής μνήμης που αντιστοιχεί στο επεξεργαζόμενο στοιχείο του τελικού διανύσματος. Οι προσβάσεις στην καθολική μνήμη για τα στοιχεία του πίνακα δεν ήταν συγχωνευμένες, καθώς διαδοχικά νήματα εκτέλεσης σε ένα ημι-στημόνι, προσπελάζουν στοιχεία που απέχουν M θέσεις (ο πίνακας είχε απεικονιστεί σε μια διάσταση κατά γραμμές). Επίσης, οι προσβάσεις στην καθολική μνήμη για τα στοιχεία του αρχικού διανύσματος δεν είναι συγχωνευμένες, καθώς την ίδια χρονική στιγμή, όλα τα νήματα εκτέλεσης προσπελάζουν την ίδια θέση μνήμης (καθώς στο i-οστό βήμα γίνεται ο πολλαπλασιασμός του i-οστού στοιχείου του αρχικού διανύσματος με το στοιχείο που η θέση του στον πίνακα είναι στην γραμμή που έχει αναλάβει το νήμα εκτέλεσης και στην i-οστή στήλη). Αντίθετα, συγχωνευμένες είναι οι προσπελάσεις στις θέσεις του τελικού διανύσματος, καθώς με τον τρόπο που επιλέχτηκε να δοθούν αύξοντες αριθμοί στα νήματα εκτέλεσης ως προς συνολικά την εφαρμογή, εξασφαλίζεται ότι διαδοχικά νήματα σε ένα στημόνι, έχουν διαδοχικούς αύξοντες αριθμούς.

Ο πυρήνας *DMV_2*, εισήγαγε τη χρήση ενός συσσωρευτή (accumulator) σε κάθε νήμα, με σκοπό την μείωση των προσπελάσεων της θέσης του στοιχείου τελικού διανύσματος (που είχε αναλάβει το κάθε νήμα) στην καθολική μνήμη. Ο συσσωρευτής στην ουσία είναι μια μεταβλητή που εδρεύει σε ένα καταχωρητή. Σε αυτήν την μεταβλητή συγκεντρώνεται το άθροισμα των πράξεων που συντελούνται από κάθε νήμα, και στο τέλος του πυρήνα, η αποθηκευμένη στον συσσωρευτή τιμή, μεταφέρεται (με συγχωνευμένες προσπελάσεις) στην θέση του τελικού διανύσματος στην καθολική μνήμη.

Στους προηγούμενους πυρήνες, σημαντικό πρόβλημα αποτελούσαν οι μη συγχωνευμένες προσβάσεις στην καθολική μνήμη για τα στοιχεία του πίνακα και του αρχικού διανύσματος-«πολλαπλασιαστή». Ο πυρήνας *DMV_3* αντιμετώπισε τις πρώτες με την απεικόνιση του πυκνού πίνακα σε μια διάσταση κατά στήλες, σε αντίθεση με την απεικόνισης κατά γραμμές την οποία χρησιμοποιούσαν οι προηγούμενες εκδόσεις. Αυτό επιτευχθεί με αναστροφή του αρχικού πυκνού πίνακα και απεικόνισή του κατά γραμμές, με το αποτέλεσμα να είναι ουσιαστικά ο αρχικός πίνακας απεικονιζόμενος κατά στήλες. Έτσι, πλέον, διαδοχικά νήματα εντός ενός στημονιού προσπελάζουν

διαδοχικές θέσεις στην καθολική μνήμη, όταν προσκομίζουν στοιχεία του πίνακα στου πυρήνες της κάρτας γραφικών.

Στον πυρήνα *DMV_4*, αντιμετωπίσαμε τις μη συγχωνευμένες προσπελάσεις στην καθολική μνήμη για τα στοιχεία του αρχικού διανύσματος. Αυτό έγινε με τη χρήση της διαμοιραζόμενης μνήμης. Όπως έγινε φανερό παραπάνω, κάθε νήμα θα προσπελάσει τελικά όλα τα στοιχεία του αρχικού διανύσματος (ένα σε κάθε βήμα από τα Μ απαιτούμενα συνολικά) προκειμένου να εκτελέσει τις απαραίτητες πράξεις για τον υπολογισμό ενός στοιχείου του τελικού διανύσματος. Σε αυτήν την έκδοση όλα τα νήματα ενός μπλοκ, συνεργάζονται για να μεταφέρουν το αρχικό διάνυσμα στη διαμοιραζόμενη μνήμη. Το διάνυσμα, όντας στη διαμοιραζόμενη μνήμη του μπλοκ, μπορεί να χρησιμοποιηθεί από όλα τα νήματα εκτέλεσης του συγκεκριμένου μπλοκ. Προσέχοντας οι προσπελάσεις κατά την «φόρτωση» του αρχικού διανύσματος στη διαμοιραζόμενη μνήμη να είναι συγχωνευμένες, γλιτώνουμε αρκετό χρόνο. Αυτό συμβαίνει γιατί πλέον κάθε νήμα αντί να φορτώνει σταδιακά όλο το αρχικό διάνυσμα (ένα στοιχείο σε κάθε βήμα υπολογισμού), φορτώνει ένα κλάσμα του συνολικού (συγκεκριμένα το 1/μέγεθος_του_μπλοκ) και με συγχωνευμένες προσπελάσεις, έναντι των μη συγχωνευμένων των προηγούμενων εκδόσεων. Πλέον, οι προσπελάσεις των στοιχείων του αρχικού διανύσματος γίνονται στη διαμοιραζόμενη μνήμη, κάτι που γλιτώνει σημαντικό χρόνο για την εφαρμογή. Επιπροσθέτως, επειδή σε κάθε βήμα όλα τα νήματα ενός στημονιού (και όλης της εφαρμογής γενικά) προσπελάζουν το ίδιο στοιχείο του αρχικού διανύσματος από την διαμοιραζόμενη μνήμη, αποφεύγονται οι διαμάχες πρόσβασης στη διαμοιραζόμενη μνήμη, λόγω του μηχανισμού εκπομπής. Όμως αυτή η έκδοση παρουσιάζει ένα πολύ σημαντικό μειονέκτημα. Λόγω του περιορισμού του μεγέθους της διαμοιραζόμενης μνήμης, μπορεί να «χωρέσει» μέχρι και κάτι λιγότερο από 4Κ αριθμούς κινητής υποδιαστολής μονής ακρίβειας, και σαν αποτέλεσμα αποτυγχάνει να ξεκινήσει την εκτέλεση του αλγορίθμου του ΠΠΠΔ για μεγέθη διανύσματος μεγαλύτερα από 4Κ αριθμούς κινητής υποδιαστολής.

Για να αντιμετωπίσει το πρόβλημα που αντιμετωπίζει ο DMV_4 όσον αφορά τα μεγέθη προβλημάτων που μπορεί να υποδεχτεί, ο πυρήνας *DMV_5* λειτουργεί ως εξής: αντί να μεταφέρει όλο το διάνυσμα με τη μια στην διαμοιραζόμενη μνήμη, το κόβει σε κομμάτια μεγέθους ίσου με το μέγεθος του μπλοκ και τα φορτώνει ένα-ένα στη διαμοιραζόμενη μνήμη που αντιστοιχεί σε κάθε μπλοκ. Όταν φορτώνεται ένα κομμάτι του αρχικού διανύσματος, ακολουθούν οι πράξεις στις οποίες συμμετέχουν τα στοιχεία αυτού του κομματιού. Μετά από συγχρονισμό των νημάτων σε επίπεδο μπλοκ, φορτώνεται το επόμενο κομμάτι του αρχικού διανύσματος στις θέσεις της διαμοιραζόμενης μνήμης, αντικαθιστώντας το προηγούμενο. Πλέον, ο νέος πυρήνας μπορεί να αντιμετωπίσει οποιοδήποτε μέγεθος προβλήματος.

Ένα πρόβλημα όμως του DMV_5, το οποίο θα έρθει στην επιφάνεια μόλις εξεταστούν τα ποσοστά χρησιμοποίησης των εκδόσεων που μελετώνται, είναι οι υψηλές απαιτήσεις σε καταχωρητές. Προκειμένου να εξαλειφθεί εν μέρει αυτός ο περιοριστικός παράγοντας, στην έκδοση *DMV_6*, ένας αριθμός από μεταβλητές που είναι ίδιες για όλα τα νήματα (αριθμός κομματιών στα οποία θα χωριστεί το αρχικό διάνυσμα, μέγεθος του κάθε κομματιού κτλ.) προϋπολογίστηκαν και περάστηκαν σαν παράμετροι. Αποτέλεσμα του παραπάνω, ήταν η μείωση των αιτήσεων κάθε νήματος σε αριθμό καταχωρητών, καθώς και η ελαφριά μείωση των πράξεων σε κάθε νήμα (λόγω του προϋπολογισμού κάποιων μεταβλητών εκτός του πυρήνα, στη μεριά του host).

Με σκοπό την περαιτέρω επιτάχυνση του αλγορίθμου του ΠΠΠΔ, στην έκδοση *DMV_7*, ξετυλίξαμε για κάποιες τιμές τους βρόχους μεταφοράς και υπολογισμού των στοιχείων του τελικού διανύσματος. Ο μεταγλωττιστής δε δεχόταν την εντολή `#pragma unroll` για το σύνθετο βρόχο της εφαρμογής και το ξετύλιγμα έγινε με το χέρι μέχρι και την περίπτωση χωρισμού του αρχικού διανύσματος σε 32 κομμάτια (περιπτώσεις με διαχωρισμό του αρχικού διανύσματος σε περισσότερα κομμάτια είναι μη πρακτική καθώς καταλήγει σε ιδιαίτερα μεγάλου μεγέθους κώδικα). Για περιπτώσεις όπου ο χωρισμός γίνεται σε περισσότερα κομμάτια χρησιμοποιείται η έκδοση DMV_6.

Παρακάτω παρατίθενται οι κώδικες των παρουσιαζόμενων υπολογιστικών πυρήνων, για λόγους πληρότητας. Σε συνδυασμό με τα προηγούμενα σχόλια, ο αναγνώστης μπορεί να αντιληφθεί με λεπτομέρεια τη διαδοχή των ενεργειών των πυρήνων αυτών.

DMV_1

```
__global__ void dmv_1(float *Matrix, float *Vector, float *Result,
unsigned long size_of_vec, unsigned long size_of_res)
{
      unsigned long i;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;

      if (id < size_of_res)
      {
            Result[id]=0.0f;

            for (i = 0; i < size_of_vec; i++)
                  Result[id] +=  Matrix[id * size_of_vec  +  i] *
Vector[i];
      }

      return;
}
```

DMV_2

```
__global__ void dmv_2(float *Matrix, float *Vector, float *Result,
unsigned long size_of_vec, unsigned long size_of_res)
{
      unsigned long i;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;
      float temp=0.0f;

      if (id < size_of_res)
            {
                  for (i = 0; i < size_of_vec; i++)
                        temp +=  Matrix[id * size_of_vec  +  i] *
Vector[i];
                  Result[id]=temp;
            }

      return;
}
```

DMV_3

```
__global__ void dmv_3(float *Matrix, float *Vector, float *Result,
unsigned long size_of_vec, unsigned long size_of_res)
{
      unsigned long i;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;
      float temp=0.0f;

      if (id < size_of_res)
            {
                  for (i = 0; i < size_of_vec; i++)
                        temp +=   Matrix[i * size_of_res  +  id] *
Vector[i];
                  Result[id]=temp;
            }

      return;
}
```

DMV_4

```
__global__ void dmv_kernel_3(float *Matrix, float *Vector, float
*Result, unsigned long size_of_vec, unsigned long size_of_res)
{
      //unsigned long i;
      int j;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;
      extern __shared__ float shared_vector[];
      float temp=0.0f;

      if (id < size_of_res)
            {
                  for   (j   =   threadIdx.x;   j   <   size_of_vec;
j+=blockDim.x)
                        shared_vector[j]=Vector[j];

                  __syncthreads();

                  for (j = 0; j < size_of_vec; j++)
                        temp +=   Matrix[j * size_of_res  +  id] *
shared_vector[j];

                  Result[id]=temp;

            }
      return;
}
```

DMV_5

```
__global__ void dmv_5(float *Matrix, float *Vector, float *Result,
unsigned long size_of_vec, unsigned long size_of_res)
{
      int i,j;
      unsigned long iterations;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;
      extern __shared__ float shared_vector[];
      float temp=0.0f;
```

```
      iterations= size_of_vec / blockDim.x ;

      if (id < size_of_res)
      {
            for (j = 0; j < iterations ; j++)
            {
                  shared_vector[threadIdx.x]=Vector[threadIdx.x   +
j * blockDim.x ];
                  __syncthreads();
                  for (i = 0; i <  blockDim.x; i++)
                        temp +=    Matrix[  (i + j * blockDim.x)  *
size_of_res  +  id] * shared_vector[i];
                  __syncthreads();
            }
            Result[id]=temp;
      }
      return;
}
```

DMV_6

```
__global__ void dmv_6(float *Matrix, float *Vector, float *Result,
unsigned  long  size_of_vec,  unsigned  long  size_of_res,  unsigned
long iterations)
{
      int i,j;
      unsigned long id = blockIdx.x * blockDim.x + threadIdx.x ;
      extern __shared__ float shared_vector[];
      float temp=0.0f;

      if (id < size_of_res)
      {
            for (j = 0; j < iterations ; j++)
            {
                  shared_vector[threadIdx.x]=Vector[threadIdx.x   +
j * blockDim.x ];
                  __syncthreads();
                  for (i = 0; i <  blockDim.x; i++)
                              temp +=    Matrix[  (i  +  j  *  block-
Dim.x) * size_of_res  +  id] * shared_vector[i];
                  __syncthreads();
}

            Result[id]=temp;
      }
      return;
}
```

Η έκδοση DMV_7, όπως αναφέρθηκε και νωρίτερα, είναι όμοια με την DMV_6, με τη διαφορά ότι περιλαμβάνει πολλούς πυρήνες με πλήρως ξετυλιγμένους τους βρόχους των επαναλήψεων, ανάλογα με τα κομμάτια στα οποία χωρίζεται το αρχικό διάνυσμα.

Σε επόμενα πειράματα χρησιμοποιήθηκε και ο τελικός πυρήνας, ο DMV_8 (ο οποίος αποτελεί μια παραλλαγή των DMV_6 και DMV_7 και του οποίου η διαφοροποίησή

θα εξηγηθεί σε επόμενη υποενότητα), ωστόσο κρίνεται σκόπιμο να παρατεθούν οι συγκριτικές δοκιμές των μέχρι στιγμής παρουσιασμένων εκδόσεων για να γίνει φανερή η επίδραση των διαφόρων στρατηγικών βελτιστοποίησης που χρησιμοποιήθηκαν σε αυτούς. Στο Σχήμα 5-27 συνοψίζονται οι στρατηγικές βελτιστοποίησης των επτά εξεταζόμενων πυρήνων, οι απαιτήσεις τους σε διαμοιραζόμενη μνήμη και καταχωρητές και τα ποσοστά χρησιμοποίησης για διάφορα μεγέθη μπλοκ.

	Έκδοση πυρήνα						
	DMV_1	DMV_2	DMV_3	DMV_4	DMV_5	DMV_6	DMV_7
Βελτιστοποιήσεις							
Χρήση συσσωρευτή για συλλογή ενδιάμεσων αποτελεσμάτων	όχι	ναι	ναι	ναι	ναι	ναι	ναι
Χρήση διαιρετής δυναμής υποδιαστολής	όχι	όχι	όχι	ναι	ναι	ναι	ναι
Συγχωνευμένες προσβάσεις στην καθολική μνήμη(για τον πυκνό πίνακα)	όχι	όχι	ναι	ναι	ναι	ναι	ναι
Συγχωνευμένες προσβάσεις στην καθολική μνήμη(για το αρχικό διάνυσμα)	όχι	όχι	όχι	ναι	ναι	ναι	ναι
Αποφυγή διαμοιρασμένων κατά την πρόσβαση στη διαμοιρασμένη μνήμη	όχι	όχι	όχι	ναι	ναι	ναι	ναι
Πολλές φορτίσεις ανά φορέα κομματιών του αρχικού διανύσματος	-	-	-	όχι	ναι	ναι	ναι
Προπαλλοποίηση σταθερών για λιγότερες πράξεις εντός νημάτων και για μείωση απαιτήσεων σε καταχωρητές	όχι	όχι	όχι	όχι	ναι	ναι	ναι
Ξετύλιγμα βρόχων	-	-	-	-	όχι	όχι	ναι
Απαιτήσεις πόρων							
Απαιτήσεις σε διαμοιραζόμενη μνήμη (by	56	56	64	64 + VEC_SZ * 4	64 + BLK_SZ * 4	64 + BLK_SZ * 4	64 + BLK_SZ * 4
Απαιτήσεις σε καταχωρητές (ανά νήμα)	9	9	10	12	17	15	10(max)/15(εκτός ορίων)
Μέγιστη χρησιμοποίηση (%)							
Μέγεθος μπλοκ							
512	66,7%	66,7%	66,7%	66,7%	0,0%	66,7%	66,7%
256	100,0%	100,0%	100,0%	33,3%	33,3%	66,7%	100,0%
128	100,0%	100,0%	100,0%	16,6%	50,0%	66,7%	100,0%
64	66,7%	66,7%	66,7%	8,3%	58,3%	66,7%	66,7%
32	33,3%	33,3%	33,3%	4,2%	29,2%	33,3%	33,3%

Σχήμα 5-27 Πίνακας χαρακτηριστικών εκδόσεων των διαφορετικών υλοποιήσεων του αλγορίθμου του ΠΠΠΔ. VEC_SZ είναι το μέγεθος του αρχικού διανύσματος και BLK_SZ το μέγεθος του μπλοκ.

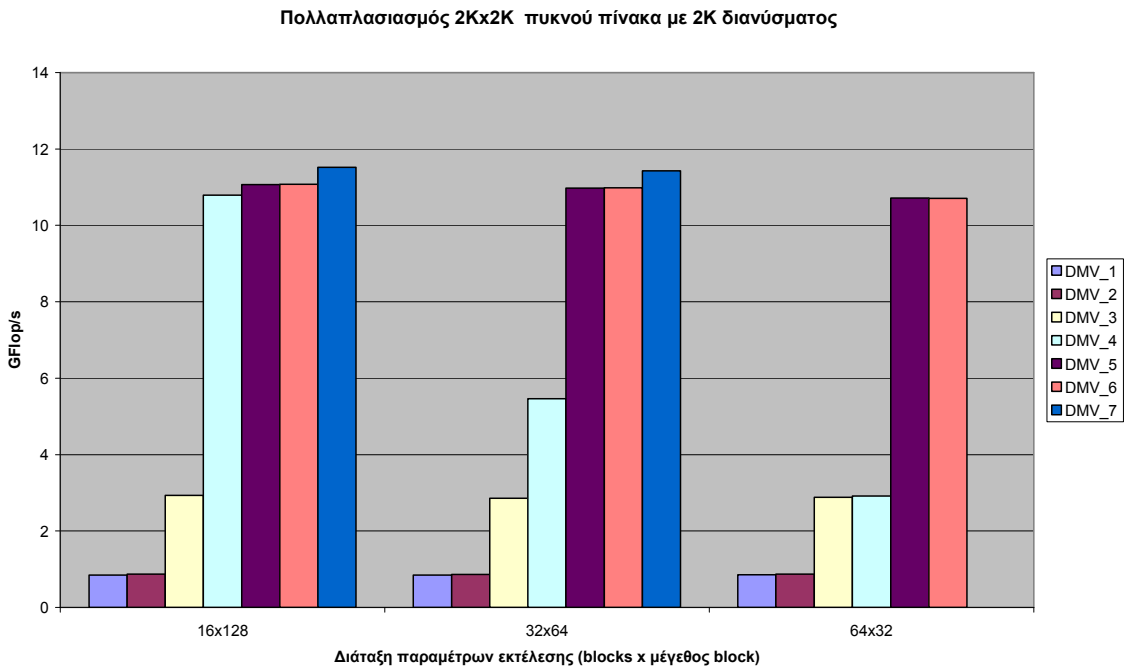
Παρατηρώντας το Σχήμα 5-27 υπάρχουν τρία σημεία τα οποία χρίζουν προσοχής και είναι σημειωμένα με κάποιο χρώμα:

- Στα διαγραμμισμένα πεδία της στήλης DMV_4 υπάρχουν οι απαιτήσεις του πυρήνα DMV_4 σε bytes διαμοιραζόμενης μνήμης, οι οποίες όταν ξεπερνούν τα 16 KB, ο πυρήνας δεν είναι δυνατόν να εκτελεστεί και αποτυγχάνει να παρέχει αποτέλεσμα αυτή η έκδοση (σε τέτοιες περιπτώσεις η χρησιμοποίηση είναι 0% αντί για τις σημειωμένες στο σχήμα, για όλα τα μεγέθη μπλοκ).
- Στα διαγραμμισμένα πεδία της στήλης DMV_5 υπάρχουν οι απαιτήσεις του πυρήνα DMV_5 σε αριθμό καταχωρητών ανά νήμα, οι οποίες είναι ιδιαίτερα αυξημένες (17 καταχωρητές ανά νήμα) . Αυτός είναι ο λόγος που η χρησιμοποίηση είναι 0% για μέγεθος μπλοκ 512 είναι ότι οι απαιτήσεις ενός τέτοιου μπλοκ ξεπερνούν τους 8192 καταχωρητές (καθώς 512x17=8704). Έτσι, ο πυρήνας για τέτοιο μέγεθος μπλοκ δεν μπορεί να εκτελεστεί.
- Στον DMV_7 οι απαιτήσεις σε καταχωρητές ανά νήμα (στο διαγραμμισμένο πεδίο της αντίστοιχης στήλης) εξαρτώνται από το σε πόσα κομμάτια θα χωρίσουμε τελικά το αρχικό διάνυσμα ώστε να το φέρουμε τμηματικά στη διαμοιραζόμενη μνήμη. Αν ο αριθμός αυτών των κομματιών είναι ίσος ή μικρότερος με 32 (στις περιπτώσεις δηλαδή που ξετυλίχτηκε με το χέρι ο βρόχος του αλγορίθμου), οι απαιτήσεις είναι το πολύ 10 καταχωρητές ανά νήμα (κάτι που σημαίνει ότι δεν αποτελούν περιοριστικό παράγοντα για το ποσοστό χρησιμοποίησης) και τα ισχύοντα ποσοστά χρησιμοποίησης είναι αυτά της στήλη του DMV_7. Σε αντίθετη περίπτωση, καλείται ο πυρήνας DMV_6 και ισχύουν οι χρησιμοποιήσεις της στήλης του DMV_6, καθώς τότε οι απαιτήσεις σε καταχωρητές είναι 15 και όχι 10.

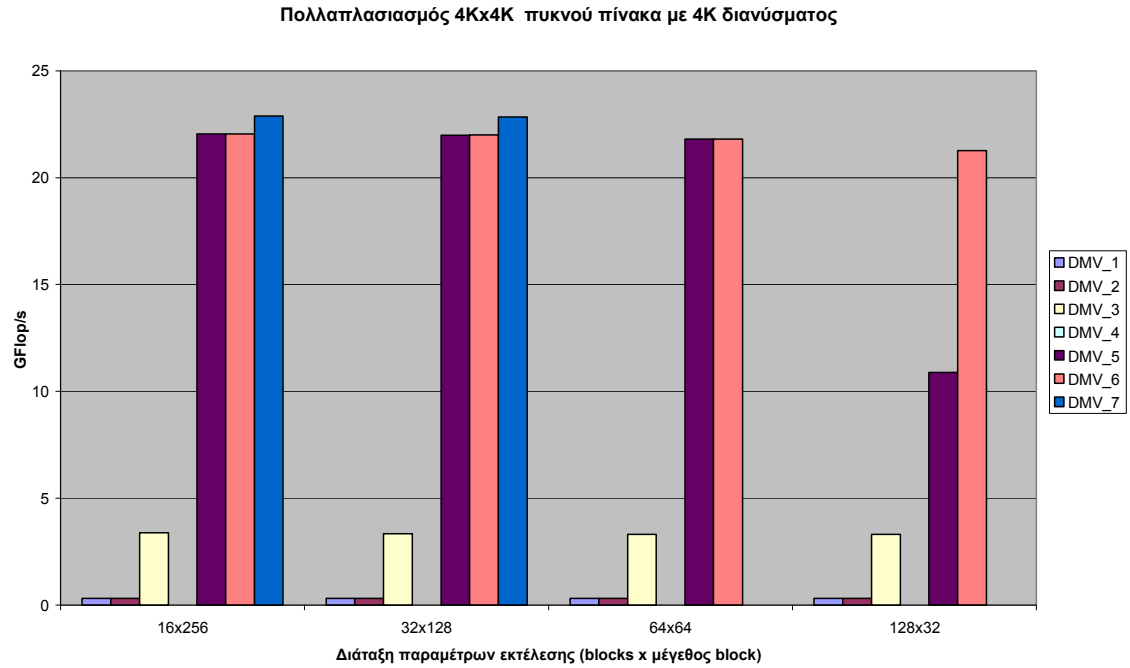
Για λόγους σύγκρισης, αναφέρεται ότι η CPU του συστήματος που χρησιμοποιήθηκε διεκπεραίωνε τον αλγόριθμο του ΠΠΠΔ με μέγιστη ταχύτητα που δεν ξεπερνούσε τα 670 MFlop/s για μεγάλα μεγέθη πίνακα και διανύσματος. Χρησιμοποιήθηκε η σημαία –O3 κατά τη μεταγλώττιση, για τη βελτιστοποίηση του παραγόμενου κώδικα. Επίσης, δοκιμάστηκαν εκδόσεις του αλγορίθμου με χρήση του συνόλου εντολών SSE, και η επίδοση κυμάνθηκε μεταξύ 840 MFlop/s (row-wise alignment) και 1.24 GFlop/s (column-wise alignment).

5.4.2 Πρώτη σειρά πειραμάτων για τη μέτρηση της επίδοσης των εκδόσεων του αλγορίθμου του ΠΠΠΔ

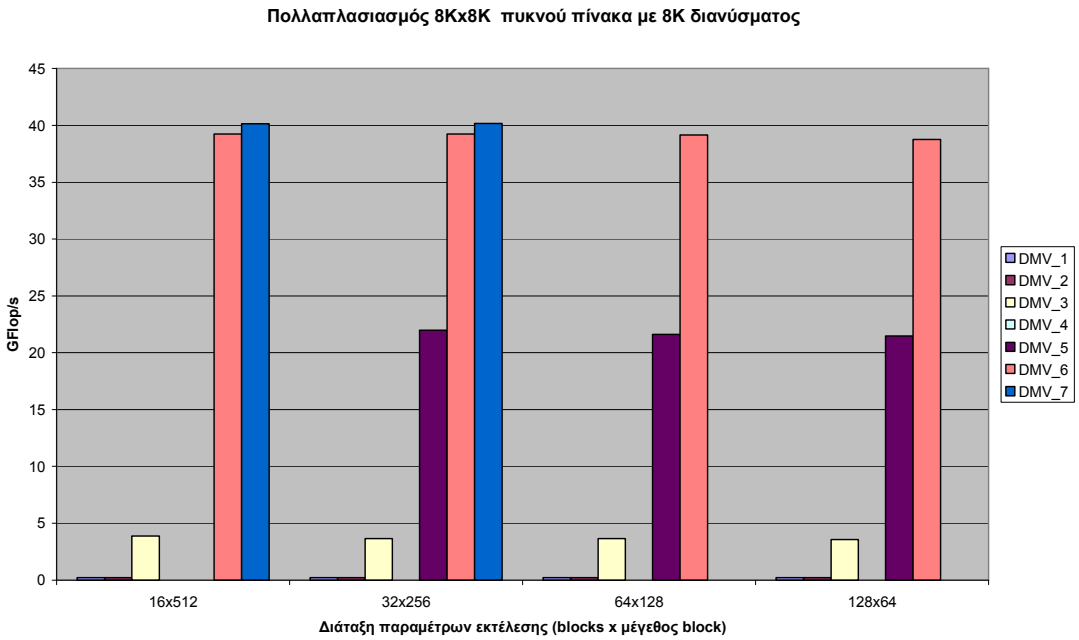
Σε αυτήν την ενότητα θα παρουσιαστούν τα αποτελέσματα πειραμάτων μέτρησης της επίδοσης (με μετρική την ταχύτητα εκτέλεσης πράξεων κινητής υποδιαστολής) για διάφορα μεγέθη προβλημάτων ΠΠΠΔ, με όλες τις διαθέσιμες εκδόσεις. Με τη βοήθεια του Σχήματος 5-26 και ιδιαίτερα της σύνοψης του βελτιστοποιήσεων και των ποσοστών χρησιμοποίησης, θα εξηγηθούν οι διαφορές στην επίδοση μεταξύ των διαφόρων εκδόσεων. Κάθε πείραμα διεξήχθη 128 φορές και οι τιμές τις ταχύτητας σε GFlop/s αποτελούν τον μέσο όρο των πειραμάτων. Έγιναν μετρήσεις για επτά διαφορετικά προβλήματα ΠΠΠΔ, με διάφορες επιλεγμένες παραμέτρους εκτέλεσης το κάθε ένα. Στα Σχήματα 5-28 έως 5-34 παρουσιάζονται τα αποτελέσματα των μετρήσεων.



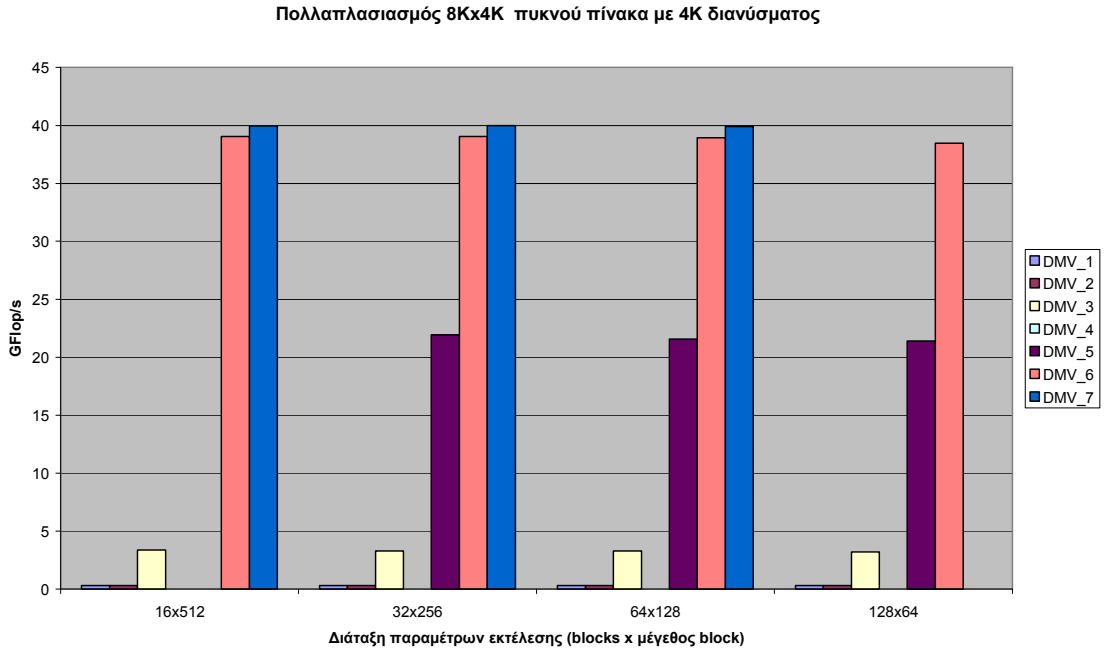
Σχήμα 5-28 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 2K x 2K στοιχείων `float` με διάνυσμα μεγέθους 2K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



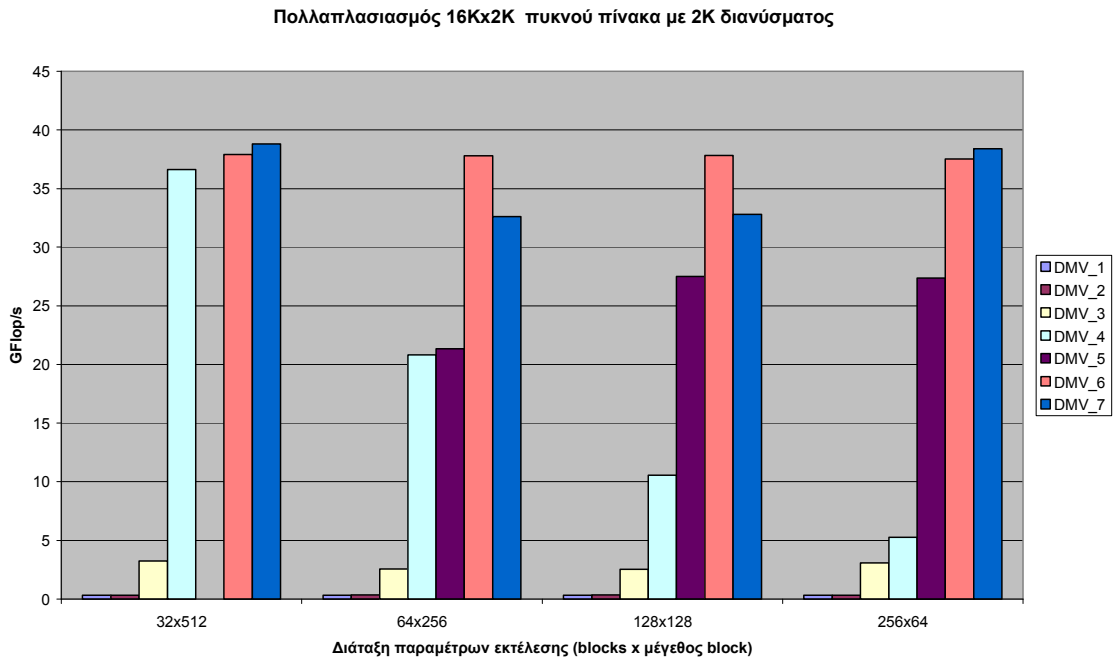
Σχήμα 5-29 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 4K x 4K στοιχείων `float` με διάνυσμα μεγέθους 4K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



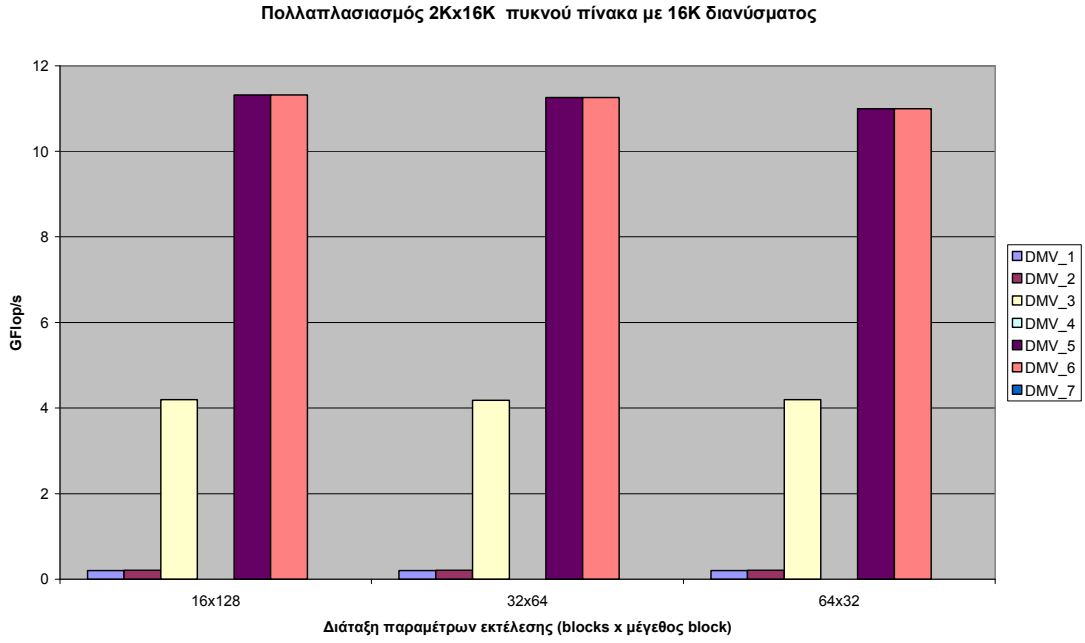
Σχήμα 5-30 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 8K x 8K στοιχείων `float` με διάνυσμα μεγέθους 8K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



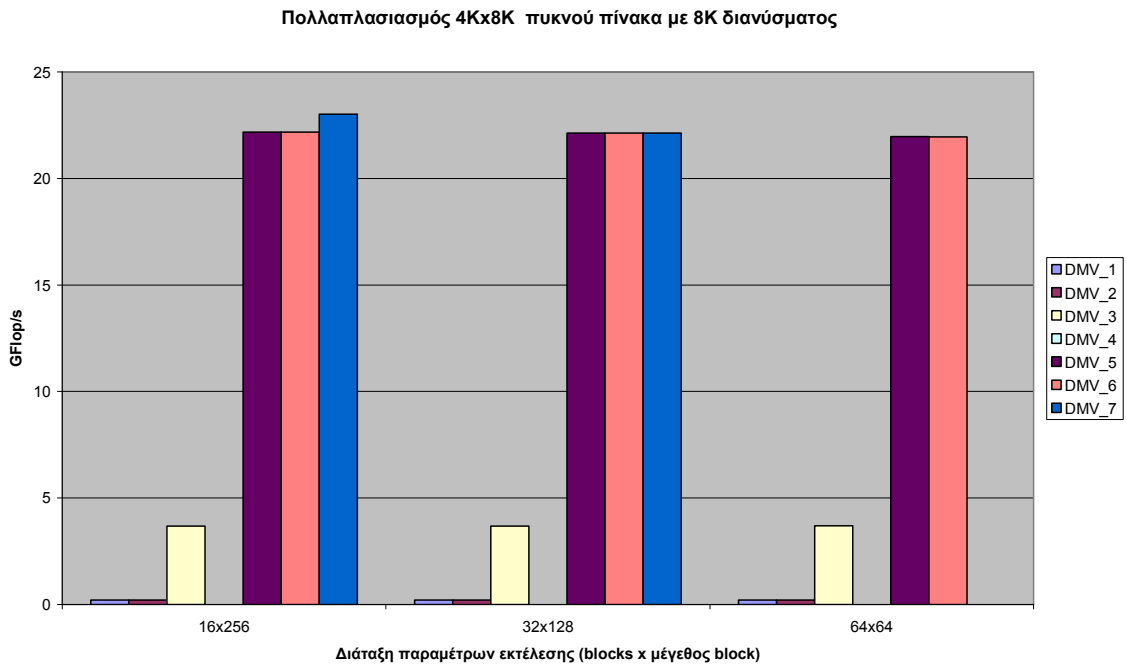
Σχήμα 5-31 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 8K x 4K στοιχείων `float` με διάνυσμα μεγέθους 4K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



Σχήμα 5-32 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 16K x 2K στοιχείων `float` με διάνυσμα μεγέθους 2K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



Σχήμα 5-33 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 2K x 16K στοιχείων `float` με διάνυσμα μεγέθους 16K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.



Σχήμα 5-34 Πρόβλημα πολλαπλασιασμού πυκνού πίνακα μεγέθους 4K x 8K στοιχείων `float` με διάνυσμα μεγέθους 8K `float` με διάφορες εκδόσεις και παραμέτρους εκτέλεσης.

Κάποια σημαντικά στοιχεία για τη σωστή ερμηνεία των διαγραμμάτων των Σχημάτων 5-28 έως 5-34 είναι τα εξής:

- Όταν απουσιάζει μέτρηση της ταχύτητας του πυρήνα DMV_4, σημαίνει ότι στη συγκεκριμένη περίπτωση δεν ήταν δυνατή η εκτέλεση λόγω του ότι δεν «χωρούσε» στη διαμοιραζόμενη μνήμη. Είναι φανερό ότι μόνο σε δύο περιπτώσεις έγινε δυνατή η εκτέλεση του συγκεκριμένου πυρήνα, όταν το αρχικό διάνυσμα είχε μέγεθος 2K `float`. Στην περίπτωση των 4K `float` που αντιστοιχούν σε ακριβώς 16KB διαμοιραζόμενης μνήμης, το αρχικό διάνυσμα δεν «χωράει» λόγω του ότι οι παράμετροι με τους οποίους καλείται ο κάθε πυρήνας, αποθηκεύονται σε θέσεις της διαμοιραζόμενης μνήμης. Όπως φαίνεται και από το Σχήμα 5-26 απαιτούνται 64 bytes για τις παραμέτρους, άρα τελικά υπερβαίνουν οι αιτήσεις τη διαθέσιμη διαμοιραζόμενη μνήμη.
- Όταν απουσιάζει μέτρηση της ταχύτητας του πυρήνα DMV_5, σημαίνει ότι δεν έγινε δυνατή η εκτέλεσή του λόγω του ότι οι απαιτήσεις σε καταχωρητές ξεπέρασαν τους διαθέσιμους. Αυτό συμβαίνει όταν τον καλούμε με παραμέτρους εκτέλεσης που καθορίζουν μέγεθος μπλοκ ίσο με 512 νήματα.
- Όταν απουσιάζει μέτρηση της ταχύτητας του πυρήνα DMV_7, σημαίνει ότι λόγω του ότι ο αριθμός των κομματιών στα οποία διαχωρίζεται το αρχικό διάνυσμα είναι μεγαλύτερος του 32, άρα καλείται ουσιαστικά ο πυρήνας D V_6.

Μελετώντας τα διαγράμματα των Σχημάτων 5-28 έως 5-34 μπορούν σημειωθούν τα εξής:

- Στις εκδόσεις DMV_1 και DMV_2 οι προσβάσεις στην καθολική μνήμη δεν είναι συγχωνευμένες και κυριαρχούν ως προς το ποσοστό του χρόνου εκτέλε-

σης της εφαρμογής που καταλαμβάνουν. Οι ταχύτητες που παρατηρήθηκαν ήταν ιδιαίτερα χαμηλές και καθώς το μέγεθος του προβλήματος αυξανόταν (δηλαδή αυξανόταν ο αριθμός των πράξεων άρα και των προσβάσεων), οι ταχύτητες όλο και μειώνονταν. Αυτό σημαίνει ότι η αρχιτεκτονική G80 δεν κλιμακώνει καθόλου καλά όταν οι προσβάσεις στη καθολική μνήμη δεν είναι συγχωνευμένες.

- Σε σύγκριση με την έκδοση DMV_1, η DMV_2 παρουσιάζει ελαφρή βελτίωση στην ταχύτητα, καθώς μειώνει τις προσβάσεις στην καθολική μνήμη. Όμως, αυτές που εξαλείφονται είναι οι συγχωνευμένες προσβάσεις, οπότε πάλι κυριαρχούν οι μη συγχωνευμένες προσβάσεις, διατηρώντας την ταχύτητα εκτέλεσης στα ίδια χαμηλά επίπεδα.
- Η DMV_3 μετατρέποντας σε συγχωνευμένες τις προσβάσεις στην καθολική μνήμη για τα στοιχεία του πίνακα, αυξάνει την ταχύτητα κατά μια τάξη μεγέθους.
- Η DMV_4, όταν επιτυγχάνεται η εκτέλεσή της, αυξάνεται άλλη μια τάξη μεγέθους την ταχύτητα εκτέλεσης του αλγορίθμου, όταν οι παράμετροι εκτέλεσής της περιλαμβάνει 512 νήματα. Παρατηρείται ότι, στις δυο περιπτώσεις που επιτυγχάνεται η εκτέλεσή της έχουμε σημαντική εξάρτηση της επίδοσης από το μέγεθος του μπλοκ. Αυτό συμβαίνει γιατί ανεξάρτητα από το μέγεθος του μπλοκ, μόνο ένα μπλοκ μπορεί να είναι ενεργό σε SM. Έτσι ανάλογα με το μέγεθος του μπλοκ, και επειδή ο συνολικός αριθμός των νημάτων εκτέλεσης πρέπει να είναι ο ίδιος για ένα συγκεκριμένο πρόβλημα, επηρεάζεται ο αριθμός των μπλοκ που θα περιλαμβάνει το πλέγμα. Αύξηση όμως του συνολικού αριθμού των μπλοκ έχει σαν συνέπεια την αύξηση του αριθμού των «φουρνιών» που θα χρειαστούν. Έτσι, για παράδειγμα, υποδιπλασιάζοντας το μέγεθος του μπλοκ, διπλασιάζεται ο αριθμός των μπλοκ, διπλασιάζονται οι απαιτούμενες «φουρνιές» και τελικά υποδιπλασιάζεται η ταχύτητα της εφαρμογής. Αυτό γίνεται φανερό στις περιπτώσεις των Σχημάτων 5-28 και 5-32 , στις οποίες για κάθε υποδιπλασιασμό του μεγέθους του μπλοκ, υποδιπλασιάζεται η ταχύτητα της εφαρμογής.
- Στην περίπτωση του DMV_5, παρατηρήθηκε σημαντική αύξηση στην ταχύτητα σε σχέση με την DMV_3, που πολλές φορές φτάνει και τον δεκαπλασιασμό. Αυτό είναι αναμενόμενο καθώς μειώνονται σημαντικά οι προσβάσεις στην κύρια μνήμη όσον αφορά τα στοιχεία του αρχικού διανύσματος. Οι προσβάσεις αυτές είναι ξεκάθαρο ότι αποτελούν τη στενωπό επίδοσης του αλγορίθμου, καθώς ο χρόνος που χρειάζονται για να ολοκληρωθούν αποτελεί το μεγαλύτερο μέρος του συνολικού χρόνου εκτέλεσης. Η σημαντική αυτή αύξηση στην ταχύτητα εκτέλεσης οφείλεται στο γεγονός ότι, αντί κάθε νήμα εκτέλεσης να φορτώνει ολόκληρο το αρχικό διάνυσμα, φορτώνει μόνο ένα κλάσμα του συνολικού.
- Η έκδοση DMV_6 παρουσίασε αρκετά μεγαλύτερη ταχύτητα σε σχέση με την DMV_5 που άγγιξε τον διπλασιασμό σε κάποιες περιπτώσεις (Σχήματα 5-30, 5-31 και 5-32), πράγμα που οφείλεται κατά κύριο λόγο στην διαφορά των ποσοστών χρησιμοποίησης. Εφόσον αυτή η έκδοση έχει χαμηλότερες απαιτήσεις σε πόρους, μπορούν ταυτόχρονα να «φιλοξενηθούν» από κάθε SM περισσότερα νήματα, κάτι που έχει αποτέλεσμα τη μείωση των απαιτούμενων «φουρνιών» μπλοκ προς εκτέλεση. Αυτή η συμπεριφορά παρατηρείται κυρίως για τα προβλήματα με μεγάλο αριθμό νημάτων, και δεν έχει σχεδόν καμία επίδραση στα μικρότερα, και πιο συγκεκριμένα στα προβλήματα στα οποία ο αριθμός των γραμμών του πίνακα είναι 2K ή 4K `float`. Αυτό συμβαίνει γιατί ο αριθ-

μός των μπλοκ αλλά και τον νημάτων είναι σχετικά μικρός και χωράνε σε μια «φουρνιά» και μόνο για οποιαδήποτε επιλογή των παραμέτρων εκτέλεσης. Επιπροσθέτως, παρουσιάζεται μια μικρή επιπλέον επιτάχυνση λόγω της αφαίρεσης κάποιων υπολογισμών σε σχέση με τον DMV_5, η οποία είναι ορατή ακόμα και στα μικρότερα σε μέγεθος προβλήματα. Πάντως, η τελευταία συνεισφέρει μικρό ποσοστό της συνολικής επιτάχυνσης στα μεγαλύτερα προβλήματα.

- Η έκδοση DMV_7, όπου καλείται παρουσιάζει αύξηση της ταχύτητας της εφαρμογής του ΠΠΠΔ κατά περίπου 1 GFlop/s σε κάθε περίπτωση. Αυτό εξηγείται αν ληφθεί υπ'όψη το πόσους ελέγχους και πράξεις αποφεύγονται όταν ξετυλίγουμε τους βρόχους, αλλά και το καλύτερο ποσοστό χρησιμοποίησης σε σχέση με την DMV_6. Εξαίρεση στην παραπάνω συμπεριφορά αποτελούν οι περιπτώσεις του πολλαπλασιασμού 16Kx2K πυκνού πίνακα επί 2K διάνυσμα, με παραμέτρους εκτέλεσης 64 μπλοκ x 256 νήματα/μπλοκ και 128 μπλοκ x 128 νήματα/μπλοκ.
- Ένα γενικό συμπέρασμα είναι ότι όλες οι προσβάσεις στην καθολική μνήμη πρέπει να είναι συγχωνευμένες. Αν συμβαίνει κάι τέτοιο, είναι εφικτές μεγάλες επιταχύνσεις σε σχέση με αντίστοιχους αλγορίθμους που τρέχουν σε πιο παραδοσιακές πλατφόρμες (π.χ. CPUs). Συνεπώς, η μετατροπή όλων των προσβάσεων στην καθολική μνήμη σε συγχωνευμένες, πρέπει να είναι το πρώτο μέλημα επίδοξων προγραμματιστών πάνω στην αρχιτεκτονική G80, για την καλύτερη εκμετάλλευσή της.
- Σε όλες τις εκδόσεις, παρατηρήθηκε μια αύξηση της μέσης ταχύτητας της εφαρμογής του ΠΠΠΔ με την αύξηση του μεγέθους του προβλήματος. Για την ακρίβεια η αύξηση της επίδοσης παρατηρήθηκε για αύξηση των γραμμών του πίνακα (που είναι ουσιαστικά ίσος με τον αριθμό των στοιχείων του τελικού διανύσματος, άρα και τον συνολικό αριθμό των νημάτων εκτέλεσης). Αυτό το γεγονός είναι ενδεικτικό της ανάγκης ύπαρξης πολλών νημάτων εκτέλεσης για να γίνει πλήρης χρήση των δυνατοτήτων της αρχιτεκτονικής G80. Επίσης, είναι φανερό ότι αυτή η αρχιτεκτονική κλιμακώνει καλά σε μεγάλης έκτασης προβλήματα.
- Όσον αφορά τη σχέση της επίδοσης των αλγορίθμων με την επιλεγμένη διάταξη παραμέτρων εκτέλεσης, παρατηρήθηκε ότι, εκτός κάποιων εξαιρέσεων, δεν υπάρχει σημαντική εξάρτηση. Στις περισσότερες από τις περιπτώσεις που μελετήθηκαν η μέση ταχύτητα της εφαρμογής ήταν περίπου η ίδια για οποιαδήποτε επιλογή παραμέτρων εκτέλεσης. Οι περιπτώσεις που παρατηρήθηκαν αποκλίσεις από αυτήν την συμπεριφορά αφορούσαν την έκδοση DMV_4 και αναλύθηκαν παραπάνω.

5.4.3 Δεύτερη σειρά πειραμάτων για την εύρεση πιο αποδοτικής έκδοσης του αλγορίθμου του ΠΠΠΔ

Σε αυτήν την υποενότητα δοκιμάστηκε η επίδοση της έκδοσης DMV_7 για διάφορα προβλήματα με πυκνούς πίνακες με ίσο αριθμό στηλών και γραμμών. Οι μετρήσεις που διεξήχθησαν αφορούν μεγέθη της κάθε διάστασης του πυκνού πίνακα από 2048 στοιχεία μέχρι και 9728, με βήμα 512 στοιχεία. Στο Σχήμα 5-35 υπάρχουν τα αποτελέσματα των μετρήσεων.

Γραμμές πίνακα / Μέγεθος τελικού διανύσματος	Στήλες πίνακα / Μέγεθος αρχικού διανύσματος	Παράμετροι εκτέλεσης		Ταχύτητα έκδοσης DMV_7 (Gflop/s)
		Αριθμός μπλοκ	Μέγεθος μπλοκ	
2048	2048	16	128	11,51626
		32	64	11,4223
2560	2560	20	128	13,81273
		40	64	13,73695
3072	3072	24	128	16,53437
		48	64	16,4672
3584	3584	28	128	19,30476
		56	64	19,14917
4096	4096	16	256	22,90417
		32	128	22,83707
4608	4608	18	256	23,78383
		36	128	24,2116
5120	5120	20	256	26,31576
		40	128	26,78997
5632	5632	22	256	28,78269
		44	128	29,29006
6144	6144	24	256	30,89075
		48	128	31,57466
6656	6656	26	256	33,30474
		52	128	33,29986
7168	7168	28	256	35,3215
		56	128	35,25957
7680	7680	30	256	37,32371
		60	128	37,19445
8192	8192	16	512	40,14626
		32	256	40,1717
8704	8704	17	512	22,22311
		34	256	22,28503
9216	9216	18	512	23,45956
		36	256	23,56471
9728	9728	19	512	24,71902
		38	256	24,72243

Σχήμα 5-35 Πίνακας αποτελεσμάτων μετρήσεων για προβλήματα με τετράγωνους πυκνούς πίνακες διαφόρων μεγεθών με δυο διαφορετικές οργανώσεις παραμέτρων εκτέλεσης ανά μέγεθος πίνακα.

Οι παράμετροι εκτέλεσης επιλέχθηκαν ώστε να υπάρχουν συνολικά τόσο νήματα όσα και ο αριθμός των στοιχείων του διανύσματος του αποτελέσματος της πράξης του ΠΠΠΔ. Επιλέχθηκαν μεγέθη μπλοκ που είχαν δοκιμαστεί στην προηγούμενη σειρά πειραμάτων. Παρατηρώντας το Σχήμα 5-35, γίνεται φανερή μια σχεδόν γραμμική εξάρτηση της ταχύτητας με το μέγεθος του προβλήματος. Αυτό το γεγονός ήταν αναμενόμενο εφόσον είναι γνωστό, από προηγούμενα πειράματα, η πολύ καλή κλιμάκωση της αρχιτεκτονικής G80 και η ανάγκη για χρήση πολλών νημάτων μέχρι να φτάσει τις πλήρεις δυνατότητές της. Με προσεκτική παρατήρηση των μετρήσεων μέχρι και της περίπτωσης του 8192x8192 πίνακα (από εκεί και μετά σταματά απότομα η γραμμική εξάρτηση) είναι φανερό ότι σε κάθε μια από τις περιπτώσεις υπάρχει μόνο μια «φουρνιά» μπλοκ προς εκτέλεση στην κάρτα γραφικών. Με άλλα λόγια, σε όλες αυτές τις περιπτώσεις, όλα τα νήματα εκτέλεσης είναι ενεργά σε κάθε στιγμή και δεν

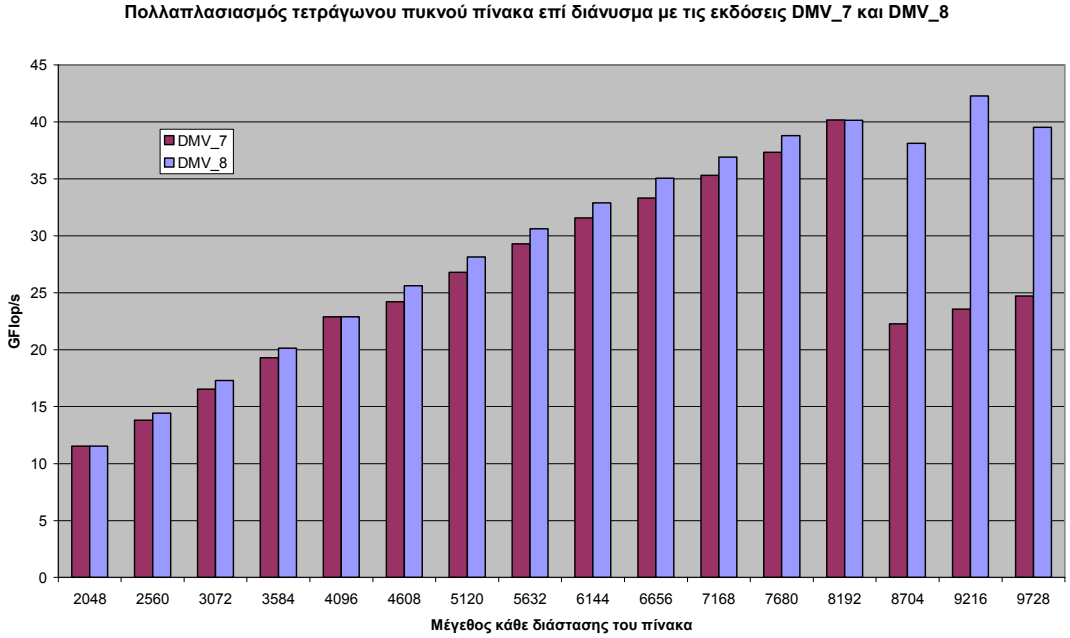
υπάρχει κανένα μπλοκ σε αναμονή προς χρονοδρομολόγηση. Έτσι, καθώς αυξάνονται τα νήματα εκτέλεσης, με την αύξηση του μεγέθους του προβλήματος, αυξάνει και ο όγκος πράξεων προς εκτέλεση με μη παράλληλη αύξηση των φουρνιών, με αποτέλεσμα την καλή κλιμάκωση της ταχύτητας. Σημαντικό στοιχείο που πρέπει να αναφερθεί είναι και το γεγονός ότι καθώς αυξάνεται ο αριθμός των μπλοκ, αυτά μοιράζονται σε SMs που μπορεί να είναι αδρανή ή να έχουν «χώρο» για φιλοξενία περαιτέρω νημάτων. Για να καταστεί το παραπάνω πιο σαφές, μπορεί κάποιος να παρατήσει ότι κατά το πέρασμα από το πρόβλημα του 4608x4608 πίνακα στο πρόβλημα του 5120x5120 πίνακα, και για μέγεθος μπλοκ 256, προστίθενται 2 μπλοκ. Ο συνολικός χρόνος της εφαρμογής δεν επηρεάζεται για τον εξής λόγο: Στο πρόβλημα του 4608x4608 υπήρχαν 18 μπλοκ προς εκτέλεση, με 14 SMs να αναλαμβάνουν από ένα μπλοκ, ενώ τα υπόλοιπα δυο αναλαμβάνουν από 2 μπλοκ. Ο χρόνος που ολοκληρώνεται η εκτέλεση της εφαρμογής καθορίζεται από το χρόνο ολοκλήρωσης και της τελευταίας λειτουργίας. Στο συγκεκριμένο παράδειγμα, ο χρόνος εκτέλεσης καθορίζεται από τον χρόνο που κάνουν να ολοκληρώσουν τις λειτουργίες τους τα SMs που φιλοξενούν δυο μπλοκ. Όμως, SMs με τον ίδιο αριθμό μπλοκ, τα οποία έχουν τον όγκο λειτουργιών προς εκτέλεση, τελειώνουν την ίδια χρονική στιγμή. Έτσι, συμπεραίνουμε ότι προσθέτοντας και ένα μπλοκ σε οποιοδήποτε SM που έχει λιγότερα μπλοκ από τον μέγιστο αριθμό μπλοκ που περιλαμβάνει οποιοδήποτε άλλο SM, δεν επηρεάζεται ο συνολικός χρόνος εκτέλεσης, εφόσον τα SMs εργάζονται παράλληλα. Αυτός είναι και ο κύριος λόγος της γραμμικής εξάρτησης της ταχύτητας με το μέγεθος του προβλήματος.

Η γραμμική αύξηση της ταχύτητας συναρτήσει του μεγέθους του προβλήματος σταματά για μέγεθος πίνακα 8192x8192. Σε εκείνο το σημείο όλα τα SMs είναι πλήρη, δηλαδή δεν χωρούν άλλα μπλοκ ή νήματα χωρίς τη δημιουργία δεύτερης «φουρνιάς» μπλοκ. Από την επόμενη μέτρηση και μετά, παρατηρούμε ένα βύθισμα στην ταχύτητα. Αυτό οφείλεται καθαρά στην δημιουργία νέας «φουρνιάς» μπλοκ, καθώς ο αριθμός των συνολικών μπλοκ μεγέθους 512 νημάτων γίνεται μεγαλύτερος του 16, άρα δεν υπάρχουν αρκετά SMs για να φιλοξενήσουν όλα τα απαιτούμενα μπλοκ σε μια και μοναδική «φουρνιά». Σε αυτά τα προβλήματα υπάρχουν δηλαδή δυο «φουρνιές, η πρώτη, κατά την οποία είναι πλήρη όλα τα SMs και η δεύτερη, κατά την οποία δεν χρησιμοποιούνται όλα τα SMs (για παράδειγμα στην περίπτωση του 8704x8704 πίνακα με παραμέτρους εκτέλεσης 17x512, στη δεύτερη φουρνιά θα υπάρχει μόνο ένα SM που θα φιλοξενεί ένα μπλοκ). Οι μετρήσεις σταματούν σε μέγεθος πίνακα 9728x9728, καθώς για μεγαλύτερους πίνακες δεν υπάρχει χώρος αποθήκευσής τους στην καθολική μνήμη της συσκευής.

Στις μετρήσεις για προβλήματα μεγαλύτερα αυτού με πίνακα 8192x8192 παρατηρείται μια τάση, νέας γραμμικής αύξησης, ανάλογης αυτής που παρατηρήθηκε στα προβλήματα από 2048x2048 μέχρι και 8192x8192. Ο λόγος είναι ο ίδιος με αυτόν που εξηγήθηκε παραπάνω και έχει να κάνει με τις θέσεις για φιλοξενία μπλοκ σε κάποια SMs (σε σχέση με τα SMs με το μεγαλύτερο πλήθος φιλοξενούμενων μπλοκ). Η μόνη διαφορά είναι ότι σε αυτήν την περίπτωση η πληρότητα αφορά τη έτερη «φουρνιά» μπλοκ, καθώς η πρώτη είναι γεμάτη και ισομερώς κατανεμημένη στα SMs.

Πάντως, το απότομο βύθισμα μετά το μέγεθος πίνακα 8192x8192 δεν είναι επιθυμητή συμπεριφορά, αλλά μπορεί να αποφευχθεί. Για το λόγο αυτό, δημιουργήθηκε ένας ελαφρώς βελτιωμένος πυρήνας, ο *DMV_8*. Η καινοτομία του σε σχέση με αυτούς που μελετήθηκαν προηγουμένως είναι η αυτόματη επιλογή των παραμέτρων εκτέλεσης με

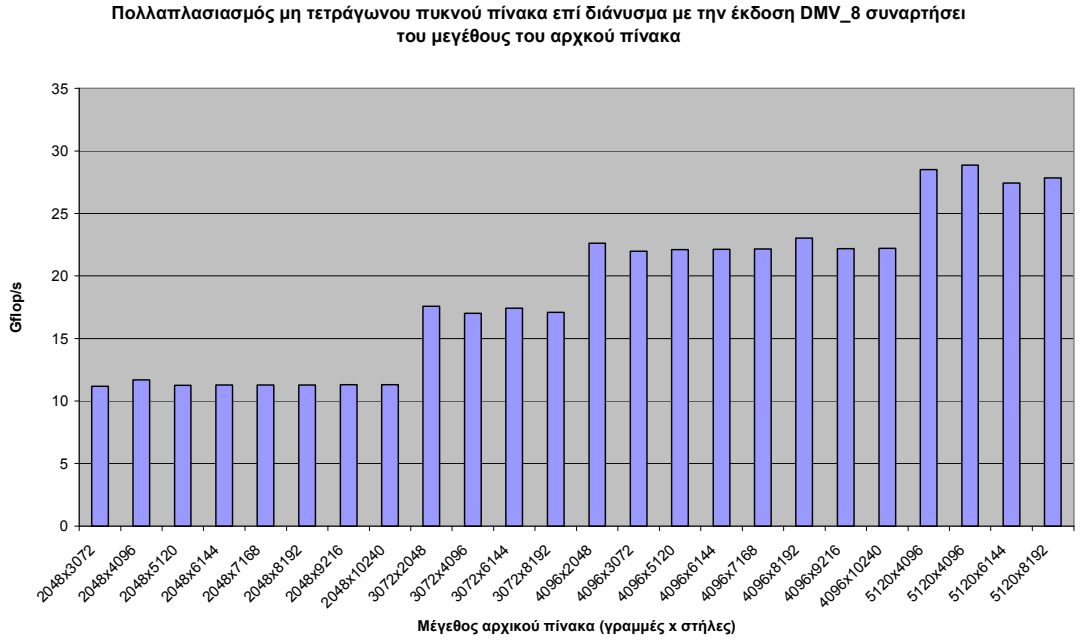
σκοπό τον καλύτερο διαμοιρασμό των νημάτων στα διαθέσιμα SMs (σε αντίθεση με τις επιλεγμένες από το χρήση παραμέτρους εκτέλεσης στις προηγούμενες εκδόσεις) Αυτό θα έχει ως αποτέλεσμα να αποφεύγονται βυθίσματα σαν αυτό που παρατηρήθηκε προηγουμένως, αλλά και να υπάρχει ισομερής διαχωρισμός των νημάτων στα διάφορα SMs. Σκοπός του τελευταίου είναι η αποφυγή ύπαρξης SMs που ολοκληρώνουν τις λειτουργίες τους πιο γρήγορα από άλλα, και παραμένουν αδρανή μέχρι την ολοκλήρωση της εφαρμογής. Οι απαιτήσεις σε bytes διαμοιραζόμενης μνήμης ανά μπλοκ και σε αριθμό καταχωρητών ανά νήμα είναι ίδιες με αυτές της έκδοσης DMV_7, όπως και ο κώδικας των πυρηνών που εκτελούνται κατά περίπτωση. Η χρησιμοποίηση όμως εδώ ρυθμίζεται αυτόματα, καθώς το μέγεθος των μπλοκ επιλέγεται τέτοιο ώστε να μοιράζονται εξ'ίσου σε όλα τα SMs τα νήματα και επιπλέον να γεμίζουν όσο γίνεται τα μπλοκ, ενώ παράλληλα ο αριθμός των «φουρνιών» που θα χρειαστούν να είναι ο ελάχιστος δυνατός. Στο Σχήμα 5-36 παρατίθενται οι μετρήσεις ταχύτητας του πυρήνα DMV_8 σε σύγκριση με αυτές του DMV_7 (επιλέχθηκαν για αυτήν την έκδοση οι παράμετροι εκτέλεσης με την καλύτερη επίδοση, αν και όλα τα μεγέθη μπλοκ που είχαν δοκιμαστεί έδιναν πολύ κοντινή επίδοση), για διάφορα μεγέθη προβλημάτων με τετράγωνους πίνακες.



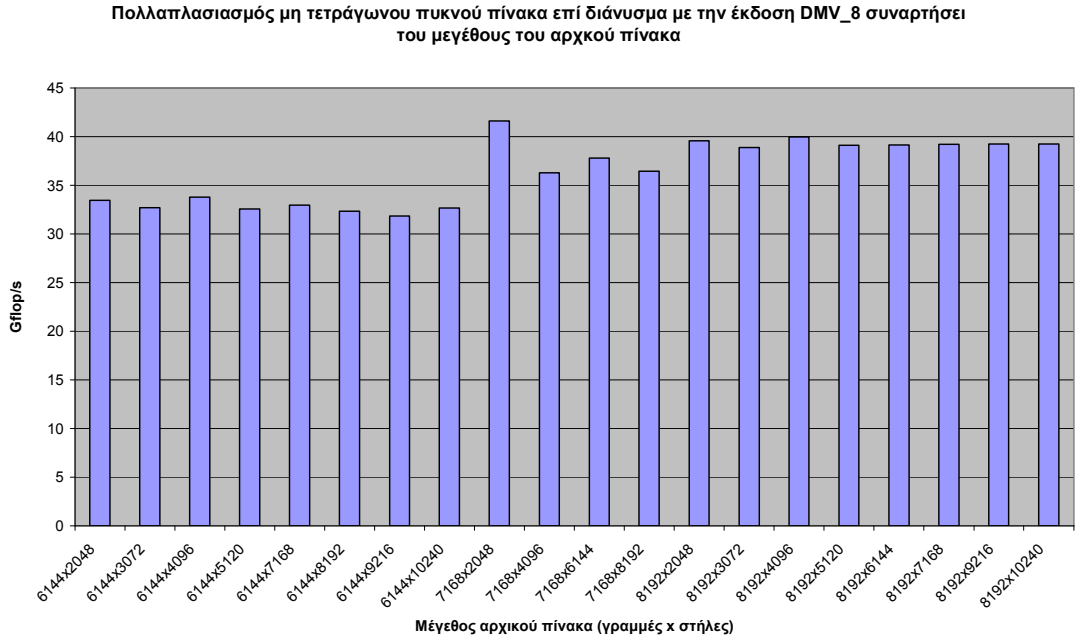
Σχήμα 5-36 Σύγκριση επίδοσης εκδόσεων DMV_7 και DMV_8 για διάφορα μεγέθη προβλημάτων ΠΠΠΔ για τετράγωνους πίνακες.

Από το Σχήμα 5-36, είναι φανερό το πλεονέκτημα της έκδοσης DMV_8 έναντι των προηγουμένων, επιβεβαιώνοντας τη σημασία της επιλογής των παραμέτρων εκτέλεσης για την επίδοση των εφαρμογών που «τρέχουν» στην αρχιτεκτονική G80. Σε όλες τις περιπτώσεις προβλημάτων που μελετήθηκαν υπήρξε μια επιτάχυνση που αυξανόταν ανάλογα με την αύξηση του μεγέθους του προβλήματος και πριν το βύθισμα του DMV_7 έφτασε μέχρι και το 1,5 GFlop/s. Μετά το βύθισμα η ταχύτητα της DMV_8 είναι εμφανώς μεγαλύτερη σε σχέση με την έκδοση DMV_7, καθώς πλέον ο καταμερισμός των λειτουργιών γίνεται με βέλτιστο τρόπο και εξ'ίσου σε όλα τα διαθέσιμα SMs.

Στο επόμενο βήμα της πειραματικής αξιολόγησης της εκτέλεσης του αλγορίθμου με χρήση της αρχιτεκτονικής G80, δοκιμάστηκε πλήθος προβλημάτων διαφόρων μεγεθών που περιελάμβαναν και μη τετράγωνους πίνακες Τα αποτελέσματα για τετραγωνικούς πίνακες έχουν παρουσιαστεί στο Σχήμα 5-36.. Τα αποτελέσματα των εν λόγω πειραμάτων περιλαμβάνονται στα διαγράμματα των Σχημάτων 5-37 και 5-38.



Σχήμα 5-37 Αποτελέσματα πειραμάτων ΠΠΠΔ για μη τετράγωνους αρχικούς πυκνούς πίνακες.



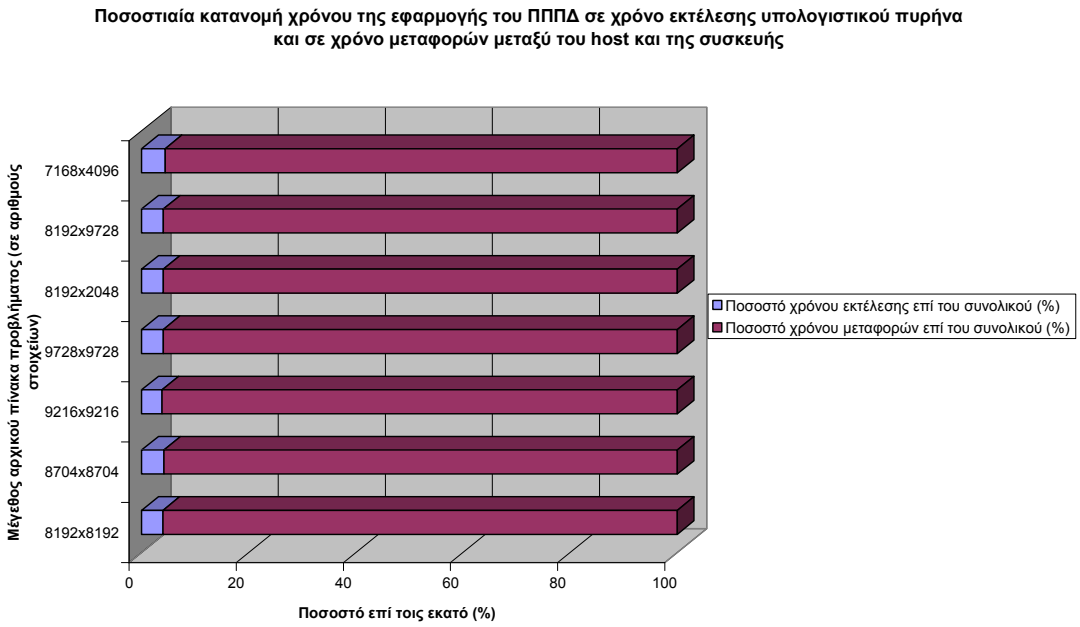
Σχήμα 5-38 Αποτελέσματα πειραμάτων ΠΠΠΔ για μη τετράγωνους αρχικούς πυκνούς πίνακες.

Μελετώντας τα Σχήματα 5-37 και 5-38 είναι δυνατόν να εξαχθούν τα εξής συμπεράσματα:

- Παρατηρείται μια βαθμιαία αύξηση της μέσης ταχύτητας εκτέλεσης του ΠΠΠΔ με αύξηση του μεγέθους του προβλήματος. Αυτή η αύξηση συνεπάγεται αύξηση των νημάτων εκτέλεσης. Για μικρούς αριθμούς νημάτων δηλαδή, δεν φτάνει στα όρια της η αρχιτεκτονική G80, καθώς απαιτείται μεγάλος αριθμός νημάτων για την πλήρη εκμετάλλευσή της. Έτσι, επιβεβαιώνεται ξανά η ανάγκη ύπαρξης πολλών νημάτων για την μεγιστοποίηση της επίδοσης ανεπτυγμένων με το CUDA εφαρμογών.
- Για ίδιο αριθμό νημάτων (αριθμός γραμμών πυκνού πίνακα) και μεταβαλλόμενο αριθμό πράξεων ανά νήμα (αριθμός στηλών πυκνού πίνακα) παρατηρείται σχεδόν σταθερή μέση ταχύτητα εκτέλεσης. Αυτό ήταν αναμενόμενο δεδομένου ότι σε όλες τις περιπτώσεις εργάζονταν ο ίδιο αριθμός νημάτων και έκαναν αναγνώσεις από τη μνήμη και πράξεις με τον ίδιο ρυθμό. Για να γίνει πιο ξεκάθαρο το παραπάνω, σημειώνουμε ότι το συντριπτικά μεγαλύτερο ποσοστό των λειτουργιών των νημάτων εκτέλεσης έχουν να κάνουν με τις αναγνώσεις και τις πράξεις μεταξύ των στοιχείων του πυκνού πίνακα και του αρχικού διανύσματος. Όταν εργάζεται ο ίδιος αριθμός νημάτων, ο ρυθμός εργασίας είναι ο ίδιος κατά τις αναγνώσεις και τις πράξεις και άρα και η μέση ταχύτητα παραμένει σταθερή, μη επηρεαζόμενη από άλλους παράγοντες. Βέβαια, ο συνολικός χρόνος για μεγαλύτερο μέγεθος στηλών του πυκνού πίνακα (με δεδομένο ότι μιλάμε για σταθερό αριθμό γραμμών του πυκνού πίνακα) θα είναι μεγαλύτερος. Όμως, και οι συνολικές πράξεις MAD σε αυτήν την περίπτωση θα είναι περισσότερες, οπότε τελικά, το πηλίκο του αριθμού των πράξεων προς τον χρόνο εκτέλεσης θα μένει σχεδόν σταθερό, όπως επιβεβαιώνεται από τα παραπάνω πειράματα.

5.5 Επίδραση των μεταφορών μεταξύ host και συσκευής στην συνολική επίδοση των εφαρμογών

Σε αυτό το σημείο πρέπει να γίνει ιδιαίτερη μνεία σε έναν σημαντικό περιοριστικό παράγοντα της επίδοσης, που είναι οι μεταφορές μεταξύ του host και της συσκευής. Όπως αποδείχτηκε και πειραματικά το διαθέσιμο εύρος ζώνης για μια τέτοια επικοινωνία, δεν ξεπερνά τα 3.2 GB/s. Αυτό έχει ως αποτέλεσμα η επίδοση να πέφτει κατακόρυφα, αν ληφθεί υπ'όψη στον συνολικό χρόνο επεξεργασίας και ο χρόνος των μεταφορών. Στο Σχήμα 5-39 είναι φανερή η κατανομή του συνολικού χρόνου εκτέλεσης της εφαρμογής, περιλαμβάνοντας και τον χρόνο που αντιστοιχεί στις μεταφορές για το πρόβλημα του ΠΠΠΔ. Παρατηρείται ότι ο χρόνος μεταφορών κυριαρχεί συντριπτικά του συνολικού χρόνου εκτέλεσης του αλγορίθμου. Αποτέλεσμα αυτού είναι η μέση ταχύτητα εκτέλεσης, λαμβανόμενου υπ'όψη και του χρόνου μεταφορών, να κυμαίνεται σε πολύ χαμηλότερα επίπεδα και να ξεπερνά με δυσκολία τα 1,6 GFlop/s, πολύ λίγο καλύτερη δηλαδή σε σχέση με τα επίπεδα της CPU.



Σχήμα 5-39 Ποσοστιαία κατανομή του χρόνου της εφαρμογής του ΠΠΠΔ σε χρόνο εκτέλεσης και χρόνο μεταφορών για διάφορα μεγάλα μεγέθη του προβλήματος.

Σε αυτό το σημείο, είναι ανάγκη να υπενθυμιστεί ότι οι μικροεφαρμογές που εξετάζονται εδώ αποτελούν στοιχειώδη κομμάτια των πραγματικών επιστημονικών εφαρμογών. Σε αυτές, οι υπολογισμοί είναι τάξεις μεγέθους περισσότεροι από τις ανάγκες μεταφορών μεταξύ host και συσκευής και, κατά συνέπεια, η μέση ταχύτητα εκτέλεσής πράξεων κινητής υποδιαστολής στην GPU είναι συντριπτικά καλύτερη εκείνης της CPU. Επιπλέον, στις πραγματικές εφαρμογές θα εκτελείται ένας μεγάλος αριθμός πυρήνων σειριακά και θα συντελούνται διαφόρων μεγεθών μεταφορές, οι οποίες μπορούν να είναι ασύγχρονες. Το προγραμματιστικό περιβάλλον του CUDA δίνει αυτή τη δυνατότητα προκειμένου να αποφεύγονται ακόμα περισσότερο συμπεριφορές όπως αυτή που παρατηρήθηκε στις μικροεφαρμογές που μελετήθηκαν στο παρόν κεφάλαιο. Με χρήση των ασύγχρονων μεταφορών, μπορεί ταυτόχρονα να εκτελείται κάποιος υπολογιστικός πυρήνας στην GPU, ενώ ταυτόχρονα να μεταφέρονται στη συσκευή δεδομένα εισόδου του επόμενου υπολογιστικού πυρήνα προς εκτέλεσης ή να μεταφέρονται στον host τα αποτελέσματα του προηγούμενου υπολογιστικού πυρήνα.

5.6 Ανακεφαλαίωση σημαντικότερων συμπερασμάτων

Μέσα από την πειραματική διαδικασία που ακολουθήθηκε, έγινε εφικτή η επιβεβαίωση των σημαντικότερων στρατηγικών βελτίωσης της επίδοσης κάποιων πυρήνων, ευρέως διαδεδομένων σε επιστημονικές εφαρμογές. Ένα από τα σημαντικότερα στοιχεία, που ο επίδοξος προγραμματιστής πρέπει να έχει υπ'όψη του, είναι η προσεκτική χρήση της καθολικής μνήμης. Αφού μειώσει κατά το δυνατόν τις προσβάσεις προς αυτήν, πρέπει να τις δομήσει έτσι ώστε να είναι πάντα συγχωνευμένες. Αυτός ο τρόπος βελτιστοποίησης δίνει τη μεγαλύτερη επιτάχυνση στην εφαρμογή, όπως κατέστη σαφές από όλα τα πειράματα που παρουσιάστηκαν στο παρόν κεφάλαιο. Επίσης. πο-

λύ σημαντικό στοιχείο είναι η χρήση όσο περισσότερων νημάτων είναι εφικτό, ώστε ο διαθέσιμος μεγάλος αριθμός υπολογιστικών πυρήνων, να εκμεταλλεύεται πλήρως από τον εκάστοτε αλγόριθμο. Μεγάλη επίδραση στην τελική επίδοση της κάθε εφαρμογής έχουν και οι επιλεγμένες παράμετροι εκτέλεσης, καθώς με προσεκτική επιλογή τους παρατηρήθηκαν σημαντικές διαφοροποιήσεις στην τελική επίδοση των εφαρμογών. Η επιλογή αυτών των παραμέτρων έχει άμεση σχέση με τους πόρους που δεσμεύει το κάθε νήμα κατά την εκτέλεσή του, κάτι στο οποίο επίσης πρέπει να δίνεται η δέουσα προσοχή. Γενικά, στόχος θα πρέπει να είναι η μεγαλύτερη δυνατή χρησιμοποίηση των διαθέσιμων πόρων των SMs. Επιβεβαιώθηκαν τελικά, σε γενικές γραμμές, αυτά που είχαν αναλυθεί σε θεωρητικό επίπεδο στο Κεφάλαιο 4. Βέβαια, η κάθε εφαρμογή έχει τις δικές της ιδιαιτερότητες και εκτός των γενικών κατευθύνσεων που έχουν δοθεί, πρέπει ο επίδοξος προγραμματιστής που ενδιαφέρεται για τη καλύτερη δυνατή επίδοση να πειραματιστεί και να προσπαθήσει να εκμεταλλευτεί κάθε λεπτομέρεια της αρχιτεκτονικής G80.

Κεφάλαιο 6

Συμπεράσματα και μελλοντικές επεκτάσεις

Στο πλαίσιο αυτής της διπλωματικής εργασίας μελετήθηκαν οι δυνατότητες μιας πλατφόρμας πολλών πυρήνων για χρήση σε επιστημονικές εφαρμογές. Η πλατφόρμα αυτή δεν ήταν γενικού σκοπού, αλλά, αντίθετα, απευθύνεται πρωταρχικά σε εξειδικευμένους υπολογισμούς γραφικών. Αρχικά, ξεδιαλύθηκε το τοπίο της σύγχρονης αγοράς επεξεργαστών και παρουσιάστηκαν οι σημαντικότερες τεχνικές αύξησης επίδοσης των υπολογιστικών συστημάτων, οι οποίες έχουν να κάνουν με τη χρήση παραλληλίας σε οποιοδήποτε επίπεδο της οργάνωσής τους. Δόθηκε ιδιαίτερη έμφαση στην παραλληλία σε επίπεδο επεξεργαστικών πυρήνων και μελετήθηκαν οι λόγοι που οδηγούν σε ολοένα αυξανόμενες ανάγκες σε αριθμό πυρήνων, καθώς και σε απλοποίηση των δομικών στοιχείων του συνολικού συστήματος. Επικεντρώνοντας το ενδιαφέρον σε μια εξειδικευμένη κατηγορία πολυεπεξεργαστικών υπολογιστικών συστημάτων, των καρτών γραφικών, έγιναν φανερές κάποιες από τις εκμεταλλεύσιμες δυνατότητές τους για εφαρμογές γενικού σκοπού. Παρουσιάστηκαν πολλές προσπάθειες χρήσης αυτών των δυνατοτήτων, οι οποίες είχαν στεφθεί με σχετική επιτυχία κατά το παρελθόν, και οι οποίες έδωσαν το έναυσμα για περαιτέρω μελέτη των διαφορετικών αυτών αρχιτεκτονικών. Με το πέρασμα του χρόνου, προσθέτονταν συνεχώς νέα χαρακτηριστικά στις GPUs, τα οποία κατέστη δυνατό να αξιοποιηθούν μέσω των αντίστοιχων προγραμματιστικών διεπαφών που έκαναν συγχρόνως την εμφάνισή τους. Πολύ σημαντικό βήμα προς την πιο αποδοτική και εύκολη χρήση των GPUs σε υπολογισμούς γενικού σκοπού ήταν η εμφάνιση της αρχιτεκτονικής G80 της εταιρεί-

ας nVidia, η οποία πρώτη εισήγαγε ενοποιημένους επεξεργαστικούς πυρήνες. Σε αυτήν βασίζονται οι GPUs της σειράς 8 της εταιρείας, και ο προγραμματισμός τους έγινε εφικτός μέσω του CUDA, μιας προγραμματιστικής διεπαφής. Παρουσιάστηκε με λεπτομέρεια η πολυπύρηνη αρχιτεκτονική G80, ενώ έγινε ιδιαίτερη μνεία στους νεωτερισμούς που αυτή εισήγαγε. Επίσης, έγινε αναφορά στα σημαντικότερα σημεία του CUDA και, κυρίως, στον τρόπο με τον οποίο αυτό μπορεί να απεικονίσει και να αναθέσει στους πολλούς επεξεργαστικούς πυρήνες την εκάστοτε εφαρμογή. Μετά από τα θεωρητικά θεμέλια που τέθηκαν στο Κεφάλαιο 3, παρουσιάστηκαν οι κυριότερες στρατηγικές βελτιστοποίησης των εφαρμογών που απευθύνονται στη μελετώμενη πλατφόρμα. Ο επίδοξος προγραμματιστής θα πρέπει να ακολουθήσει τις παραπάνω, ώστε να γίνεται η καλύτερη δυνατή αξιοποίηση των επεξεργαστικών πόρων των GPUs. Όλα τα παραπάνω βρήκαν εφαρμογή στο Κεφάλαιο 5, στο οποίο διεξήχθη πλήθος πειραμάτων. Αυτά είχαν να κάνουν με την αξιολόγηση της επίδοσης κάποιων υπολογιστικών πυρήνων ευρέως χρησιμοποιούμενων σε επιστημονικές εφαρμογές. Αρχικά, εξερευνήθηκαν οι δυνατότητες εκτέλεσης πολλών αριθμητικών πράξεων με στόχο να βρεθεί ένα άνω όριο της ταχύτητας εκτέλεσης πράξεων κινητής υποδιαστολής. Μελετήθηκε το πρόβλημα της αναγωγής, με χρήση ενός παράλληλου αλγορίθμου και εφαρμόστηκαν πολλές από τις προτεινόμενες στρατηγικές βελτιστοποίησης και αξιολογήθηκε η επίδρασή τους στη συνολική επίδοση. Κάτι ανάλογο έγινε και τον Πολλαπλασιασμό Πυκνού Πίνακα επί Διάνυσμα, ενός πολύ συχνά συναντώμενου υπολογιστικού πυρήνα σε πλήθος εφαρμογών. Σε κάθε περίπτωση η εφαρμογή των στρατηγικών επίλυσης έγινε βαθμιαία, ώστε να γίνει ορατό το ποσοστό της επίδρασης της κάθε μιας ξεχωριστά.

Μετά από τα πειράματα που διεξήχθησαν, έγινε φανερό ότι η αρχιτεκτονική G80 μπορεί να αποτελέσει μια πολύ χρήσιμη πλατφόρμα ανάπτυξης εφαρμογών, ιδιαίτερα όταν πρόκειται για προβλήματα που μπορούν να παραλληλοποιηθούν κατά βούληση. Η δυνατότητα να υποστηριχτούν χιλιάδες νημάτων εκτέλεσης επιτρέπει τον εύκολο χειρισμό ιδιαίτερα ογκωδών συνόλων δεδομένων και την εκτέλεση πολύπλοκων λειτουργιών μαζικά παράλληλα. Η προγραμματιστική διεπαφή που προσφέρεται μαθαίνεται εύκολα από επίδοξους προγραμματιστές, ενώ η μόνη δυσκολία είναι η εφαρμογή και ο έλεγχος των στρατηγικών βελτιστοποίησης, προκειμένου σε κάθε εφαρμογή να επιτυγχάνεται η μέγιστη δυνατή επίδοση. Το τελευταίο αποτελεί ένα σχετικά χρονοβόρο και επίπονο έργο, λόγω του πολυδιάστατου χώρου παραμέτρων που επηρεάζουν την επίδοση. Πολλές από αυτές τις παραμέτρους δεν είναι ανεξάρτητες μεταξύ τους και προκειμένου κάποιος να είναι σίγουρος για το αποτέλεσμα της αλλαγής μιας εκ των παραμέτρων πρέπει να έχει πολύ καλή εποπτεία της αρχιτεκτονικής και του μοντέλου εκτέλεσης. Έχει γίνει φανερή η ανάγκη ανάπτυξης κάποιου αυτοματοποιημένου τρόπου επιλογής αυτών των παραμέτρων [58, 59, 60].

Από την ενασχόληση με τις κάρτες γραφικών της αρχιτεκτονικής G80 προκύπτει η επιθυμία αύξησης της ταχύτητας επικοινωνίας μεταξύ CPU και GPU, κάτι πάντως που δεν εξαρτάται μόνο από τις εταιρείες κατασκευής καρτών γραφικών αλλά από τον συνολικό σχεδιασμό των σύγχρονων προσωπικών υπολογιστών. Επίσης, επιθυμητή είναι η βελτίωση του εύρους ζώνης επικοινωνίας μεταξύ των υπολογιστικών πυρήνων και της καθολικής μνήμης των καρτών γραφικών, καθώς οι προσβάσεις σε αυτήν αποτελούν τη μεγαλύτερη στενωπό επίδοσης. Τέλος, λόγω του περιορισμένου μεγέθους της διαμοιραζόμενης μνήμης ανά SM είναι πιθανό κάποιοι αλγόριθμοι να μην μπορούν να απεικονιστούν βέλτιστα, άρα μια αύξηση του μεγέθους της σε μελλοντικές γενιές θα ήταν κάτι παραπάνω από σημαντικό βήμα εξέλιξης.

Οδηγείται κανείς στο συμπέρασμα ότι οι νέες αρχιτεκτονικές των GPUs μπορούν να προσφέρουν πολλά και σε εφαρμογές πέρα από τα γραφικά. Το παραπάνω έγινε φανερό από τη μελέτη της αρχιτεκτονικής G80, η οποία μαζί με την αρχιτεκτονική R600 της AMD αποτελούν την πρώτη σημαντική προσπάθεια ενοποίησης του τρόπου υπολογισμών στο χώρο των γραφικών. Το νέο μοντέλο υπολογισμού σε ροές, που σε γενικές γραμμές ακολουθούν, φαίνεται να είναι ο καλύτερος τρόπος αξιοποίησης των πολλών πυρήνων που προσφέρουν οι υποκείμενες αρχιτεκτονικές. Οι αρχιτεκτονικές αυτές, μαζί με τα πακέτα ανάπτυξης λογισμικού που τα συνοδεύουν, έχουν στρέψει πάνω τους μεγάλο ενδιαφέρον της επιστημονικής και ερευνητικής κοινότητας. Ενδεικτικό της στροφής αυτής είναι η έκρηξη που παρατηρήθηκε μέσα στο 2008, όσον αφορά τη χρησιμοποίηση του CUDA και των καρτών που στηρίζονται στη G80 σε πλήθος εφαρμογών με πολύ ενθαρρυντικά αποτελέσματα [61].

Οι ενδείξεις οδηγούν στο συμπέρασμα, ότι στο μέλλον υπάρχει μεγάλη πιθανότητα οι αρχιτεκτονικές των GPUs και των CPUs να οδηγηθούν σε μια είδους σύγκλιση ως προς τα χαρακτηριστικά που θα υλοποιούν. Είναι εμφανής η τάση των CPUs να περιλαμβάνουν στο ίδιο τσιπ πολλούς πυρήνες, που με την πάροδο του χρόνου ίσως φτάσουν σε αριθμό τις τωρινές GPUs, ενώ επίσης υποστηρίζεταί μεγάλος αριθμός νημάτων σε επίπεδο υλικού τα τελευταία χρόνια. Επίσης, οι GPUs όσο εξελίσσονται, διαφαίνεται μια τάση να υιοθετούν ολοένα αυξανόμενα στοιχεία που μέχρι τώρα ήταν ορατά μόνο σε CPUs. Χαρακτηριστικό παράδειγμα αποτελούν το τσιπ της Intel, Larrabee, και οι νέες σειρές καρτων γραφικών της nVidia.

Η GPU Larrabee αποτελεί μια προσπάθεια της Intel να εισχωρήσει στον χώρο του GPGPU, με ιδιαίτερα υποσχόμενες προοπτικές [62, 64]. Το Larrabee θα κάνει χρήση της αρχιτεκτονικής x86 για την εκτέλεση των προγραμμάτων σκίασης, ενώ θα υλοποιεί πρωτόκολλα συνάφειας των κρυφών μνημών των επεξεργαστικών του πυρήνων. Αποτέλεσμα θα είναι να είναι πολύ πιο ευέλικτο από τις προσπάθειες των εταιρειών γραφικών, ενώ το έδαφος για γενικού τύπου εφαρμογές αναμένεται να είναι πιο πρόσφορο.

Οι νέες σειρές καρτών γραφικών της nVidia προσθέτουν ιδιαίτερα χρήσιμα και εκμεταλλεύσιμα χαρακτηριστικά, όπως η υποστήριξη αριθμών κινητής υποδιαστολής διπλής ακρίβειας, ενώ τα τεχνικά τους χαρακτηριστικά αναβαθμίζονται σημαντικά. Πρόκειται για τη σειρά 9 της nVidia και τη σειρά που θα υποστηρίζει την αρχιτεκτονική G200 [63].

Όλα τα παραπάνω γεγονότα οδηγούν στην αντίληψη ότι στο άμεσο μέλλον η χρήση των GPUs αναμένεται να χρησιμοποιηθεί εκτενώς και σε ολοένα πιο σύνθετα και δυσεπίλυτα προβλήματα. Όπως έγινε φανερό και από την παρούσα διπλωματική εργασία, οι δυνατότητες που προσφέρονται από τα τσιπ γραφικών ακόμα και σήμερα, αν χρησιμοποιηθούν σωστά, μπορούν να έχουν αποτέλεσμα θεαματικές επιδόσεις σε σωρεία επιστημονικών εφαρμογών. Στο μέλλον και με την αναμενόμενη εμφάνιση ολοένα και πιο ισχυρών και πιο γενικών GPUs, η παραπάνω αντίληψη ενισχύεται και γίνεται ορατή πλέον η πιθανότητα επίτευξης μιας σύγκλισης στις αρχιτεκτονικές αρχές, και κατ' επέκταση στον τρόπο προγραμματισμού, που θα βασίζονται από εδώ και στο εξής τα πολυπύρηνα υπολογιστικά συστήματα.

Βιβλιογραφία

[1]Singh J. P., Culler D., and Gupta A.. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, 1998.

[2]“x86-64”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/X86-64>.

[3]Lo J. L., Eggers S. J., Emer J. S., Levy Henry M., Stamm Rebecca L., Tullsen Dean M.. Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading. ACM Trans. on Comp. Syst., 1997.

[4]Koufaty, D.; Marr, D.T., "Hyperthreading technology in the netburst microarchitecture," Micro, IEEE , vol.23, no.2, pp. 56-65, March-April 2003.

[5]Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," IEEE Micro, vol. 25, no. 2, pp. 21-29, Mar/Apr, 2005 .

[6]Asanovic, K., et al.: The Landscape of Parallel Computing Research: A View from Berkeley, Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183 (December 18, 2006).

[7]Shalf J : The New Landscape of Parallel Computer Architecture, Journal of Physics: Conference Series, 2007 .

[8]Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, Takeshi Yamazaki, "Synergistic Processing in Cell's Multicore Architecture," IEEE Micro, vol. 26, no. 2, pp. 10-24, Mar/Apr, 2006 .

[9]“Cell (microprocessor)”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/Cell_%28microprocessor%29>.

[10]Chetana N. Keltcher, Kevin J. McGrath, Ardsher Ahmed, Pat Conway, "The AMD Opteron Processor for Multiprocessor Servers," IEEE Micro, vol. 23, no. 2, pp. 66-76, Mar/Apr, 2003.

[11]“AMD Opteron”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/AMD_Opteron>.

[12]“Non-Uniform Memory Access”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access>.

[13]“UltraSPARC T1”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/UltraSPARC_T1>.

[14]Intel 64 and IA-32 Architectures Optimization Reference Manual.

[15]“Intel Core 2”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/Intel_Core_2>.

[16]OpenGL Programming Guide. 25 Αυγούστου 2008: <http://www.glprogramming.com/red/>.

[17]OpenGL 2.1 Reference Pages. 25 Αυγούστου 2008: <http://www.opengl.org/sdk/docs/man/>.

[18]Programming Guide for Direct3D 10. 25 Αυγούστου 2008: <http://msdn.microsoft.com/el-gr/library/bb205067(en-us,VS.85).aspx>.

[19]Direct3D Reference. 25 Αυγούστου 2008: <http://msdn.microsoft.com/el-gr/library/bb205148(en-us,VS.85).aspx>.

[20]EN600.407 General Purpose Computing on the GPU, John Hopkins University, Spring 2008.

[21]“Flynn's taxonomy”. Wikipedia: The free encyclopedia. 25 Αυγούστου 2008: <http://en.wikipedia.org/wiki/Flynn%27s_Taxonomy>.

[22]John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum, 26(1):80–113, March 2007.

[23]Ιστοτόπος για την τεκμηρίωση του CUDA. 25 Αυγούστου 2008: <http://www.nvidia.com/object/cuda_develop.html>.

[24]ATI CTM Guide. 25 Αυγούστου 2008: <http://ati.amd.com/companyinfo/researcher/documents/ATI_CTM_Guide.pdf>.

[25]Brook specification. 25 Αυγούστου 2008: <http://graphics.stanford.edu/papers/brookspec-v0.2/brookspec-v0.2.pdf.>.

[26]Sh Documentation. 25 Αυγούστου 2008: <http://libsh.org/api/>.

[27]Σελίδα υποστήριξης προγραμματιστών RapidMind. 25 Αυγούστου 2008: <https://developer.rapidmind.net/>.

[28]Tarditi, D.; Puri, S.; Oglesby, J. Accelerator: using data parallelism to program gpus for general-purpose uses. SIGOPS Oper. Syst. Rev., ACM, New York, NY, USA, v. 40, n. 5, p. 325–335, 2006. ISSN 0163-5980.

[29]Matthew Papakipos. The PeakStream Platform:
High-Productivity Software Development for Multi-Core Processors, Peakstream Inc, Arpil 10, 2007.

[30]Cg Language Specification. 25 Αυγούστου 2008:
<ftp://download.nvidia.com/developer/cg/Cg_Specification.pdf.>.

[31]Reference for HLSL. 25 Αυγούστου 2008: <http://msdn.microsoft.com/en-us/library/bb509638(VS.85).aspx>.

[32]John Kessenich. The OpenGL® Shading Language v *1.20*, September 2006.

[33]MATLAB on-line documentation. 25 Αυγούστου 2008:
<http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html>.

[34]Govindaraju N. K., Lloyd B.,Wang W., Lin M., Manocha D.: Fast computation of database operations using graphics processors. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (June 2004), pp. 215–226.

[35]Je˛Dezejewski M.: Computation of Room Acoustics on Programmable Video Hardware. Master's thesis, Polish-Japanese Institute of Information Technology, Warsaw, Poland, 2004.

[36]BionicFX. 25 Αυγούστου 2008 <http://www.bionicfx.com/>.

[37]Purcell T. J., Donner C., Cammarano M., Jensen H. W., Hanrahan P.: Photon mapping on programmable graphics hardware. In Graphics Hardware 2003 (July 2003), pp. 41–50.

[38]Carr N. A., Hall J. D., Hart J. C.: GPU algorithms for radiosity and subsurface scattering. In Graphics Hardware 2003 (July 2003), pp. 51–59.

[39]Thrane N., Simonsen L. O.: A Comparison of Acceleration structures for GPU Assisted Ray Tracing. Master's thesis, University of Aarhus, Aug. 2005.

[40]Weiskopf D., Schafhitzel T., Ertl T.: GPU-based nonlinear ray tracing. Computer Graphics Forum 23, 3 (Sept. 2004), 625–633.

[41]Carr N. A., Hall J. D., Hart J. C.: The ray engine. In Graphics Hardware 2002 (Sept. 2002), pp. 37– 46

[42]Larsen B. D., Christesen N. J.: Simulating photon mapping for real-time applications. In Eurographics Symposium on Rendering (June 2004), pp. 123–132.

[43]Wyman C., Davis S.: Interactive image-space techniques for approximating caustics. In SI3D ’06: Proceedings of the 2006 Symposium on Interact

[44]Shah M. A., Konttinen J., Pattanaik S. N.: Caustics mapping: An image-space technique for real-time caustics. IEEE Transactions on Visualization and Computer Graphics (2006). To appear.

[45]OpenVIDIA: GPU-accelerated computer vision library. 25 Αυγούστου 2008: <http://openvidia.sourceforge.net/, 2006>.

[46]Yang R., Pollefeys M.: A versatile stereo implementation on commodity graphics hardware. Real- Time Imaging 11, 1 (Feb. 2005), 7–18.

[47]Govindaraju N. K., Larsen S., Gray J., Manocha D.: A memory model for scientific algorithms on graphics processors. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (Nov. 2006). To appear.

[48]Sumanaweera T., LIU D.: Medical image reconstruction with the FFT. In GPU Gems 2, Pharr M., (Ed.). Addison Wesley, Mar. 2005, ch. 48, pp. 765–784.

[49]Owens J. D., Sengupta S., Horn D.: Assessment of Graphic Processing Units (GPUs) for Department of Defense (DoD) Digital Signal Processing (DSP) Applications. Tech. Rep. ECE-CE-2005-3, Department of Electrical and Computer Engineering, University of California, Davis, Oct. 2005. <http://www.ece.ucdavis.edu/cerl/techreports/2005-3/>.

[50]Green S.: Image processing tricks in OpenGL. In Proceedings of Game Developers Conference 2005 (Mar. 2005). <http://download.nvidia.com/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_Image_Processing_Tricks.pdf>.

[51]Govindaraju N. K., LIN M. C., Manocha D.: Quick-Cullide: Efficient inter- and intra-object collision culling using graphics hardware. In Proceedings of IEEE Virtual Reality (Mar. 2005), pp. 59–66.

[52]Guha S., Krishnan S., Munagala K.,Venkatasubramanian S.: Application of the twosided depth test to CSG rendering. In 2003 ACM Symposium on Interactive 3D Graphics (Apr. 2003), pp. 177– 180.

[53]Govindaraju N. K., Henson M., Lin M. C., Manocha D.: Interactive visibility ordering of geometric primitives in complex environments. In Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games (Apr. 2005), pp. 49– 56.

[54]Strzodka R., Telea A.: Generalized distance transforms and skeletons in graphics hardware. In Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym ’04) (2004), pp. 221–230.

[55]Harris Mark J., Fast Fluid Dynamics Simulation on the GPU, GPU Gems 3.

[56]Folding@Home Distribyted Computing. 25 Αυγούστου 2008: <http://folding.stanford.edu/English/FAQ-ATI>.

[57]Κεντρικός ιστοτόπος General Purpose Computations Using Graphics Hardware. 25 Αυγούστου 2008: <gpgpu.org>.

[58]S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, and W. W. Hwu. Program optimization study on a 128-core GPU. In *The First Workshop on General Purpose Processing on Graphics Processing Units*, October 2007 .

[59]S. Ryoo, C. I. Rodriguez, S. S. Baghorkhi, S. S. Stone, D. B. Kirk and W. W. Hwu. Optimization Principles and Application Performance Evaluation of a Multi-threaded GPU Using CUDA. In PPoPP, pages 73-82, 2008.

[60] S. Ryoo, C. I. Rodriguez, S. S. Stone, S. S. Baghorkhi, S.-Z. Ueng, J.A. Stratton and W. W. Hwu. Program Optimization Space Pruning for a Multithreaded GPU. In PPoPP, CGO ’08 April 5-10 2008, Boston, Massachusetts.

[61]CUDA zone. 25 Αυγούστου 2008: <http://www.nvidia.com/object/cuda_home.html#>.

[62]First Details on a Future Intel Design Codenamed 'Larrabee'. 25 Αυγούστου 2008: <http://www.intel.com/pressroom/archive/releases/20080804fact.htm?iid=pr1_releasepri_20080804fact>.

[63]nVidia Graphics processors. 25 Αυγούστου 2008: <http-p://www.nvidia.com/object/geforce_family.html>.

[64] Seiler, L., Carmean D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Ju kins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Han-rahan, P., 2008. "Larrabee: A Many-Core x86 Architecture for Visual Computing," ACM Transactions on Graphics, 27, 3, 2008.