ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον Αλγοριθμικών Μετασχηματισμών Συνεπεξεργαστών Υλικού με OpenCL και Συσχετισμός Αρχιτεκτονικών FPGA και GPU

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΕΥΘΥΜΙΑΣ ΚΑΖΑΚΟΥ

Επιβλέπων : Γεώργιος Οικονομάκος
Επίκουρος καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον Αλγοριθμικών Μετασχηματισμών Συνεπεξεργαστών Υλικού με OpenCL και Συσχετισμός Αρχιτεκτονικών FPGA και GPU

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΕΥΘΥΜΙΑΣ ΚΑΖΑΚΟΥ

Επιβλέπων : Γεώργιος Οικονομάκος
Επίκουρος καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Μαΐου 2015.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Οικονομάκος	Κιαμάλ Πεκμεστζή	Δημήτριος Σούντρης
Επίκουρος καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2015

(Υπογραφή)

.....

ΚΑΖΑΚΟΥ ΕΥΘΥΜΙΑ

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

Copyright © Ευθυμία Γ. Καζάκου, 2015
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε υπό την επίβλεψη του επίκουρου καθηγητή του Ε.Μ.Π. κυρίου Γεωργίου Οικονομάκου, τον οποίο θα ήθελα να ευχαριστήσω θερμά για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου το παρόν θέμα έρευνας, για την πολύτιμη βοήθειά του αλλά και την καθοδήγησή του καθ' όλη τη διάρκεια της εργασίας αυτής, οποιαδήποτε στιγμή την χρειάστηκα.

Ακόμη θα ήθελα να ευχαριστήσω τον ερευνητή του Ε.Μ.Π. Ευστάθιο Σωτηρίου Ξανθόπουλο για την βοήθειά του στο πειραματικό κομμάτι της διπλωματικής εργασίας και για την όμορφη συνεργασία μας. Ευχαριστώ επίσης τον καθηγητή του Ε.Μ.Π κύριο Κιαμάλ Πεκμεστζή και τον αναπληρωτή καθηγητή του Ε.Μ.Π κύριο Δημήτριο Σούντρη για την εμπιστοσύνη και την θερμή υποδοχή τους στο εργαστήριο.

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αγάπη, τη στήριξη και την εμψύχωση καθόλη τη διάρκεια της προσπάθειας αυτής και συγκεκριμένα τους γονείς μου Γεώργιο και Γεωργία, την αδελφή μου Αλκμήνη και τον θείο μου Παναγιώτη.

Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου και συμφοιτητές μου για όλες τις όμορφες και δημιουργικές στιγμές που ζήσαμε κατά τη διάρκεια των σπουδών μας και ιδιαίτερα τον συμφοιτητή μου Αναστάσιο Διονύσιο Καραγιάννη για την πολύτιμη βοήθειά του και τη στήριξη που μου προσέφερε.

Περίληψη

Το αντικείμενο της παρούσας διπλωματικής είναι η μελέτη και η παραλληλοποίηση του αλγορίθμου της οπισθοπροβολής κάνοντας χρήση της γλώσσας προγραμματισμού OpenCL, καθώς επίσης και η αξιοποίηση των διαφόρων ευκαιριών υλοποίησης που προσφέρονται από το εργαλείο ανάπτυξης Altera SDK για την OpenCL. Οι προτεινόμενες υλοποιήσεις κώδικα εκτελούνται σε CPUs, GPUS και σε έναν FPGA. Η διπλωματική χωρίζεται σε έξι τμήματα.

Στο *πρώτο τμήμα*, παρέχεται το θεωρητικό υπόβαθρο για τις αρχές του παράλληλου προγραμματισμού και των ετερογενών αρχιτεκτονικών. Επιπρόσθετα, δίδεται μια πλήρης περιγραφή του προτύπου OpenCL, επεξηγώντας τα μοντέλα εκτέλεσης, μνήμης, προγραμματισμού και πλατφόρμας της OpenCL.

Στο *δεύτερο τμήμα* παρέχεται μια θεωρητική περιγραφή των συσκευών FPGA και των πλεονεκτημάτων τους καθώς και η επεξήγηση του υλικού τους και η σπουδαιότητα της χρήσης τους για τις OpenCL εφαρμογές.

Το *τρίτο τμήμα* αναλύει το θεωρητικό υπόβαθρο του αλγορίθμου οπισθοπροβολής και φιλτραρισμένης οπισθοπροβολής, που χρησιμοποιείται για την ανακατασκευή ιατρικής εικόνας. Πιο συγκεκριμένα, παρουσιάζεται η τεχνική του αλγορίθμου της οπισθοπροβολής, από την οποία παράγονται δισδιάστατες εικόνες από δεδομένα προβολών, καθώς και η μαθηματική περιγραφή των προβολών μέσω του αλγορίθμου Radon. Επιπρόσθετα, παρέχεται η συνεχής ανάλυση του αλγορίθμου.

Στο *τέταρτο τμήμα*, αναφέρεται ο διακριτός αλγόριθμος οπισθοπροβολής για εφαρμογές ψηφιακών κυκλωμάτων και αναλύονται και οι πέντε συσκευές που χρησιμοποιούνται για την εφαρμογή των προτεινόμενων παράλληλων υλοποιήσεων. Συγκεκριμένα, περιγράφεται το Altera Cyclone V SoC FPGA που χρησιμοποιείται στην πλακέτα Terasic DE1-SoC καθώς και το εργαλείο ανάπτυξης λογισμικού της Altera για OpenCL, το οποίο χρησιμοποιείται για τις υλοποιήσεις. Ακόμα, περιγράφεται η αρχιτεκτονική των δύο χρησιμοποιούμενων CPUs (Intel Core i7, Intel Core i5) και των δύο χρησιμοποιούμενων GPUs (AMD FirePro M5100, NVIDIA GeForce 310M).

Το *πέμπτο τμήμα*, παρουσιάζει τα τέσσερα προτεινόμενα σχήματα κώδικα (Direct, Direct Workgroup, Direct Tiles and Direct Recursive) και τις δυνατότητες εφαρμογής τους που ερευνήθηκαν μαζί με τα πειραματικά αποτελέσματά τους στις πέντε υπολογιστικές συσκευές.

Στο *έκτο τμήμα*, συνοψίζονται τα αποτελέσματα της διπλωματικής εργασίας και παρέχονται στον αναγνώστη συγκρίσεις ανάμεσα στις διαφορετικές υλοποιήσεις κώδικα που παρουσιάζονται, με βάση τους χρόνους εκτέλεσης σε όλες τις υλοποιήσεις για όλα τα μεγέθη προβολών. Πιο συγκεκριμένα, γίνονται συγκρίσεις με βάση την κατανάλωση ενέργειας, τις διαφορετικές υπολογιστικές συσκευές, τον αριθμό των προβολών και τα μεγέθη διαίρεσης και προτείνεται η καλύτερη υλοποίηση με την καλύτερη επιτάχυνση.

Τέλος, το *έβδομο τμήμα* αποτελεί τον επίλογο με τα συμπεράσματα της παρούσης διπλωματικής εργασίας και παρέχει προτάσεις για θέματα μελλοντικής εργασίας.

Λέξεις κλειδιά: Οπισθοπροβολή, OpenCL, πολυπύρηνες αρχιτεκτονικές, FPGAs, GPUs, σύνθεση υψηλού επιπέδου, ιατρικές συσκευές

Abstract

The main objective of this diploma thesis is to study and parallelize the backprojection algorithm using OpenCL, as well as to exploit the different implementation opportunities offered by the Altera SDK for OpenCL. The proposed coding schemes are implemented in CPUs, GPUs and FPGA. The thesis is divided in six parts.

In the *first part*, theoretical background is provided on the principles of parallel programming and heterogeneous architectures. In addition, a full description of the OpenCL standard is given explaining its execution, memory, programming and platform model.

The *second part* provides a theoretical description of the FPGA devices and their advantages and explains the FPGA fabric and its usage and importance in OpenCL implementations

The *third part* analyses the theoretical background of the Backprojection and Filtered Backprojection algorithm used for medical image reconstruction. More specifically, the Backprojection algorithm technique is shown, from which two-dimensional images are generated from projected data, as well as the projections' mathematical representation via Radon transform. Moreover, the continuous analysis of the algorithm in provided.

In the *fourth part*, the discrete Back Projection algorithm for digital circuit implementations is mentioned and also all five devices used for the proposed coding schemes' implementation are analyzed. In particular, the FPGA device Altera Cyclone V SoC used in the Terasic DE1-SoC evaluation board is described as well as the Altera Software Development Kit for OpenCL that was used for the implementations. In addition, the architecture of the two used CPUs (Intel Core i7, Intel Core i5) and the two used GPUs (AMD FirePro M5100, NVIDIA GeForce 310M) are described.

The *fifth part*, presents the four different proposed coding schemes (Direct, Direct Workgroup, Direct Tiles and Direct Recursive) and the implementation opportunities investigated, presented together with their experimental results in the five computing devices.

In the *sixth part*, the results of this thesis are summarized providing the reader with comparisons between the different presented coding schemes based on the execution times of all implementations of all different number of sinograms. More precisely, comparisons are made based on the energy consumption, the different computing devices, the number of sinograms and the divide numbers and the best coding scheme with the best speedup is proposed.

Finally, the *seventh part*, gives the conclusion of the diploma thesis and provides future work proposals.

Keys words: Backprojection, OpenCL, multi-core architectures, FPGAs, GPUs, highlevel synthesis, medical devices

Περιεχόμενα

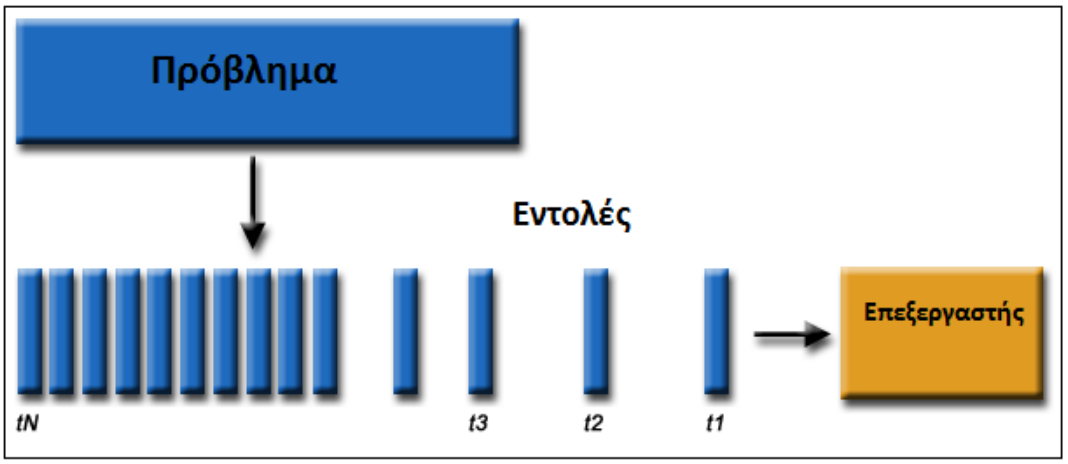
1.Κεφάλαιο για την OpenCL

1.1. Εισαγωγικές έννοιες παράλληλου προγραμματισμού

Τα τελευταία χρόνια οι συσκευές με δυνατότητες παραλληλισμού έχουν αυξηθεί δραματικά τόσο σε αριθμό όσο και σε δυνατότητες. Έτσι, παρουσιάζεται η ανάγκη για κατανόηση των βασικών εννοιών του παράλληλου προγραμματισμού και των δυνατοτήτων αξιοποίησής του από την παρούσα διπλωματική εργασία.

1.1.1. Ορισμός παράλληλου προγραμματισμού

Παραδοσιακά, τα προγράμματα γράφονταν με την λογική της σειριακής επεξεργασίας των εντολών τους. Έτσι, ένα πρόγραμμα μπορούσε να διασπαστεί σε διακριτές εντολές, οι οποίες με τη σειρά τους θα εκτελεστούν η μία μετά την άλλη σε έναν επεξεργαστή, με τον περιορισμό ότι μία μόνο μπορούσε να εκτελεστεί σε κάθε χρονική στιγμή. Κάτι τέτοιο παρουσιάζεται σχηματικά και στο επόμενο σχήμα [1].



Σχήμα 1.1: Σειριακή εκτέλεση εντολών

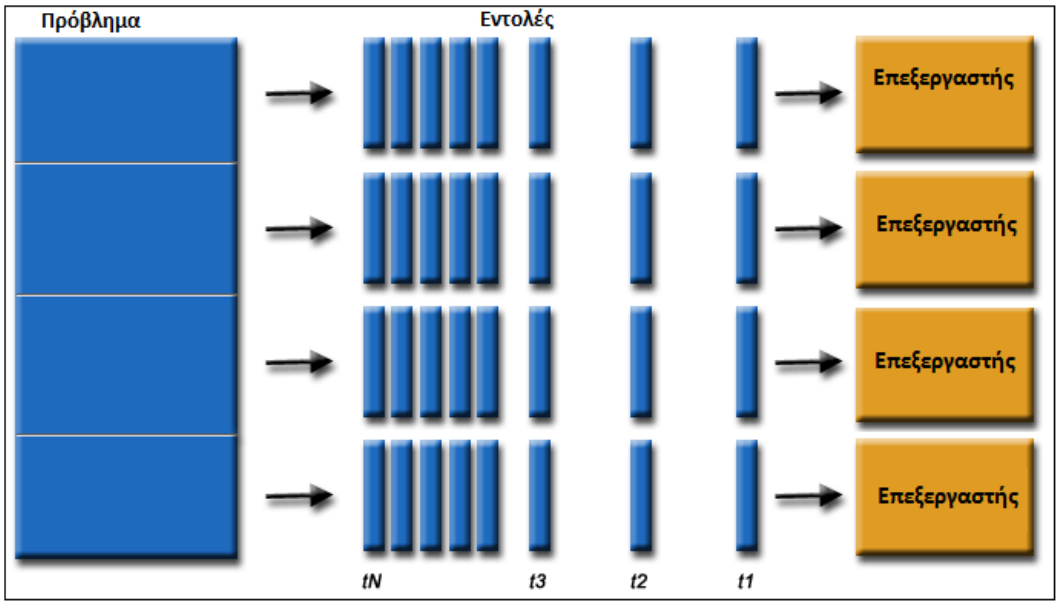
Ο παράλληλος προγραμματισμός μας δίνει την δυνατότητα της ταυτόχρονης χρήσης των πολλαπλών υπολογιστικών πόρων ενός συστήματος για την επίλυση ενός υπολογιστικού προβλήματος. Το πρόβλημα διασπάται σε επιμέρους μικρότερα διακριτά μέρη, τα οποία μπορούν να εκτελεστούν ταυτόχρονα ενώ κάθε μέρος διασπάται περαιτέρω σε επιμέρους εντολές. Οι εντολές του κάθε μέρους μπορούν να εκτελεστούν ταυτόχρονα σε διαφορετικούς επεξεργαστές όπως φαίνεται και στο σχήμα 1.2, ενώ ένας μηχανισμός ελέγχου της συνολικής διαδικασίας είναι απαραίτητος προκειμένου να διασφαλιστεί η σωστή λειτουργία του συστήματος.

Το υπολογιστικό πρόβλημα θα πρέπει να μπορεί να

- διασπαστεί σε μικρότερα τμήματα που να μπορούν να εκτελεστούν ταυτόχρονα.
- εκτελεί πολλαπλές εντολές προγράμματος σε οποιαδήποτε χρονική στιγμή.
- επιλύεται σε λιγότερο χρόνο με πολλαπλούς υπολογιστικούς πόρους σε σχέση με τον ένα επεξεργαστή.

Οι υπολογιστικοί πόροι είναι συνήθως:

- ένας υπολογιστής με πολλαπλούς επεξεργαστές.
- ένας αριθμός τέτοιων υπολογιστών συνδεδεμένων σε ένα δίκτυο.



Σχήμα 1.2: Παράλληλη εκτέλεση εντολών

Το μέγεθος του παραλληλισμού που μπορεί να επιτευχθεί είναι άμεσα συσχετιζόμενο με τη φύση του προβλήματος και του αλγορίθμου επίλυσής του, αλλά εναπόκειται και στον προγραμματιστή προκειμένου να προσδιορίσει και να αξιοποιήσει τις διάφορες μορφές παραλληλισμού στο εκάστοτε πρόβλημα.

Στο κομμάτι που αφορά την υψηλή απόδοση των υπολογισμών (high-performance computing), χρησιμοποιούνται οι κλασσικές προσεγγίσεις για να επιταχύνουν τον υπολογισμό, όταν υπάρχουν πολλαπλές υπολογιστικές πηγές. Κάποιες από αυτές τις προσεγγίσεις είναι η μέθοδος «διαίρει και βασίλευε» (divide-and-conquer) και η μέθοδος «διασκόρπισε και συγκέντρωσε» (scatter-and-gather), οι οποίες απλοποιούν το πρόβλημα και δίνουν στον προγραμματιστή ένα σύνολο στρατηγικών ώστε να εκμεταλλευτεί αποδοτικά τις διαθέσιμες πηγές παραλληλισμού του συστήματος.

Η μέθοδος «διαίρει και βασίλευε» διαιρεί συνεχόμενα το πρόβλημα σε υποπροβλήματα έως ότου τα υποπροβλήματα να ταιριάζουν σε πλήθος στους διαθέσιμους υπολογιστικούς πόρους. Η μέθοδος «διασκόρπισε και συγκέντρωσε» στέλνει σε κάθε

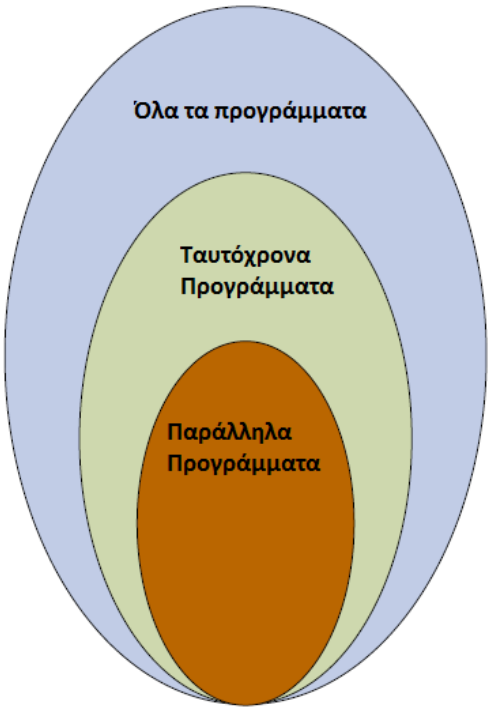
παράλληλη πηγή ένα υποσύνολο των δεδομένων εισόδου και στη συνέχεια συγκεντρώνει τα αποτελέσματα των υπολογισμών, συνδυάζοντάς τα σε ένα σύνολο δεδομένων εξόδου.

Όπως αναφέρθηκε και προηγουμένως, ο διαχωρισμός εξαρτάται από το μέγεθος των υποσυνόλων και τις δυνατότητες των υπολογιστικών πόρων.

Ο ταυτοχρονισμός και ο παραλληλισμός είναι δύο έννοιες που δεν θα πρέπει να συγχέονται. Ο ταυτοχρονισμός συμβαίνει όταν δύο ή περισσότερες εργασίες υλοποιούνται την ίδια χρονική στιγμή. Όταν αναφέρεται ο όρος ταυτοχρονισμός στον προγραμματισμό, εννοείται ότι σε ένα σύστημα με έναν επεξεργαστή εκτελούνται πολλαπλές εργασίες ανεξάρτητα η μια από την άλλη.

Παρόλα αυτά, είναι πιθανό οι ταυτόχρονες εργασίες να εκτελεστούν την ίδια χρονική στιγμή αν υπάρχουν οι διαθέσιμοι υπολογιστικοί πόροι, κάτι όμως που δεν είναι απαραίτητο. Στην περίπτωση του ενός επεξεργαστή, το λειτουργικό σύστημα εναλλάσσει τις εργασίες επιτρέποντας και στις δύο να εκτελεστούν στον πυρήνα.

Ο παραλληλισμός επιτυγχάνεται όταν δύο ή περισσότερες εργασίες εκτελούνται παράλληλα, με στόχο την αύξηση της συνολικής απόδοσης. Τα παράλληλα προγράμματα πρέπει να εκτελούνται ταυτόχρονα. Τα ταυτόχρονα όμως προγράμματα δεν είναι απαραίτητο να είναι παράλληλα. Κάποιες φορές, παρότι τα ταυτόχρονα προγράμματα μπορούν να εκτελεστούν παράλληλα, υπάρχουν κάποιες εξαρτήσεις μεταξύ των εντολών τους, που εμποδίζουν την παράλληλη εκτέλεση. Σαν αποτέλεσμα, μόνο ένα υποσύνολο των ταυτόχρονων προγραμμάτων είναι παραλληλίσιμο [2]. Το σχήμα 1.3 παρουσιάζει αυτή την σχέση μεταξύ των διαφόρων τύπων προγραμμάτων.



Σχήμα 1.3: Παράλληλα και ταυτόχρονα προγράμματα ως υποσύνολα του συνόλου των προγραμμάτων.

1.1.2. Συγχρονισμός

Στην επιστήμη των υπολογιστών, ο συγχρονισμός αναφέρεται σε ένα ή δύο ξεχωριστά αλλά συσχετιζόμενα πλαίσια: στο συγχρονισμό των διαδικασιών (synchronization of processes) και στο συγχρονισμό των δεδομένων (synchronization of data).

Ο συγχρονισμός των διαδικασιών αναφέρεται στην ιδέα ότι πολλαπλές διαδικασίες πρέπει να «βρεθούν» σε ένα συγκεκριμένο σημείο ώστε να φτάσουν σε μια συμφωνία ή να εκτελέσουν μια συγκεκριμένη σειρά ενεργειών. Ο συγχρονισμός δεδομένων αναφέρεται στην ιδέα ότι κρατούνται πολλαπλά αντίγραφα ενός συνόλου δεδομένων σε συγχρονισμό μεταξύ τους. Οι θεμελιώδεις αρχές συγχρονισμού των διαδικασιών χρησιμοποιούνται για να εφαρμοστεί ο συγχρονισμός δεδομένων. Ο τρόπος διαχείρισης μιας σειράς εργασιών που εκτελούνται είναι ένα πολύ βασικό ζήτημα στα περισσότερα παράλληλα προγράμματα. Πολλές φορές μπορεί να αποτελέσει ένα σημαντικό παράγοντα για την βελτίωση (ή μη) της απόδοσης του προγράμματος.

Οι τύποι συγχρονισμού είναι :

- Όρια (Barriers): Συνήθως υπονοείται ότι όλες οι εργασίες εμπλέκονται σ ’αυτόν τον τύπο συγχρονισμού. Η κάθε εργασία εκτελείται μέχρι να φτάσει σε προκαθορισμένο όριο. Μετά σταματάει ή αλλιώς μπλοκάρεται. Όταν και η τελευταία εργασία φτάσει στο δοσμένο όριο, όλες οι εργασίες έχουν συγχρονιστεί. Το τι μπορεί να εκτελεστεί από το σημείο αυτό και μετά ποικίλει. Συχνά, σε αυτό το σημείο εκτελείται κάποια σειριακή εργασία. Σε άλλες περιπτώσεις, οι εργασίες απελευθερώνονται αυτόματα ώστε να συνεχίσουν την εκτέλεσή τους.

- Κλειδώματα/Σημαφόροι (Locks/Semaphores): Σε αυτόν τον τύπο συγχρονισμού μπορούν να εμπλακούν όσες εργασίες επιθυμούν. Συνήθως οι σημαφόροι χρησιμοποιούνται προκειμένου να σειριοποιήσουν (προστατεύσουν) την πρόσβαση στην γενική μνήμη ή σε κάποιο τμήμα του κώδικα. Μόνο μια εργασία σε κάθε χρονική στιγμή μπορεί να χρησιμοποιήσει τον σημαφόρο. Η πρώτη εργασία που αποκτά πρόσβαση στο κλείδωμα το «ενεργοποιεί». Η εργασία αυτή μπορεί στη συνέχεια με ασφάλεια να προσεγγίσει τα προστατευμένα δεδομένα ή τον κώδικα. Οι υπόλοιπες εργασίες πρέπει να περιμένουν μέχρις ότου η εργασία που έχει θέσει το κλείδωμα το απελευθερώσει. Τα κλειδώματα μπορούν να είναι μπλοκαρισμένα ή όχι (blocking or non-blocking).

- Σύγχρονη επικοινωνία: Περιλαμβάνει μόνο εκείνες τις εργασίες που εκτελούν μια «διαδικασία επικοινωνίας». Όταν μια εργασία εκτελεί μια διαδικασία επικοινωνίας, χρειάζεται μια μορφή συντονισμού για τις υπόλοιπες εργασίες που συμμετέχουν στην επικοινωνία. Για παράδειγμα, μια εργασία πριν την εκτέλεσή της, πρέπει πρώτα να δεχτεί ένα αναγνωριστικό από την εργασία «αποστολέα», ότι είναι όλα είναι εντάξει προκειμένου να εκτελεστεί.

1.1.3. Ετερογενή υπολογιστικά συστήματα

Τα τελευταία χρόνια, τα διάφορα υπολογιστικά περιβάλλοντα έχουν εξελιχθεί σε πολύπλευρα συστήματα, εκμεταλλευόμενα τις δυνατότητες όλων των διαθέσιμων υπολογιστικών πόρων όπως είναι οι πολυπύρηνοι μικροεπεξεργαστές, οι κεντρικές μονάδες επεξεργασίας (CPU), οι επαναπρογραμματιζόμενες συσκευές (FPGAs) αλλά και οι κάρτες γραφικών (GPU).

Στα πλαίσια του ετερογενούς προγραμματισμού, η διαδικασία παραγωγής αξιόπιστου κώδικα για τέτοιου είδους αρχιτεκτονικές, παραθέτει ένα μεγάλο αριθμό προκλήσεων στην κοινότητα του προγραμματισμού [2].

Αρχικά πρέπει να αναφερθεί πως η κάθε εφαρμογή υποδεικνύει διαφορετική συμπεριφορά ανάλογα με το φόρτο εργασίας που της ανατίθεται. Έτσι για παράδειγμα, οι εφαρμογές μπορούν να διακριθούν σε διάφορες κατηγορίες ως εξής:

- Εντατικού ελέγχου: όπως είναι για παράδειγμα οι αλγόριθμοι αναζήτησης και ταξινόμησης.
- Δεδομένων: όπως είναι η επεξεργασία εικόνας, η αναπαράσταση και μοντελοποίηση δεδομένων.
- Επεξεργασίας: όπως είναι αριθμητικές μέθοδοι, χρηματοοικονομικές μέθοδοι και υπολογιστικές μέθοδοι.

Κάθε μια απ' αυτές τις εφαρμογές συνήθως εκτελείται πιο αποδοτικά σε ένα συγκεκριμένο είδος αρχιτεκτονικής υλικού (hardware). Με λίγα λόγια, δεν υπάρχει ένα είδος αρχιτεκτονικής που να είναι το ιδανικό για όλες αυτές τις κατηγορίες εφαρμογών. Τις περισσότερες μάλιστα φορές το βέλτιστο είναι ένας συνδυασμός χαρακτηριστικών από διάφορες αρχιτεκτονικές ώστε να επιτευχθεί η μέγιστη απόδοση. Τα τελευταία χρόνια, οι συσκευές που υποστηρίζουν παράλληλο προγραμματισμό έχουν αυξηθεί και έχουν αυξήσει ταυτόχρονα και τις επεξεργαστικές τους δυνατότητες. Συγκεκριμένα, με την είσοδο και αξιοποίηση των καρτών γραφικών (GPU) στο χώρο του υπολογισμού, δημιουργήθηκαν νέες προοπτικές για υπολογιστική δύναμη με μικρό κόστος.

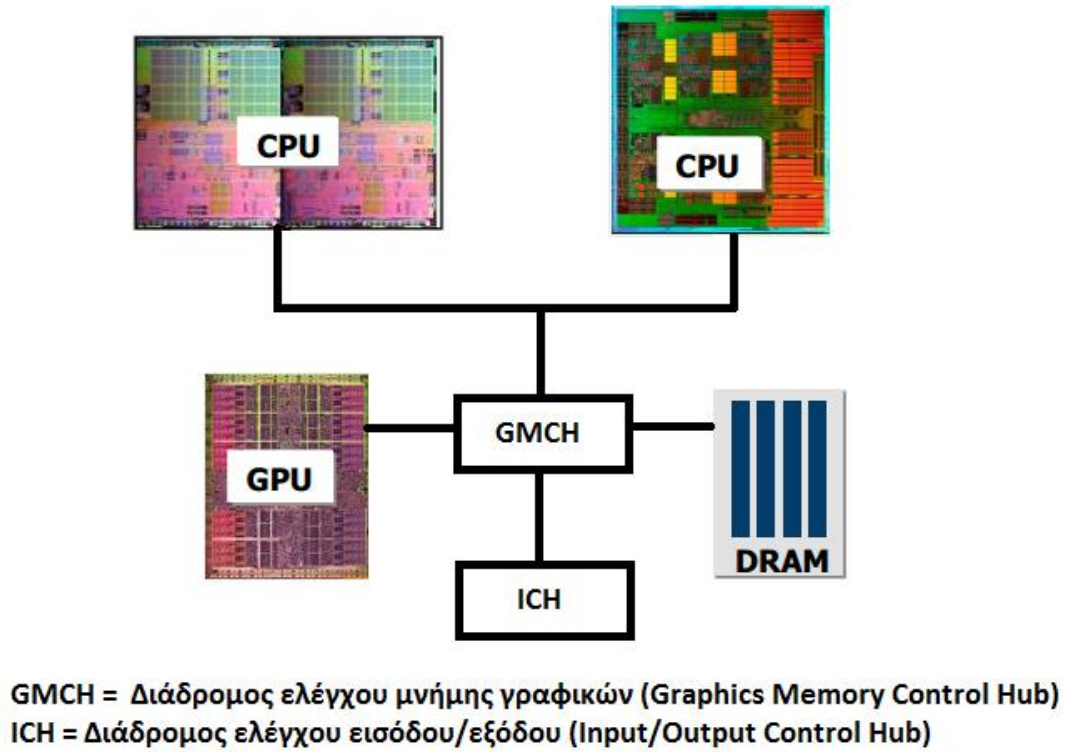
Έτσι οι κάρτες γραφικών εξελίχθηκαν σε πλήρως προγραμματιζόμενες και χρησιμοποιούνται σε δύο είδη αρχιτεκτονικών:

1. Τις αρχιτεκτονικές που βασίζονται στο μοντέλο διαχωρισμού έργων (Task-based architecture): Οι αρχιτεκτονικές που είναι βασισμένες στο μοντέλο διαχωρισμού έργων, εστιάζουν στις διαδικασίες ή αλλιώς στα νήματα εκτέλεσης του προγράμματος (threads). Κάθε επεξεργαστής καλείται να εκτελέσει μια διαφορετική διαδικασία στα ίδια ή σε διαφορετικά δεδομένα. Οι διαδικασίες αυτές εκτελούν τον ίδιο ή και διαφορετικό κώδικα. Η επικοινωνία γίνεται περνώντας δεδομένα από το ένα νήμα στο άλλο, γεγονός που συχνά στρέφει την προσοχή του προγραμματιστή στην ανάγκη για επικοινωνία μεταξύ των νημάτων.

2. Τις αρχιτεκτονικές που βασίζονται στο διαχωρισμό δεδομένων (Data-based architecture): Οι αρχιτεκτονικές που είναι βασισμένες στο μοντέλο διαχωρισμού δεδομένων εστιάζουν στην εκτέλεση εργασιών σ' ένα σύνολο δεδομένων (συνήθως πίνακα). Ένα σύνολο έργων θα επεξεργαστεί αυτά τα δεδομένα, αφού προηγουμένως τα δεδομένα διαχωριστούν σε κατάλληλα υποσύνολα δεδομένων. Έτσι, το ίδιο έργο εκτελείται ξεχωριστά σε κάθε υποσύνολο δεδομένων.

Οι κατασκευαστές συνδυάζουν πλέον στο ίδιο σύστημα τις κάρτες γραφικών με τις κεντρικές μονάδες επεξεργασίας, εγκαινιάζοντας μια νέα γενιά συστημάτων ετερογενούς προγραμματισμού.

Ως ετερογενής προγραμματισμός, αναφέρεται το σύστημα εκείνο που χρησιμοποιεί παραπάνω από ένα είδος επεξεργαστή [3]. Τα συστήματα αυτά υπερέχουν σε απόδοση όχι μόνο στην περίπτωση που χρησιμοποιηθούν όμοιοι επεξεργαστές αλλά και ανόμοιοι, συνήθως χρησιμοποιώντας συγκεκριμένες επεξεργαστικές δυνατότητες για τη διαχείριση συγκεκριμένων έργων. Ένα παράδειγμα ετερογενούς συστήματος παρουσιάζεται και στο παρακάτω σχήμα.



Σχήμα 1.4: Παράδειγμα συστήματος ετερογενούς αρχιτεκτονικής

Με βάση τα παραπάνω, γίνεται διαχωρισμός των διεργασιών ανάλογα με το αν ανήκουν στο μοντέλο διαχωρισμού έργων ή στο μοντέλο διαχωρισμού δεδομένων και η κατηγορία αυτή ανατίθεται στους πυρήνες (kernels) που εκτελούνται στις κάρτες γραφικώ, ενώ τα υπόλοιπα έργα (non kernel) εκτελούνται στην κεντρική μονάδα επεξεργασίας.

1.1.4. Πλεονεκτήματα παράλληλου προγραμματισμού

Βάσει των παραπάνω, γίνεται φανερό ότι ο παράλληλος προγραμματισμός προσφέρει πολλά πλεονεκτήματα στο χώρο της επιστήμης των υπολογιστών, μερικά από τα οποία παρουσιάζονται παρακάτω.

Ο παράλληλος προγραμματισμός:

- Μειώνει χρόνο και κόστος: Στη θεωρία, η διάθεση πολλών πόρων στην εκτέλεση μιας εργασίας μπορεί να ελαττώσει το χρόνο ολοκλήρωσής της και με τον τρόπο αυτό να επιτευχθεί μείωση κόστους. Οι παράλληλες συστάδες (clusters) μπορούν να κατασκευαστούν από φτηνά και ευρέως διαδεδομένα υλικά.

- Παρέχει ταυτοχρονισμό: Ένας υπολογιστικός πόρος μπορεί να εκτελέσει μόνο μία εργασία σε κάθε χρονική στιγμή. Επομένως, πολλαπλοί υπολογιστικοί πόροι παρέχουν την δυνατότητα ταυτόχρονης εκτέλεσης πολλαπλών εργασιών.

- Αξιοποιεί και μη τοπικούς πόρους (non-local resources): Υπάρχει η δυνατότητα χρήσης υπολογιστικών πόρων σε μια ευρεία περιοχή δικτύου ή στο διαδίκτυο, όταν οι τοπικοί υπολογιστικοί πόροι είναι ανεπαρκείς.

Επιπρόσθετα, τόσο φυσικοί όσο και πρακτικοί λόγοι θέτουν σημαντικούς περιορισμούς στην περαιτέρω ανάπτυξη ακόμα ταχύτερων σειριακών υπολογιστών. Αυτοί είναι οι εξής:

- Επιδόσεις μετάδοσης: Η ταχύτητα ενός σειριακού υπολογιστή είναι άμεσα συνδεδεμένη με την ταχύτητα μεταφοράς των δεδομένων μέσω του διαθέσιμου υλικού. Η αύξηση των ταχυτήτων μεταφοράς απαιτεί αύξηση της εγγύτητας των επεξεργαστικών στοιχείων (μονάδων).
- Όρια στη σμίκρυνση : Η προηγμένη τεχνολογία των επεξεργαστών επιτρέπει σήμερα την τοποθέτηση ενός αυξανόμενου αριθμού από transistors στο chip. Παρόλα αυτά, θα υπάρξει στο μέλλον κάποιο φυσικό όριο στην περαιτέρω σμίκρυνση των στοιχείων.
- Οικονομικοί περιορισμοί: Από τα προηγούμενα συμπεραίνεται ότι καθίσταται ακριβή η υλοποίηση της αύξησης της ταχύτητας ενός επεξεργαστή. Η παράλληλη όμως χρήση ενός μεγάλου αριθμού διαθέσιμων επεξεργαστών για την επίτευξη της ίδιας (ή και ακόμα καλύτερης) απόδοσης ενός υπολογιστικού συστήματος, θα είναι οπωσδήποτε πιο οικονομική.

Η παράλληλη επεξεργασία είναι πολύ πιο γρήγορη από την σειριακή επεξεργασία όταν αυτή αφορά υπολογισμούς σε μεγάλους όγκους δεδομένων. Αυτό συμβαίνει διότι ένας παράλληλος επεξεργαστής έχει την δυνατότητα να εκτελέσει «πολυνηματικές» λειτουργίες (multithreading) σε μεγάλη κλίμακα και έτσι μπορεί να επεξεργαστεί ταυτόχρονα μεγάλους όγκους δεδομένων.

Συμπερασματικά, το βασικότερο πλεονέκτημα του παράλληλου προγραμματισμού είναι ότι είναι πολύ πιο γρήγορος για απλούς συνεχόμενους υπολογισμούς που επεξεργάζονται μεγάλους όγκους δεδομένων.

1.2. Περιγραφή της OpenCL

Η OpenCL ή αλλιώς Open Computing Language, είναι ένα πλαίσιο (framework) ετερογενούς προγραμματισμού που ανήκει στη μη κερδοσκοπική κοινοπραξία Khronos Group. Πρόκειται για ένα πλαίσιο που εκτελεί εφαρμογές, χρησιμοποιώντας διάφορους τύπους συσκευών ακόμα και από διαφορετικές εταιρείες κατασκευαστών.

Επιπλέον, υποστηρίζει ένα μεγάλο εύρος επιπέδων παραλληλισμού και ταιριάζει αποδοτικά τόσο σε ομογενές ή ετερογενές σύστημα, μονό ή πολλαπλό σύστημα συσκευών συμπεριλαμβανομένων των καρτών γραφικών (GPU), των κεντρικών μονάδων επεξεργασίας (CPU), των ψηφιακών επεξεργαστών σήματος (DSP), των επαναπρογραμματιζόμενων επεξεργαστών (FPGAs) και άλλων επεξεργαστών.

Η OpenCL προσφέρει μια γλώσσα που απευθύνεται τόσο σε συσκευές (kernels), όσο και στον τρόπο χειρισμού των συσκευών από την μεριά του οικοδεσπότη επεξεργαστή (host). Η κάθε μία από τις υπολογιστικές συσκευές αποτελείται από πολλές ανεξάρτητες επεξεργαστικές μονάδες (processing elements -PEs) και ο κάθε πυρήνας (kernel) που εκτελείται, ουσιαστικά εκτελείται σε όλες ή σε μερικές από τις επεξεργαστικές μονάδες αυτές. Η δομή των υπολογιστικών συσκευών θα αναλυθεί παρακάτω και με μεγαλύτερη λεπτομέρεια.

Η γλώσσα που απευθύνεται στις συσκευές είναι σχεδιασμένη ώστε να ταιριάξει με ένα ευρύ φάσμα από συστήματα μνήμης και είναι βασισμένη στην C99. Από την άλλη, η γλώσσα του οικοδεσπότη επεξεργαστή (host) στοχεύει στην υποστήριξη και γεφύρωση περίπλοκων ταυτόχρονων προγραμμάτων με χαμηλή επιβάρυνση (overhead).

Έτσι, αυτές οι δύο γλώσσες μαζί προσφέρουν στον προγραμματιστή τη δυνατότητα εύκολης μετάβασης από τον αλγόριθμο στην υλοποίηση. Η OpenCL παρέχει παράλληλους υπολογισμούς σε επίπεδο διαχωρισμού έργων (task-based parallelism) ή διαχωρισμού δεδομένων (data-based parallelism), όπως αυτά αναλύθηκαν στην προηγούμενη ενότητα.

Κλείνοντας το εισαγωγικό κομμάτι περιγραφής της OpenCL, αξίζει να αναφερθεί ότι για τον σκοπό της διπλωματικής αυτής επιλέχθηκε η συγκεκριμένη γλώσσα γιατί προσφέρει τα πλεονεκτήματα της σε μεγάλο βαθμό μεταφερσιμότητας(portability), του παράλληλου προγραμματισμού (parallel programming) αλλά και της τυποποιημένης επεξεργασίας διανυσμάτων (standardized vector processing).

1.2.1. Προφίλ της OpenCL

Η ανάλυση της OpenCL βασίζεται σε τέσσερα τμήματα που ονομάζονται μοντέλα (models) [2] και επιγραμματικά ομαδοποιούνται ως εξής:

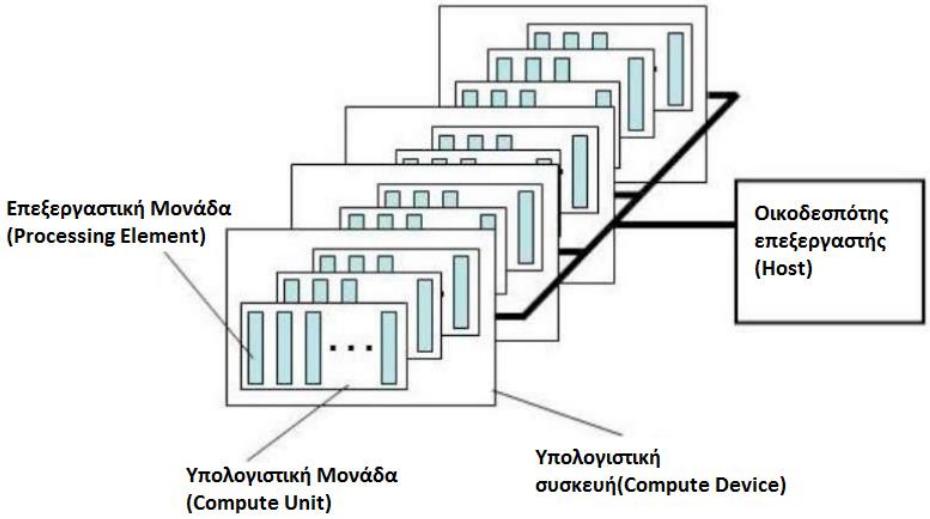
1. Μοντέλο πλατφόρμας (Platform model): Το μοντέλο πλατφόρμας υποδεικνύει ότι ένας επεξεργαστής διευθύνει την εκτέλεση (οικοδεσπότης επεξεργαστής-host) και ένας ή περισσότεροι επεξεργαστές εκτελούν τον OpenCL C κώδικα (συσκευές-devices).
2. Μοντέλο εκτέλεσης (Execution model): Το μοντέλο εκτέλεσης καθορίζει πώς το περιβάλλον της OpenCL ρυθμίζεται στον οικοδεσπότη επεξεργαστή και πώς οι πυρήνες εκτελούνται στις συσκευές. Αυτό περιλαμβάνει το στήσιμο ενός γενικού πλαισίου στην OpenCL (context) στον οικοδεσπότη επεξεργαστή, δίνοντας μηχανισμούς για αλληλεπίδραση οικοδεσπότη-συσκευής και καθορίζει ένα μοντέλο συγχρονισμού χρησιμοποιούμενο από τον πυρήνα για εκτέλεση στις συσκευές.
3. Μοντέλο μνήμης (Memory model): Το μοντέλο μνήμης καθορίζει το αφηρημένο μοντέλο ιεραρχίας μνήμης που χρησιμοποιούν οι πυρήνες, ανεξάρτητα από το πραγματικό μοντέλο μνήμης που χρησιμοποιείται σε κάθε συγκεκριμένη αρχιτεκτονική. Το μοντέλο αυτό θυμίζει αρκετά το μοντέλο μνήμης της κάρτας γραφικών.
4. Μοντέλο προγραμματισμού (Programming model): Το μοντέλο προγραμματισμού καθορίζει πώς το μοντέλο συγχρονισμού αντιστοιχίζεται σε φυσικό υλικό.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά τα παραπάνω μοντέλα.

1.2.1.1. Μοντέλο Πλατφόρμας

Το μοντέλο πλατφόρμας της OpenCL καθορίζει τον ρόλο του οικοδεσπότη επεξεργαστή και των συσκευών και παρουσιάζει ένα αφηρημένο μοντέλο υλικού για τις συσκευές. Παρατηρώντας και το σχήμα που παρατίθεται παρακάτω, συμπεραίνεται πως ένας μόνο επεξεργαστής διευθύνει την εκτέλεση σε μία ή περισσότερες συσκευές.

Η κάθε συσκευή με τη σειρά της, αποτελείται από μια συστοιχία υπολογιστικών μονάδων, κάθε μια από τις οποίες διαχωρίζεται περαιτέρω σε μικρότερες επεξεργαστικές μονάδες. Οι εντολές του οικοδεσπότη προς τις συσκευές αφορούν την μεταφορά δεδομένων από τη μνήμη του οικοδεσπότη προς τις συσκευές, καθώς και την εκτέλεση προγραμμάτων στις συσκευές. Η γλώσσα που χρησιμοποιείται από τον οικοδεσπότη είναι η C ή η C++, ενώ από τις συσκευές είναι η OpenCL C, όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου [4].



Σχήμα 1.5 : Μοντέλο πλατφόρμας OpenCL

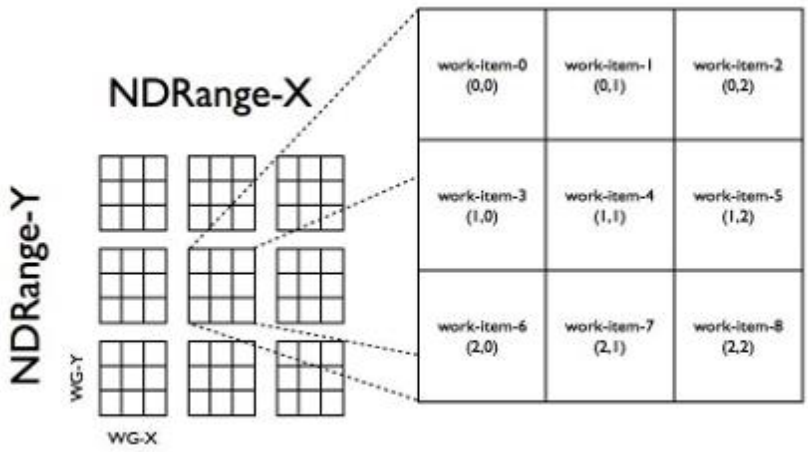
1.2.1.2. Μοντέλο Εκτέλεσης

Σύμφωνα με το μοντέλο εκτέλεσης, η βασική μονάδα εκτέλεσης προγραμμάτων στην OpenCL είναι το στοιχείο εργασίας ή αλλιώς work-item. Το καθένα από αυτά τα work-items εκτελεί το σώμα του πυρήνα (kernel) και αν υποτεθεί ότι στον σειριακό κώδικα υπήρχε ένας βρόχος εκτέλεσης εντολών, στην ταυτόχρονη εκτέλεση το καθένα από τα work-items αποτελεί μια από τις επαναλήψεις του βρόχου αυτού. Το περιβάλλον εκτέλεσης (runtime) της OpenCL δημιουργεί τόσα work-items όσα στοιχεία έχει και ο κάθε πίνακας που χρησιμοποιείται στην εκτέλεση του προγράμματος.

Όταν ένας πυρήνας σταλεί για εκτέλεση, καθορίζεται ένας δείκτης ούτως ώστε το κάθε work-item να αποκτήσει υπόσταση και να εκτελεστεί. Το κάθε work-item θα αναγνωρίζεται από το δικό του μοναδικό χαρακτηριστικό αριθμό (global ID) και θα εκτελέσει τον κώδικα που υπάρχει στον πυρήνα. Τα work-items χωρίζονται σε ομάδες και κάθε ομάδα λαμβάνει ένα δικό της χαρακτηριστικό αριθμό (ID), γνωστό ως χαρακτηριστικό αριθμό ομάδας (work-group ID) και η κάθε μια από αυτές τις ομάδες εκτελείται συγχρόνως στις επεξεργαστικές μονάδες από τις οποίες αποτελείται μια υπολογιστική μονάδα.

Ο δείκτης που αναφέρθηκε νωρίτερα και που είναι απαραίτητος στην εκτέλεση του προγράμματος είναι ο NDRange, ο οποίος περιγράφει έναν χώρο N-διαστάσεων, με το N να μπορεί να πάρει τιμές από 1 έως 3. Το κάθε work-item έχει το δικό του global ID καθώς και ένα local ID όταν τα στοιχεία κατηγοριοποιούνται σε ομάδες, ώστε να ξεχωρίζει από τα υπόλοιπα. Το ίδιο συμβαίνει και με τα work group IDs.

Τα παραπάνω παρουσιάζονται με μεγαλύτερη ευκρίνεια στο σχήμα που ακολουθεί. Έτσι, στο παρακάτω σχήμα παρουσιάζεται ένα δισδιάστατο πλέγμα όπου το WG-X υποδηλώνει το μήκος των γραμμών μιας ομάδας, ενώ το WG-Y υποδεικνύει το μήκος των στηλών για την ίδια ομάδα αλλά και το πώς τα work-items ομαδοποιούνται στην ομάδα [5].



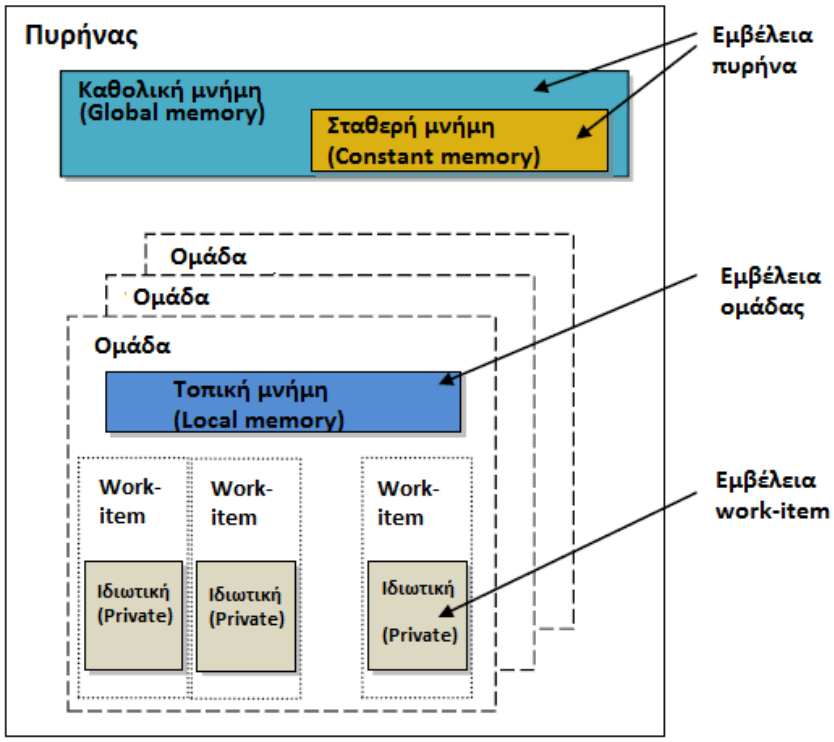
Σχήμα 1.6 : Διαχωρισμός Work-items και Work-groups στην OpenCL

Πριν από την εκτέλεση των πυρήνων στις συσκευές, το πρόγραμμα του οικοδεσπότη επεξεργαστή παίζει έναν πολύ σημαντικό ρόλο κι αυτός είναι να ορίσει ένα γενικό πλαίσιο (context) με τις συσκευές προκειμένου να καθορίσει τη σειρά εκτέλεσης των εργασιών (tasks). Το πρόγραμμα του οικοδεσπότη επεξεργαστή δημιουργεί το γενικό πλαίσιο καθορίζοντας ή και δημιουργώντας τα εξής:

- Όλες οι συσκευές πρέπει να χρησιμοποιούνται από το πρόγραμμα του οικοδεσπότη επεξεργαστή.
- Οι OpenCL πυρήνες πρέπει να τρέξουν σ ’αυτές τις συσκευές.
- Τα αντικείμενα της μνήμης πρέπει να είναι έγκλειστα στα δεδομένα που θα χρησιμοποιηθούν.
- Όταν καθοριστούν αυτά, ο οικοδεσπότης επεξεργαστής πρέπει να δημιουργήσει μια δομή δεδομένων που λέγεται ουρά εντολών (command queue), η οποία θα χρησιμοποιηθεί από αυτόν ώστε να διευθύνει την εκτέλεση των πυρήνων στις συσκευές και για να μπουν στην ουρά οι εντολές και να προγραμματιστούν στις συσκευές. Η ουρά εντολών αποδέχεται : εντολές εκτέλεσης πυρήνα, εντολές μεταφοράς μνήμης και εντολές συγχρονισμού. Επιπρόσθετα, οι ουρές εντολών εκτελούν τις εντολές με τη σειρά (in-order) όπως τους έχει δοθεί αλλά και εκτός σειράς (out of order). Εάν το πρόγραμμα διασπαστεί σε μικρότερες ανεξάρτητες εργασίες, είναι πιθανό να δημιουργηθούν πολλαπλές ουρές εντολών που να στοχεύουν σε διαφορετικές συσκευές στις οποίες θα προγραμματιστούν οι εργασίες αυτές και μετά η OpenCL θα τις εκτελέσει ταυτοχρόνως.

1.2.1.3. Μοντέλο Μνήμης

Τα διάφορα συστήματα μνήμης ποικίλουν από πλατφόρμα σε πλατφόρμα και παρουσιάζουν διαφορετικά χαρακτηριστικά. Για το λόγο αυτό η OpenCL καθορίζει ένα σύστημα τεσσάρων επιπέδων στην ιεραρχία μνήμης για κάθε υπολογιστική συσκευή όπως αυτά φαίνονται στο σχήμα 1.7 . Η ευθύνη του οικοδεσπότη επεξεργαστή είναι να διασφαλίζει πάντα ότι τα απαραίτητα δεδομένα είναι διαθέσιμα στη συσκευή πριν να ξεκινήσει η εκτέλεση του πυρήνα.



Σχήμα 1.7: Το μοντέλο μνήμης της OpenCL

Το μοντέλο μνήμης περιλαμβάνει τα εξής είδη μνημών:

- Καθολική μνήμη (Global memory): Η καθολική μνήμη είναι ορατή απ' όλες τις υπολογιστικές μονάδες της συσκευής, παρουσιάζει όμως μεγάλη καθυστέρηση πρόσβασης (high access latency). Κάθε φορά που μεταφέρονται δεδομένα από τον οικοδεσπότη επεξεργαστή στη συσκευή, τα δεδομένα περνάνε στην καθολική μνήμη. Ακόμα, τα δεδομένα που υπάρχουν στη συσκευή και πρέπει να μεταφερθούν πίσω στον οικοδεσπότη επεξεργαστή, διαμένουν στην καθολική μνήμη. Η λέξη κλειδί __global αποτελεί δείκτη που υποδηλώνει ότι τα δεδομένα θα μεταφερθούν στην καθολική μνήμη.

- Σταθερή μνήμη (Constant memory): Η σταθερή μνήμη δεν πρέπει να συγχέεται με τη μνήμη που σχεδιάστηκε μόνο για διάβασμα δεδομένων (read-only), αλλά για δεδομένα που προσπελαύνονται από όλα τα work-items. Οι μεταβλητές των οποίων οι τιμές παραμένουν αμετάβλητες (όπως για παράδειγμα του αριθμού π) εμπίπτουν σ 'αυτή την κατηγορία. Η σταθερή μνήμη, όπως παρατηρείται και σχηματικά, αποτελεί κομμάτι της καθολικής μνήμης, με αποτέλεσμα δεδομένα που μεταφέρονται στην καθολική μνήμη να μπορούν να προσδιοριστούν ως «σταθερά». Η λέξη κλειδί __constant αποτελεί δείκτη που υποδηλώνει ότι τα δεδομένα θα μεταφερθούν στην σταθερή μνήμη.

- Τοπική μνήμη (Local memory): Η τοπική μνήμη είναι διαθέσιμη σε όλα τα work-items που εκτελούνται στην ίδια ομάδα (work-group) και υπάρχει σε κάθε επεξεργαστική μονάδα. Από πλευράς υλικού, οι τοπικές μνήμες είναι συνήθως «on-chip» μνήμες αν και αυτό δεν είναι πάντα απαραίτητο. Γίνεται πάντως κατανοητό ότι οι προσβάσεις στη τοπική μνήμη έχουν μικρή καθυστέρηση πρόσβασης (low access latency). Η λέξη κλειδί __local αποτελεί δείκτη που υποδηλώνει ότι τα δεδομένα θα μεταφερθούν στην τοπική μνήμη.

- Ιδιωτική μνήμη (Private memory): Η ιδιωτική μνήμη είναι μοναδική για κάθε work-item. Οι τοπικές μεταβλητές και οι τιμές των παραμέτρων του πυρήνα που δεν αποτελούν δείκτες, είναι ιδιωτικές από προεπιλογή. Συνήθως, οι ιδιωτικές μεταβλητές καταχωρούνται στην ιδιωτική μνήμη αν και οι ιδιωτικοί πίνακες μπορεί να καταχωρηθούν σε off-chip μνήμη (με αρκετή καθυστέρηση).

1.2.1.4. Μοντέλο Προγραμματισμού

Η OpenCL έχει τη δυνατότητα να πραγματοποιήσει παράλληλους υπολογισμούς σε επίπεδο διαχωρισμού έργων (task-based parallelism) αλλά και σε επίπεδο διαχωρισμού δεδομένων (data-based parallelism), όπως αναφέρθηκε σε προηγούμενη ενότητα. Παρακάτω αναλύεται-ο τρόπος με τον οποίο αυτό πραγματοποιείται στην OpenCL [6].

- Στο επίπεδο διαχωρισμού δεδομένων καθορίζεται ο συσχετισμός των work-items με τα δεδομένα, τα οποία ανατίθενται στα work-items. Σε ένα αυστηρό μοντέλο παραλληλισμού, σε επίπεδο διαχωρισμού δεδομένων, υπάρχει μία προς μία σχέση μεταξύ των work-items και των στοιχείων στη μνήμη που ο πυρήνας μπορεί να εκτελεστεί παράλληλα. Η OpenCL εφαρμόζει μια ελαστική εκδοχή του παραλληλισμού στο επίπεδο διαχωρισμού δεδομένων, στην οποία δεν είναι απαραίτητη η μία προς μία σχέση που περιγράφηκε παραπάνω, αλλά παρέχει ένα ιεραρχικό μοντέλο παράλληλου προγραμματισμού.

Υπάρχουν δύο τρόποι να προσδιοριστεί αυτή η ιεραρχία. Στο σαφώς ορισμένο μοντέλο, ο προγραμματιστής καθορίζει τον συνολικό αριθμό των work-items που θα εκτελεστούν παράλληλα καθώς και τον τρόπο με τον οποίο θα χωριστούν τα work-items αυτά σε ομάδες (workgroups). Στο έμμεσο μοντέλο, ο προγραμματιστής προσδιορίζει μόνο τον συνολικό αριθμό των work-items που θα εκτελεστούν παράλληλα, ενώ ο καθορισμός των ομάδων στις οποίες θα διαχωριστούν τα work-items πραγματοποιείται αυτόματα από την ίδια την OpenCL.

- Στο επίπεδο διαχωρισμού έργων καθορίζεται ένα μοντέλο στο οποίο μια μοναδική υπόσταση ενός πυρήνα εκτελείται ανεξάρτητα από κάθε χώρο δείκτη. Είναι ισοδύναμο με την εκτέλεση ενός πυρήνα σε μια υπολογιστική μονάδα μιας ομάδας η οποία περιέχει ένα μόνο work-item. Σύμφωνα με αυτό το μοντέλο, οι χρήστες εκφράζουν τον παραλληλισμό με τους παρακάτω τρόπους:

    - Χρησιμοποιώντας τύπους δεδομένων που καθορίζονται από τη συσκευή.
    - Εισάγοντας ατόφιους πυρήνες που έχουν αναπτυχθεί χρησιμοποιώντας ένα προγραμματιστικό μοντέλο ορθογώνιο στην OpenCL.

2.Κεφάλαιο για τα FPGAs

2.1. Εισαγωγικά για τα FPGAs

Μια ηλεκτρονική συσκευή ή ενσωματωμένο σύστημα ονομάζεται προγραμματιζόμενο στο πεδίο (field-programmable ή in-place programmable) αν το υλικολογισμικό του (firmware), που βρίσκεται αποθηκευμένο σε μη πτητική μνήμη όπως η ROM, μπορεί να τροποποιηθεί επί τόπου χωρίς να αποσυναρμολογηθεί η συσκευή ή να επιστραφεί στον κατασκευαστή της.

Αυτό είναι συχνά ένα πολύ επιθυμητό χαρακτηριστικό καθώς υπάρχει έτσι η δυνατότητα να μειωθεί τόσο το κόστος όσο και ο χρόνος διόρθωσης λαθών αλλά και του firmware. Για παράδειγμα, μια εταιρεία κατασκευής καμερών μπορεί να αναπτύξει ένα firmware, το οποίο θα έχουν την δυνατότητα να εγκαταστήσουν όλοι οι πελάτες της συγκεκριμένης εταιρείας, μέσω ενός υπολογιστή με χρήση καλωδίου USB. Στο παρόν κεφάλαιο αναλύεται η σημασία και η λειτουργικότητα συσκευών που υποστηρίζουν αυτά τα χαρακτηριστικά , αλλά και πώς αυτές μπορούν να παρέχουν σημαντικά οφέλη σε ποικίλες εφαρμογές όπως είναι για παράδειγμα οι ιατρικές, με τις οποίες ασχολείται η παρούσα διπλωματική εργασία.

2.1.1. Περιγραφή-Ορισμός των FPGAs

Το FPGA ή Field Programmable Gate Array, ή συστοιχία επι τόπου προγραμματιζόμενων πυλών είναι ένας τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης, το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης και άλλα. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες.

Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία. Ο κώδικας με τον οποίο προγραμματίζεται το FPGA γράφεται σε γλώσσες περιγραφής υλικού (VHDL, AHDL, Verilog).

Το FPGA έχει παρόμοιο πεδίο εφαρμογών με άλλα προγραμματιζόμενα ολοκληρωμένα ψηφιακά κυκλώματα όπως τα PLD (programmable logic device) και τα ASIC (application-specific integrated circuit) με ορισμένες όμως διαφορές. Το FPGA χάνει τον προγραμματισμό του κάθε φορά που διακόπτεται η τάση τροφοδοσίας του. Επομένως, απαιτεί εξωτερικό μικροεπεξεργαστή, ή μνήμη με μόνιμη συγκράτηση δεδομένων (μη πτητική μνήμη), από τα οποία θα προγραμματίζεται κάθε φορά που επανέρχεται η τάση τροφοδοσίας.

Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του μικροεπεξεργαστή ή τα δεδομένα της μνήμης που το ελέγχει. Δεν υπάρχει όριο στο πόσες φορές μπορεί να επαναπρογραμματιστεί. Η κατανάλωση ισχύος είναι σημαντικά αυξημένη, σε σχέση με τα ASIC.

Βασική δομική μονάδα του FPGA είναι το λογικό μπλοκ, με τη χρήση του οποίου υλοποιούνται οι λογικές συναρτήσεις που εκφράζουν τη λειτουργία ενός ψηφιακού κυκλώματος. Ανάλογα με το μέγεθος του κυκλώματος πολλά λογικά μπλοκ συνδέονται προκειμένου να υλοποιήσουν το πλήθος των απαραίτητων λογικών συναρτήσεων. Χρησιμοποιώντας τα «prebuilt» μπλοκ και προγραμματισμένες πηγές ρουτίνας, είναι δυνατόν να ρυθμιστούν τα FPGAs, ούτως ώστε να μη χρειαστεί επέμβαση από μεριάς υλικού. Τα FPGAs όντας πλήρως επαναριθμιζόμενα, αποκτούν αμέσως μια νέα «προσωπικότητα» όταν εφαρμόζεται μια νέα ρύθμιση κυκλώματος.

Στο παρελθόν, η τεχνολογία των FPGA μπορούσε να χρησιμοποιηθεί μόνο από μηχανικούς που είχαν βαθιά κατανόηση της ψηφιακής σχεδίασης υλικού. Πλέον, η εμφάνιση των σχεδιαστικών εργαλείων υψηλού επιπέδου αλλάζει τους κανόνες προγραμματισμού των FPGA, κάνοντας χρήση νέων τεχνολογιών που μετατρέπουν τα μπλοκ διαγράμματα ή τον κώδικα C σε ψηφιακά κυκλώματα.

Η υιοθέτηση των FPGA από διάφορες βιομηχανίες οφείλεται στο γεγονός ότι τα FPGA συνδυάζουν τα καλύτερα κομμάτια των ASICs και των συστημάτων που είναι βασισμένα σε επεξεργαστή. Τα FPGA παρέχουν ταχύτητα υλικού και αξιοπιστία, αλλά δεν απαιτούν υψηλές ποσότητες που θα μπορούσαν να δικαιολογήσουν το μεγάλο έξοδο του ASIC σχεδιασμού.

Έτσι το FPGA είναι ιδιαίτερα κατάλληλο στην περίπτωση που οι παράμετροι λειτουργίας του ενός συστήματος πρέπει να αλλάζουν συχνά ή για μικρές ποσότητες παραγωγής, ενώ το ASIC, λόγω μαζικής παραγωγής, είναι φτηνότερο όταν απαιτούνται μεγάλες ποσότητες και η επιθυμητή λειτουργία είναι αυστηρά προκαθορισμένη και χωρίς σφάλματα καθώς το ASIC δεν έχει την δυνατότητα να επαναπρογραμματίζεται.

Αντίθετα με τους επεξεργαστές, τα FPGAs είναι παράλληλα από τη φύση τους, έτσι ώστε διαφορετικές επεξεργαστικές λειτουργίες να μην χρειάζεται να ανταγωνιστούν για τους ίδιους υπολογιστικούς πόρους. Κάθε ανεξάρτητη επεξεργαστική εργασία ανατίθεται σε ένα συγκεκριμένο τμήμα του chip και μπορεί να δράσει αυτόνομα χωρίς καμία επιρροή από άλλα λογικά μπλοκ. Αυτό έχει ως αποτέλεσμα να μην επηρεάζεται η επίδοση ενός τμήματος της εφαρμογής , όταν προστίθεται επιπλέον φόρτος επεξεργασίας.

Σε αντίθεση με τα παλαιότερης γενεάς FPGAs που χρησιμοποιούσαν I/O με προγραμματιζόμενη λογική και διασυνδέσεις, τα σημερινά FPGAs αποτελούνται από ποικίλα σύνολα από επαναρυθμιζόμενες ενσωματωμένες SRAM, υψηλής ταχύτητας πομποδέκτες, υψηλής ταχύτητας I/O, λογικά μπλοκ και δρομολόγηση.

Ειδικότερα, ένα FPGA περιέχει λογικά προγραμματιζόμενα μέρη που ονομάζονται λογικά στοιχεία (logic elements -LEs) και μια ιεραρχία από επαναρυθμιζόμενες συνδέσεις που επιτρέπουν στα LEs να συνδεθούν φυσικά. Υπάρχει η δυνατότητα ρύθμισης των LEs ώστε να εκτελέσουν περίπλοκες συνδυαστικές συναρτήσεις ή απλά για να αποτελέσουν λογικές πύλες όπως είναι οι AND και οι XOR. Στα περισσότερα FPGAs, τα λογικά μπλοκ περιλαμβάνουν στοιχεία μνήμης τα οποία μπορεί να είναι απλά flip-flops ή πιο περίπλοκα μπλοκ μνήμης.

Όσο τα FPGAs εξελίσσονται περαιτέρω, οι συσκευές γίνονται πιο ενσωματωμένες. Έτσι, παρέχονται συναρτήσεις με δυνατότητα χαμηλότερης ενέργειας και κόστους. Νεότερες οικογένειες FPGA που εμπεριέχουν σκληρούς ενσωματωμένους επεξεργαστές αναπτύσσονται, μετατρέποντας τις συσκευές σε συστήματα στο chip (systems on a chip - SoC).

2.1.2. Πλεονεκτήματα FPGAs

Η χρήση FPGAs παρουσιάζει πολλά και σημαντικά πλεονεκτήματα μερικά από τα οποία παρουσιάζονται παρακάτω [7].

- Κόστος : Η αρχική επένδυση σε ASICs είναι πολύ μεγαλύτερη σε σχέση με τις λύσεις που είναι βασισμένες σε FPGA chips. Το υψηλό κόστος των ASICs είναι εύκολο να δικαιολογηθεί όταν αφορά την παραγωγή χιλιάδων chips τον χρόνο, αλλά πολλοί τελικοί χρήστες έχουν ανάγκη προσαρμοσμένων λειτουργιών υλικού για τα δεκάδες έως εκατοντάδες των συστημάτων τους, που βρίσκονται σε ανάπτυξη. Επιπλέον, η ίδια η φύση του προγραμματιζόμενου πυριτίου σημαίνει ότι δεν έχει το ίδιο κόστος κατασκευής ή μεγάλους χρόνους αναμονής για τη συναρμολόγηση. Επειδή οι απαιτήσεις του συστήματος συχνά αλλάζουν με την πάροδο του χρόνου, το κόστος των επιπρόσθετων αλλαγών σε FPGA chips είναι αμελητέο σε σύγκριση με το μεγάλο έξοδο επαναστροβιλισμού ενός ASIC.

- Μακροχρόνια συντήρηση και διαθεσιμότητα: Όπως αναφέρθηκε προηγουμένως, τα FPGA chips επιδέχονται αναβάθμισης και δεν απαιτούν τον χρόνο και το κόστος που θα χρειαζόταν για έναν ανασχεδιασμό σε ASIC. Τα ψηφιακά πρωτόκολλα επικοινωνίας, για παράδειγμα, έχουν προδιαγραφές που μπορούν να αλλάξουν με την πάροδο του χρόνου, και ως εκ τούτου επαφές που είναι βασισμένες σε ASIC μπορούν να προκαλέσουν προβλήματα συντήρησης και συμβατότητας. Αντίθετα, τα FPGA chips που έχουν την δυνατότητα αναδιαμόρφωσης των χαρακτηριστικών τους, μπορούν να συμβαδίσουν με μελλοντικές τροποποιήσεις που μπορεί να είναι απαραίτητες. Έτσι, καθώς τα συστήματα ωριμάζουν, μπορούν να γίνουν λειτουργικές βελτιώσεις χωρίς να ξοδέψει κανείς χρόνο για τον επανασχεδιασμό του υλικού ή την τροποποίηση της διάταξης του σχεδίου του πίνακα (board layout).

- Μπορεί να ενημερώνεται και να αναβαθμίζεται στο χώρο του πελάτη: Τα FPGAs σε αντίθεση με τα παραδοσιακά chip υπολογιστών είναι πλήρως παραμετροποιήσιμα. Οι ενημερώσεις και οι διάφορες λειτουργίες βελτίωσης μπορούν να πραγματοποιηθούν ακόμα και μετά την παράδοση στο χώρο του πελάτη.

- Εξαιρετικά μικρό χρονικό διάστημα από την παραγωγή στην αγορά: Μέσω της χρήσης των FPGAs παρέχεται ευελιξία και ταχύτητα στην ανάπτυξη πρωτοτύπων υλικού καθώς ένα μεγάλο μέρος της διαδικασίας ανάπτυξης του υλικού διευκολύνεται από την αυξανόμενη διαθεσιμότητα εργαλείων λογισμικού (Software tools). Επιπλέον, λόγω της πρώιμης διαθεσιμότητας των πρωτοτύπων υλικού, χρονοβόρες δραστηριότητες, όπως η εκκίνηση και ο εντοπισμός σφαλμάτων του υλικού μεταφέρονται στα αρχικά στάδια της συνολικής ανάπτυξης.

- Γρήγορα και αποτελεσματικά συστήματα : Τα διαθέσιμα συστήματα καλύπτουν ένα ευρύ φάσμα χρηστών και κατά συνέπεια, συχνά αποτελούν ένα συμβιβασμό ανάμεσα στην απόδοση και την συμβατότητα. Με τα FPGAs, τα συστήματα μπορούν να αναπτυχθούν και να είναι ακριβώς προσαρμοσμένα για την καθορισμένη εργασία και για το λόγο αυτό λειτουργούν πολύ αποδοτικά.

- Κέρδος απόδοσης για εφαρμογές λογισμικού: Οι σύνθετες εργασίες συχνά αντιμετωπίζονται μέσω της ανάπτυξης εφαρμογών λογισμικού σε συνδυασμό με την χρήση επεξεργαστών υψηλής απόδοσης. Στην περίπτωση αυτή, τα FPGAs προσφέρουν μια ανταγωνιστική εναλλακτική λύση, η οποία μέσω του παραλληλισμού και της δυνατότητας προσαρμογής σε συγκεκριμένη εργασία δημιουργούν επιπλέον κέρδος απόδοσης.

- Εφαρμογές σε πραγματικό χρόνο: Τα FPGAs είναι απολύτως κατάλληλα για εφαρμογές σε συστήματα στα οποία ο χρόνος είναι κρίσιμος. Σε αντίθεση με τις λύσεις λογισμικού με βάση τα λειτουργικά συστήματα πραγματικού χρόνου, τα FPGAs παρέχουν πραγματική αιτιοκρατική συμπεριφορά. Με τη βοήθεια αυτής της ευελιξίας ακόμα και περίπλοκοι υπολογισμοί μπορούν να εκτελεστούν σε εξαιρετικά σύντομο χρονικό διάστημα.

- Μαζική, παράλληλη επεξεργασία των δεδομένων : Ο όγκος των δεδομένων στα σύγχρονα συστήματα αυξάνεται συνεχώς, γεγονός που οδηγεί στο πρόβλημα ότι τα συστήματα σειριακής επεξεργασίας δεν είναι πλέον σε θέση να επεξεργαστούν τα δεδομένα στον απαιτούμενο χρόνο. Κυρίως μέσω του παραλληλισμού, τα FPGAs παρέχουν μια λύση σε αυτό το πρόβλημα, η οποία επιπλέον κλιμακώνεται άριστα.

2.2. FPGAs και OpenCL

Οι σημερινοί δέκα κορυφαίοι υπερυπολογιστές από πλευράς ενεργειακής απόδοσης, είναι όλοι ετερογενή υπολογιστικά συστήματα. Ενώ αυτά τα συστήματα χρησιμοποιούν σήμερα κάρτες γραφικών ως ετερογενείς επιταχυντές, πρόσφατη έρευνα επισημαίνει ότι τα FPGAs μπορούν να χρησιμοποιηθούν ως επιταχυντές αποτελεσματικότερα σε ορισμένες περιοχές υπολογιστικής υψηλής απόδοσης.

Ένα βασικό όμως μειονέκτημα στη χρήση FPGAs είναι η δυσκολία στον προγραμματισμό τους. Ο παραδοσιακός τρόπος προγραμματισμού ενός FPGA είναι μέσω της χρήσης γλωσσών περιγραφής υλικού (Hardware Description Languages -HDL) όπως η Verilog και η VHDL. Ο μέσος προγραμματιστής ηλεκτρονικών υπολογιστών σπάνια κατέχει την τεχνογνωσία αυτών των γλωσσών που απαιτούν ειδικές τεχνικές ικανότητες. Ένας αριθμός λύσεων υψηλότερου επιπέδου, γνωστών και ως «από -C-σε-πύλες", έχουν προταθεί και στο εμπόριο. Ωστόσο, αυτές οι προσεγγίσεις δεν είναι αποκλειστικές και ειδικές ανά προμηθευτή, καθώς υπολείπονται σε ένα αρχιτεκτονικό πλαίσιο αφαίρεσης.

Το ανερχόμενο και ανοιχτό πρότυπο προγραμματισμού για ετερογενείς υπολογισμούς είναι η OpenCL, όπως αναφέρθηκε σε προηγούμενο κεφάλαιο. Η OpenCL προσφέρει ένα ενιαίο μοντέλο προγραμματισμού C για οποιαδήποτε συσκευή που συμμορφώνεται με τα πρότυπά της. Αυτό την καθιστά μεταφέρσιμη από μια υπολογιστική συσκευή σε μια άλλη, έτσι ώστε να μπορεί εύκολα να επωφεληθεί από μία αναβάθμιση του hardware. Στην αγορά κυκλοφορούν κάποια εργαλεία ανάπτυξης λογισμικού σε OpenCL (Software Development Kits-SDKs) για FPGAs. Τα εργαλεία αυτά επιτρέπουν στα FPGAs να αντιμετωπίζονται ως επιταχυντές με χρήση της OpenCL.

Τα FPGAs έχουν μεγάλες δυνατότητες για εφαρμογές επεξεργασίας υψηλών επιδόσεων και για παράλληλες εφαρμογές, δεδομένου ότι είναι συσκευές χαμηλής ισχύος σε σύγκριση με τους πολυπύρηνους επεξεργαστές και τις κάρτες γραφικών που χρησιμοποιούνται συνήθως σε εφαρμογές υψηλών αποδόσεων (High- performance computing-HPC). Προσφέρουν ένα τεράστιο βαθμό παραλληλισμού και ένα σημαντικό επίπεδο εύρους ζώνης της εσωτερικής μνήμης [8] .

Με βάση τα παραπάνω συνοψίζονται τα βασικά πλεονεκτήματα εκτέλεσης της OpenCL σε FPGA:

- Τυποποιημένο πολυπύρηνο προγραμματιστικό μοντέλο: Η OpenCL υποστηρίζεται από μια κοινοπραξία από εταιρείες που προσπαθούν να διαμορφώσουν ένα φορητό παράλληλο προγραμματιστικό μοντέλο.
- Μεταφερσιμότητα και παράλειψη μερικών βασικών λεπτομερειών σχετικών με το hardware: Η OpenCL βασίζεται στο πρότυπο C με λίγες επεκτάσεις. Έτσι, δεν είναι απαραίτητο να προσδιορίζεται η συμπεριφορά κύκλο προς κύκλο.
- Διευκολύνεται η κωδικοποίηση κλιμακούμενων λύσεων: Ο παραλληλισμός των εφαρμογών καθορίζεται από τον προγραμματιστή. Έτσι ο Compiler και οι ρουτίνες του λειτουργικού διανέμουν τον φόρτο εργασίας ανάλογα με τα χαρακτηριστικά του επιταχυντή.

- Διαχειρίζονται μεγάλοι χρόνοι μεταγλώττισης: Μια μεγάλη κατηγορία των αλλαγών σχεδιασμού μπορεί να αντιμετωπιστεί ακαριαία.

3.Οπισθοπροβολή

Οπισθοπροβολή (backprojection) είναι η αλγοριθμική τεχνική ανακατασκευής εικόνας σύμφωνα με την οποία η ανακατασκευασμένη εικόνα δημιουργείται κατανέμοντας τις τιμές των δεδομένων προβολής ομοιόμορφα κατά μήκος της ακτίνας προβολής. Οπισθοπροβάλοντας τα δεδομένα προβολής απ' όλες τις όψεις, προσδιορίζεται μια εκτίμηση της συνάρτησης χωρικής κατανομής που παριστάνει το δισδιάστατο αντικείμενο [9].

3.1. Δεδομένα εισόδου για την ανακατασκευή εικόνας

3.1.1. Συλλογή Δεδομένων

Η οπισθοπροβολή, όπως προαναφέρθηκε, χρησιμοποιείται στα πλαίσια της τομογραφίας για την ανακατασκευή εικόνας. Τομογραφία ονομάζεται η τεχνική απεικόνισης μιας τομής ενός αντικειμένου, η συνολική εικόνα του οποίου μπορεί να αναπαραχθεί με την χρήση πολλών διαφορετικών τομών του.

Ανακατασκευή εικόνας (Image Reconstruction) ονομάζεται η μαθηματική επεξεργασία μιας ομάδας δεδομένων προβολής επερχόμενων με μη επεμβατικό τρόπο από τον οργανισμό, η οποία έχει ως σκοπό την παραγωγή μιας τομογραφικής εικόνας υψηλής ποιότητας. Οι σύγχρονες τομογραφικές μέθοδοι που χρησιμοποιούνται στην ιατρική στηρίζονται στην αλληλεπίδραση διαφόρων μορφών ακτινοβολίας με την ύλη. Ενδεικτικά, στον πίνακα 3.1 παρουσιάζονται τα χαρακτηριστικά των κυριότερων τομογραφικών μεθόδων.

Τομογραφική μέθοδος	**Είδος ακτινοβολίας**	**Μετρούμενες παράμετροι**
Αξονική τομογραφία ($X-Ray\ CT$)	Ακτίνες X ($20-150\ keV$)	Συντελεστής Εξασθένισης
Μαγνητική τομογραφία (MRI)	Ηλεκτρομαγνητική RF	Πυκνότητα Πρωτονίων Χρόνοι Αποκατάστασης
Τομογραφία Υπερήχων ($Ultrasound\ CT$)	Υπέρηχοι ($1-50\ MHz$)	Δείκτης Διάθλασης Συντελεστής Απορρόφησης
Τομογραφία Εκπομπής Ποζιτρονίου (PET)	Ακτίνες γ ($511\ keV$)	Συγκέντρωση Ραδιενεργού Ιχνηθέτη
Τομογραφία Εκπομπής Φωτονίου ($SPECT$)	Ακτίνες γ ($20-150\ keV$)	Συγκέντρωση Ραδιενεργού Ιχνηθέτη

Πίνακας 3.1: Χαρακτηριστικά των κυριότερων τομογραφικών μεθόδων

Η παρούσα διπλωματική εργασία ασχολείται με την αξονική τομογραφία, στην οποία χρησιμοποιείται κυρίως ο αλγόριθμος οπισθοπροβολής. Όπως παρουσιάζεται και στον παραπάνω πίνακα, η αξονική τομογραφία (Computed Tomography-CT) βασίζεται στην ανακατασκευή της εσωτερικής μορφολογίας των διάφορων οργάνων του σώματος με τη σύνθεση πολλαπλών προβολών εγκάρσιων τομών του συγκεκριμένου οργάνου, χρησιμοποιώντας ακτίνες Χ και μετρώντας τον συντελεστή εξασθένισης.

Η μέτρηση του συντελεστή εξασθένισης γίνεται με την βοήθεια λυχνίας ακτίνων Χ, που κινείται γύρω από το προς εξέταση σώμα του ασθενή και εκπέμπει λεπτή δέσμη ακτινοβολίας ανά συγκεκριμένη γωνιακή απόσταση. Η λεπτή δέσμη ακτινοβολίας εξέρχεται εξασθενημένη λόγω της απορρόφησης ενέργειας από τους διάφορους ιστούς του σώματος που παρεμβάλλονται στην διαδρομή της.

Οι διάφορες τιμές εξασθένισης της ακτινοβολίας από κάθε προβολή καταγράφονται με τη βοήθεια ειδικών ανιχνευτών (detectors) που βρίσκονται σε αντιδιαμετρική θέση με την εστία της λυχνίας. Στη συνέχεια, οι τιμές αυτές μετατρέπονται σε ηλεκτρικά σήματα που μέσω του Η/Υ επεξεργάζονται και με τους κατάλληλους αλγορίθμους ανακατασκευής συνθέτουν την τελική εικόνα.

Ο μαθηματικός τύπος που δίνει την ένταση της ακτινοβολίας μετά την έξοδό της από το σώμα είναι ο εξής:

$$I = I_0 e^{-(\mu E)^d} \tag{3.1}$$

Όπου: I είναι η ένταση της ακτινοβολίας εξόδου, I_0 είναι η ένταση της ακτινοβολίας εισόδου, E είναι η ενέργεια ακτινοβολίας, μ είναι ο συντελεστής εξασθένισης της ακτινοβολίας και d είναι το πάχος της τομής.

3.1.2. Μαθηματικός ορισμός

Θεωρείται ότι ένα δισδιάστατο αντικείμενο παριστάνεται από τη συνάρτηση $f(x, y)$ της χωρικής κατανομής μιας φυσικής ποσότητας στο ορθογώνιο σύστημα συντεταγμένων (t, s)που έχει στραφεί ως προς το σύστημα (x, y)κατά γωνία θ και ισχύει η παρακάτω σχέση:

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.2}$$

Το επικαμπύλιο ολοκλήρωμα της $f(x, y)$ κατά μήκος μιας γραμμής L παράλληλης προς τον άξονα s, η οποία ονομάζεται τομογραφική ακτίνα, αποτελεί τα δεδομένα προβολής και ισούται με:

$$P(\theta, t) = \int_L f(x, y) ds \tag{3.3}$$

Πρόκειται ουσιαστικά για ένα μονοδιάστατο προφίλ της μετρούμενης ποσότητας ως συνάρτηση της θέσης και αντιστοιχεί σε μια δεδομένη γωνία θ. Η συλλογή πολλών διαφορετικών προβολών μπορεί να παρασταθεί σ' ένα δισδιάστατο διάγραμμα, όπου ο ένας άξονας είναι η θέση t και ο άλλος η γωνία θ. Το διάγραμμα αυτό ονομάζεται Σινόγραμμα (Sinogram) ή Μετασχηματισμός Radon (Radon Transform).

Ο μετασχηματισμός Fourier μιας ομάδας δεδομένων προβολής σε μια διεύθυνση δίνεται από τη σχέση:

$$\hat{f}(\theta,\omega) = \int_L P(\theta,t)e^{-j2\pi\omega t}dt \tag{3.4}$$

Ο μετασχηματισμός Radon καταλήγει στην εξής σχέση:

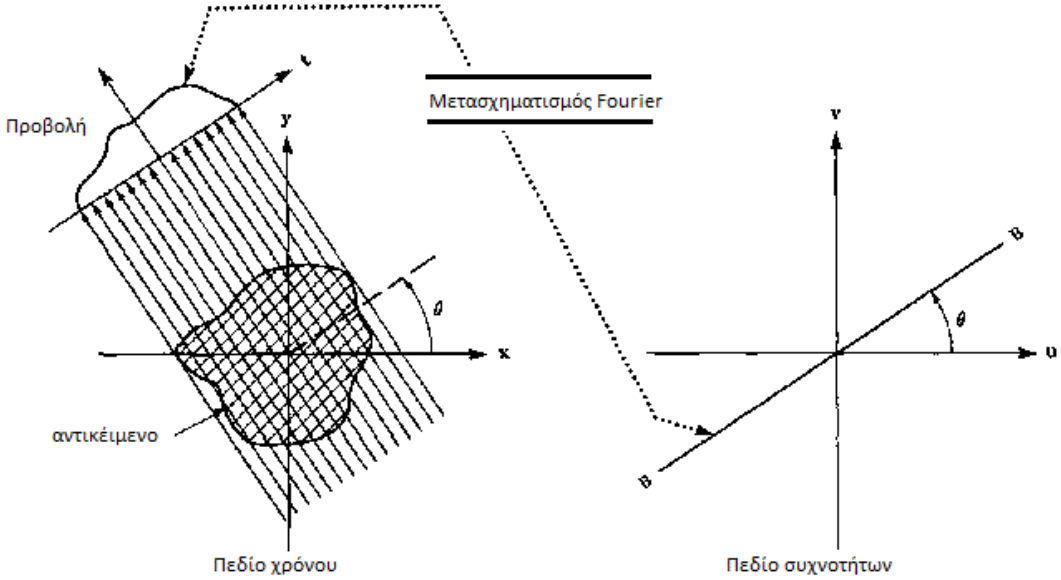
$$\hat{f}(\theta,\omega) = \iint f(x,y)e^{-2\mathrm{j}\pi\omega(xcos\theta+ysin\theta)}dxdy \tag{3.5}$$

που είναι ο δισδιάστατος μετασχηματισμός Fourier $F(u,v)$ της συνάρτησης $f(x,y)$ με την προϋπόθεση ότι $u = \omega cos\theta$ και $v = \omega sin\theta$. Η παραπάνω σχέση με χρήση των ιδιοτήτων της κρουστικής συνάρτησης (Dirac delta function) μετατρέπεται στην παρακάτω σχέση:

$$\hat{f}(\theta,r) = \iint f(x,y)\delta(r - cos\theta - ysin\theta)dxdy \tag{3.6}$$

Όπου r και θ είναι οι πολικές συντεταγμένες και δ η κρουστική συνάρτηση (Dirac delta function).

Ο μετασχηματισμός Fourier διευκολύνει την γραφική κατανόηση του μετασχηματισμού Radon καθώς παρατηρείται, όπως φαίνεται και στο παρακάτω σχήμα, ότι ο μετασχηματισμός Fourier μιας προβολής ισούται με το μετασχηματισμό Fourier του αντικειμένου κατά μήκος μιας ακτινικής γραμμής. Επιπρόσθετα, οι υπολογισμοί στο πεδίο της συχνότητας πραγματοποιούνται ευκολότερα και προσφέρουν πληροφορία και για τον θόρυβο των σημάτων προς επεξεργασία.



Σχήμα 3.1:Μετασχηματισμός Fourier δεδομένων προβολής

3.2. Αλγόριθμος φιλτραρισμένης οπισθοπροβολής

3.2.1. Βασικά στοιχεία αλγορίθμου –Θεωρητικό υπόβαθρο

Η βασική ιδέα του απλού αλγορίθμου της οπισθοπροβολής (Backprojection) είναι η δημιουργία της ανακατασκευασμένης εικόνας κατανέμοντας τις τιμές των δεδομένων προβολής ομοιόμορφα κατά μήκος της ακτίνας προβολής. Οπισθοπροβάλοντας τα δεδομένα προβολής απ' όλες τις γωνίες λήψης προβολών, προσδιορίζεται μια εκτίμηση της χωρικής συνάρτησης κατανομής του δισδιάστατου αντικειμένου. Έτσι, η ζητούμενη συνάρτηση $f(x,y)$ δίνεται από την εξής σχέση:

$$f(x,y) = \int_0^{\pi} \hat{f}(xcos\theta + ysin\theta)d\theta \qquad (3.7)$$

όπου $\hat{f}$ οι προβολές όπως παρουσιάστηκαν στην προηγούμενη ενότητα ή αλλιώς σινόγραμμα (Sinogram) και θ η γωνία λήψης προβολών. Στο σημείο αυτό πρέπει να σημειωθεί ότι έγινε υπόθεση ότι οι γωνίες λήψης προβολών είναι κατανεμημένες στο διάστημα $[0,\pi)$.

Η διαφορά μεταξύ του απλού αλγορίθμου οπισθοπροβολής και του αλγορίθμου φιλτραρισμένης οπισθοπροβολής, έγκειται στο γεγονός ότι στα δεδομένα προβολής εφαρμόζεται κατάλληλο φίλτρο από διαφορετικές γωνίες.

Έτσι, οι προβολές $P(\theta,t)$ προκειμένου να φιλτραριστούν πρέπει να υποστούν συνέλιξη με μια κατάλληλη συνάρτηση $h(t)$ όπως παρουσιάζεται και στην εξής σχέση:

$$P'(\theta, t) = P(\theta, t) \otimes h(t) \tag{3.8}$$

Εναλλακτικά, οι προβολές μπορούν να πολλαπλασιαστούν με τη συνάρτηση $H(\omega)$ αν μετασχηματιστούν κατά Fourier στον χώρο συχνοτήτων με την χρήση των μεθόδων FFT (Fast Fourier Transform) δηλαδή :

$$p(\omega, \theta) = FT\{P(\theta, t)\} \tag{3.9}$$

και στη συνέχεια με τον αντίστροφο μετασχηματισμό Fourier να ληφθούν οι τιμές στο χώρο.

Η διαδικασία αυτή έχει στόχο την βελτίωση της ποιότητας ανακατασκευής της εικόνας όσο αυξάνεται ο αριθμός των προβολών καθώς και τη μείωση του θορύβου σε σχέση με τις μη φιλτραρισμένες προβολές, ειδικότερα γύρω από τις αιχμές.

4.Χρησιμοποιηθέντα εργαλεία

Στο κεφάλαιο αυτό παρουσιάζονται τα εργαλεία που χρησιμοποιήθηκαν για την εκτέλεση του αλγορίθμου της οπισθοπροβολής και την λήψη μετρήσεων. Επιγραμματικά οι συσκευές που χρησιμοποιήθηκαν ήταν οι εξής :

- FPGA → Altera Cyclone V SoC 5CSEMA5F31 (800MHz Dual-core ARM Cortex-A9, 50MHz FPGA fabric)
- CPU1 → Intel Core i7-4710MQ 2.5GHz
- CPU2 → Intel Core i5-M520 2.4GHz
- GPU1 → AMD FirePro M5100 775MHz
- GPU2 → NVIDIA GeForce 310M 1530MHz

Στις ακόλουθες ενότητες συνοψίζονται τα χαρακτηριστικά της κάθε συσκευής.

4.1. Περιγραφή των χρησιμοποιηθέντων εργαλείων

Οι υλοποιήσεις του αλγορίθμου της οπισθοπροβολής εφαρμόστηκαν στο FPGA με την χρήση του εργαλείου ανάπτυξης λογισμικού της Altera για OpenCL (Altera SDK for OpenCL). Η συσκευή που χρησιμοποιήθηκε για τις μετρήσεις ήταν το Altera Cyclone V SoC 5CSEMA5F31C6, συμπεριλαμβανομένου και ενός dual-core ARM επεξεργαστή και των βασικών αρχών σχεδίασης ενός FGPA (FGPA fabric) στην αναπτυξιακή πλακέτα Terasic DE1-SoC. Η αρχιτεκτονική της συγκεκριμένης συσκευής υποστηρίζει το μοντέλο εκτέλεσης της OpenCL όπως αυτό παρουσιάστηκε στο κεφάλαιο 1.

Ο διπύρηνος (dual-core) ARM επεξεργαστής παίζει το ρόλο του host επεξεργαστή, ενώ ο κώδικας του πυρήνα εκτελείται στο FPGA. Η επικοινωνία μεταξύ των δύο επιτυγχάνεται μέσω του τοπικού διαδρόμου του επεξεργαστή (processor local bus), ο οποίος στην περίπτωση του ARM επεξεργαστή είναι ένας διάδρομος μικροεπεξεργαστή (Advanced Microcontroller Bus - AXI bus). Το Altera SDK για OpenCL περιλαμβάνει όλα τα απαραίτητα εργαλεία λογισμικού προκειμένου να γίνει εκκίνηση (boot) του λειτουργικού συστήματος Linux στον ARM επεξεργαστή, να προγραμματιστεί το FPGA μέσω ήδη παραχθέντων αρχείων (generated files) σε Windows ή Linux περιβάλλον και να εκτελεστεί ο κώδικας του πυρήνα και του επεξεργαστή host.

Η παρούσα διπλωματική εκμεταλλεύεται τις διαφορετικές δυνατότητες εφαρμογής που προσφέρονται από το Altera SDK για την OpenCL, για το σχεδιασμό του αλγορίθμου της οπισθοπροβολής. Χωρίς απώλεια της γενικότητας, το περιβάλλον Altera έχει επιλεγεί για λόγους διαθεσιμότητας.

Στη συνέχεια παρατίθενται λεπτομέρειες για το Altera SDK και το FPGA που χρησιμοποιήθηκε.

4.1.1. Altera SDK for OpenCL

Το Altera SDK (Software Development Kit-SDK) είναι το εργαλείο ανάπτυξης λογισμικού της Altera για OpenCL, το οποίο επιτρέπει στο χρήστη την ταχύτερη και υψηλότερου επιπέδου ανάπτυξη λογισμικού καθώς του δίνει την δυνατότητα αποφυγής της παραδοσιακής ροής ανάπτυξης FPGA υλικού.

Παρέχει τη δυνατότητα εξομοίωσης σε δευτερόλεπτα του OpenCL C επιταχυμένου κώδικα σε σε έναν x86 host επεξεργαστή με την δημιουργία μιας λεπτομερούς έκθεσης βελτιστοποίησης με πληροφορίες εξαρτήσεων σωλήνωσης (pipeline) για συγκεκριμένο αλγόριθμο, ή την δυνατότητα προτυποποίησης του πυρήνα επιταχυντή σε περιβάλλον FPGA μέσα σε λίγα λεπτά.

Το Altera SDK για OpenCL επιτρέπει την εύκολη υλοποίηση εφαρμογών σε FPGAs καθώς αφαιρεί την πολυπλοκότητα του υλικού ενός FPGA, επιτρέποντας έτσι στους προγραμματιστές να γράφουν κώδικα για επιταχυνόμενες λειτουργίες λογισμικού του πυρήνα σε OpenCL C (μια γλώσσα με ANSI C βάση με πρόσθετες κλήσεις OpenCL). Το εργαλείο εμπεριέχει ένα σύνολο εργαλείων με στόχο την ταχεία ροή ανάπτυξης των προγραμμάτων λογισμικού. Στην διαδικασία ανάπτυξης προγραμμάτων συμπεριλαμβάνονται:

- Ένας εξομοιωτής προκειμένου να ελεγχθεί ο κώδικας σε x86 βήμα προς βήμα, και να επιβεβαιώσει ο προγραμματιστής την σωστή λειτουργία του προγράμματος.
- Μια λεπτομερής αναφορά βελτιστοποίησης για να γίνουν κατανοητές οι εξαρτήσεις βρόγχου (load-store).
- Ένα εργαλείο ταχείας προτυποποίησης για χρονική μετατόπιση των μεγάλων χρόνων μεταγλώττισης που σχετίζονται με την οικοδόμηση ενός FPGA και εκτέλεση του κώδικα του πυρήνα σε ένα προ-χτισμένο (pre-built) πρότυπο FPGA.
- Ένας profiler που δείχνει την απόδοση στον πυρήνα για να εξασφαλιστεί η ορθή συνένωση μνήμης και να καθυστερήσουν τα pipelines υλικού.
- Ένας μεταγλωττιστής OpenCL ικανός να υλοποιεί πάνω από 300 βελτιστοποιήσεις στον κώδικα του πυρήνα και να παράγει ολόκληρη την εικόνα FPGA σε ένα βήμα.

Το Altera SDK για OpenCL είναι πλήρως διαθέσιμο στην παραγωγή, καθιστώντας την Altera την πρώτη εταιρεία FPGA που παρέχει λύσεις που ανταποκρίνονται στις προδιαγραφές της OpenCL. Το Altera SDK για OpenCL υποστηρίζει μια ποικιλία από host επεξεργαστές, συμπεριλαμβανομένων των ενσωματωμένων ARM Cortex-A9 processor cores σε συσκευές ενσωματωμένες σε chip (system on a chip –SoC), των IBM Power Series επεξεργαστών, καθώς και της τυπικής CPU x86.

Το Altera SDK για OpenCL προσφέρει κλιμακούμενες σε απόδοση λύσεις για πολλαπλά FPGAs και πολλαπλές κάρτες, καθώς και μια ποικιλία μνημών, όπως DDR SDRAM (Double data rate synchronous dynamic random-access memory) για σειριακή προσβάση στη μνήμη, QDR SRAM (Quad Data Rate SRAM) για τυχαία προσπέλαση ,` μνήμης ή την εσωτερική μνήμη FPGA για πρόσβαση στη μνήμη με χαμηλή καθυστέρηση (low-latency).

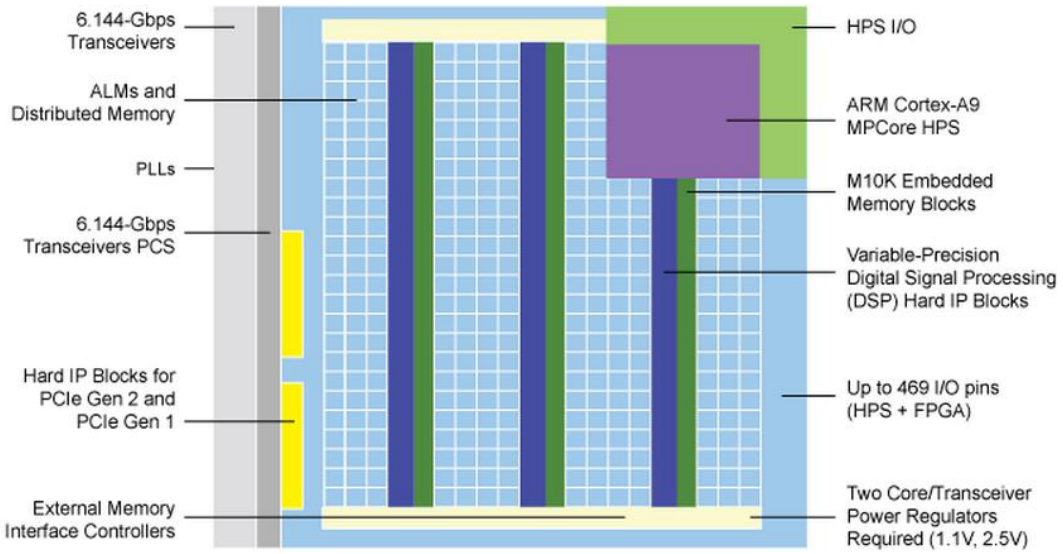
Υποστηρίζεται επίσης μισή-ακρίβεια καθώς και μονή και διπλή ακρίβεια κινητής υποδιαστολής [10].

4.1.2. Cyclone V SoCs

Το Cyclone V SoC (System on a chip) της Altera επιλέχθηκε καθώς προσφέρει το χαμηλότερο κόστος και την χαμηλότερη κατανάλωση ενέργειας από όλα τα διαθέσιμα συστήματα της βιομηχανίας. Τα υψηλά επίπεδα απόδοσης των SoCs είναι ιδανικά για τη διαφοροποίηση των εφαρμογών υψηλής έντασης, όπως είναι οι μονάδες ελέγχου βιομηχανικού κινητήρα, το πρωτόκολλο γεφύρωσης (protocol bridging), ο μετατροπέας βίντεο και οι κάρτες καταγραφής, αλλά και φορητές συσκευές.

Τα SoCs ανήκουν σε ένα ευρύ φάσμα προγραμματιζόμενης λογικής με πολλές λειτουργίες στο επίπεδο συστήματος ενός dual-core Cortex ARM επεξεργαστή, A9 hard processor σύστημα επεξεργαστή (HPS), ενσωματωμένα περιφερειακά, ελεγκτές μνήμης πολλαπλών θυρών, σειριακοί πομποδέκτες, και υλικό δίαυλου ηλεκτρονικού υπολογιστή (PCI Express θύρες - Peripheral Component Interconnect Express Ports) [11].

Η αρχιτεκτονική του Cyclone V SoC φαίνεται στο παρακάτω σχήμα και στον πίνακα 4.1 ενώ στις ενότητες που ακολουθούν αναλύονται κάποια βασικά χαρακτηριστικά του.



Σχήμα 4.1 : Αρχιτεκτονική του Cyclone V SoC

Device	Όλες οι Cyclone V SoC devices (SE, SX, ST)
Processor	ARM Cortex™-A9 MPCore™ processor with ARM CoreSight™ debug and trace technology (single and dual-core variants) • 925 MHz CPU clock rate in -C6 speed grade • 800 MHz CPU clock rate in -C7, -I7 speed grades • 700 MHz CPU clock rate in -A7 speed grade • 600 MHz CPU clock rate in -C8 speed grade
Coprocessors	ARM Neon™ media processing engine with vector floating-point (VFP) v3 double-precision floating-point unit for each processor, snoop control unit (SCU), acceleration coherency port (ACP)
Level 1 cache	32 KB L1 instruction cache, 32 KB L1 data cache
Level 2 cache	512 KB shared L2 cache
On-chip memory	64 KB on-chip RAM, 64 KB on-chip ROM
HPS hard memory controller	Multiport SDRAM controller with support for DDR2, DDR3, DDR3L and LPDDR2 with optional error correction code (ECC) support 400 MHz/800 Mbps external memory interface User-configurable memory width of 8, 16, 16+ECC, 32, 32+EEC Up to 4 GB address range with built-in memory protection control
Quad serial peripheral interface (SPI) flash controller	Supports SPIx1, SPIx2, or SPIx4 (quad SPI) serial NOR flash devices Up to 4 chip selects
SD/SDIO/MMC controller	Supports SD, eSD, SDIO, eSDIO, MMC, eMMC, and CE-ATA with integrated DMA
NAND flash controller	Supports 8 bit ONFI 1.0 NAND flash devices Programmable-hardware ECC for single-level cell (SLC) and multilevel cell (MLC) devices
Ethernet media access controller (EMAC)	2 x 10/100/1000 EMAC with RGMII external PHY interface and integrated DMA
USB On-The-Go controller (OTG)	2 x USB 2.0 OTG controllers with ULPI external PHY interface and integrated DMA
UART controller	2 x UART 16550 compatible
SPI controller	2 x SPI masters 2 x SPI slaves
I²C controller	4 x I²C

CAN controller	2 x CAN Protocol specification 2.0 (A and B)
General-purpose I/O (GPIO)	Up to 71 GPIO and 14 input-only pins, with digital de-bounce and configurable interrupt mode
Direct memory access (DMA) controller	8-channel DMA Supports flow control with 31 peripheral handshake interfaces
Timers	Private interval and watchdog timer for each processor Global timer for processor subsystem 4 X general-purpose timers 2 X watchdog timers
Maximum HPS I/O	181
HPS phased-lock loops (PLLs)	3

Πίνακας 4.1 : Περιγραφή Cyclone V SoC Family Hard Processor συστήματος

4.1.2.1. ARM-Based Hard Processor σύστημα επεξεργαστή (HPS)

Το Cyclone V SoC HPS αποτελείται από ένα dual-core επεξεργαστή Cortex ARM -A9 MPCore, ένα σύνολο περιφερειακών και έναν ελεγκτή μνήμης πολλαπλών σημείων που βασίζεται στη λογική του FPGA, παρέχοντας την ευελιξία της προγραμματιζόμενης λογικής και την εξοικονόμηση κόστους λόγω:

- Του απλού ή διπύρηνου επεξεργαστή με πάνω από 925 MHz μέγιστη συχνότητα.
- Των περιφερειακών με ενσωματωμένο κώδικα για εκτέλεση λειτουργιών, που αλλιώς θα έπρεπε να αναπτυχθούν μέσω προγραμμάτων, απελευθερώνοντας έτσι περισσότερους πόρους του FPGA και καθιστώντας τους διαθέσιμους για εξειδικευμένες εφαρμογές και μειώνοντας την κατανάλωση ενέργειας.
- Του ελεγκτή μνήμης πολλαπλών θυρών που από κοινού με τον επεξεργαστή και τη λογική των FPGA, υποστηρίζει τις δυναμικές μνήμες τυχαίας προσπέλασης (Dynamic Random Access Memory- DRAM) DDR2, DDR3, και LPDDR2 (Low Power Double Data Rate Synchronous) και συσκευές με ενσωματωμένο κώδικα διόρθωσης σφαλμάτων (error correction code – ECC) για την επίτευξη υψηλής αξιοπιστίας και ασφάλειας για τις κρίσιμες εφαρμογές.

4.1.2.2. Υψηλό εύρος ζώνης διασύνδεσης

Οι διάδρομοι δεδομένων (datapaths) υψηλής απόδοσης μεταξύ των HPS και FPGAs παρέχουν απόδοση που δεν είναι εφικτή σε διασύνδεση δύο-chip. Αυτή η ολοκλήρωση παρέχει:

- Πάνω από 100 Gbps εύρος ζώνης κορυφής.
- Ολοκληρωμένη συνοχή των δεδομένων.

Σημαντική εξοικονόμηση ενέργειας του συστήματος, εξαλείφοντας τις εξωτερικές διασυνδέσεις εισόδου - εξόδου (I/O) μεταξύ του επεξεργαστή και του FPGA.

4.1.2.3. Flexible FPGA Fabric

Η λογική των FPGA επιτρέπει τη διαφοροποιήση του συστήματος με την εφαρμογή custom IP ή off-the-shelf προρυθμισμένο IP από την Altera ή τους συνεργάτες της. Αυτό επιτρέπει:

- Τη γρήγορη προσαρμογή στα μεταβαλλόμενα και ποικίλα πρότυπα πρωτοκόλλων και στις διασυνδέσεις.
- Την προσθήκη προσαρμοσμένου υλικού στο FPGA για την επιτάχυνση του χρόνου κρίσιμων αλγορίθμων και τη δημιουργία ενός ανταγωνιστικού πλεονεκτήματος.
- Τη γρήγορη ανάπτυξη ενός επεξεργαστή ARM χωρίς τον εκτεταμένο σχεδιασμό, την επαλήθευση, και τα μη-επαναλαμβανόμενα (non-recurring ,NRE) κόστη που απαιτούνται στα ASICs.

4.1.2.4. Αρχιτεκτονικά ζητήματα (Architecture Matters)

Επειδή το Cyclone V SoCs ενσωματώνει πολλές ενότητες πνευματικής ιδιοκτησίας (IP blocks) δίδεται η δυνατότητα να μειώσει κανείς το συνολικό κόστος του συστήματος, την ενέργεια και το χρόνο σχεδιασμού. Τα SoCs είναι κάτι περισσότερο από το άθροισμα ή τα τμήματά τους. Είναι ο τρόπος με τον οποίο τα συστήματα επεξεργασίας και τα FPGA συστήματα συνεργάζονται σε μεγάλο βαθμό για την επίτευξη της βέλτιστης απόδοσης, αξιοπιστίας και ευελιξίας του συστήματος. Τα SoCs της Altera έχουν σχεδιαστεί για να:

- Διατηρήσουν την ευελιξία του επεξεργαστή εκκίνησης των FPGA ρυθμίσεων, την απόκριση του συστήματος για επαναφορά του επεξεργαστή, και τις διεπαφές μνήμης ανεξάρτητες από τη λύση των δύο chip.
- Διατηρήσουν την ακεραιότητα των δεδομένων και την αξιοπιστία με την ενσωματωμένη διόρθωση σφάλματος.
- Προστατέψουν την DRAM μνήμη από τον επεξεργαστή και το FPGA με μια ολοκληρωμένη μονάδα προστασίας μνήμης.
- Ενεργοποιήσουν το σύστημα εντοπισμού σφαλμάτων με FPGA-προσαρμοσμένη αποσφαλμάτωση (debugging) της Altera για έλεγχο του συνόλου της συσκευής.

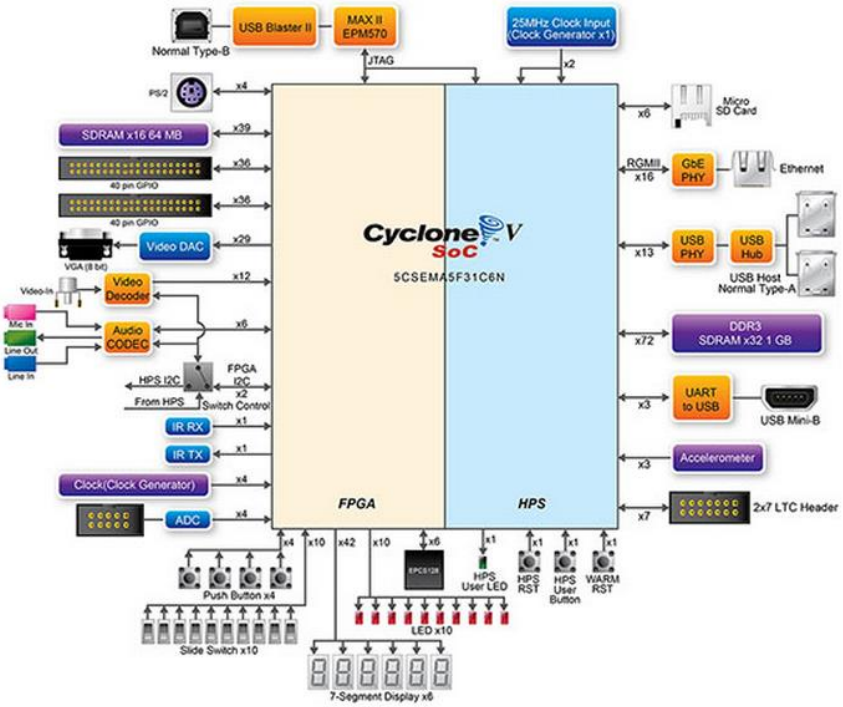
4.1.3. DE1-SoC Board

Το DE1-SoC Development Kit παρουσιάζει μια ισχυρή πλατφόρμα σχεδιασμού υλικού χτισμένο γύρω από το Altera System-on-Chip (SoC) FPGA, το οποίο συνδυάζει τους τελευταίους dual-core Cortex-A9 ενσωματωμένους πυρήνες με προγραμματιζόμενη λογική, και προσφέρει απόλυτη ευελιξία σχεδιασμού. Οι χρήστες μπορούν έτσι να εκμεταλλευτούν τη δύναμη της μεγάλης παραμετροποίησης σε συνδυασμό με ένα υψηλής απόδοσης και χαμηλής ισχύος σύστημα.

Τα SoC της Altera ενσωματώνουν ένα hard processor σύστημα επεξεργαστή βασισμένο σε ARM που αποτελείται από τον επεξεργαστή, τα περιφερειακά και διεπαφές μνήμης συνδυασμένα αρμονικά στο FPGA, χρησιμοποιώντας μια διασύνδεση υψηλού εύρους ζώνης. Η αναπτυξιακή πλακέτα DE1-SoC περιλαμβάνει υλικό όπως είναι η υψηλής ταχύτητας DDR3 μνήμη, βίντεο και δυνατότητες ήχου, δικτύωση Ethernet και πολλά άλλα.

Το DE1-SOC Development Kit περιέχει όλα τα συστατικά που απαιτούνται για να χρησιμοποιηθεί το board σε συνδυασμό με έναν υπολογιστή με λειτουργικό Microsoft Windows XP ή νεότερη έκδοση (64-bit λειτουργικό σύστημα και Quartus II 64-bit απαιτούνται για να μεταγλωττιστούν προγράμματα για DE1-SoC) [12].

Μερικά από τα χαρακτηριστικά του συγκεκριμένου board φαίνονται παρακάτω.



Σχήμα 4.2 : Μπλοκ διάγραμμα του DE1-SOC Board

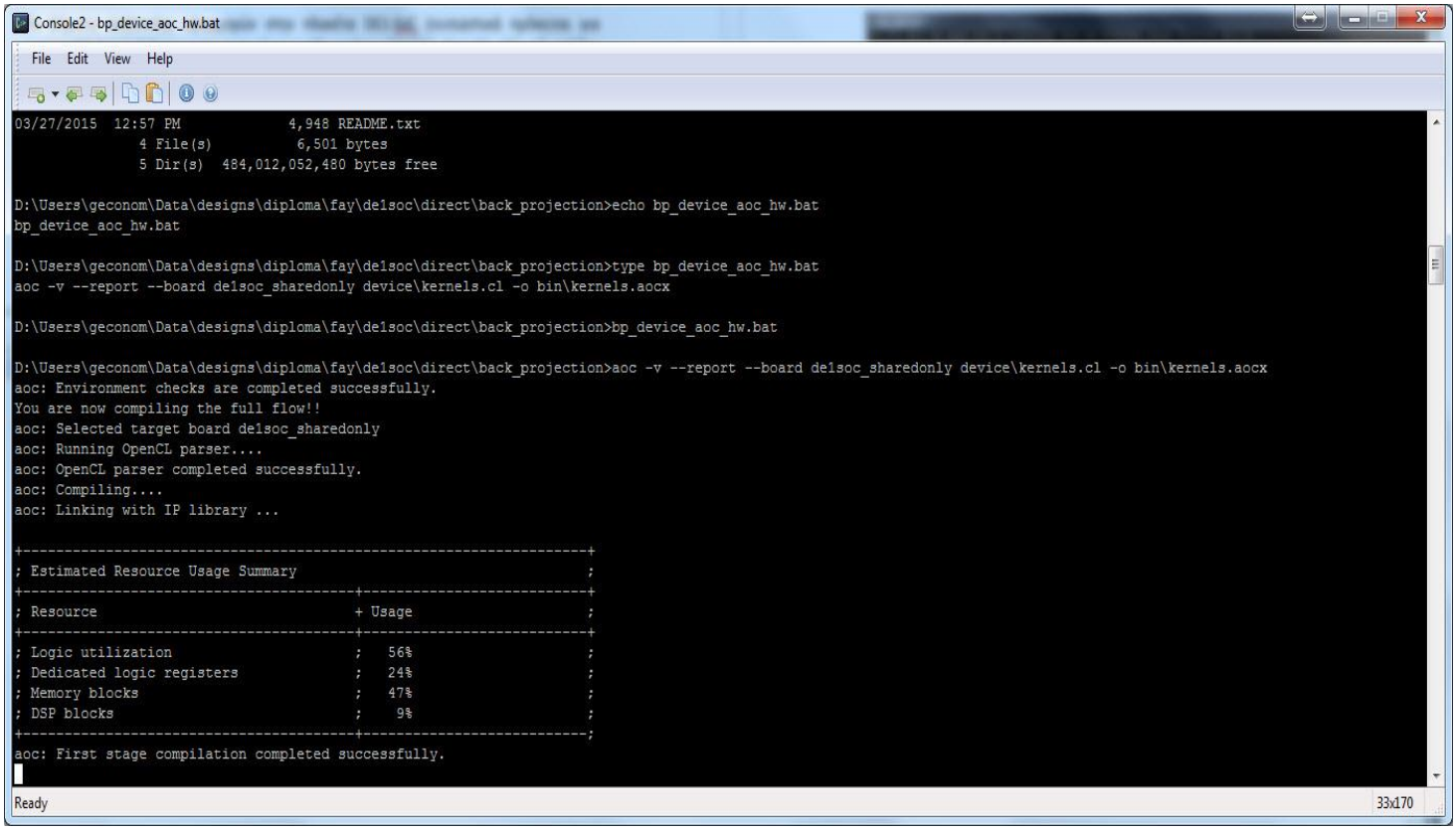
FPGA Device	• Cyclone V SoC 5CSEMA5F31C6 Device • Dual-core ARM Cortex-A9 (HPS) • 85K Programmable Logic Elements • 4,450 Kbits embedded memory • 6 Fractional PLLs • 2 Hard Memory Controllers
Configuration and Debug	• Serial Configuration device – EPCS128 on FPGA • On-Board USB Blaster II (Normal type B USB connector)
Memory Device	• 64MB (32Mx16) SDRAM on FPGA • 1GB (2x256Mx16) DDR3 SDRAM on HPS • Micro SD Card Socket on HPS
Communication	• Two Port USB 2.0 Host (ULPI interface with USB type A connector) • USB to UART (micro USB type B connector) • 10/100/1000 Ethernet • PS/2 mouse/keyboard • IR Emitter/Receiver
Connectors	• Two 40-pin Expansion Headers (voltage levels: 3.3V) • One 10-pin ADC Input Header • One LTC connector (One Serial Peripheral Interface (SPI) Master ,one I2C and one GPIO interface)
Display	• 24-bit VGA DAC
ADC	• Fast throughput rate: 1 MSPS • Channel number: 8 • Resolution: 12 bits • Analog input range : 0 ~ 2.5 V or 0 ~ 5V as selected via the RANGE bit in the control regist
Switches, Buttons and Indicators	• 4 User Keys (FPGA x4) • 10 User switches (FPGA x10) • 11 User LEDs (FPGA x10 ; HPS x 1) • 2 HPS Reset Buttons (HPS_RST_n and HPS_WARM_RST_n) • Six 7-segment displays
Power	• 12V DC input

Πίνακας 4.2 : Χαρακτηριστικά του DE1-SoC Board

4.1.3.1. Βήματα ανάπτυξης εφαρμογών στην πλακέτα DE1-SoC

Τα βήματα ανάπτυξης εφαρμογών στην πλακέτα DE1-SoC σε περιβάλλον Microsoft Windows είναι τα ακόλουθα:

1. Σε ένα terminal του υπολογιστή μεταγλωττίζουμε τον kernel που θέλουμε, με την εντολή π.χ. aoc -v --report --board de1soc_sharedonly device\kernels.cl -o bin\kernels.aocx, όπως φαίνεται και παρακάτω.



Σχήμα 4.3 : Μεταγλώτιση του kernel

2.Σε ένα άλλο terminal, που ανοίγει με την εφαρμογή Altera SoC EDS Command Shell και προετοιμάζει όλες τις μεταβλητές συστήματος για να υποστηρίζεται cross compilation για επεξεργαστή ARM, μεταγλωττίζουμε την κεντρική εφαρμογή (main) που θα καλεί το kernel, πηγαίνοντας στο κατάλληλο directory και εκτελώντας την εντολή make.

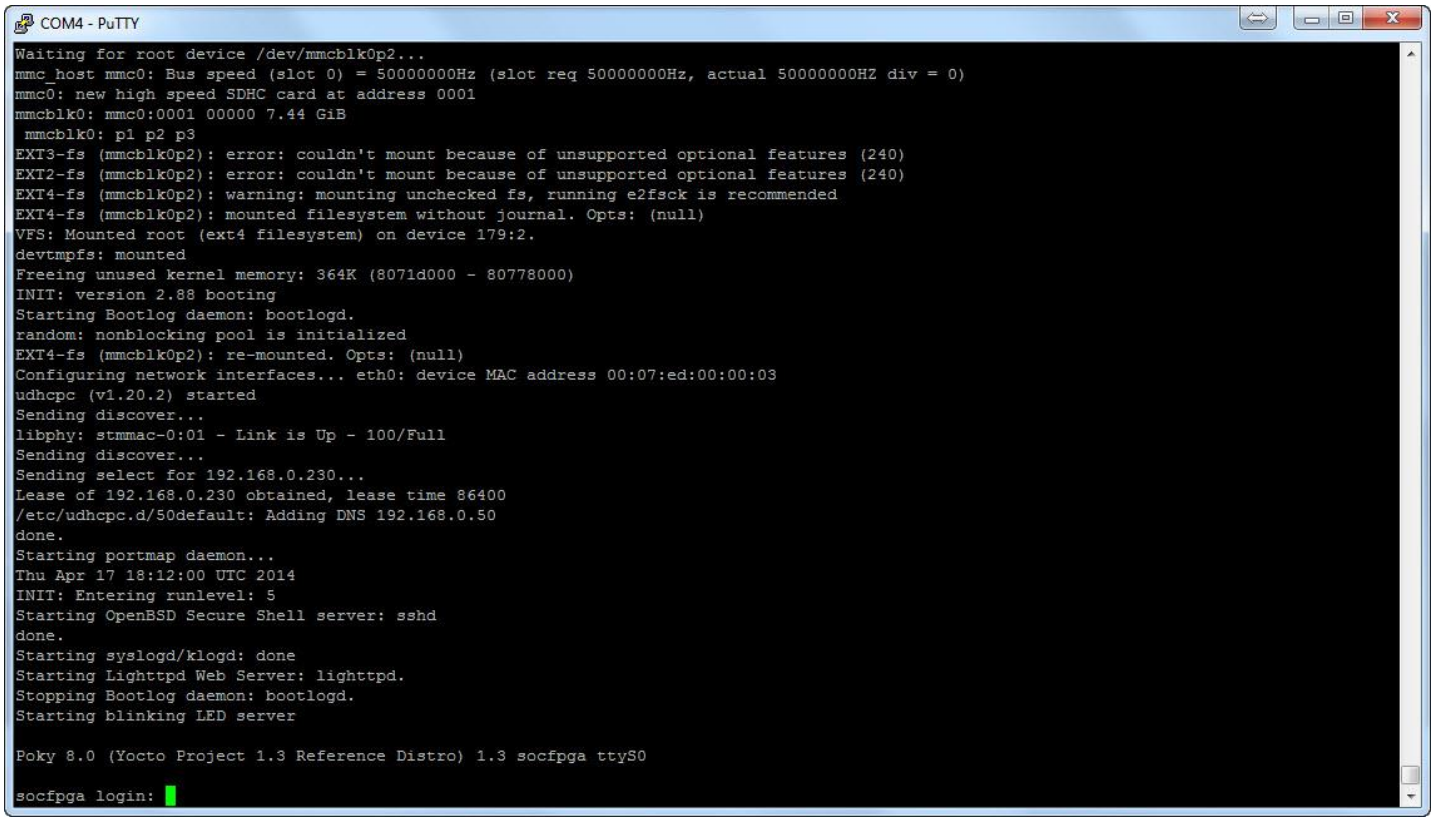
3.Αντιγράφουμε σε μια κάρτα microSD το Linux με υποστήριξη OpenCL που δίνει η εταιρεία (αρχείο DE1-SoC_openCL_BSP.zip και από εκεί το linux_sd_card_image.zip, με την απαραίτητη αποσυμπίεση).

4.Φτιάχνουμε τους διακόπτες επιλογής MSEL = 01010.

5.Τοποθετούμε την κάρτα microSD στη σχετική υποδοχή της πλακέτας, συνδέουμε την πλακέτα στο ρεύμα και στη θύρα προγραμματισμού USB και πατάμε το διακόπτη On/Off.

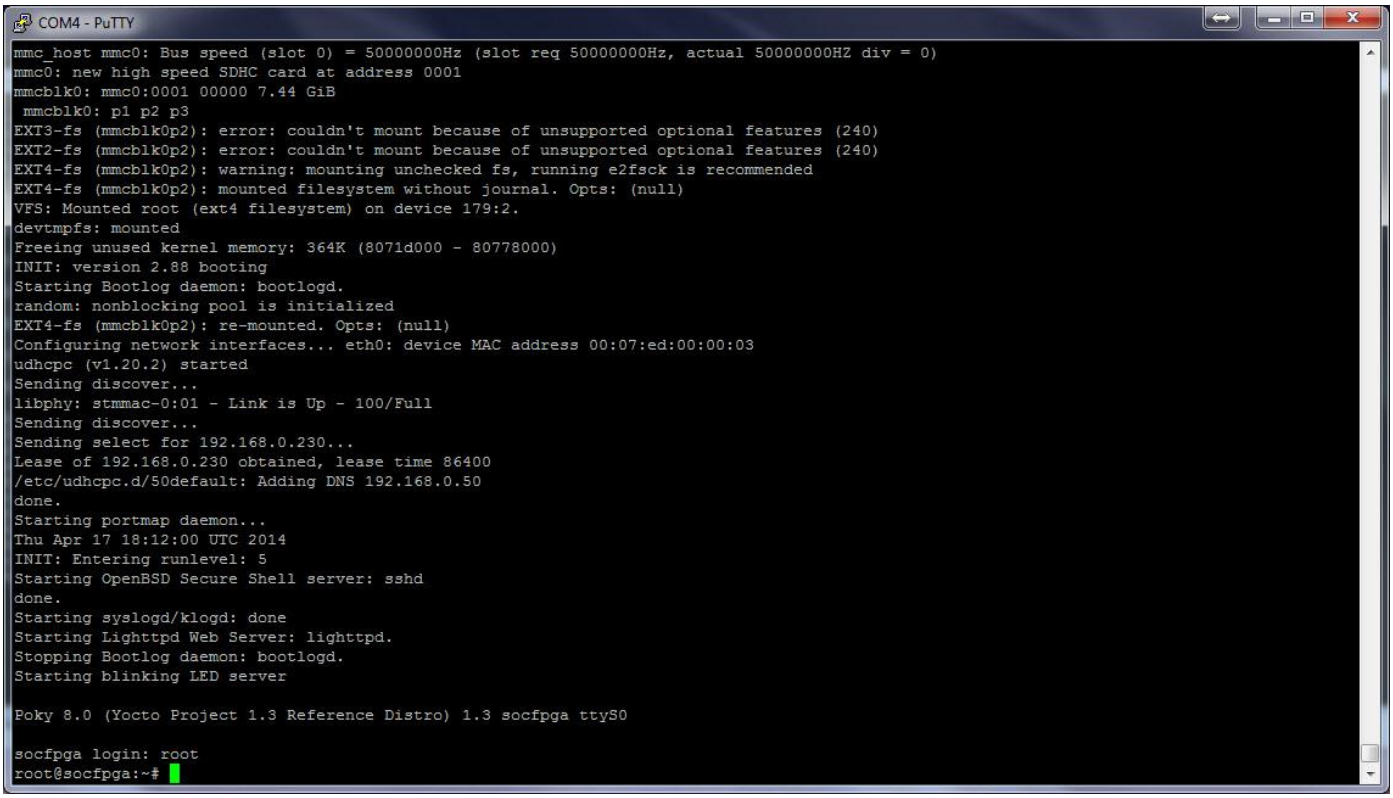
6.Σε περίπτωση που συνδέουμε την πλακέτα για πρώτη φορά, γίνεται εγκατάσταση μιας συσκευής USB2UART και εμφανίζεται μια νέα σειριακή θύρα, με την οποία μπορεί να γίνει επικοινωνία του υπολογιστή με την πλακέτα.

7.Μπορούμε με το πρόγραμμα putty και τις κατάλληλες παραμέτρους να συνδεθούμε με την πλακέτα. Η αρχική οθόνη είναι η ακόλουθη.



Σχήμα 4.4 : Σύνδεση στην πλακέτα

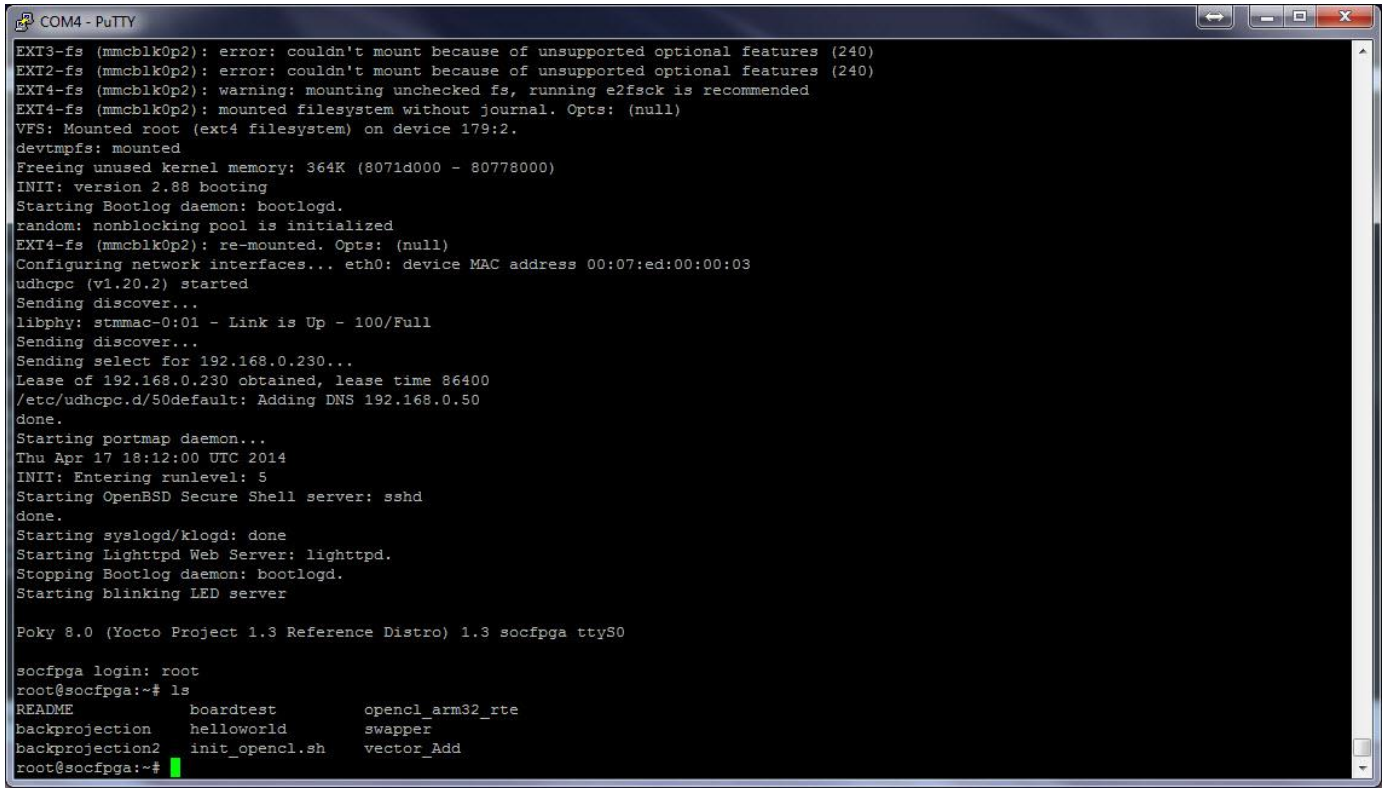
8.Βάζουμε για username το root και κανένα password, όπως παρακάτω.



Σχήμα 4.5 : Είσοδος στην πλακέτα

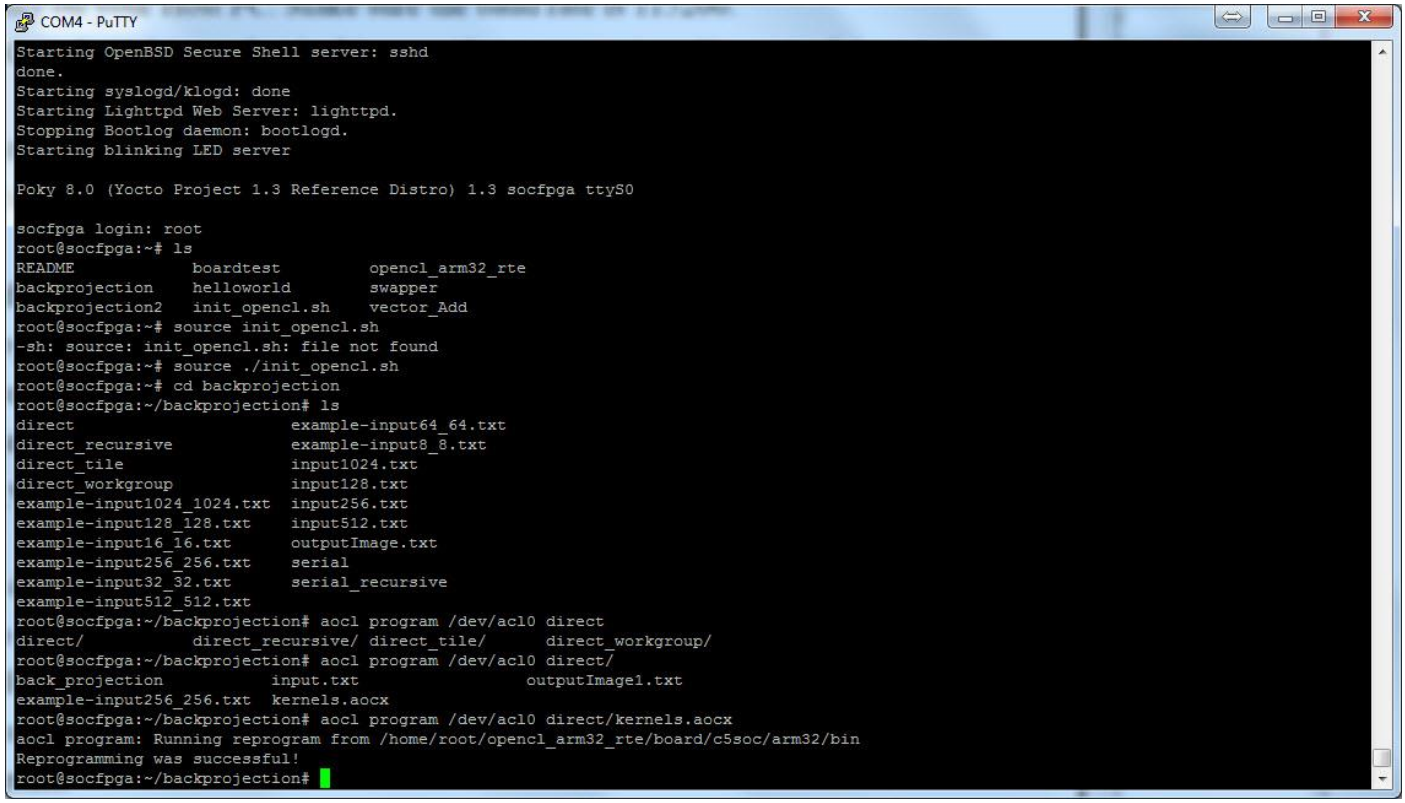
9.Ο κατάλογος στον οποίο βρισκόμαστε περιλαμβάνει τα αρχεία που φαίνονται παρακάτω.

10.Στέλνουμε τα μεταγλωττισμένα kernel και κεντρική εφαρμογή στην πλακέτα. Ένας τρόπος να το κάνουμε αυτό είναι να συνδέσουμε καλώδιο Ethernet, να γράψουμε στο linux της πλακέτας ifconfig eth0 192.168.0.100 (ή οποιαδήποτε άλλη διεύθυνση) και να χρησιμοποιήσουμε την εντολή pscp kernel.aocx root@192.168.0.100:/home και pscp main.exe root@192.168.0.100:/home, ή κάτι σχετικό.



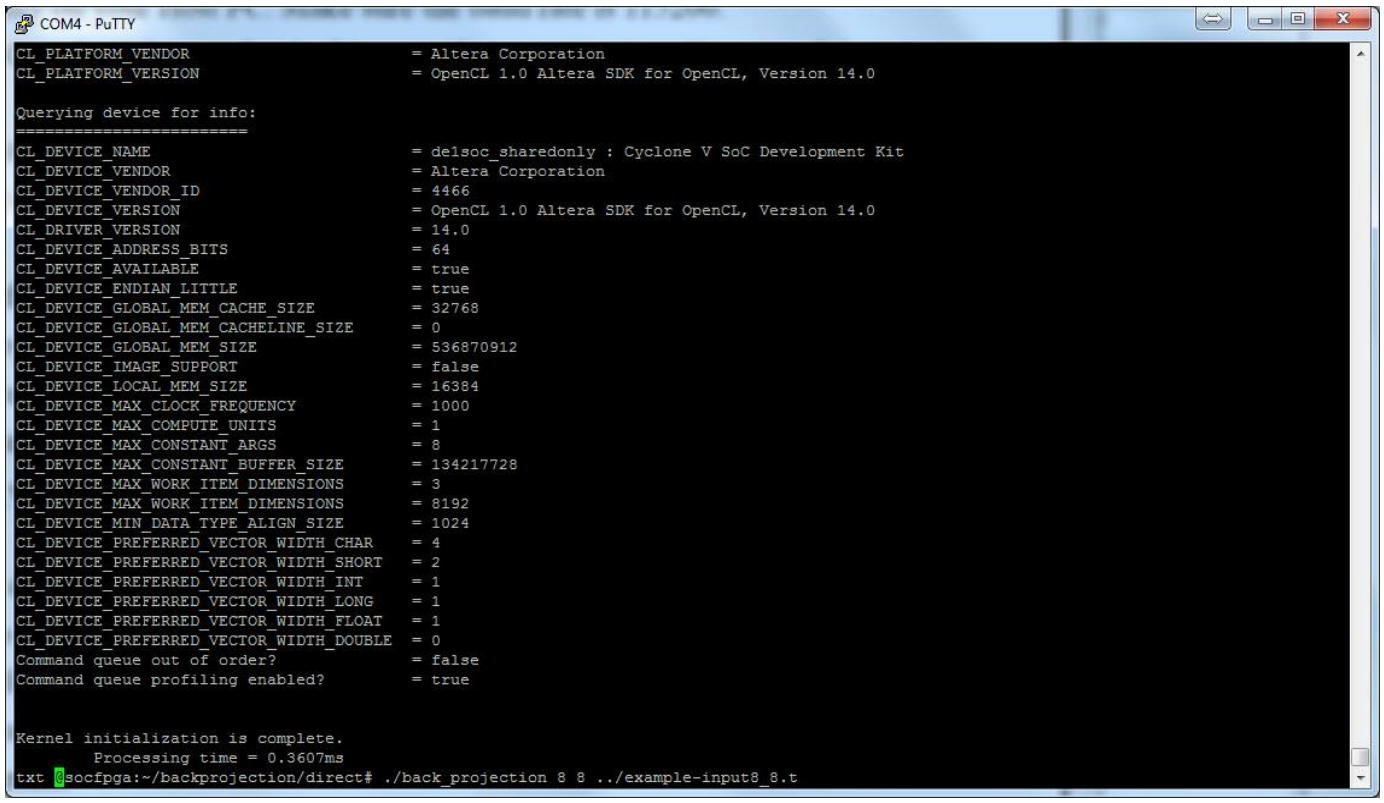
Σχήμα 4.6 : Αποστολή των μεταγλωττισμένων kernel και κεντρική εφαρμογή στην πλακέτα

11.Τρέχουμε πρώτα την εντολή source ./init_opencl.sh για να είναι διαθέσιμες οι εφαρμογές της Altera και στη συνέχεια προγραμματίζουμε το FPGA κομμάτι της SoC συσκευής με την εντολή aocl program /dev/acl0 kernel.aocx, όπου kernel.aocx το οποιοδήποτε OpenCL kernel έχουμε προηγουμένως μεταγλωττίσει (εκτελέσιμη μορφή). Ένα παράδειγμα φαίνεται παρακάτω.



Σχήμα 4.7 : Εκτέλεση της εντολής source

12.Τέλος, τρέχουμε την κεντρική εφαρμογή που καλεί τον kernel, όπως φαίνεται παρακάτω.



Σχήμα 4.8 : Εκτέλεση κεντρικής εφαρμογής που καλέι τον kernel

4.2. Περιγραφή χαρακτηριστικών CPU1

Στην ενότητα αυτή παρουσιάζονται κάποια βασικά χαρακτηριστικά της CPU1 στην οποία έγιναν μετρήσεις για τον αλγόριθμο οπισθοπροβολής. [13]

Clock speed	2.5 GHz
Turbo clock speed	3.5 GHz
Cores	Quad core
Is unlocked	No
Is hyperthreaded	Yes
Architecture	x86-64
Threads	8 threads
L2 cache	1 MB
L2 cache per core	0.25 MB/core
L3 cache	6 MB
L3 cache per core	1.5 MB/core
Manufacture process	22 nm
Max CPUs	1
Overclocked clock speed	3.43 GHz
Overclocked clock speed (Water)	2.5 GHz
Overclocked clock speed (Air)	3.43 GHz
TDP	47W
Performance per watt	20.77 pt/W
Typical power consumption	38.19W
Memory controller	Built-in
Memory type	DDR3
Channels	Dual Channel
Supports ECC	No
Maximum bandwidth	12,800 MB/s

Πίνακας 4.3: Χαρακτηριστικά της CPU1 – Intel core i7

4.3. Περιγραφή χαρακτηριστικών CPU2

Στην ενότητα αυτή παρουσιάζονται κάποια βασικά χαρακτηριστικά της CPU2 στην οποία έγιναν μετρήσεις για τον αλγόριθμο οπισθοπροβολής [14].

Clock speed	2.4 GHz
Turbo clock speed	2.93 GHz
Cores	Dual core
Is unlocked	No
Is hyperthreaded	Yes
Architecture	x86-64
Threads	4 threads
L2 cache	0.5 MB
L2 cache per core	0.25 MB/core
L3 cache	3 MB
L3 cache per core	1.5 MB/core
Max CPUs	1
Overclocked clock speed	2.86 GHz
Overclocked clock speed (Water)	2.4 GHz
PassMark (Overclocked)	1,455.5
Overclocked clock speed (Air)	2.86 GHz
TDP	35W
Performance per watt	10.65 pt/W
Typical power consumption	28.44W
Memory controller	Built-in
Memory type	DDR3-1066 DDR3-800 DDR3
Channels	Dual Channel
Supports ECC	No
Maximum bandwidth	17,066.66 MB/s

Πίνακας 4.4: Χαρακτηριστικά της CPU2 – Intel core i5

4.4. Περιγραφή χαρακτηριστικών GPU1

Στην ενότητα αυτή παρουσιάζονται κάποια βασικά χαρακτηριστικά της GPU1 στην οποία έγιναν μετρήσεις για τον αλγόριθμο οπισθοπροβολής.

GPU Name:	Venus
GPU Variant:	Venus XT
Process Size:	28 nm
Transistors:	1,500 million
Die Size:	123 mm^2
GPU Clock:	725 MHz
Boost Clock:	775 MHz
Memory Clock:	1125 MHz 4500 MHz effective
GPU Clock:	725 MHz
Boost Clock:	775 MHz
GPU Clock:	725 MHz
Boost Clock:	775 MHz
Memory Clock:	1125 MHz 4500 MHz effective
Memory Size:	2048 MB
Memory Type:	GDDR5
Memory Bus:	128 bit
Bandwidth:	72.0 GB/s
Shading Units:	640
TMUs:	40
ROPs:	16
Compute Units:	10
Pixel Rate:	11.6 GPixel/s
Texture Rate:	29.0 GTexel/s
Floating-point performance:	928.0 GFLOPS

Πίνακας 4.5: Χαρακτηριστικά της GPU1 – AMD FirePro M5100

4.5. Περιγραφή χαρακτηριστικών GPU2

Στην ενότητα αυτή παρουσιάζονται κάποια βασικά χαρακτηριστικά της GPU2 στην οποία έγιναν μετρήσεις για τον αλγόριθμο οπισθοπροβολής [15].

CUDA Cores	16
Processor Clock (MHz)	1530
Gigaflops	73
Memory Clock	Up to 800 (DDR3), Up to 800 (GDDR3) MHz
Standard Memory Config	Up to 1GB
Memory Interface Width	64-bit
Bus Support	PCI-E 2.0
Supported Technologies	CUDA
Power Management	8.0
Multi Monitor	Yes
Maximum Digital Resolution	2560x1600
Maximum VGA Resolution	2048x1536
Standard Display Connectors	DisplayPort HDMI VGA Dual Link DVI Single Link DVI

Πίνακας 4.6 : Χαρακτηριστικά της GPU2- NVIDIA GeForce 310M

5.Προτεινόμενες παράλληλες υλοποιήσεις

Στην ενότητα 5.1 παρουσιάζεται αναλυτικά ο αλγόριθμος που χρησιμοποιήθηκε για την παρούσα διπλωματική, βάσει του οποίου παρουσιάζονται οι τέσσερις προτεινόμενες υλοποιήσεις παράλληλου κώδικα της φιλτραρισμένης οπισθοπροβολής. Οι υλοποιήσεις αυτές παρουσιάζονται αναλυτικά για την κατανόηση του παραλληλισμού του κώδικα και συνοδεύονται από τα αποτελέσματα για κάθε μια από τις πλατφόρμες δοκιμής όπως αυτές αναλύθηκαν στο κεφάλαιο 4.

5.1. Περιγραφή βασικών αλγορίθμων

5.1.1. Βασικός direct αλγόριθμος οπισθοπροβολής

Ο αλγόριθμος που χρησιμοποιείται για την εκπόνηση της συγκεκριμένης διπλωματικής εργασίας βασίζεται στην θεωρητική ανάλυση που έγινε στο κεφάλαιο 3 με τη διαφορά ότι για να πραγματοποιηθεί η οπισθοπροβολή (φιλτραρισμένη και μη) πρέπει να διακριτοποιηθεί η διαδικασία ανακατασκευής της εικόνας. Αυτό συνεπάγεται ότι χρησιμοποιείται διακριτός υπολογισμός που αντιμετωπίζει την εικόνα ως έναν τετραγωνικό πίνακα με $N \times N$ εικονοστοιχεία (pixel).

Έτσι, η διακριτή οπισθοπροβολή πραγματοποιείται για κάθε εικονοστοιχείο $f(m,n)$ ως το άθροισμα των τιμών προβολής για όλες τις γωνίες θ:

$$f(m,n) = \sum_{\theta} \hat{f}(mcos\theta + vsin\theta)\Delta\theta \qquad (5.1)$$

όπου θ είναι η εκάστοτε γωνία προβολής και $\Delta\theta$ είναι η γωνιακή απόσταση μεταξύ διαδοχικών προβολών.

Ακόμα, θεωρείται ότι ως είσοδος λαμβάνεται αρχείο με P στήλες ή αλλιώς προβολές (projections). Συνολικά, το αρχείο περιλαμβάνει $X \times P$ τιμές δηλαδή X τιμές σε κάθε στήλη. Οι τιμές X και P παρουσιάζουν τις τιμές της θέσης X για κάθε μια από τις γωνίες λήψης των προβολών θ.

Στόχος του αλγορίθμου είναι ο υπολογισμός των τιμών όλων των εικονοστοιχείων της εικόνας ή αλλιώς ο υπολογισμός της $f(m,n)$ για κάθε εικονοστοιχείο. Ο υπολογισμός αυτός στο εξής θα αναφέρεται ως απευθείας μέθοδος ή αλλιώς $direct$ μέθοδος. Σκοπός όλων των υλοποιήσεων είναι ο υπολογισμός της direct μεθόδου, δηλαδή ο υπολογισμός όλων των εικονοστοιχειων για $m,n \in [0,N]$ όπου N η διάσταση του τετραγωνικού πίνακα. Η direct μέθοδος κάνει αυτόν τον υπολογισμό εκτελώντας $N \times N$ φορές την $sumsino$ συνάρτηση όπως φαίνεται και στο παρακάτω σχήμα.

```
void direct(sinograms* g,int size,float* tau,image* ans)
{
  int m;        //x direction
  int n;        //y direction
  float temp;

  char* filename = "output.txt";
  FILE* output;

  float* curRow;
  float middle = ((float)(g->size))/2-0.5;
  float halfSize = 0.5*size;
  int numSino = g->num;
  float scale = M_PI/numSino;

  //for each pixel in the image
  for(n=0;n<size;n++){
    curRow = (ans->pixel)[n];
    for(m=0;m<size;m++){
      temp = sumSino(g,m,n,tau,halfSize,middle);
      curRow[m] = temp*scale;

    } //end for n
  } //end for m

}
```

Σχήμα 5.1 : Κώδικας της Direct συνάρτησης του σειριακού κώδικα της οπισθοπροβολής

Η συνάρτηση *sumsino* πραγματοποιεί τον υπολογισμό της τιμής καθενός εικονοστοιχείου. Έτσι, σε πρώτη φάση λαμβάνει ως είσοδο τις συντεταγμένες του κάθε εικονοστοιχείου (m, n) και υπολογίζει την πραγματική τους θέση στο χώρο $(realm, realn)$ ώστε στη συνέχεια να πραγματοποιηθεί ο υπολογισμός με βάση τις πραγματικές θέσεις των εικονοστοιχείων. Το άθροισμα υπολογίζεται από όλες τις προβολλες εισόδου P, για κάθε είσοδο της κάθε προβολής και στη συνέχεια πολλαπλασιάζεται με μια τίμη ώστε να γίνει παρεμβολή και να βρεθεί η ακριβής τιμή για το συγκεκριμένο εικονοστοιχείο (interpolation).

```
float sumSino(sinograms* g,int m,int n,float* tau,float
halfSize,float middle)
{
  int p;
  int k;
  int P=g->num;
  int sinoSize;

  float i;
  float sum=0;
  float* curSino;

  float temp;

  float realm=m-halfSize+0.5;     //we need to adjust m,n in order to
change them from array indices to
  float realn=n-halfSize+0.5;     //coordinates on the target image.

  float scaledRealm = realm*oneOverT;
  float scaledRealn = realn*oneOverT;

  float lowerXLimit;
  float upperXLimit;
  float lowerYLimit;
  float upperYLimit;

  float spXSupport;
  float spYSupport;
  float intpSupport;

  float curr;
  float lowerKBound;
  float upperKBound;

  int lowerKBound2;
  int upperKBound2;

  float theta;

  float exactRadialPosition;

  intpSupport = RADIAL_SUPPORT;


  sinoSize = g->size;

  //for each sinogram
  for(p=0;p<P;p++){

    curSino = g->sino[p];
    exactRadialPosition = scaledRealm*(g->cosine)[p] +
scaledRealn*(g->sine)[p];

    exactRadialPosition -= tau[p]-middle;
    //exactRadialPosition is now in terms of array index! (although
it's still a float)

    //since spSupport is zero
    lowerKBound = exactRadialPosition-intpSupport;
    upperKBound = exactRadialPosition+intpSupport;
```

```
lowerKBound2 = maxInt((int)(lowerKBound-ERROR_TOLERANCE)+1,0);
    upperKBound2 = minInt((int)(upperKBound),(g->size)-1);     //use
(int) instead of floorf, since upperKBound is a bound for array index
which is >=0

    //for nearest interpolator, this needs to be checked
    if(upperKBound2<lowerKBound2){
      upperKBound2 = lowerKBound2;
    }

    //for each entry in this sinogram
    for(k=lowerKBound2;k<=upperKBound2;k++){

      curr = curSino[k];

      temp = weight(exactRadialPosition,k);
      sum += curr*temp;

    }//end for k
  }//end for p
    //printf("%f\n",sum);
  return sum;
}
```

Σχήμα 5.2: Κώδικας sumsino συνάρτησης του σειριακού αλγορίθμου της οπισθοπροβολής

5.1.2. Αναδρομικός αλγόριθμος οπισθοπροβολής

Οι Basu και Bresler [16] έδειξαν ότι για συγκεκριμένες διαστάσεις της εικόνας, το πρόβλημα μπορεί να απλοποιηθεί αναδρομικά σε πρόβλημα υπο-εικόνων. Έδειξαν δηλαδή ότι δεν είναι απαραίτητο να χρησιμοποιηθούν όλα τα δεδομένα εισόδου για τον υπολογισμό των υπο-εικόνων. Έτσι, αν για παράδειγμα χωριστεί η εικόνα μας σε τέσσερις υποεικόνες των $N/2 \times N/2$ τότε μόνο οι μισές προβολές $P/2$ θα χρειάζονταν για τον υπολογισμό τους.

Με βάση την απόδειξη αυτή, οι Pipatsrisawat, Gacic, Franchetti, Püschel και Moura [17] πρότειναν μια υλοποίηση βασισμένη σε αναδρομή (recursion). Ο χρήστης θέτει σαν όρισμα εισόδου το βασικό μέγεθος (basesize) B μέχρι το οποίο εφαρμόζεται η αναδρομή και ο διαχωρισμός των απαραίτητων προβολών για τους υπολογισμούς και από εκείνο το σημείο και έπειτα εφαρμόζεται σε όλα τα δεδομένα η direct μέθοδος. Έτσι, επιτυγχάνεται ο υπολογισμός όλων των τιμών για κάθε block της εικόνας από τα οποία αυτή ανακατασκευάζεται.

Η διαδικασία αυτή παρουσιάζεται στον παρακάτω ψευδοκώδικα / σχήμα, ενώ στις σελίδες που ακολουθούν παρατίθεται ο κώδικας της recursive υλοποίησης.

BACKPROJECTION $(N, \tilde{f}(\rho, \theta))$

if $N \leq B$ **then**

return DIRECT BACKPROJECTION $(N, \tilde{f}(\rho, \theta))$

else

for $i = 1 \ldots 4$ **do**
$\widetilde{f}_i(\rho, \theta) \leftarrow$ SEGMENT$(\widetilde{f}_i(\rho, \theta)), i)$
$b_i(x, y) \leftarrow$ BACKPROJECTION

endfor
return TILE BLOCKS$(b_0(x, y), \ldots, b_3(x, y))$
endif

Σχήμα 5.3 : Ψευδοκώδικας Recursive μεθόδου του αλγορίθμου της οπισθοπροβολής

```
/* This function is the top-level function for recursive
backprojection.
 * It makes a call to 'direct' once size<=baseSize.
 */
void bp(sinograms* sino,int size,float* tau,image* ans)
{
  sinograms* newSino;
  image* subImage;

  int i,j,m,n,k,p,u,v;

  float* curRow;
  float temp1,temp2;

  int pp;

  int newSize;
  int numSino;
  int sinoSize;
  int offsetX;
  int offsetY;

  float shiftX,shiftY;

  float* nu_i;
  float* nu_temp;
  float nu_p,nu_2n;

  int P = sino->num;
  float angle;

  float sum=0;
  float shift;

  int newSinoSize;
  int newNumSino;
  int constShiftingFactor;

  if(size<=baseSize || sino->num <=1){
    //BASE CASE!!!!

    direct(sino,size,tau,ans);

    return;

  } else {
    //GENERAL CASE

    subImage = (image*)malloc(1*sizeof(image));          //prepare
subImage for recursive call
    newSize = size/2;
    numSino = (sino->num);
    sinoSize = (sino->size);

    subImage->size = newSize;
    (subImage->pixel) = (float**)malloc(newSize*sizeof(float*));
    shift = (float)size/4 * oneOverT;  //save some computations when
computing nu's by premultiply with 1/T

    newSinoSize = ceilf((float)sinoSize/2) + (sinoSize%4==3 ||
sinoSize%4==2?1:0);
```

```
constShiftingFactor = ((float)newSinoSize/2) - ((float)sinoSize/2);

    //for each part (that we break the image into)
    for(i=0;i<NUM_PARTS;i++){

      if(i==0){
    shiftX = shift;
    shiftY = shift;
      } else if(i==1){
    shiftX = -shift;
    shiftY = shift;
      } else if(i==2){
    shiftX = -shift;
    shiftY = -shift;
      } else {
    shiftX = shift;
    shiftY = -shift;
      }

      /************/

      nu_i = (float*)malloc(numSino*sizeof(float));
      nu_temp = (float*)malloc(numSino*sizeof(float));

      for(p=0;p<numSino;p++){
    //for each angle, compute nu_i,p

    nu_i[p] = nu(tau[p],shiftX,shiftY,(sino->sine)[p],(sino-
>cosine)[p]);//,oneOverT);
    nu_temp[p] = rintf(nu_i[p]);
      }
    newSino =
newSinoForNextIter2(sino,newSinoSize,constShiftingFactor,nu_i);
    pp = 1;
      //}

      /************/

      //manipulate the right part of the image
      if(i<2){
    // 0 and 1
    offsetY = newSize;

      } else {
    // 2 and 3
    offsetY = 0;
      }

      if(i==0 || i==3){
    // 0 and 3
    offsetX = newSize;
      } else {
    // 1 and 2
    offsetX = 0;
      }
```

```
for(m=0;m<newSize;m++){
    (subImage->pixel)[m] = (ans->pixel)[m+offsetY] + offsetX;
      }

      newNumSino = (newSino->num);
      //for each nu_p, compute <nu_p>
      for(p=0;p<newNumSino;p++){
    nu_i[p] = nu_i[pp*p]-nu_temp[pp*p];
      }


      //call bp recursively
      ////////////////////////////////////////////////
      bp(newSino,/*ker,*/newSize,nu_i,subImage);     //
      ////////////////////////////////////////////////


      freeSino(newSino);
      free(newSino);
      newSino = NULL;
      free(nu_i);
      free(nu_temp);
      nu_i = NULL;

    }//end for i

    free(subImage);
    free(subImage->pixel);
    //do not free subImage's pixels, because they are the output!

    return;

  }//end if not base case


}//end function bp
```

Σχήμα 5.4: Κώδικας σειριακού αναδρομικού αλγορίθμου της οπισθοπροβολής

5.1.3. Χρησιμοποιούμενες δομές

Οι δύο βασικές δομές που χρησιμοποιήθηκαν στην υλοποιήση του αλγορίθμου της οπισθοπροβολής, ήταν η δομή *image* στην οποία καταχωρείται κάθε εικονοστοιχείο της τελικής εικόνας αλλά και η δομή *sinogram* όπου αποθηκεύονται οι προβολές εισόδου. Οι δομές αυτές φαίνονται παρακάτω.

```
typedef struct{

  int size;
  float** pixel;                //pixel[0] refers to the lowest row
of the image, pixel[i][0] refers to the
                                  //leftmost column of the image

} image;
```

Σχήμα 5.5 : Δομή image στην οποία φυλάσσονται οι τιμές των εικονοστοιχείων εξόδου

Η δομή *image* αποτελείται από μια μεταβλητή *size* στην οποία καταχωρείται το μέγεθος της εικόνας εξόδου το οποίο είναι η μία εκ των δύο διαστάσεων της εικόνας καθώς ο πίνακας είναι τετραγωνικός. Ακόμα στον πίνακα *pixel* αποθηκεύονται οι τιμές των εικονοστοιχείων έχοντας ως $pixel[0]$ την τελευταία σειρά της εικόνας και $pixel[i][0]$ την αριστερή στήλη της εικόνας.

```
typedef struct{

  int num;
  int size;
  float T;
  float** sino;
  float* sine;
  float* cosine;

} sinograms;
```

Σχήμα 5.6 : Δομή *sinogram* στην οποία φυλάσσονται οι τιμές των προβολών εισόδου

Η δομή *sinogram* αποτελείται από τη μεταβλητή *num* που παρουσιάζει τον αριθμό των προβολών εισόδου ή αλλιώς τον αριθμό των γωνιών από τις οποίες έχουν ληφθεί οι προβολές, τη μεταβλητή *size* που δείχνει το μήκος της κάθε προβολής και τον αριθμό T οποίος είναι η αναλογία μεταξύ του φυσικού μεγέθους μιας προβολής και του φυσικού μεγέθους ενός εικονοστοιχείου της εικόνας. Ακόμα, στον πίνακα *sino* αποθηκεύονται οι ρητές τιμές των προβολών και στους πίνακες *sine* και *cosine* υπολογίζονται οι τιμές των ημιτόνων και συνημιτόνων αντίστοιχα για κάθε μία από τις γωνίες από τις οποίες λήφθηκαν οι προβολές.

5.2. Προτεινόμενες παράλληλες υλοποιήσεις

Σ' αυτό το κεφάλαιο έχουν χρησιμοποιηθεί διαφορετικά σχήματα κώδικα για την εφαρμογή του αλγορίθμου της οπισθοπροβολής. Όλα βασίζονται στην εφαρμογή του απευθείας (Direct) αλγορίθμου, απαιτώντας $O(N^3)$ υπολογισμούς, παρέχοντας όμως υψηλή ποιότητα εικόνας εξόδου. Στόχος ήταν να ερευνηθεί η αξιοπιστία της κάθε διαφορετικής υλοποίησης παράλληλου κώδικα κάνοντας χρήση της OpenCL σε συσκευές CPU, GPU και FPGA.

Τα πειραματικά αποτελέσματα που καταγράφηκαν για τους χρόνους εκτέλεσης όλων των υλοποιήσεων, έγιναν για προβολές μεγέθους 8-256 με 367 γραμμές απόκρισης. Σε κάθε πίνακα παρουσιάζονται οι χρόνοι εκτέλεσης σε ms για τις διάφορες εισόδους, με διαφορετικές ρυθμίσεις στα απαιτούμενα μεγέθη όπως το μέγεθος των work-groups, των tiles και του βάθους αναδρομής όπως αυτά θα εξηγηθούν παρακάτω.

Αρχικός στόχος ήταν η πειραματική μελέτη των υλοποιήσεων αυτών με δεδομένα εισόδου 512 και 1024 (αριθμός προβολών). Στην πορεία όμως και με τη συγκεκριμένη υλοποίηση, φάνηκε ότι οι χρησιμοποιούμενοι buffers ήταν πολλοί σε αριθμό και από το μέγεθος προβολών 512 και πάνω παρατηρήθηκε υπερχείλιση και αδυναμία του προγράμματος να μεταφέρει μέσω των buffers τα δεδομένα εισόδου σωστά. Έτσι η τελική υλοποίηση αφορά δεδομένα εισόδου με αριθμό προβολών έως 256.

Και οι τέσσερις υλοποιήσεις μέσω των ειδικών εντολών της OpenCL αρχικοποιούν κατάλληλα έναν αριθμό από καταχωρητές (buffers) και μεταβλητές ώστε να περαστούν στον πυρήνα οι απαραίτητες για την εκτέλεσή του παράμετροι. Τόσο οι καταχωρητές όσο και οι παράμετροι αναλύονται παρακάτω.

Οι καταχωρητές για τη μεταφορά δεδομένων και την εκτέλεση του πυρήνα μπορούν να είναι είτε ανάγνωσης (CL_MEM_READ_ONLY) που μεταφέρουν απλά δεδομένα, είτε εγγραφής (CL_MEM_WRITE_ONLY) που αποθηκεύουν δεδομένα ώστε να επισταφούν στον host μετά την εκτέλεση του πυρήνα. Δηλώνονται ως __global ώστε όλα να work-items να έχουν πρόσβαση στα δεδομένα τους. Θα μπορούσαν να δηλωθούν και ως _constant κάτι τέτοιο όμως δεν επιτρεπόταν λόγω του μεγάλου μεγέθους των καταχωρητών. Οι καταχωρητές που χρησιμοποιήθηκαν για τις συγκεκριμένες υλοποιήσεις είναι οι εξής :

- $bufferSino$: Καταχωρητής ανάγνωσης μέσω του οποίου περνούνται οι προβολές εισόδου στον πυρήνα.
- $bufferTau$: Καταχωρητής ανάγνωσης μέσω του οποίου περνούνται οι τιμές του μεγέθους T για κάθε γωνία λήψης προβολών.
- $bufferSine, bufferCos$: Καταχωρητές ανάγνωσης μέσω των οποίων περνούνται οι τιμές των ημιτόνων και συνημιτόνων αντίστοιχα για κάθε γωνία λήψης προβολών.
- $bufferSum$: Καταχωρητής εγγραφής στον οποίο αποθηκεύονται οι τιμές των εικονοστοιχείων εξόδου.

Οι μεταβλητές που χρησιμοποιήθηκαν ήταν οι εξής:

- $middle$: Η τιμή του μεγέθους της εικόνας αν διαιρεθεί δια 2 και αφαιρεθεί το 0,5 $(size/2 - 0.5)$
- $halfsize$: Το μισό του μεγέθους της τετραγωνικής εικόνας.
- $numSino$: Ο αριθμός των προβολών (αριθμός γωνιών λήψης των προβολών).
- $size$: Το μέγεθος της τετραγωνικής εικόνας.
- $sinsize$: Το μέγεθος της κάθε προβολής.
- $scale$: Το απαραίτητο μέγεθος ώστε να γίνει το scale της εικόνας.
- $oneOverT$: Ο αντίστροφος του αριθμού Τ.
- $divide$: Το μέγεθος της υποεικόνας που θα υπολογιστεί στην υλοποιήση Direct Tiles όπως θα παρουσιαστεί παρακάτω.

5.2.1. Direct υλοποιήση

Η πρώτη υλοποίηση ονομάζεται Direct και χρησιμοποιεί index space work-items μίας διάστασης. Καθένα απ' αυτά τα work items εκτελεί τον πυρήνα (kernel) προκειμένου να υπολογίσει την τιμή ενός εικονοστοιχείου της εικόνας εξόδου $N \times N$, όπως αυτή υπολογίζεται από την εξίσωση 5.1. Ο κώδικας του πυρήνα παρουσιάζεται στις επόμενες σελίδες.

Έτσι, μέσω των παρακάτω εντολών δηλώνεται ότι τα work-items θα είναι οργανωμένα σε μορφή πλέγματος πράγμα που φαίνεται από τις τιμές του $globalWorkSize$. Το index space είναι μιας διάστασης καθώς ορίζουμε ως NULL την τιμή $localWorkSize$ επιτρέποντας έτσι στο λειτουργικό της OpenCL να επιλέξει από μόνο του τον διαχωρισμό που θα κάνει στα work-items ώστε να εκτελεστούν στο μεγαλύτερο δυνατό αριθμό παραλληλίας.

```
size_t globalWorkSize[2]={size,size};
status=clEnqueueNDRangeKernel(cmdQueue,kernel,2,NULL,globalWorkSize,
NULL,0,NULL,NULL);
```

Σχήμα 5.7: Αρχικοποιήση work-items της direct υλοποίησης

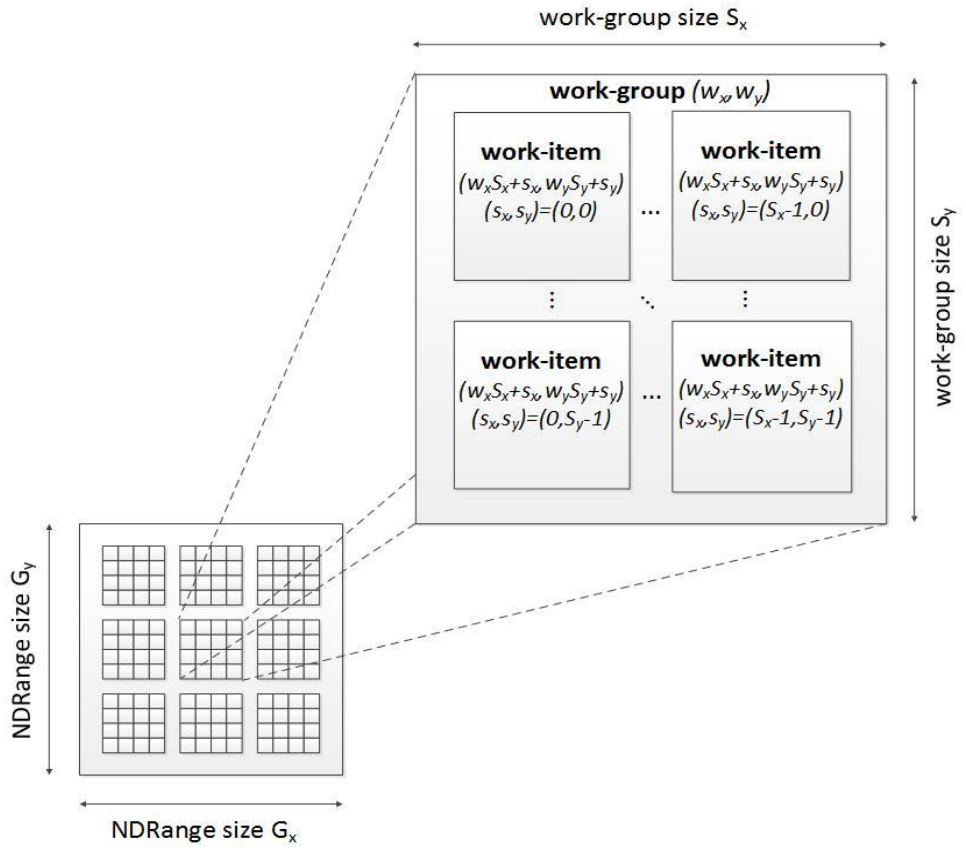
```
float weight(float exactRadialPosition,int k)
{
  #define RADIAL_INTERP(x) LINEAR_INTERP(x)
  #ifndef LINEAR_INTERP
  #define LINEAR_INTERP(x) (1.0-fabs(x))
  #endif

  float argToIntp;
  float ans;

  argToIntp = exactRadialPosition-k;     //the unit of argToIntp is
in samples
  ans = RADIAL_INTERP(argToIntp);

  return ans;
}


__kernel void sum_sino(__global float* bufsino,__global float*
bufsum,float mid,float halfs,int nsino,int siz,__global float*
buftau,__global float* bufsine,__global float* bufcos,int
sinoSize,float scal,float ooT)
  {

  #define ERROR_TOLERANCE 0.0001
  #define RADIAL_SUPPORT LINEAR_SUPPORT
  #define LINEAR_SUPPORT 1

  int m=get_global_id(0);
  int n=get_global_id(1);
  int ip=n*siz+m;
  int p;
  int k;
  int P=nsino;
  float i;
  float my_sum=0;
  float temp;

  float realm=m-halfs+0.5;
  float realn=n-halfs+0.5;

  float scaledRealm = realm*ooT;
  float scaledRealn = realn*ooT;

  float lowerXLimit;
  float upperXLimit;
  float lowerYLimit;
  float upperYLimit;

  float intpSupport;

  float curr;
  float lowerKBound;
  float upperKBound;

  int lowerKBound2;
  int upperKBound2;

  float theta;

  float exactRadialPosition;
```

```
int lower_tolerance;
int upper_tolerance;

intpSupport = RADIAL_SUPPORT;

for(p=0;p<P;p++){

     exactRadialPosition = scaledRealm*bufcos[p] +
scaledRealn*bufsine[p];
     exactRadialPosition -= buftau[p]-mid;

     lowerKBound = exactRadialPosition-intpSupport;
     upperKBound = exactRadialPosition+intpSupport;

     lower_tolerance = (int) (lowerKBound - ERROR_TOLERANCE + 1);
     upper_tolerance = (int) upperKBound;

     lowerKBound2 = (lower_tolerance > 0 ? lower_tolerance : 0);
     upperKBound2 = (upper_tolerance < (sinoSize-1) ? upper_tolerance
: (sinoSize-1));

    if(upperKBound2<lowerKBound2){
      upperKBound2 = lowerKBound2;

     }

    for(k=lowerKBound2;k<=upperKBound2;k++){

      curr = bufsino[p*sinoSize+k];
      temp = weight(exactRadialPosition,k);
      my_sum += curr*temp;

    }
  }

  bufsum[ip]=my_sum*scal;

}
```

Σχήμα 5.8: Κώδικας πυρήνα της direct υλοποίησης

5.2.2. Direct Workgroups υλοποίηση

Η τρίτη υλοποίηση ονομάζεται Direct Workgroup, και είναι ακριβώς ίδια με την Direct υλοποίηση, με τη διαφορά ότι αυτή τη φορά χρησιμοποιείται ένα index space δύο διαστάσεων από work-items και το μέγεθος του κάθε group καθορίζεται από τον χρήστη. Για κάθε group χρησιμοποιείται ο πυρήνας του Direct σχήματος. Παρακάτω παρατίθεται το σχήμα που παρουσιάζει την εκτέλεση του κώδικα.



Σχήμα 5.9: Σχηματική απεικόνιση εκτέλεσης της Direct Workgroup υλοποίησης

Εδώ παρατηρείται πώς τα work-items οργανώνονται και πάλι σε μορφή πλέγματος όπως φαίνεται από τη μεταβλητή ***globalWokrSize***, αυτή τη φορά όμως ο χρήστης δηλώνει μέσω του ***localWorkSize*** μεγέθους τη διάσταση που επιθυμεί να έχουν τα work-groups.

```
size_t globalWorkSize[2]={size,size};
size_t localWorkSize[2]={WorkSize,WorkSize};

status=clEnqueueNDRangeKernel(cmdQueue,kernel,2,NULL,globalWorkSize,
localWorkSize,0,NULL,NULL);
```

Σχήμα 5.10: Αρχικοποιήση work-items της Direct Workgroup υλοποίησης

5.2.3. Direct tiles υλοποίηση

Η δεύτερη υλοποίηση ονομάζεται Direct Tile και καλεί μια συνάρτηση πυρήνα προκειμένου να υπολογίσει τον αλγόριθμο οπισθοπροβολής για ένα tile $K \times K$ εικονοστοιχείων. Ο εν λόγω κώδικας δίνεται παρακάτω. Ο κώδικας είναι παρόμοιος με τον κώδικα του Direct αλγορίθμου οπισθοπροβολής, με τη διαφορά ότι το σημείο εισόδου είναι η συνάρτηση *calculate_tile* η οποία αποτελεί μια εμφωλευμένη δομή βρόχου. Κάθε εκτέλεση του βρόχου υπολογίζει ένα εικονοστοιχείο της εικόνας εξόδου. Το όριο K του βρόχου είναι το μέγεθος του κομματιού (tile) και καθορίζεται από το χρήστη.

Όπως και στην Direct υλοποίηση ο διαχωρισμός των work-items πραγματοποιείται από το λειτουργικό για μέγιστη παραλληλία. Οι διαφορές από την Direct Workgroup υλοποίηση είναι οι εξής:

1. Ο κώδικας εκτελείται για $size/divide$ work-items
2. Ο υπολογισμός των work-items του κάθε tile πραγματοποιείται σειριακά και η παραλληλία λαμβάνει χώρα μόνο ανάμεσα στα διαφορετικά tiles.

```
int numItems=size/divide;
size_t globalWorkSize[2]={numItems,numItems};
status=clEnqueueNDRangeKernel(cmdQueue,kernel,2,NULL,globalWorkSize,
NULL,0,NULL,NULL);
```

Σχήμα 5.11: Αρχικοποιήση work-items της Direct Tiles υλοποίησης

```
float weight(float exactRadialPosition,int k)
{
  #define RADIAL_INTERP(x) LINEAR_INTERP(x)
  #ifndef LINEAR_INTERP
  #define LINEAR_INTERP(x) (1.0-fabs(x))
  #endif

  float argToIntp;
  float ans;

  argToIntp = exactRadialPosition-k;     //the unit of argToIntp is
in samples
  ans = RADIAL_INTERP(argToIntp);

  return ans;
}


float sum_sino(__global float* bufsino,float mid,float halfs,int
nsino,int siz,__global float* buftau,__global float* bufsine,__global
float* bufcos,int sinoSize,float scal,float ooT,int m,int n)
  {

  #define ERROR_TOLERANCE 0.0001
  #define RADIAL_SUPPORT LINEAR_SUPPORT
  #define LINEAR_SUPPORT 1


  int p;
  int k;
  int P=nsino;
  float i;
  float my_sum=0;
  float temp;

  float realm=m-halfs+0.5;
  float realn=n-halfs+0.5;

  float scaledRealm = realm*ooT;
  float scaledRealn = realn*ooT;

  float lowerXLimit;
  float upperXLimit;
  float lowerYLimit;
  float upperYLimit;

  float intpSupport;

  float curr;
  float lowerKBound;
  float upperKBound;

  int lowerKBound2;
  int upperKBound2;

  float theta;

  float exactRadialPosition;
  int lower_tolerance;
  int upper_tolerance;
```

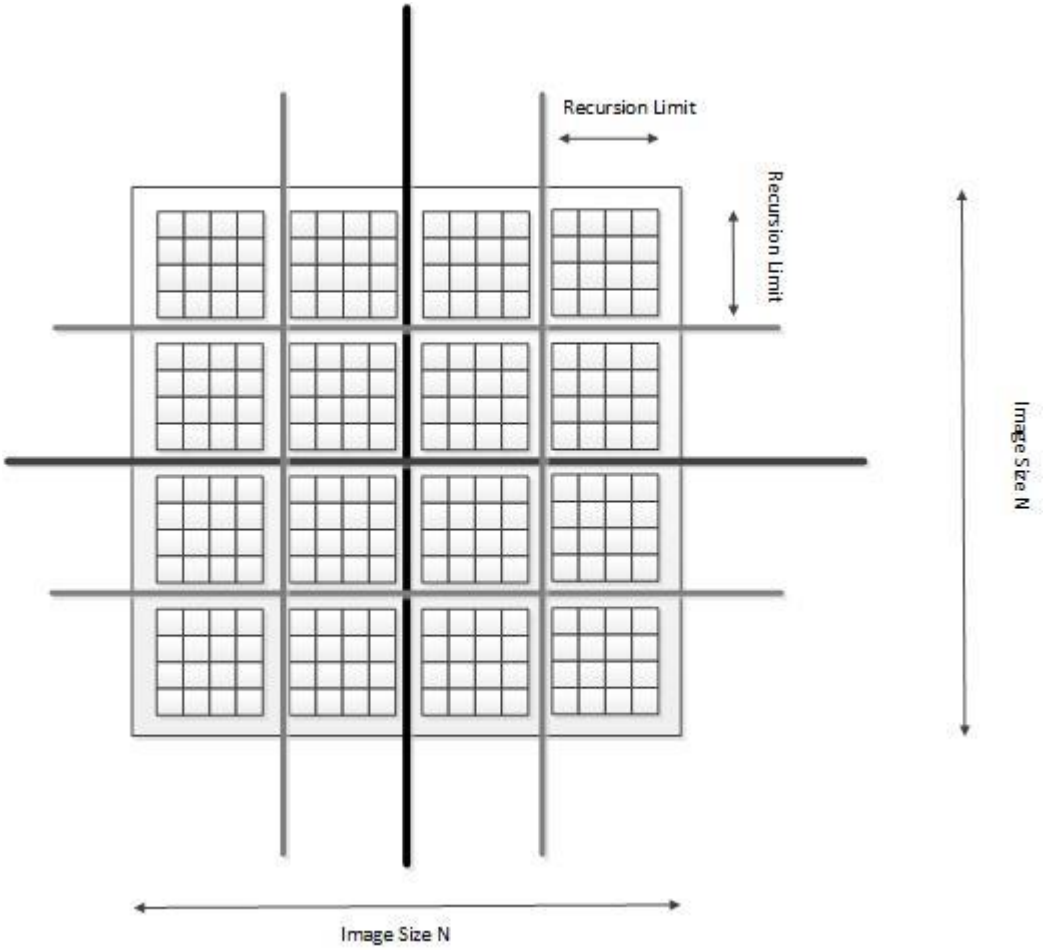
```
  intpSupport = RADIAL_SUPPORT;

  for(p=0;p<P;p++){

     exactRadialPosition = scaledRealm*bufcos[p] +
scaledRealn*bufsine[p];
     exactRadialPosition -= buftau[p]-mid;

     lowerKBound = exactRadialPosition-intpSupport;
     upperKBound = exactRadialPosition+intpSupport;

     lower_tolerance = (int) (lowerKBound - ERROR_TOLERANCE + 1);
     upper_tolerance = (int) upperKBound;

     lowerKBound2 = (lower_tolerance > 0 ? lower_tolerance : 0);
     upperKBound2 = (upper_tolerance < (sinoSize-1) ? upper_tolerance
: (sinoSize-1));

    if(upperKBound2<lowerKBound2){
      upperKBound2 = lowerKBound2;

     }

    for(k=lowerKBound2;k<=upperKBound2;k++){

      curr = bufsino[p*sinoSize+k];
      temp = weight(exactRadialPosition,k);
      my_sum += curr*temp;

    }
  }

  return my_sum*scal;

}

__kernel void calculate_tile(__global float* bufsino,__global float*
bufsum,float mid,float halfs,int nsino,int siz,__global float*
buftau,__global float* bufsine,__global float* bufcos,int
sinoSize,float scal,float ooT,int divideNum)

{
  int idx=get_global_id(0);
  int idy=get_global_id(1);
  int k,l,m,n,position;


  m=idx*divideNum;

  for(k=0;k<divideNum;k++){
   n=idy*divideNum;
   for(l=0;l<divideNum;l++){
      position=m*siz+n;

bufsum[position]=sum_sino(bufsino,mid,halfs,nsino,siz,buftau,bufsine,
bufcos,sinoSize,scal,ooT,n,m);
      n++;
      }
   m++;
  }}
```

Σχήμα 5.12: Κώδικας πυρήνα της Direct Tiles υλοποίησης

5.2.4. Direct Recursive υλοποίηση

Η τέταρτη υλοποίηση ονομάζεται Direct Recursive και χρησιμοποιεί μια αναδρομική συνάρτηση από την πλευρά του host επεξεργαστή για να διασπάσει τον υπολογισμό της οπισθοπροβολής σε τεταρτημόρια μέχρι ένα καθορισμένο βάθος K , όπως φαίνεται και στο παρακάτω σχήμα. Από το βάθος αυτό και μετά εκτελείται η Direct υλοποίηση για κάθε ένα από τα τεταρτημόρια αυτά.

Η δήλωση των μεγεθών των work-items είναι ίδια με αυτή της Direct υλοποίησης για το δεδομένο βάθος K της αναδρομής στο οποίο έχει φτάσει ο αλγόριθμος.



Σχήμα 5.13: Σχηματική απεικόνιση εκτέλεσης της Direct Recursive υλοποίησης

5.3. Αποτελέσματα στις CPU

Στην ενότητα αυτή παρουσιάζονται σε μορφή πινάκων οι χρόνοι εκτέλεσης όλων των υλοποιήσεων στις CPU1 και CPU2.

Χρόνος Εκτέλεσης (ms)						
Αριθμός προβολών εισόδου						
	8	16	32	64	128	256
Direct μέθοδος	0,12	0,15	0,18	0,45	2,31	18,14
Serial μέθοδος	0,30	0,55	4,04	32,69	252,39	2022,83

Πίνακας 5.1 : Χρόνοι εκτέλεσης της Direct και της σειριακής υλοποίησης στη CPU1

Χρόνος Εκτέλεσης (ms)						
Αριθμός προβολών εισόδου						
	8	16	32	64	128	256
Direct μέθοδος	0,16	0,19	0,40	2,68	18,20	128,35
Serial μέθοδος	0,19	0,63	4,88	41,67	355,44	2842,79

Πίνακας 5.2 : Χρόνοι εκτέλεσης της Direct και της σειριακής υλοποίησης στη CPU2

Tiles	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,13	0,15	0,18	0,47	2,34	18,35
4	0,14	0,15	0,18	0,47	2,48	19,80
8		0,16	0,19	0,52	2,67	20,27
16			0,20	0,79	2,89	21,86
32				0,86	5,50	23,65
64					5,76	50,03
128						53,79

Πίνακας 5.3 : Χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στη CPU1

Tiles	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,18	0,21	0,43	2,92	20,75	131,93
4	0,17	0,21	0,48	3,54	21,16	130,60
8		0,22	0,52	3,23	21,77	131,84
16			0,54	3,64	22,66	134,24
32				3,19	22,05	136,44
64					21,69	145,89
128						147,29

Πίνακας 5.4 : Χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στη CPU2

Workgroups	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,13	0,15	0,19	0,47	2,38	18,14
4	0,14	0,15	0,19	0,47	2,40	19,32
8		0,16	0,19	0,46	2,30	19,41
16			0,18	0,48	2,40	18,76
32				0,46	2,41	18,75
64					2,40	18,34
128						19,71

Πίνακας 5.5 : Χρόνοι εκτέλεσης της Direct Workgroups υλοποίησης στη CPU1

Workgroups	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,18	0,19	0,40	2,38	19,48	128,13
4	0,16	0,23	0,45	2,30	17,13	129,98
8		0,19	0,39	2,64	18,39	130,72
16			0,39	2,40	18,29	132,67
32				3,25	18,27	133,54
64					19,58	133,22
128						132,05

Πίνακας 5.6 : Χρόνοι εκτέλεσης της Direct Workgroups υλοποίησης στη CPU2

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,96	1,65	10,38	72,38	561,32	4378,59
4	0,47	0,97	5,65	43,17	330,71	2603,49
8		0,61	4,41	33,78	273,24	2154,43
16			3,98	33,40	264,56	2042,91
32				32,38	255,29	2032,07
64					255,24	2016,76
128						2027,73

Πίνακας 5.7 : Χρόνοι εκτέλεσης της σειριακής Recursive υλοποίησης στη CPU1

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,50	3,87	16,62	127,42	888,84	6852,02
4	0,24	1,40	9,02	67,85	506,31	3924,57
8		0,85	6,41	52,58	413,51	3246,56
16			5,39	48,33	387,42	3006,65
32				43,69	370,92	2942,46
64					360,83	2925,21
128						2994,87

Πίνακας 5.8 : Χρόνοι εκτέλεσης της σειριακής Recursive υλοποίησης στη CPU2

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,64	2,07	8,29	30,84	125,45	542,72
4	0,30	0,69	2,30	9,10	40,37	186,52
8		0,28	0,82	2,98	12,68	58,65
16			0,33	1,10	5,07	29,74
32				0,61	3,10	20,33
64					2,57	18,36
128						18,27

Πίνακας 5.9: Χρόνοι εκτέλεσης της Direct Recursive υλοποίησης στη CPU1

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,80	3,47	13,53	56,62	236,67	1064,22
4	0,28	1,11	4,15	18,72	81,67	405,66
8		0,33	1,67	7,51	35,62	204,15
16			0,66	4,57	22,04	147,01
32				3,20	23,82	149,01
64					19,65	136,85
128						134,38

Πίνακας 5.10 : Χρόνοι εκτέλεσης της Direct Recursive υλοποίησης στη CPU2

5.4. Αποτελέσματα στις GPU

Στην ενότητα αυτή παρουσιάζονται σε μορφή πινάκων οι χρόνοι εκτέλεσης όλων των υλοποιήσεων στις GPU1 και GPU2.

Χρόνος Εκτέλεσης (ms)						
Αριθμός προβολών εισόδου						
	8	16	32	64	128	256
Direct GPU1	5,08	5,51	6,04	6,42	6,26	7,77
Direct GPU2	2,21	2,55	2,72	3,40	9,12	40,63

Πίνακας 5.11 : Χρόνοι εκτέλεσης της Direct υλοποίησης στη GPU1 και GPU2

	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
Tiles	8	16	32	64	128	256
2	5,51	6,15	6,32	6,42	6,75	8,40
4	5,83	6,18	6,45	7,14	7,86	11,24
8		6,70	7,77	8,70	13,45	24,41
16			13,89	21,60	42,87	84,53
32				69,04	147,10	316,76
64					559,04	1138,31
128						4507,67

Πίνακας 5.12 : Χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στη GPU1

	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
Tiles	8	16	32	64	128	256
2	2,65	2,69	2,83	3,66	8,42	41,42
4	3,11	2,97	3,52	4,72	12,13	54,23
8		4,37	6,47	10,90	20,44	90,07
16			18,64	35,28	69,75	167,34
32				131,25	264,19	541,31
64					1036,96	2091,05
128						8287,97

Πίνακας 5.13 : Χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στη GPU2

Workgroups	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	6,04	6,34	6,32	6,45	8,16	18,40
4	6,00	6,28	6,08	6,37	6,80	12,15
8		6,29	6,30	6,05	6,28	7,69
16			6,25	6,48	6,35	7,62
32				6,96	6,69	7,30
64					6,97	7,54
128						7,16

Πίνακας 5.14 : Χρόνοι εκτέλεσης της Direct Workgroups υλοποίησης στη GPU1

Workgroups	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	2,11	2,59	3,20	10,52	66,19	510,64
4	2,17	2,57	3,07	4,53	18,44	129,46
8		2,51	2,60	3,68	9,61	43,12
16			2,63	3,82	9,41	41,57
32				3,73	9,79	41,90
64					9,83	40,87
128						40,73

Πίνακας 5.15 : Χρόνοι εκτέλεσης της Direct Workgroups μεθόδου στη GPU2

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	9,15	19,39	66,45	375,23	2737,55	11274,13
4	5,92	9,09	20,18	114,43	720,27	2930,48
8		6,42	9,21	23,80	174,32	897,25
16			6,75	10,15	31,71	210,01
32				6,59	12,20	55,61
64					6,88	14,32
128						8,97

Πίνακας 5.16 : Χρόνοι εκτέλεσης της Direct Recursive υλοποίησης στη GPU1

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	22,89	92,24	374,41	1606,71	6786,14	31183,73
4	6,03	24,31	95,76	396,65	1692,02	7805,23
8		6,10	24,74	101,55	427,99	1757,03
16			6,16	20,89	98,18	437,77
32				6,15	27,87	145,82
64					13,08	67,85
128						44,54

Πίνακας 5.17 : Χρόνοι εκτέλεσης της Direct Recursive υλοποίησης στη GPU2

5.5. Αποτελέσματα στο FPGA

Στην ενότητα αυτή παρουσιάζονται οι μετρήσεις που πραγματοποιήθηκαν στο FPGA για όλες τις παραπάνω υλοποιήσεις, τόσο για την σειριακή εκδοχή τους όσο και για την παράλληλη.

Χρόνος Εκτέλεσης (ms)						
Αριθμός προβολών εισόδου						
	8	16	32	64	128	256
Direct	0,35	0,37	0,81	4,01	29,66	234,96
Serial	0,20	1,51	11,96	94,81	762,72	6102,34

Πίνακας 5.18 : Χρόνοι εκτέλεσης της Direct και της σειριακής υλοποίησης στο FPGA

Tiles	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,38	0,47	0,88	4,43	31,36	248,36
4	0,52	0,68	1,01	4,27	32,56	248,43
8		1,60	2,87	7,62	31,36	264,63
16			10,16	27,76	58,36	248,30
32				101,56	229,55	508,86
64					862,35	1937,79
128						7022,57

Πίνακας 5.19 : Χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στο FPGA

Workgroups	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	0,35	0,37	0,85	4,22	29,69	234,96
4	0,35	0,37	0,84	4,24	29,86	235,06
8		0,38	0,85	4,26	29,76	235,95
16			0,85	4,42	29,84	234,98
32				4,23	29,78	234,98
64					29,84	235,21
128						235,26

Πίνακας 5.20 : Χρόνοι εκτέλεσης της Direct Workgroup υλοποίησης στο FPGA

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	1,42	7,82	43,64	270,74	1893,52	14045,36
4	0,59	4,04	23,92	157,41	1113,68	8478,19
8		2,33	16,59	118,57	882,62	6839,60
16			13,48	104,35	803,80	6308,04
32				98,26	776,80	6151,17
64					765,80	6118,20
128						6109,06

Πίνακας 5.21 : Χρόνοι εκτέλεσης της σειριακής Recursive υλοποίησης στο FPGA

Recursion Limit	Χρόνοι Εκτέλεσης (ms)					
	Αριθμός προβολών εισόδου					
	8	16	32	64	128	256
2	3,31	13,48	60,39	86,19	158,51	415,80
4	0,93	3,50	15,55	78,01	119,95	393,15
8		1,01	4,30	20,56	113,82	373,67
16			1,40	7,13	45,66	320,25
32				4,69	34,99	254,45
64					30,62	239,13
128						236,12

Πίνακας 5.22 : Χρόνοι εκτέλεσης της Direct Recursive υλοποίησης στο FPGA

6.Συμπεράσματα

Στο κεφάλαιο αυτό, παρουσιάζονται αναλυτικά τα συμπεράσματα που προκύπτουν από τα αποτελέσματα των μετρήσεων για όλες τις υλοποιήσεις που πραγματοποιήθηκαν όπως αυτές παρουσιάστηκαν στο κεφάλαιο 5. Έτσι, γίνονται συγκρίσεις ανάμεσα στις διαφορετικές υλοποιήσεις του αλγορίθμου της οπισθοπροβολής με διάφορα κριτήρια και υποδεικνύονται οι καλύτεροι χρόνοι, οι καλύτερες επιταχύνσεις και η ελάχιστη κατανάλωση ενέργειας που προκύπτουν από τη συγκεκριμένη διπλωματική εργασία.

6.1. Συκρίσεις με βάση τη συσκευή

6.1.1. Επιταχύνσεις υλοποιήσεων CPU

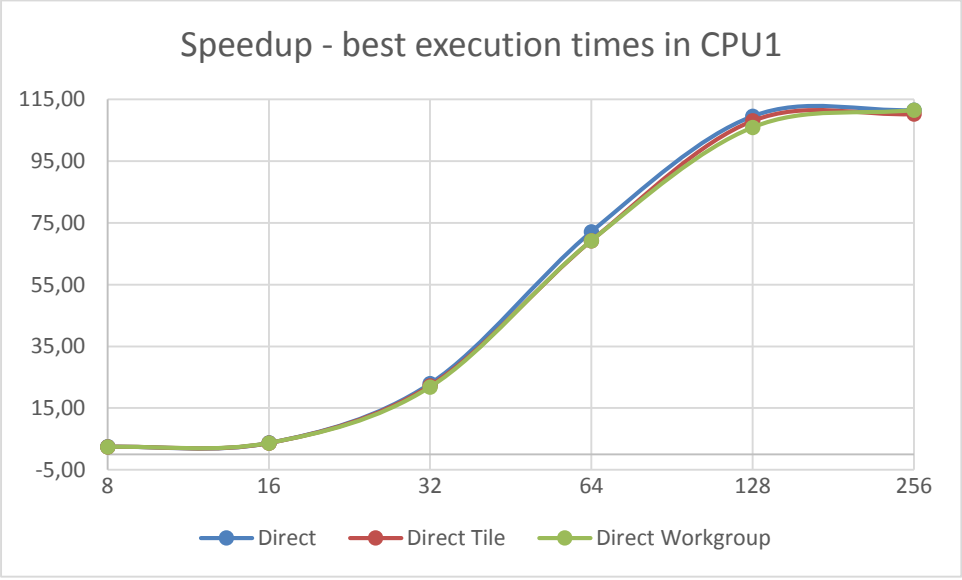
Στην ενότητα αυτή παρουσιάζεται η επιτάχυνση (speedup) κάθε υλοποιήσης συγκρίνοντας την παράλληλη μορφή της σε σχέση με την σειριακή.

Η επιτάχυνση δίνεται από τη σχέση :

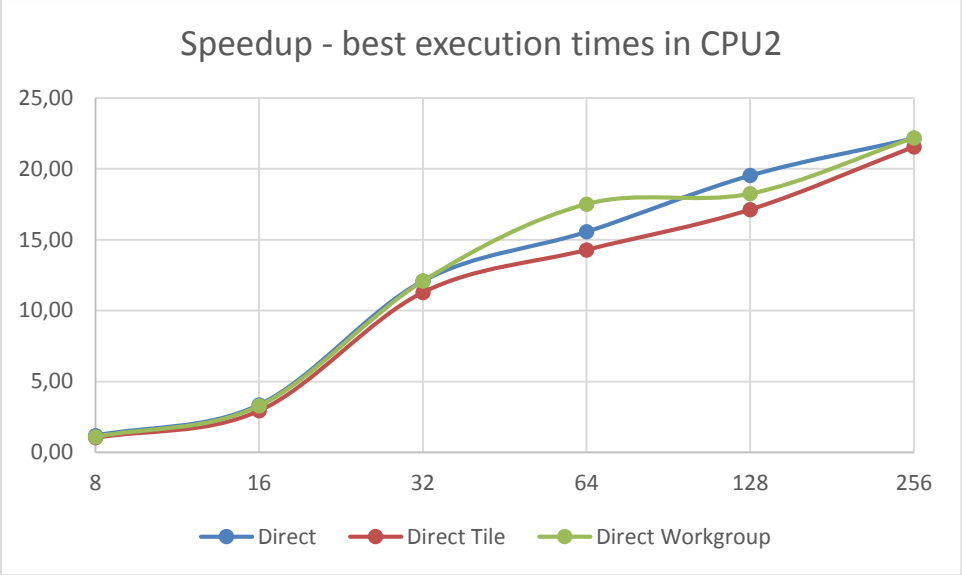
$$S = \frac{T_{old}}{T_{new}} \tag{6.1}$$

Όπου S η επιτάχυνση (speedup), T_{old} ο παλιός χρόνος εκτέλεσης χωρίς τη βελτίωση και T_{new} ο νέος χρόνος εκτέλεσης μετά τη βελτίωση του κώδικα.

Στα πρώτα δύο σχήματα παρατηρούνται οι επιταχύνσεις με βάση τους καλύτερους χρόνους εκτέλεσης για τις υλοποιήσεις Direct, Direct Tiles και Direct Workgroups στις CPU1 και CPU2.

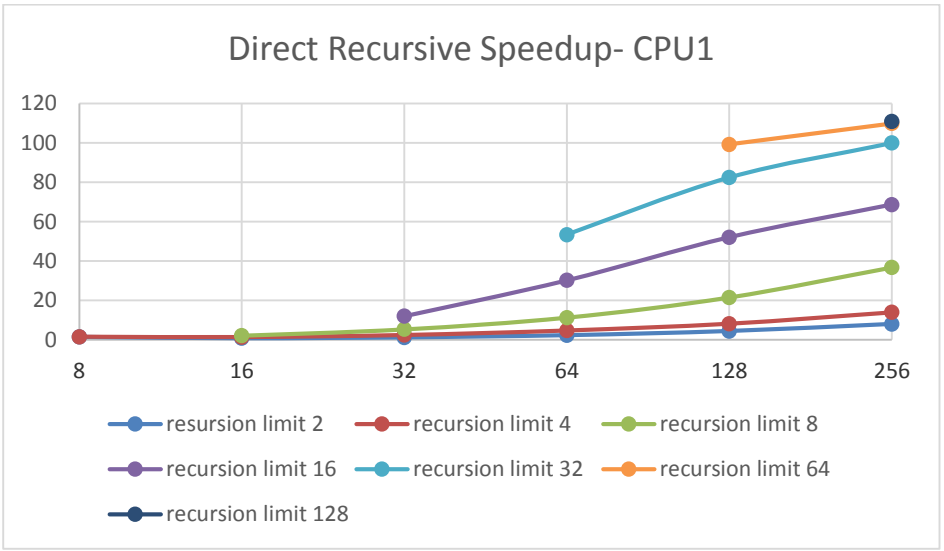


Σχήμα 6.1: Επιτάχυνση καλύτερων χρόνων των υλοποιήσεων Direct, Direct Tile και Direct Workgroup στη CPU1

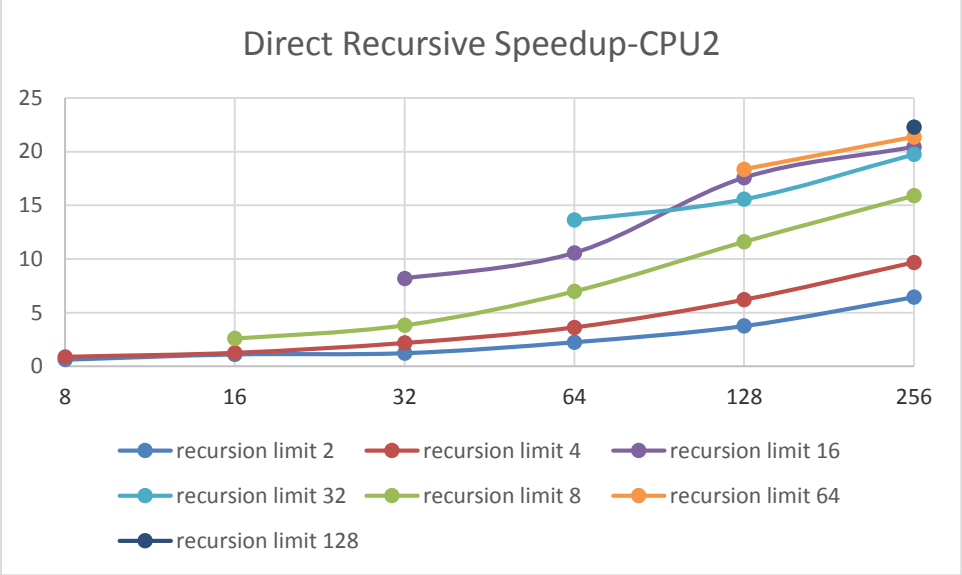


Σχήμα 6.2: Επιτάχυνση καλύτερων χρόνων των υλοποιήσεων Direct, Direct Tile και Direct Workgroup στη CPU2

Στη συνέχεια παρουσιάζονται οι επιταχύνσεις στις δύο CPU της υλοποίησης Direct Recursive οι οποίες έχουν προκύψει συγκρίνοντας τους χρόνους εκτέλεσης της Serial Recursive υλοποίησης με την Direct Recursive. Χρησιμοποιούνται ξεχωριστά διαγράμματα για τις Direct υλοποιήσεις και τις Recursive μιας και οι σειριακοί χρόνοι στις δύο περιπτώσεις είναι διαφορετικοί.



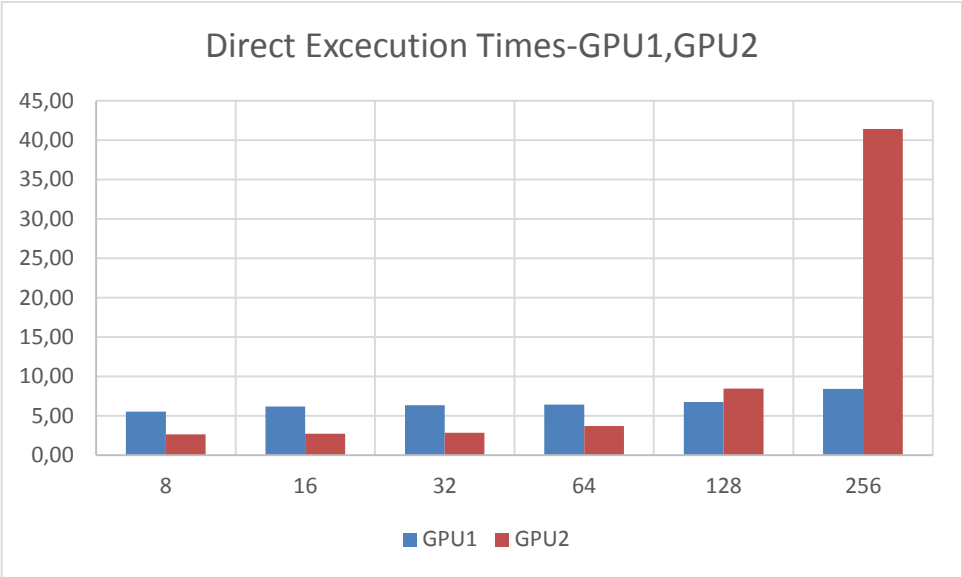
Σχήμα 6.3: Επιτάχυνση Direct Recursive υλοποίησης στη CPU1 συγκρινόμενη με την Serial Recursive υλοποίηση



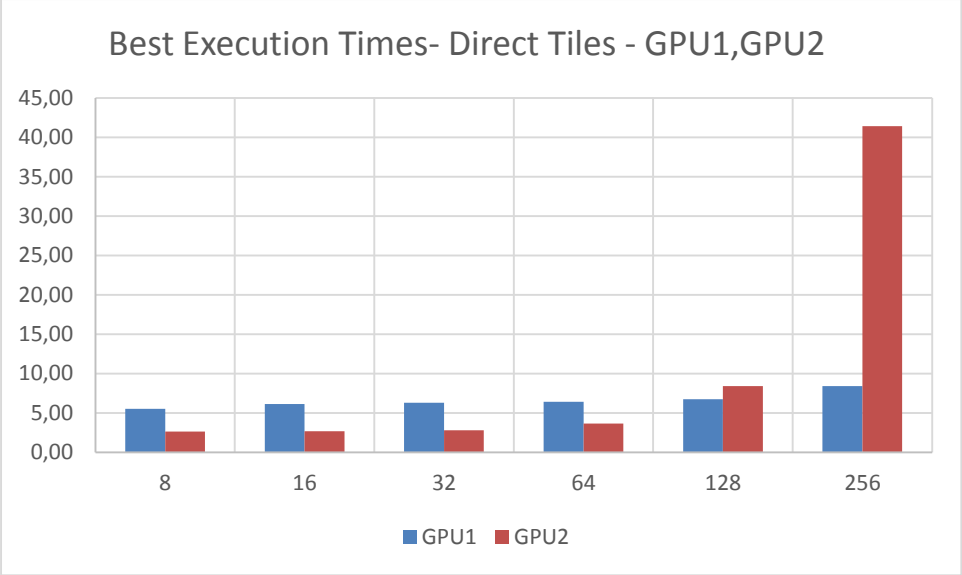
Σχήμα 6.4 : Επιτάχυνση Direct Recursive υλοποίησης στη CPU2 συγκρινόμενη με την Serial Recursive υλοποίηση

6.1.2. Διαγράμματα χρόνων εκτέλεσης στη GPU

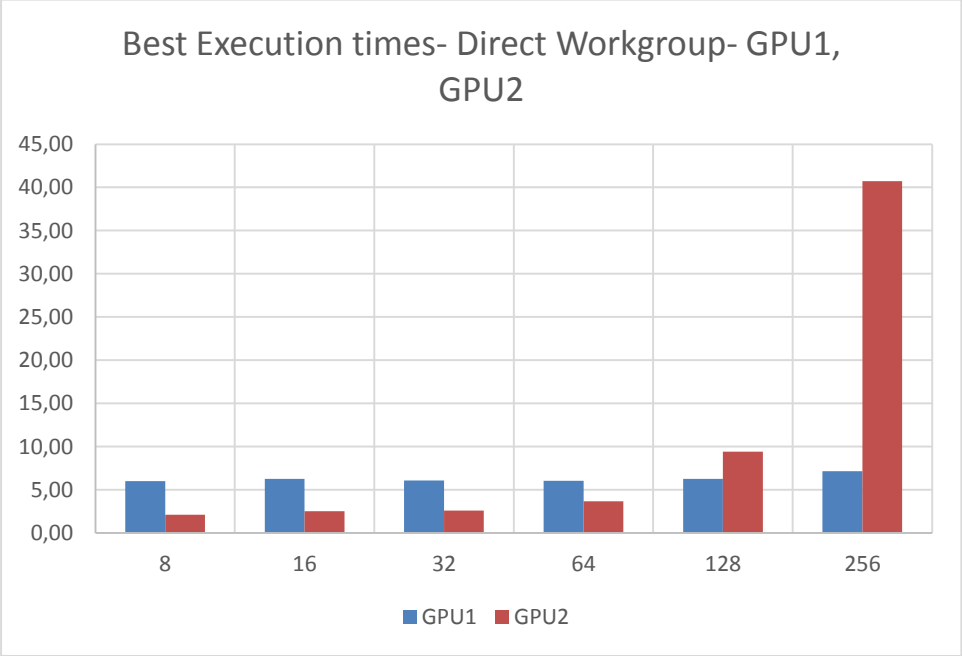
Στην ενότητα αυτή παρουσιάζονται συγκριτικά σχεδιαγράμματα για τους καλύτερους χρόνους εκτέλεσης της κάθε υλοποίησης στις GPU1 στις GPU2. Η επιτάχυνση δεν είναι δυνατόν να υπολογιστεί καθώς οι μετρήσεις που λήφθηκαν για την GPU ήταν δυνατόν να πραγματοποιηθούν μόνο για τις παράλληλες υλοποιήσεις. Αντ'αυτού, θεωρήθηκε σημαντική η σύγκριση των χρόνων μεταξύ των δύο GPUs.



Σχήμα 6.5 : Χρόνοι εκτέλεσης της Direct υλοποίησης στις GPU1 και GPU2



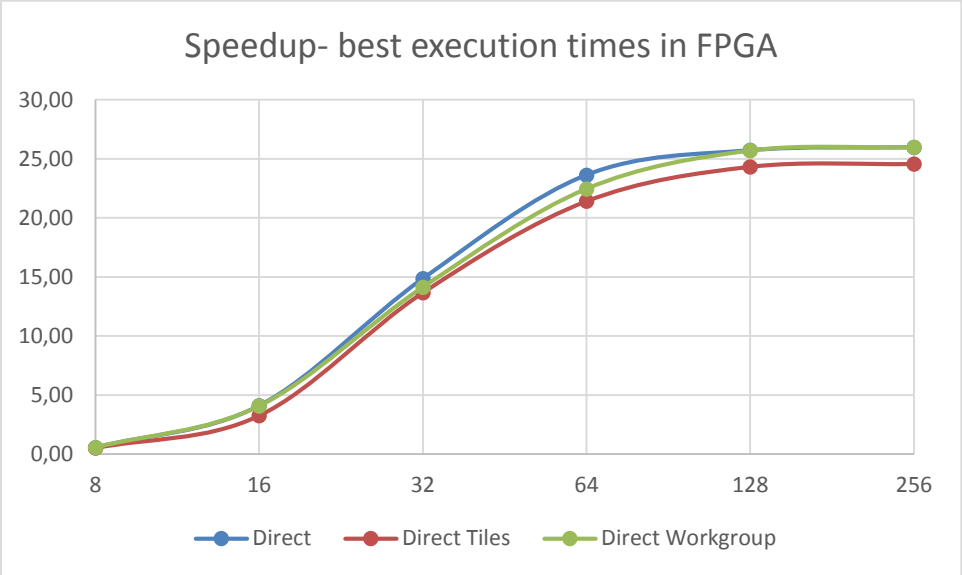
Σχήμα 6.6 : Καλύτεροι χρόνοι εκτέλεσης της Direct Tiles υλοποίησης στις GPU1 και GPU2 (για μέγεθος tile=2)



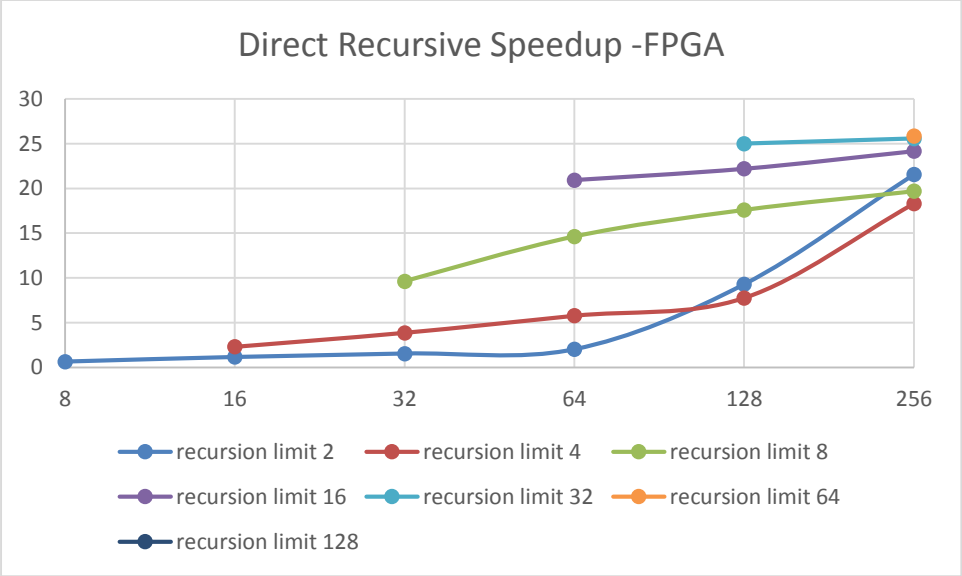
Σχήμα 6.7 : Καλύτεροι χρόνοι εκτέλεσης της Direct Workgroup υλοποίησης στις GPU1 και GPU2

6.1.3. Επιταχύνσεις υλοποιήσεων στο FPGA

Στο πρώτο σχήμα παρατηρούνται οι επιταχύνσεις με βάση τους καλύτερους χρόνους εκτέλεσης για τις υλοποιήσεις Direct, Direct Tiles και Direct Workgroups στο FPGA. Στη συνέχεια παρουσιάζονται οι επιταχύνσεις στο FPGA της υλοποίησης Direct Recursive η οποία έχει προκύψει συγκρίνοντας τους χρόνους εκτέλεσης της Serial Recursive υλοποίησης με την Direct Recursive. Χρησιμοποιούνται όπως και πριν, ξεχωριστά διαγράμματα για τις Direct υλοποιήσεις και τις Recursive μιας και οι σειριακοί χρόνοι στις δύο περιπτώσεις είναι διαφορετικοί.



Σχήμα 6.8 : Επιτάχυνση καλύτερων χρόνων των υλοποιήσεων Direct, Direct Tile και Direct Workgroup στο FPGA



Σχήμα 6.9 : Επιτάχυνση της Direct Recursive υλοποίησης συγκρινόμενη με την σειριακή Recursive υλοποίηση για όλα τα όρια της αναδρομής στο FPGA

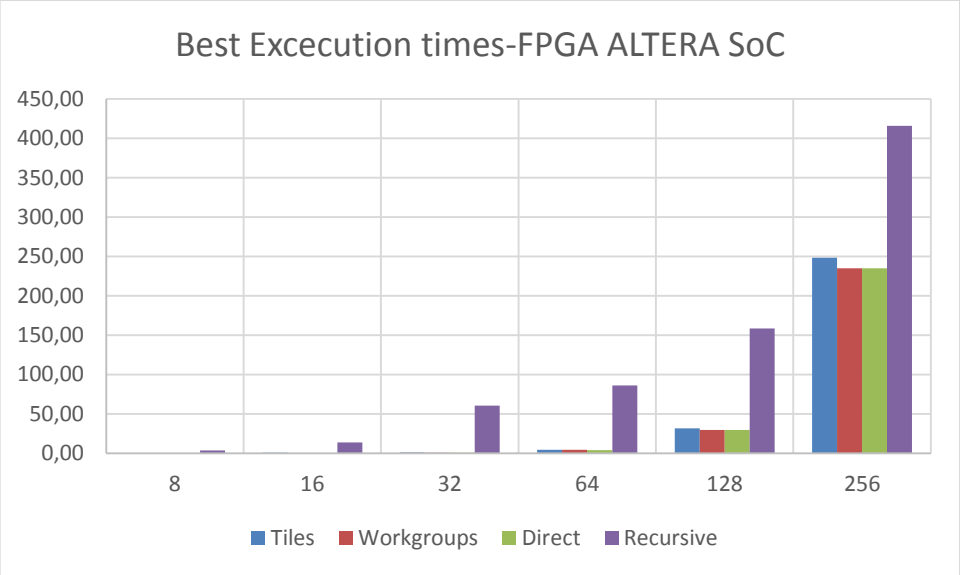
6.2. Συγκρίσεις με βάση τα coding schemes

6.2.1. Ταχύτερη μέθοδος

Παρατηρώντας τις μετρήσεις του κεφαλαίου 5, η Direct υλοποίηση είναι εκείνη που δίνει τα ταχύτερα αποτελέσματα για όλα τα διαφορετικά σινογράμματα. Οι χρόνοι της Direct υλοποίησης παρουσιάζονται συγκεντρωτικά στον παρακάτω πίνακα για κάθε συσκευή που χρησιμοποιήθηκε αλλά και στο συγκεντρωτικό διάγραμμα που δείχνει ότι η direct είναι καλύτερη για κάθε μέγεθος σινογραμμάτων.

Direct time (ms)	Sinograms					
	8	16	32	64	128	256
FPGA	0,35	0,37	0,81	4,01	29,66	234,96
CPU1	0,12	0,15	0,18	0,45	2,31	18,14
GPU1	5,08	5,51	6,04	6,42	6,26	7,77
CPU2	0,16	0,19	0,40	2,68	18,20	128,35
GPU2	2,21	2,55	2,72	3,40	9,12	40,63

Πίνακας 6.1 : Χρόνοι εκτέλεσης της Direct μεθόδου σε όλες τις υπολογιστικές συσκευές



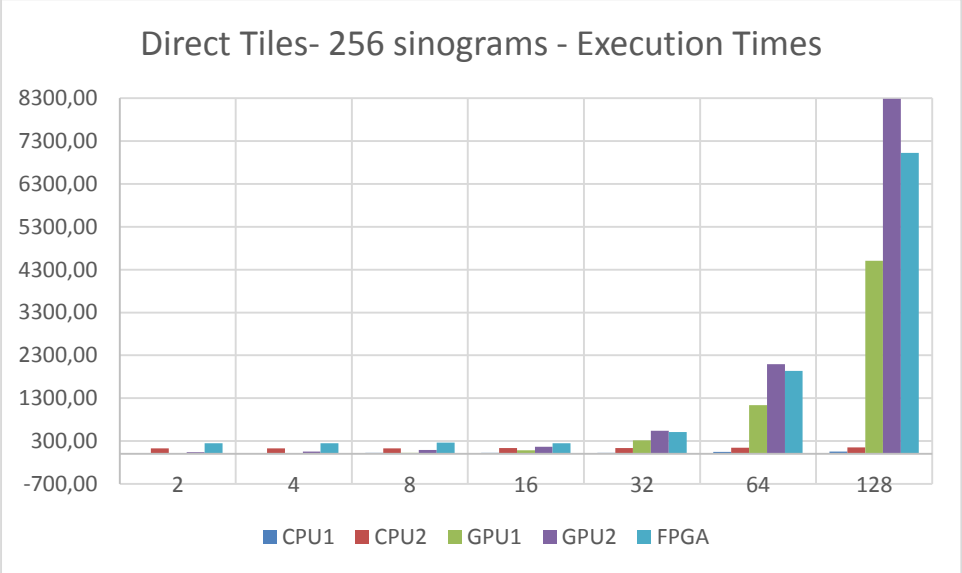
Σχήμα 6.10 : Καλύτεροι χρόνοι εκτέλεσης όλων των υλοποιήσεων στο FPGA

Το γεγονός ότι η Direct είναι η ταχύτερη υλοποίηση για κάθε συσκευή δείχνει ότι το μοντέλο μνήμης της OpenCL όπως εφαρμόζεται στις GPUs αλλά και στο FPGA συμπεριφέρεται αντίστοιχα καλά τόσο σε μία όσο και σε δύο διαστάσεων index-spaces. Όπως ήταν αναμενόμενο, οι σειριακές υλοποιήσεις (απλή σειριακή και recursive σειριακή) καταναλώνουν περισσότερο χρόνο μιας και κάθε γενικής χρήσης CPU συμπεριφέρεται χειρότερα από επιταχυντές υλικού για συγκεκριμένο σκοπό (specialized hardware accelerators).

6.2.2. Σύγκριση της direct μεθόδου με την direct tiles

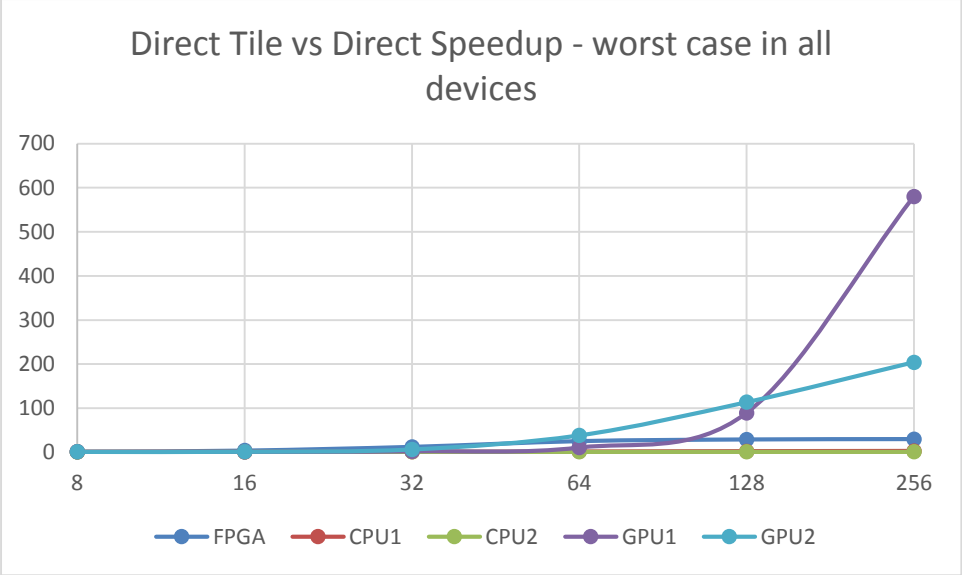
Προκειμένου να γίνει μια ποσοτική σύγκριση, η επιτάχυνση (speedup) της Direct μεθόδου συγκρίνεται με την επιτάχυνση της Direct tile μεθόδου. Τα αποτελέσματα που προκύπτουν είναι κατά μέσο όρο 100X ενώ στην GPU1 παρατηρείται επιτάχυνση έως και 580X στην χειρότερη περίπτωση (στοιχεία της τελευταίας γραμμής, μεγάλα μεγέθη tiles πράγμα που σημαίνει περισσότεροι υπολογισμοί σε κάθε βρόχο).

Στην καλύτερη περίπτωση (πρώτες γραμμές του πίνακα, μικρότερες διαστάσεις που σημαίνει λιγότερες εκτελέσεις σε κάθε βρόχο) οι χρόνοι εκτέλεσης των δύο υλοποιήσεων είναι σχεδόν ταυτόσημες. Ο παρακάτω πίνακας παρουσιάζει τους χρόνους εκτέλεσης της Direct Tiles υλοποίησης για μέγεθος προβολών εισόδου 256 για όλες τις υπολογιστικές συσκευές.



Σχήμα 6.11 : Χρόνοι εκτέλεσης της direct tiles μεθόδου για μέγεθος προβολών = 256 σε όλες τις υπολογιστικές συσκευές.

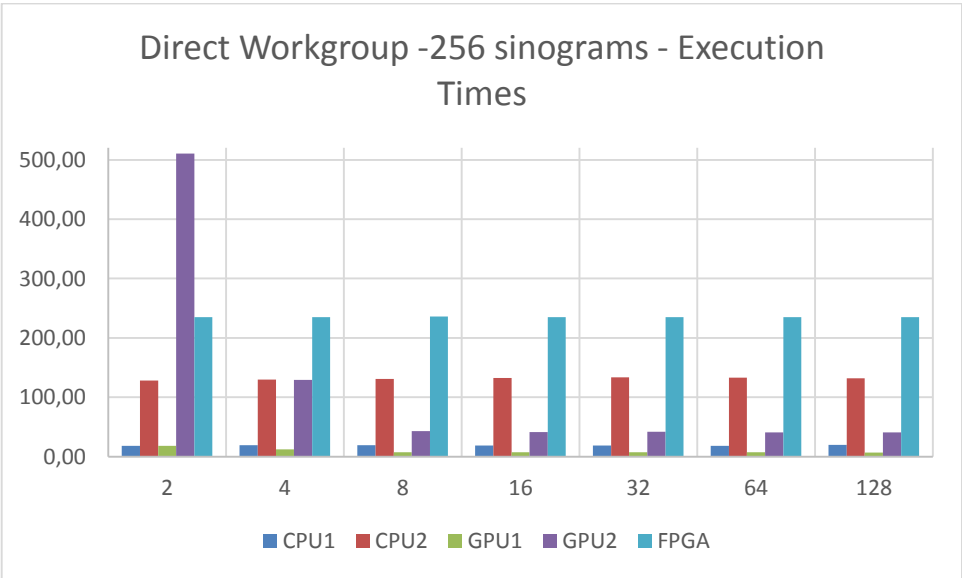
Στο διάγραμμα που ακολουθεί φαίνεται η επιτάχυνση για κάθε υπολογιστική συσκευή για τις χειρότερες περιπτώσεις για όλα τα σινογράμματα εισόδου.



Σχήμα 6.12 : Επιτάχυνση χειρότερης περίπτωσης της Direct υλοποίησης σε σχέση με την Direct Tile υλοποίηση σε όλες τις χρησιμοποιούμενες συσκευές

6.2.3. Σύγκριση της direct μεθόδου με την direct workgroup

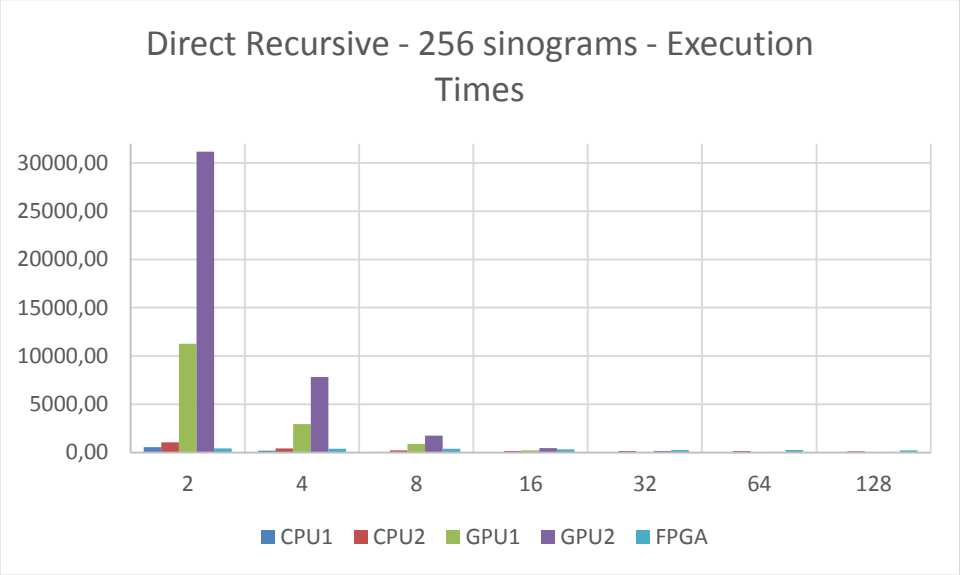
Η Direct Workgroup υλοποίηση παρουσιάζει σχεδόν ίδιους χρόνους με την Direct με εξαίρεση τις δύο πρώτες γραμμές (workgroup size 2,4) όπου η Direct εμφανίζεται να είναι 1,5Χ ταχύτερη και στην περίπτωση της GPU2 κατά 12,57Χ ταχύτερη, τιμή όμως που αποτελεί μεμονωμένο περιστατικό. Ο παρακάτω πίνακας παρουσιάζει τους χρόνους εκτέλεσης της Direct Tiles υλοποίησης για μέγεθος προβολών εισόδου 256 για όλες τις υπολογιστικές συσκευές.



Σχήμα 6.13 : Χρόνοι εκτέλεσης της direct workgroup μεθόδου για μέγεθος προβολών = 256 σε όλες τις υπολογιστικές συσκευές.

6.2.4. Σύγκριση της direct μεθόδου με την direct recursive

Σχετικά με την Direct Recursive μέθοδο, η Direct είναι ταχύτερη 1.5X έως 2.5X φορές σε όλες τις περιπτώσεις εκτός από τα πολύ μικρά όρια βάθους αναδρομής όπου υπάρχει ένας μεγάλος αριθμός σειριακών εκτελέσεων και η επιτάχυνση μπορεί να φτάσει μέχρι και 100X. Σε αυτές τις περιπτώσεις ο αλγόριθμος εκτελείται κυρίως στο software και προστίθενται καθυστερήσεις του λειτουργικού της OpenCL στον ARM επεξεργαστή.



Σχήμα 6.14 : Χρόνοι εκτέλεσης της direct recursive μεθόδου για μέγεθος προβολών 256 σε όλες τις υπολογιστικές συσκευές.

6.3. Κατανάλωση ενέργειας

Προκειμένου να ολοκληρωθεί η μελέτη, οι παρακάτω πίνακες παρουσιάζουν την χρήση υπολογιστικών πόρων και την κατανάλωση ενέργειας. Στον πρώτο πίνακα παρουσιάζεται η κατανάλωση ισχύος και πόρων στο FPGA για τις διάφορες υλοποιήσεις, ενώ στον δεύτερο παρουσιάζεται η κατανάλωση ισχύος στις συσκευές CPU και GPU όπου η κατανάλωση ισχύος είναι ίδια για όλες τις υλοποιήσεις σε κάθε μια συσκευή.

Υλοποιήσεις	Resources			
	ALMs	BRAM	DSPs	Power(mW)
Direct	14320 (45%)	999232 (25%)	10 (11%)	1128,33
Direct Tiles	18413 (57%)	1099200 (27%)	18 (21%)	1287,54
Direct Workgroups	14320 (45%)	999232 (25%)	10 (11%)	1128,33

Πίνακας 6.2 : Κατανάλωση ισχύος και πόρων στο FPGA για τις υλοποιήσεις Direct, Direct Tiles και Direct Workgroup

Power(mW)			
CPU1	CPU2	GPU1	GPU2
TDP=47 W	TDP=35 W	TDP=50 W	TDP =14W
30550	22750	36660	12220

Πίνακας 6.3 : Κατανάλωση ισχύος στις CPU1, CPU2, GPU1 και GPU2

Όπως γίνεται αντιληπτό και στους δύο πίνακες, οι εφαρμογές της OpenCL δίνουν καλύτερα αποτελέσματα κάνοντας την προτεινόμενη μέθοδο αξιόπιστη. Τέλος, υπολογίζεται η καταναλισκόμενη ενέργεια ως το γινόμενο της ηλεκτρικής ισχύος επί τον χρόνο στον οποίο καταναλώθηκε αυτή.

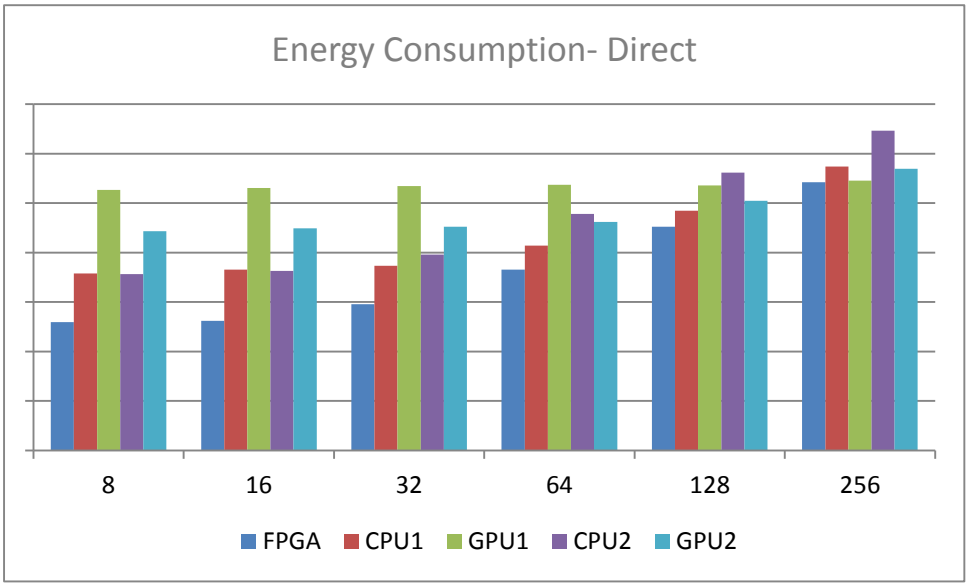
Έτσι με βάση τον τύπο:

$$P = \frac{E}{t} \tag{6.2}$$

όπου P είναι η ηλεκτρική ισχύς σε watt (W), E η κατανάλωση ενέργειας σε joule (J) και t ο χρόνος σε seconds (s) προκύπτει η συνολική κατανάλωση ενέργειας ανά υλοποίηση για το FPGA.

	Energy(μJ)						Average
FPGA	396,27	416,47	908,87	4527,88	33462,21	265110,84	50803,75
CPU1	3775,98	4515,29	5392,08	13857,48	70433,03	554259,49	108705,56
GPU1	186210,80	201952,61	221349,41	235243,55	229509,93	284950,85	226536,19
CPU2	3687,78	4261,08	9186,45	60924,50	414043,18	2919910,18	568668,86
GPU2	26991,54	31193,99	33180,97	41512,56	111458,62	496447,28	123464,16

Πίνακας 6.4 : Κατανάλωση ενέργειας σε κάθε συσκευή



Σχήμα 6.15 : Κατανάλωση ενέργειας σε κάθε συσκευή

Συνολικά, όλα τα πειραματικά αποτελέσματα δείχνουν ότι η OpenCL οδηγεί σε βέλτιστες υλοποιήσεις υλικού FPGA, συλλαμβάνοντας την εσωτερική τους δομή και το διαθέσιμο παραλληλισμό με έναν ιδανικό τρόπο καταναλώνοντας ενέργεια λιγότερη του ενός Watt.

7.Επίλογος- Μελλοντικές προεκτάσεις

Στην παρούσα διπλωματική εργασία προτάθηκαν διαφορετικές υλοποιήσεις κώδικα προκειμένου να υλοποιηθεί ο αλγόριθμος της οπισθοπροβολής τόσο σε software όσο και σε hardware. Η OpenCL χρησιμοποιήθηκε ως η γλώσσα εισόδου και ένα SoC FPGA αλλά και GPUs χρησιμοποιήθηκαν ως πλατφόρμες υλοποίησης.

Μετά από εκτενείς πειραματισμούς, έγινε φανερό ότι οι καθαρά OpenCL υλοποιήσεις οι οποίες εκκινούν έναν νέο πυρήνα για κάθε εικονοστοιχείο εξόδου, δίνουν καλύτερα αποτελέσματα κυρίως στην ταχύτητα αλλά και στην κατανάλωση ισχύος. Τα αποτελέσματα αυτά δείχνουν ότι η συγκεκριμένη μεθοδολογία είναι πολλά υποσχόμενη και ότι η OpenCL , μετά από πολλές ερευνητικές προσπάθειες στον τομέα του highlevel hardware design, μπορεί να αποτελέσει τον βασικό τρόπο αξιοποίησης του παραλληλισμού που είναι διαθέσιμος στα FPGA με φυσικό τρόπο.

Σαν μελλοντική δουλειά, προτείνεται:

- η περαιτέρω αξιοποίηση του συγκεκριμένου tool flow ώστε να υλοποιηθούν κι άλλες εφαρμογές μεγάλων απαιτήσεων και να μπορούν να γενικευτούν τα αποτελέσματα της παρούσας διπλωματικής.
- η χρήση πιο δυνατών FPGA συσκευών ώστε να συνδυαστούν και να συγκριθούν με ετερογενείς αρχιτεκτονικές κάνοντας χρήση των CPUs, DSPs, GPUs και FPGAs και να καταγραφεί συνολική επιτάχυνση της εφαρμογής.
- η υλοποίηση των συγκεκριμένων παράλληλων εφαρμογών με βάση τον αλγόριθμο της οπισθοπροβολής και για μεγέθη 512 και 1024 ώστε να μπορούν να προκύψουν πιο γενικά συμπεράσματα για την επεκτασιμότητα των υλοποιήσεών.

Βιβλιογραφία

[1] B. Barney, «Introduction to Parallel Computing,» [Ηλεκτρονικό]. Available: https://computing.llnl.gov/tutorials/parallel_comp/.

[2] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry και D. Schaa, Heterogeneous Computing with Opencl, 225 Wyman Street, Waltham, MA 02451, USA: Morgan Kaupmann, 2012, pp. 1-13.

[3] «Wikipedia, the free encyclopedia,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Heterogeneous_computing.

[4] M. Zahran, «New York University,» [Ηλεκτρονικό]. Available: http://cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/lecture11.pdf.

[5] R. Tay, σε *OpenCL Parallel Programming Developing Cookbook*, BIRMINGHAM - MUMBAI, PACKT Publishing, 2013, pp. 7-15.

[6] K. O. W. Group, «The OpenCL Specification Version: 1.2,» Aaftab Munshi.

[7] «af inventions - embedded electronics,» [Ηλεκτρονικό]. Available: http://www.af-inventions.de/en/our-services/fpga-design/fpga-advantages/.

[8] O. Segal, N. Nasiri, M. Margala και W. Vanderbauwhede, «High Level Programming of FPGAs for HPC and Data Centric Applications,» *High Performance Extreme Computing Conference (HPEC), 2014 IEEE,* pp. 1 - 3, 9-11 Sept. 2014.

[9] Σ.Παυλόπουλος, Δ.Κουτσούρης και Κ.Νικήτα, Ιατρικά απεικονιστικά συστήματα, Εκδόσεις Τζιόλα, 2004.

[10] Altera. [Ηλεκτρονικό]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[11] Altera. [Ηλεκτρονικό]. Available: https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html.

[12] Terasic. [Ηλεκτρονικό]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=836.

[13] «cpuboss,» [Ηλεκτρονικό]. Available: http://cpuboss.com/cpu/Intel-Core-i7-4710MQ.

[14] «cpuboss,» [Ηλεκτρονικό]. Available: http://cpuboss.com/cpu/Intel-Core-i5-520M.

[15] «GEFORCE,» [Ηλεκτρονικό]. Available: http://www.geforce.com/hardware/notebook-gpus/geforce-310m/specifications.

[16] S.Basu και Y.Bresler, «O(N^2 log N) Filtered Backprojection Reconstruction Algorithm for Tomography,» *IEEE Trans. on Image Processing vol.9,* pp. 1760-1772, October 2000.

[17] T. Pipatsrisawat, A. Gacic, F. Franchetti, M. Püschel και J. M. F. Moura, «Performance analysis of the filtered backprojection image reconstruction algorithms.,» *IEEE international conference on acoustics speech and signal processing (ICASSP),* pp. 153-156, 19-23 March 2005.

[18] S. Gilliland, C. Boulet, T. Gonnot και J. Saniie, «Multidimensional representation of ultrasonic data processed by reconfigurable ultrasonic system-on-chip using OpenCL high-level synthesis,» *International Ultrasonics Symposium. IEEE,* p. 1936–1939., 2014.

[19] J. Dongarra, R. Graybil, W. Harrod, R. Lucas, E. Lusk και P. Luszczek, «DARPA's HPCS program: History, models, tools, languages,",» *Advances in Computers vol 72,* pp. 1-100, 2008.

[20] G. Martin και G. Smith, «High-level synthesis: Past, present, and future,» *IEEE Design and Test of Computers vol26,* pp. 18-25, 2009.

[21] M. Lin, I. Lebedev και J. Wawrzynek, «OpenRCL: Low-power highperformance computing with reconfigurable devices,» *International Conference on Field Programmable Logic and Applications IEEE,* pp. 458-463, 2010.

[22] M. Owaida, N. Bellas, K. Daloukas και C. D. Antonopoulos, «Synthesis of platform architectures from OpenCL programs,» *International Symposium on Field-Programmable Custom Computing Machines IEEE,* pp. 186-193, 2011.

[23] M. Owaida, N. Bellas, C. D. Antonopoulos, K. Daloukas και C. Antoniadis, «Massively parallel programming models used as hardware description languages: The opencl case,» *International Conference on Computer-Aided Design. ACM/IEEE,* p. 326–333., 2011.

[24] D. Chen και D. Singh, «Invited paper: Using OpenCL to evaluate the efficiency of CPUs, GPUs and FPGAs for information filtering,» *22nd International Conference on Field Programmable Logic and Applications IEEE,* pp. 5-12, 2012.

[25] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras και D. P. Singh, «From OpenCL to high-performance hardware on

FPGAs," in 22nd International Conference on Field Programmable Logic and Applications,» *IEEE,* pp. 531-534, 2012.

[26] G.Economakos, «ESL as a gateway from OpenCL to FPGAs: Basic ideas and methodology evaluation,» *16th Panhellenic Conference on Informatics IEEE,* pp. 80-85, 2012.

[27] K. Shagrithaya, K. K?pa και P. Athanas, «Enabling development of OpenCL applications on FPGA platforms,» *4th International Conference on Application-Specific Systems, Architectures and Processors IEEE,* pp. 20-26, 2013.

[28] K. Shagrithaya, K. Kpa και P. Athanas, «Enabling development of OpenCL applications on FPGA platforms,» *24th International Conference on Application-Specific Systems, Architectures and Processors IEEE,* p. 26–30, 2013.

[29] J. Coole και G. Stitt, «Fast, flexible high-level synthesis from OpenCL using reconfigurable contexts,» *IEEE Micro,* τόμ. 34, p. 42–53, 2014.

[30] P. O. Jaaskelainen, C. S. d. L. Lama, P. Huerta και J. H. Takala, «Opencl-based design methodology for application-specific processors,» *10th International Conference on Embedded Computer Systems Architectures, Modeling and Simulation. IEEE,* p. 223–230, 2010.

[31] J. Coole και G. Stitt, «Fast, flexible high-level synthesis from OpenCL using reconfigurable contexts,» *IEEE Micro, vol. 34, no. 1,* p. 42–53, 2014.

[32] I. Lebedev, S. Cheng, A. Doupnik, J. Martin, C. Fletcher, D. B. Lin και J. Wawrzynek, «MARC: A many-core approach to reconfigurable computing,» *in International Conference on Reconfigurable Computing IEEE,* pp. 7-10, 2012.

[33] E. Cartwright, S. Ma, D. Andrews και M. Huang, «Creating HW/SW co-designed MPSoPCs from high level programming models,» *in International Conference on High Performance Computing and Simulation IEEE,* pp. 554-560, 2011.

[34] H. Tomiyama, T. Hieda, N. Nishiyama, N. Etani και T. I., «SMYLE OpenCL: A programming framework for embedded many-core SoCs,» *in 18th Asia and South Pacific Design Automation Conference ACM/IEEE,,* pp. 565-567, 2013.

[35] J. Hoffmann και M. Bogdan, «Cyfield-RISP: An OpenCL-generated processor for reconfigurable hardware,» *in International Conference on High Performance Computing and Simulation. IEEE,* pp. 703-706, 2014.

[36] V. Mirian και P. Chow, «Using an OpenCL framework to evaluate interconnect implementations on FPGAs,» *24th International Conference on Field Programmable Logic and Applications. IEEE,* 2014.

[37] J. Andrade, V. Silva και G. Falcao, «From OpenCL to gates: the FFT,» *Global Conference on Signal and Information Processing IEEE,* p. 1238–1241, 2013.

[38] D. Chen και D. Singh, «Fractal video compression in OpenCL: An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms,» *18th Asia and South Pacific Design Automation Conference. ACM/IEEE,,* p. 297–304, 2013.

[39] S. Gao και J. Chritz, «Characterization of OpenCL on a scalable FPGA architecture,» *International Conference on Reconfigurable Computing and FPGAs IEEE,* 2014.