Ατομική Διπλωματική Εργασία

EXECUTING PARALLEL APPLICATIONS ON THE PARALLELLA BOARD

Πιερής Χατζηναστάση

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Executing parallel applications on the Parallella Board

Πιερής Χατζηαναστάση

Επιβλέπων Καθηγητής

Δρ. Ευριπίδου Παρασκευάς

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Ευχαριστίες

Όταν φτάνεις στο τέλος μιας επίμονης και επίπονης προσπάθειας για την κατάκτηση ενός στόχου και σκοπού και αναλογίζεσαι την πορεία που διήνυσες μέχρι να φτάσεις έως εκεί, τότε συνειδητοποιείς τη συνολική σου πορεία και τους ανθρώπους που σε στήριξαν, σε εμψύχωσαν και σε επηρέασαν θετικά. Στους ανθρώπους αυτούς οφείλω από καρδίας άπειρες ευχαριστίες για την έμπρακτη υλική και ψυχολογική βοήθεια που μου πρόσφεραν κατά την διάρκεια της πορείας μου αυτής.

Αρχικά, θα ήθελα να εκφράσω την εκτίμηση και τις θερμές μου ευχαριστίες προς τον επιβλέποντα καθηγητή μου Δρ. Ευριπίδου Παρασκευά, τόσο προς την τιμή και την εμπιστοσύνη του προς το πρόσωπό μου, όσο και για την ενθάρρυνσή του να ασχοληθώ με το θέμα αυτό. Ευχαριστώ επίσης θερμά τον υποψήφιο διδάκτορα Γιώργο Ματθαίου για τη πολύτιμη και ανιδιοτελή βοήθεια που μου προσέφερε κατά την διάρκεια της έρευνας και συγγραφής αυτής της εργασίας.

Επιπλέον, θα ήθελα να ευχαριστήσω τον ακαδημαϊκό μου σύμβουλο Δρ. Pedro Trancoso, για τις σημαντικές και καίριες συμβουλές του, όσο και για τη καθοδήγηση του στο πρόγραμμα σπουδών μου.

Τέλος, θα ήταν μεγάλη και ασυγχώρητη παράλειψη, αν δεν ευχαριστούσα τους γονείς μου, Πασχάλη και Δήμητρα, για την ψυχολογική, οικονομική και ποικιλότροπη βοήθεια που μου πρόσφεραν όλα αυτά τα χρόνια. Χωρίς τη δική τους πολύτιμη συμβολή θα ήταν αδύνατη η διεκπεραίωση της παραπάνω πορεία μου.

Περίληψη

Η παρούσα διπλωματική εργασία που κατατίθεται προς έγκριση, ασχολείται με την υλοποίηση παράλληλων εφαρμογών/αλγορίθμων στο Parallella Board και αποτελείται από πέντε κεφάλαια.

Αρχικά, στο πρώτο κεφάλαιο γίνεται μια πρώτη εισαγωγή με τη διασαφήνιση του όρου Parallella Board, κατατίθεται ο σκοπός αυτής της διπλωματικής εργασίας και περιγράφονται τα τεχνικά χαρακτηριστικά του καθώς και ένας γρήγορος οδηγός εγκατάστασης. Στο δεύτερο κεφάλαιο γίνεται μία περιεκτική ανάλυση της αρχιτεκτονικής του Parallella Board, το design decisions, memory structure και epiphany chip implementation.

Ακολούθως, δηλαδή στο τρίτο κεφάλαιο γίνεται μια περιγραφεί του πως προγραμματίζουμε ένα αλγόριθμό στο Parallella Board και στο framework OpenMP.

Στο τέταρτο κεφάλαιο γίνεται περιγραφή των αλγορίθμων, πώς υλοποιήθηκαν στο framework του board και λεπτομέρειες υλοποίησης κώδικα, ενώ στο τέλος του κεφαλαίου περιγράφονται τα framework OpenMP. Στο πέμπτο κεφάλαιο παρουσιάζονται όλα τα αποτελέσματα σε γραφικές παραστάσεις και γίνεται ο σχολιασμός των γραφικών αυτών.

Τέλος, στο τελευταίο κεφάλαιο συνοψίζονται τα συμπεράσματα, περιγράφονται τα προβλήματα που είχα αντιμετωπίσει και γίνεται μια αναφορά για μελλοντική έρευνα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

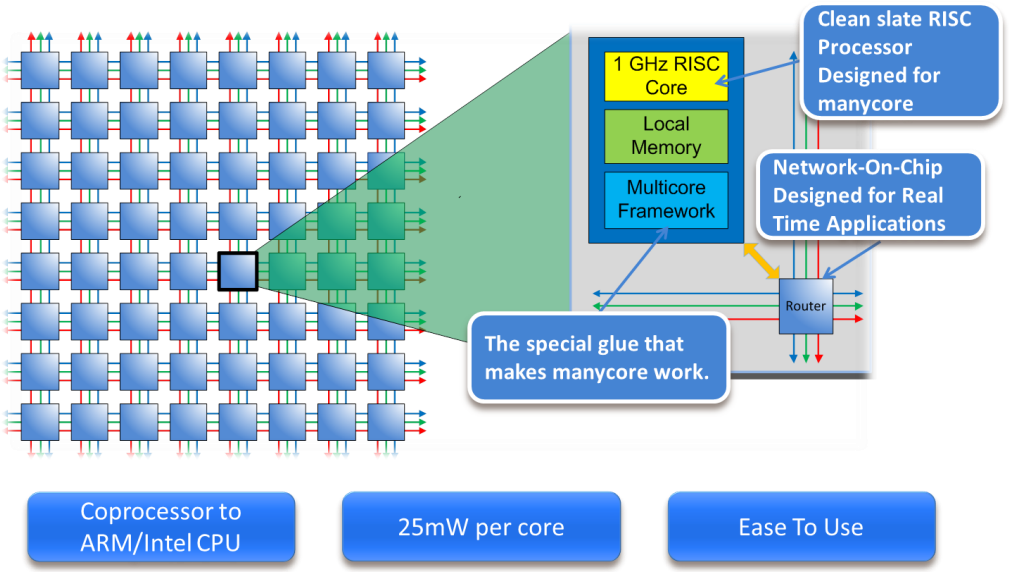
1.1 Τι είναι το Parallella Board.

Ο υπολογιστής Parallella, είναι ένας υπολογιστής υψηλής απόδοσης, μεγέθους πιστωτικής κάρτας υπολογιστή με βάση το Epiphany multi-core chips από την Adapteva. Το Epiphany multi-core είναι μια επεκτάσιμη αρχιτεκτονική κοινόχρηστης μνήμης, η οποία διαθέτει έως και 4.096 επεξεργαστές σε ένα ενιαίο chip και συνδέεται μέσω ενός μεγάλου εύρους ζώνης on-chip δίκτυο[1]. Κάθε Epiphany πυρήνας επεξεργαστή περιλαμβάνει ένα μικροσκοπικό επεξεργαστή RISC υψηλής απόδοσης κινητής υποδιαστολής, ο οποίος χτίστηκε από το μηδέν για την επεξεργασία πολλαπλών πυρήνων υψηλού εύρους ζώνης του συστήματος τοπικής μνήμης και ένα εκτεταμένο σύνολο χτισμένο σε χαρακτηριστικά του υλικού για την επικοινωνία πολλαπλών πυρήνων[1]. Η Parallella μπορεί να χρησιμοποιηθεί ως ένας αυτόνομος υπολογιστής ή ως μια ενσωματωμένη συσκευή ή ως συστατικό σε μια κλίμακα από σύμπλεγμα παράλληλου διακομιστή.

Η Parallella περιλαμβάνει ένα χαμηλής ισχύος επεξεργαστής διπλού πυρήνα ARM A9 που βρίσκεται στο FPGA και διατηρεί περισσότερες από τις δημοφιλής διανομές Linux, συμπεριλαμβανομένου και του Ubuntu. Ο επεξεργαστής ARM A9 είναι μια δημοφιλής επιλογή γενικής χρήσης για χαμηλής ισχύος ή θερμικά περιορισμένου κόστους, για ευαίσθητες συσκευές. Ο επεξεργαστής είναι μια ώριμη επιλογή που έχει εισαχθεί το 2008 και παραμένει μια πολύ δημοφιλής επιλογή σε smartphones και

ψηφιακές τηλεοράσεις για τους καταναλωτές. Ως μοναδικής λύσης επεξεργαστής, ο επεξεργαστής ARM A9 προσφέρει μια συνολική αύξηση της απόδοσης περίπου πάνω από το 50% σε σύγκριση με τις λύσεις ARM A8. Ο επεξεργαστής A9 είναι διαθέσιμος σε μια σειρά που υποστηρίζουν την τεχνολογία ARM[2].

Το πλέγμα των ανεξάρτητων πυρήνων συνδέονται μεταξύ τους με ένα γρήγορο chip στο δίκτυο, μέσα σε μια κατανεμημένη κοινή μνήμη με αρχιτεκτονική. Η Epiphany co-processor είναι σε ANSI-C και OpenCL και λειτουργεί σε συνεργασία με το πρότυπο μικροεπεξεργαστών για να παρέχει άνευ προηγουμένου, επίπεδο επεξεργασίας σε πραγματικό χρόνο και για να αποδίδει τη δύναμη για τις κινητές συσκευές όπως smartphones και υπολογιστές tablets. Καθώς και για να βελτιώνει τα επίπεδα απόδοσης σε μια σειρά από άλλες παράλληλες υπολογιστικές πλατφόρμες[1].



Εικόνα 1: The Epiphany Multicore Accelerator. [1]

Η Parallella προτείνεται ως ένας ιδανικός μελλοντικός υπολογιστής για όσους ασχολούνται με τον τομέα των παράλληλων υπολογιστών, δηλαδή με τον παράλληλο προγραμματισμό.

1.2 Σκοπός

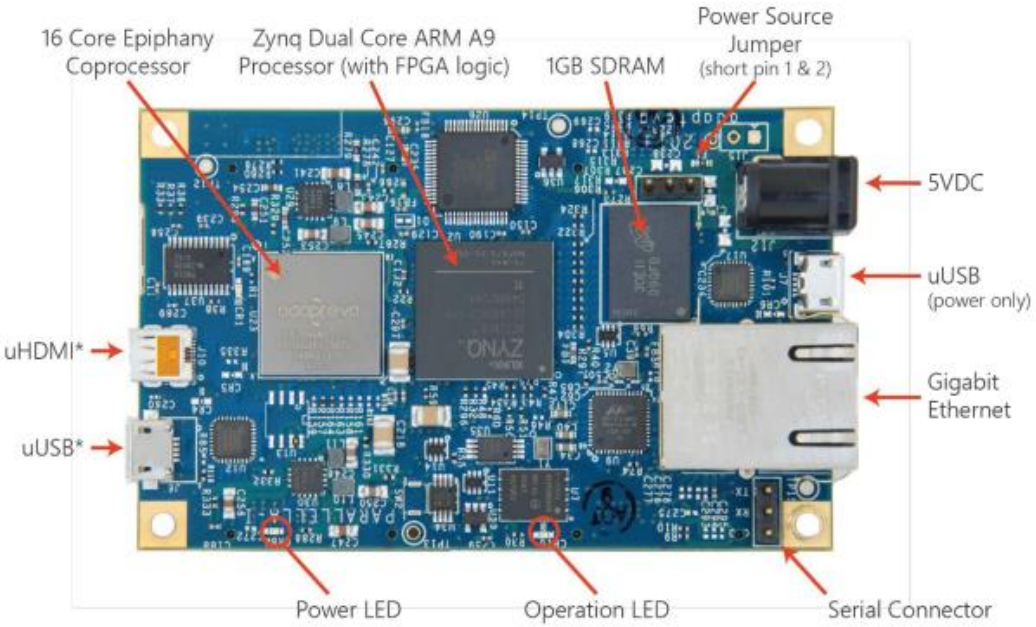
Ο κύριος σκοπός αυτής της διπλωματικής εργασίας, είναι ή κατάδειξη του τρόπου της υλοποίησης παράλληλων αλγορίθμων/εφαρμογών στο Parallella Board. Καθώς και

η υλοποίηση τους στα γνωστά framework OpenMP και OpenCL στο Arm που βρίσκεται στο FPGA.

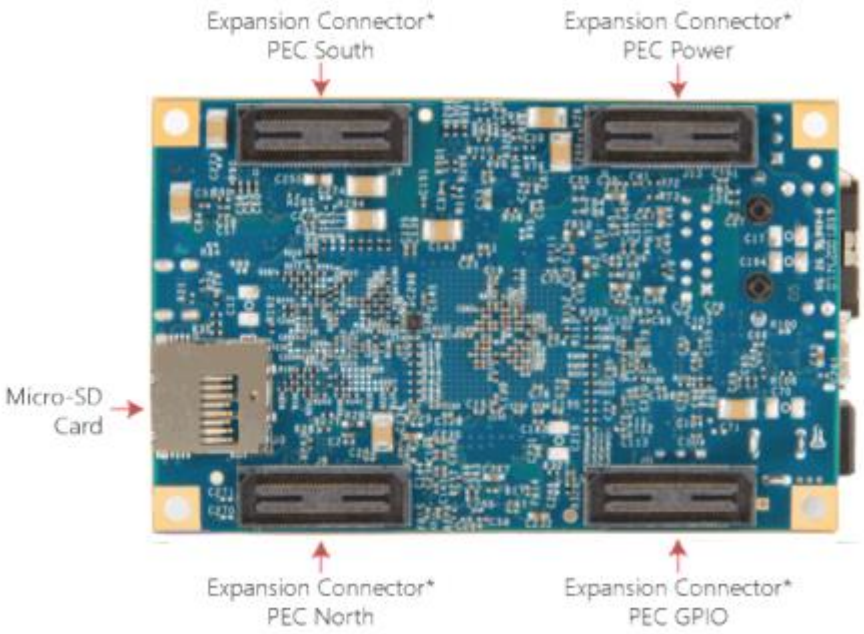
1.3 Τεχνικά χαρακτηριστικά.

Το Parallella Board αποτελείται από:

- Zynq-Z7010 or Z7020 Dual-core ARM A9 CPU
- 16-core Epiphany Coprocessor
- 1GB RAM
- MicroSD Card
- USB 2.0
- Up to 48 GPIO signal
- Gigabit Ethernet
- HDMI port
- Linux Operating System
- 54mm x 87mm form factor



Εικόνα 2: Πάνω όψη πλακέτας.[3]



Εικόνα 3: Κάτω όψη πλακέτας.[3]

Κεφάλαιο 2

Αρχιτεκτονική Parallella Board

2.1 Εισαγωγή

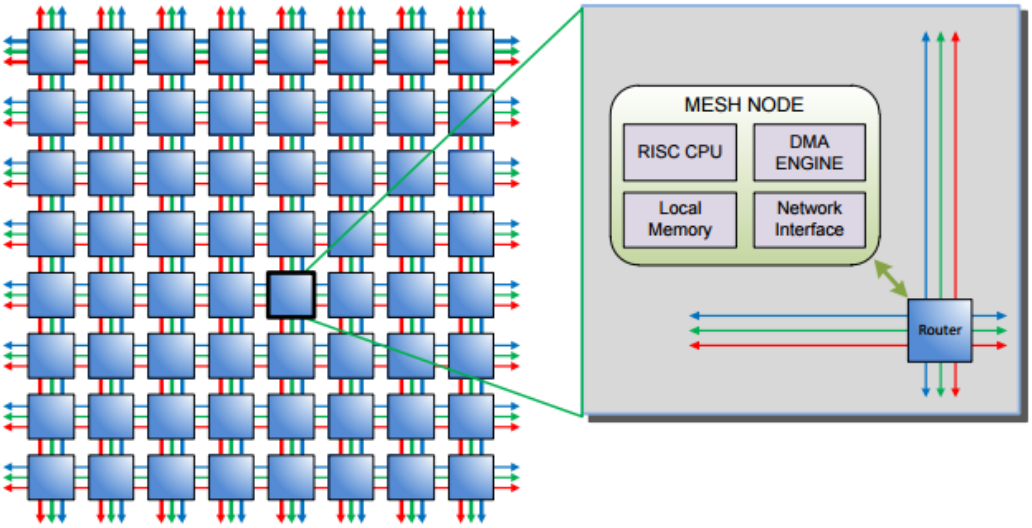
Σήμερα, επειδή η υπολογιστική επίδοση παρουσιάζει ελλειπτικό χαρακτήρα και άρα δεν είναι σε ολοκληρωμένη μορφή, στην προσπάθειά μας να αυξήσουμε την επίδοσής της, επιχειρήσαμε την ενσωμάτωση πολλών processor core σε ένα chip, το οποίο περιγράφεται ως multicore ή manycore αρχιτεκτονικές. Παραδείγματα από manycore προσπάθειες είναι το picoArray με 322 cores, το Ambric με 336 cores και το Tile64 με 64 cores. Όλοι αυτοί οι επεξεργαστές σχεδιάστηκαν για πολύ συγκεκριμένες και προηγμένες αγορές και έχουν μεγάλη κατανάλωση ενέργειας σε ισχύ για τα περισσότερα mobile συστήματα.

Στο συγκεκριμένο σημείο θα επιχειρήσουμε να περιγράψουμε τις προσπάθειες που καταβλήθηκαν, αλλά και τα αποτελέσματα της σχεδίασης μιας αρχιτεκτονικής manycore, με υψηλή απόδοση και αποτελεσματικά ενεργειακής και την χρήση της στα ενσωματωμένα (embedded) συστήματα. Την αρχιτεκτονική Epiphany.

Αρχικά θα αναφερθούμε με συντομία στην ιστορία και στους στόχους της σχεδίασης αυτής για την αρχιτεκτονική. Στη συνέχεια θα περιγράψουμε την αρχιτεκτονική στην οποία καταλήξαμε, τα processor core της, τη μνήμη και την επικοινωνία. Ακολούθως θα παρουσιάσουμε κάποια εργαλεία τα οποία είναι διαθέσιμα και σε εξέλιξη για το Epiphany και θα κοιτάξουμε κάποια από τα πρόσφατα application τα οποία αναπτύχθηκαν για το Epiphany.

Περιγραφή Αρχιτεκτονικής:

Η αρχιτεκτονική Epiphany αποτελείται από ένα 2D mesh και από 16 επεξεργαστές κόμβους (“eNodes”) τοποθετημένους σε ένα πλέγμα 4 x 4. Ο καθένας περιέχει ένα 32-bit floating point RISC CPU(“eCore”), πολυεπίπεδη κύρια μνήμη, μια μηχανή για άμεση πρόσβαση στη μνήμη και ένα network interface. Κάθε κόμβος ενώνεται σε ένα Network on chip(NoC) (“eMesh”) μέσω του Network interface.



Εικόνα 4: An Implementation of the Epiphany Architecture.[6]

Η αρχιτεκτονική Epiphany δεν περιέχει την παραδοσιακή L1/L2 hardware caching, γιατί στοχεύει στην μεγιστοποίηση του local storage, memory bandwidth μέσα από multibanked scratchpad memory. Η multibanked local memory υποστηρίζει ταυτόχρονα instruction fetching, data fetching και multicore communication με τέσσερα 8-byte-wide memory banks ανα core.

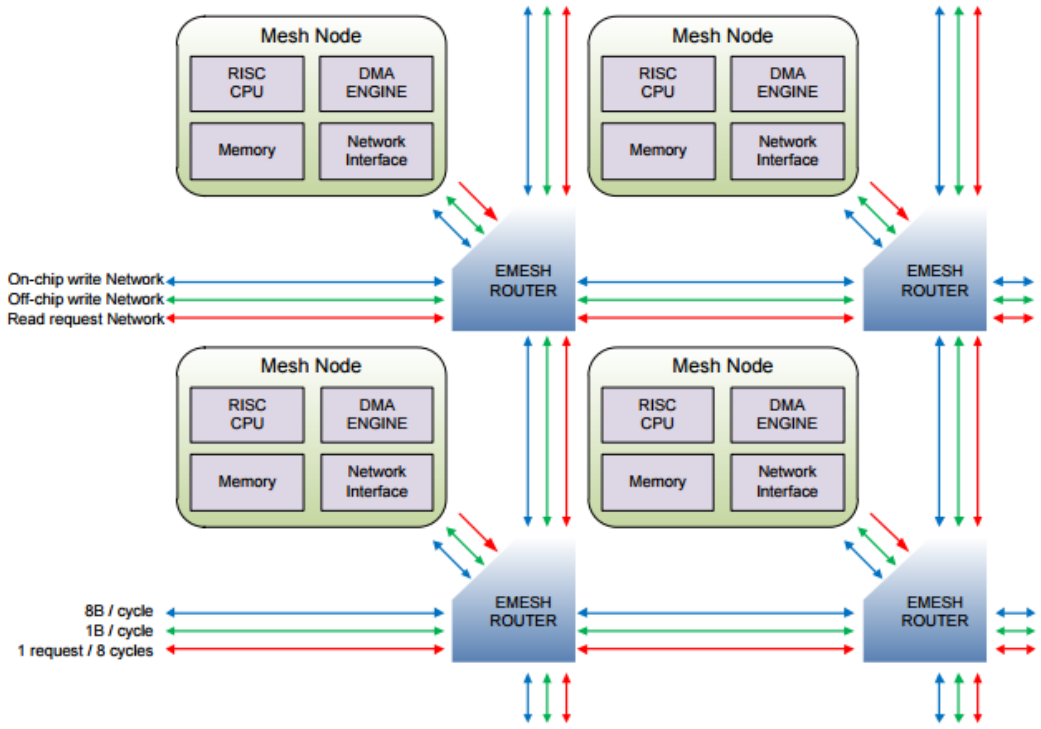
Η epiphany αρχιτεκτονική μνήμης βασίζεται σε ένα flat disturbed memory map κατά το οποίο κάθε node καθορίζεται σε ένα μοναδικό addressable slide της μνήμης. Η συνολική μνήμη χωρίζεται σε blocks των μέχρι 1MB.

Κάθε κόμβος έχει επίσης ένα id, το οποίο καθορίζεται από δείκτη (σειρά, στήλη) στο πλέγμα. Κάθε πυρήνας μπορεί να έχει πρόσβαση σε κάθε άλλο πυρήνα, χρησιμοποιώντας αυτό το σύστημα συντεταγμένων (μέσω εντολών load/store).

Μια άλλη συνέπεια αυτής της οργάνωσης είναι ότι κάθε πυρήνας θα μπορούσε θεωρητικά να προγραμματιστεί για να διενεργήσει διάφορες εργασίες. Αυτός είναι ένας από τους λόγους για τους οποίους η αρχιτεκτονική Epiphany είναι μια αρχιτεκτονική MIMD (multipliable input multipliable data), σε αντίθεση με την αρχιτεκτονική SIMD (Single instruction, multiple data) βρίσκεται στα GPUs.

Η επικοινωνία στο epiphany υποστηρίζεται από eMesh Network on chip(NoC), το οποίο αποτελείται από 4 ανεξάρτητα 2D scalable mesh networks, το καθένα με τέσσερα διπλά links σε κάθε node.

Ο μηχανισμός routing βασίζεται σε ένα κατανεμημένο address based routing το οποίο προσφέρει 1 cycle wait, που σημαίνει ότι υπάρχει single cycle routing latency per node. Καθένα από τα 4 mesh έχει διαφορετικό σκοπό.



Εικόνα 5: eMesh™ Network-On-Chip Overview.[6]

Για συναλλαγές διαβάσματος και εγγραφής που κάνουν πρόσβαση σε μη τοπική μνήμη. Οι περιορισμοί σειράς της μνήμης χαλαρώθηκαν για να βελτιώσουν την επίδοση. Αυτή η χαλάρωση του συγχρονισμού μεταξύ των εντολών memory-access και των surrounding εντολών αναφέρονται ως μία αδύναμη ταξινόμηση των load και store. Αδύναμη ταξινόμηση σημαίνει ότι ο χρόνος της πραγματικής ολοκλήρωσης των

διαδικασιών μνήμης μπορεί να μην ευθυγραμμίζονται με το πώς εμφανίζονται στη σειρά του source code του προγράμματός.

Σημαντικές λειτουργίες του on-chip mesh network περιλαμβάνουν:

- Βελτιστοποίηση των συναλλαγών εγγραφής πάνω από τις συναλλαγές ανάγνωσης.
- Διαχωρισμός της διακίνησης δεδομένων on-chip και off-chip.
- Λειτουργία χωρίς αδιεξόδους.
- Επεκτασιμότητα.

2.2 Design Decisions

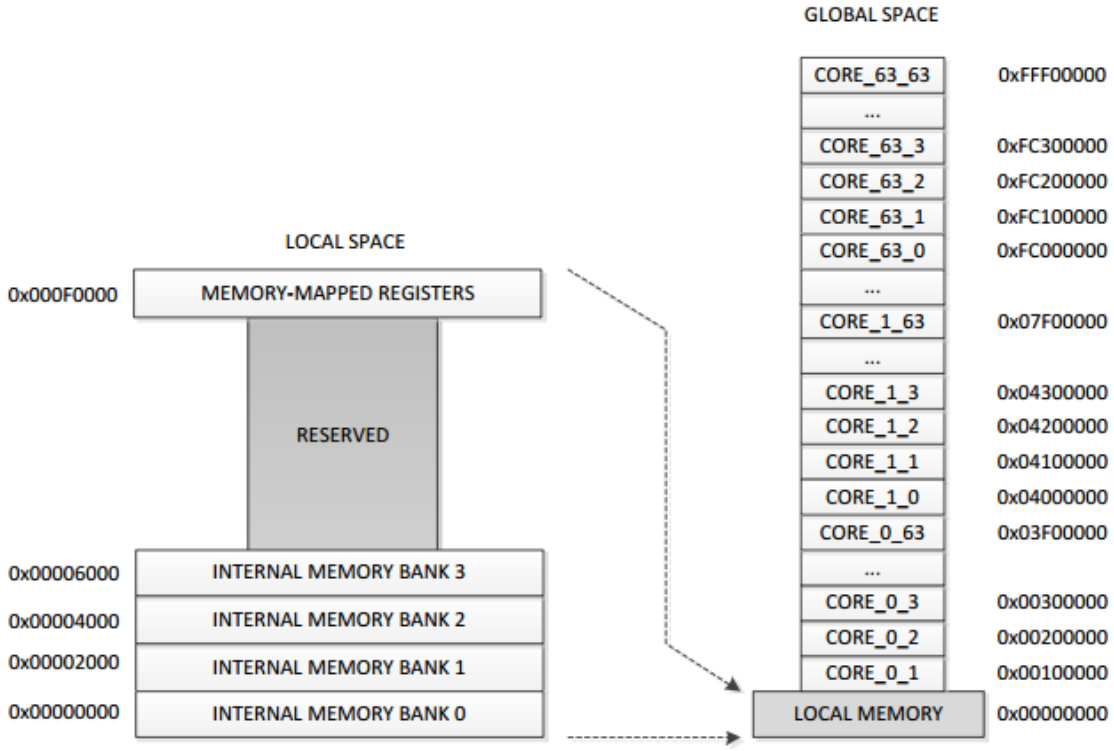
Η φιλοσοφία του eCore CPU στόχευε στην όσο το δυνατό μεγαλύτερη απλούστευσή του. Στο τομέα των massively parallel computers υπήρξε ένας αριθμός από αρχιτεκτονικές με εκπληκτικό σχεδιασμό και απόδοση, αλλά ήταν δύσκολο να προγραμματιστούν. Ο σχεδιαστικός στόχος του Epiphany ήταν να προσφέρει 10 φορές περισσότερο αποτελεσματικά και ενεργειακά από το CPU architecture όμως υποστηρίζοντας ένα ANSI-C προγραμματισμό τον οποίο ένας μέσος προγραμματιστής μπορούσε να χειριστεί.

Τα ακόλουθα C-friendly χαρακτηριστικά περιέχονται στην αρχιτεκτονική epiphany:

- Διπλό προγραμματιστικό θέμα για την ενίσχυση της απόδοσης για σχεδόν όλες τις εφαρμογές.
- Αρχέγονες οδηγίες IEEE κινητής υποδιαστολής για να αφήσει προγραμματιστές να επικεντρώνονται στους μαθηματικούς αλγορίθμους αντί του τελεστή κλιμάκωσης και την αριθμητική ακρίβεια.
- Ένα μεγάλο 64 entry register file για να επιτρέψει αποτελεσματικά loop Unrolling και επαναχρησιμοποίηση μεταβλητών.
- Ένα 64 bit load/store path για να αυξήσει την απόδοση για πολλούς μαθηματικούς αλγόριθμους.
- Χαμηλού κόστους χαρακτηριστικά όπως interrupt controller, debug unit και τους timers για να υλοποιήσουν τα manycores.

2.3 Memory structure

Το παρακάτω σχήμα δίνει μια επισκόπηση της διάταξης μνήμης της κάθε memory bank πάνω στο chip Epiphany.



Εικόνα 6: Epiphany Global Address Map.[7]

Η μνήμη είναι μια ενιαία και επίπεδη με little-endian χώρο διευθύνσεων που οργανώνεται για να αντανακλούν στη διάταξη του πλέγματος 4 x 4 των πυρήνων. Κάθε κόμβος έχει μια μεμονωμένη memory bank των 32KB. Επίσης, σε κάθε κόμβο, υπάρχει μια τοπική μνήμη που είναι οργανωμένη σε 4 των 8KB bank. Για την βέλτιστη της απόδοση, προτείνεται ανεπιφύλακτα ότι τα δεδομένα θα πρέπει να τοποθετούνται σε ξεχωριστές τράπεζες.

Αξίζει να σημειωθεί επίσης, στις τράπεζες της τοπικής μνήμης δεν υπάρχει κρυφή μνήμη, καθώς δεν υπάρχει πάγια πολιτική της συνοχής μεταξύ τους. Επίσης, μετά την επικοινωνία που είχα με τον κ. Andrea Olofsson, ο οποίος εργάζεται στην κατασκευαστική εταιρεία του Parallella Board και τη διεξοδική συζήτηση που είχαμε, καταλήξαμε από κοινού στο συμπέρασμα ότι δεν υπάρχει κρυφή μνήμη για τους ακόλουθους λόγους: Αρχικά, είναι ανάγκη να υπογραμμίσουμε ότι είναι πολύ πιο αποτελεσματικό να μην υπάρχει κρυφή μνήμη σε θέματα παροχής και δύναμης. Ακόμα

ένας λόγος που συνηγορεί στο να μην υπάρχει κρυφή μνήμη, είναι ότι με κρυφή μνήμη δεν κλιμακώνεται καλά, δηλαδή υποθετικά μιλώντας 1000 επεξεργαστές να προσπαθούν να αποκτήσουν πρόσβαση με αναλογία cache miss 1%. Υπάρχει επιπλέον χώρος που προορίζεται για την επέκταση, αλλά δεν αντιμετωπίζεται προς το παρόν. Υπάρχει επίσης κάποιος επιπλέον χώρος που προορίζεται για τις 64 τοπικά μητρώα. 8-bit, 16-bit, 32-bit και 64-bit ακέραιο (int και char) Οι τύποι δεδομένων που υποστηρίζονται, μαζί με τους αριθμούς κινητής υποδιαστολής 32-bit (float).[7]

2.4 Epiphany chip implementation.

Η αρχιτεκτονική epiphany έχει τοποθετηθεί σε 4 διαφορετικά chip αποκορυφώνοντας στη τελευταία $4^{η}$ γενιά 28 nm epiphany iv 64 core design. Το epiphany iv περιέχει 64 πανομοιότυπα floating point eCores, το καθένα με 32KB εσωτερικής μνήμης. [7]

Το epiphany-iv έχει υλοποιηθεί σε ένα 28nm LP CMOS με 9 επίπεδα χαλκού για routing και περιέχει περισσότερα από 160 εκατομμύρια τρανζίστορ σε μια περιοχή των 10mm^2. Το chip υλοποιήθηκε σε μια ιεραρχική tiled layout μεθοδολογία. [7]

Το epiphany-iv έχει υλοποιηθεί σε λιγότερο από 12 εβδομάδες από 2 engineers, σε μια αποτελεσματικότητα των 1M τρανζίστορ κάθε engineer κάθε μέρα. Η ενεργειακή απόδοση του epiphany-iv είναι περισσότερη, λόγω της αρχιτεκτονικής σχεδίασης, παρά του σχεδιασμού κυκλώματος. Κανένα custom logic δεν χρησιμοποιήθηκε την υλοποίηση και shelf 9-track standard cell βιβλιοθήκες χρησιμοποιήθηκαν για την υλοποίηση του design. Η διαρροή της ενέργειας έχει μειωθεί από την μη χρήση LVT threshold τρανζίστορ ενώ η δυναμική ενέργεια έχει μειωθεί από την εκτενή χρήση του clock gating και power saving idle καταστάσεις. [7]

Το epiphany-iv έχει λειτουργήσει μέχρι 800MHz στο εργαστήριο, προσφέροντας μια θεωρητική απόδοση των 102GFLOPS. Θεωρητικά έχει 102 gb/s διχοτομημένη ταχύτητα. 1.6 TB/s από local memory bandwidth και 7.2 gb/s συνολικό off-chip badwidth. Η κορυφαία ενεργειακή αποτελεσματικότητα των 70 gflops/w έχει καταγραφεί στα 0.9V και συχνότητα 500 MHz.[7]

Κεφάλαιο 3

Περιγραφή Προγραμματισμού στο Parallella Board και OpenMP

3.1 Περιγραφή προγραμματισμού για το Parallella Board

3.1.1 Περιγραφή προγράμματος (κώδικα) που τυπώνει «Hello World» για κάθε core.

Αρχικά, ας δούμε με αρκετή λεπτομέρεια πως υλοποιείτε ένα απλό πρόγραμμα που τυπώνει «Hello World» για ένα κάθε core του Epiphany. Θα χρειαστούμε δυο αρχεία τα οποία, θα είναι ανεξάρτητα το ένα αρχείο που μεταγλωττίζεται στο Arm, θα το ονομάσουμε main.c και το άλλο που θα μεταγλωττίζεται στο Epiphany, θα το ονομάσουμε e_main.c.

Στο αρχείο main.c, έχουμε δημιουργήσει αντικείμενο τύπου e_platform_t για την πλατφόρμα του Epiphany και το αντικείμενο τύπου e_epiphany_t για την ομάδα εργασίας του Epiphany. Αφού γίνει αρχικοποίηση του συστήματος, επαναφέρουμε την πλατφόρμα και παίρνουμε τις πραγματικές παραμέτρους του συστήματος. Έπειτα, διαθέτουμε ένα buffer στην κοινόχρηστη μνήμη για το μήνυμα που θα περνά από τα eCore στο Arm(Host).

Αρχικά, δηλώνουμε δύο απαριθμητές επαναλήψεις (for) μέχρι τον αριθμό τέσσερα και οι δύο (που αντιστοιχούν στο πλέγμα 4 x 4). Μέσων των εντολών e_open()

και e_reset_group(), ανοίγουμε ένα ενιαίο core της ομάδας εργασίας (αρχικά θα ανοίξει το 0,0 έπειτα το 0,1 και ούτω κάθε εξής) και επαναφέρουμε το core σε περίπτωση που μια προηγούμενη διαδικασία βρίσκεται σε λειτουργία. Μετά εκτελούμε την εντολή e_load() για να φορτωθεί το πρόγραμμα στην συσκευή πάνω στο επιλεγμένο eCore. Στη συνέχεια διαβάζουμε το μήνυμα από το shared buffer. Αφού εκτυπωθεί το μήνυμα, κλείνει η ομάδα εργασίας.

Τέλος, σε αυτό το αρχείο (main.c), όπως όλα τα προγράμματα που κάνουν χρήση του Epiphany έτσι και τώρα καλούμε της συναρτήσεις e_close() και e_finalize() δεδομένου ότι δεν χρειαζόμαστε πια το chip Epiphany.

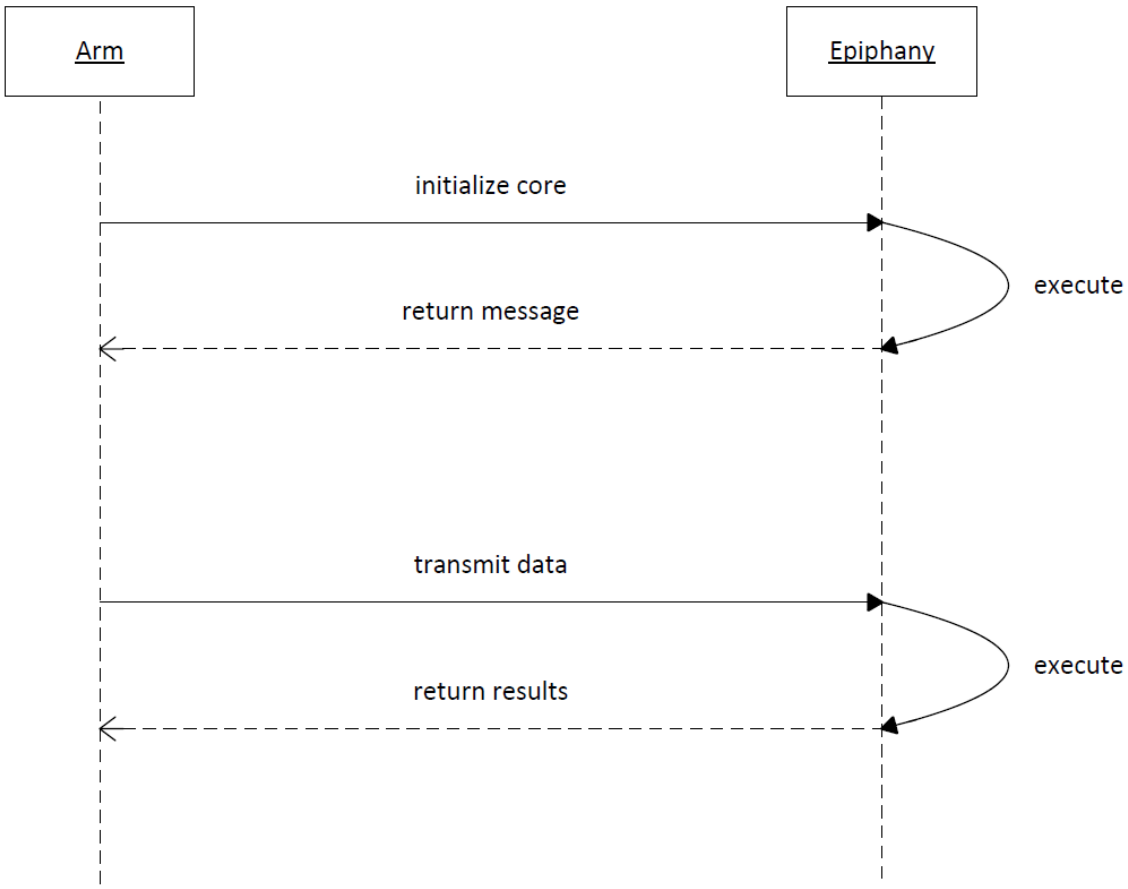
Στο αρχείο e_main.c περιέχεται ο κώδικας που μπορεί να φορτωθεί σε κάθε eCore (πυρήνα Epiphany) από το Arm/Host - κώδικα που βρίσκεται στο άλλο αρχείο (main.c). Στην αρχή ζητάμε από το core να δώσει το core id του ενώ στην εσωτερική μνήμη δεν δικαιούμαστε να χρησιμοποιούμε τις συναρτήσεις printf(). Έτσι χρησιμοποιούμε ένα πίνακα από χαρακτήρες τοποθετημένο στην shared DRAM και στο τέλος του αρχείου χρησιμοποιούμε την συνάρτηση sprintf() για να αποθηκεύσουμε στον πίνακα το μήνυμα που θέλουμε να τυπωθεί "Hello World from core ‘‘διεύθυνση‘‘ ".

3.1.2 Περιγραφή προγράμματος υλοποίησης (με σχεδιάγραμμα ακολουθιακό) αλγόριθμού Reduction.

Αυτός ο αλγόριθμος δίνει το εσωτερικό άθροισμα δυο πινάκων είναι το άθροισμα όλων των στοιχείων των πινάκων αυτών. Για παράδειγμα A = {2, 5, 7} και B = {1, 3, 6}, αναλυτικά το εσωτερικό άθροισμα αυτών των πινάκων είναι (2 + 1) + (5 + 3) + (7 + 6) = 3 + 8 + 13 = 24.

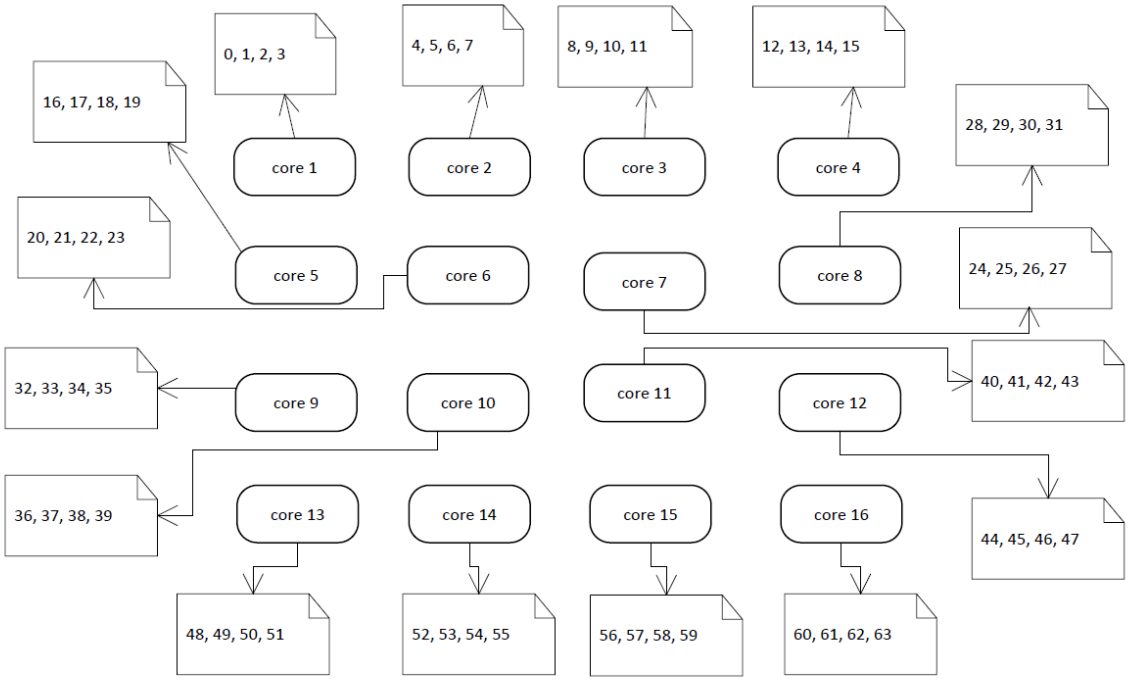
Σημείωση: ικανή και αναγκαία προϋπόθεση είναι το μήκος των δυο πινάκων να είναι το ίδιο.

Στη παρακάτω εικόνα φαίνεται η επικοινωνία του Arm με το Epiphany:



Εικόνα 7: Επικοινωνία Arm με Epiphany.

Στη πιο κάτω εικόνα φαίνεται πως διανέμονται τα δεδομένα (από κάθε πίνακα A και B) σε κάθε core ξεχωριστά:



Εικόνα 8: Διαχωρισμός δεδομένων σε κάθε core ξεχωριστά (για N = 64 και 16 core)

3.2 OpenMP

Αν θα θέλαμε να προσδιορίσουμε τι είναι το OpenMP, θα λέγαμε ότι είναι μία διεπαφή προγράμματος εφαρμογής (API), που μπορεί να χρησιμοποιηθεί για να κατευθύνει ρητά multi-threaded και κοινόχρηστη μνήμη με παραλληλισμό. Αυτή η διεπαφή αποτελείται από τρία κύρια στοιχεία API τα οποία δίνουν οδηγίες κατά τη μεταγλώττιση (η παράμετρος- fopenmp). Είναι επίσης Runtime Library Routines και Environment Variables.

Δυο συντομογραφίες για το OpenMP που χρησιμοποιούνται είναι οι εξής:

- Short version: Open Multi-Processing

- Long version: Open specifications for Multi-Processing via collaborative work between interested parties from the hardware and software industry, government and academia.

Ως προς τη χρήση του το OpenMP, θα λέγαμε ότι δεν χρησιμοποιείτε για παράλληλα συστήματα κατανεμημένης μνήμης (από μόνο του), και αναγκαστικά εφαρμόζονται με τον ίδιο τρόπο. Δεν δίνει εγγύηση για την πιο αποτελεσματική χρήση της κοινόχρηστης μνήμης, ούτε για τον έλεγχο τυχόν εξαρτήσεως των δεδομένων, για συγκρούσεις δεδομένων, συνθήκες, αδιέξοδα κλπ. Επίσης, δεν έχει ως καθήκον του να ελέγξει τις τυχόν ακολουθίες κώδικα οι οποίες προκαλούν ένα πρόγραμμα να χαρακτηριστεί ως μη συμμορφούμενο. Προορισμός του είναι η κάλυψη compiler που δημιουργεί αυτόματη παραλληλοποίηση και δίδει οδηγίες προς τον compiler για να βοηθήσει στον παραλληλισμό. Σχεδιάστηκε επίσης για να διασφαλίζει τη σύγχρονη είσοδο ή έξοδο στο ίδιο αρχείο κατά την παράλληλη εκτέλεσή τους. Ο κάθε προγραμματιστής είναι υπεύθυνος για το συγχρονισμό εισόδου και εξόδου.

Ως προς τους κύριους στόχους του OpenMP, αναφέρουμε πρώτα την τυποποίηση, η οποία παρέχει ένα πρότυπο ανάμεσα σε μια ποικιλία της shared memory architectures/platforms και καθορίστηκε από κοινού και εγκρίθηκε από μια μεγάλη ομάδα ηλεκτρονικών υπολογιστών και λογισμικού. Ένας δεύτερος στόχος του OpenMP είναι η ανάγκη να είναι λιτή και απλή. Καθιέρωσε λοιπόν ένα απλό και περιορισμένο σύνολο οδηγιών (εντολών) για τον προγραμματισμό shared memory machines. Σημαντικός παραλληλισμός μπορεί να υλοποιηθεί με τη χρήση μόνο 3 ή 4 οδηγιών. Τρίτος στόχος που τέθηκε είναι η ευκολία στη χρήση, που σημαίνει την παροχής της δυνατότητας να παραλληλιστεί σταδιακά ένα σειριακό πρόγραμμα, σε αντίθεση με βιβλιοθήκες ανταλλαγή μηνυμάτων που συνήθως απαιτούν μια προσέγγιση, ή όλα ή τίποτα. Επίσης, παρέχει τη δυνατότητα της εφαρμογής και fine-grain παραλληλισμό. Τελευταίος στόχος που τέθηκε αφορά στη φορητότητα, δηλαδή το API έχει καθοριστεί για την C / C ++ και Fortran, δημόσιο φόρουμ για το API και την ένταξη και οι περισσότερες από τις μεγαλύτερες πλατφόρμες έχουν υλοποιηθεί με openMP, όπως για παράδειγμα Unix / Linux και Windows.

Κεφάλαιο 4

Αλγόριθμοι

4.1 Αλγόριθμος Reduction

Αυτός ο αλγόριθμος δίνει το εσωτερικό άθροισμα δυο πινάκων είναι το άθροισμα όλων των στοιχείων των πινάκων αυτών. Για παράδειγμα A = {2, 5, 7} και B = {1, 3, 6}, αναλυτικά το εσωτερικό άθροισμα αυτών των πινάκων είναι (2 + 1) + (5 + 3) + (7 + 6) = 3 + 8 + 13 = 24.

Σημείωση: ικανή και αναγκαία προϋπόθεση είναι το μήκος των δυο πινάκων να είναι το ίδιο.

Για να μπορεί να παραλληλιστεί το εσωτερικό άθροισμα των δύο πινάκων από τα στοιχεία και c cores είναι ανάγκη:

1. Να αναθέτουμε n/c στοιχεία από κάθε πίνακα σε κάθε πυρήνα.
2. Κάθε πυρήνας στη συνέχεια θα υπολογίσει στην τοπική του μνήμη το άθροισμα των εσωτερικών αθροισμάτων χρησιμοποιώντας τα στοιχεία του n/c που έχουν ανατεθεί στο εν λόγω core και θα το στέλνει πίσω στο host.
3. Ο host με την δική του σειρά θα αθροίσει όλες τις τοπικές μνήμες μαζί για να δώσουν την τελική απάντηση / άθροισμα.

Ένα παράδειγμα ίσως μας βοηθήσει στην καλύτερη κατανόηση.

Ας εξετάσουμε τους πίνακες A = {1, 2, 3, 4} και B = {5, 6, 7, 8}. Επιθυμούμε να υπολογιστεί το άθροισμα των δύο πινάκων σε τέσσερα core (c = 4), επίσης κάθε πίνακας έχει τέσσερα στοιχεία (n = 4).

Για να παραλληλιστεί αυτό το πρόγραμμα στο Epiphany προαπαιτούνται τα εξής στο Host:

1. Ο Host προετοιμάζει την συσκευή και δημιουργεί ομάδα εργασίας από τέσσερα core.
2. Ο Host αναθέτει n/c = 4/4 = 1 στοιχείο από κάθε πίνακα σε κάθε core. Έτσι, ο host εκχωρεί στην τοπική μνήμη του πρώτου core τα στοιχεία {1} και {5}. Στο δεύτερο core στην τοπική μνήμη του θα εκχωρηθούν τα στοιχεία {2} και {6}, στο τρίτο core στην τοπική μνήμη του θα εκχωρηθούν τα στοιχεία {3} και {7}, και τέλος στο τέταρτο core στην τοπική μνήμη του θα εκχωρηθούν τα στοιχεία {4} και {8}.
3. Ο Host εκτελεί το πρόγραμμα της συσκευής για κάθε core ξεχωριστά. Όταν το πρόγραμμα τελειώσει την εκτέλεση του, ο host «περνά» μέσα από κάθε core και παίρνει τα τοπικά ποσά (αθροίσματα) και τα προσθέτει μεταξύ τους. Αυτό το τελικό άθροισμα εξάγετε από τον χρήστη.

Η συσκευή από την δική της πλευρά, κάνει τα εξής για κάθε core ξεχωριστά:

1. Σε κάθε core έχουν ανατεθεί n/c στοιχεία από κάθε πίνακα (A και B) και υπολογίζει το άθροισμα αυτών. Έτσι το 1ο core θα υπολογίσει 1 + 5 = 6, το 2ο core θα υπολογίσει 2 + 6 = 8, το 3ο core θα υπολογίσει 3 + 7 = 10 και το 4ο core θα υπολογίσει 4 + 8 = 12.
2. Μόλις το κάθε core τελειώσει με τον υπολογισμό του τμήματος των αρχικών πινάκων που έχουν ανατεθεί, επιστρέφει το τοπικό άθροισμα και τερματίζει την λειτουργία του.

Λεπτομέρειες Κώδικα

Το πρόγραμμα που θα τρέξει στο Epiphany βρίσκετε στο αρχείο e_main.c και ο κώδικας του Host είναι στο αρχείο main.c. Επίσης έχουμε ένα header αρχείο που το έχουμε ονομάσει common.h το οποίο περιέχει την σταθερά N και την σταθερά CORES που χρησιμοποιούνται τόσο στο Host όσο και στο Epiphany.

Το αρχείο e_main.c, περιλαμβάνει το header αρχείο e-lib.h και το common.h. Αφού δηλώσουμε τρεις ακέραιους δείκτες (a, b και c), τους θέτουμε στις τρεις ξεχωριστές τράπεζες μνήμης που βρίσκονται σε κάθε core. Η c σε αντίθεση με την a και b δεν είναι ένας πίνακας, αλλά μια στατική τιμή και την αρχικοποιούμε με μηδέν. Ο αλγόριθμος μοιάζει με σειριακό αλγόριθμο όπου κρατάμε το αποτέλεσμα (άθροισμα) μεταξύ των στοιχείων του a και b.

Το αρχείο main.c, βρίσκετε ο κώδικάς του Host. Αφού συμπεριλάβουμε την υποχρεωτική βιβλιοθήκη e-hal.h και το header αρχείο common.h. Στην κύρια λειτουργία τώρα έχουμε δημιουργήσει αντικείμενο τύπου e_platform_t για την πλατφόρμα του Epiphany και το αντικείμενο τύπου e_epiphany_t για την ομάδα εργασίας του Epiphany. Έπειτα, δηλώνουμε δύο στατικούς πίνακες μεγέθους N x N και ένα στατικό πίνακα μεγέθους CORES που θα περιέχει τοπικά αθροίσματα του κάθε core. Επίσης, έχουμε μια μεταβλητή all_done θα μας βοηθήσει για να καταλάβουμε ότι έχουν τελειώσει την εργασία τους όλα τα core, μια μεταβλητή sum που θα κράτα το τελικό – συνολικό άθροισμα.

Χρησιμοποιούμε τις συναρτήσεις e_init(), e_reset_system() και e_get_platform_info() της βιβλιοθήκης e-hal.h για να προετοιμάσει, να γίνει μια πλήρη επαναφορά του υλικού Epiphany και να πάρει πληροφορίες για το chip του Epiphany. Στην συνέχεια μέσω της συνάρτησης e_open() της βιβλιοθήκης e-hal.h πάλη, δημιουργούμε μια ομάδα εργασίας από ένα συγκεκριμένο μέγεθος (το μέγιστο είναι 4 x 4 (= R x C) = 16 core). Έπειτα συμπληρώνουμε τους πίνακας a και b με ακεραίους τυχαίους αριθμούς, αφού κάνουμε ee_reset_core() το κάθε core, που είναι στη θέση (i,j). Χρησιμοποιώντας την συνάρτηση e_load_group() της βιβλιοθήκης e-hal.h

«φορτώνουμε» το πρόγραμμα πάνω στην ομάδα εργασίας (dev), σε όλα τα core που θέλουμε να τρέξουν.

Έχουμε μια επανάληψη απεριόριστου μεγέθους όπου το πρόγραμμα εκτελεί συνεχώς «κύκλους» μέσα σε κάθε ένα core (ενεργοποιημένο) της συσκευής. Για κάθε core, διαβάζει τη τιμή που είναι αποθηκευμένη στη θέση μνήμης που ορίσαμε εμείς και την αποθηκεύουμε στη θέση i * C + j του πίνακα done. Αυτή η τιμή θα είναι ή ένα ή μηδέν και την προσθέτουμε στην μεταβλητή all_done. Όταν το all_done θα γίνει ίσο με τη ποσότητα όλων των ενεργοποιημένων core τότε θα τερματιστεί ο βρόγχος.

Τώρα που ξέρουμε ότι σε κάθε core έχει ένα τοπικό ποσό (άθροισμα), περνούμε από κάθε core και παίρνουμε το άθροισμα του, τοποθετώντας το στο πίνακα c στη θέση i * C + j. Έτσι τοποθετούνται όλα τα αθροίσματα των κάθε core στον πίνακα c. Έπειτα, αθροίζουμε όλες τις θέσεις του πίνακα c με την χρήση μια μεταβλητής και έχουμε στην εν λόγω μεταβλητή το συνολικό/τελικό άθροισμα.

Τέλος, όπως όλα τα προγράμματα που κάνουν χρήση του Epiphany έτσι και τώρα καλούμε της συναρτήσεις e_close() και e_finalize() δεδομένου ότι δεν χρειαζόμαστε πια το chip Epiphany.

<u>Σειριακή Υλοποίηση στο Arm (Host).</u>

Στο αρχείο που βρίσκεται η main() που τρέχει σειριακά στο arm υπάρχει μία συνάρτηση όπου είναι υλοποιημένος ο αλγόριθμος σειριακά. Αφού υπολογιστεί το παράλληλο άθροισμα. τότε καλούμε τη συνάρτηση αυτή για να τρέξει ο αλγόριθμος σειριακά και να μας επιστρέψει το αποτέλεσμα (άθροισμα).

<u>Υπολογισμός Χρόνου Εκτέλεσης.</u>

Με τη χρήση ενός timer από τη βιβλιοθήκη sys/time.h παίρνουμε το τοπικό χρόνο πριν τρέξει ο αλγόριθμος σειριακά στο Arm ή παράλληλα στο Epiphany και μετά αφού τελειώσει ο αλγόριθμος, αφαιρούμε το τελικό πλην τον αρχικό χρόνο και υπολογίζουμε πόσα milliseconds χρειάζεται για να εκτελεστεί.

4.2 Αλγόριθμος Matrix Multiplication.

Ο πολλαπλασιασμός πινάκων C = A x B ορίζεται όταν οι αριθμοί των στηλών του πίνακα A είναι ίσοι με τον αριθμό των γραμμών του πίνακα B και ο νέος πίνακας C που προκύπτει μετά το πολλαπλασιασμό A x B έχει διαστάσεις ίσες με τον αριθμό των γραμμών του πίνακα A, επί τον αριθμό των στηλών του πίνακα B.

Για παράδειγμα |A| = n x m και |B| = h x p, τότε ορίζεται C = A x B αν και μόνο αν m = h, τέτοιο ώστε |C| = n x p. Διαφορετικά, αν m ≠ h, δεν ορίζεται ο πολλαπλασιασμός A x B.

Σε αυτό το παράδειγμα θα δούμε πώς οι πολλαπλοί κόμβους πλέγματος Epiphany μπορούν να συνδυαστούν για να βελτιώσου τη συνολική απόδοση ενός υπολογισμού. Matrix Multiplication μπορεί να αντιπροσωπεύεται από τον ακόλουθο τύπο:

$$C[i,j] = \sum_{k=0}^{N-1} (\mathrm{A[i,k]\ x\ B[k,j]})$$

Όπου A και B είναι οι πίνακες εισόδου, ο C είναι το αποτέλεσμα, i και j αντιπροσωπεύουν τη σειρά και στήλη αντίστοιχα των συντεταγμένων στοιχείων των πινάκων.

Μια αφελής, αλλά σωστή υλοποίηση του αλγόριθμου πολλαπλασιασμού πινάκων που τρέχει σε έναν ενιαίο πυρήνα δίνεται στην παρακάτω εικόνα.

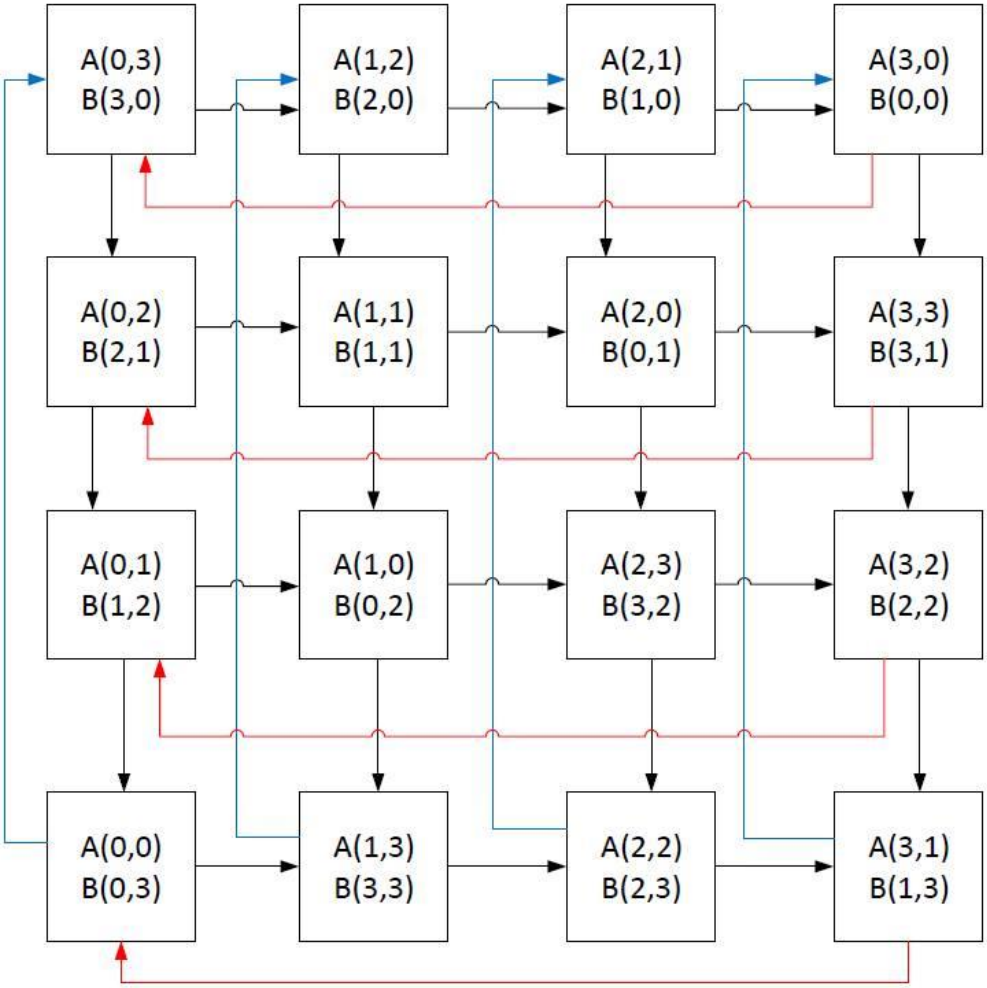
```
for (i=0; i<N; i++)
    for (j=0; j<M; j++) {
        C[i][j] = 0;
        for (k=0; k<K; k++)
            C[i][j] += A[i][k] * B[k][j];
    }
```

Εικόνα 9: Σειριακός κώδικας εκτελέσεις του αλγορίθμου.

Ο κώδικας παραπάνω μπορεί να γραφτεί στο πρότυπο C / C ++ και για να τρέξει σε ένα ενιαίο πυρήνα, με πίνακες A, B, και C. Σε αυτό το απλό παράδειγμα προγραμματισμού, δεν υπάρχει καμία διαφορά μεταξύ της αρχιτεκτονικής του Epiphany και για κάθε άλλο σπείρωμα πλατφόρμα επεξεργαστή.

Για να επιταχυνθεί αυτός ο υπολογισμός χρησιμοποιώντας διάφορους κόμβους του πλέγματος ταυτόχρονα, πρέπει πρώτα να διανέμουμε τις μήτρες A, B και C πάνω P εργασίες. Λόγω της φύσης μήτρα της αρχιτεκτονικής, ο φυσικός τρόπος για τη διανομή μεγάλων πινάκων είναι με την κοπή τους σε μικρότερα τεμάχια (υπό πίνακες). Στη συνέχεια κατασκευάζουμε ένα πρόγραμμα SPMD που τρέχει σε κάθε ένα από τους κόμβους πλέγματος/core.

Στην επόμενη εικόνα δείχνει πως ο Matrix Multiplication μπορεί να χωριστεί σε 16 επιμέρους εργασίες και χαρτογραφούνται επί 16 κόμβους mesh. Ανταλλαγή δεδομένων μεταξύ των επιμέρους καθήκοντα που μπορεί να γίνει με το πέρασμα των δεδομένων μεταξύ των πυρήνων, χρησιμοποιώντας ένα μήνυμα που περνά API που παρέχεται στο Epiphany SDK ή γράφοντας ρητά στην κοινή μνήμη.



Εικόνα 10: Matrix Multiplication Data Flow.

Σε κάθε βήμα της διαδικασίας, συνεισφοράς στο τοπικό πίνακα C συσσωρεύεται σε κάθε εργασία, μετά την οποία ο τοπικός πίνακας A κινείται προς τα κάτω και ο τοπικός πίνακας B κινείται προς τα δεξιά. Το συνολικό παράδειγμα μπορεί να ολοκληρωθεί με τη χρήση τυποποιημένων δομών προγραμματισμού ANSI. Οι λειτουργίες του χρόνου εκτέλεσης (του Epiphany) που προβλέπονται για την απλοποίηση πολλαπλών πυρήνων προγραμματισμού, όμως η χρήση τους δεν είναι υποχρεωτική. Η αρχιτεκτονική επιτρέπει στους προγραμματιστές να καινοτομούν σε όλα τα επίπεδα. Ο αλγόριθμος Matrix Multiplication σε αυτό το παράδειγμα κλιμακώνεται σε πολλούς πυρήνες και δείχνει πως οι κλίμακες απόδοσης της αρχιτεκτονικής Epiphany αυξάνεται γραμμικά με τον αριθμό των πυρήνων στο

σύστημα όταν χρησιμοποιούνται κατάλληλα μοντέλα κατανομή των δεδομένων και δομημένος προγραμματισμός.

Λεπτομέρειες Κώδικα

Αφού γίνει πλήρης αρχικοποίησης των μεταβλητών από τη βιβλιοθήκη e-hal.h της πλακέτας γίνετε σύνδεση μέσων κάποιων ειδικών συναρτήσεων από την εν λόγω βιβλιοθήκη για επικοινωνία με το Epiphany από το Arm και έτσι η συσκευή προετοιμάζεται. Έπειτα, γίνεται αρχικοποίηση του Epiphany για να γίνει η κατάσταση «έτοιμη». Στη συνέχεια αρχικοποιούνται οι πίνακές A και B μέσων της συνάρτησης matrix_init(). Μέσω της συνάρτησης matmul_go() τρέχει ο αλγόριθμος Matrix Multiplication στο Epiphany στα 2, 4, 8 και 16 core, ανάλογα με τα τις σταθερές του κώδικα, ενώ μέσω της συνάντησης matmul() τρέχει σειριακά στο Arm. Μέσω της συνάρτησης gettimeofday() της βιβλιοθήκης sys/time.h. παίρνουμε το χρόνο εκτέλεσης πριν και μετά την εκτέλεση της κάθε συνάρτησης και ουσιαστικά υπολογίζουμε πόσο χρόνο χρειάζεται να εκτελεστεί ο αλγόριθμος για τα 2, 4, 8, 16 core και πόσο χρόνο χρειάζεται για να εκτελεστεί σειριακά δια διάφορες διαστάσεις πινάκων.

Σειριακή Υλοποίηση στο Arm (Host).

Στο αρχείο που βρίσκεται η main() που τρέχει στο arm υπάρχει μια συνάρτηση όπου είναι υλοποιημένος ο αλγόριθμος σειριακά. Αφού υπολογιστεί το παράλληλο άθροισμα καλούμε τη συνάρτηση αυτή για να τρέξει ο αλγόριθμος σειριακά και να μας επιστρέψει το αποτέλεσμα (άθροισμα).

Υπολογισμός Χρόνου Εκτέλεσης.

Με τη χρήση ενός timer από τη βιβλιοθήκη sys/time.h παίρνουμε το τοπικό χρόνο πριν τρέξει ο αλγόριθμος σειριακά στο Arm ή παράλληλα στο Epiphany και μετά αφού τελειώσει ο αλγόριθμος, αφαιρούμε το τελικό πλην τον αρχικό χρόνο και υπολογίζουμε πόσα milliseconds χρειάζεται για να εκτελεστεί.

4.3 Αλγόριθμος Blacksholes.

Αυτός ο αλγόριθμος υπολογίζει την αξία των δικαιωμάτων προαίρεσης με βάση την εξίσωση Black & Scholes. Μοιάζει αρκετά με τον αλγόριθμό Reduction.

Για να μπορεί να παραλληλιστεί αυτός ο αλγόριθμος από τα στοιχεία (6 πίνακες των n στοιχείων) και c cores είναι ανάγκη:

1. Να αναθέτουμε n/c στοιχεία από κάθε πίνακα σε κάθε πυρήνα.
2. Κάθε πυρήνας στη συνέχεια θα υπολογίσει στην τοπική του μνήμη η τιμή τις εξίσωσης Black & Scholes χρησιμοποιώντας τα στοιχεία του n/c που έχουν ανατεθεί στο εν λόγω core και θα το στέλνει πίσω στο host το τελικό αποτέλεσμα.
3. Ο host με την δική του σειρά θα αθροίσει όλες τις τοπικές μνήμες μαζί για να δώσουν την τελική απάντηση / άθροισμα.

Η συσκευή από την δική της πλευρά, κάνει τα εξής για κάθε core ξεχωριστά:

1. Σε κάθε core έχουν ανατεθεί n/c στοιχεία από κάθε πίνακα και υπολογίζει με βάσει την εξίσωση αυτών.
2. Μόλις το κάθε core τελειώσει με τον υπολογισμό του τμήματος των αρχικών πινάκων που έχουν ανατεθεί, επιστρέφει τις τελικές τιμές και τερματίζει την λειτουργία του.

<u>Λεπτομέρειες Κώδικα</u>

Το πρόγραμμα που θα τρέξει στο Epiphany βρίσκετε στο αρχείο e_main.c και ο κώδικας του Host είναι στο αρχείο main.c. Επίσης έχουμε ένα header αρχείο που το έχουμε ονομάσει common.h το οποίο περιέχει την σταθερά N και την σταθερά CORES που χρησιμοποιούνται τόσο στο Host όσο και στο Epiphany.

Το αρχείο e_main.c, περιλαμβάνει το header αρχείο e-lib.h και το common.h. Αφού δηλώσουμε δείκτες, τους θέτουμε σε ξεχωριστές τράπεζες μνήμης που βρίσκονται σε κάθε core και την αρχικοποιούμε με μηδέν.

Το αρχείο main.c, βρίσκετε ο κώδικάς του Host. Αφού συμπεριλάβουμε την υποχρεωτική βιβλιοθήκη e-hal.h και το header αρχείο common.h. Στην κύρια λειτουργία τώρα έχουμε δημιουργήσει αντικείμενο τύπου e_platform_t για την πλατφόρμα του Epiphany και το αντικείμενο τύπου e_epiphany_t για την ομάδα εργασίας του Epiphany. Έπειτα, δηλώνουμε δύο στατικούς πίνακες μεγέθους N και ένα στατικό πίνακα μεγέθους CORES που θα περιέχει τοπικά αθροίσματα του κάθε core. Επίσης, έχουμε μια μεταβλητή all_done θα μας βοηθήσει για να καταλάβουμε ότι έχουν τελειώσει την εργασία τους όλα τα core, ένα πίνακα μεγέθους N που θα κρατά τις τελικές τιμές.

Χρησιμοποιούμε τις συναρτήσεις e_init(), e_reset_system() και e_get_platform_info() της βιβλιοθήκης e-hal.h για να προετοιμάσει, να γίνει μια πλήρη επαναφορά του υλικού Epiphany και να πάρει πληροφορίες για το chip του Epiphany. Στην συνέχεια μέσων της συνάρτησης e_open() της βιβλιοθήκης e-hal.h πάλη, δημιουργούμε μια ομάδα εργασίας από ένα συγκεκριμένο μέγεθος (το μέγιστο είναι 4 x 4 (= R x C) = 16 core). Έπειτα συμπληρώνουμε τους πίνακας με δεδομένα από αρχείο εισόδου, αφού κάνουμε ee_reset_core() το κάθε core, που είναι στη θέση (i,j). Χρησιμοποιώντας την συνάρτηση e_load_group() της βιβλιοθήκης e-hal.h «φορτώνουμε» το πρόγραμμα πάνω στην ομάδα εργασίας (dev), σε όλα τα core που θέλουμε να τρέξουν.

Έχουμε μια επανάληψη απεριόριστου μεγέθους όπου το πρόγραμμα εκτελεί συνεχώς «κύκλους» μέσα σε κάθε ένα core (ενεργοποιημένο) της συσκευής. Για κάθε core, διαβάζει τη τιμή που είναι αποθηκευμένη στη θέση μνήμης που ορίσαμε εμείς και την αποθηκεύουμε στο πίνακα done. Αυτή η τιμή θα είναι ή ένα ή μηδέν και την προσθέτουμε στην μεταβλητή all_done. Όταν το all_done θα γίνει ίσο με τη ποσότητα όλων των ενεργοποιημένων core τότε θα τερματιστεί ο βρόγχος.

Τώρα που ξέρουμε ότι σε κάθε core υπάρχουν οι τελικές τιμές, περνούμε από κάθε core και παίρνουμε τις τιμές αυτές, τοποθετώντας το στο πίνακα princes. Έτσι τοποθετούνται όλες οι τιμές στον πίνακα αυτό.

Τέλος, όπως όλα τα προγράμματα που κάνουν χρήση του Epiphany έτσι και τώρα καλούμε της συναρτήσεις e_close() και e_finalize() δεδομένου ότι δεν χρειαζόμαστε πια το chip Epiphany.

Σειριακή Υλοποίηση στο Arm (Host).

Στο αρχείο που βρίσκεται η main() που τρέχει σειριακά στο arm υπάρχει μια συνάρτηση όπου είναι υλοποιημένος ο αλγόριθμος σειριακά. Αφού υπολογιστούν η παράλληλες τιμές, τότε καλούμε τη συνάρτηση αυτή για να τρέξει ο αλγόριθμος σειριακά και να μας επιστρέψει τις τελικές τιμές με βάση την εξίσωση του Blacksholes.

Υπολογισμός Χρόνου Εκτέλεσης.

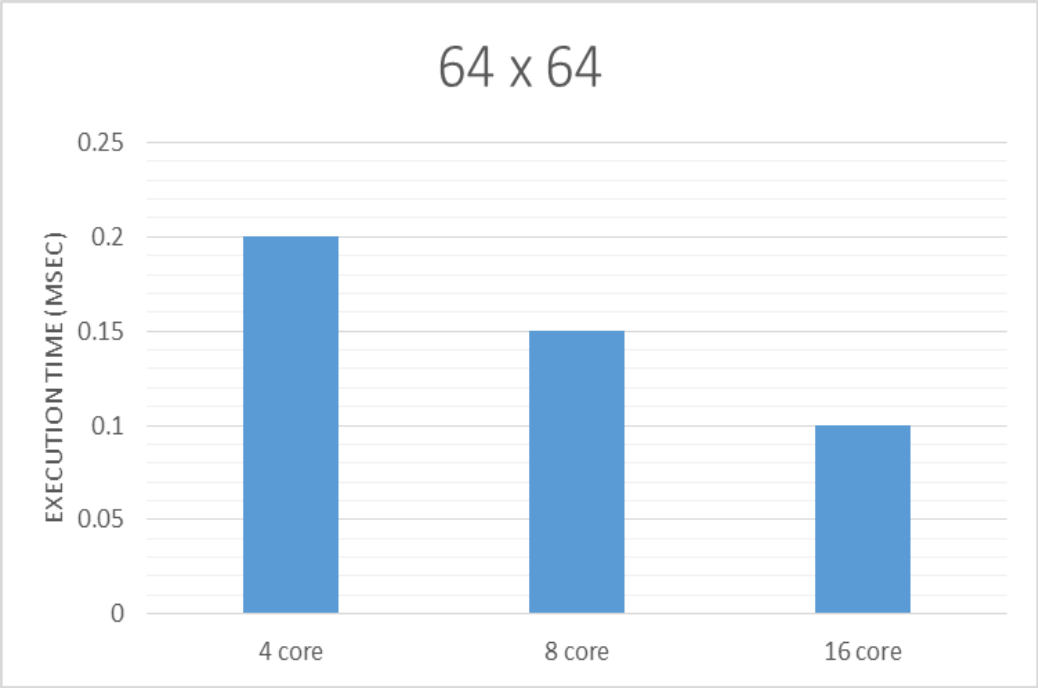
Με τη χρήση ενός timer από τη βιβλιοθήκη sys/time.h παίρνουμε το τοπικό χρόνο πριν τρέξει ο αλγόριθμος σειριακά στο Arm ή παράλληλα στο Epiphany και μετά αφού τελειώσει ο αλγόριθμος, αφαιρούμε το τελικό πλην τον αρχικό χρόνο και υπολογίζουμε πόσα milliseconds χρειάζεται για να εκτελεστεί.

Κεφάλαιο 5

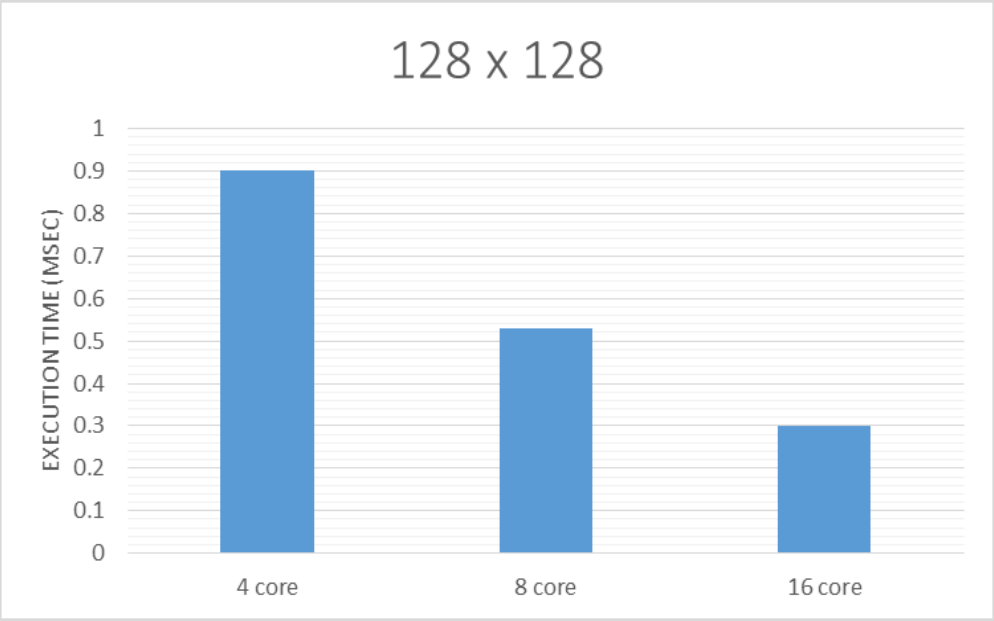
Αποτελέσματα

5.1 Αλγόριθμος Reduction.

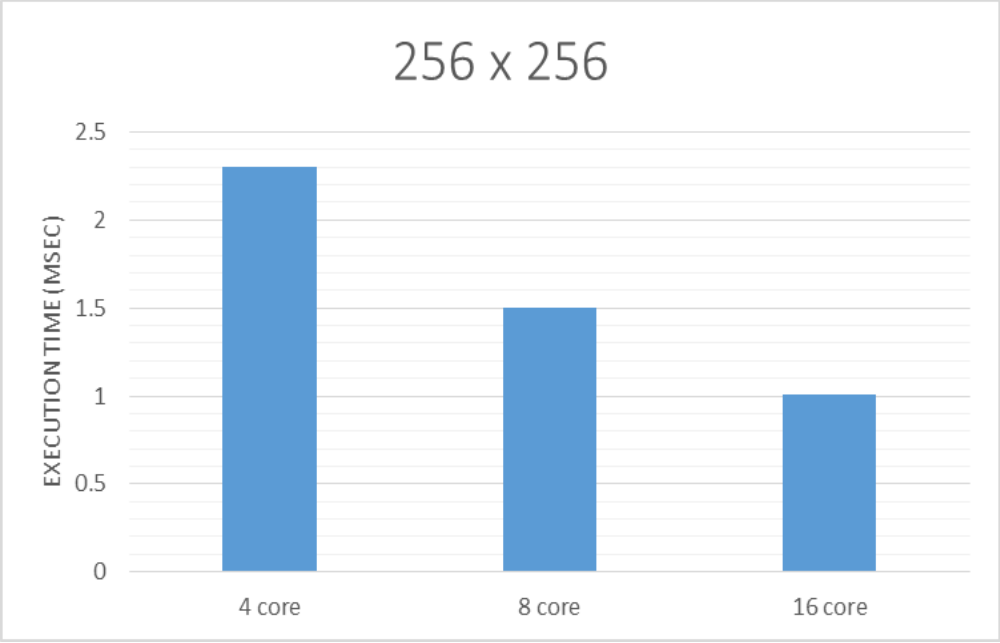
5.1.1 Για εισόδους δεδομένων 64 x 64



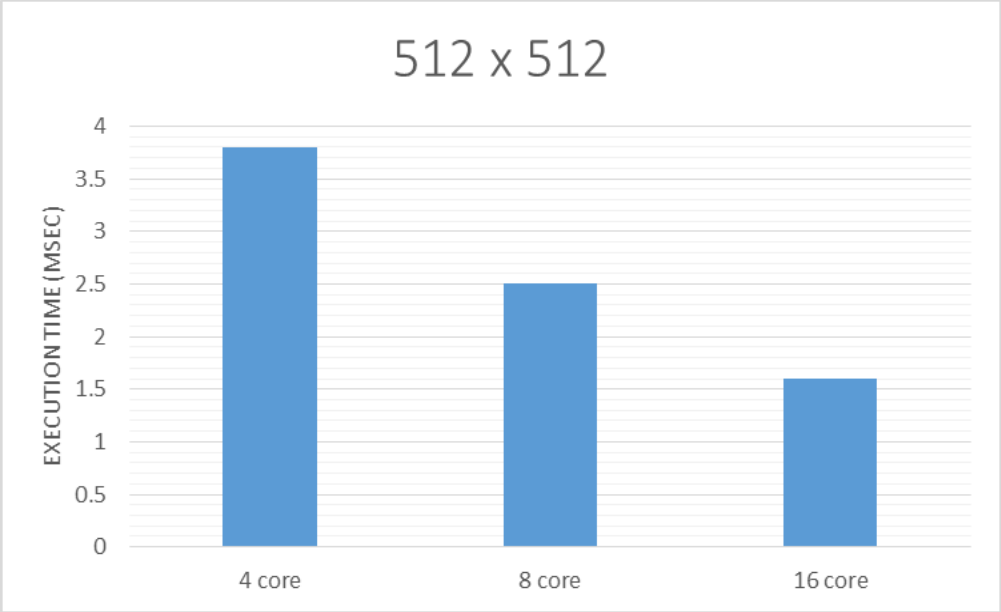
5.1.2 Για εισόδους δεδομένων 128 x 128



5.1.3 Για εισόδους δεδομένων 256 x 256



5.1.4 Για εισόδους δεδομένων 512 x 512

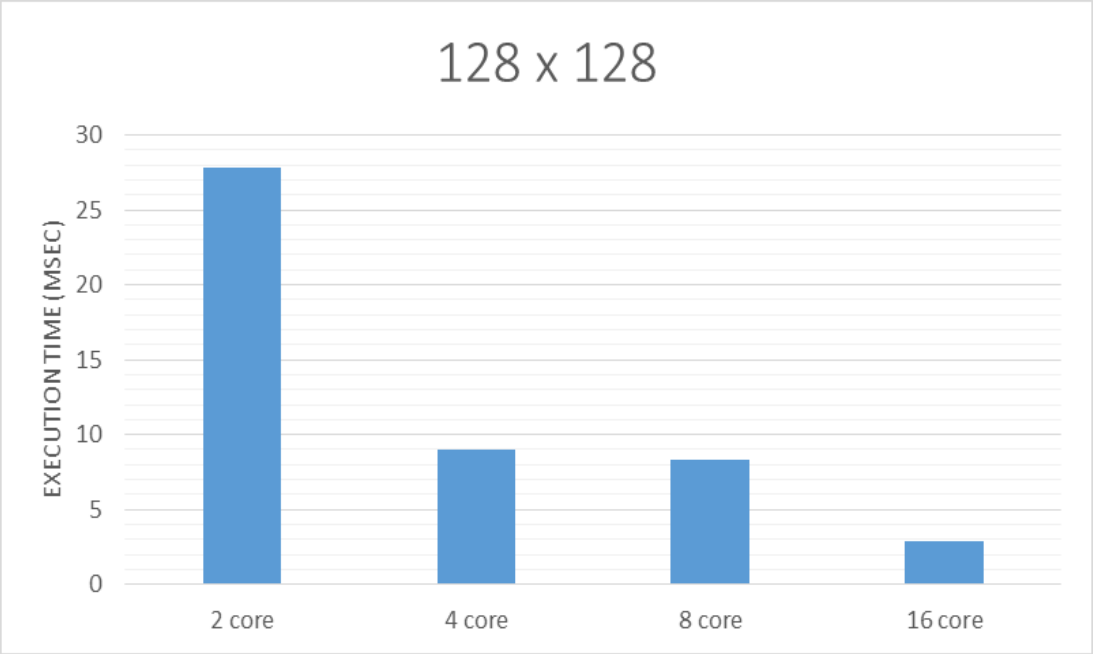


<u>Σχόλια Αποτελεσμάτων 5.1.1-5.1.4:</u>

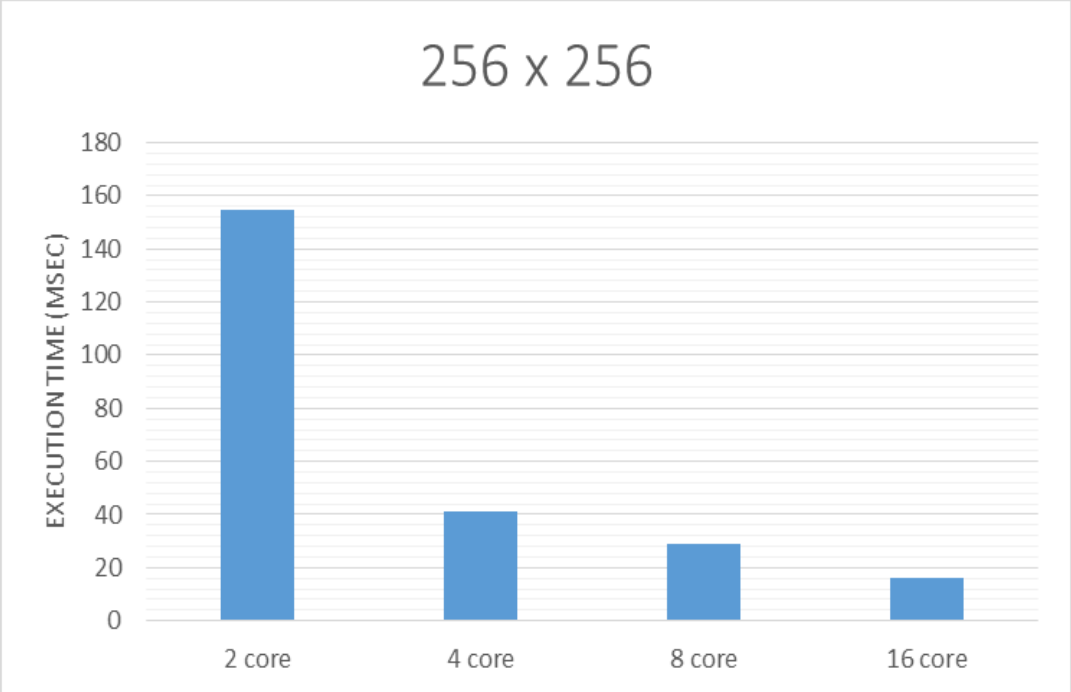
Παρατηρούμε ότι όσο αυξάνονται τα core μειώνεται ο χρόνος εκτέλεσης του αλγορίθμου. Επίσης, παρατηρούμε ότι όσο αυξάνονται να δεδομένα εισόδου αυξάνεται και ο χρόνος εκτέλεσης εκθετικά.

5.2 Αλγόριθμος Matrix Multiplication

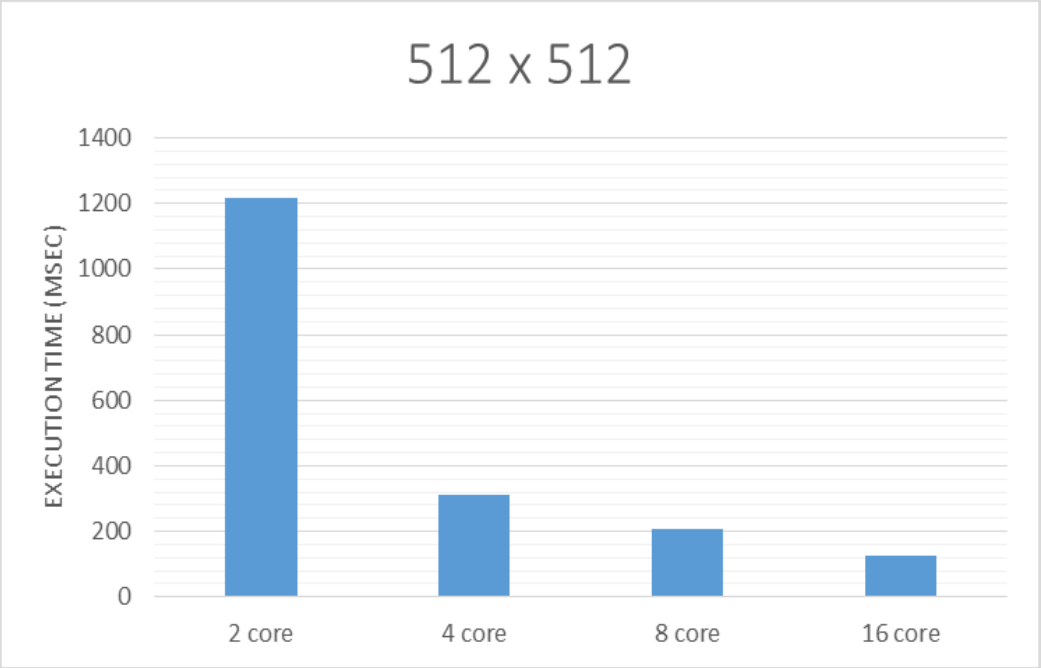
5.2.1 Για εισόδους δεδομένων 128 x 128



5.2.2 Για εισόδους δεδομένων 256 x 256



5.2.3 Για εισόδους δεδομένων 512 x 512

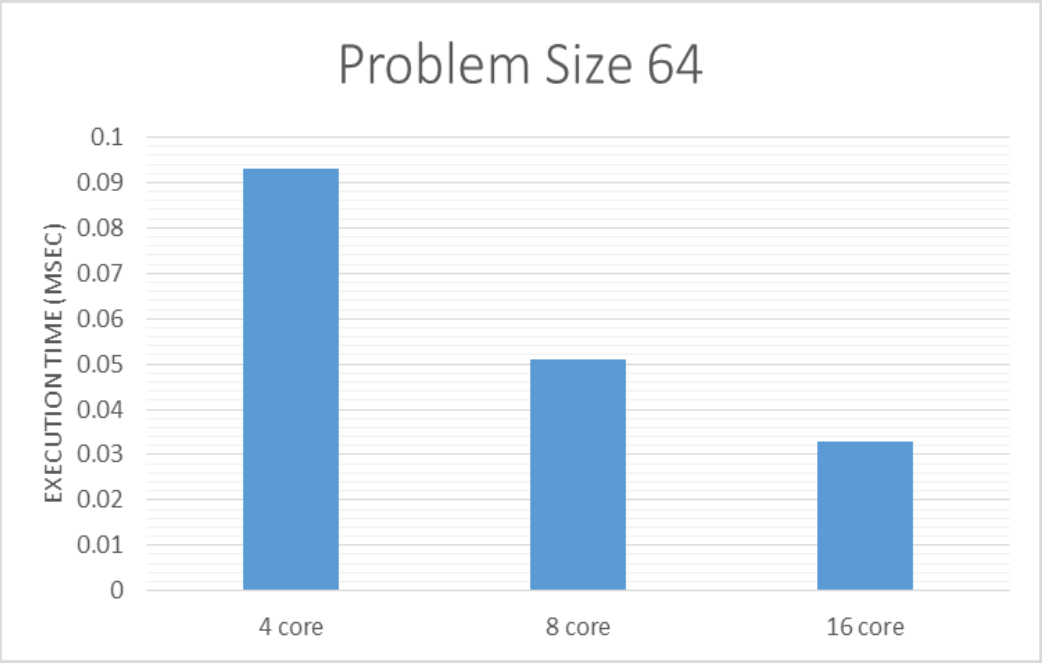


<u>Σχόλια Αποτελεσμάτων 5.2.1 - 5.2.3:</u>

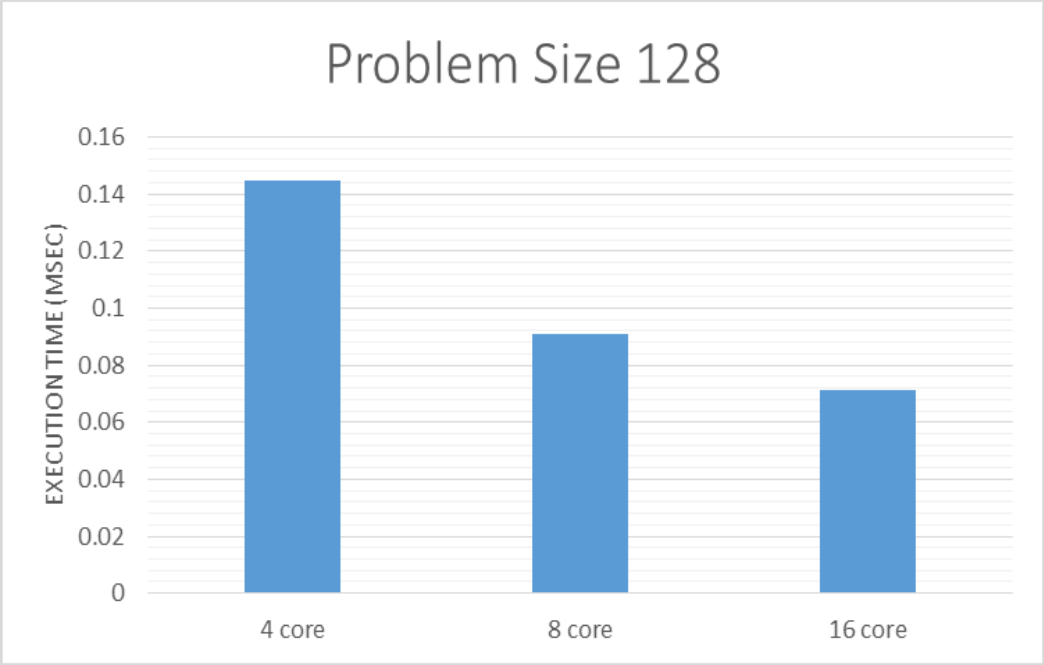
Παρατηρούμε ότι όσο αυξάνονται τα core μειώνετε ο χρόνος εκτέλεσης του αλγορίθμου. Επίσης, παρατηρούμε ότι όσο αυξάνονται να δεδομένα εισόδου αυξάνεται και ο χρόνος εκτέλεσης εκθετικά.

5.3 Αλγόριθμος Blacksholes

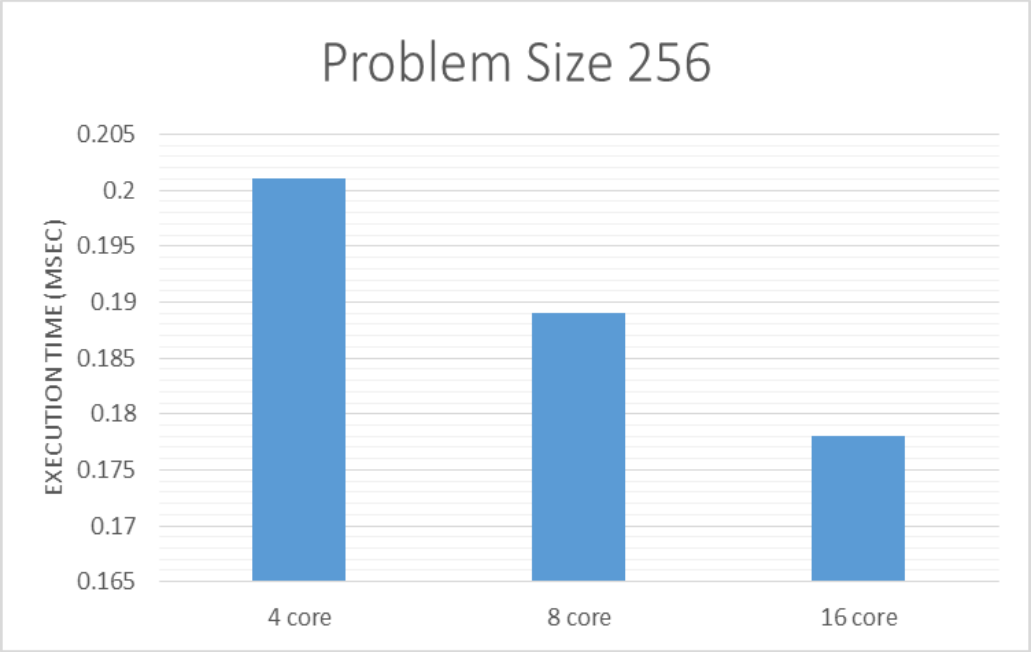
5.3.1 Για εισόδους δεδομένων 64



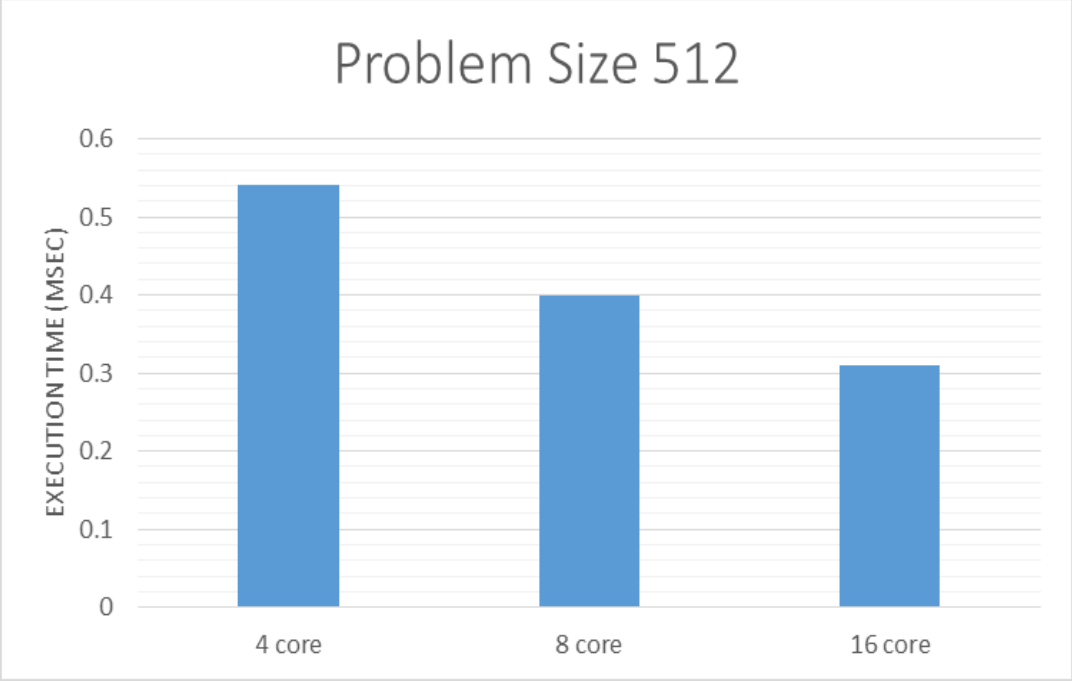
5.3.2 Για εισόδους δεδομένων 128



5.3.3 Για εισόδους δεδομένων 256



5.3.4 Για εισόδους δεδομένων 512

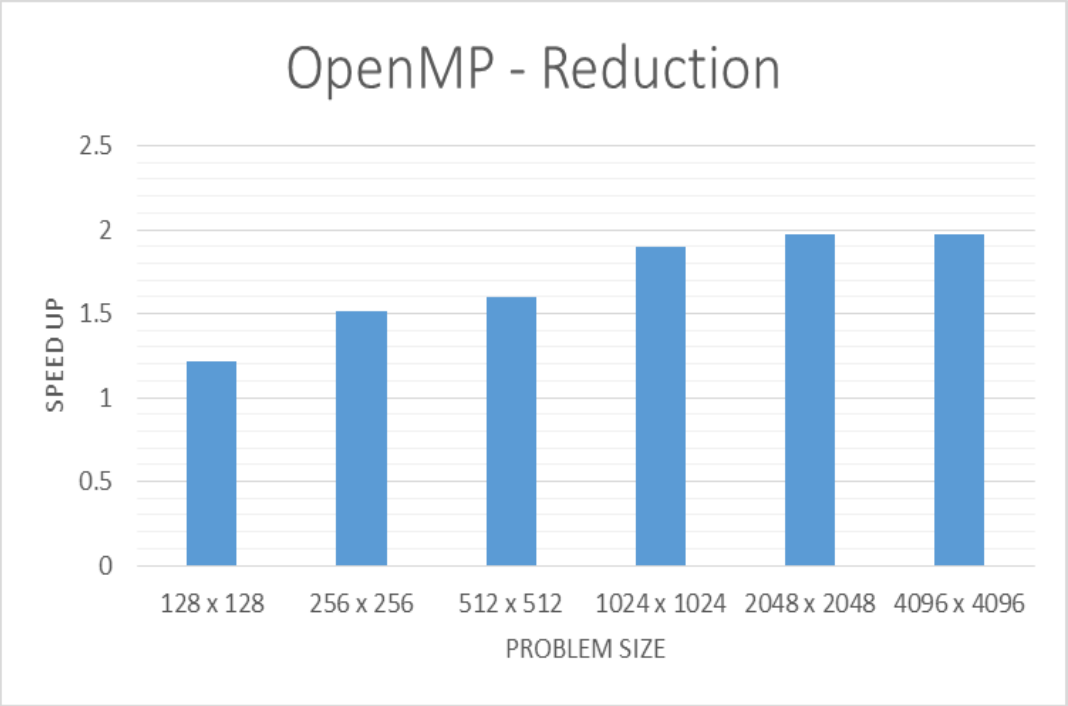


<u>Σχόλια Αποτελεσμάτων 5.3.1 - 5.3.4:</u>

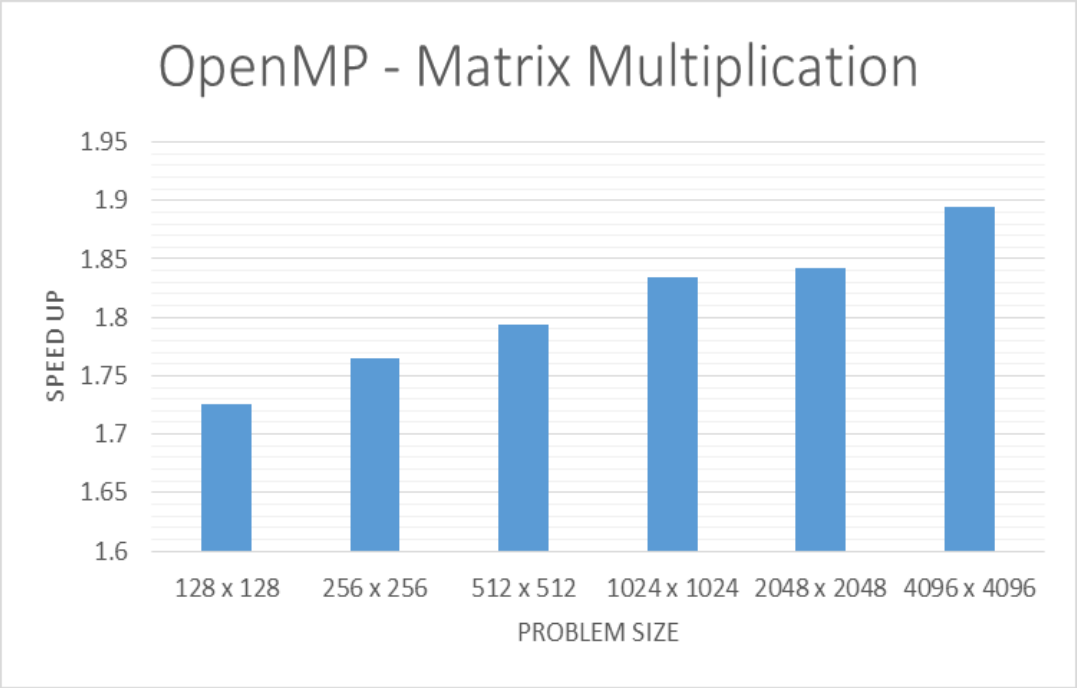
Παρατηρούμε ότι όσο αυξάνονται τα core μειώνετε ο χρόνος εκτέλεσης του αλγορίθμου. Επίσης, παρατηρούμε ότι όσο αυξάνονται να δεδομένα εισόδου αυξάνεται και ο χρόνος εκτέλεσης εκθετικά.

5.4 OpenMP

5.4.1 Αποτελέσματα αλγορίθμου Reduction με framework OpenMP



5.4.2 Αποτελέσματα αλγορίθμου Matrix Multiplication με framework OpenMP



Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα

Κατά τη διάρκεια της υλοποίησης των αλγορίθμων κατάληξα στο συμπέρασμα ότι ο προγραμματισμός της Parallella Board είναι αρκετά δύσκολος. Επίσης, όσο αυξάνονται τα core μειώνεται ο χρόνος εκτέλεσης του αλγόριθμου και όταν διπλασιάζουμε τους πυρήνες (από 2 core σε 4 core) ο χρόνος περίπου είναι ο μίσος του προηγούμενου.

6.2 Προβλήματα που αντιμετώπισα

Κατά τη διάρκεια υλοποίησης των αλγορίθμων στο Parallella Board, αντιμετώπισα αρκετά προβλήματα. Αρχικά, η θερμοκρασία της πλακέτας ανέβαινε αρκετά ψηλά και υπήρχε κίνδυνος να καεί. Έτσι παραγγείλαμε μία ψήκτρα και την τοποθετήσαμε στη πλακέτα.

Επίσης, λόγω της περίπλοκης αρχιτεκτονικής της πλακέτας ο προγραμματισμός στο framework της ήταν αρκετά δύσκολος και αρκετά ιδιαίτερος, ειδικά αν κάποιο πρόσωπο δεν έχει παρακολουθήσει το μάθημα της αρχιτεκτονικής υπολογιστών καθώς και το μάθημα της παράλληλης επεξεργασίας.

Ένα άλλο πρόβλημα που είχα αντιμετωπίσει είναι όταν ήταν σε λειτουργία δύο με τριών ωρών το board κολλούσε χωρίς να μπορώ να κάνω κάτι με αποτέλεσμα να αναγκάζομαι να το αποσυνδέω από το ρεύμα και να το ξανασυνδέω.

Τέλος, όταν το έθετα σε λειτουργία μερικές φορές το ποντίκι και το πληκτρολόγιο δεν δουλεύαν ενώ ήταν συνδεδεμένα με το board και μοναδική λύση και να το αποσυνδέω και να το ξανασυνδέω.

6.3 Μελλοντική έρευνα

Ως αρχικό στόχο, θα μπορούσαμε να θέσουμε, να πάρουμε ένα δεύτερο Parallella Board και να υλοποιούσαμε αλγόριθμους με το γνωστό framework MPI. Επίσης, θα μπορούσαμε να υλοποιήσουμε το αλγόριθμο Matrix Multiplication σε block για να το συγκρίνουμε με αυτό που υλοποιήθηκε.

Βιβλιογραφία

[1] http://www.adapteva.com/epiphany-multicore-intellectual-property/

[2] http://www.arm.com/cortex-a9.php

[3] http://www.parallella.org/docs/parallella_manual.pdf

[4] https://www.parallella.org/quick-start/

[5] http://www.parallella.org/install-the-heatsink/

[6] http://adapteva.com/docs/epiphany_arch_ref.pdf

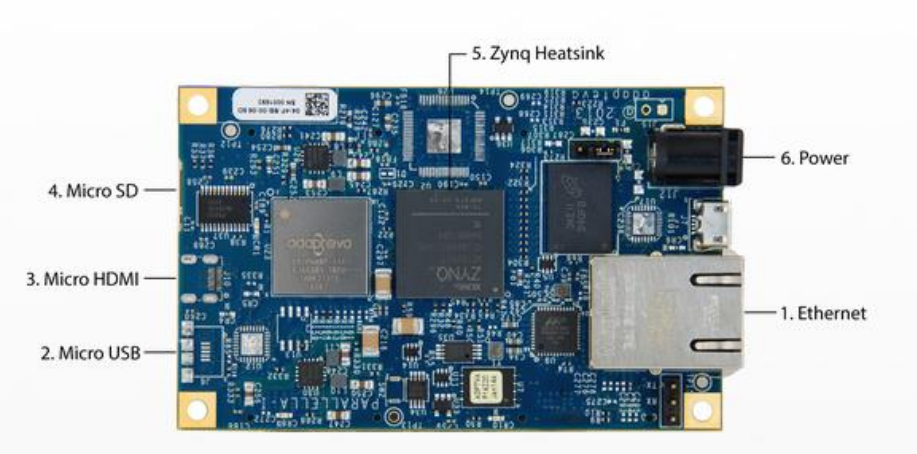
[7] http://adapteva.com/docs/epiphany_arch_ref.pdf

[8] http://www.notebookcheck.net/FAQ-GPU-Features-in-Detail.86478.0.html

[9] http://arch.icte.uowm.gr/courses/parallel/

Παραρτήματα Α

Γρήγορος οδηγός εγκατάστασης



Εικόνα 11: Πάνω όψη πλακέτας. [4]

Βήμα 1: Βεβαιωθείτε ότι έχετε τα απαραίτητα εξαρτήματα

1. Μια υψηλής ποιότητας 2000mA 5V DC ονομαστική παροχή ηλεκτρικού ρεύματος με 5,5 χιλιοστά OD / 2,1 χιλιοστά κέντρο ID θετικό βύσμα πολικότητας.
2. Ένα μικρό HDMI σε HDMI καλώδιο.
3. Ένα male USB Micro-B σε female Standard-A cable.
4. Ένα καλώδιο Ethernet.
5. Ένας ανεμιστήρας (συνιστάται ανεπιφύλακτα!).

Βήμα 2: Δημιουργία ενός bootable κάρτας micro-SD

Οδηγίες για τη δημιουργία μια κάρτας SD.

6 Λήψη Binaries (με οθόνη).

"Desktop" έκδοση: Κατεβάστε το αρχείο ubuntu-14.04-hdmi-z7010-20140611.img.gz από τον σύνδεσμό http://www.parallella.org/create-sdcard/.

7 Ακολουθήστε τις οδηγίες εγκατάστασης.

i. Τοποθετήστε τη κάρτα SD σε κανονικό υπολογιστή (με λογισμικό Windows).

ii. Αποσυμπιέστε την εικόνα του Ubuntu:

Χρησιμοποιήστε 7-Zip ή ένα αντίστοιχο πρόγραμμα για να αποσυμπιέσετε το αρχείο <>.img.gz .

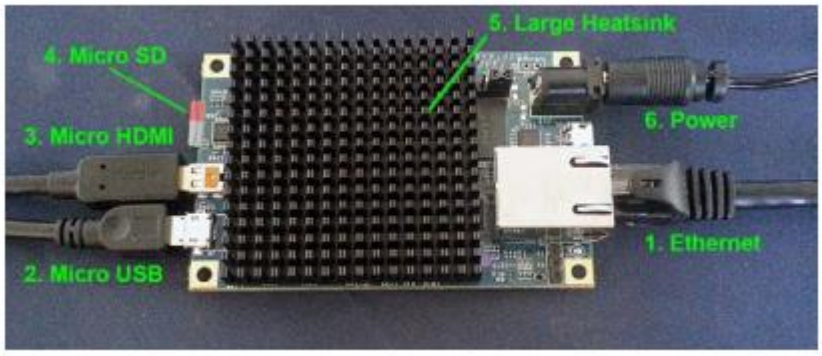
iii. Κατεβάστε από το σύνδεσμο http://sourceforge.net/projects/win32diskimager/ το Win32 disk imager και κάνετε εγκατάσταση μέσω αυτού το αρχείο <>.img που αποσυμπιέσατε στη κάρτα SD.

Βήμα 3: Αποποίηση Ευθυνών

Επειδή το board ζεσταίνεται, είναι απολύτως απαραίτητη ανάγκη να ληφθούν τα κατάλληλα μέτρα ώστε να παγώσει σωστά με σκοπό να πέσει η θερμοκρασία. Πριν χρησιμοποιηθεί το board θα πρέπει για ώρες να τοποθετείται οριζόντια σε μια λεία επιφάνεια. Είναι ανάγκη επίσης να σημειώσουμε ότι είναι ευαίσθητό στο στατικό ηλεκτρισμό. Εάν το board ήρθε πριν από τον Μάρτιο του 2014, τότε θα πρέπει να χρησιμοποιηθεί ένα usb hub. Εάν έχει παραληφθεί το baord πριν τον Ιούλιο 2014, τότε θα πρέπει να χρησιμοποιηθεί ένας μικρός ανεμιστήρα με το baord.

Βήμα 4: Συνδέστε τα καλώδια και τη ψήκτρα

1. Συνδέστε τα καλώδια (από τον αριθμό 1 μέχρι και τον αριθμό 4) όπως φαίνεται στην εικόνα 4 πιο πάνω και στην εικόνα 5 που ακολουθεί.

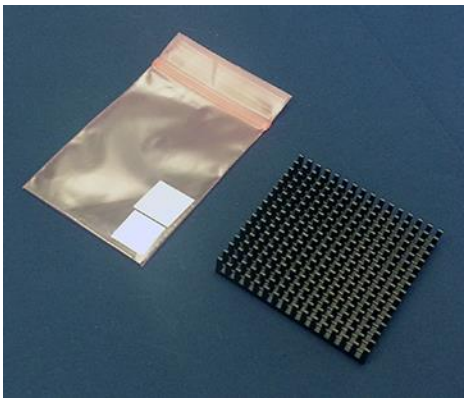


Εικόνα 12: Πάνω όψη πλακέτας με συνδεδεμένα όλα τα καλώδια και η ψήκτρα τοποθετημένη στην θέση της. [4]

2. Σύνδεση της ψήκτρα.

i. Αφού παρατηρήσετε στο κουτί, θα δείτε ότι περιέχει τη ψήκτρα και δύο τεμάχια διπλής όψης θερμική ταινία με εξαιρετικές ιδιότητες θερμική αγωγιμότητα για να τοποθετεί η ψήκτρα στο board. Όπως φαίνονται στην εικόνα 6 παρακάτω.

Σημείωση: Με αυτή τη ψήκτρα το Parallella Board θα λειτουργεί υπό κανονικές συνθήκες χωρίς να απαιτείται ανεμιστήρας. Αν το board είναι σε ένα ζεστό περιβάλλον, ή μέσα σε ένα περίβλημα, θα πρέπει να τοποθετηθεί ένας ανεμιστήρας για να διατηρηθεί μια λογική θερμοκρασία.



Εικόνα 13: Η ψήκτρα και οι ταινίες διπλής όψεως [5]

ii. Καθαρισμός

Χρησιμοποιήστε κάποια αλκοόλη (π.χ. οινόπνευμα) σε ένα πανί χωρίς χνούδι για να καθαρίσετε την κάτω επιφάνεια της ψήκτρας και τα δύο chips πριν από την τοποθέτηση της κολλητική ταινία.

Οι εργασίες πρέπει να γίνονται πάνω σε μια καθαρή επιφάνεια για το λόγο αυτό θα πρέπει να χρησιμοποιηθούν γάντια, τσιμπιδάκια και ένα μαχαίρι για το χειρισμό των κομματιών της ταινίας. Θα πρέπει επίσης να πραγματοποιηθούν και τα παρακάτω βήματα.:

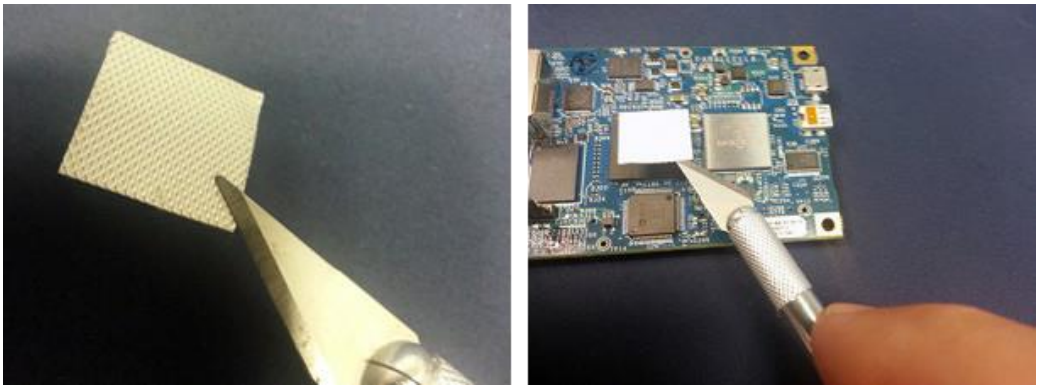
iii. Τοποθετούμε την πρώτη πλευρά της ταινίας για σημάδι

Πρώτα τοποθετούμε τη πλακέτα σε μια καθαρή επιφάνεια, όπως ένα καθαρό κομμάτι από χαρτόνι ή κάτι παρόμοιο για να αποφύγουμε τυχόν ζημιά στην πλακέτα. Ακολούθως, καθαρίζουμε ένα από τα προστατευτικά καλύμματα, δηλαδή τη μία πλευρά της κολλητική ταινία με ένα μαχαίρι (όπως φαίνεται και στην εικόνα 7).

Προσοχή: Χρειάζεται προσοχή ώστε να μην αγγιχθεί η επιφάνεια της κολλητικής ταινία γιατί είναι κολλώδεις και θα θιγόταν επίσης η θερμική αγωγιμότητα.

Γυρίζουμε το πάνω με τσιμπιδάκι και το κολλούμε πάνω στο τσιπ (τοποθετούμε τη ταινία προσεκτικά στο κέντρο του chip χωρίς επικάλυψη). Με το προστατευτικό κάλυμμα ακόμα στη θέση του (δεν αγγίζουμε τη κολλητική επιφάνεια),

χρησιμοποιούμε το δάχτυλό μας για να πιέσουμε σταθερά πάνω στη ταινία για τριάντα δευτερόλεπτα. Μόλις τοποθετηθεί η ταινία, δεν συνίσταται η μετακίνηση της διότι αυτό θα επηρεάσει τη δυνατότητα της συγκόλλησης της.

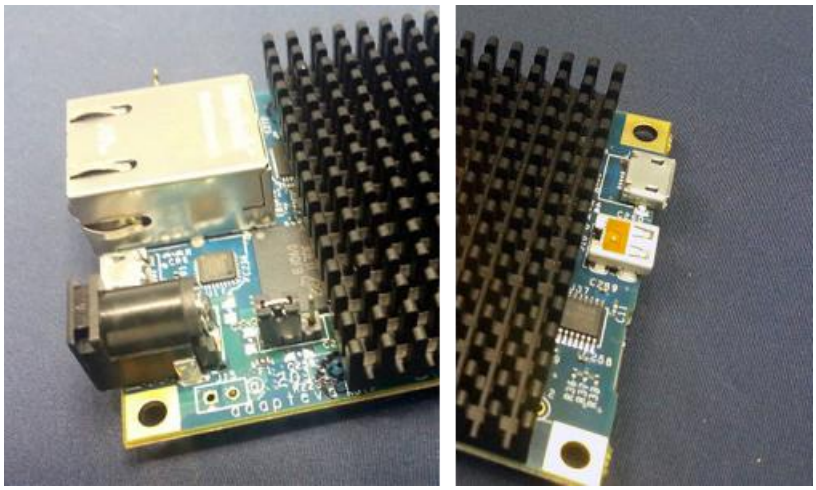


Εικόνα 14: Η ταινία και ο χειρισμός της με τσιμπιδάκι/μαχαίρι [5]

iv. Αφαιρούμε το δεύτερο προστατευτικό κάλυμμα και τοποθετούμε τη ψήκτρα.

Πρακτική τοποθέτηση της ψήκτρας στη θέση της έτσι ώστε οι άκρες να είναι ευθυγραμμισμένες με τις μεγάλες πλευρές του board και τις εσωτερικές ακμές. Δεν αγγίζουμε τις υποδοχές σύνδεσης και στα δύο άκρα (HDMI και Jumper J11). Προσεκτικά αφαιρούμε το δεύτερο προστατευτικό κάλυμμα της ταινίας με ένα μαχαίρι. Τοποθετούμε τη ψήκτρα πάνω στη περιοχή της προσεκτικά και πιέζουμε προς τα κάτω πάνω στη θέση της. Κρατούμε τη ψήκτρα στη θέση με ελαφριά πίεση για 30 δευτερόλεπτα εφαρμόζοντας αρχικά μια πολύ απαλή κίνηση. Εξακριβώνουμε αν έχει στερεωθεί. Δεν μετακινούμε ούτε πιέζουμε αρκετά τη ψήκτρα.

3. Σε περίπτωση υπερθέρμανσης απαιτείται η χρήση ενώ μικρού ανεμιστήρα.
4. Εφαρμογή δύναμης (αριθμός 6, εικόνα 4/εικόνα 5).



Εικόνα 15: Η ψήκτρα τοποθετημένη ευθύγραμμα στο board. [5]

Βήμα 5: Είσοδος

Ενώνουμε στη θήρα microUSB του board (σημείο 2, εικόνα 5) με ένα καλώδιο που έχει την ιδιότητα να μετατραπεί η θήρα microUSB σε υποδοχή USB, στη συνέχει ενώνουμε ένα πληκτρολόγιο και ένα ποντίκι. Έπειτα ενώνουμε το board (σημείο 3, εικόνα 5) με μια τηλεόραση που υποστηρίζει HDMI. Τώρα προσφέρεται η δυνατότητα να κάνουμε σύνδεση με το username linaro και κωδικό linaro, αφού περιμένουμε λίγο μέχρι να εμφανιστεί στην οθόνη η επιλογή της σύνδεσης, εικόνα 9.

Εικόνα 16: Σύνδεση

Εγκατάσταση ψήκτρας (fan)

Λόγω υψηλής θερμότητας του board, έχουμε κάνει εγκατάσταση μιας ψήκτρας (http://groundelectronics.com/products/parallella-open-case-and-cooling-kit)

Εικόνα 17: Η ψήκτρα (fan) τοποθετημένη στο board.