ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΥΠΟΣΤΗΡΙΞΗ ΣΕ ΕΠΙΠΕΔΟ ΜΕΤΑΓΛΩΤΤΙΣΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΟΣ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ ΤΟΥ ΜΟΝΤΕΛΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ OpenCL ΣΕ ΕΠΕΞΕΡΓΑΣΤΕΣ ΠΟΛΛΑΠΛΩΝ ΠΥΡΗΝΩΝ ΜΕ ΚΟΙΝΗ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΗ ΜΝΗΜΗ

Compiler and Run-Time Support for OpenCL on Hardware- and Software-Managed Cache Multicores

Μεταπτυχιακή Διατριβή

Κωνσταντής Α. Νταλούκας

Επιβλέποντες Καθηγητές : Νικόλαος Μπέλλας
Αναπληρωτής Καθηγητής

Ιωάννης Κατσαβουνίδης
Αναπληρωτής Καθηγητής

Σπυρίδων – Γεράσιμος Λάλης
Αναπληρωτής Καθηγητής

Βόλος, Ιούλιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ,
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΥΠΟΣΤΗΡΙΞΗ ΣΕ ΕΠΙΠΕΔΟ ΜΕΤΑΓΛΩΤΤΙΣΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΟΣ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ ΤΟΥ ΜΟΝΤΕΛΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ OpenCL ΣΕ ΕΠΕΞΕΡΓΑΣΤΕΣ ΠΟΛΛΑΠΛΩΝ ΠΥΡΗΝΩΝ ΜΕ ΚΟΙΝΗ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΗ ΜΝΗΜΗ

Μεταπτυχιακή Διατριβή

Κωνσταντής Α. Νταλούκας

Επιβλέποντες Καθηγητές : Νικόλαος Μπέλλας
Αναπληρωτής Καθηγητής

Ιωάννης Κατσαβουνίδης
Αναπληρωτής Καθηγητής

Σπυρίδων – Γεράσιμος Λάλης
Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Ιουλίου 2010

.....	.....	.....
Ν. Μπέλλας	Ιωάννης Κατσαβουνίδης	Σπυρίδων – Γεράσιμος Λάλης
Αναπληρωτής Καθηγητής	Αναπληρωτής Καθηγητής	Αναπληρωτής Καθηγητής

Μεταπτυχιακή Διατριβή για την απόκτηση του Μεταπτυχιακού Διπλώματος Ειδίκευσης "Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων" του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

……..

Κωνσταντής Νταλούκας

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Στην οικογένειά μου και στους φίλους μου

Ευχαριστίες

Με την περάτωση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω θερμά τους κύριους επιβλέποντες της μεταπτυχιακής διατριβής κ. Νικόλαο Μπέλλα και κ. Χρήστο Αντωνόπουλο για την εμπιστοσύνη που επέδειξαν στο πρόσωπό μου, την άριστη συνεργασία, την συνεχή καθοδήγηση και τις ουσιώδεις υποδείξεις και παρεμβάσεις, που διευκόλυναν την εκπόνηση της μεταπτυχιακής διατριβής.

Επίσης, θα ήθελα να ευχαριστήσω τους κυρίους Ιωάννη Κατσαβουνίδη και Σπύρο Λάλη που δέχθηκαν να συμμετάσχουν στην τριμελή επιτροπή αξιολόγησης της μεταπτυχιακής διατριβής μου. Οι εύστοχες παρατηρήσεις και διορθώσεις τους συνέβαλλαν σημαντικά στην βελτίωση αυτής.

Ακόμη, οφείλω ένα μεγάλο ευχαριστώ σε όλα τα μέλη της οικογενείας μου για την συμπαράσταση και την ψυχολογική στήριξη. Όντας δίπλα μου τόσο καθ' όλη την διάρκεια των σπουδών μου όσο και κατά την εκπόνηση της παρούσας διατριβής, με περιέβαλλαν με αγάπη, αμέριστη υπομονή και κατανόηση, συμβάλλοντας έτσι στην επιτυχή ολοκλήρωσή τους.

Τέλος, δεν θα μπορούσα να παραλείψω τις ιδιαιτέρως θερμές ευχαριστίες προς όλους τους φίλους μου που με στήριξαν και συνεχίζουν να με στηρίζουν σε αυτή την προσπάθειά μου.

Κωνσταντής Νταλούκας
Βόλος, 5η Ιουλίου 2010

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Συντομογραφιών

API	Application Programming Interface
AST	Abstact Syntax Tree
CBE	Cell Broadband Engine
CPU	Central Processing Unit
CU	Compute Unit
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
EIB	Element Interconnect Bus
ELF	Executable and Linkable Format
GPU	Graphics Processing Unit
HPC	High-Performance Computing
LIFO	Last-In, First-Out
PDE	Partial Differential Equation
PE	Processing Element
PPE	Power Processing Element
SDK	Software Development Kit
SIMD	Single Instruction, Multiple Data
SMT	Simultaneous Multithreading
SPE	Synergistic Processing Element
SPMD	Single Program, Multiple Data
SPU	Synergistic Processing Unit

Περίληψη

Το μοντέλο προγραμματισμού OpenCL αποτελεί ένα πρότυπο προγραμματισμού, με ευρεία υποστήριξη από την βιομηχανία, που στοχεύει στην ανάπτυξη εφαρμογών τόσο για επεξεργαστές πολλαπλών πυρήνων όσο και για συστήματα που βασίζονται σε επιταχυντές (accelerators). Τέτοια συστήματα είναι οι Graphics Processing Units (GPUs) ή τα Synergistic Processing Elements (SPEs) του επεξεργαστή Cell B.E. Η παρούσα εργασία παρουσιάζει την υποδομή GLOpenCL, ένα ενοποιημένο πλαίσιο εργασίας (framework) για την υποστήριξη του μοντέλου OpenCL τόσο σε ομογενείς επεξεργαστές πολλαπλών πυρήνων με κοινή μνήμη, όσο και σε ετερογενείς επεξεργαστές πολλαπλών πυρήνων που ενσωματώνουν κατανεμημένη μνήμη.

Η υποδομή ενσωματώνει έναν μεταγλωττιστή, που βασίζεται στην υποδομή του μεταγλωττιστή LLVM, και μία βιβλιοθήκη συστήματος χρόνου εκτέλεσης (run-time library), διατηρώντας το μεγαλύτερο μέρος της αρχιτεκτονικής κοινό για όλες τις αρχιτεκτονικές προορισμού. Ο μεταγλωττιστής δέχεται ως είσοδο εφαρμογές που έχουν αναπτυχθεί με το μοντέλο OpenCL, εκτελεί μετασχηματισμούς στο επίπεδο του πηγαίου κώδικα της εφαρμογής (source-to-source transformations), οι οποίοι στοχεύουν τόσο στην αποδοτικότητα όσο και στην ορθότητα της εφαρμογής, και προσθέτει κλήσεις σε συναρτήσεις προς την βιβλιοθήκη του συστήματος χρόνου εκτέλεσης. Η βιβλιοθήκη του συστήματος χρόνου εκτέλεσης προσφέρει λειτουργικότητα για δημιουργία, διαχείριση και εκτέλεση του υπολογιστικού μέρους της εφαρμογής αλλά και για μεταφορά των δεδομένων.

Για την αξιολόγηση της υποδομής, χρησιμοποιούμε μία σειρά εφαρμογών (benchmarks) από τα Software Development Kits (SDKs) για την υποστήριξη της OpenCL που παρέχουν η AMD/ATI και η IBM. Από τα αποτελέσματα της αξιολόγησης παρατηρούμε ότι η προτεινόμενη υποδομή επιτυγχάνει παρόμοια ή καλύτερη απόδοση από τις βελτιστοποιημένες υποδομές που στοχεύουν στην υποστήριξη της OpenCL για μία συγκεκριμένη αρχιτεκτονική.

Η OpenCL στοχεύει στο να αποτελέσει ένα πλαίσιο εργασίας για την ανάπτυξη εφαρμογών που μπορούν να εκτελεστούν διαφανώς σε όλες τις αρχιτεκτονικές που υποστηρίζουν το μοντέλο. Εφαρμογές που έχουν αναπτυχθεί με αυτό το μοντέλο μπορούν να μεταφερθούν μεταξύ αρχιτεκτονικών, χωρίς να απαιτούνται τροποποιήσεις στον πηγαίο κώδικα, ώστε να μην είναι απαραίτητη η γνώση των λεπτομερειών χαμηλού επιπέδου της αρχιτεκτονικής για την οποία προορίζεται η εκάστοτε εφαρμογή. Όμως, το πρότυπο επιτρέπει την εφαρμογή πολλών βελτιστοποιήσεων που στοχεύουν στην βελτίωση της απόδοσης μίας εφαρμογής σε μία συγκεκριμένη αρχιτεκτονική. Οι πειραματισμοί μας με διαφορετικές υλοποιήσεις υπολογιστικών πυρήνων (kernels), προσαρμοσμένων για συγκεκριμένες αρχιτεκτονικές, καταδεικνύουν ότι η γνώση της αρχιτεκτονικής και η εφαρμογή αντίστοιχων βελτιστοποιήσεων είναι απαραίτητη σε περίπτωση που η απόδοση της εφαρμογής αποτελεί μεγαλύτερη προτεραιότητα από την ευκολία ανάπτυξης αυτής. Αυτό είναι ιδιαιτέρως σημαντικό για εφαρμογές που προορίζονται για εκτέλεση σε μη συμβατικούς επεξεργαστές πολλαπλών πυρήνων, όπως είναι ο επεξεργαστής Cell Broadband Engine (CBE).

Λέξεις Κλειδιά:
OpenCL, Παράλληλος Προγραμματισμός, Μετασχηματισμοί Πηγαίου Κώδικα, Υποστήριξη Συστήματος Χρόνου Εκτέλεσης, Επεξεργαστές Πολλαπλών Πυρήνων με Διαχείριση Κρυφής Μνήμης από το Υλικό, Επεξεργαστές Πολλαπλών Πυρήνων με Διαχείριση Κρυφής Μνήμης από το Λογισμικό

Abstract

OpenCL is an industry supported standard for writing programs that execute on multicore platforms as well as on accelerators, such as GPUs or the SPEs of the Cell B.E. In this paper we introduce GLOpenCL, a unified development framework which supports OpenCL on both homogeneous, shared memory, as well as on heterogeneous, distributed memory multicores.

The framework consists of a compiler, based on the LLVM compiler infrastructure, and a run-time library, sharing the same basic architecture across all target platforms. The compiler recognizes OpenCL constructs, performs source-to-source code transformations targeting both efficiency and semantical correctness, and adds calls to the run-time library. The latter offers functionality for work creation, management and execution, as well as for data transfers.

We evaluate our framework using benchmarks from the distributions of OpenCL implementations by hardware vendors. We find that our generic system performs comparably or better to customized, platform-specific vendor distributions.

OpenCL is designed and marketed as a write-once run-anywhere software development framework. However, the standard leaves enough room for target platform specific optimizations. Our experimentation with different, customized implementations of kernels reveals that optimized, hardware mapped implementations are both possible and necessary in the context of OpenCL - especially on non-conventional multicores - if performance is considered a higher priority than programmability.

Keywords:

OpenCL, Parallel Programming, Source-to-Source Transformations, Run-time Support, Hardware-Controlled Cache Multi-core Processors, Software-Controlled Cache Multi-core Processors

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του Προβλήματος και Συμβολή της Εργασίας

Τα τελευταία χρόνια, παρατηρούμε μία συνεχή μεταστροφή προς την χρήση παράλληλων αρχιτεκτονικών. Οι τεχνολογικές εξελίξεις έχουν επιτρέψει την ενσωμάτωση πολλαπλών πυρήνων σε ένα υπόστρωμα, με αποτέλεσμα να καταστεί δυνατή η ανάπτυξη μίας πληθώρας παράλληλων, υψηλής απόδοσης αρχιτεκτονικών. Αρχιτεκτονικές όπως οι ομογενείς ή οι ετερογενείς επεξεργαστές πολλαπλών πυρήνων, αλλά και οι GPUs, οι οποίες συγκαταλέγονται στις πιο πρόσφατες παράλληλες αρχιτεκτονικές, έχουν επιτρέψει την εκτέλεση υπολογιστικά πολύπλοκων εφαρμογών με πολύ μικρό κόστος. Λόγω των απαιτήσεων σε υπολογιστική ισχύ και ενέργεια, η εκτέλεση αυτού του είδους των εφαρμογών σε προγενέστερες αρχιτεκτονικές ήταν αρκετά δύσκολη, εάν όχι αδύνατη. Πλέον, με την αξιοποίηση των παράλληλων αρχιτεκτονικών είναι δυνατή η αποδοτική εκτέλεση αυτών τόσο όσον αφορά στον χρόνο εκτέλεσης όσο και στην ενέργεια που απαιτείται. Όμως, για να είναι δυνατή η εκμετάλλευση των δυνατοτήτων που προσφέρουν οι αρχιτεκτονικές επεξεργαστών πολλαπλών πυρήνων αλλά και οι αρχιτεκτονικές που βασίζονται σε επιταχυντές (accelerators), απαιτείται η τροποποίηση των εφαρμογών για την εκμετάλλευση του εγγενούς παραλληλισμού που αυτές εμφανίζουν. Η ανάπτυξη όμως παράλληλων προγραμμάτων είναι ένα αρκετά πιο περίπλοκο εγχείρημα σε σχέση με την ανάπτυξη σειριακών προγραμμάτων. Ο προγραμματιστής είναι αντιμέτωπος με αρκετά εμπόδια όπως οι διάφορες καταστάσεις ανταγωνισμού (race conditions), τα θέματα συγχρονισμού και επικοινωνίας μεταξύ των παράλληλων τμημάτων, η διαχείριση και η δρομολόγηση του υπολογιστικού φορτίου, αλλά και με τα θέματα χαμηλού επιπέδου που άπτονται της εκάστοτε αρχιτεκτονικής.

Προκειμένου να είναι δυνατή η αποδοτική ανάπτυξη εφαρμογών για παράλληλες αρχιτεκτονικές, έχει προταθεί μία πλειάδα από προγραμματιστικά μοντέλα. Αυτά τα προγραμματιστικά μοντέλα έχουν ως στόχο να απαλλάξουν τον προγραμματιστή από την διαχείριση όσο το δυνατόν περισσοτέρων από τα ανωτέρω προβλήματα, ώστε να του επιτρέψουν να επικεντρωθεί περισσότερο σε αλγοριθμικά και λιγότερο σε τεχνικά θέματα. Ανάμεσα στα πιο διαδεδομένα μοντέλα προγραμματισμού που στοχεύουν σε παράλληλες αρχιτεκτονικές συγκαταλέγονται τα μοντέλα PThreads [20], OpenMP [24], UPC [40], ή το μοντέλο Cilk [6] για τον προγραμματισμό ομογενών ή ετερογενών επεξεργαστών πολλαπλών πυρήνων, το περιβάλλον ανάπτυξης εφαρμογών Cell SDK [15] και το μοντέλο CellSs [25] που στοχεύουν στην ανάπτυξη εφαρμογών για τον επεξεργαστή Cell B.E. [14], όπως επίσης και τα μοντέλα CUDA [23] και AMD/ATI Stream SDK [3] για την ανάπτυξη εφαρμογών για τις GPUs που παρέχονται από την NVidia και την ATI.

Παρά τον μεγάλο αριθμό από διαθέσιμα προγραμματιστικά μοντέλα, κανένα από αυτά δεν

μπορεί να χρησιμοποιηθεί για την ταυτόχρονη ανάπτυξη εφαρμογών που πρόκειται να εκτελεστούν τόσο σε επεξεργαστές πολλαπλών πυρήνων, ομογενείς ή ετερογενείς, αλλά και σε συστήματα που βασίζονται σε επιταχυντές. Αυτό οφείλεται στις μεγάλες αρχιτεκτονικές διαφορές μεταξύ αυτών των συστημάτων. Επί παραδείγματι, οι εφαρμογές που προορίζονται για εκτέλεση στον επεξεργαστή Cell αναπτύσσονται κυρίως με χρήση του IBM SDK. Το εν λόγω SDK είναι προσαρμοσμένο για τον επεξεργαστή Cell, με αποτέλεσμα η μεταφορά των εφαρμογών σε μία διαφορετική αρχιτεκτονική να αποτελεί ένα αρκετά χρονοβόρο εγχείρημα. Επομένως, η ανάγκη για ένα προγραμματιστικό μοντέλο, το οποίο θα επιτρέπει την ανάπτυξη εφαρμογών χωρίς να είναι απαραίτητη η εκ των προτέρων γνώση της αρχιτεκτονικής στην οποία πρόκειται να εκτελεστεί μία εφαρμογή, αναδεικνύεται περισσότερο από σημαντική. Ένα τέτοιο προγραμματιστικό μοντέλο θα διευκολύνει σε μεγάλο βαθμό την φορητότητα (portability) των εφαρμογών μεταξύ αρχιτεκτονικών καθώς δεν θα απαιτούνται μεγάλης κλίμακας τροποποιήσεις. Επομένως, ο προγραμματιστής θα μπορεί να αναπτύξει την εφαρμογή χρησιμοποιώντας το εν λόγω μοντέλο και έπειτα να την εκτελέσει σε όλες τις αρχιτεκτονικές που υποστηρίζουν αυτό το μοντέλο, επιλέγοντας την καταλληλότερη από αυτές. Ένα τέτοιο μοντέλο είναι το μοντέλο προγραμματισμού OpenCL [17]. Το μοντέλο OpenCL είναι το αποτέλεσμα μίας προσπάθειας, με ευρεία υποστήριξη από την βιομηχανία, για την ανάπτυξη ενός προγραμματιστικού μοντέλου και της αντίστοιχης υποδομής για ετερογενείς αρχιτεκτονικές πολλαπλών πυρήνων που ενσωματώνουν Central Processing Units (CPUs), αλλά και διαφόρων ειδών επιταχυντές.

Η παρούσα εργασία αναλύει την σχεδίαση και παρουσιάζει την υλοποίηση της υποδομής GLOpenCL (GLobal OpenCL). Η υποδομή GLOpenCL αποτελεί μία ενοποιημένη υποδομή, που ενσωματώνει έναν μεταγλωττιστή και μία βιβλιοθήκη συστήματος χρόνου εκτέλεσης (run-time library). Η υποδομή GLOpenCL αποτελεί μία από τις πρώτες υλοποιήσεις του προτύπου OpenCL που επιτρέπει την εγγενή εκτέλεση εφαρμογών που έχουν αναπτυχθεί με αυτό το μοντέλο σε συστήματα με διαφορετική αρχιτεκτονική. Πιο συγκεκριμένα, η υποδομή επιτρέπει την εκτέλεση εφαρμογών τόσο σε αρχιτεκτονικές όπου η ιεραρχία μνήμης (memory hierarchy) ελέγχεται από το υλικό (hardware-controlled memory hierarchy) όσο και σε αρχιτεκτονικές όπου η ιεραρχία μνήμης ελέγχεται από το λογισμικό (software-controlled memory hierarchy). Η εργασία εξετάζει λεπτομερώς τους μετασχηματισμούς που είναι απαραίτητοι σε επίπεδο μεταγλωττιστή και την υποστήριξη που είναι απαραίτητη σε επίπεδο συστήματος χρόνου εκτέλεσης προκειμένου να επιτευχθεί η προαναφερθείσα λειτουργικότητα. Επίσης, παρουσιάζει όλες τις απαραίτητες σχεδιαστικές αποφάσεις που λήφθηκαν κατά την διάρκεια σχεδίασης της υποδομής.

Παρόμοιες υλοποιήσεις του μοντέλου OpenCL περιλαμβάνουν τις OpenCL SDK [16] από την IBM και το ενοποιημένο περιβάλλον ανάπτυξης εφαρμογών CUDA/OpenCL [22] που παρέχεται από την NVidia. Αυτές οι υλοποιήσεις είναι βελτιστοποιημένες για την εκτέλεση εφαρμογών, που έχουν αναπτυχθεί βάσει του μοντέλου OpenCL, στον επεξεργαστή Cell B.E και σε GPUs από την NVidia αντίστοιχα. Επίσης, η υποδομή AMD/ATI Stream SDK [3] από την AMD και την ATI υποστηρίζει την εκτέλεση εφαρμογών OpenCL τόσο σε CPUs που βασίζονται στην αρχιτεκτονική x86 όσο και σε GPUs που κατασκευάζονται από την ATI. Όμως, όπως προκύπτει από τα πειραματικά αποτελέσματα στο Κεφάλαιο 5, η υποδομή GLOpenCL επιτυγχάνει καλύτερη απόδοση από αυτή για CPUs που βασίζονται στην αρχιτεκτονική x86. Τέλος, η υποδομή MCUDA [28] προσφέρει αποδοτική εκτέλεση εφαρμογών που έχουν αναπτυχθεί με το μοντέλο προγραμματισμού CUDA σε CPUs που βασίζονται στην αρχιτεκτονική x86. Η υποδομή αυτή διαφέρει από την υποδομή που περιγράφεται στην εργασία καθώς επιτρέπει την εκτέλεση εφαρμογών μόνο σε μία συγκεκριμένη αρχιτεκτονική. Πιο συγκεκριμένα, επιτρέπει την εκτέλεση εφαρμογών που έχουν αναπτυχθεί με το μοντέλο προγραμματισμού CUDA μόνο σε επεξεργαστές πολλαπλών πυρήνων, με ελεγχόμενη από το υλικό ιεραρχία μνήμης. Επίσης, βασίζεται μόνο σε μετασχηματι-

σμούς στο επίπεδο του μεταγλωττιστή, χωρίς να παρέχει υποστήριξη σε επίπεδο χρόνου εκτέλεσης, και απευθύνεται σε εφαρμογές που ακολουθούν το μοντέλο προγραμματισμού CUDA αντί του μοντέλου OpenCL.

Για την αξιολόγηση της υποδομής GLOpenCL εκτελέσαμε μία σειρά από μετρικές εφαρμογές σύγκρισης (benchmarks) σε δύο αντιπροσωπευτικές αρχιτεκτονικές, και πιο συγκεκριμένα σε έναν επεξεργαστή Core i7 (Nehalem) από την Intel και στον επεξεργαστή Cell B.E. Από τα αποτελέσματα παρατηρήσαμε ότι η υποδομή GLOpenCL επιτυγχάνει συγκρίσιμη ή ακόμη και καλύτερη απόδοση από τις υλοποιήσεις που είναι βελτιστοποιημένες για υποστήριξη του μοντέλου σε μία συγκεκριμένη αρχιτεκτονική. Η προτεινόμενη υποδομή επιτυγχάνει μέση επιτάχυνση σε σχέση με την υποδομή που προσφέρεται από την AMD και την ATI ίση με 1.84x, με την μέγιστη επιτάχυνση να είναι ίση με 2.67x. Συγκρινόμενη με την υποδομή που παρέχεται από την IBM, η υποδομή GLOpenCL επιτυγχάνει μικρή επιβάρυνση στον χρόνο εκτέλεσης της εφαρμογής, κυρίως για εφαρμογές με μεγάλες απαιτήσεις σε υπολογιστική ισχύ (compute-bound applications). Βάσει των αποτελεσμάτων της διαδικασίας αξιολόγησης, μπορούμε να συμπεράνουμε ότι η προαναφερθείσα επιβάρυνση προέρχεται από το εξωτερικό, επιπρόσθετο τμήμα που χρησιμοποιείται για να παρέχει έλεγχο του υποσυστήματος μνήμης του επεξεργαστή Cell, δηλαδή τόσο της κύριας μνήμης όσο και της Local Store των SPEs, μέσω λογισμικού. Αυτό το τμήμα λογισμικού δεν αποτελεί τμήμα του συστήματος χρόνου εκτέλεσης και χρησιμοποιείται επικουρικά για την διαχείριση της μνήμης, όπως περιγράφεται στην ενότητα 4.3.

Το μοντέλο OpenCL στοχεύει στο να αποτελέσει ένα προγραμματιστικό μοντέλο που θα επιτρέπει την αποδοτική εκτέλεση εφαρμογών σε ετερογενείς αρχιτεκτονικές, ενώ παράλληλα θα επιτρέπει την φορητότητα των εφαρμογών. Παρ' όλα αυτά, η πειραματική αξιολόγηση με μία σειρά εφαρμογών αναδεικνύει ότι γνώση της αρχιτεκτονικής, στην οποία πρόκειται να εκτελεστεί η εφαρμογή, σε συνδυασμό με ανάπτυξη της εφαρμογής η οποία θα λαμβάνει υπόψη την αποδοτική εκτέλεση, μπορεί να οδηγήσει σε αύξηση της απόδοσης αυτής. Αυτό το χαρακτηριστικό είναι ιδιαίτερα εμφανές κυρίως σε ετερογενείς αρχιτεκτονικές.

1.2 Διάρθρωση της Διπλωματικής Εργασίας

Στο Κεφάλαιο 2 περιγράφουμε το μοντέλο προγραμματισμού OpenCL και αναλύουμε τα κύρια χαρακτηριστικά αυτού.

Το Κεφάλαιο 3 περιγράφει την υποστήριξη, που είναι απαραίτητη στο επίπεδο του μεταγλωττιστή, προκειμένου να επιτευχθεί η αποδοτική εκτέλεση OpenCL εφαρμογών. Σε αυτό το κεφάλαιο αναλύονται όλοι οι μετασχηματισμοί, στο επίπεδο του πηγαίου κώδικα της εφαρμογής, που είναι απαραίτητοι καθώς και τα χαρακτηριστικά του μοντέλου που καθιστούν επιβλητική την ύπαρξη αυτών.

Εν συνεχεία, στο Κεφάλαιο 4 παρουσιάζεται η αρχιτεκτονική του συστήματος χρόνου εκτέλεσης που ενσωματώνει η υποδομή. Το κεφάλαιο περιγράφει την υλοποίηση του συστήματος για κάθε μία από τις αρχιτεκτονικές που υποστηρίζονται και αναλύονται οι λεπτομέρειες υλοποίησης των επιμέρους τμημάτων.

Έπειτα, στο Κεφάλαιο 5 πραγματοποιείται η αξιολόγηση της υποδομής, χρησιμοποιώντας μία σειρά εφαρμογών, σε δύο συστήματα επεξεργαστών πολλαπλών πυρήνων. Η αξιολόγηση πραγματοποιήθηκε τόσο σε έναν ομογενή επεξεργαστή με διαχείριση του υποσυστήματος μνήμης από το υλικό όσο και σε έναν ετερογενή επεξεργαστή με διαχείριση μνήμης από το λογισμικό.

Τέλος, το Κεφάλαιο 6 παρουσιάζει τα κύρια συμπεράσματα που προέκυψαν από την παρούσα εργασία.

Κεφάλαιο 2

Το Μοντέλο Προγραμματισμού OpenCL

2.1 Εισαγωγή

Οι νέες αρχιτεκτονικές των επεξεργαστών έχουν υιοθετήσει τον παραλληλισμό ως το μέσο για την αύξηση της απόδοσης των εφαρμογών. Οι τεχνικές προκλήσεις που εμφανίζονται με την αύξηση της ταχύτητας του επεξεργαστή, δεδομένης μίας σταθερής τιμής όσον αφορά στην κατανάλωση ισχύος, έχουν οδηγήσει τους σχεδιαστές μικροεπεξεργαστών στην κατασκευή επεξεργαστών με μεγαλύτερο αριθμό από απλούστερους πυρήνες στο ίδιο chip. Πιο πρόσφατα, και οι Μονάδες Επεξεργασίας Γραφικών (Graphics Processing Units (GPUs)) εξελίχθηκαν από συσκευές που χρησιμοποιούνται μόνο για συγκεκριμένες λειτουργίες απεικόνισης γραφικών σε προγραμματιζόμενους, παράλληλους επεξεργαστές. Αυτό είχε ως αποτέλεσμα τα σύγχρονα υπολογιστικά συστήματα να ενσωματώνουν συχνά αρκετές παράλληλες μονάδες επεξεργασίας, όπως CPUs, GPUs αλλά και άλλων ειδών επεξεργαστές ή επιταχυντές (accelerators). Επομένως, η διευκόλυνση του προγραμματιστή εφαρμογών προκειμένου να είναι εφικτή η πλήρης εκμετάλλευση των δυνατοτήτων που προσφέρουν αυτές οι αρχιτεκτονικές καθίσταται ιδιαιτέρως σημαντική.

Η ανάπτυξη εφαρμογών για ετερογενείς, παράλληλες αρχιτεκτονικές είναι ένα ιδιαιτέρως δύσκολο εγχείρημα καθώς οι παραδοσιακές προσεγγίσεις για τον προγραμματισμό επεξεργαστών πολλαπλών πυρήνων και GPUs διαφέρουν σε πολύ μεγάλο βαθμό. Η πλειοψηφία των παράλληλων προγραμματιστικών μοντέλων για CPUs βασίζεται στο μοντέλο κοινού χώρου διευθύνσεων μνήμης (shared address space), ενώ τα περισσότερα δεν ενσωματώνουν πράξεις μεταξύ διανυσμάτων (vector operations). Σε αντίθεση, τα προγραμματιστικά μοντέλα γενικού σκοπού για GPUs λαμβάνουν μέριμνα για πολύπλοκες ιεραρχίες μνήμης, παρέχουν διανυσματικές πράξεις, αλλά είναι προσαρμοσμένα για συγκεκριμένες αρχιτεκτονικές ή για συγκεκριμένο υλικό. Αυτοί οι περιορισμοί καθιστούν αδύνατη την αξιοποίηση της υπολογιστικής ισχύος ετερογενών συστημάτων που ενσωματώνουν CPUs, GPUs και άλλα είδη επιταχυντών χρησιμοποιώντας έναν μοναδικό πηγαίο κώδικα της εφαρμογής, που έχει αναπτυχθεί για μία συγκεκριμένη αρχιτεκτονική, σε διαφορετικές αρχιτεκτονικές. Επομένως, η ανάπτυξη της κατάλληλης υποδομής που θα επιτρέψει στον προγραμματιστή εφαρμογών να αξιοποιήσει πλήρως τα πλεονεκτήματα των ετερογενών αρχιτεκτονικών, οι οποίες θα ενσωματώνουν επεξεργαστές πολλαπλών πυρήνων, GPUs και άλλους επεξεργαστές, όπως Digital Signal Processors (DSPs) ή ο επεξεργαστής Cell B.E., είναι περισσότερο από επιτακτική.

Το μοντέλο προγραμματισμού OpenCL (Open Computing Language) [17] είναι ένα ανοικτό

πρότυπο που στοχεύει στον γενικού σκοπού προγραμματισμό παράλληλων αρχιτεκτονικών. Το μοντέλο αντιμετωπίζει τα προβλήματα του προγραμματισμού και της μεταφερσιμότητας των εφαρμογών μεταξύ αρχιτεκτονικών παρέχοντας μία υποδομή για παράλληλο προγραμματισμό, κατάλληλη τόσο για επεξεργαστές γενικού σκοπού, που ενσωματώνουν πολλαπλούς πυρήνες, όσο και για άλλες αρχιτεκτονικές επιταχυντών, όπως είναι οι GPUs ή ο επεξεργαστής Cell B.E. Ιδανικά, οι εφαρμογές που αναπτύσσονται βάσει αυτού του μοντέλου, μπορούν να μεταφερθούν χωρίς κάποια τροποποίηση μεταξύ διαφορετικών αρχιτεκτονικών που παρέχουν υποστήριξη για την υποδομή.

Το μοντέλο OpenCL παρέχει μία χαμηλού επιπέδου, αλλά υψηλής απόδοσης και φορητή, αφαίρεση του υλικού με αποτέλεσμα να μπορεί να υποστηρίξει ένα ευρύ πλήθος εφαρμογών, από εφαρμογές για ενσωματωμένα συστήματα και εφαρμογές ευρείας κατανάλωσης μέχρι εφαρμογές για επιστημονικούς υπολογισμούς υψηλών επιδόσεων (High-Performance Computing (HPC)). Δημιουργώντας μία αποδοτική και χαμηλού επιπέδου προγραμματιστική διεπαφή, το μοντέλο OpenCL θα αποτελέσει την βάση για ένα σύνολο εργαλείων, ενδιάμεσου λογισμικού (middleware) και εφαρμογών, το οποίο δεν θα εξαρτάται από την εκάστοτε αρχιτεκτονική.

Το κεφάλαιο αυτό πρώτα περιγράφει τις βασικές έννοιες και την αρχιτεκτονική του μοντέλου OpenCL και έπειτα παρέχει μία λεπτομερή περιγραφή του μοντέλου εκτέλεσης και του μοντέλου μνήμης καθώς και του μοντέλου συγχρονισμού που παρέχει η υποδομή.

2.2 Μοντέλο Αρχιτεκτονικής

Το σχήμα 2.1 παρουσιάζει το μοντέλο αρχιτεκτονικής της υποδομής OpenCL. Όπως παρατηρούμε, το μοντέλο περιλαμβάνει έναν host συνδεδεμένο με μία ή περισσότερες συσκευές που υποστηρίζουν το μοντέλο OpenCL. Μία τέτοια συσκευή χωρίζεται περαιτέρω σε μία ή περισσότερες υπολογιστικές μονάδες (Compute Units (CUs)), με κάθε μία CU να διαιρείται σε ένα ή περισσότερα επεξεργαστικά στοιχεία (Processing Elements (PEs)). Τα επεξεργαστικά στοιχεία είναι αυτά που εκτελούν το μεγαλύτερο μέρος του υπολογιστικού τμήματος μίας εφαρμογής.

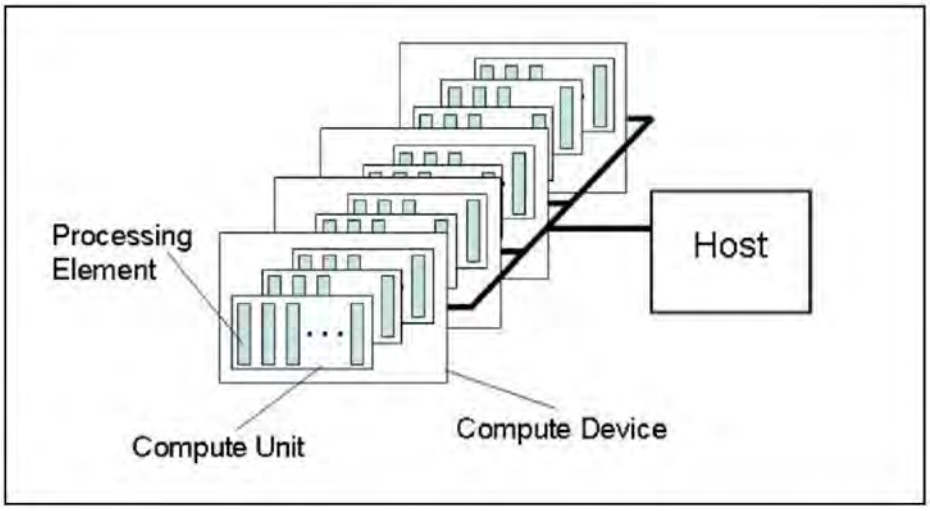
Μία εφαρμογή που έχει αναπτυχθεί βάσει του μοντέλου OpenCL εκτελείται στον host βάσει των εγγενών μοντέλων της αρχιτεκτονικής αυτού. Η εκτέλεση υπολογισμών στα PEs μίας συσκευής επιτυγχάνεται μέσω εντολών (commands) που υποβάλλει το τμήμα της εφαρμογής που εκτελείται στον host. Τα επεξεργαστικά στοιχεία μίας υπολογιστικής μονάδος εκτελούν μία μοναδική ακολουθία εντολών είτε ως μονάδες Single Instruction, Multiple Data (SIMD) είτε ως μονάδες Single Program, Multiple Data (SPMD).

2.3 Μοντέλο Εκτέλεσης

Μία OpenCL εφαρμογή αποτελείται από δύο μέρη: τους kernels που εκτελούνται σε μία ή περισσότερες συσκευές και το τμήμα της εφαρμογής που εκτελείται στον host. Το τελευταίο είναι αυτό που καθορίζει το πλαίσιο εκτέλεσης (context) και διαχειρίζεται τους kernels.

Ο πυρήνας του μοντέλου εκτέλεσης καθορίζεται από τον τρόπο με τον οποίο εκτελούνται οι kernels μίας εφαρμογής. Όταν το τμήμα της εφαρμογής που εκτελείται στον host υποβάλλει προς εκτέλεση έναν kernel δημιουργείται ένας χώρος δεικτών (index space). Ο χώρος δεικτών είναι δύο επιπέδων και τριών διαστάσεων[1]. Κάθε σημείο εντός αυτού του χώρου εκτελεί ένα στιγμιότυπο του αντίστοιχου kernel. Αυτό το στιγμιότυπο του kernel ονομάζεται στοιχείο εκτέλεσης (work-item) και αναγνωρίζεται από το σημείο, εντός του χώρου δεικτών, στο οποίο αντιστοιχεί. Σε κάθε

[1] Ο τρέχων μέγιστος αριθμός διαστάσεων που υποστηρίζεται από το πρότυπο OpenCL είναι τρεις διαστάσεις.

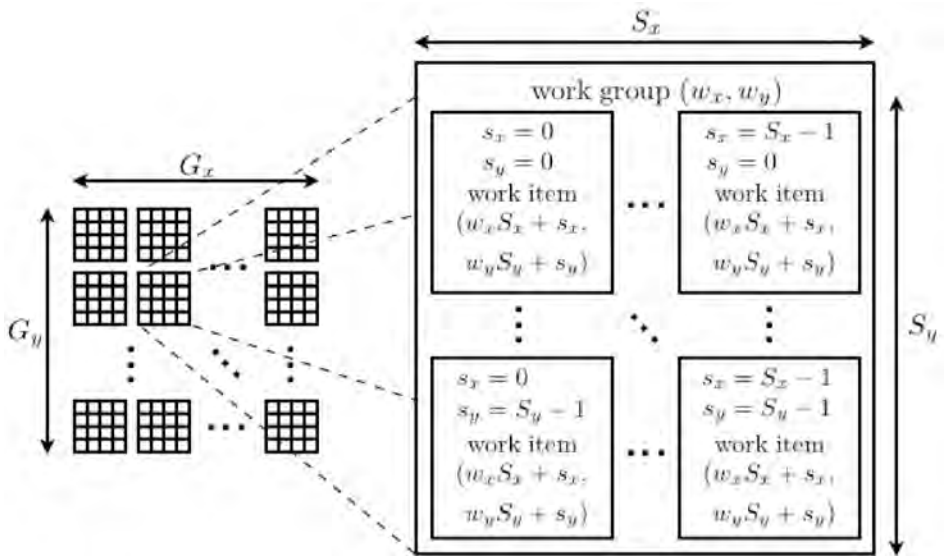


Σχήμα 2.1: Το μοντέλο αρχιτεκτονικής του μοντέλου OpenCL.

στοιχείο εκτέλεσης ανατίθεται μία μοναδική πλειάδα τριών αναγνωριστικών (τρισδιάστατος χώρος αναγνωριστικών εντός της γεωμετρίας - global ID). Όλα τα στοιχεία εκτέλεσης εκτελούν τον ίδιο κώδικα, αλλά τα δεδομένα καθώς και η συγκεκριμένη ακολουθία εντολών δύνανται να διαφέρουν μεταξύ στοιχείων εκτέλεσης.

Τα στοιχεία εκτέλεσης ομαδοποιούνται σε μεγαλύτερες μονάδες που ονομάζονται σύνολα εργασίας (work-groups). Τα σύνολα εργασίας παρέχουν μία λιγότερο λεπτή διαμέριση του index space. Κάθε work-group μπορεί να έχει μέχρι τρεις διαστάσεις. Επομένως, ο υπολογισμός μπορεί να τεμαχιστεί σε work-groups, τα οποία οργανώνονται σε έναν χώρο με μέγιστο αριθμό διαστάσεων ίσο με τρεις. Σε κάθε work-group ανατίθεται ένα μοναδικό αναγνωριστικό με διάσταση ίδια με αυτή των αναγνωριστικών των work-items (work-group ID). Επιπλέον, μοναδικά αναγνωριστικά ανατίθενται και στα work-items εντός ενός work-group (local ID). Αυτό έχει σαν αποτέλεσμα ένα work-item να προσδιορίζεται μοναδικά είτε από το global ID είτε από έναν συνδυασμό των work-group ID και local ID. Τα work-items που ανήκουν σε ένα work-group εκτελούνται παράλληλα σε μία συγκεκριμένη υπολογιστική μονάδα.

Ο χώρος δεικτών που υποστηρίζεται από το πρότυπο OpenCL ονομάζεται NDRange. Ένα NDRange είναι ένας χώρος δεικτών με διάσταση ίση με N, όπου το N ισούται με ένα, δύο ή τρία. Ένα NDRange ορίζεται από ένα διάνυσμα ακεραίων, μεγέθους ίσου με N, το οποίο ορίζει το εύρος του χώρου δεικτών σε κάθε διάσταση. Τα αναγνωριστικά global ID και local ID ενός work-item αποτελούν και αυτά διανύσματα μεγέθους ίσου με N. Οι συνιστώσες των global ID αναγνωριστικών λαμβάνουν τιμές που κυμαίνονται από την τιμή μηδέν μέχρι το μέγεθος της αντίστοιχης διάστασης μείον ένα. Τα αναγνωριστικά των work-groups ανατίθενται σε αυτά με τρόπο όμοιο με αυτόν που χρησιμοποιείται για τα global IDs των work-items. Τα work-items ανατίθενται σε ένα work-group και λαμβάνουν local ID αναγνωριστικά με συνιστώσες των οποίων οι τιμές κυμαίνονται από την τιμή μηδέν μέχρι το μέγεθος της αντίστοιχης διάστασης του work-group μείον ένα. Επομένως, όπως αναφέρθηκε και παραπάνω, ο συνδυασμός των local ID και work-group ID προσδιορίζει μοναδικά ένα work-item. Συνεπώς, κάθε work-item μπορεί να προσδιορισθεί είτε βάσει του global ID είτε βάσει του συνδυασμού των local και work-group ID. Όπως μπορούμε να συμπεράνουμε, ένα μεγάλο εύρος προγραμματιστικών μοντέλων μπορούν να απεικονισθούν σε



Σχήμα 2.2: Το μοντέλο εκτέλεσης των kernel συναρτήσεων στο μοντέλο OpenCL.

αυτό το μοντέλο εκτέλεσης. Ενδεικτικά παραδείγματα αυτών αποτελούν τα μοντέλα data parallel και task parallel.

Η παραπάνω περιγραφή απεικονίζεται στο σχήμα 2.2, για έναν kernel που εκτελείται σε δύο διαστάσεις. Όπως παρατηρούμε, ο προγραμματιστής έχει ορίσει τις διαστάσεις του χώρου, G_x και G_y, καθώς και το μέγεθος του κάθε work-group (S_x και S_y). Τα work-items ομαδοποιούνται σε work-groups, με κάθε work-group να λαμβάνει ένα μοναδικό id δύο διαστάσεων (w_x, w_y). Τέλος, εντός του work-group, τα work-items λαμβάνουν μοναδικά ids, τα οποία απεικονίζονται στο σχήμα ως s_x και s_y.

Πλαίσια Εκτέλεσης και Ουρές Εντολών

Τα δομικά στοιχεία του μοντέλου εκτέλεσης του προτύπου OpenCL είναι οι kernels, οι ουρές εντολών (command queues) και οι περιοχές μνήμης προσωρινής αποθήκευσης (buffers). Πριν την εκτέλεση ενός kernel, το τμήμα της εφαρμογής που εκτελείται στον host αναλαμβάνει να δημιουργήσει ένα πλαίσιο εκτέλεσης για τον συγκεκριμένο kernel. Αυτό το πλαίσιο εκτέλεσης περιλαμβάνει τους παρακάτω πόρους:

- **Συσκευές** (**Devices**): Η συλλογή των συσκευών που θα χρησιμοποιηθούν από τον host.
- **Kernels**: Οι OpenCL συναρτήσεις που θα εκτελεστούν στις συσκευές.
- **Program Objects**: Ο πηγαίος κώδικας και το εκτελέσιμο αρχείο του τμήματος της εφαρμογής που υλοποιεί τον εκάστοτε kernel.
- **Αντικείμενα Μνήμης** (**Memory Objects**): Ένα σύνολο από περιοχές μνήμης που είναι ορατές στον host και στις συσκευές. Σε αυτές τις περιοχές αποθηκεύονται δεδομένα που θα χρησιμοποιηθούν από τα στιγμιότυπα των kernels.

Υπεύθυνος για την δημιουργία και τη διαχείριση των πλαισίων εκτέλεσης είναι ο host. Οι λειτουργίες αυτές πραγματοποιούνται μέσω του OpenCL Application Programming Interface (API). Ο host δημιουργεί μία command queue για τον συντονισμό της εκτέλεσης των kernels στις συσκευές και χρησιμοποιεί την command queue για την τοποθέτηση εντολών οι οποίες δρομολογούνται προς εκτέλεση στις συσκευές που είναι διαθέσιμες στο πλαίσιο εκτέλεσης. Μία εντολή μπορεί να ανήκει στις ακόλουθες κατηγορίες εντολών:

- **Εντολές Εκτέλεσης Kernels** (**Kernel Execution Commands**): Με αυτές τις εντολές επιτυγχάνεται η εκτέλεση ενός kernel στα επεξεργαστικά στοιχεία μίας συσκευής.
- **Εντολές Διαχείρισης Μνήμης** (**Memory Commands**): Αυτές οι εντολές χρησιμοποιούνται για μεταφορά δεδομένων από, προς ή μεταξύ αντικειμένων μνήμης καθώς και για λειτουργίες απεικόνισης (map - unmap operations) των αντικειμένων στην περιοχή της μνήμης του host.
- **Εντολές Συγχρονισμού** (**Synchronization Commands**): Οι εντολές αυτές χρησιμοποιούνται για τον συγχρονισμό της εκτέλεσης των kernels.

Μία command queue δρομολογεί εντολές προς εκτέλεση σε μία συσκευή. Οι εντολές αυτές εκτελούνται ασύγχρονα μεταξύ του host και της συσκευής. Η σχετική εκτέλεση μεταξύ των εντολών μπορεί να πραγματοποιείται βάσει των ακόλουθων μεθόδων:

- **Σειριακή Εκτέλεση** (**In-order Execution**): Οι εντολές εκτελούνται βάσει της σειράς με την οποία τοποθετούνται στην command queue και περατώνουν την εκτέλεσή τους με την ίδια σειρά. Αυτή η μέθοδος έχει ως αποτέλεσμα την σειριοποίηση της εκτέλεσης των διαθέσιμων εντολών.
- **Εκτέλεση εκτός σειράς** (**Out-of-order Execution**): Σε αυτή την μέθοδο εκτέλεσης, οι εντολές εισάγονται σειριακά στην command queue αλλά δεν αναβάλουν την εκτέλεσή τους μέχρι να εκτελεστούν οι προγενέστερες εντολές. Πιθανοί περιορισμοί στην ακολουθία εκτέλεσης επιβάλλονται ρητά από τον προγραμματιστή, με χρήση των αντίστοιχων εντολών συγχρονισμού.

Οι εντολές εκτέλεσης ενός kernel και οι εντολές διαχείρισης μνήμης που εισάγονται σε μία command queue ενδέχεται να παράγουν κάποια γεγονότα (events). Αυτά μπορούν να χρησιμοποιηθούν για τον έλεγχο της εκτέλεσης των εντολών καθώς και για τον συγχρονισμό μεταξύ του host και των συσκευών.

Κατηγορίες Kernels

Το μοντέλο εκτέλεσης του OpenCL υποστηρίζει δύο κατηγορίες από kernels:

- **OpenCL Kernels**: Αυτού του είδους οι kernels έχουν αναπτυχθεί χρησιμοποιώντας την γλώσσα προγραμματισμού OpenCL και έχουν μεταγλωττιστεί με χρήση του αντίστοιχου OpenCL μεταγλωττιστή. Κάθε υλοποίηση του προτύπου θα πρέπει να ενσωματώνει έναν τέτοιο μεταγλωττιστή.
- **Native Kernels**: Αυτοί οι kernels προσπελαύνονται βάσει ενός δείκτη σε συνάρτηση και εισάγονται προς εκτέλεση σε μία συσκευή μαζί με άλλους OpenCL kernels. Θα πρέπει να αναφερθεί ότι η ικανότητα εκτέλεσης των native kernels είναι μία προαιρετική λειτουργία

που προσφέρει το πρότυπο και η σημασιολογία της εκτέλεσης εξαρτάται από την εκάστοτε υλοποίηση.

2.4 Μοντέλο Μνήμης

Τα work-items που ανήκουν σε έναν kernel έχουν πρόσβαση στις ακόλουθες τέσσερις διακριτές περιοχές μνήμης:

- **Καθολική Μνήμη** (**Global Memory**): Αυτή η περιοχή μνήμης επιτρέπει τόσο πρόσβαση για εγγραφή όσο και για ανάγνωση σε όλα τα work-items του χώρου δεικτών. Κάθε work-item έχει την δυνατότητα εγγραφής από και ανάγνωση σε κάθε στοιχείο που ανήκει σε αυτού του είδους τα αντικείμενα μνήμης.

- **Μη εγγράψιμη, σταθερή μνήμη** (**Constant Memory**): Αυτή η περιοχή μνήμης αποτελεί τμήμα της καθολικής μνήμης και τα περιεχόμενά της δεν μεταβάλλονται κατά την διάρκεια της εκτέλεσης της εφαρμογής. Ο host είναι υπεύθυνος για την δέσμευση και την αρχικοποίηση των αντικειμένων που τοποθετούνται σε αυτή την περιοχή της μνήμης.

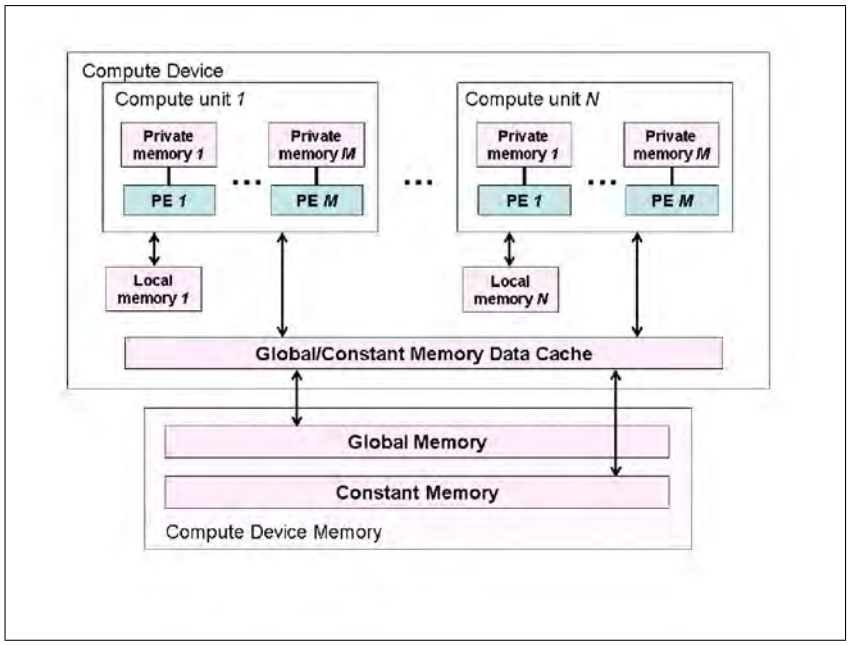
- **Τοπική Μνήμη** (**Local Memory**): Η πρόσβαση σε αυτού του είδους τις περιοχές μνήμης επιτρέπεται μόνο στα work-items του αντίστοιχου work-group. Αυτές οι περιοχές μπορούν να χρησιμοποιηθούν για την αποθήκευση μεταβλητών που διαμοιράζονται από τα work-items που ανήκουν σε ένα work-group. Μόνο τα work-items που ανήκουν σε ένα work-group έχουν την δυνατότητα να επικοινωνήσουν άμεσα μεταξύ τους και αυτό επιτυγχάνεται μέσω των περιοχών τοπικής μνήμης.

- **Ιδιωτική Μνήμη** (**Private Memory**): Η ιδιωτική μνήμη είναι περιοχή της μνήμης που προσπελαύνεται μόνο από ένα work-item. Οι μεταβλητές που αποθηκεύονται σε αυτή την περιοχή της μνήμης δεν είναι ορατές από τα υπόλοιπα work-items.

Το σχήμα 2.3 παρουσιάζει τις ανωτέρω περιοχές μνήμης και την απεικόνισή τους στο μοντέλο της αρχιτεκτονικής του προτύπου OpenCL. Το τμήμα της εφαρμογής που εκτελείται στον host χρησιμοποιεί τις εντολές που παρέχονται από το OpenCL API για την δημιουργία αντικειμένων καθολικής μνήμης και για την εισαγωγή εντολών διαχείριση μνήμης (όπως αυτές περιγράφηκαν στην ενότητα 2.3), οι οποίες επεξεργάζονται αυτά τα αντικείμενα μνήμης.

Οι περιοχές μνήμης του host και των συσκευών είναι ανεξάρτητες για το μεγαλύτερο μέρος της εκτέλεσης της εφαρμογής. Υπάρχουν όμως σημεία που αυτές πρέπει να αλληλεπιδράσουν για την ανταλλαγή δεδομένων. Αυτή η αλληλεπίδραση επιτυγχάνεται είτε με αντιγραφή δεδομένων είτε με απεικόνιση αυτών των αντικειμένων μνήμης στην μνήμη του host.

Η αντιγραφή των δεδομένων επιτυγχάνεται με την χρήση εντολών για την μεταφορά δεδομένων μεταξύ των αντικειμένων μνήμης και της μνήμης του host. Οι εντολές αυτές εισάγονται στην command queue και μπορεί να είναι σύγχρονες ή ασύγχρονες. Η κλήση μίας σύγχρονης εντολής για μεταφορά δεδομένων επιστρέφει όταν έχει ολοκληρωθεί η μεταφορά, οπότε η περιοχή μνήμης του host μπορεί να χρησιμοποιηθεί ασφαλώς. Για μία ασύγχρονη εντολή, η αντίστοιχη συνάρτηση επιστρέφει μόλις η εντολή εισαχθεί στην command queue, ανεξάρτητα από το κατά πόσο η μνήμη μπορεί να χρησιμοποιηθεί. Παρόμοια είναι και η λειτουργία των εντολών για απεικόνιση των αντικειμένων μνήμης στην κύρια περιοχή μνήμης του host.



Σχήμα 2.3: Το μοντέλο μνήμης του μοντέλου OpenCL και η συσχέτιση των διαφόρων περιοχών σε κάθε δομική μονάδα.

Μοντέλο Συνεκτικότητας Μνήμης (Memory Consistency)

Ένα σημαντικό χαρακτηριστικό των υποσυστημάτων μνήμης σε παράλληλα συστήματα είναι το μοντέλο συνεκτικότητας της μνήμης. Το πρότυπο OpenCL χρησιμοποιεί ένα απλουστευμένο μοντέλο συνεκτικότητας της μνήμης (relaxed memory consistency). Αυτό το μοντέλο δεν εγγυάται ότι η κατάσταση της μνήμης που είναι ορατή σε ένα work-item είναι συνεπής μεταξύ όλων των work-items.

Εντός ενός work-item, η μνήμη ακολουθεί το μοντέλο συνεκτικότητας load / store (load / store consistency). Οι περιοχές τοπικής μνήμης είναι συνεπής ανάμεσα στα work-items ενός συγκεκριμένου work-group σε ένα σημείο συγχρονισμού (work-group barrier). Ομοίως και για τις περιοχές της καθολικής μνήμης. Όμως, το μοντέλο δεν παρέχει εγγυήσεις όσον αφορά στην συνεκτικότητα της μνήμης ανάμεσα σε διαφορετικά work-groups που ανήκουν στον ίδιο kernel.

2.5 Μοντέλο Προγραμματισμού

Το μοντέλο εκτέλεσης του προτύπου OpenCL υποστηρίζει τόσο μοντέλα προγραμματισμού data parallel και task parallel όσο και υβριδικές παραλλαγές αυτών. Το κύριο μοντέλο, στο οποίο βασίζεται το μοντέλο OpenCL, είναι το μοντέλο data parallel.

Data Parallel Μοντέλο Προγραμματισμού

Το data parallel μοντέλο προγραμματισμού ορίζει έναν υπολογισμό βάσει μίας ακολουθίας εντολών που εκτελούνται σε πολλαπλά στοιχεία ενός αντικειμένου μνήμης. Ο χώρος δεικτών, που συσχετίζεται με το μοντέλο εκτέλεσης της OpenCL, ορίζει τα work-items και τον τρόπο απεικόνι-

σης των δεδομένων σε αυτά. Σε ένα data parallel μοντέλο προγραμματισμού, υπάρχει μία άμεση απεικόνιση μεταξύ των work-items και των περιοχών της μνήμης που κάθε ένα έχει πρόσβαση. Το πρότυπο OpenCL υλοποιεί μία τροποποιημένη έκδοση αυτού του μοντέλου όπου η άμεση απεικόνιση δεν αποτελεί αναγκαία προϋπόθεση.

Το μοντέλο OpenCL παρέχει ένα ιεραρχικό data parallel μοντέλο προγραμματισμού. Ο καθορισμός της ιεραρχίας μπορεί να πραγματοποιηθεί είτε άμεσα είτε έμμεσα. Στο πρώτο μοντέλο, ο προγραμματιστής καθορίζει τον συνολικό αριθμό από work-items που θα εκτελεστούν παράλληλα και την διαμέριση αυτών στα work-groups. Σε αντίθεση με αυτό, στον έμμεσο τρόπο ο προγραμματιστής καθορίζει μόνο τον συνολικό αριθμό από work-items και η διαμέριση αυτών σε work-groups πραγματοποιείται από την υλοποίηση του μοντέλου OpenCL.

Task Parallel Μοντέλο Προγραμματισμού

Το task parallel μοντέλο προγραμματισμού ορίζει ένα μοντέλο στο οποίο ένα μοναδικό στιγμιότυπο ενός kernel εκτελείται ανεξάρτητα από τον χώρο δεικτών. Αυτό είναι λογικά ισοδύναμο με την εκτέλεση ενός kernel σε μία υπολογιστική μονάδα και με ένα work-group, το οποίο αποτελείται από ένα μοναδικό work-item. Χρησιμοποιώντας αυτό το μοντέλο, ο χρήστης μπορεί να εκφράσει τον παραλληλισμό της εφαρμογής χρησιμοποιώντας τους τύπους διανυσματικών δεδομένων που υποστηρίζονται από το μοντέλο, εισάγοντας πολλαπλά tasks προς εκτέλεση σε μία command queue αλλά και χρησιμοποιώντας εγγενείς kernels που έχουν αναπτυχθεί με ένα ορθογώνιο μοντέλο προς το μοντέλο OpenCL.

Συγχρονισμός

Το μοντέλο OpenCL προσφέρει δύο τομείς συγχρονισμού: συγχρονισμό σε επίπεδο work-items που ανήκουν σε ένα work-group και συγχρονισμό σε επίπεδο εντολών που εισάγονται προς εκτέλεση σε μία command queue. Ο συγχρονισμός μεταξύ των work-items που ανήκουν σε ένα work-group επιτυγχάνεται με χρήση μίας εντολής barrier(). Όλα τα work-items εντός του work-group πρέπει να εκτελέσουν αυτή την εντολή προκειμένου να συνεχίσουν την εκτέλεσή των εντολών που ακολουθούν αυτή την εντολή. Μία τέτοια εντολή θα πρέπει είτε να μην εκτελεστεί από κανένα work-item είτε να εκτελεστεί από όλα τα work-items εντός του work-group. Όσον αφορά στον συγχρονισμό μεταξύ των work-groups, το μοντέλο δεν προσφέρει κάποιον αντίστοιχο μηχανισμό με αποτέλεσμα τα work-groups να είναι ανεξάρτητα και να μπορούν να εκτελεστούν παράλληλα.

Όσον αφορά στα σημεία συγχρονισμού μεταξύ εντολών που εισάγονται προς εκτέλεση σε μία command queue, αυτά είναι τα ακόλουθα:

- Command-queue barriers: Αντίστοιχα με τις εντολές barrier() που προσφέρονται για εκτέλεση από τα work-items ενός work-group, το μοντέλο παρέχει εντολές barrier() που μπορούν να εισαχθούν σε μία command queue για τον συγχρονισμό των εντολών που υπάρχουν σε αυτή. Μία command queue barrier() εντολή διασφαλίζει ότι όλες οι εντολές που έχουν εισαχθεί στην αντίστοιχη command queue θα τερματίσουν την εκτέλεσή τους και θα ενημερώσουν τα όποια αποτελέσματα στην μνήμη πριν την εκτέλεση των εντολών που έπονται της εντολής barrier(). Αυτού του είδους η εντολή μπορεί να χρησιμοποιηθεί μόνο για συγχρονισμό των εντολών μίας συγκεκριμένης command queue.

- Εντολές αναμονής για κάποιο γεγονός: Όπως αναφέρθηκε στην ενότητα 2.3, όλες οι εντολές που προσφέρονται από το OpenCL API και χρησιμοποιούνται για την εισαγωγή κάποιας

εντολής σε μία command queue μπορούν να επιστρέψουν ένα γεγονός (event) το οποίο προσδιορίζει την εντολή και τα αντικείμενα μνήμης που αυτή διαχειρίζεται. Μία επόμενη εντολή μπορεί να χρησιμοποιήσει αυτό το γεγονός προκειμένου να επιτευχθεί ένα σχήμα αναμονής, όπου η δεύτερη εντολή θα αναμένει για ολοκλήρωση της πρώτης χρησιμοποιώντας το αντίστοιχο event. Με αυτόν τον τρόπο διασφαλίζεται ότι οι όποιες αλλαγές πραγματοποιηθούν στα αντικείμενα μνήμης από την πρώτη εντολή θα είναι ορατές στην δεύτερη εντολή.

2.6 Αντικείμενα Μνήμης

Τα αντικείμενα μνήμης (memory objects) κατηγοριοποιούνται σε δύο τύπους: αντικείμενα τύπου buffer και αντικείμενα τύπου image. Ένα αντικείμενο τύπου buffer αποθηκεύει τα δεδομένα ως μία μονοδιάστατη συλλογή ενώ ένα αντικείμενο τύπου image χρησιμοποιείται για την αποθήκευση δισδιάστατων ή τρισδιάστατων συλλογών δεδομένων. Οι kernels λαμβάνουν ως είσοδο αντικείμενα μνήμης και αποθηκεύουν τα αποτελέσματα εξόδου σε ένα ή περισσότερα αντικείμενα μνήμης.

Τα στοιχεία ενός αντικειμένου τύπου buffer μπορεί να είναι βαθμωτές μεταβλητές, όπως int ή float, διανυσματικές μεταβλητές ή τύποι δεδομένων που έχουν οριστεί από τον χρήστη. Ένα αντικείμενο τύπου image χρησιμοποιείται για να αναπαραστήσει έναν προσωρινό χώρο αποθήκευσης που μπορεί να χρησιμοποιηθεί είτε ως χώρος που αποθηκεύεται κάποιο texture είτε ως ένας framebuffer. Ο ελάχιστος αριθμός στοιχείων που μπορούν να αποθηκευθούν σε ένα αντικείμενο μνήμης είναι ίσος με ένα.

Όσον αφορά στην πρόσβαση στα αποθηκευμένα δεδομένα, η πρόσβαση σε ένα αντικείμενο τύπου buffer πραγματοποιείται με χρήση ενός δείκτη (pointer) στα δεδομένα από τον αντίστοιχο kernel. Αυτό όμως δεν ισχύει για ένα αντικείμενο τύπου image και η πρόσβαση επιτυγχάνεται με την χρήση συναρτήσεων που προσφέρει το μοντέλο. Επίσης, ο τρόπος αποθήκευσης των δεδομένων σε ένα αντικείμενο τύπου buffer είναι ίδιος με τον τρόπο που αυτά προσπελαύνονται από τον αντίστοιχο kernel, ενώ στην περίπτωση ενός αντικειμένου τύπου image ο τρόπος πρόσβασης στα δεδομένα μπορεί να μην είναι ίδιος με το πρότυπο προσπέλασης των δεδομένων από τον kernel. Γι' αυτόν τον σκοπό χρησιμοποιούνται ειδικές συναρτήσεις που προσφέρει το OpenCL API.

Κεφάλαιο 3

Περιγραφή Υποδομής στο Επίπεδο του Μεταγλωττιστή

3.1 Εισαγωγή

Στην συνήθη μορφή της, μία συνάρτηση που αντιστοιχίζεται σε έναν kernel στο μοντέλο προγραμματισμού OpenCL, περιγράφει το τμήμα του υπολογισμού που θα εκτελεστεί από κάθε ένα work-item. Όπως αναφέρθηκε και στην ενότητα 2.3, κάθε work-item εντός του χώρου δεικτών εκτελεί ένα στιγμιότυπο του εκάστοτε kernel. Επομένως, μπορούμε να παραλληλίσουμε ένα work-item ως ένα λογικό νήμα (logical thread), το οποίο είναι υπεύθυνο για την εκτέλεση του αντίστοιχου στιγμιοτύπου.

Ένας kernel στο μοντέλο προγραμματισμού OpenCL εκφράζει τον παραλληλισμό της εφαρμογής στην λεπτότερη δυνατή διαμέριση (finest granularity). Παρ' όλα αυτά, η αποδοτική αξιοποίηση του διαθέσιμου παραλληλισμού και η απεικόνιση αυτού στις διαθέσιμες υπολογιστικές μονάδες της αρχιτεκτονικής δεν είναι άμεσα εμφανείς. Μία άμεση προσέγγιση είναι αυτή της απεικόνισης κάθε εκτέλεσης ενός στιγμιοτύπου του kernel είτε σε ένα νήμα επιπέδου χρήστη (user-level thread) είτε σε ένα νήμα επιπέδου πυρήνα (kernel-level thread). Για αρχιτεκτονικές που προσφέρουν υποστήριξη στο υλικό για μεγάλο αριθμό νημάτων (fine-grained threading) αυτή η προσέγγιση είναι εφικτή. Όμως, σε άλλες αρχιτεκτονικές, όπως οι κοινοί επεξεργαστές πολλαπλών πυρήνων ή ο επεξεργαστής Cell B.E., αυτού του είδους η απεικόνιση δεν αποτελεί την βέλτιστη λύση. Μία τέτοιου είδους προσέγγιση θα δημιουργούσε μεγάλη επιβάρυνση στον χρόνο εκτέλεσης της εφαρμογής, η οποία θα προερχόταν τόσο από την διαχείριση των πολλαπλών νημάτων που θα έπρεπε να δημιουργηθούν όσο και από την διαχείριση και την εκτέλεση των διαθέσιμων τμημάτων εργασίας (work tasks). Ουσιαστικά, ένα work-task αντιστοιχεί σε ένα work-group του NDRange του εκάστοτε kernel. Επομένως, για να ελαχιστοποιηθεί η επιπλέον επιβάρυνση γι' αυτού του είδους τις αρχιτεκτονικές, απαιτείται μία λιγότερο λεπτή διαμέριση του διαθέσιμου παραλληλισμού (coarser-grained parallelism). Συν τοις άλλοις, η προαναφερθείσα, άμεση απεικόνιση του παραλληλισμού, σε πολλές περιπτώσεις, δεν εκμεταλλεύεται την χωρική και χρονική τοπικότητα που εμφανίζεται στα work-items που ανήκουν σε έναν work-group.

Προκειμένου να επιτευχθεί η αποδοτική εκτέλεση των kernels μίας OpenCL εφαρμογής, εφαρμόζουμε μία σειρά μετασχηματισμών στον πηγαίο κώδικα (source code) της κάθε kernel συνάρτησης μίας εφαρμογής OpenCL. Αυτοί οι μετασχηματισμοί στοχεύουν στην τροποποίηση της λεπτότητας της διαμέρισης ενός kernel, ώστε πλέον ένας kernel να αναπαριστά το τμήμα του υπολογισμού που πρέπει να εκτελεστεί για ένα work-group και όχι για ένα λογικό νήμα. Αυτή η

προσέγγιση μειώνει την επιβάρυνση που συνεπάγεται μία λεπτή διαμέριση, ενώ επιπροσθέτως μειώνει το μέγεθος της μνήμης που απαιτείται για την εκτέλεση των εφαρμογών, καθώς πολλαπλά λογικά νήματα μπορούν να χρησιμοποιούν τον ίδιο χώρο για την αποθήκευση των τοπικών μεταβλητών τους. Μετά την εφαρμογή αυτής της σειράς των μετασχηματισμών, η τροποποιημένη kernel συνάρτηση αναπαριστά το τμήμα του υπολογισμού που αντιστοιχεί σε κάθε work-group στον χώρο των δεικτών της εφαρμογής. Δεδομένου ότι τα work-groups μπορούν να εκτελεστούν παράλληλα, όπως αναφέρθηκε στην ενότητα 2.5, είναι δυνατή η απεικόνιση ενός work-group σε μία διαθέσιμη μονάδα εκτέλεσης της αρχιτεκτονικής στην οποία εκτελείται η εφαρμογή. Η διαδικασία απεικόνισης περιγράφεται με περισσότερες λεπτομέρειες στο Κεφάλαιο 4.

Η διαδικασία των μετασχηματισμών αποτελείται από τρία βασικά βήματα: σειριοποίηση των λογικών νημάτων, εξάλειψη των όποιων πράξεων συγχρονισμού υπάρχουν στον kernel και εύρεση των μεταβλητών για τις οποίες δεν μπορεί να χρησιμοποιηθεί η ίδια θέση μνήμης από όλα τα λογικά νήματα που ανήκουν στο work-group. Όλοι οι μετασχηματισμοί πραγματοποιούνται στο επίπεδο του πηγαίου κώδικα της κάθε kernel συνάρτησης (source-to-source transformations). Για τον σκοπό αυτό τροποποιήσαμε κατάλληλα την υποδομή του μεταγλωττιστή LLVM [18], και πιο συγκεκριμένα την υποδομή Clang [10] που αποτελεί το εμπρόσθιο τμημά (front-end) της υποδομής LLVM. Η υποδομή Clang λαμβάνει ως είσοδο προγράμματα που έχουν αναπτυχθεί με χρήση των γλωσσών προγραμματισμού C/C++ και Objective C/C++. Επομένως, ήταν απαραίτητη η επέκταση διαφόρων τμημάτων που χρησιμοποιούνται για την αναπαράσταση των διαφόρων δομών, προκειμένου να είναι δυνατή η υποστήριξη των στοιχείων που προσθέτει η σύνταξη του μοντέλου OpenCL. Ενδεικτικά, ένα από αυτά τα στοιχεία αποτελούν τα προσδιοριστικά για τις διάφορες περιοχές μνήμης (`__global`, `__local`, `__constant`, `__private`). Τέλος, θα πρέπει να αναφερθεί ότι όλοι οι μετασχηματισμοί εκτελούνται στο επίπεδο του αφηρημένου συντακτικού δένδρου (Abstact Syntax Tree (AST)) που προκύπτει μετά την λεξιλογική, συντακτική και σημασιολογική ανάλυση του εκάστοτε αρχείου πηγαίου κώδικα. Εδώ θα πρέπει να αναφερθεί ότι υποθέτουμε ότι η συνάρτηση ακολουθεί το πρότυπο ANSI/C όσον αφορά στην δήλωση των μεταβλητών. Πιο συγκεκριμένα, οι μεταβλητές δηλώνονται στην αρχή της συνάρτησης, πριν από την εκτέλεση οποιασδήποτε εντολής και εκτός block εντολών.

Εκτός από τους ανωτέρω μετασχηματισμούς, επιπρόσθετοι μετασχηματισμοί απαιτούνται προκειμένου να είναι δυνατή η ενσωμάτωση ενός εξωτερικού τμήματος λογισμικού (software module) για την διαχείριση μέσω λογισμικού των επιπέδων κρυφής μνήμης (software cache module) καθώς και για την υποστήριξη εκτέλεσης kernel συναρτήσεων που έχουν μεταβλητό αριθμό και τύπο ορισμάτων. Αυτοί οι μετασχηματισμοί συνδέονται περισσότερο με την υποστήριξη σε επίπεδο χρόνου εκτέλεσης που παρέχει η υποδομή και γι' αυτό τον λόγο περιγράφονται αναλυτικά στις ενότητες 4.3 και 4.4.

3.2 Σειριοποίηση Λογικών Νημάτων

Ο πρώτος μετασχηματισμός στην ακολουθία μετασχηματισμών που εφαρμόζονται είναι η σειριοποίηση των λογικών νημάτων. Αυτός ο μετασχηματισμός έχει ως στόχο την αύξηση του υπολογιστικού φόρτου που βρίσκεται σε μία kernel συνάρτηση μέσω της σειριοποίησης της εκτέλεσης των λογικών νημάτων. Η σειριοποίηση αυτή είναι δυνατή καθώς τα work-items (logical threads) μπορούν να εκτελεστούν παράλληλα, σε περίπτωση που δεν υπάρχει κάποια εντολή συγχρονισμού. Επομένως, μπορούμε να καθορίσουμε μία αυθαίρετη ακολουθία εκτέλεσης αυτών, χωρίς να επηρεάζεται η ορθότητα εκτέλεσης της εφαρμογής. Το σχήμα 3.1 παρουσιάζει τον kernel της εφαρμογής BlackScholes πριν (3.1a) και μετά (3.1b) την εφαρμογή της σειριοποίησης των λογικών

```
__kernel void BlackScholes(
    __global float *d_Call, //Call option price
    __global float *d_Put,  //Put option price
    __global float *d_S,    //Current stock price
    __global float *d_X,    //Option strike price
    __global float *d_T,    //Option years
    float R,                //Riskless rate of return
    float V,                //Stock volatility
    unsigned int optN
){
    unsigned int opt;
    for(opt = get_global_id(0); opt < optN; opt += get_global_size(0))
        BlackScholesBody(
            &d_Call[opt],
            &d_Put[opt],
            d_S[opt],
            d_X[opt],
            d_T[opt],
            R,
            V
        );
}
```

(a)

```
__kernel void BlackScholes (
    __global float *d_Call, //Call option price
    __global float *d_Put,  //Put option price
    __global float *d_S,    //Current stock price
    __global float *d_X,    //Option strike price
    __global float *d_T,    //Option years
    float R,                //Riskless rate of return
    float V,                //Stock volatility
    unsigned int optN)
{
    int ___kernel_indices[3];
    unsigned int opt;

    ___kernel_indices[2] = 0;
    while (___kernel_indices[2] < get_local_size (2)) {
        ___kernel_indices[1] = 0;
        while (___kernel_indices[1] < get_local_size (1)) {
            ___kernel_indices[0] = 0;
            while (___kernel_indices[0] < get_local_size (0)) {

                for (opt = get_global_id (0); opt < optN; opt += get_global_size (0) )
                    BlackScholesBody (&d_Call[opt], &d_Put[opt], d_S[opt],
                                      d_X[opt], d_T[opt], R, V);

                ___kernel_indices[0]++;
            }
            ___kernel_indices[1]++;
        }
        ___kernel_indices[2]++;
    }
}
```

(b)

Σχήμα 3.1: Σειριοποίηση των λογικών νημάτων εντός μίας kernel συνάρτησης. Η kernel συνάρτηση πριν (a) και μετά (b) τον μετασχηματισμό.

νημάτων. Η σειριοποίηση επιτυγχάνεται με την τοποθέτηση των εντολών του σώματος του kernel εντός τριών εμφωλευμένων βρόγχων. Καθώς ο μέγιστος αριθμός διαστάσεων που μπορεί να έχει ένα work-group είναι ίσος με τρεις, οι τρεις βρόγχοι είναι ο αναγκαίος αριθμός για την εκτέλεση των λογικών νημάτων σε κάθε διάσταση του work-group. Κάθε βρόγχος διατρέχει τα λογικά νήματα στην αντίστοιχη διάσταση του work-group, οπότε επιτυγχάνεται η σειριακή εκτέλεση των λογικών νημάτων.

Όσον αφορά στην συνάρτηση get_local_size(N), αυτή χρησιμοποιείται για την λήψη του μεγέθους της N-οστής διάστασης του work-group. Η παράμετρος N μπορεί να λαμβάνει τιμές από 0 έως 2. Όπως παρατηρούμε στην εικόνα 3.1, η εκτέλεση των λογικών νημάτων ακολουθεί μία διάταξη που δεν είναι διαισθητικά ορθή. Η διαισθητικά ορθή διάταξη θα ήταν η 0, 1, 2 και όχι η διάταξη 2, 1, 0. Όμως, η τελευταία επιτρέπει την πιο αποδοτική εκτέλεση της συνάρτησης. Το μοντέλο OpenCL δίνει την δυνατότητα στον χρήστη να καθορίζει το μέγεθος των διαστάσεων του κάθε work-group κατά την εισαγωγή της αντίστοιχης εντολής για εκτέλεση του kernel στην command queue. Σε περίπτωση που το μέγεθος μίας διάστασης δεν καθοριστεί, το σύστημα υποστήριξης χρόνου εκτέλεσης θέτει το μέγεθος της αντίστοιχης διάστασης ίση με 1. Αυτό, σε συνδυασμό με την διάταξη διαπέρασης των διαστάσεων, οδηγεί στην εκτέλεση του ελάχιστου πλήθους εντολών για την ορθή διαπέραση των διαστάσεων. Σε περίπτωση που ακολουθούνταν η διάταξη 0, 1, 2, τότε σε κάθε εκτέλεση των εμφωλευμένων βρόγχων, εκτός από την εκτέλεση των αρχικών εντολών της συνάρτησης, θα έπρεπε να εκτελεστούν και οι εντολές που εισάγουν οι

βρόγχοι με πλήθος επαναλήψεων (trip count) ίσο με 1[1]. Η εκτέλεση αυτών των εντολών γίνεται μόνο μία φορά σε περίπτωση που χρησιμοποιείται η διάταξη 2, 1, 0. Επομένως, με χρήση της τελευταίας διάταξης είναι δυνατή η ελαχιστοποίηση της επιβάρυνσης που εισάγει η διάταξη των εμφωλευμένων βρόγχων, με αποτέλεσμα ο μετασχηματισμός αυτός να μην εισάγει επιπλέον επιβάρυνση στην εκτέλεση της εφαρμογής.

Επίσης, όπως παρατηρούμε ο μετασχηματισμός της σειριοποίησης των λογικών νημάτων έχει ως αποτέλεσμα την μείωση του συνολικού μεγέθους της μνήμης που απαιτείται για την εκτέλεση του συνόλου των κλήσεων μίας kernel συνάρτησης, καθώς μόνο ένα λογικό νήμα είναι ενεργό σε κάθε χρονική στιγμή και η μνήμη που χρησιμοποιείται για την αποθήκευση των τοπικών μεταβλητών μπορεί να επαναχρησιμοποιηθεί. Επί παραδείγματι, ο χώρος μνήμης που χρησιμοποιείται για την αποθήκευση της τοπικής μεταβλητής opt είναι κοινός για όλα τα νήματα του work-group.

Τέλος, η επιλογή του work-group ως του κατάλληλου βαθμού σειριοποίησης των λογικών νημάτων μπορεί εκ πρώτης όψεως να ομοιάζει με τυχαία. Όμως, στην ακόλουθη ενότητα θα γίνει αντιληπτό ότι άλλοι βαθμοί σειριοποίησης εισάγουν προβλήματα που συχνά είναι δυσεπίλυτα, ιδιαίτερα στην περίπτωση που υπάρχουν εντολές συγχρονισμού ή πολλαπλά σημεία εξόδου από έναν kernel. Συν τοις άλλοις, το προγραμματιστικό μοντέλο OpenCL δίνει την δυνατότητα στον προγραμματιστή να καθορίσει το μέγεθος που θα έχουν τα work-groups κατά την εκτέλεση ενός kernel. Συνήθως, ο προγραμματιστής λαμβάνει υπόψη του την επαναχρησιμοποίηση των δεδομένων ή ακόμη προσαρμόζει το μέγεθος του footprint του work-group για συγκεκριμένα επίπεδα της ιεραρχίας μνήμης. Έτσι, παρόλο που σε συγκεκριμένες εφαρμογές η χρήση διαφορετικού βαθμού σειριοποίησης μπορεί να είναι εφικτή και να διασφαλίζει την ορθότητα της εκτέλεσης του kernel, αυτό ενδέχεται να έχει αρνητική επίδραση στην απόδοση της εφαρμογής. Επίσης, η χρήση μεγαλύτερου βαθμού σειριοποίησης θα οδηγούσε σε άνιση κατανομή του φόρτου εργασίας για μη κανονικές εφαρμογές. Χαρακτηριστικά παραδείγματα τέτοιων εφαρμογών αποτελούν αρκετές από τις εφαρμογές αριθμητικής ανάλυσης, οι οποίες εμφανίζουν διαφορετικές απαιτήσεις σε υπολογιστική ισχύ για τα διαφορετικά τμήματα εργασίας.

3.3 Εξάλειψη των Πράξεων Συγχρονισμού

Ο επόμενος μετασχηματισμός στοχεύει στην επίλυση των προβλημάτων που εισάγονται από τις εντολές συγχρονισμού ή τα πολλαπλά σημεία εξόδου (exit points) που ενδέχεται να υπάρχουν σε κάποια kernel συνάρτηση. Όπως αναφέρθηκε στην ενότητα 2.5, ο συγχρονισμός μεταξύ των work-items που ανήκουν σε ένα work-group επιτυγχάνεται με χρήση της συνάρτησης barrier(). Όλα τα work-items θα πρέπει να εκτελέσουν αυτή την εντολή προκειμένου να συνεχίσουν την εκτέλεση των επόμενων εντολών. Ομοίως, σε περίπτωση που μία τέτοια εντολή βρίσκεται εντός ενός βρόγχου, όλα τα work-items θα πρέπει να την εκτελέσουν πριν συνεχίσουν με την εκτέλεση της επόμενης επανάληψης του βρόγχου. Τέλος, σε περίπτωση που μία εντολή barrier βρίσκεται εντός του block εντολών μίας εντολής συνθήκης, ο προγραμματιστής είναι υπεύθυνος προκειμένου να διασφαλίσει ότι η ροή ελέγχου του προγράμματος λόγω αυτής της συνθήκης θα είναι ίδια για όλα τα work-items. Αυτό έχει ως αποτέλεσμα η εντολή barrier είτε να μην εκτελεστεί είτε να εκτελεστεί από όλα τα work-items. Σε διαφορετική περίπτωση, υπάρχει το ενδεχόμενο η εντολή να εκτελεστεί από ένα τμήμα των work-items, τα οποία θα αναμένουν για την εκτέλεση της εντολής από τα υπόλοιπα work-items. Τα υπόλοιπα work-items δεν θα εκτελέσουν την εντολή με αποτέλεσμα την εμφάνιση deadlock.

[1] Οι βρόγχοι αυτοί αντιστοιχούν στις διαστάσεις του work-group με μέγεθος που δεν έχει οριστεί από τον προγραμματιστή.

```
__kernel void transpose(__global float *odata, __global float *idata,
                        int width, int height, __local float* block)
{
        // read the matrix tile into shared memory
        unsigned int xIndex = get_global_id(0);
        unsigned int yIndex = get_global_id(1);
        unsigned int index_in, index_out;
        if((xIndex < width) && (yIndex < height))
        {
            index_in = yIndex * width + xIndex;
            block[get_local_id(1)*(BLOCK_DIM+1)+get_local_id(0)] =
            idata[index_in];
        }

        barrier(CLK_LOCAL_MEM_FENCE);

        // write the transposed matrix tile to global memory
        xIndex = get_group_id(1) * BLOCK_DIM + get_local_id(0);
        yIndex = get_group_id(0) * BLOCK_DIM + get_local_id(1);
        if((xIndex < height) && (yIndex < width))
        {
            index_out = yIndex * height + xIndex;
            odata[index_out] =
            block[get_local_id(0)*(BLOCK_DIM+1)+get_local_id(1)];
        }
}
```

(a)

```
__kernel void transpose (__global float *odata, __global float *idata,
                         int width, int height, __local float *block)
{
    int ___kernel_indices[3];
    unsigned int xIndex, yIndex, index_in, index_out;

    triple_nested_loop {
        xIndex = (get_global_id (0) + ___kernel_indices[0]);
        yIndex = (get_global_id (1) + ___kernel_indices[1]);
        if ((xIndex < width) && (yIndex < height)) {
            index_in = yIndex * width + xIndex;
            block[get_local_id(1)*(BLOCK_DIM+1)+get_local_id(0)] =
            idata[index_in];
        }
    }
    //barrier (CLK_LOCAL_MEM_FENCE);
    triple_nested_loop {
        xIndex = get_group_id (1) * BLOCK_DIM + get_local_id (0);
        yIndex = get_group_id (0) * BLOCK_DIM + get_local_id (1);

        if ((xIndex < height) && (yIndex < width)) {

            index_out = yIndex * height + xIndex;
            odata[index_out] =
            block[get_local_id(0)*(BLOCK_DIM+1)+get_local_id(1)];
        }
    }
}
```

(b)

Σχήμα 3.2: Η εφαρμογής της τεχνικής του loop fission για μία εντολή συγχρονισμού. Το σχήμα παρουσιάζει την kernel συνάρτηση της εφαρμογής Matrix Transpose πριν (a) και μετά (b) την εφαρμογή του μετασχηματισμού.

Μία kernel συνάρτηση που έχει σειριοποιηθεί στο επίπεδο του work-group εισάγει έμμεσο συγχρονισμό των λογικών νημάτων πριν την πρώτη και μετά την τελευταία επανάληψη των βρόγχων που έχουν εισαχθεί από το πρώτο στάδιο της διαδικασίας των μετασχηματισμών. Παρόμοια, μία εντολή barrier απαιτεί τα λογικά νήματα να συγχρονίζονται πριν την διέλευση από αυτή. Παρατηρούμε λοιπόν μία αναλογία ανάμεσα στην δομή των τριπλά εμφωλευμένων βρόγχων και της εντολής barrier() όσον αφορά στον συγχρονισμό των λογικών νημάτων.

Όπως μπορούμε εύκολα να παρατηρήσουμε, σε περίπτωση που υπάρχει μία εντολή barrier() εντός μίας kernel συνάρτησης, ο προηγούμενος μετασχηματισμός της σειριοποίησης των λογικών νημάτων έχει ως αποτέλεσμα την λανθασμένη εκτέλεση της συνάρτησης. Μετά τον μετασχηματισμό, κάθε λογικό νήμα θα εκτελέσει όλες τις εντολές στο σώμα της συνάρτησης πριν συνεχίσει η εκτέλεση των υπολοίπων λογικών νημάτων. Αυτή η ακολουθία εκτέλεσης είναι λανθασμένη βάσει της σημασιολογίας της εντολής barrier().

Για να διασφαλιστεί η ορθή εκτέλεση του τροποποιημένου kernel, εφαρμόζουμε την τεχνική του loop fission για κάθε εντολή συγχρονισμού. Με αυτή την τεχνική διαχωρίζουμε τις εντολές σε blocks ώστε κάθε block να μην περιέχει καμία εντολή συγχρονισμού. Αυτός ο μετασχηματισμός για μία συνάρτηση που περιέχει μία πράξη συγχρονισμού παρουσιάζεται στο σχήμα 3.2. Η δομή `triple_nested_loop` αναπαριστά, εν συντομία, τους τριπλά εμφωλευμένους βρόγχους που έχουν εισαχθεί στον μετασχηματισμό της σειριοποίησης των λογικών νημάτων.

Όπως παρατηρούμε, η συνάρτηση περιέχει μόνο μία εντολή συγχρονισμού οπότε απαιτούνται δύο δομές βρόγχων για να διασφαλιστεί η ορθή εκτέλεση των work-items εντός των work-groups. Σε αυτή την νέα ακολουθία βρόγχων, τα λογικά νήματα συγχρονίζονται έμμεσα κάθε φορά που οι συνθήκες των βρόγχων αποτιμώνται. Αυτός είναι ένας έγκυρος μετασχηματισμός καθώς το προγραμματιστικό μοντέλο OpenCL απαιτεί κάθε ροή ελέγχου που επηρεάζει ένα σημείο συγχρονισμού να μην εξαρτάται από την εκτέλεση του εκάστοτε λογικού νήματος.

Ο αλγόριθμος διατρέχει το αφηρημένο συντακτικό δένδρο και ελέγχει την εμβέλεια που περιέχει την εκάστοτε εντολή συγχρονισμού. Σε περίπτωση που η εμβέλεια δεν είναι μία δομή βρόγχου τότε ο αλγόριθμος διαμερίζει τις εντολές πριν και μετά την εντολή σε δύο blocks οπότε σχηματίζονται δύο εμφωλευμένες δομές βρόγχων. Σε διαφορετική περίπτωση, ο αλγόριθμος εφαρμόζει την τεχνική του loop fission στις εντολές που υπάρχουν στην δομή `triple_nested_loop`.

Παρόμοια επιλύεται και το πρόβλημα που εμφανίζεται σε block εντολών με περισσότερα του ενός σημεία εξόδου (multiple exit points). Τα πολλαπλά σημεία εξόδου εμφανίζονται από εντολές όπως continue, break και return, οι οποίες μεταβάλλουν την ροή ελέγχου της εφαρμογής. Σε περίπτωση που μία τέτοια εντολή βρίσκεται σε μία δομή `triple_nested_loop`, η εκτέλεση του kernel δεν θα ακολουθεί την προηγούμενη σημασιολογία, με αποτέλεσμα η εκτέλεση της εφαρμογής να μην είναι ορθή. Αυτό συμβαίνει καθώς μόνο το πρώτο από τα λογικά νήματα θα έχει την δυνατότητα να εκτελέσει τις εντολές πριν από την συγκεκριμένη εντολή. Για να επιλύσουμε αυτό το πρόβλημα, ο αλγόριθμος θεωρεί και αυτές τις εντολές ως επιπλέον εντολές συγχρονισμού. Έτσι, περικλείει τις εντολές πριν και μετά από αυτές σε επιπλέον εμφωλευμένες δομές βρόγχου.

Από τα παραπάνω, ενισχύεται η επιλογή του βαθμού του work-group ως τον κατάλληλο για την σειριοποίηση των λογικών νημάτων καθώς γίνεται κατανοητό ότι η επιλογή μικρότερου βαθμού για την σειριοποίηση των λογικών νημάτων από αυτή του work-group θα εισήγαγε επιπλέον περιπλοκές στην υλοποίηση της υποδομής. Η επιλογή μικρότερου βαθμού σειριοποίησης θα καθιστούσε δύσκολη την αντιστοίχιση μεταξύ των τμημάτων εργασίας (work-tasks) στα διαθέσιμα πλαίσια εκτέλεσης της αρχιτεκτονικής και θα δημιουργούσε επιπλέον επιβάρυνση στο σύστημα χρόνου εκτέλεσης. Πλέον, κάθε τμήμα εργασίας θα έπρεπε να μπλοκάρει την εκτέλεσή του μέχρι να εκτελεστούν και τα υπόλοιπα λογικά νήματα του αντίστοιχου work-group. Συνεπώς, θα απαιτούνταν αλλαγή στο πλαίσιο εκτέλεσης (context switch) για την εκτέλεση είτε των νημάτων σε επίπεδο χρήστη είτε των νημάτων σε επίπεδο συστήματος στα οποία θα είχαν ανατεθεί τα υπόλοιπα τμήματα εργασίας. Αυτή η εναλλαγή θα επιβάρυνε περαιτέρω το σύστημα χρόνου εκτέλεσης, το οποίο θα ήταν επιφορτισμένο με την διαχείριση αυτής της διαδικασίας.

Η τεχνική του loop fission εφαρμόζεται επαναληπτικά και απαιτεί πολλαπλές διαπεράσεις του αφηρημένου συντακτικού δένδρου (AST). Η εφαρμογή του μετασχηματισμού για μία πράξη συγχρονισμού ή για μία πράξη που επηρεάζει την ροή ελέγχου του κώδικα της συνάρτησης ενδέχεται να εμφανίσει επιπλέον εντολές που πρέπει να επεξεργαστούν ως εντολές συγχρονισμού, με αποτέλεσμα να απαιτούνται επιπλέον διαπεράσεις του δένδρου.

3.4 Ιδιωτικοποίηση Μεταβλητών (Variable Privatization)

Μετά την εφαρμογή της τεχνικής του loop fission για τις εντολές συγχρονισμού και ροής ελέγχου, ο μεταγλωττιστής θα πρέπει να επιλύσει το πρόβλημα που εμφανίζουν οι μεταβλητές των οποίων η διάρκεια ζωής εκτείνεται πέραν ενός σημείου συγχρονισμού.

Κάθε λογικό νήμα στην αρχική kernel συνάρτηση έχει την δική του ιδιωτική μνήμη για την αποθήκευση των τοπικών μεταβλητών. Όμως, μετά την εφαρμογή του μετασχηματισμού σει-

ριοποίησης, τα λογικά νήματα που ανήκουν σε ένα work-group διαμοιράζονται την μνήμη που χρησιμοποιείται για την αποθήκευση των τοπικών μεταβλητών. Στην συνήθη περίπτωση αυτό είναι τόσο έγκυρο όσο και επιθυμητό. Καθώς κάθε λογικό νήμα έχει τελειώσει την εκτέλεσή του πριν να αρχίσει να εκτελείται το επόμενο νήμα, δεν χρειάζεται τις τιμές των μεταβλητών και αυτός ο χώρος μνήμης μπορεί να επαναχρησιμοποιηθεί. Όμως αυτό δεν ισχύει για τον χώρο μνήμης όπου αποθηκεύονται οι τιμές των μεταβλητών με διάρκεια ζωής που εκτείνεται πέραν ενός σημείου συγχρονισμού, δηλαδή για μεταβλητές στις οποίες ανατίθεται μία τιμή πριν και χρησιμοποιείται μετά από ένα σημείο συγχρονισμού. Αυτό συμβαίνει καθώς η τιμή που ανατίθεται σε αυτή την μεταβλητή από ένα λογικό νήμα στην πρώτη δομή εμφωλευμένων βρόγχων (`triple_nested_loop`), η οποία βρίσκεται πριν το σημείο συγχρονισμού, δεν μπορεί να χρησιμοποιηθεί στην δομή εμφωλευμένων βρόγχων που βρίσκεται μετά το σημείο συγχρονισμού καθώς τα περιεχόμενά της μεταβλητής έχουν αλλαχθεί μετά την εκτέλεση των επόμενων λογικών νημάτων στην πρώτη δομή. Αυτό συνεπάγεται την μη ορθή εκτέλεση του κώδικα της εφαρμογής.

Η υποδομή μεταγλώττισης είναι υπεύθυνη για την επίλυση του προβλήματος. Γι' αυτό τον σκοπό, εκτελεί μία ανάλυση χρησιμοποιούμενων μεταβλητών (live variable analysis) ώστε να αναγνωρισθούν οι μεταβλητές που έχουν διάρκεια ζωής η οποία εκτείνεται πέραν των ορίων μίας δομής εμφωλευμένων βρόγχων. Έπειτα, εφαρμόζουμε την τεχνική της ιδιωτικοποίησης μεταβλητών (variable privatization) [1] γι' αυτές τις μεταβλητές. Η τεχνική της ιδιωτικοποίησης μεταβλητών εφαρμόζεται από μεταγλωττιστές βελτιστοποίησης προκειμένου να επιλυθούν εξαρτήσεις μεταξύ διαδοχικών επαναλήψεων διαφόρων βρόγχων και να είναι δυνατή η παραλληλοποίηση αυτών. Η εφαρμογή της τεχνικής αυτής έχει ως αποτέλεσμα την αποθήκευση των τοπικών μεταβλητών, με διάρκεια ζωής μεγαλύτερη από την εκτέλεση μίας δομής εμφωλευμένων βρόγχων, σε χώρο μνήμης που είναι ιδιωτικός για κάθε λογικό νήμα. Ουσιαστικά, αντί μίας βαθμωτής μεταβλητής, πλέον έχουμε έναν πίνακα από μεταβλητές, όπου κάθε θέση του πίνακα αποθηκεύει την τιμή της μεταβλητής για το αντίστοιχο λογικό νήμα, ώστε να προσομοιάζεται πλήρως η ιδιωτική μνήμη που διαχειρίζονταν τα λογικά νήματα πριν την σειριοποίηση. Ως τελευταίο βήμα, κάθε αναφορά σε αυτές τις βαθμωτές μεταβλητές θα πρέπει να αντικατασταθεί από την αναφορά στην κατάλληλη θέση του αντίστοιχου πίνακα. Όπως μπορούμε να συμπεράνουμε, ο μετασχηματισμός αυτός αυξάνει τις απαιτήσεις της μνήμης κατά την διάρκεια εκτέλεσης της εφαρμογής. Όμως αυτό είναι απαραίτητο για την διασφάλιση της ορθής εκτέλεσης του κώδικα της kernel συνάρτησης.

Το σχήμα 3.3 παρουσιάζει τον τελευταίο στην ακολουθία των μετασχηματισμών που εφαρμόζονται από την υποδομή μεταγλώττισης. Το σχήμα 3.3a παρουσιάζει την kernel συνάρτηση της εφαρμογής Matrix Multiplication πριν την εφαρμογή της τεχνικής της ιδιωτικοποίησης μεταβλητών, αφού έχουν εκτελεσθεί όλοι οι υπόλοιποι μετασχηματισμοί, και το σχήμα 3.3b απεικονίζει την μορφή που λαμβάνει η συνάρτηση μετά την εφαρμογή του μετασχηματισμού. Όπως παρατηρούμε, σε αυτή την εφαρμογή, αρκετές μεταβλητές έχουν διάρκεια ζωής που εκτείνεται πέραν μίας δομής `triple_nested_loop`. Επί παραδείγματι, η μεταβλητή `Csub` ορίζεται στην τρίτη, κατά σειρά, δομή `triple_nested_loop` και χρησιμοποιείται στην τέταρτη δομή εμφωλευμένων βρόγχων. Επομένως, απαιτείται η δέσμευση επιπλέον χώρου μνήμης για την αποθήκευση των αντιγράφων της μεταβλητής για τα λογικά νήματα. Γι' αυτό τον λόγο, η δήλωση της βαθμωτής μεταβλητής αντικαθίσταται από την δήλωση ενός πίνακα. Επίσης, παρατηρούμε την αντικατάσταση των αναφορών σε αυτές τις μεταβλητές με αναφορά στην αντίστοιχη θέση του κατάλληλου πίνακα. Ο εκάστοτε πίνακας δεικτοδοτείται από έναν ακέραιο, thread_id, η τιμή του οποίου αποτελεί έναν συνδυασμό των αναγνωριστικών των λογικών νημάτων σε κάθε διάσταση του work-group. Σε αυτό το σημείο θα πρέπει να αναφερθεί ότι η ιδιωτικοποίηση των μεταβλητών εφαρμόζεται μόνο για όσες μεταβλητές αυτό κρίνεται απαραίτητο μετά την ανάλυση. Έτσι, δεν

```
__kernel void matrixMul( __global float *A, __global float *B, __global float *C,
                         __local float *As, __local float *Bs, int Awidth, int Bwidth ) {

        int a, b, c, aEnd, k;
        float Csub;

        triple_nested_loop {
                aEnd = Awidth * get_local_id(1) + get_local_id(0);
                a = Awidth * BLOCK_SIZE * get_group_id(1) + aEnd;
                b = BLOCK_SIZE * get_group_id(0);
                b = a + b;
                aEnd = a + Awidth;
                Csub = 0;
        }
        while( a < aEnd ) {
                triple_nested_loop {
                        As[ get_local_id(1) * BLOCK_SIZE + get_local_id(0) ] = A[a];
                        Bs[ get_local_id(1) * BLOCK_SIZE + get_local_id(0) ] = B[b];
                        a += BLOCK_SIZE;
                        b += BLOCK_SIZE * Bwidth;
                }
                triple_nested_loop {
                        for( k = 0; k < BLOCK_SIZE; k++ ) {
                                Csub += As[ get_local_id(1) * BLOCK_SIZE + k ] *
                                     Bs[k * BLOCK_SIZE + get_local_id(0) ];
                        }
                }
        }
        triple_nested_loop {
                C[c] = Csub;
        }
}
```

(a)

```
__kernel void matrixMul( __global float *A, __global float *B, __global float *C,
                         __local float *As, __local float *Bs, int Awidth, int Bwidth ) {

        int a[], b[], c[], aEnd[], k;
        float Csub[];

        triple_nested_loop {
                aEnd[thread_id] = Awidth * get_local_id(1) + get_local_id(0);
                a[thread_id] = ( Awidth * BLOCK_SIZE ) * get_group_id(1) + aEnd[thread_id];
                b[thread_id] = BLOCK_SIZE * get_group_id(0);
                b[thread_id] = a[thread_id] + b[thread_id];
                aEnd[thread_id] = a[thread_id] + Awidth;
                Csub[thread_id] = 0;
        }
        while( a[thread_id] < aEnd[thread_id] ) {
                triple_nested_loop {
                        As[ get_local_id(1) * BLOCK_SIZE + get_local_id(0) ] = A[a[thread_id]];
                        Bs[ get_local_id(1) * BLOCK_SIZE + get_local_id(0) ] = B[b[thread_id]];
                        a[thread_id] += BLOCK_SIZE;
                        b[thread_id] += BLOCK_SIZE * Bwidth;
                }
                triple_nested_loop {
                        for( k = 0; k < BLOCK_SIZE; k++ ) {
                                Csub[thread_id] += As[ get_local_id(1) * BLOCK_SIZE + k ] *
                                     Bs[ ( k * BLOCK_SIZE ) + get_local_id(0) ];
                        }
                }
        }
        triple_nested_loop {
                C[c[thread_id]] = Csub[thread_id];
        }
}
```

(b)

Σχήμα 3.3: Η εφαρμογής της τεχνικής της ιδιωτικοποίησης μεταβλητών. Το σχήμα παρουσιάζει την kernel συνάρτηση της εφαρμογής Matrix Multiplication πριν (a) και μετά (b) την εφαρμογή του μετασχηματισμού.

απαιτείται δέσμευση επιπλέον χώρου για την μεταβλητή k αφού αυτή έχει διάρκεια ζωής μόνο εντός της τέταρτης, κατά σειρά, δομής `triple_nested_loop`.

Τέλος, είναι χρήσιμο να παρατηρήσουμε ότι παρόλο που το μοντέλο OpenCL ορίζει ξεχωριστές περιοχές μνήμης (ενότητα 2.4) για την αποθήκευση των μεταβλητών, η υποδομή απεικονίζει όλες τις διαφορετικές περιοχές μνήμης στο ίδιο σύστημα διαμοιραζόμενης μνήμης για τις αρχιτεκτονικές όπου το υποσύστημα μνήμης ελέγχεται από το υλικό. Αυτό όμως δεν ισχύει και για αρχιτεκτονικές με κατανεμημένο υποσύστημα μνήμης, όπως είναι ο επεξεργαστής Cell B.E. Κάθε SPE του επεξεργαστή διαθέτει την δική του τοπική μνήμη (Local Store), οπότε τόσο οι τοπικές και οι ιδιωτικές περιοχές μνήμης όσο και οι τοπικές μεταβλητές των kernel συναρτήσεων αποθηκεύονται σε αυτό το τμήμα της ιεραρχίας μνήμης.

Κεφάλαιο 4

Αρχιτεκτονική Συστήματος Χρόνου Εκτέλεσης

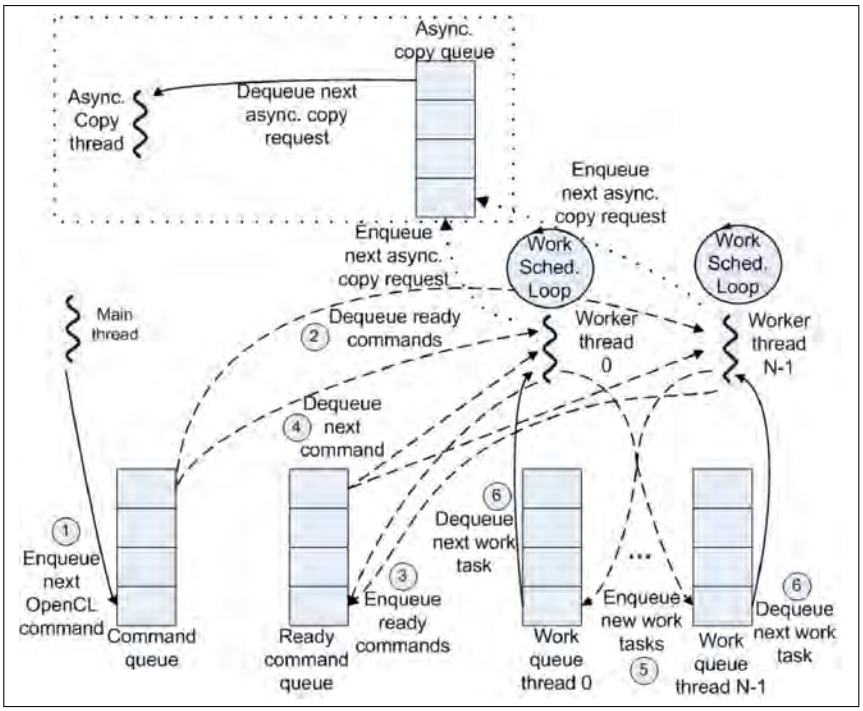
4.1 Εισαγωγή

Για να επιτευχθεί αποδοτική εκτέλεση μίας OpenCL εφαρμογής σε επεξεργαστές πολλαπλών πυρήνων, εκτός από την υποστήριξη σε επίπεδο μεταγλωττιστή, απαιτείται υποστήριξη και στο επίπεδο του χρόνου εκτέλεσης. Το σύστημα χρόνου εκτέλεσης (run-time system) είναι υπεύθυνο για την διαχείριση του υπολογιστικού φόρτου, δηλαδή των τμημάτων εργασίας όπως αυτά προκύπτουν από τις τροποποιημένες kernel συναρτήσεις, την διαχείριση των προσωρινών χώρων μνήμης και την παροχή υποστήριξης για την δυναμική εκτέλεση των kernel συναρτήσεων.

Το σύστημα χρόνου εκτέλεσης παρέχεται υπό την μορφή βιβλιοθήκης, η οποία διασυνδέεται με μία OpenCL εφαρμογή, και αναλαμβάνει να χειριστεί τις κλήσεις προς τις συναρτήσεις που προσφέρει το μοντέλο προγραμματισμού OpenCL μέσω του αντίστοιχου API.

Το σχήμα 4.1 παρουσιάζει την αρχιτεκτονική του συστήματος χρόνου εκτέλεσης της υποδομής GLOpenCL για αρχιτεκτονικές όπου το υποσύστημα μνήμης ελέγχεται από το υλικό (αρχιτεκτονική Intel x86), ενώ το σχήμα 4.2 απεικονίζει την αρχιτεκτονική για επεξεργαστές πολλαπλών πυρήνων με ιεραρχία μνήμης ελεγχόμενη από το λογισμικό, όπως ο επεξεργαστής Cell B.E. Οι διακεκομμένες γραμμές στα δύο σχήματα παρουσιάζουν τα τμήματα της αρχιτεκτονικής και τις διάφορες λειτουργίες που δεν είναι κοινές ανάμεσα στις δύο υλοποιήσεις. Όπως μπορούμε να παρατηρήσουμε, το μεγαλύτερο μέρος της αρχιτεκτονικής του συστήματος χρόνου εκτέλεσης είναι κοινό και για τις δύο αρχιτεκτονικές επεξεργαστών που υποστηρίζονται. Αυτή η προσέγγιση μίας ενοποιημένης αρχιτεκτονικής επιτρέπει την αποδοτική εκτέλεση εφαρμογών που χρησιμοποιούν το μοντέλο OpenCL στις δύο αρχιτεκτονικές, με ελάχιστες τροποποιήσεις στη αρχιτεκτονική του συστήματος χρόνου εκτέλεσης. Τέλος, η αρίθμηση στα δύο σχήματα υποδηλώνει μία τυπική ακολουθία ενεργειών για την εκτέλεση μίας εφαρμογής.

Το σύστημα χρόνου εκτέλεσης δημιουργεί έναν αριθμό από νήματα επιπέδου συστήματος (kernel-level threads). Και στις δύο υλοποιήσεις, η δημιουργία και η διαχείριση των νημάτων γίνεται με χρήση της βιβλιοθήκης νημάτων PThreads [20]. Αυτά τα νήματα είτε χρησιμοποιούνται για την εκτέλεση του υπολογιστικού τμήματος της εφαρμογής (worker threads) είτε επιτελούν βοηθητικές λειτουργίες (helper threads). Το κυρίως νήμα (main thread) της αρχιτεκτονικής είναι υπεύθυνο για την εκτέλεση του τμήματος της OpenCL εφαρμογής που εκτελείται στον host. Τα worker threads δημιουργούνται κατά την αρχικοποίηση του συστήματος χρόνου εκτέλεσης και είναι υπεύθυνα για την εκτέλεση του κυρίως υπολογιστικού τμήματος της εφαρμογής. Το



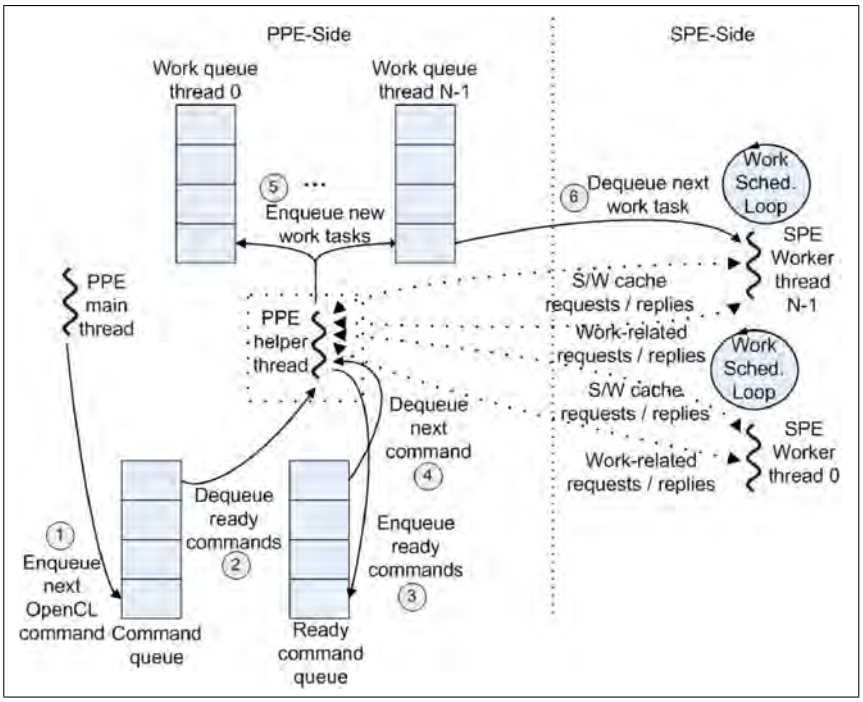
Σχήμα 4.1: Η αρχιτεκτονική του συστήματος χρόνου εκτέλεσης για την αρχιτεκτονική Intel.

σύστημα χρόνου εκτέλεσης της υποδομής GLOpenCL δημιουργεί έναν αριθμό νημάτων ίσο με τον αριθμό των πλαισίων εκτέλεσης - πυρήνων - που προσφέρει η εκάστοτε αρχιτεκτονική. Τέλος, το βοηθητικό νήμα (helper thread) αναλαμβάνει τις διάφορες λειτουργίες διαχείρισης. Οι ακριβείς λειτουργίες του βοηθητικού νήματος εξαρτώνται από την αρχιτεκτονική και θα περιγραφούν στις ακόλουθες ενότητες.

4.2 Διαχείριση Υπολογιστικού Φόρτου

Για κάθε εντολή που εκτελείται στην πλευρά του host και εισάγει μία εντολή σε μία ουρά εντολών ενός πλαισίου εκτέλεσης, το κυρίως νήμα δημιουργεί μία κατάλληλη δομή που περιγράφει την εντολή (command descriptor) και την εισάγει στην ουρά εντολών (command queue) του συστήματος χρόνου εκτέλεσης. Η ουρά εντολών του συστήματος χρόνου εκτέλεσης αποτελεί την υλοποίηση της ουράς εντολών του μοντέλου OpenCL. Σε αυτή την δομή έχουν πρόσβαση τόσο το κυρίως νήμα όσο και τα βοηθητικά νήματα. Επομένως, η πρόσβαση θα πρέπει να είναι αμοιβαίως αποκλειόμενη. Γι' αυτό τον λόγο χρησιμοποιούμε τον μηχανισμό των futexes [11]. Αυτός ο μηχανισμός επιτρέπει την υλοποίηση αμοιβαίου αποκλεισμού με ελάχιστη επιβάρυνση, στο επίπεδο του χρήστη και χωρίς να απαιτείται συνεχής πρόσβαση στο επίπεδο του συστήματος. Όταν το κυρίως νήμα εκτελέσει την τελευταία εντολή του τμήματος της εφαρμογής που εκτελείται στον host, αυτό μπλοκάρει μέχρι την εκτέλεση όλων των εντολών που έχουν εισαχθεί στην ουρά των εντολών.

Οι εντολές μεταφέρονται από την ουρά εντολών στην ουρά έτοιμων εντολών (ready command queue) του συστήματος χρόνου εκτέλεσης όταν αυτές είναι έτοιμες προς εκτέλεση. Αυτή η ουρά προσπελαύνεται μόνο από ένα νήμα κάθε φορά (helper thread) οπότε δεν χρησιμοποιείται κάποιος



Σχήμα 4.2: Η αρχιτεκτονική του συστήματος χρόνου εκτέλεσης για τον επεξεργαστή Cell B.E.

μηχανισμός για τον συγχρονισμό στην πρόσβαση σε αυτή. Μία ουρά εντολών ενός πλαισίου εκτέλεσης του μοντέλου OpenCL μπορεί να υποστηρίζει εκτέλεση των εντολών είτε εντός σειράς (in-order) είτε εκτός σειράς (out-of-order). Σε περίπτωση που η ουρά εντολών εκτελεί τις εντολές εντός σειράς, μία εντολή μεταφέρεται στην ουρά των έτοιμων διεργασιών όταν αυτή βρίσκεται στην κορυφή της ουράς, και δεδομένου ότι όλες οι προηγούμενες εντολές, που έχουν εισαχθεί στην ουρά πριν από αυτή, έχουν τελειώσει την εκτέλεσή τους.

Σε διαφορετική περίπτωση, όταν μία ουρά εντολών εκτελεί τις εντολές εκτός σειράς, χρησιμοποιείται μία δρομολόγηση των εντολών που βασίζεται στις εξαρτήσεις μεταξύ αυτών (dependence-driven scheduling). Σε αυτή την περίπτωση, ο προγραμματιστής χρησιμοποιεί τα γεγονότα (βλ. ενότητα 2.5) για τον συγχρονισμό μεταξύ των εντολών. Κατ' αυτόν τον τρόπο, μία εντολή εξαρτάται από μία ή περισσότερες προηγούμενες εντολές που έχουν εισαχθεί στην ουρά εντολών. Κάθε εντολή που τερματίζει την εκτέλεσή της ενημερώνει τις εξαρτήσεις των εντολών που εξαρτώνται από αυτή. Μόλις μία εντολή έχει ικανοποιήσει τις εξαρτήσεις της, εισάγεται στην ουρά των έτοιμων εντολών.

Τα τμήματα υπολογισμού (work-tasks) δημιουργούνται όταν μία εντολή που αφορά στην εκτέλεση ενός kernel εξάγεται από την ουρά των έτοιμων εντολών προς επεξεργασία. Κάθε τμήμα υπολογισμού αντιπροσωπεύει τον υπολογισμό που πρέπει να εκτελεστεί από κάθε work-group στον χώρο δεικτών της εφαρμογής. Ένα work-task αντιστοιχεί σε μία κλήση της εκάστοτε τροποποιημένης kernel συνάρτησης. Ο αριθμός των work-tasks που δημιουργείται εξαρτάται από τον τεμαχισμό του καθολικού χώρου δεικτών. Το μοντέλο OpenCL επιτρέπει στον προγραμματιστή της εφαρμογής να καθορίσει τον τεμαχισμό του χώρου δεικτών μέσω του καθορισμού των διαστάσεων των work-groups. Σε περίπτωση που αυτό δεν συμβαίνει, το σύστημα χρόνου εκτέλεσης εκτελεί έναν έμμεσο και στατικό τεμαχισμό του χώρου των δεικτών, λαμβάνοντας υπόψη των διαθέσιμο αριθμό των worker threads.

Προκειμένου να αποφευχθούν καταστάσεις "λιμοκτονίας", όπου τα worker threads αναμένουν διαρκώς για διαθέσιμα work-tasks, η εκτέλεση των τελευταίων έχουν μεγαλύτερη προτεραιότητα έναντι της εκτέλεσης των έτοιμων εντολών που βρίσκονται στην ready command queue. Αυτό σημαίνει ότι μία έτοιμη προς εκτέλεση εντολή λαμβάνεται προς επεξεργασία μόνο όταν δεν υπάρχουν άλλα έτοιμα work-tasks που αναμένουν για επεξεργασία. Στην υλοποίηση για την αρχιτεκτονική Intel x86 οι εντολές μεταφέρονται από την ουρά εντολών στην ουρά έτοιμων εντολών από ένα worker thread. Το πρώτο worker thread που διαπιστώνει ότι δεν υπάρχουν διαθέσιμα work-tasks αναλαμβάνει να επεξεργαστεί την πρώτη έτοιμη εντολή που είναι διαθέσιμη στην αντίστοιχη ουρά. Μία εντολή μπορεί να είναι είτε εντολή για εκτέλεση ενός kernel είτε εντολή για μεταφορά δεδομένων μεταξύ κάποιων αντικειμένων μνήμης. Μόνο οι εντολές που αφορούν σε εκτέλεση ενός kernel έχουν ως αποτέλεσμα την δημιουργία work-tasks. Σε περίπτωση που πραγματοποιείται επεξεργασία μίας τέτοιας εντολής, τα work-tasks που παράγονται διανέμονται στις work-queues. Όσον αφορά στην υλοποίηση για τον επεξεργαστή Cell, αυτή η ακολουθία ενεργειών εκτελείται από το helper thread που εκτελείται στην πλευρά του Power Processing Element (PPE).

Κάθε worker thread αναμένει διαρκώς για διαθέσιμα work-tasks. Μόλις υπάρχουν διαθέσιμα work-tasks στην ουρά, το αντίστοιχο worker thread λαμβάνει το work-task στη κορυφή της work-queue και εκτελεί την αντίστοιχη kernel συνάρτηση. Στην υλοποίηση για την αρχιτεκτονική x86, ένα worker thread έχει άμεση πρόσβαση στην αντίστοιχη work-queue οπότε μπορεί να εξαγάγει άμεσα το διαθέσιμο work-task. Όσον αφορά στην υλοποίηση για τον επεξεργαστή Cell, οι work-queues βρίσκονται στον χώρο μνήμης του PPE ενώ τα worker threads εκτελούνται στα SPEs οπότε δεν έχουν άμεση πρόσβαση σε αυτές. Για τον σκοπό αυτό η υποδομή υλοποιεί ένα σχήμα συγχρονισμού μεταξύ του helper thread και των worker threads χρησιμοποιώντας τον μηχανισμό των γραμματοκιβωτίων (mailboxes). Κατά την αρχικοποίηση του συστήματος χρόνου εκτέλεσης, το PPE δεσμεύει, για κάθε ένα worker thread, χώρο στην μνήμη όπου θα αποθηκεύεται ο work-descriptor για το work-task που του ανατίθεται. Όταν ένα worker thread έχει τελειώσει με την εκτέλεση του work-task που του έχει ανατεθεί, επικοινωνεί με το helper-thread ώστε να λάβει το επόμενο. Σε περίπτωση που υπάρχει ένα διαθέσιμο work-task, το helper thread αποθηκεύει τις απαραίτητες πληροφορίες στον αντίστοιχο work-descriptor και ενημερώνει το αντίστοιχο worker thread. Εν συνεχεία, το worker thread μεταφέρει, μέσω του μηχανισμού Direct Memory Access (DMA), τον work-descriptor και συνεχίζει την εκτέλεσή του. Σε περίπτωση που δεν υπάρχει διαθέσιμο κάποιο work-task, το helper-thread "μαρκάρει" το αντίστοιχο worker thread ως ανενεργό μέχρις ότου υπάρξει ένα διαθέσιμο work-task.

Το σύστημα χρόνου εκτέλεσης υλοποιεί ένα σχήμα πολλαπλών work-queues, όπου κάθε worker thread έχει αποκλειστική πρόσβαση στην αντίστοιχη, τοπική work-queue. Κάθε work-queue υλοποιείται ως μία Last-In, First-Out (LIFO) δομή που επιτρέπει όμως την εξαγωγή κάποιου στοιχείου τόσο από την κορυφή όσο και από την ουρά αυτής. Το σχήμα των πολλαπλών work-queues έχει πολλαπλά οφέλη για την απόδοση του συστήματος χρόνου εκτέλεσης. Εξαλείφει την επιβάρυνση που θα απαιτούνταν για τον συγχρονισμό όσον αφορά στις λειτουργίες από τα διαφορετικά νήματα σε περίπτωση που υπήρχε μία μοναδική work-queue, βελτιώνει την τοπικότητα και ενισχύει την καλύτερη κλιμάκωση του συστήματος χρόνου εκτέλεσης σε αρχιτεκτονικές με περισσότερους πυρήνες. Τα οφέλη από την τοπικότητα είναι ιδιαιτέρως εμφανή σε αρχιτεκτονικές με ελεγχόμενη από το λογισμικό ιεραρχία μνήμης, όπως ο επεξεργαστής Cell B.E. Τα διαθέσιμα work-tasks που δημιουργούνται κατανέμονται στα διαθέσιμα worker threads με ένα στατικό σχήμα ανάθεσης. Αυτή η ανάθεση τείνει να βελτιώσει την τοπικότητα καθώς γειτονικά, στον χώρο των δεικτών, work-groups συχνά προσπελαύνουν δεδομένα που βρίσκονται σε γειτονικές θέσεις μνήμης.

Ένα από τα πιθανά μειονεκτήματα που ενδέχεται να εμφανίσει το σχήμα με πολλαπλές work-

queues είναι η άνιση κατανομή του φόρτου εργασίας. Για να διασφαλίσουμε την ίση κατανομή των διαθέσιμων work-tasks, εφαρμόζουμε την τεχνική του work-stealing [7] ανάμεσα στις work-queues. Κάθε worker thread που δεν έχει διαθέσιμα work-tasks προς εκτέλεση στην δική του work-queue μπορεί να επιχειρήσει να εντοπίσει κάποιο διαθέσιμο work-task σε μία work-queue ενός άλλου worker thread. Σε περίπτωση που κάποια άλλη work-queue έχει διαθέσιμα work-tasks, το worker thread που επιχειρεί το work stealing λαμβάνει το work-task που βρίσκεται στην ουρά της δομής και συνεχίζει την εκτέλεση του.

Στην υλοποίηση της υποδομής για την αρχιτεκτονική Intel x86 τα worker threads εκτελούν την τεχνική του work-stealing άμεσα. Καθώς πολλαπλά νήματα έχουν πλέον την δυνατότητα πρόσβασης σε μία work-queue, κάθε δομή υλοποιεί ένα σχήμα συγχρονισμού χωρίς κλείδωμα (lock-free synchronization) [13] μεταξύ των νημάτων που προσπελαύνουν την εκάστοτε work-queue. Για την υλοποίηση αυτού του σχήματος συγχρονισμού χρησιμοποιούνται τα synchronization primitives που παρέχει η αρχιτεκτονική x86, όπως για παράδειγμα η εντολή `cmpxchgl`. Όσον αφορά στην υλοποίηση για την αρχιτεκτονική του επεξεργαστή Cell, τα νήματα που είναι ανενεργά, χωρίς διαθέσιμα work-tasks προς εκτέλεση, υποβοηθούνται από το helper thread για την υλοποίηση του work-stealing. Μόλις ένα worker thread ενημερώσει το helper thread για την λήψη του επόμενου διαθέσιμου work-task, το helper thread ελέγχει την αντίστοιχη work-queue. Σε περίπτωση που δεν υπάρχει κάποιο διαθέσιμο work-task, το helper thread ελέγχει τις work-queues και των υπολοίπων worker threads και σε περίπτωση που εντοπίσει κάποιο διαθέσιμο work-task, το αφαιρεί από την work-queue και ενημερώνει το αντίστοιχο worker thread.

4.3 Διαχείριση των Προσωρινών Χώρων Μνήμης (Memory Buffers)

Στο μοντέλο προγραμματισμού OpenCL, η επικοινωνία μεταξύ του host και των υπολογιστικών συσκευών επιτυγχάνεται μέσω των αντικειμένων μνήμης, και κυρίως μέσω των memory buffers. Επίσης, οι memory buffers χρησιμοποιούνται και για την επικοινωνία μεταξύ των work-items που ανήκουν σε ένα work-group. Αυτές οι περιοχές μνήμης μπορεί να είναι είτε καθολικές (global) είτε τοπικές (local). Οι καθολικές περιοχές μνήμης (global memory buffers) προσπελαύνονται από οποιοδήποτε work-item στον χώρο των δεικτών της εφαρμογής, ενώ οι τοπικές περιοχές μνήμης (local memory buffers) προσπελαύνονται μόνο από τα work-items που ανήκουν σε ένα work-group.

Οι καθολικές περιοχές μνήμης παρουσιάζουν μία ιδιαίτερη πρόκληση για την υλοποίηση του συστήματος χρόνου εκτέλεσης για αρχιτεκτονικές με ελεγχόμενη από το λογισμικό ιεραρχία μνήμης, όπως ο επεξεργαστής Cell B.E. Η χωρητικότητα των επιπέδων μνήμης που βρίσκονται πιο κοντά στον επεξεργαστή, όπως οι Local Stores των SPEs για την περίπτωση του επεξεργαστή Cell B.E., είναι περιορισμένη και στις περισσότερες των περιπτώσεων μη επαρκής για να αποθηκεύσει τα δεδομένα του συνόλου εργασίας του work-group. Το πρόβλημα είναι εντονότερο όταν δεν υπάρχει υποστήριξη από το υλικό για την υλοποίηση κάποιου πρωτοκόλλου συνοχής (coherency protocol) μεταξύ των κρυφών μνημών (caches) των διαφορετικών επεξεργαστικών συσκευών. Αυτή είναι και η περίπτωση του επεξεργαστή Cell B.E., όπου δεν υπάρχει κάποιο πρωτόκολλο για την υποστήριξη της συνοχής μεταξύ των Local Stores των διαφορετικών SPEs. Επίσης, όπως θα αναλυθεί και στο Κεφάλαιο 5, οι καθολικές περιοχές μνήμης εμπίπτουν περισσότερο στην έννοια της κοινής μνήμης (shared address space), με την οποία είναι εξοικειωμένοι οι περισσότεροι προγραμματιστές εφαρμογών. Έτσι, παρόλο που σε πολλές εφαρμογές η χρήση των τοπικών περιοχών μνήμης θα ήταν δυνατή, αυτό δεν γινόταν και αντί των τοπικών περιοχών μνήμης χρησιμοποιούνταν οι καθολικές περιοχές μνήμης.

Και τα δύο προβλήματα μπορούν να επιλυθούν με την χρήση ενός εξωτερικού τμήματος λο-

γισμικού (software module) για την διαχείριση μέσω λογισμικού των επιπέδων κρυφής μνήμης (software cache module). Αρκετές υλοποιήσεις μίας software cache έχουν προταθεί για τον επεξεργαστή Cell, όπως αυτές που προτείνουν οι συγγραφείς των εργασιών [4], [8], [9], [12] και [19]. Η υλοποίηση ενός τέτοιου τμήματος λογισμικού δεν αποτελεί μέρος της παρούσας εργασίας. Έτσι χρησιμοποιήθηκε το σύστημα διαμοιραζόμενης μνήμης COMIC [19], το οποίο αποτελεί ένα από τα ελάχιστα διαθέσιμα συστήματα τέτοιου είδους για τον επεξεργαστή Cell B.E. Το σύστημα λογισμικού COMIC αποτελεί μία ολοκληρωμένη υποδομή που περιλαμβάνει ένα σύστημα για την διαχείριση μέσω λογισμικού των επιπέδων κρυφής μνήμης και ένα σύστημα για την εκτέλεση παράλληλων εφαρμογών στον επεξεργαστή Cell (υποσύστημα νημάτων - threading system). Προκειμένου να αφαιρεθεί το υποσύστημα νημάτων, το οποίο δεν ήταν απαραίτητο για την υποδομή GLOpenCL, το σύστημα λογισμικού τροποποιήθηκε ώστε να εκμεταλλευτούμε μόνο τις δυνατότητες για διαχείριση της κρυφής μνήμης. Έπειτα, πραγματοποιήσαμε την ενσωμάτωση αυτού στην υποδομή GLOpenCL. Η χρήση του συστήματος COMIC συνίσταται στην διαχείριση των καθολικών περιοχών μνήμης. Κάθε αναφορά σε στοιχεία αυτών των περιοχών σε μία kernel συνάρτηση μετασχηματίζεται από τον μεταγλωττιστή ώστε να κατευθυνθεί στο λογισμικό διαχείρισης της κρυφής μνήμης. Εν συνεχεία, το λογισμικό εκτελεί όλες τις απαραίτητες ενέργειες στις Local Stores των SPEs για να διασφαλιστεί η συνέπεια των προσπελάσεων στην αντίστοιχη καθολική περιοχή μνήμης.

Το τμήμα της υποδομής COMIC που αντιστοιχεί στον εξυπηρετητή των αιτήσεων εκτελείται στο helper thread στην πλευρά του PPE ενώ το υπόλοιπο του λογισμικού εκτελείται στην πλευρά των worker threads, σε κάθε SPE. Ο εξυπηρετητής διατηρεί στην κύρια μνήμη του PPE τις απαραίτητες δομές για την ορθή λειτουργία του λογισμικού. Το τμήμα του λογισμικού που εκτελείται στα worker threads δημιουργεί αιτήσεις, οι οποίες κατευθύνονται στον εξυπηρετητή. Εν συνεχεία, ο εξυπηρετητής επιστρέφει τα κατάλληλα δεδομένα και ενημερώνει τα worker threads ώστε να επιλυθούν οι οποίες συγκρούσεις και να διασφαλιστεί η συνοχή των δεδομένων. Αυτή η διαδικασία παρουσιάζεται και στο σχήμα 4.2 από τα διακεκομμένα βέλη που αφορούν στις αιτήσεις και στις αντίστοιχες απαντήσεις μεταξύ των worker threads και του helper thread για την software cache (S/W cache request/replies). Όπως μπορεί να γίνει κατανοητό, ένας τέτοιου είδους μηχανισμός είναι περιττός για την υλοποίηση της υποδομής για την αρχιτεκτονική Intel x86 καθώς η διαχείριση της κρυφής μνήμης πραγματοποιείται μέσω υλικού και το πρωτόκολλο συνοχής είναι ήδη υλοποιημένο στο υλικό. Η ύπαρξη αυτού του τμήματος λογισμικού αποτελεί και μία από τις διαφορές στην αρχιτεκτονική της υποδομής για τις δύο αρχιτεκτονικές.

Το τελευταίο τμήμα της υποδομής του συστήματος χρόνου εκτέλεσης για την αρχιτεκτονική Intel x86 είναι το async. copy thread και η αντίστοιχη ουρά. Αυτό είναι και το δεύτερο σημείο στο οποίο διαφέρουν οι δύο υλοποιήσεις. Το μοντέλο OpenCL προσφέρει συναρτήσεις που επιτρέπουν την ασύγχρονη αντιγραφή δεδομένων μεταξύ καθολικών και τοπικών περιοχών μνήμης στο επίπεδο ενός work-group. Κάθε εντολή για ασύγχρονη μεταφορά δεδομένων πρέπει να εκτελεστεί από κάθε work-item που ανήκει σε ένα work-group. Μία εντολή ασύγχρονης μεταφοράς επιστρέφει ένα γεγονός το οποίο μπορεί μετέπειτα να χρησιμοποιηθεί για την αναμονή της ολοκλήρωσης της μεταφοράς.

Προκειμένου να είναι δυνατή η υλοποίηση των ασύγχρονων μεταφορών, χρησιμοποιούμε ένα ακόμη helper thread, το async. copy thread, το οποίο αναλαμβάνει να εξυπηρετήσει τις αιτήσεις που δημιουργούνται από την εκτέλεση μίας εντολής για ασύγχρονη μεταφορά δεδομένων. Αυτό είναι απαραίτητο καθώς δεν παρέχεται πρόσβαση στον μηχανισμό της άμεσης πρόσβασης στην μνήμη (DMA) που παρέχει η αρχιτεκτονική Intel x86, οπότε δεν υπάρχει κάποια άλλη εναλλακτική επιλογή για υλοποίηση των ασύγχρονων μεταφορών. Όταν ένα worker thread εκτελέσει μία τέτοιου είδους ασύγχρονη εντολή, το σύστημα χρόνου εκτέλεσης δημιουργεί μία αίτηση και την εισάγει

στην αντίστοιχη ουρά. Εν συνεχεία, το async. copy thread είναι υπεύθυνο για να αφαιρέσει από την ουρά την επόμενη διαθέσιμη αίτηση, να αντιγράψει τα κατάλληλα δεδομένα και να ενημερώσει το γεγονός που σχετίζεται με αυτή την αίτηση.

Η υλοποίηση των ασύγχρονων μεταφορών δεδομένων στις περισσότερες αρχιτεκτονικές όπου τα επίπεδα της ιεραρχίας κρυφής μνήμης διαχειρίζονται από το λογισμικό είναι πιο άμεση. Συνήθως, αυτού του είδους οι αρχιτεκτονικές παρέχουν στον προγραμματιστή εφαρμογών την πρόσβαση στη διεπαφή για τις πράξεις άμεσης πρόσβασης στην μνήμη (DMA), καθώς αυτός είναι ο μηχανισμός που προτιμάται για την πρόσβαση στην ιεραρχία της μνήμης. Για παράδειγμα στον επεξεργαστή Cell B.E., μία ασύγχρονη πράξη για αντιγραφή δεδομένων υλοποιείται ως μία ή περισσότερες DMA μεταφορές, οι οποίες είναι εκ φύσεως ασύγχρονες. Επομένως, δεν προκύπτει η ανάγκη για την ύπαρξη ενός επιπλέον νήματος το οποίο θα αναλάμβανε την υλοποίηση των ασύγχρονων μεταφορών.

4.4 Δυναμική Εκτέλεση των Kernel Συναρτήσεων

Το τελευταίο χαρακτηριστικό του συστήματος χρόνου εκτέλεσης, είναι η υποστήριξη για την δυναμική εκτέλεση των kernel συναρτήσεων. Όταν ένα worker thread αναλαμβάνει να εκτελέσει ένα work task, θα πρέπει να εκτελέσει την τροποποιημένη kernel συνάρτηση με τον κατάλληλο αριθμό ορισμάτων. Η βέλτιστη προσέγγιση για την δυναμική κλήση μίας kernel συνάρτησης κατά την εκτέλεση της εφαρμογής είναι η χρήση ενός δείκτη σε συνάρτηση. Αυτή η προσέγγιση απαιτεί να είναι γνωστή τόσο η διεύθυνση της συνάρτησης για την αρχικοποίηση του δείκτη όσο και το πλήθος των ορισμάτων της συνάρτησης.

Μία εφαρμογή OpenCL ενδέχεται να καλέσει πολλαπλές kernel συναρτήσεις κατά την διάρκεια εκτέλεσής της. Για τον σκοπό αυτό, το OpenCL API παρέχει συναρτήσεις για την δημιουργία ενός kernel. Κάθε kernel συνάρτηση προσδιορίζεται από το όνομά της, δηλαδή ένα αλφαριθμητικό. Επιπλέον, το πλήθος και ο τύπος των ορισμάτων των διαφόρων kernel συναρτήσεων ενδέχεται να διαφέρει. Επομένως, προκύπτουν πολλαπλές περιπλοκές στην υλοποίηση του σχήματος κλήσης μίας kernel συνάρτησης μέσω ενός δείκτη. Το σύστημα χρόνου εκτέλεσης πρέπει να γνωρίζει εκ των προτέρων το πλήθος και τον τύπο των ορισμάτων της συνάρτησης και να έχει την δυνατότητα να εντοπίσει την διεύθυνση της συνάρτησης χρησιμοποιώντας μόνο την αλφαριθμητική αναπαράσταση του ονόματός της.

Για να καταστεί δυνατή η υποστήριξη kernel συναρτήσεων με μεταβλητό πλήθος και τύπους ορισμάτων, το σύστημα χρόνου εκτέλεσης παρέχει μία διαφανή διεπαφή για την κλήση των kernel συναρτήσεων. Γι' αυτό τον σκοπό, η υποδομή στο επίπεδο του μεταγλωττιστή δημιουργεί μία wrapper συνάρτηση για κάθε τροποποιημένη kernel συνάρτηση στην OpenCL εφαρμογή. Η τεχνική των wrapper συναρτήσεων είναι μία ευρέως διαδεδομένη τεχνική για την υποστήριξη διαφανούς κλήσης μίας συνάρτησης, με κύρια εφαρμογή σε συστήματα χρόνου εκτέλεσης για την υποστήριξη του μοντέλου OpenMP. Μία wrapper συνάρτηση λαμβάνει ένα μοναδικό όρισμα, το οποίο είναι ένας πίνακας void δεικτών. Αυτή η συνάρτηση είναι υπεύθυνη για την εξαγωγή από τον πίνακα των κατάλληλων ορισμάτων και την μετέπειτα κλήση της αντίστοιχης συνάρτησης. Με αυτή την προσέγγιση, κάθε work-task descriptor πρέπει να περιέχει μόνο την διεύθυνση της wrapper συνάρτησης που αντιστοιχεί στον kernel στον οποίο ανήκει το work-task και έναν δείκτη προς τον πίνακα όπου αποθηκεύονται τα ορίσματα της kernel συνάρτησης.

Η αντιμετώπιση του προβλήματος του εντοπισμού της διεύθυνσης της συνάρτησης χρησιμοποιώντας μόνο την αλφαριθμητική αναπαράσταση του ονόματός της είναι απλή για τα συστήματα που υποστηρίζουν δυναμική σύνδεση του κώδικα, όπως είναι τα συστήματα που προορί-

ζονται για την αρχιτεκτονική Intel x86. Η προσέγγιση που ακολουθείται σε αυτή την περίπτωση είναι η φόρτωση όλων των συμβόλων του κώδικα (code symbols) στον πίνακα συμβόλων (symbol table) του παραγόμενου εκτελέσιμου. Η διαδικασία αυτή πραγματοποιείται κατά την φάση σύνδεσης του κώδικα (link time). Έπειτα, μπορούμε να εκμεταλλευτούμε τις δυνατότητες του συστήματος δυναμικής διασύνδεσης κώδικα (run-time dynamic linker) ώστε να εντοπίσουμε την διεύθυνση της συνάρτησης με δεδομένο το όνομά της. Το σύστημα δυναμικής διασύνδεσης κώδικα παρέχει μία διεπαφή για την πρόσβαση στις διάφορες λειτουργίες για άνοιγμα (dlopen()) ή κλείσιμο (dlclose()) ενός αρχείου όπως και εντοπισμό της διεύθυνσης ενός συμβόλου βάσει ονόματος (dlsym()). Χρησιμοποιώντας αυτή την διεπαφή, μπορούμε να ανοίξουμε εκ νέου το εκτελέσιμο που έχει παραχθεί και εντός αυτού να εντοπίσουμε την διεύθυνση της kernel συνάρτησης.

Η ανωτέρω προσέγγιση δεν είναι εφικτή για συστήματα όπου παράγονται διαφορετικά αρχεία για τον host και τις υπολογιστικές συσκευές όπως και για συστήματα όπου δεν υποστηρίζεται δυναμική σύνδεση του κώδικα. Αυτή είναι και η περίπτωση της υλοποίησης για τον επεξεργαστή Cell B.E. Στην υλοποίηση της υποδομής για τον επεξεργαστή Cell, τα εκτελέσιμα που παράγονται για τα SPEs, τα οποία είναι και αυτά που περιέχουν τις kernel-wrapper συναρτήσεις, διασυνδέονται στατικά με το εκτελέσιμο που θα εκτελεστεί στο PPE, προκειμένου να παραχθεί ένα μοναδικό εκτελέσιμο. Επομένως, δεν είναι δυνατή η χρήση του μηχανισμού της δυναμικής αναζήτησης συμβόλων. Η προσέγγιση που υλοποιείται στην υποδομή βασίζεται στην παρατήρηση ότι τόσο το PPE όσο και τα SPEs έχουν πρόσβαση στις περιοχές του εκτελέσιμου αρχείου (Executable and Linkable Format (ELF)) που παράγεται. Επομένως, για την αποθήκευση των κατάλληλων πληροφοριών των kernel-wrapper συναρτήσεων, δεσμεύουμε μία περιοχή στο εκτελέσιμο αρχείο στην οποία έχουν πρόσβαση τόσο το PPE όσο και τα SPEs. Η διεύθυνση της περιοχής αποστέλλεται σε κάθε SPE κατά την αρχικοποίηση του συστήματος χρόνου εκτέλεσης. Για κάθε kernel συνάρτηση στην OpenCL εφαρμογή, η περιοχή αυτή αποθηκεύει ένα ζεύγος που αποτελείται από το όνομα και την διεύθυνση της συνάρτησης. Η αποθήκευση γίνεται με χρήση των κατάλληλων pragmas και attributes που προσφέρει ο μεταγλωττιστής και πραγματοποιείται από την υποδομή μεταγλώττισης, η οποία αναλαμβάνει να τοποθετήσει τα κατάλληλα pragmas και attributes για τις αντίστοιχες συναρτήσεις.

Όταν, κατά την εκτέλεση της εφαρμογής, κληθεί η αντίστοιχη συνάρτηση για την δημιουργία ενός αντικειμένου kernel, το τμήμα της υποδομής που εκτελείται στο PPE προσπελαύνει το κοινό τμήμα και εντοπίζει την θέση στην οποία βρίσκεται η εγγραφή για την kernel-wrapper συνάρτηση. Έπειτα, χρησιμοποιεί την μετατόπιση της θέσης από την αρχή του τμήματος ως το αναγνωριστικό για την αντίστοιχη συνάρτηση. Αυτό το αναγνωριστικό αποθηκεύεται στους work-descriptors που παράγονται για τον kernel και αποστέλλεται στα SPEs που εκτελούν τα work-tasks. Κάθε SPE χρησιμοποιεί αυτή την μετατόπιση για να δεικτοδοτήσει το κοινό τμήμα ώστε να αποκτήσει τον δείκτη προς την αντίστοιχη συνάρτηση, τον οποίο και χρησιμοποιεί για την κλήση της συνάρτησης.

Οι δύο προαναφερθείσες προσεγγίσεις λειτουργούν συμπληρωματικά. Το επιπρόσθετο, και επιθυμητό, αποτέλεσμα της τελευταίας είναι η επίλυση του προβλήματος της περιορισμένης χωρητικότητας της Local Store των SPEs, όσον αφορά στο μέγεθος του χώρου που είναι διαθέσιμος για την αποθήκευση του κώδικα. Οι Local Stores των SPEs είναι ενοποιημένες και αποθηκεύουν τόσο τον κώδικα της εφαρμογής που εκτελείται στο SPE όσο και τα δεδομένα. Αυτό έχει ως συνέπεια ο διαθέσιμος χώρος να μην είναι επαρκής για εφαρμογές που χρησιμοποιούν αρκετές kernel συναρτήσεις κατά την διάρκεια εκτέλεσής τους, καθώς η Local Store δεν θα μπορεί να αποθηκεύσει τον κώδικα της κάθε συνάρτησης. Όμως, η προσέγγιση που ακολουθείται για την κλήση μίας kernel συνάρτησης μέσω ενός pointer, σε συνδυασμό με τον μηχανισμό των SPU

code overlays [26], μπορεί να επιλύσει το πρόβλημα της περιορισμένης χωρητικότητας της Local Store. Ο μηχανισμός των overlays προσφέρει την δυνατότητα για μεταφορά εντολών από και προς την Local Store. Ένα overlay είναι ένα αντικείμενο στο object αρχείο ενός Synergistic Processing Unit (SPU) το οποίο μπορεί να φορτωθεί δυναμικά σε μία συγκεκριμένη περιοχή κατά την εκτέλεση της εφαρμογής. Ο μηχανισμός επιτρέπει πολλαπλά overlays να φορτώνονται στην ίδια περιοχή μνήμης αλλά σε διαφορετικές χρονικές στιγμές καθώς κάθε νέο overlay διαγράφει το προηγούμενο. Επομένως, μπορούμε να χρησιμοποιήσουμε τον μηχανισμό των code overlays και σε συνδυασμό με την κλήση συνάρτησης μέσω pointer να επιτύχουμε δυναμική φόρτωση του κώδικα των συναρτήσεων στην Local Store των SPEs, επεκτείνοντας έτσι "εικονικά" την διαθέσιμη μνήμη.

Κεφάλαιο 5

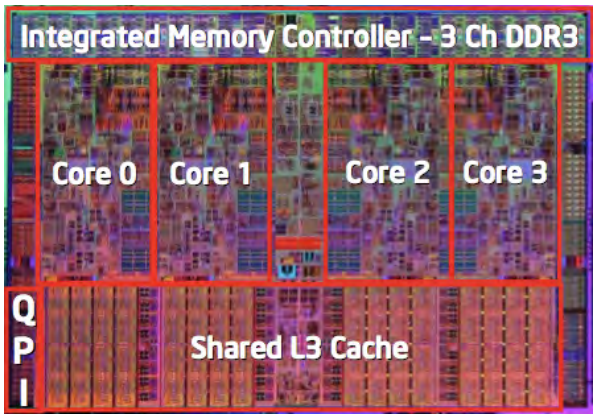
Πειραματική Αξιολόγηση

5.1 Εισαγωγή

Προκειμένου να διαπιστωθεί η αποδοτικότητα της προτεινόμενης υποδομής, πραγματοποιήθηκε η αξιολόγηση της υποδομής GLOpenCL σε δύο σύγχρονες αρχιτεκτονικές. Σε αυτό το κεφάλαιο παρουσιάζονται τα αποτελέσματα από την αξιολόγηση της υποδομής σε δύο αντιπροσωπευτικές αρχιτεκτονικές και χρησιμοποιώντας μετρικές εφαρμογές (benchmarks) από τις υλοποιήσεις του μοντέλου OpenCL που παρέχουν οι κατασκευαστές AMD/ATI και IBM. Πιο συγκεκριμένα, πραγματοποιήσαμε την αξιολόγηση της υποδομής σε έναν ομογενή επεξεργαστή πολλαπλών πυρήνων, όπου ο έλεγχος της ιεραρχίας κρυφής μνήμης πραγματοποιείται μέσω υλικού, και σε έναν ετερογενή επεξεργαστή πολλαπλών πυρήνων με έλεγχο της ιεραρχίας κρυφής μνήμης μέσω λογισμικού.

Ο ομογενής επεξεργαστής πολλαπλών πυρήνων που χρησιμοποιήθηκε είναι ο επεξεργαστής Intel E5520 i7, ο οποίος βασίζεται στην αρχιτεκτονική Intel Nehalem. Ο επεξεργαστής λειτουργεί στην συχνότητα των $2.27GHz$ και το σύστημα που χρησιμοποιήσαμε ενσωματώνει μνήμη RAM με μέγεθος $4GB$. Ο επεξεργαστής ενσωματώνει τέσσερις πανομοιότυπους πυρήνες. Κάθε πυρήνας διαθέτει διαχωριζόμενη κρυφή μνήμη δεδομένων και εντολών επιπέδου L1, με χωρητικότητα $32KB$ η κάθε μία. Στο επίπεδο L2, ο κάθε πυρήνας διαθέτει μία ενοποιημένη κρυφή μνήμη εντολών και δεδομένων, με μέγεθος ίσο με $256KB$. Όλοι οι πυρήνες διαμοιράζονται μία κρυφή μνήμη επιπέδου L3 με χωρητικότητα ίση με $8MB$. Η κρυφή μνήμη του επιπέδου L3 βασίζεται στην inclusion property των ιεραρχιών κρυφής μνήμης, σύμφωνα με την οποία κάθε υψηλότερο επίπεδο στην ιεραρχία κρυφής μνήμης εμπεριέχεται στα χαμηλότερα επίπεδα. Επί παραδείγματι, τα περιεχόμενα του επιπέδου L1 εμπεριέχονται στην κρυφή μνήμη επίπεδου L2. Το σχήμα 5.1 απεικονίζει την αρχιτεκτονική του επεξεργαστή Intel E5520.

Όσον αφορά στον ετερογενή επεξεργαστή που χρησιμοποιήθηκε, αυτός είναι ο επεξεργαστής Cell B.E. Ο επεξεργαστής Cell B.E. αποτελεί μία αντιπροσωπευτική αρχιτεκτονική ετερογενούς επεξεργαστή πολλαπλών πυρήνων, με ιεραρχία κρυφής μνήμης που ελέγχεται από το λογισμικό. Το σύστημα που χρησιμοποιήθηκε ήταν ένα σύστημα IBM QS20 Cell Blade το οποίο ενσωματώνει δύο επεξεργαστές Cell, με συχνότητα λειτουργίας ίση με $3.2GHz$. Για την πειραματική αξιολόγηση χρησιμοποιήθηκε μόνο ο ένας από τους δύο διαθέσιμους επεξεργαστές. Ο επεξεργαστής Cell ενσωματώνει έναν πυρήνα (PPE) που βασίζεται στην αρχιτεκτονική PowerPC και υλοποιεί εκτέλεση νημάτων δύο δρόμων σε επίπεδο υλικού (2-way Simultaneous Multithreading (SMT)) και οκτώ συνεργατικά στοιχεία επεξεργασίας (Synergistic Processing Elements (SPEs)). Κάθε SPE έχει πρόσβαση σε μία τοπική μνήμη (Local Store) που ελέγχεται από το λογισμικό και έχει



Σχήμα 5.1: Η αρχιτεκτονική του επεξεργαστή Intel E5520.

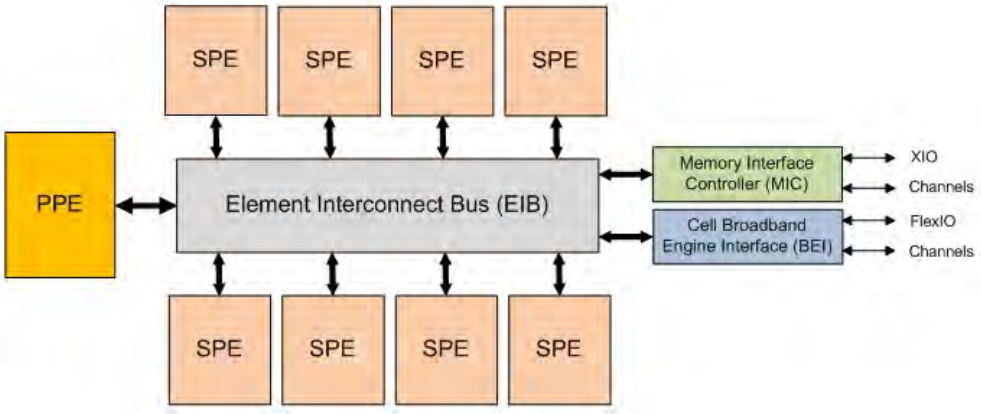
μέγεθος ίσο με $256KB$. Τα επιμέρους τμήματα του επεξεργαστή συνδέονται μεταξύ τους με έναν δίαυλο υψηλής απόδοσης, το Element Interconnect Bus (EIB). Επιπλέον, το σύστημα που χρησιμοποιήθηκε ενσωματώνει μνήμη RAM με χωρητικότητα ίση με $1GB$. Η αρχιτεκτονική του επεξεργαστή παρουσιάζεται στο σχήμα 5.2.

Για την μεταγλώττιση των εφαρμογών για την εκτέλεσή τους με την υποδομή GLOpenCL χρησιμοποιήθηκε ο Intel C Compiler (v11.1) για τον επεξεργαστή Intel E5520 i7 και ο μεταγλωττιστής IBM XL Compiler (v9.0) για τον επεξεργαστή Cell B.E. Όσον αφορά στην μεταγλώττιση των εφαρμογών για την εκτέλεση με τις υποδομές που παρέχονται από την AMD/ATI και την IBM, αυτή πραγματοποιήθηκε χρησιμοποιώντας την ροή μεταγλώττισης του αντίστοιχου SDK. Σε κάθε περίπτωση, χρησιμοποιήθηκαν οι επιλογές (flags) του μεταγλωττιστή που οδηγούσαν στην παραγωγή εκτελέσιμου με τον μικρότερο χρόνο εκτέλεσης. Σε αυτό το σημείο, θα πρέπει να αναφερθεί ότι δεν χρησιμοποιήθηκαν οι δυνατότητες που προσφέρει το OpenCL API για την μεταγλώττιση του κώδικα των kernel συναρτήσεων κατά την διάρκεια εκτέλεσης του προγράμματος (just-in-time compilation).

Οι διαθέσιμες υποδομές που χρησιμοποιήθηκαν για την σύγκριση της προτεινόμενης υποδομής (GLOpenCL) είναι η υποδομή Stream SDK (v2.1) από την AMD/ATI για τον επεξεργαστή Intel E5520 i7 και η υποδομή OpenCL SDK (v0.1.1) που παρέχεται από την IBM για τον επεξεργαστή Cell σε συστήματα QS20 blade servers. Και οι δύο υποδομές είναι σχεδιασμένες και βελτιστοποιημένες για την υποστήριξη αποδοτικής εκτέλεσης των εφαρμογών στην αντίστοιχη αρχιτεκτονική. Σε αυτό το σημείο θα πρέπει να αναφερθεί ότι τα αποτελέσματα της αξιολόγησης των υποδομών αναφέρονται στο σύνολο της υποδομής. Αυτό σημαίνει ότι δεν αξιολογούνται ξεχωριστά τα επιμέρους τμήματα των υποδομών, δηλαδή η υποδομή μεταγλώττισης και η υποδομή συστήματος χρόνου εκτέλεσης. Άλλωστε, αυτό δεν ήταν δυνατόν καθώς οι διαθέσιμες υποδομές είναι closed-source, με το υποσύστημα μεταγλώττισης να είναι άμεσα συνδεδεμένο με το σύστημα χρόνου εκτέλεσης. Αυτό έχει ως αποτέλεσμα να μην είναι δυνατή η μεταφορά εκτελέσιμων που έχουν παραχθεί σε μία υποδομή για εκτέλεση σε κάποια άλλη υποδομή.

5.2 Περιγραφή των Εφαρμογών

Για την αξιολόγηση της αποδοτικότητας της προτεινόμενης υποδομής χρησιμοποιήσαμε ένα σύνολο εφαρμογών (benchmarks) που παρέχονται μαζί με τις διαθέσιμες υποδομές, που υλοποιούν



Σχήμα 5.2: Η αρχιτεκτονική του επεξεργαστή Cell.

το μοντέλο OpenCL, από την AMD / ATI και την IBM. Επιλέξαμε ετερογενείς εφαρμογές ώστε να είναι δυνατή η αξιολόγηση της υποδομής για εφαρμογές με διαφορετικό φόρτο εργασίας και με διαφορετικά χαρακτηριστικά. Οι ακόλουθες παράγραφοι περιγράφουν τα χαρακτηριστικά κάθε μίας από τις εφαρμογές που χρησιμοποιήσαμε.

Εφαρμογή Vector Add

Η εφαρμογή Vector Add είναι μία αρκετά απλή εφαρμογή και χρησιμοποιείται ως running example. Η λειτουργία που επιτελεί η εφαρμογή είναι η πρόσθεση των στοιχείων δύο διανυσμάτων. Η εφαρμογή χαρακτηρίζεται από τις μεγάλες απαιτήσεις σε επικοινωνία καθώς το ποσοστό του χρόνου εκτέλεσης που αφιερώνεται σε υπολογισμούς είναι αρκετά μικρό. Η εφαρμογή δεν εμφανίζει χωρική ή χρονική τοπικότητα όσον αφορά στην πρόσβαση στα δεδομένα.

Εφαρμογή 2D DCT

Η δεύτερη εφαρμογή που χρησιμοποιήθηκε είναι η εφαρμογή 2D DCT. Η εφαρμογή υπολογίζει τον δισδιάστατο διακριτό μετασχηματισμό συνημιτόνου Discrete Cosine Transform (DCT) σε μία εικόνα που παρέχεται ως είσοδος. Η εφαρμογή εκτελεί τους υπολογισμούς σε blocks και αποτελεί μία υπολογιστικά πολύπλοκη εφαρμογή. Η εκτέλεση των υπολογισμών κατά blocks έχει ως αποτέλεσμα η εφαρμογή να χαρακτηρίζεται από την εμφάνιση χωρικής και χρονικής τοπικότητας.

Εφαρμογή Sobel Filter

Η εφαρμογή Sobel Filter [27] αποτελεί μία εφαρμογή επεξεργασίας εικόνας και υπολογίζει τον Sobel operator. Ο Sobel operator είναι ένας διακριτός τελεστής διαφοροποίησης (discrete differentiation) και χρησιμοποιείται για εφαρμογές ανίχνευσης ακμών (edge detection). Η εφαρμογή παρουσιάζει παρόμοια χαρακτηριστικά με την εφαρμογή 2D DCT (5.2), δηλαδή εκτελεί τους υπολογισμούς κατά blocks, απαιτεί σημαντική υπολογιστική ισχύ και εμφανίζει τόσο χωρική όσο και χρονική τοπικότητα.

Εφαρμογή 2D AES

Η πέμπτη εφαρμογή είναι μία εφαρμογή από τον τομέα της κρυπτογραφίας. Η εφαρμογή αυτή εκτελεί τον αλγόριθμο AES [21] για την κρυπτογράφηση μίας εικόνας. Ο αλγόριθμος AES είναι ένας αλγόριθμος συμμετρικού κλειδιού και ανήκει στην κατηγορία των block cipher αλγορίθμων. Η εφαρμογή αποτελεί την υπολογιστικά πολυπλοκότερη εφαρμογή και εμφανίζει παρόμοια χαρακτηριστικά με τις εφαρμογές Sobel Filter (5.2) και 2D DCT (5.2).

Εφαρμογή BlackScholes

Η εφαρμογή BlackScholes [5] επιλύει τις μερικές διαφορικές εξισώσεις (PDEs) των Black-Scholes για έναν αριθμό από μετοχές. Η επίλυση αυτού του μαθηματικού τύπου βρίσκει ευρεία εφαρμογή σε μετοχές όπου το δικαίωμα επιλογής μπορεί να εξασκηθεί μόνο κατά την ημερομηνία λήξης (European-style options). Χαρακτηριστικά της εφαρμογής αποτελούν οι μεγάλες απαιτήσεις σε υπολογιστική ισχύ και οι πολύπλοκες μαθηματικές πράξεις αριθμητικής κινητής υποδιαστολής που χρησιμοποιεί η εφαρμογή. Αυτές αποτελούν έναν εν δυνάμει παράγοντα που μπορεί να μειώσει την απόδοση της εφαρμογής σε αρχιτεκτονικές όπου αυτού του είδους οι πράξεις δεν υποστηρίζονται αποδοτικά, όπως ο επεξεργαστής Cell. Επίσης, η εφαρμογή χαρακτηρίζεται από την ύπαρξη χωρικής τοπικότητας ενώ δεν εμφανίζει χρονική τοπικότητα στην προσπέλαση των δεδομένων.

Για την εφαρμογή BlackScholes χρησιμοποιήθηκαν δύο εκδόσεις των kernel συναρτήσεων, όπως αυτές παρέχονται από το SDK της IBM. Η πρώτη έκδοση χρησιμοποιεί global buffers, τους οποίους και προσπελαύνει απευθείας, οπότε οι προσπελάσεις μνήμης κατευθύνονται είτε στο υλικό είτε στο λογισμικό που ελέγχει την ιεραρχία της κρυφής μνήμης ενώ η δεύτερη έκδοση (async. version) χρησιμοποιεί την δυνατότητα για ασύγχρονη μεταφορά των δεδομένων (asynchronoys copy) που παρέχει το μοντέλο OpenCL, σε ένα σχήμα double buffering όπου χρησιμοποιούνται δύο buffers για την αποθήκευση των δεδομένων. Η δεύτερη έκδοση της kernel συνάρτησης χρησιμοποιήθηκε για να διαπιστωθεί κατά πόσο βελτιστοποιήσεις που αφορούν συγκεκριμένες αρχιτεκτονικές μπορούν να επιδράσουν θετικά στην απόδοση κάποιας εφαρμογής.

5.3 Αποτελέσματα Αξιολόγησης στον Επεξεργαστή Intel E5520

Αρχικά, όλες οι εφαρμογές εκτελέστηκαν στον επεξεργαστή Intel E5520 Core i7, με χρήση του μέγιστου διαθέσιμου αριθμού πλαισίων εκτέλεσης (νημάτων) που προσφέρει η αρχιτεκτονική. Κάθε εφαρμογή εκτελέστηκε τόσο για διαφορετικά μεγέθη συνόλου εργασίας όσο και για διαφορετικά μεγέθη work-group, ώστε να μελετηθεί η επίδραση αυτών των μεγεθών στην απόδοση της υποδομής. Για την διευκόλυνση του αναγνώστη, τα γραφήματα με τα αποτελέσματα της αξιολόγησης της υποδομής για την αρχιτεκτονική Intel x86 παρατίθενται στα σχήματα A.1 - A.6. Τα σχήματα παρουσιάζουν τους χρόνους εκτέλεσης των εφαρμογών στην υποδομή GLOpenCL και στην υποδομή που παρέχεται από την AMD/ATI (AMD OpenCL). Το κάτω τμήμα του άξονα x απεικονίζει τα μεγέθη του συνόλου εργασίας ενώ το άνω τμήμα παρουσιάζει τα μεγέθη του work-group για τις διάφορες εκτελέσεις των εφαρμογών. Κάποιες από τις εφαρμογές χρησιμοποιούν τεμαχισμό του χώρου δεικτών σε δύο διαστάσεις. Σε αυτή την περίπτωση, οι τιμές στον άξονα x παρουσιάζουν το μέγεθος της κάθε διάστασης, τόσο για τον χώρο δεικτών όσο και για το κάθε work-group.

Όπως μπορούμε να παρατηρήσουμε, η προτεινόμενη υποδομή, παρά την γενικότητα της αρχιτεκτονικής της, επιτυγχάνει καλύτερη απόδοση από την υποδομή που παρέχεται από την

AMD/ATI σε όλες τις εφαρμογές που χρησιμοποιήθηκαν. Η μέση επιτάχυνση εκτέλεσης που επιτυγχάνει η υποδομή είναι ίση με $1.84x$, με την μέγιστη επιτάχυνση να ανέρχεται σε $2.67x$. Επίσης, μπορούμε να παρατηρήσουμε ότι ο χρόνος εκτέλεσης κλιμακώνεται γραμμικά ως προς την αύξηση του μεγέθους του συνόλου εργασίας. Αυτό είναι εξαιρετικά σημαντικό καθώς συνεπάγεται ότι η υποδομή δεν εισάγει επιπλέον επιβάρυνση στην απόδοση της εκάστοτε εφαρμογής με την αύξηση του μεγέθους του συνόλου εργασίας. Συν τοις άλλοις, η αύξηση του μεγέθους των work-groups που δημιουργούνται επιδρά θετικά στην απόδοση της εφαρμογής, μειώνοντας τον συνολικό χρόνο εκτέλεσης. Η αύξηση του μεγέθους του work-group έχει ως αποτέλεσμα την μείωση του αριθμού των work-groups που δημιουργούνται οπότε μειώνεται η επιβάρυνση που προκύπτει από την δημιουργία, την διαχείριση και την εκτέλεση των work-tasks που δημιουργεί η υποδομή. Επιπλέον, η αύξηση του μεγέθους του work-group επιδρά θετικά στην απόδοση και των δύο υποδομών καθώς αυξάνεται η χρονική τοπικότητα των εφαρμογών, κυρίως σε εφαρμογές που εκτελούν τους υπολογισμούς κατά blocks.

Μία ακόμη ενδιαφέρουσα παρατήρηση αποτελεί το γεγονός της μηδενικής επίδρασης της χρήσης των ασύγχρονων πράξεων για μεταφορές δεδομένων (asynchronous copy) στην απόδοση των υποδομών. Η αντικατάσταση των άμεσων αναφορών στους global buffers της εκάστοτε εφαρμογής, οι οποίες υποβοηθούνται από το υλικό, με ασύγχρονες πράξεις αντιγραφής δεδομένων και χρήση της τεχνικής του double-buffering για την εφαρμογή Black-Scholes (βλ. σχήμα A.6) δεν επιφέρει κάποια μετρήσιμα οφέλη στην απόδοση. Οι σύγχρονες αρχιτεκτονικές ενσωματώνουν ιεραρχίες μνήμης που έχουν την δυνατότητα να εξαλείφουν την καθυστέρηση που οφείλεται στις προσβάσεις μνήμης. Αυτό συμβαίνει κυρίως για προσβάσεις που δεν απαιτούν συναλλαγές (transactions) σε δίκτυα διασύνδεσης με πολλαπλούς κόμβους και για βρόγχους με κανονική δομή, όπου συνήθεις βελτιστοποιήσεις - όπως το ξεδίπλωμα βρόγχων (loop unrolling) - αλλά και άλλες βελτιστοποιήσεις με έμφαση στην καλύτερη απόδοση - όπως η προφόρτωση δεδομένων από τον μεταγλωττιστή (compiler assisted prefetching) - μπορούν να εφαρμοσθούν. Αυτό ισχύει για τις περισσότερες αρχιτεκτονικές επεξεργαστών πολλαπλών πυρήνων με μεγάλης χωρητικότητας, κρυφές μνήμες στα τελευταία επίπεδα της ιεραρχίας μνήμης, όπως για παράδειγμα ο επεξεργαστής Intel E5520 που ενσωματώνει κρυφή μνήμη επιπέδου L3 με χωρητικότητα ίση με $8MB$. Συνήθως, αυτού του είδους οι αρχιτεκτονικές είναι πιο "επιεικείς" στην εκτέλεση μη βελτιστοποιημένων εφαρμογών, ώστε να μην επηρεάζουν σε μεγάλο βαθμό την απόδοσή τους.

Τέλος, από τα πειραματικά αποτελέσματα παρατηρούμε ότι η υποδομή GLOpenCL μπορούσε να εκτελέσει τις εφαρμογές με μεγαλύτερο μέγεθος των work-groups αλλά και να αντιμετωπίσει επιτυχώς μεγαλύτερα σύνολα εργασίας των εφαρμογών. Το μεγαλύτερο μέγεθος του συνόλου εργασίας που παρουσιάζεται στα διαγράμματα είναι το όριο πέραν του οποίου η υποδομή AMD/ATI OpenCL SDK δεν μπορούσε να πραγματοποιήσει την δέσμευση των απαιτούμενων buffers. Εξαίρεση αποτελούν οι εφαρμογές AES και Sobel Filter οι οποίες χρησιμοποιούνται για την απεικόνιση αυτού του χαρακτηριστικού. Συν τοις άλλοις, σε κάποιες περιπτώσεις, η υποδομή AMD/ATI OpenCL SDK δεν μπόρεσε να εκμεταλλευτεί την αύξηση του μεγέθους του work-group για δεδομένο μέγεθος του συνόλου εργασίας. Χαρακτηριστικά παραδείγματα αποτελούν οι εφαρμογές Vector Add, DCT όπως επίσης και οι δύο εκδόσεις της εφαρμογής BlackScholes.

5.4 Αποτελέσματα Αξιολόγησης στον Επεξεργαστή Cell B.E.

Μετά την αξιολόγηση της υποδομής σε μία αρχιτεκτονική όπου η ιεραρχία κρυφής μνήμης ελέγχεται από το υλικό, πραγματοποιήθηκε αξιολόγηση αυτής στον επεξεργαστή Cell B.E, ο οποίος ενσωματώνει ιεραρχία μνήμης ελεγχόμενη από το λογισμικό. Τα σχήματα A.7 - A.12 παρουσιάζουν

τα αποτελέσματα της εκτέλεσης των εφαρμογών για την υποδομή GLOpenCL και την υποδομή IBM OpenCL SDK. Η υποδομή IBM OpenCL SDK εκτελεί τις εφαρμογές κατά μέσο όρο 27.5% πιο γρήγορα από την υποδομή GLOpenCL. Αυτή η διαφορά μπορεί να αιτιολογηθεί από την διαφορετική απόδοση ανάμεσα στις δύο υλοποιήσεις που χρησιμοποιούνται για έλεγχο της ιεραρχίας μνήμης μέσω λογισμικού στις δύο υποδομές. Η υποδομή που παρέχεται από την IBM χρησιμοποιεί μία βελτιστοποιημένη υλοποίηση κρυφής μνήμης σε λογισμικό (software cache), η οποία επιτυγχάνει πολύ μεγαλύτερη απόδοση από την αντίστοιχη υλοποίηση που προσφέρεται στο SDK της IBM για την ανάπτυξη εφαρμογών στον επεξεργαστή Cell. Επίσης, η υλοποίηση της κρυφής μνήμης είναι ενσωματωμένη με τον μεταγλωττιστή που χρησιμοποιείται για την μεταγλώττιση των OpenCL εφαρμογών και αυτό επιτρέπει την ανάλυση του προτύπου που εμφανίζουν οι προσβάσεις στα δεδομένα (memory access pattern analysis) καθώς και την πραγματοποίηση κατάλληλων βελτιστοποιήσεων κατά την διάρκεια μεταγλώττισης της εφαρμογής. Χαρακτηριστικό είναι το παράδειγμα της εφαρμογής Vector Add, η οποία εμφανίζει μεγάλες απαιτήσεις σε επικοινωνία. Σε αυτή την εφαρμογή η υποδομή IBM OpenCL επιτυγχάνει καλύτερη απόδοση κατά 52.6% από τον συνδυασμό της υποδομής GLOpenCL και του τμήματος λογισμικού COMIC.

Εάν δεν λάβουμε υπόψη μας την εφαρμογή Vector Add, η διαφορά στην απόδοση μεταξύ της υποδομής GLOpenCL και της υποδομής που παρέχεται από την IBM μειώνεται σε 19.1% ή ακόμη και σε 13.2% όταν χρησιμοποιηθεί το βέλτιστο, όσον αφορά στο μέγεθος του συνόλου εργασίας, μέγεθος για τα work-groups. Η συμπεριφορά αυτή εμφανίζεται στις εφαρμογές 2D DCT και στις δύο εκδόσεις της εφαρμογής BlackScholes. Επίσης, για εφαρμογές με μεγάλες υπολογιστικές απαιτήσεις, όπως είναι οι εφαρμογές AES και Sobel, η διαφορά στην απόδοση των δύο υποδομών είναι αμελητέα, κυρίως για μεγάλα μεγέθη του συνόλου εργασίας.

Η υποδομή της IBM για την υποστήριξη του μοντέλου OpenCL επιτυγχάνει καλύτερη απόδοση από την υποδομή GLOpenCL και για την έκδοση της εφαρμογής BlackScholes όπου πραγματοποιείται ασύγχρονη μεταφορά των δεδομένων μέσω πράξεων asynchronous copy. Η χρήση του μηχανισμού των ασύγχρονων μεταφορών δεδομένων έχει ως αποτέλεσμα την απενεργοποίηση του λογισμικού για έλεγχο της κρυφής μνήμης, οπότε η απόδοση του αντίστοιχου λογισμικού δεν επηρεάζει την συνολική απόδοση της υποδομής. Αυτό όμως δεν συμβαίνει και για το λογισμικό COMIC. Όπως αναφέρθηκε και στην ενότητα 4.3, το λογισμικό COMIC ενσωματώνει τον μηχανισμό ελέγχου της κρυφής μνήμης μαζί με ένα σύστημα για την διαχείριση νημάτων. Παρόλο που έχουμε απενεργοποιήσει το μεγαλύτερο μέρος του τμήματος για την διαχείριση των νημάτων και έχουμε ενοποιήσει το λογισμικό με το σύστημα χρόνου εκτέλεσης της υποδομής GLOpenCL, κάποιες λειτουργίες του λογισμικού COMIC για την διαχείριση της κρυφής μνήμης είναι άρρηκτα συνδεδεμένες με το σύστημα διαχείρισης νημάτων και γι' αυτό τον λόγο δεν μπορούν να απενεργοποιηθούν. Επομένως, αυτές οι λειτουργίες παρεμβάλλονται με το κυρίως και το βοηθητικό νήμα της υποδομής GLOpenCL ακόμη και όταν ο μηχανισμός της κρυφής μνήμης δεν χρησιμοποιείται από το τμήμα της εφαρμογής που εκτελείται στα SPEs. Αυτό έχει ως αποτέλεσμα την επιπλέον επιβάρυνση της υποδομής, επιβάρυνση που δεν υπάρχει στην υποδομή της IBM.

Μία αρκετά ενδιαφέρουσα παρατήρηση είναι η δεκαπλάσια αύξηση στην απόδοση και των δύο υποδομών όταν χρησιμοποιείται ο μηχανισμός των ασύγχρονων μεταφορών δεδομένων, σε συνδυασμό με την βελτιστοποίηση του double-buffering, στην εφαρμογή BlackScholes. Παρά το γεγονός ότι το μοντέλο προγραμματισμού OpenCL έχει σχεδιασθεί ώστε να αποτελέσει ένα πλαίσιο ανάπτυξης εφαρμογών, όπου οι εφαρμογές δεν θα πρέπει να γραφούν εκ νέου προκειμένου να μεταφερθούν σε διαφορετικές αρχιτεκτονικές, η προαναφερθείσα παρατήρηση είναι ενδεικτική των σημαντικών οφελών που έχει στην απόδοση των εφαρμογών η απεικόνιση της εκάστοτε εφαρμογής στα χαρακτηριστικά της αρχιτεκτονικής στην οποία πρόκειται να εκτελεστεί.

Τέλος, όπως και στην περίπτωση της αρχιτεκτονικής Intel x86, η υποδομή GLOpenCL μπο-

ρούσε να εργασθεί τόσο με μεγαλύτερα μεγέθη των work-groups όσο και με μεγαλύτερα μεγέθη προβλημάτων σε σχέση με την υποδομή IBM OpenCL SDK στο σύστημα Cell blade. Αυτό αποτελεί και ένα από τα πλεονεκτήματα της γενικής αρχιτεκτονικής της υποδομής GLOpenCL. Καθώς δεν είναι βελτιστοποιημένη για μία συγκεκριμένη αρχιτεκτονική, επιτυγχάνει την εκτέλεση των εφαρμογών χωρίς να θέτει αυστηρούς περιορισμούς όσον αφορά στα μεγέθη του συνόλου εργασίας και των work-groups.

Κεφάλαιο 6

Επίλογος

Στο πλαίσιο της παρούσας εργασίας παρουσιάστηκαν οι γενικές αρχές, η υλοποίηση και η αξιολόγηση της υποδομής GLOpenCL. Η υποδομή GLOpenCL είναι μία ενοποιημένη υποδομή που περιλαμβάνει έναν μεταγλωττιστή και ένα σύστημα χρόνου εκτέλεσης για την υποστήριξη εκτέλεσης εφαρμογών που έχουν αναπτυχθεί βάσει του μοντέλου OpenCL σε παράλληλα επεξεργαστικά συστήματα με διαφορετικά αρχιτεκτονικά χαρακτηριστικά. Πιο συγκεκριμένα, η υποδομή επιτρέπει την εκτέλεση εφαρμογών τόσο σε αρχιτεκτονικές όπου η ιεραρχία μνήμης (memory hierarchy) ελέγχεται από το υλικό (hardware-controlled memory hierarchy) όσο και σε αρχιτεκτονικές όπου η ιεραρχία μνήμης ελέγχεται από το λογισμικό (software-controlled memory hierarchy). Η υποδομή βασίζεται σε μία γενική αρχιτεκτονική η οποία επιτρέπει την εκτέλεση εφαρμογών στις δύο αρχιτεκτονικές με ελάχιστες τροποποιήσεις στο επίπεδο του συστήματος χρόνου εκτέλεσης. Η προτεινόμενη υποδομή επιτυγχάνει παρόμοια, ή ακόμη και καλύτερη, απόδοση από τις υλοποιήσεις που παρέχονται από τις AMD/ATI και IBM, υλοποιήσεις που είναι βελτιστοποιημένες για τις αρχιτεκτονικές που προορίζονται.

Τα αποτελέσματα από την πειραματική αξιολόγηση της υποδομής καταδεικνύουν ότι ακόμη και όταν οι προγραμματιστές χρησιμοποιούν προγραμματιστικά μοντέλα που υποστηρίζουν συστήματα με μεγάλες αρχιτεκτονικές διαφορές ή ακόμη και ετερογενή συστήματα, η γνώση των λεπτομερειών της αρχιτεκτονικής στην οποία πρόκειται να εκτελεστεί η εφαρμογή και η προσεκτική απεικόνιση αυτής στα χαρακτηριστικά της αρχιτεκτονικής μπορούν να επιφέρουν σημαντικά οφέλη στην απόδοση. Σε πολλές περιπτώσεις, αυτά τα οφέλη είναι δύσκολο να μην ληφθούν υπόψη, ακόμη και εάν η μεταφερσιμότητα της εφαρμογής αποτελεί έναν από τους πρωτεύοντες στόχους.

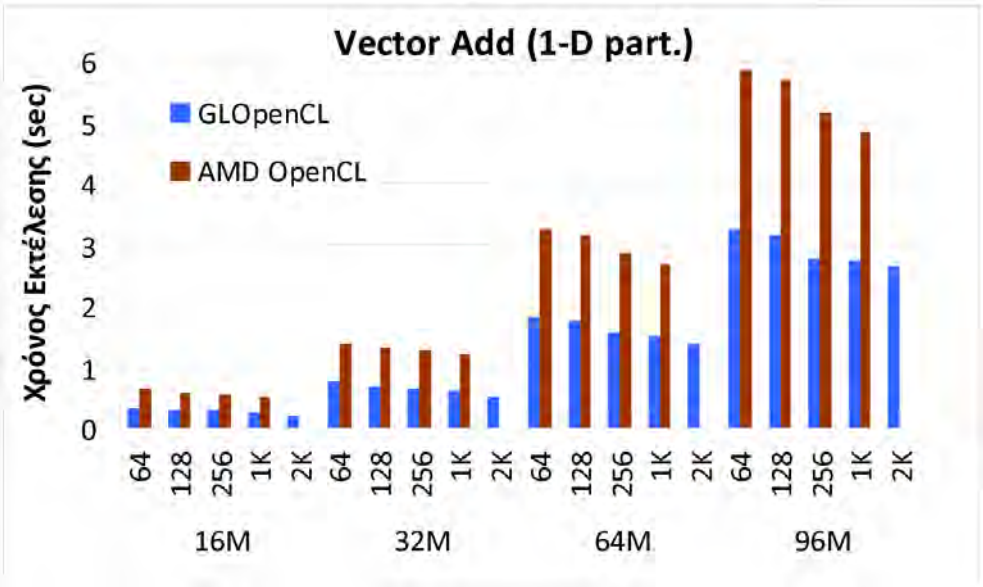
Σε μελλοντική εργασία, σχεδιάζουμε να ερευνήσουμε την δυνατότητα μείωσης της λειτουργικότητας του λογισμικού για τον έλεγχο της ιεραρχίας κρυφής μνήμης, ή ακόμη και την αφαίρεση αυτού, για συστήματα όπου ο έλεγχος της ιεραρχίας μνήμης πραγματοποιείται από το λογισμικό. Αυτό προκύπτει από την παρατήρηση ότι αυτό το τμήμα λογισμικού αποτελούσε τον κυρίαρχο ανασχετικό παράγοντα για την απόδοση της υποδομής στον επεξεργαστή Cell B.E. Η ανάγκη για την χρήση ενός τέτοιου λογισμικού προκύπτει από την εξοικείωση των προγραμματιστών με μοντέλα προγραμματισμού που βασίζονται σε κοινή μνήμη (shared address space). Αυτή η εξοικείωση έχει ως αποτέλεσμα να χρησιμοποιούνται διαμοιραζόμενες δομές δεδομένων, ακόμη και όταν αυτό θα μπορούσε να έχει αποφευχθεί με κατάλληλο τεμαχισμό των δεδομένων. Χαρακτηριστικό παράδειγμα αποτελεί η εφαρμογή BlackScholes που μελετήθηκε στην παρούσα εργασία. Στην δεύτερη έκδοση της εφαρμογής, ο προγραμματιστής είχε πραγματοποιήσει τον τεμαχισμό των δεδομένων, με αποτέλεσμα την μείωση στην χρήση των καθολικών περιοχών

μνήμης και την συνεπακόλουθη εξάλειψη της επίδρασης, στην απόδοση της εφαρμογής, του τμήματος λογισμικού για τον έλεγχο της ιεραρχίας μνήμης. Τεχνικές όπως ο αυτόματος τεμαχισμός κώδικα (automatic code slicing) και ο υπολογισμός εκ των προτέρων των διευθύνσεων μνήμης (memory address precomputation) που προσπελαύνει η εφαρμογή, είτε στο τμήμα του κώδικα που εκτελείται στον host είτε στο τμήμα που εκτελείται στις υπολογιστικές συσκευές, μπορούν να αποδειχθούν ως σημαντικό εργαλείο για την μείωση της επιβάρυνσης που προκαλείται από τις ανάγκες της εκάστοτε εφαρμογής για επικοινωνία.

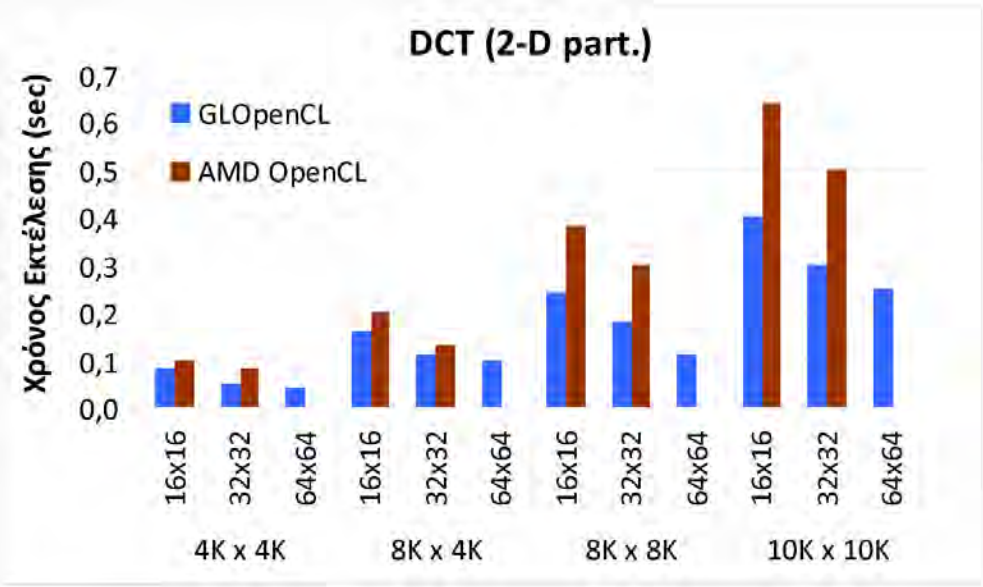
Παράρτημα Α

Γραφήματα Αποτελεσμάτων

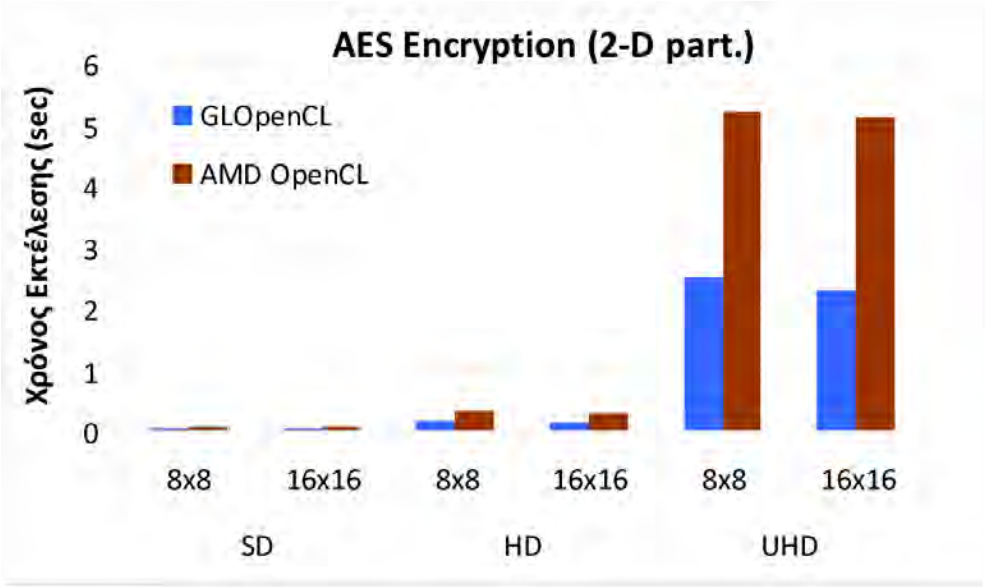
Στο παρόν παράρτημα παραθέτουμε τα γραφήματα με τα αποτελέσματα από την αξιολόγηση της απόδοσης της υποδομής για τις δύο αρχιτεκτονικές. Πρώτα παρουσιάζονται τα γραφήματα με τον χρόνο εκτέλεσης των εφαρμογών στην αρχιτεκτονική Intel x86 και έπειτα ακολουθούν τα γραφήματα με τα αποτελέσματα της αξιολόγησης της υποδομής στον επεξεργαστή Cell B.E.



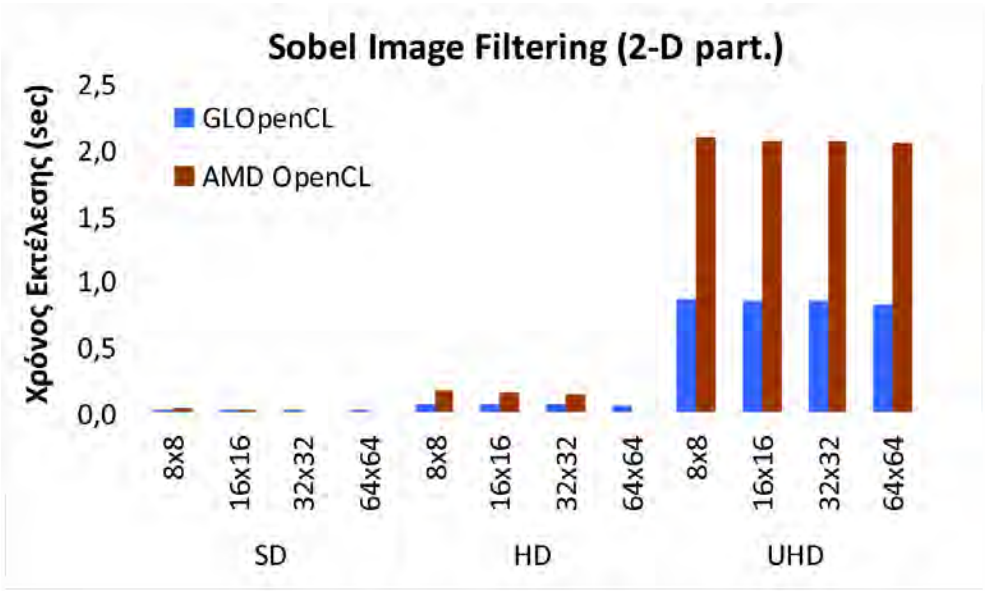
Σχήμα Α.1: Ο χρόνος εκτέλεσης της εφαρμογής Vector Add στον επεξεργαστή Intel E5520.



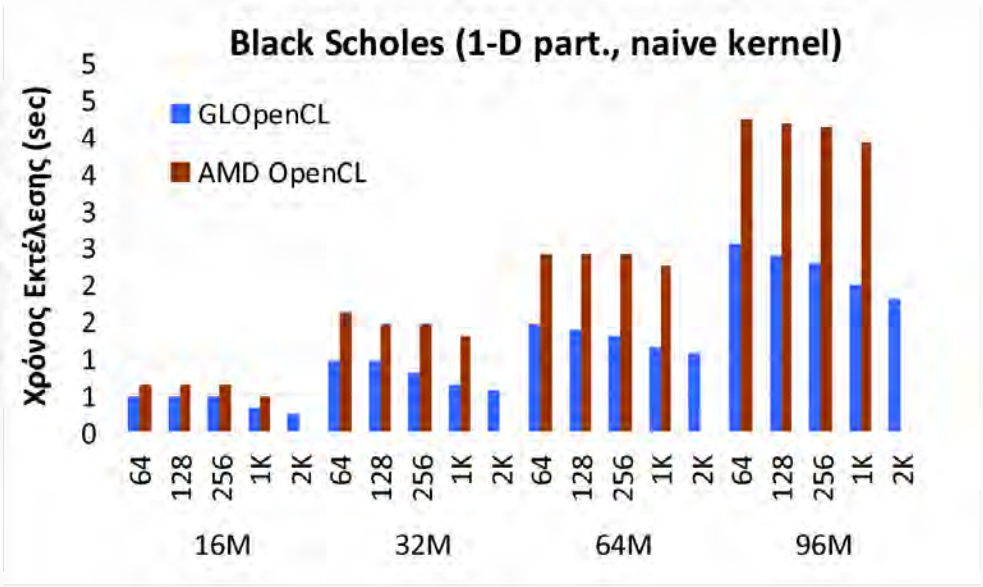
Σχήμα Α.2: Ο χρόνος εκτέλεσης της εφαρμογής 2D DCT στον επεξεργαστή Intel E5520.



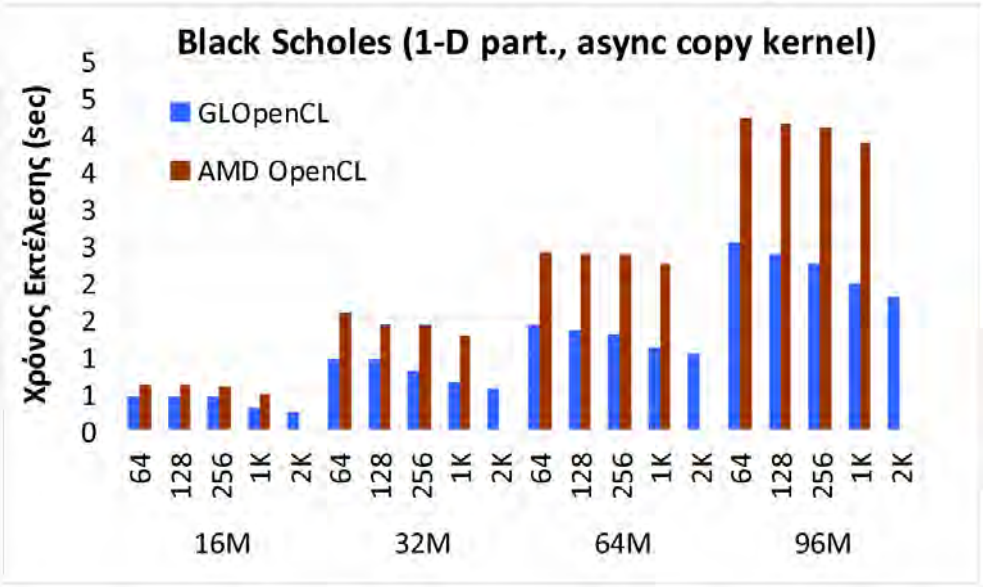
Σχήμα Α.3: Ο χρόνος εκτέλεσης της εφαρμογής AES στον επεξεργαστή Intel E5520.



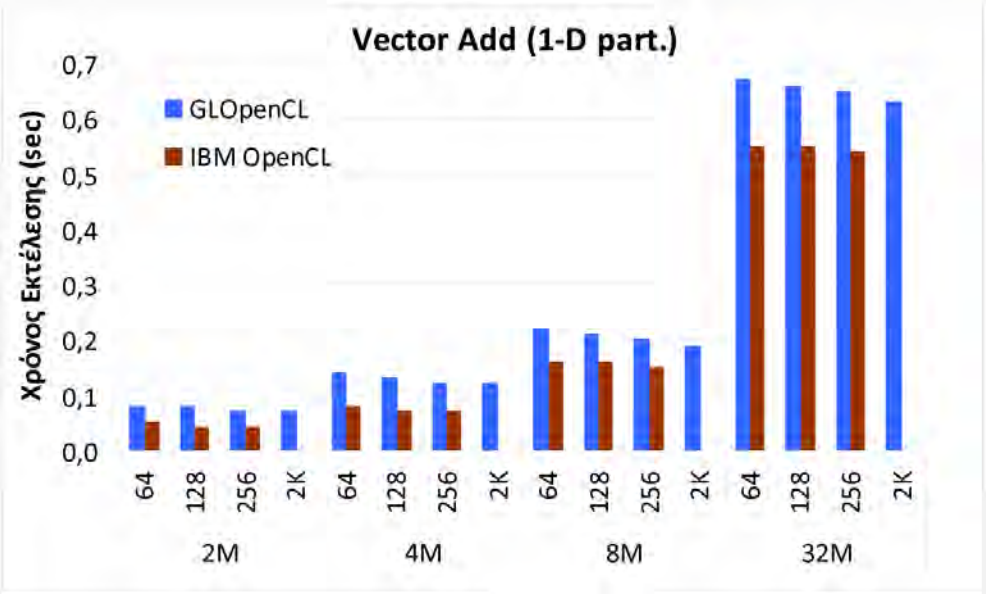
Σχήμα Α.4: Ο χρόνος εκτέλεσης της εφαρμογής Sobel Filter στον επεξεργαστή Intel E5520.



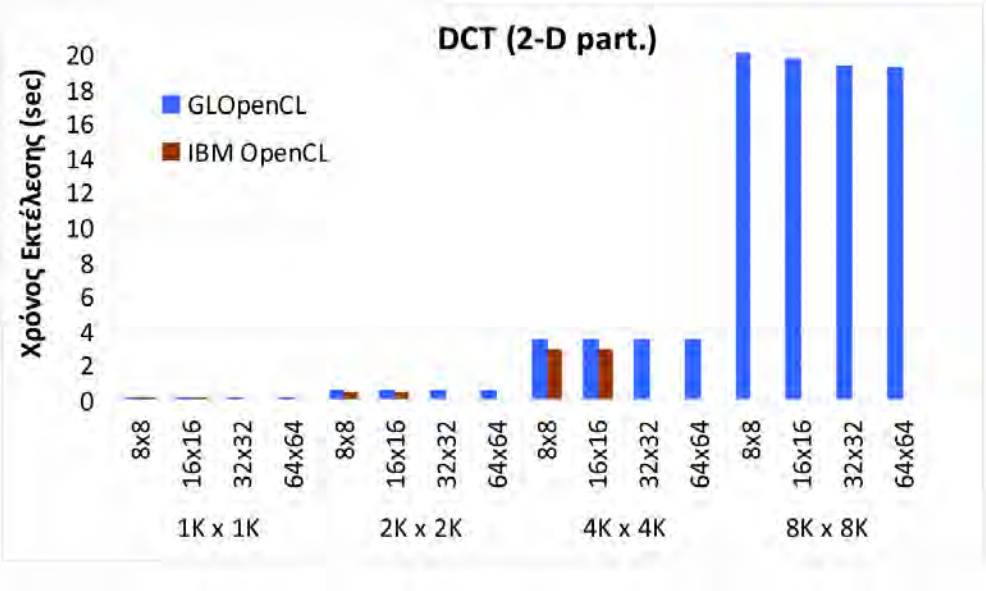
Σχήμα Α.5: Ο χρόνος εκτέλεσης της έκδοσης της εφαρμογής BlackScholes στην οποία δεν χρησιμοποιούνται ασύγχρονες μεταφορές δεδομένων, στον επεξεργαστή Intel E5520.



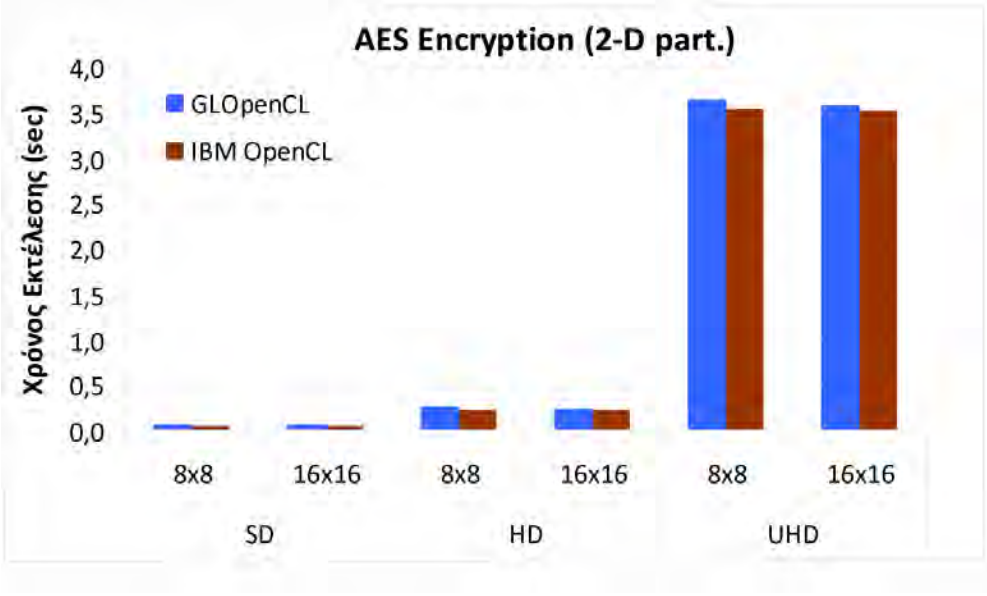
Σχήμα Α.6: Ο χρόνος εκτέλεσης της έκδοσης της εφαρμογής BlackScholes στην οποία χρησιμοποιούνται ασύγχρονες μεταφορές δεδομένων, στον επεξεργαστή Intel E5520.



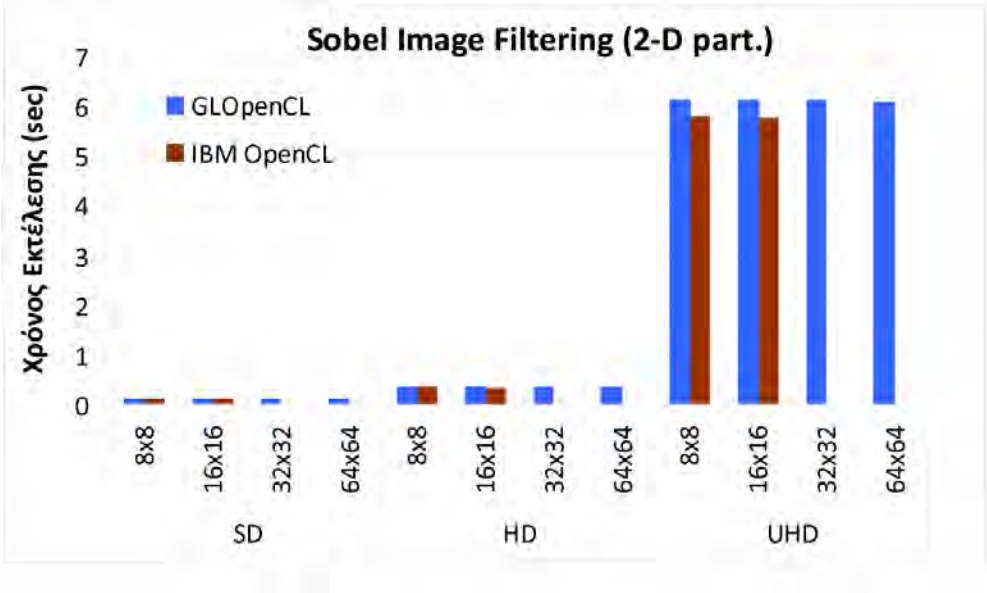
Σχήμα Α.7: Ο χρόνος εκτέλεσης της εφαρμογής Vector Add στον επεξεργαστή Cell B.E.



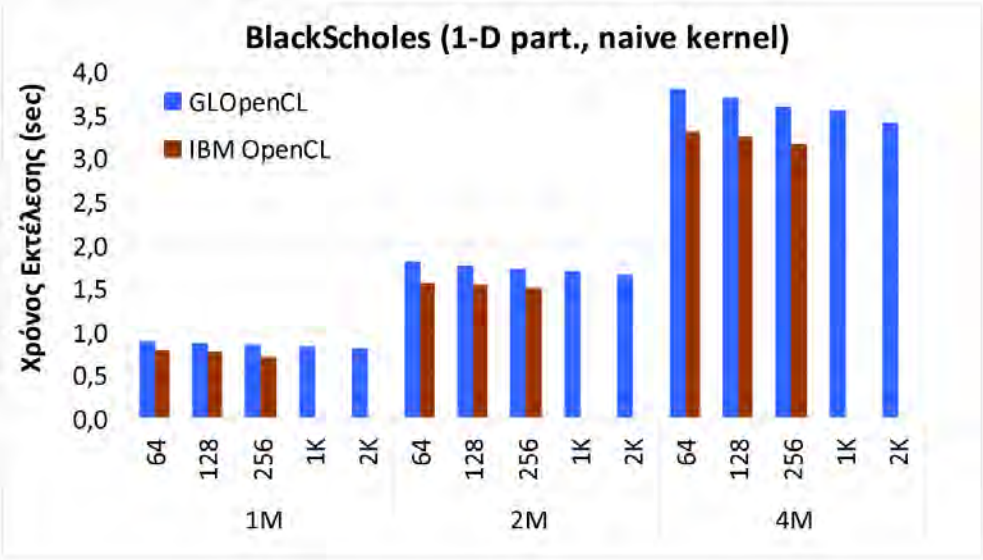
Σχήμα Α.8: Ο χρόνος εκτέλεσης της εφαρμογής 2D DCT στον επεξεργαστή Cell B.E.



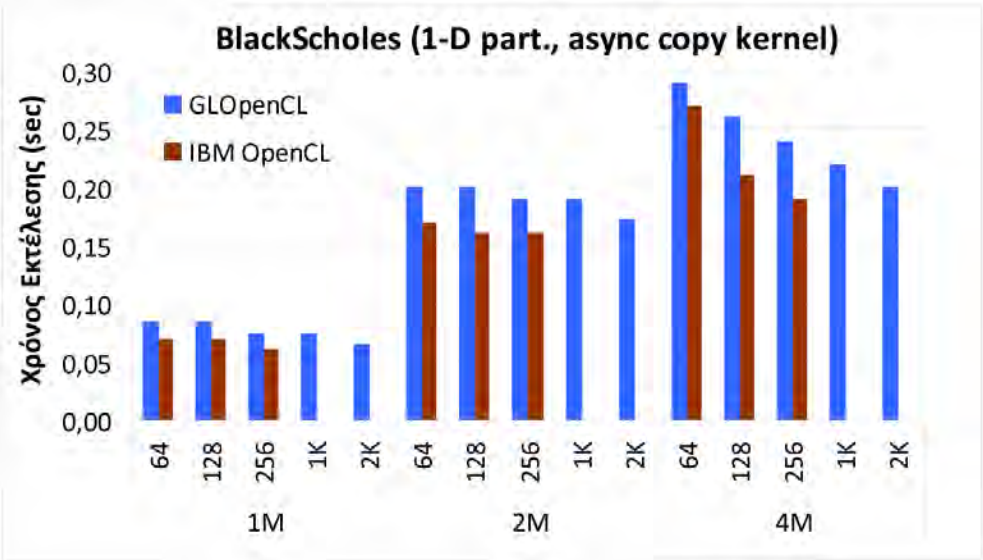
Σχήμα Α.9: Ο χρόνος εκτέλεσης της εφαρμογής AES στον επεξεργαστή Cell B.E.



Σχήμα Α.10: Ο χρόνος εκτέλεσης της εφαρμογής Sobel Filter στον επεξεργαστή Cell B.E.



Σχήμα Α.11: Ο χρόνος εκτέλεσης της έκδοσης της εφαρμογής BlackScholes στην οποία δεν χρησιμοποιούνται ασύγχρονες μεταφορές δεδομένων, στον επεξεργαστή Cell B.E.



Σχήμα Α.12: Ο χρόνος εκτέλεσης της έκδοσης της εφαρμογής BlackScholes στην οποία χρησιμοποιούνται ασύγχρονες μεταφορές δεδομένων, στον επεξεργαστή Cell B.E.

Βιβλιογραφία

[1] R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. Morgan Kaufmann, 2002.

[2] Arevalo, A. et al, *Programming the* C*ell* B*roadband* E*ngine:* E*xamples and* B*est* P*ractices*. International Business Machines Corporation (IBM), October 2008.

[3] ATI-AMD. ATI Stream Software Development Kit (SDK) v2.1. [Online]. Available: http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx

[4] J. Balart, M. Gonzalez, X. Martorell, E. Ayguade, Z. Sura, T. Chen, T. Zhang, K. O'Brien, and K. O'Brien, "A Novel Asynchronous Software Cache Implementation for the Cell-BE Processor," in *Languages and Compilers for Parallel Computing: 20th International Workshop, LCPC 2007, Revised Selected Papers*, 2008, pp. 125--140.

[5] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *The Journal of Political Economy*, vol. 81, no. 3, pp. 637--654, 1973.

[6] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An Efficient Multithreaded Runtime System," *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 55--69, 1996.

[7] R. D. Blumofe and C. E. Leiserson, "Scheduling Multithreaded Computations by Work Stealing," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, 1994, pp. 356--368.

[8] T. Chen, H. Lin, and T. Zhang, "Orchestrating Data Transfer for the Cell/B.E. Processor," in *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, 2008, pp. 289--298.

[9] T. Chen, T. Zhang, Z. Sura, and M. G. Tallada, "Prefetching Irregular References for Software Cache on Cell," in *CGO '08: Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, 2008, pp. 155--164.

[10] Clang: A C Language Family Frontend for LLVM. [Online]. Available: http://clang.llvm.org/

[11] H. Franke and R. Russel, "Fuss, Futexes and Furlocks: Fast User-Space Locking in Linux," in *Proceedings of the Otawa Linux Symposium*, 2002, pp. 85--97.

[12] M. Gonzàlez, N. Vujic, X. Martorell, E. Ayguadé, A. E. Eichenberger, T. Chen, Z. Sura, T. Zhang, K. O'Brien, and K. O'Brien, "Hybrid Access-Specific Software Cache Techniques for the Cell BE Architecture," in *PACT '08: Proceedings of the 17th international conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 292--302.

[13] M. Herlihy, "Wait-Free Synchronization," *ACM Transactions on Programming Languages and Systems*, vol. 13, pp. 124--149, 1993.

[14] H. P. Hofstee, "Power Efficient Processor Architecture and The Cell Processor," *International Symposium on High-Performance Computer Architecture*, vol. 0, pp. 258--262, 2005.

[15] International Business Machines Corporation (IBM). IBM SDK for Multicore Acceleration Version 3.1. [Online]. Available: http://www.ibm.com/developerworks/power/cell/downloads.html

[16] International Business Machines Corporation (IBM). OpenCL Development Kit for Linux on Power. [Online]. Available: http://www.alphaworks.ibm.com/tech/opencl

[17] Khronos OpenCL Working Group and A. Munshi, "The OpenCL Specification Version: 1.0 Document Revision: 48," 2009.

[18] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *CGO '04: Proceedings of the International Symposium on Code Generation and Optimization*, 2004, pp. 75--86.

[19] J. Lee, S. Seo, C. Kim, J. Kim, P. Chun, Z. Sura, J. Kim, and S. Han, "COMIC: A Coherent Shared Memory Interface for Cell BE," in *PACT '08: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 303--314.

[20] F. Mueller, "A Library Implementation of POSIX Threads under Unix," in *Proceedings of the USENIX Conference*, 1993, pp. 29--41.

[21] National Institute of Standards and Technology (NIST), "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," Federal Information Processing Standards Publication 197, November 2001.

[22] NVIDIA. CUDA Technology. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[23] NVIDIA, "CUDA Programming Guide, version 3.0," NVIDIA Corporation, Tech. Rep., 2010.

[24] OpenMP. The OpenMP API. [Online]. Available: http://openmp.org/wp/

[25] J. P. Perez, P. Bellens, R. M. Badia, and J. Labarta, "CellSs: Making it Easier to Program the Cell Broadband Engine Processor," *IBM Journal of Research and Development*, vol. 51, no. 5, pp. 593--604, 2007.

[26] M. Scarpino, *Programming the Cell Processor: For Games, Graphics, and Computation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[27] I. Sobel and G. Feldman, "A 3x3 Isotropic Gradient Operator for Image Processing," Talk presented at the Stanford Artificial Project, 1968.

[28] J. A. Stratton, S. S. Stone, and W.-M. W. Hwu, "MCUDA: An Efficient Implementation of CUDA Kernels for Multi-core CPUs," in *Languages and Compilers for Parallel Computing: 21th International Workshop, LCPC 2008, Revised Selected Papers*, 2008, pp. 16--30.

[29] International Business Machines Corporation (IBM). Online documentation and articles for the Cell Broadband Engine Architecture. [Online]. Available: http://www.ibm.com/developerworks/power/cell/

[30] International Business Machines Corporation (IBM). PowerXCell 8i processor product brief. [Online]. Available: http://www-03.ibm.com/technology/cell/pdf/PowerXCell_PB_7May2008_pub.pdf

[31] International Business Machines Corporation (IBM). The Cell project at IBM Research. [Online]. Available: http://www.research.ibm.com/cell/home.html

[32] International Business Machines Corporation (IBM). IBM online course on Cell. [Online]. Available: http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/topic/com.ibm.iea.cbe/cbe_coverpage.html

[33] International Business Machines Corporation (IBM). IBM Hands-On Tutorial on Cell programming. [Online]. Available: http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/topic/com.ibm.iea.cbe/cbe/1.0/Programming.html

[34] International Business Machines Corporation (IBM), "PowerPC microprocessor family: Vector/SIMD Multimedia Extension Technology programming environments manual, version 2.07c," 2006.

[35] International Business Machines Corporation (IBM), "Cell Broadband Engine Architecture, version 1.02," 2007.

[36] International Business Machines Corporation (IBM), "Cell Broadband Engine Programming Handbook. version 1.1," 2007.

[37] International Business Machines Corporation (IBM), "Software Development Kit for Multicore Acceleration, Programmer's Guide, version 3.0," 2007.

[38] International Business Machines Corporation (IBM), "Software Development Kit for Multicore Acceleration, Programming Tutorial, version 3.0," 2007.

[39] International Business Machines Corporation (IBM), "Synergistic Processor Unit Instruction Set Architecture, version 1.2," 2007.

[40] UPC Consortium, "UPC Language Specifications, v1.2," Lawrence Berkeley National Lab, Tech. Rep. LBNL-59208, 2005.