Πανεπιστήμιο Θεσσαλίας

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών
Τηλεπικοινωνιών & Δικτύων

Διπλωματική Εργασία

«Μεταφορά και Βελτιστοποίηση Εφαρμογής Υπολογιστικής Βιολογίας (RAxML) σε Many-Core Σύστημα»

"Porting and Optimization of a Computational Biology Application (RAxML) on a Many-Core System"

Παπαχαρισίου Κωνσταντίνος

Επιβλέποντες Καθηγητές:

Αντωνόπουλος Χρήστος

Επίκουρος Καθηγητής

Μπέλλας Νικόλαος

Αναπληρωτής Καθηγητής

Βόλος, Σεπτέμβριος 2012

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Περίληψη

Η παρούσα εργασία είχε ως αφορμή την εκμετάλλευση της τεράστιας υπολογιστικής ισχύος που μας δίνεται με την αξιοποίηση της νέας γενιάς καρτών γραφικών για την επιτάχυνση της εκτέλεσης εφαρμογών γενικού σκοπού και τη βελτιστοποίηση της απόδοσής τους.

Στόχος της διπλωματικής είναι η περιγραφή της διαδικασίας που ακολουθήσαμε για τη μεταφορά της εκτέλεσης μιας υπολογιστικά απαιτητικής εφαρμογής βιοπληροφορικής (RAxML) σε ένα πολυ-πυρηνικό σύστημα (GPU). Συγκεκριμένα γίνεται περιγραφή των προγραμματιστικών τεχνικών που χρησιμοποιήθηκαν για την μεταφορά της, τη βελτιστοποίησή της καθώς και την επίλυση των προβλημάτων που προέκυψαν.

Αρχικά γίνεται μία αναφορά στην εφαρμογή RAxML επικεντρώνοντας στο παράλληλο κομμάτι, της εκτέλεσης της φυλογενετικής συνάρτησης πιθανοφάνειας. Έπειτα, ανάλυση της τεχνικής προγραμματισμού μαζικά παράλληλων επεξεργαστών και συγκεκριμένα το μοντέλο CUDA και περιγραφή της αρχιτεκτονικής Fermi της κάρτας GTX480 στην οποία θα γίνει η ανάπτυξη του κώδικα.

Στο κύριο μέρος της εργασίας, σε κάθε βήμα περιγράφονται οι στόχοι και τα προβλήματα που εμφανίστηκαν για την επίτευξή τους. Έπειτα, η τεχνική επίλυσης των προβλημάτων καθώς και η επίδραση στο χρόνο εκτέλεσης της εφαρμογής και τέλος οι περιορισμοί που επιφέρει η κάθε λύση. Τέλος γίνεται η αξιολόγηση της απόδοσης με βάση τη μορφή της εφαρμογής.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Ευχαριστίες

Με το πέρας της διπλωματικής θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Αντωνόπουλο Χρήστο για την έμπνευση ενασχόλησης με τα συστήματα υψηλών επιδόσεων, την πρόταση του θέματος καθώς και την καθοδήγησή του κατά την υλοποίηση. Επίσης ένα μεγάλο ευχαριστώ στην οικογένεια και του φίλους μου για την στήριξή τους καθ' όλη τη διάρκεια των σπουδών μου.

Τέλος θα ήθελα να επισημάνω την κύρια συμβολή στην επίλυση των προβλημάτων που προέκυψαν, της κοινότητας των προγραμματιστών λογισμικού που έχει αναπτυχθεί στο διαδίκτυο.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Περιεχόμενα

Περίληψη 2

Εισαγωγή 6

Φυλογενετική ανάλυση 7

Γενικά 7

Το πρόγραμμα RAxML 8

Η φυλογενετική συνάρτηση πιθανοφάνειας 9

General Purpose computing on Graphics Processor Units (GPGPU) 11

Γενικά 11

Το μοντέλο προγραμματισμού CUDA (Compute Unified Device Architecture) - Η Αρχιτεκτονική Fermi 13

Διαδικασία Μεταφοράς RAxML σε GPGPU 17

Profiling 17

Γενική λεπτομερής παραλληλοποίηση εφαρμογής 17

Αρχική στρατηγική υβριδικής εκτέλεσης σε CPU/GPU 19

Διαχείριση μεταφοράς δεδομένων από και προς τη GPU 21

Οργάνωση μνήμης για μεταφορά δεδομένων με μία κλήση 21

Aligned memory error 22

Αποτελέσματα στο χρόνο εκτέλεσης 23

Μείωση πολλαπλών κλήσεων του device 24

Master – worker scheme on GPU 24

Υλοποίηση GPU global barrier 26

Μεταφορά φυλογενετικού δέντρου (διπλά διασυνδεμένη λίστα) 29

Διαχείριση αναδρομικών κλήσεων 30

Συμπεράσματα - Αποτελέσματα στο χρόνο εκτέλεσης 31

Τεχνικές μη δυνατές για εκμετάλλευση λόγω της δομής του αλγορίθμου 33

Βιβλιογραφία 34

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Εισαγωγή

Στη βιοπληροφορική (Bioinformatics) η ανακατασκευή του εξελικτικού δέντρου από μια μοριακή ακολουθία δεδομένων είναι σχετικά παλιό πρόβλημα, δεδομένου ότι σχεδόν τρεις δεκαετίες πριν (1981) ο Joe Felsenstein δημοσίευσε την εργασία του [3] σχετικά με τον υπολογισμό του βαθμού πιθανοφάνειας. Ωστόσο σημαντικές εξελίξεις στις τεχνικές μοριακών ακολουθιών με την εισαγωγή για παράδειγμα του, “the 454 sequencers” [4], έφεραν στην επιφάνεια μια πρωτοφανή πληθώρα δεδομένων.

Επιπλέον, τα τελευταία χρόνια παρατηρείται η ανάδειξη των πολυπύρηνων (multi-cores) αρχιτεκτονικών όπως οι GPUs ή ο IBM Cell που έφεραν νέες προκλήσεις στην φυλογενετική ανάλυση και ειδικά όσον αφορά την οργάνωση της εκτέλεσης της φυλογενετικής συνάρτησης πιθανοφάνειας (Phylogenetic Likelihood Function - PLF). Στην πραγματικότητα, η κοινότητα της βιοπληροφορικής βρίσκεται σε διαρκή αγώνα να ανταπεξέλθει στην ολοένα και μεγαλύτερη συσσώρευση δεδομένων αναπτύσσοντας ολοένα και πιο δυνατά εργαλεία για την επεξεργασία και διαχείριση των δεδομένων.

Ενδεικτικά είναι συνηθισμένο για ένα πρόγραμμα στον πραγματικό κόσμο που εκτελεί ανάλυση μέγιστης πιθανοφάνειας (Maximum Likelihood - ML) να απαιτεί περισσότερα από 50GB μνήμη και η εκτέλεσή του να παίρνει πάνω από 2 εκατομμύρια ώρες στη CPU. Επομένως η μελέτη της συμπεριφοράς του αλγορίθμου σε μία αρχιτεκτονική πολλαπλών πυρήνων όπως οι νέας γενιάς GPU παρουσιάζει ενδιαφέρον.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Φυλογενετική ανάλυση

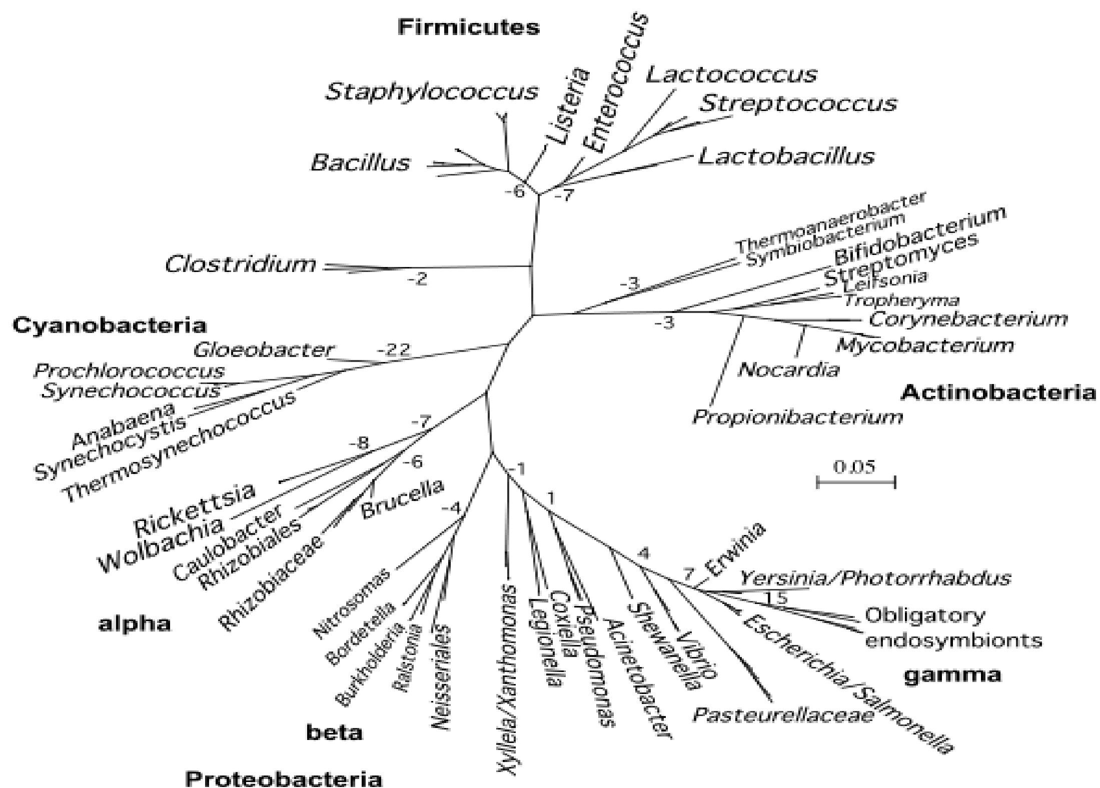
Γενικά

Σκοπός της φυλογενετικής ανάλυσης (Phylogenetic inference) είναι η ανακατασκευή της εξελικτικής ιστορίας ενός πλήθους **n** οργανισμών από τις αντίστοιχες μοριακές τους ακολουθίες. Μια μοριακή ακολουθία (taxa) που αναπαριστά έναν οργανισμό μπορεί να αποτελείται από ένα μίγμα DNA, πρωτεϊνών, ή/και μορφολογικών χαρακτήρων.

Τα δεδομένα εισόδου για μια φυλογενετική ανάλυση αποτελούνται από μια πολλαπλή «καλά» ευθυγραμμισμένη ακολουθία **taxa** των **n** οργανισμών ίδιου μεγέθους **m**. Ένα παράδειγμα μιας τέτοιας ακολουθίας τεσσάρων οργανισμών φαίνεται παρακάτω.

```
Cow       ATGGCATATCCCA-ACAACTAGGATTCCAAGA----ACATCACCAATCATAGAAGAACTA
Chicken   ATGGCCAACCACTCCCAACTAGGCTTTC-AGACGCCTCA-CCCCCATCATAGAAGAGCTC
Human     ATGGCACAT---GCGCAAGTAGGTCTAC-AGACGCTACT-CCCCTATCATAGAAGAGCTT
Mouse     ATGG----CCCATTCCAACTTGGTCTACAAGACGCCACATCCCCTATTATAGAAGAGCTA
```

Ένα φυλογενετικό δέντρο συνήθως αναπαρίσταται ως ένα δυαδικό δέντρο χωρίς ρίζα. Το σύνολο των **n** οργανισμών (taxa) που εξετάζονται τοποθετούνται στα φύλλα του δέντρου και οι εσωτερικοί κόμβοι αναπαριστούν τους εξαφανισμένους κοινούς προγόνους. Οι αριθμοί σε κάθε διακλάδωση φανερώνουν την εκτίμηση για τον χρονικό προσδιορισμό της μετάλλαξης/εξέλιξης.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Είναι σημαντικό να τονίσουμε ότι το πρόβλημα της εύρεσης του βέλτιστου δέντρου είναι **NP-hard**. Ο αριθμός των εναλλακτικών διακριτών δέντρων χωρίς ρίζα για n οργανισμούς είναι $\prod_{i=3}^{n}(2i-5)$ [9]. Ενώ υπάρχει αρκετή βιβλιογραφία πάνω στους ευριστικούς αλγορίθμους αναζήτησης που χρησιμοποιούνται για το πρόβλημα της βελτιστοποίησης του βαθμού πιθανοφάνειας, όλοι βασίζονται στην επαναληπτική εκτέλεση της συνάρτησης πιθανοφάνειας για την εξερεύνηση του δέντρου, που αντιπροσωπεύει και το κύριο σημείο συμφόρησης της μνήμης και εντατικών υπολογισμών.

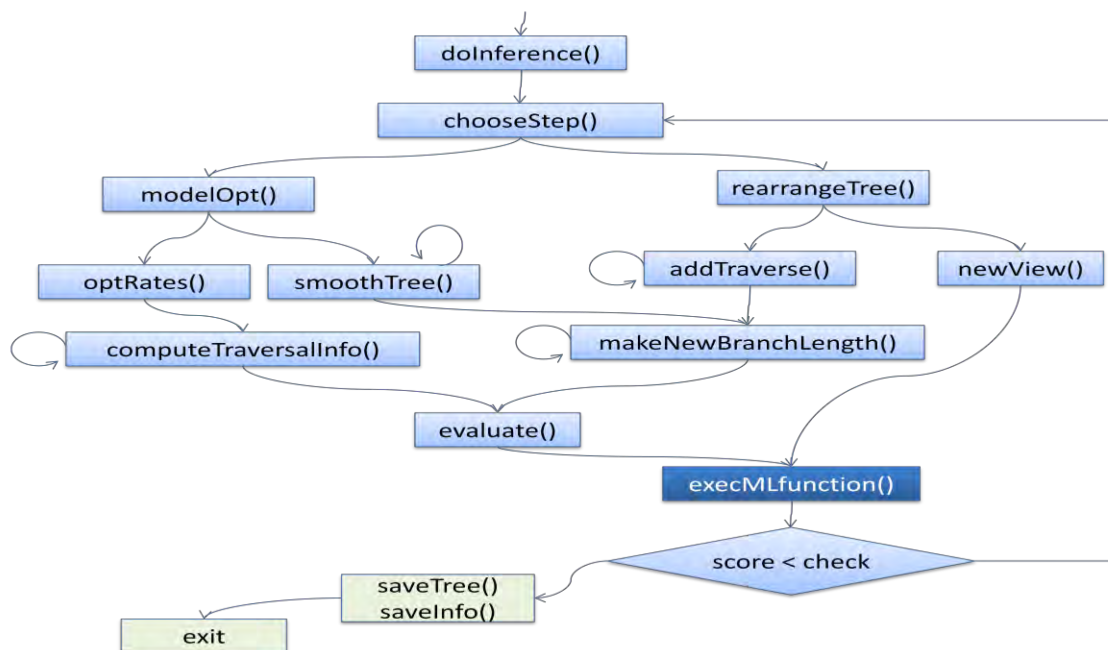
Τα φυλογενετικά δέντρα έχουν αρκετές σημαντικές εφαρμογές στην έρευνα της ιατρικής και της βιολογίας, με κυρίαρχα προγράμματα φυλογένειας μέγιστης πιθανοφάνειας (**Maximum Likelihood - ML**) να είναι τα PAML, PHYML, PAUP, GARLI, RAxML, IQPNNI, MrBayes, PhyloBayes, BEAST [1].

Το πρόγραμμα RAxML

Το πρόγραμμα RAxML (Randomized Accelerated Maximum Likelihood) χρησιμοποιείται για την εκτέλεση μεγάλης κλίμακας φυλογενετικών αναλύσεων με τη μέθοδο μέγιστης πιθανοφάνειας.

Αρχικά δημιουργεί ένα φειδωλό δέντρο (parsimony tree) χρησιμοποιώντας το πρόγραμμα dnapars του πακέτου PHILIP και ξεκινάει μια ακολουθία αναδιατάξεων στα υποδέντρα. Σε κάθε αναδιάταξη υπολογίζει το βαθμό πιθανοφάνειας (Likelihood Score) εκτελώντας την φυλογενετική συνάρτηση (Phylogenetic Function) για κάθε τοπολογία. Αν ο βαθμός είναι μεγαλύτερος τότε χρησιμοποιείται το δέντρο για τη συνέχιση των αναδιατάξεων. Μετά από κάθε βήμα αναδιάταξης πραγματοποιείται η διαδικασία βελτιστοποίησης των μηκών των κλαδιών (branch length optimization) και επαναλαμβάνεται ο υπολογισμός του βαθμού πιθανοφάνειας, μέχρι να καλυφθούν ορισμένα κριτήρια σύγκλησης οπότε και σταματάει η διαδικασία.

Παρακάτω φαίνεται ένα απλοποιημένο διάγραμμα ροής της εφαρμογής:

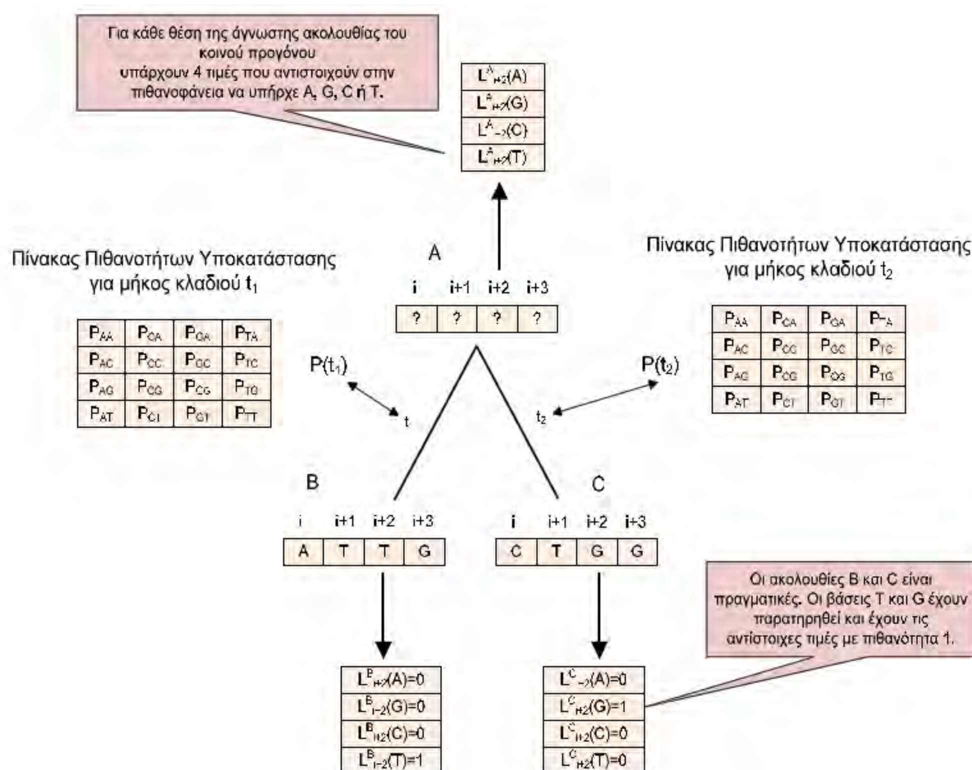


Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Η φυλογενετική συνάρτηση πιθανοφάνειας

Ας δούμε πιο αναλυτικά τον τρόπο λειτουργίας της φυλογενετικής συνάρτησης πιθανοφάνειας η οποία είναι και ο πυρήνας της εφαρμογής. Όπως έχουμε πει είσοδός μας είναι οι n ακολουθίες οργανισμών μήκους m. Υποθέτουμε ότι έχουμε μια δοσμένη τοπολογία δέντρου από τη οποία θα υπολογίσουμε το βαθμό πιθανοφάνειας.

Οι περισσότεροι υπολογισμοί επιδρούν, εκτός από τις άλλες παραμέτρους που θα αναλύσουμε παρακάτω, κυρίως στους πίνακες πιθανοτήτων $\boldsymbol{L}$. Κάθε εγγραφή του πίνακα πιθανοτήτων $L(c)$ στα φύλλα και στους εσωτερικούς κόμβους του δέντρου μας, περιέχει στη θέση c =1..m τις τέσσερεις πιθανότητες P(A), P(C), P(G), P(T). Αυτές εκφράζουν την πιθανότητα να παρατηρηθεί ένα νουκλεοτίδιο A, C, G, T σε μια συγκεκριμένη στήλη c της ακολουθίας εισόδου. Οι πίνακες L αρχικοποιούνται κατάλληλα στα φύλα θέτοντας την τιμή 1.0 στην αντίστοιχη θέση του νουκλεοτιδίου. Για παράδειγμα στην παρατήρηση του νουκλεοτιδίου A ο πίνακας $\boldsymbol{L}$ θα είναι L=[1.0, 0.0, 0.0, 0.0]. Η μορφή των πινάκων αυτών φαίνεται στην παρακάτω εικόνα:



Έπειτα έχοντας τον πατρικό κόμβο **k** και τους δύο κόμβων-παιδιά του **i** και **j**, τους πίνακες πιθανοτήτων $\boldsymbol{L}^{(i)}$, $\boldsymbol{L}^{(j)}$, τα αντίστοιχα μήκη διακλάδωσης $\boldsymbol{b_i}$, $\boldsymbol{b_j}$ και τους πίνακες πιθανοτήτων μετάβασης $\boldsymbol{P}(\boldsymbol{b_i})$, $\boldsymbol{P}(\boldsymbol{b_j})$ ο υπολογισμός της πιθανότητας να παρατηρηθεί μια **A** στη θέση **c** του διανύσματος $\boldsymbol{L}$ στον πατρικό κόμβο **k** γίνεται ως εξής:

$$L_A^{(k)}(c) = \left(\sum_{S=A}^{T} P_{AS}(b_i) L_S^{(i)}(c)\right)\left(\sum_{S=A}^{T} P_{AS}(b_j) L_S^{(j)}(c)\right)$$

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Ο πίνακας πιθανοτήτων μετάβασης P(b) για ένα δοσμένο μήκος διακλάδωσης b υπολογίζεται ως $P(b) = e^{Qb}$, όπου **Q** ο (4x4) πίνακας υποκατάστασης που περιέχει τις πιθανότητες μετάβασης μεταξύ των νουκλεοτιδίων.

Μόλις υπολογιστούν οι πίνακες πιθανοτήτων μετάβασης των δύο παιδιών της ρίζας i, j ο βαθμός πιθανοφάνειας $\boldsymbol{l(c)}$ για κάθε στήλη **c**=1…m προκύπτει ως ακολούθως (δεδομένου και του μήκους διακλάδωσης $\boldsymbol{b_{vr}}$ μεταξύ των κόμβων I,j):

$$l(c) = \sum_{R=A}^{T} (\pi_R L_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr}) L_S^{(j)}(c))$$

Ο τελικός βαθμός πιθανοφάνειας του δέντρου υπολογίζεται προσθέτοντας το λογάριθμο του κάθε στοιχείου του προηγούμενου διανύσματος:

$$LnL = \sum_{c=1}^{m} \log(l(c))$$

Μια σημαντική ιδιότητα, που μας ενδιαφέρει άμεσα στην παράλληλη εκτέλεση της συνάρτησης είναι η υπόθεση ότι οι οργανισμοί εξελίσσονται ανεξάρτητα, δηλαδή κάθε εγγραφή **c** του πίνακα πιθανοτήτων L μπορεί να υπολογιστεί ανεξάρτητα. Επομένως για μια πλήρη διαπέραση του δέντρου χρειάζεται μόνο μια πράξη ελαχιστοποίησης (reduction) του αποτελέσματος άρα ένα σημείο συγχρονισμού.

Επίσης όσον αφορά τη μνήμη που χρειαζόμαστε για την εκτέλεση της εφαρμογής εκτός από τις δομές αποθήκευσης που περιγράψαμε έως τώρα, μπορεί να χρειαστούμε και επιπλέον μνήμη ανάλογα με το μοντέλο που χρησιμοποιούμε για την ανάλυση. Εμείς θα χρησιμοποιήσουμε το Γ μοντέλο (GAMMA model) στο οποίο οι πράξεις είναι ελάχιστα πιο περίπλοκες καθώς για τον υπολογισμό των πινάκων πιθανοτήτων μετάβασης $P(t)$ λαμβάνονται υπόψη και τέσσερεις διακριτές συντελεστές r0, r1, r2,r3 έτσι ώστε $P(t) = e^{Qtri}$. Αυτό σημαίνει ότι τετραπλασιάζονται οι πράξεις και η κατανάλωση μνήμης.

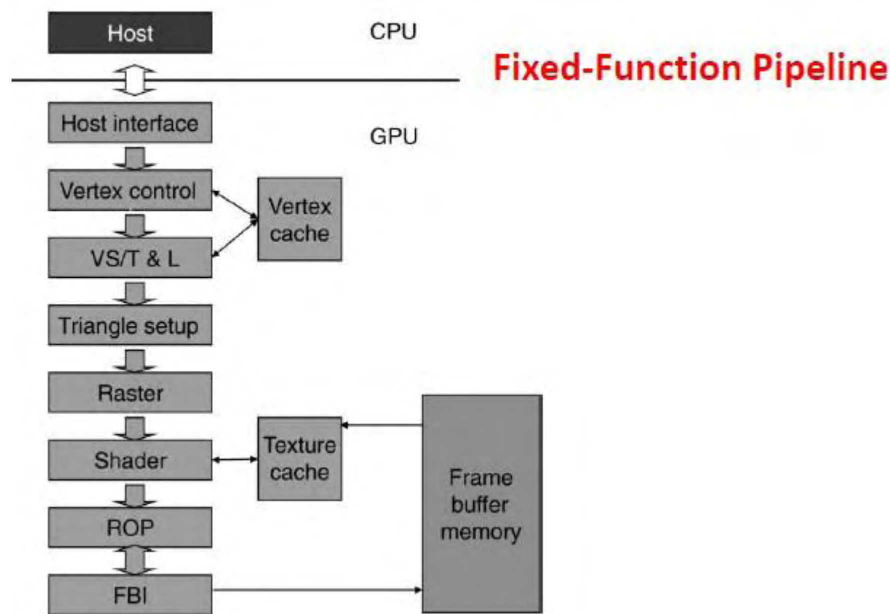
Το πρόγραμμα RAxML ενσωματώνει και χρησιμοποιεί πολύ περισσότερες παραμέτρους όπως επιβάλλεται και από το αντίστοιχο βιολογικό μοντέλο, περαιτέρω ανάλυση των οποίων δεν έχει νόημα για τους σκοπούς της παρούσας εργασίας. Μία αναλυτική περιγραφή γίνεται στο [1].

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

General Purpose computing on Graphics Processor Units (GPGPU)

Γενικά

Από την αρχή της δεκαετίας του 1980 μέχρι το τέλος της δεκαετίας του 1990, το κορυφαίο σε απόδοση υλικό γραφικών χρησιμοποιούσε διοχετεύσεις σταθερών συναρτήσεων (fixed-functions pipelines), οι οποίες ήταν διευθετήσιμες (configurable) αλλά όχι προγραμματίσιμες (programmable). Τότε, απέκτησαν δημοτικότητα διάφορες σημαντικές βιβλιοθήκες διασύνδεσης προγραμματισμού εφαρμογών γραφικών (API – Application Programming Interface) όπως είναι το DirectX (Microsoft) και το OpenGL (open source).



Εικόνα 1 fixed functions pipelines

Την περίοδο 1999-2001 οι επιστήμονες άρχισαν να χρησιμοποιούν τις κάρτες γραφικών για την επιτάχυνση της εκτέλεσης εφαρμογών από διάφορους τομείς καθώς τότε δόθηκε μια γενική δυνατότητα προγραμματισμού σε διάφορα στάδια που μέχρι πριν ήταν σταθερών συναρτήσεων (π.χ. η GPU GeForce 3 το 2001 στο στάδιο VS/T&L). Αυτή ήταν και η απαρχή της χρήσης του όρου GPGPU (General - Purpose computation on GPU) στον προγραμματισμό εφαρμογών.

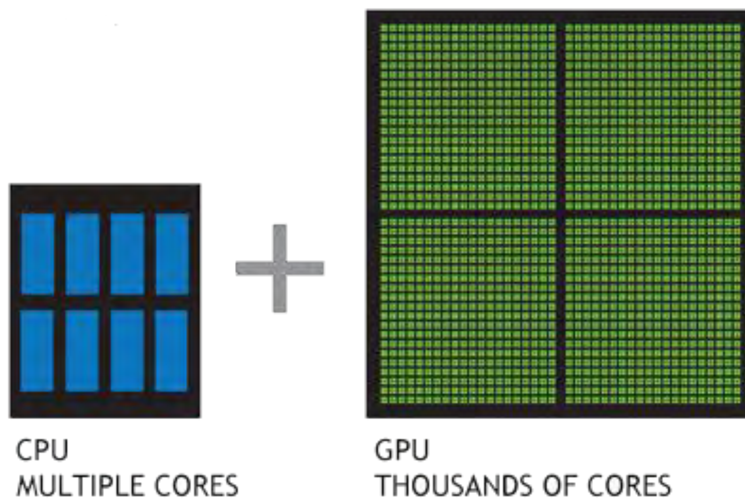
Οποιαδήποτε επιτάχυνση στο χρόνο εκτέλεσης ερχόταν με τη χρήση του API για τον προγραμματισμό της κάρτας, πράγμα το οποίο περιόριζε την πλήρη εκμετάλλευση των δυνατοτήτων της και το εύρος των προγραμματιστών που μπορούσαν να τις αξιοποιήσουν. Για να προσπελάσει ένας προγραμματιστής υπολογιστικούς πόρους έπρεπε να εκφράσει το πρόβλημά του σε εγγενείς λειτουργίες γραφικών έτσι ώστε ο υπολογισμός να μπορεί να ξεκινήσει μέσω κλήσεων API του OpenGL ή DirectX.

Το 2006 με την ανάπτυξη της αρχιτεκτονικής Tesla, η nVidia συνειδητοποίησε ότι η πιθανή χρησιμότητά της θα ήταν πολύ μεγαλύτερη αν οι προγραμματιστές μπορούσαν να θεωρήσουν την GPU ως έναν επεξεργαστή δηλώνοντας ρητά τις πτυχές της εργασία τους που περιέχουν παραλληλία δεδομένων. Ταυτόχρονα με την διάθεση μοντέλων

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

προγραμματισμού όπως τα CUDA (nVidia) και OpenCL (open source) ουσιαστικά οι κάρτες γραφικών έγιναν πλήρως προγραμματίσιμες και καθώς επίσης τα μοντέλα είναι επέκταση γνωστών γλωσσών προγραμματισμού (C/C++, FORTRAN), έφεραν τη δυνατότητα ανάπτυξης εφαρμογών σε ένα τεράστιο εύρος προγραμματιστών.

Το GPU computing έφερε επιταχύνσεις στην εκτέλεση εφαρμογών γενικού σκοπού μεταφέροντας τα υπολογιστικά χρονοβόρα κομμάτια τους στην GPU ενώ η υπόλοιπη εφαρμογή εκτελείτε στην CPU. Έτσι ο συνδυασμός CPU+GPU είναι αρκετά δυνατός διότι η CPU αποτελείται από λίγους πυρήνες βελτιστοποιημένους για σειριακή εκτέλεση ενώ η GPU από χιλιάδες μικρότερους πιο αποτελεσματικούς πυρήνες σχεδιασμένους για παράλληλη απόδοση. Τα σειριακά κομμάτια του κώδικα εκτελούνται στη CPU ενώ τα παράλληλα στη GPU.



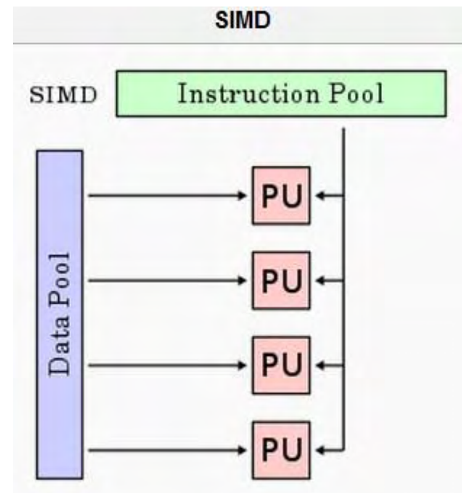
Εικόνα 2 hybrid programming

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Το μοντέλο προγραμματισμού CUDA (Compute Unified Device Architecture) - Η Αρχιτεκτονική Fermi

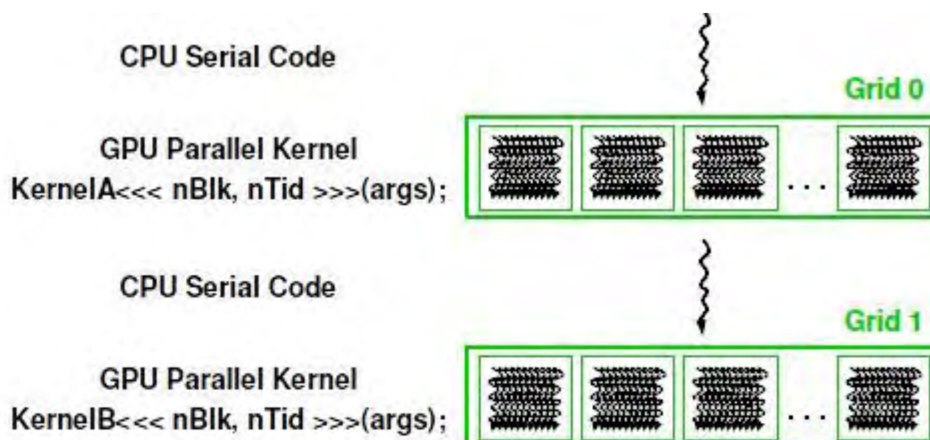
Για τους προγραμματιστές CUDA και OpenCL, οι μονάδες επεξεργασίας γραφικών (GPU) είναι μαζικά παράλληλοι επεξεργαστές αριθμητικών υπολογισμών που προγραμματίζονται σε C/C++ με επεκτάσεις της. Ο προγραμματισμός αυτών των επεξεργαστών δεν απαιτεί την κατανόηση των αλγορίθμων ή της ορολογίας γραφικών.

Σύμφωνα με την κατηγοριοποίηση του Flynn στις παράλληλες αρχιτεκτονικές το μοντέλο CUDA ανήκει στην SIMD(Single Instruction Multiple Data) όπου η ίδια εντολή σε πολλαπλές μονάδες εκτέλεσης επιδρά σε διαφορετικά δεδομένα εκμεταλλευόμενη έτσι τον (πιθανό) εγγενή παραλληλισμό της εκάστοτε εφαρμογής.



Παρακάτω περιγράφουμε εν συντομία το μοντέλο. Ένα πρόγραμμα σε CUDA καλεί το παράλληλο τμήμα κώδικα, τους πυρήνες. Ένας πυρήνας περιέχει ένα πλήθος ανεξάρτητων νημάτων για εκτέλεση οργανωμένων από τον προγραμματιστή σε μπλοκ νημάτων (thread blocks) και πλέγματα (grids). Επομένως η μονάδα εκτέλεσης γραφικών (GPU) δημιουργεί ένα πλέγμα από μπλοκ νημάτων κάθε ένα νήμα από τα οποία εκτελεί ένα στιγμιότυπο του πυρήνα και έχει (μπορεί να εξάγει) το δικό του μοναδικό ID, καταχωρητές (registers), ιδιωτική μνήμη.

Σε επίπεδο εκτέλεσης μπλοκ το μοντέλο CUDA προσφέρει κοινή μνήμη ανάμεσα στα νήματα του ίδιου μπλοκ (shared memory), συγχρονισμό νημάτων με φράγματα (barrier), ενώ κάθε μπλοκ έχει μοναδικό ID. Τέλος σε επίπεδο πλέγματος τα μπλοκ εκτελούν ανεξάρτητα μεταξύ τους τον ίδιο πυρήνα και επικοινωνούν μέσω την καθολικής μνήμης (global memory) έχοντας ως μοναδικό καθολικό σημείο συγχρονισμού την περάτωση της εκτέλεσης του πυρήνα.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

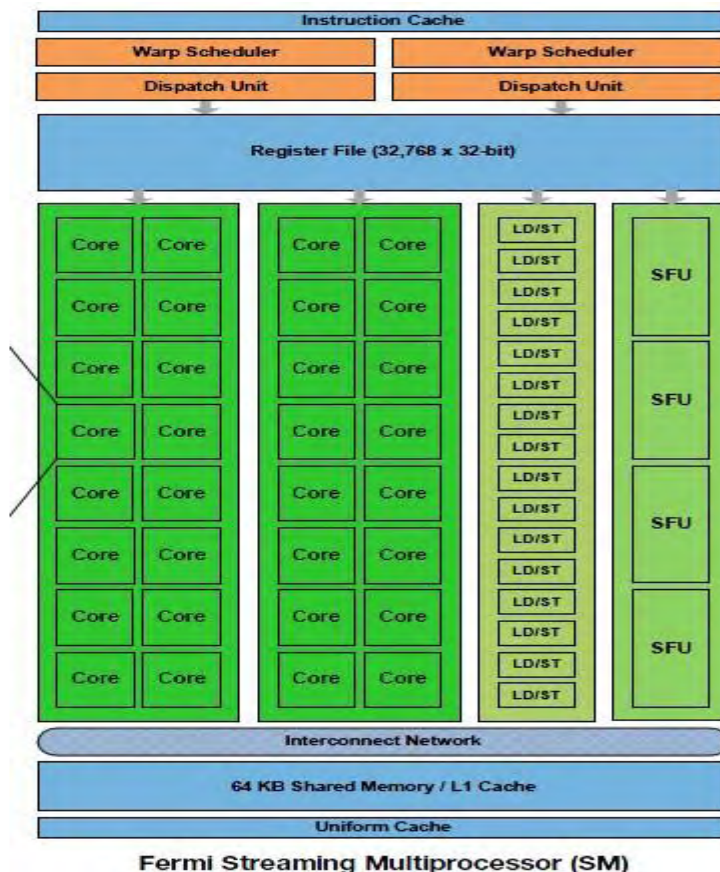
Επομένως, σύμφωνα με τα παραπάνω καταλαβαίνουμε ότι στην αρχιτεκτονική Fermi και συγκεκριμένα στην κάρτα που θα δουλέψουμε GTX480 έχουμε μονάδες εκτέλεσης με διάφορα επίπεδα μνήμης τα οποία είναι κοινά σε διαφορετικές ιεραρχίες πλέγματος/μπλοκ/νήματος. Επίσης διαφορετική είναι και η ταχύτητα προσπέλασης του κάθε επιπέδου μνήμης για την ανάγνωση/εγγραφή δεδομένων.

Οι μονάδες εκτέλεσης είναι οργανωμένες σε Streaming Multiprocessors. Η GTX480 διαθέτει 15 SM με 32 cores/SM. Άρα συνολικά μπορεί να εκτελεί ταυτόχρονα 15*32=480 νήματα. Η αρχιτεκτονική όμως επιτρέπει τη δημιουργία πολύ περισσότερων νημάτων. Συγκεκριμένα τα χαρακτηριστικά της συσκευής μας είναι:

```
Total amount of global memory:                    1576468480 bytes
Number of multiprocessors:                                     15
Number of cores:                                              480
Total amount of constant memory:                      65536 bytes
Total amount of shared memory per block:              49152 bytes
Total number of registers available per block:              32768
Warp size:                                                     32
Maximum number of threads per block:                         1024
Maximum sizes of each dimension of a block:       1024 x 1024 x 64
Maximum sizes of each dimension of a grid:      65535 x 65535 x 1
```

Πιο αναλυτικά, η δρομολόγηση της εκτέλεσης γίνεται σε επίπεδο warp. Τα νήματα κάθε μπλοκ οργανώνονται σε ομάδες των 32 και ανατίθενται σε κάθε SM για εκτέλεση. Το υλικό μπορεί αυτόματα να διαχειριστεί την απόκλιση (divergence) που πιθανόν να υπάρξει στην εκτέλεση του warp. Επίσης κάθε SM έχει 2 warp schedulers, 2 dispatch units, μονάδες ειδικών συναρτήσεων (SFU), μονάδες φόρτωσης/αποθήκευσης, καθώς και 64KB παραμετροποιήσιμη cache/shared memory.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

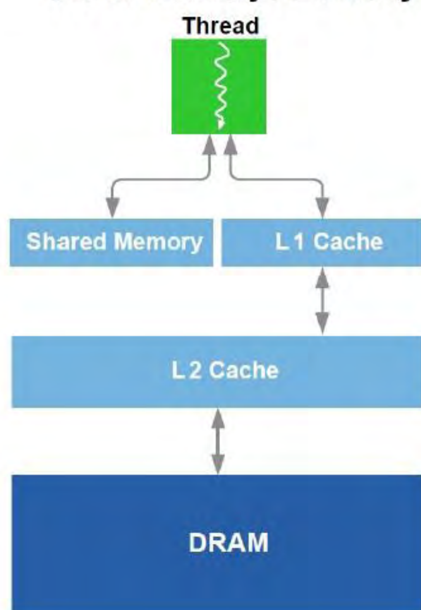


Fermi Streaming Multiprocessor (SM)

Το γεγονός του πλήθους νημάτων που η αρχιτεκτονική μας επιτρέπει να δημιουργήσουμε, σε συνδυασμό με την εναλλαγή τους χωρίς κόστος, επιδρά θετικά στο χρόνο εκτέλεσης καθώς μπορεί να επικαλυφθεί η καθυστέρηση πρόσβασης στη μνήμη με την εκτέλεση νημάτων που είναι έτοιμα. Επομένως ο παραλληλισμός των εφαρμογών για μία τέτοια αρχιτεκτονική επιβάλει τη λεπτομερή διαμέριση του προβλήματος έτσι ώστε να δημιουργηθεί ένας ικανός αριθμός νημάτων για την εκμετάλλευσή της.

Σε επίπεδο μνήμης η αρχιτεκτονική Fermi περιέχει τις shared memory, L1 cache, L2 cache, DRAM. Η shared memory, διαχειρίσιμη από τον προγραμματιστή, είναι γρήγορη μνήμη μεγέθους 48 ή 16KB (ανάλογα το μέγεθος της L1 cache) η οποία είναι κοινή σε όλα τα νήματα του μπλοκ. Η Global memory DRAM είναι κοινή σε όλα τα νήματα έχει αρκετό μέγεθος για την αποθήκευση των δεδομένων (1,5GB) αλλά η προσπέλασή της είναι πιο αργή (100x σε σχέση με την shared). Η DRAM χρησιμοποιείται για την αποθήκευση των δεδομένων που μεταφέρονται από τον host και των αποτελεσμάτων μετά την επεξεργασία τους. Ενδιάμεσα για την επιτάχυνση των προσπελάσεων από την Global memory υπάρχει η L1 και η L2 cache. Η L1 cache είναι non-coherent δηλαδή μία αλλαγή από κάποιο νήμα στην Global memory, μεταβλητής που υπάρχει στην L1 cache δεν ενημερώνει την υπάρχουσα τιμή στην cache. Το μέγεθός της μπορούμε να το αυξομειώσουμε κατά το compile time σε σχέση με τη shared memory (16/48KB). Τέλος η L2 cache είναι μεγαλύτερη (768 KB), κοινή σε όλα τα μπλοκ και είναι coherent.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Διαδικασία Μεταφοράς RAxML σε GPGPU

Profiling

Το profiling της εφαρμογής έγινε με το πρόγραμμα **vTune** της **Intel**. Όπως κάθε άλλο πρόγραμμα που χρησιμοποιείται σε φυλογενετικές αναλύσεις έτσι και στο RAxML το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης καταναλώνεται στον υπολογισμό του βαθμού πιθανοφάνειας μιας δοσμένης τοπολογίας δέντρου.

Συγκεκριμένα το profiling έδειξε ότι το 98% του χρόνου καταναλώνεται σε τρεις συναρτήσεις εκ των οποίων: το 80% στη **newView**(), η οποία υπολογίζει τους πίνακες πιθανοφάνειας που περιγράψαμε στην αρχή, 15% στη **makenewz**() η οποία βελτιστοποιεί το μήκος των κλαδιών χρησιμοποιώντας τη μέθοδο Newton – Raphson και το 3% στην **evaluate**() όπου υπολογίζεται ο τελικός βαθμός πιθανοφάνειας.

Θεωρητικά, σύμφωνα με το νόμο του Amdahl το μέγιστο speedup που μπορούμε να πετύχουμε μεταφέροντας τη newView() στην GTX480 είναι:

$$S(N) = \frac{1}{(1-p)+\frac{p}{N}} = \frac{1}{(1-0.8)+\frac{0.8}{480}} \simeq 4.95$$

Γενική λεπτομερής παραλληλοποίηση εφαρμογής

Στην παρούσα stable έκδοση του RAxML-7.2.6 έχει υλοποιηθεί με pthreads ένα σχήμα από master-worker threads. Το μόνο νήμα ουσιαστικά που χρειάζεται να «καταλάβει» την τοπολογία του δέντρου είναι το master. Τα worker threads χρησιμοποιούνται για να διεκπεραιώσουν τους εντατικούς υπολογισμούς κινητής υποδιαστολής, δεσμεύουν τη δικιά τους περιοχή μνήμης για την επεξεργασία των δεδομένων και επιδρούν μόνο στο δικό τους κομμάτι από τους πίνακες πιθανοτήτων L.

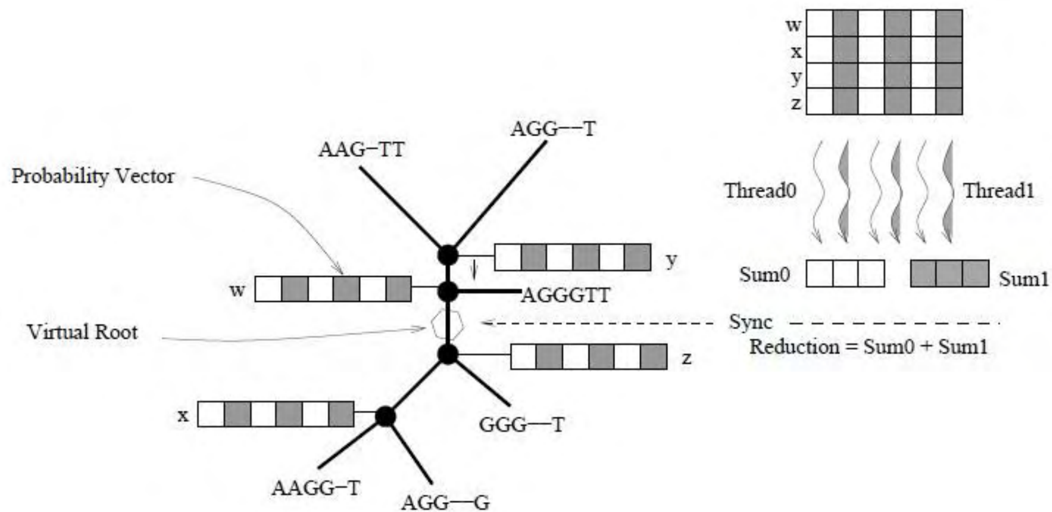
Οι workers κάνουν τη διαπέραση του δέντρου άρα και την εκτέλεση των πράξεων κάθε φορά, σύμφωνα με τα δεδομένα που είναι αποθηκευμένα στη δομή που περιέχει τις πληροφορίες διάτρεξης (traversal descriptor). Συγκεκριμένα κάθε φορά πρέπει να ξέρουν τον αριθμό των κόμβων που θα προσπελάσουν (count), το είδος του κόμβου(tipCase), τους κόμβους πατέρα-παιδιά(pNumber, rNumber, qNumber), και το αντίστοιχο μήκος διακλάδωσης μεταξύ τους (qz, rz).

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

```
typedef struct
{
  int tipCase; /* tip case */
  int pNumber; /* ancestral vector number */
  int qNumber; /* left child number */
  int rNumber; /* right child number */
  double qz[NUM_BRANCHES]; /* branch length(s) for p <-> q */
  double rz[NUM_BRANCHES]; /* branch length(s) for p <-> r */
} traversalInfo;

typedef struct
{
  traversalInfo *ti; /* array of traversalInfo entries */
  int count;         /* number of nodes to traverse    */
} traversalData;
```

Τελικά όπως φαίνεται και στην επόμενη εικόνα κάθε worker thread μπορεί να εκτελέσει τους υπολογισμούς ανεξάρτητα από τους υπόλοιπους σεβόμενο την σειρά διαπέρασης των κόμβων που έχει αποφασίσει το master thread. Άρα καταλήγουμε στο ότι μοναδικό σημείο συγχρονισμού μεταξύ τους είναι η εικονική ρίζα του δέντρου για τον υπολογισμό του βαθμού πιθανοφάνειας.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Αρχική στρατηγική υβριδικής εκτέλεσης σε CPU/GPU

Ο γενικός αλγόριθμος εκτέλεσης της εφαρμογής είναι:

1. Υπολογισμός της τοπολογίας του δέντρου
2. Εξαγωγή της σειράς διάτρεξης των κόμβων
3. Βελτιστοποίηση μήκους διακλαδώσεων
4. Εκτέλεση των υπολογισμών (ML function) και αποθήκευση του βαθμού πιθανοφάνειας
5. Έλεγχος του αποτελέσματος και επιστροφή στο βήμα 2 ή συνέχεια

Στο βήμα 4 είναι η εκτέλεση της συνάρτησης υπολογισμού βαθμού πιθανοφάνειας και περιλαμβάνει το κομμάτι που καταναλώνει το 80% του χρόνου εκτέλεσης όπως περιγράψαμε πριν. Επίσης όπως έχουμε πει μπορεί να εκτελεστεί παράλληλα άρα είναι το αρχικό υποψήφιο κομμάτι κώδικα για μεταφορά της εκτέλεσης στη GPU και την επιτάχυνσή του. Επομένως ο αλγόριθμος γίνεται:

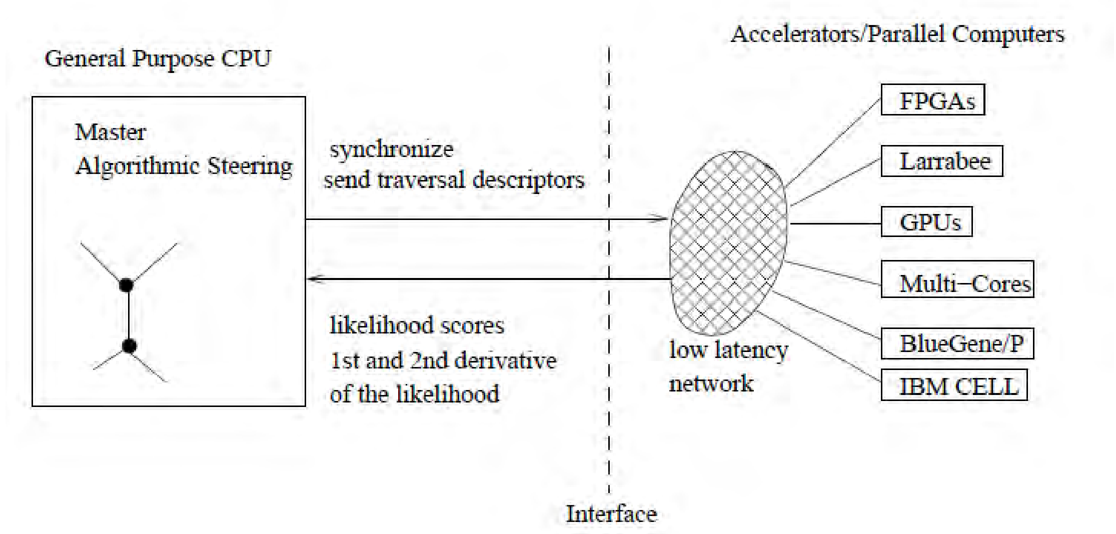
1. Υπολογισμός της τοπολογίας του δέντρου
2. Εξαγωγή της σειράς διάτρεξης των κόμβων
3. Βελτιστοποίηση μήκους διακλαδώσεων
4. Μεταφορά δεδομένων προς τη GPU
5. Εκτέλεση των υπολογισμών στη GPU
6. Επιστροφή αποτελέσματος στη RAM (διάνυσμα με το αποτέλεσμα κάθε GPU νήματος)
7. Reduction με πρόσθεση του αποτελέσματος
8. Έλεγχος του αποτελέσματος και επιστροφή στο βήμα 2 ή συνέχεια

Τα βήματα 1, 2, 3, 7, 8 εκτελούνται στη CPU ενώ το βήμα 3 - όπου περιλαμβάνονται και οι συναρτήσεις που καταναλώνουν τον περισσότερο χρόνο εκτέλεσης – εκτελείται ταυτόχρονα από τα νήματα τηςGPU.

Με τη μεταφορά της εκτέλεσης των workers στην GPU έχουμε καταφέρει η ίδια δουλειά να σπάσει σε περισσότερα κομμάτια σε σχέση με την CPU διότι έχουμε περισσότερους πυρήνες(μονάδες εκτέλεσης) στη διάθεσή μας με αποτέλεσμα κάθε νήμα εκτέλεσης να έχει λιγότερους υπολογισμούς. Η εκτέλεση του κύριου νήματος, με τον υπολογισμό της τοπολογίας και την αρχικοποίηση/βελτιστοποίηση των μεταβλητών σε κάθε φάση και την

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

ενορχήστρωση της εκτέλεσης παραμένει στη CPU.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

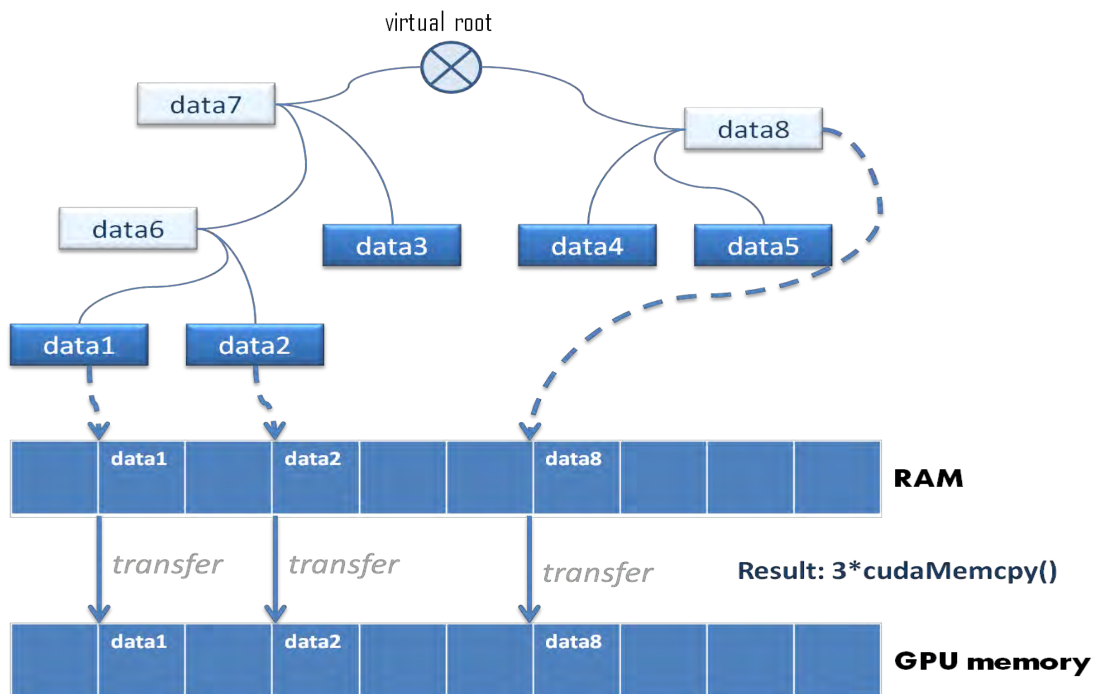
Διαχείριση μεταφοράς δεδομένων από και προς τη GPU

Καθώς ο χώρος δεδομένων δεν είναι κοινός μεταξύ των μονάδων εκτέλεσης πρέπει κάθε αλλαγή σε μία μονάδα να κρατά ενήμερη την άλλη. Ο μόνος τρόπος για να γίνει αυτό είναι μέσω της μεταφοράς των αλλαγών από τη CPU στη GPU και αντίστροφα. Συγκεκριμένα στην παρούσα φάση του αλγορίθμου πριν την εκτέλεση της συνάρτησης πιθανοφάνειας πρέπει να μεταφέρονται εκτός από τον traversal descriptor για κάθε κόμβο του δέντρου και όλες οι παράμετροι που περιγράψαμε μέχρι τώρα οι οποίες υπολογίζονται στην CPU.

Οργάνωση μνήμης για μεταφορά δεδομένων με μία κλήση

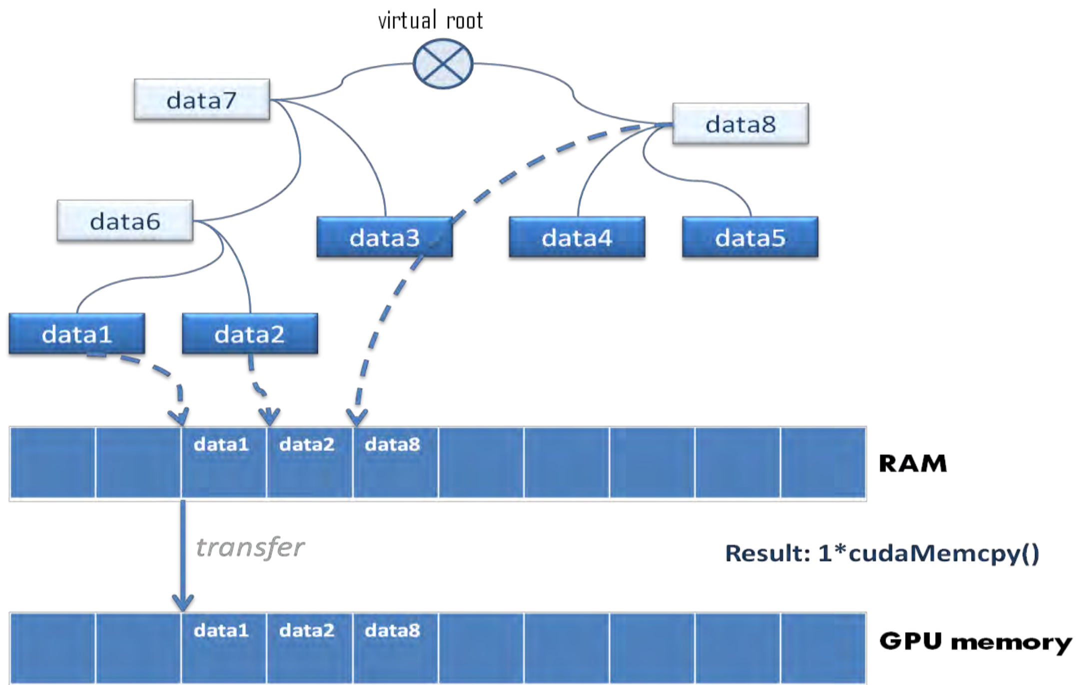
Πολλαπλές μικρές αποστολές δεδομένων προς την GPU έχουν αρνητικά αποτελέσματα ως προς τον χρόνο εκτέλεσης διότι, πρώτον δεν εκμεταλλευόμαστε πλήρως το bandwidth (μπορούσαμε να στείλουμε περισσότερα δεδομένα στον ίδιο χρόνο) και δεύτερον αναπόφευκτα με κάθε κλήση μεταφοράς δεδομένων – cudaMemcpy() προσθέτουμε overhead της κλήσης/εναλλαγής στο περιβάλλον της GPU.

Επομένως υλοποιήσαμε έναν memory manager ο οποίος οργανώνει τη δέσμευση συνεχόμενου χώρου μνήμης στη RAM έτσι ώστε η αποστολή των δεδομένων στη GPU να γίνεται με μία κλήση της cudaMemcpy(). Όπως φαίνεται και στο επόμενο σχήμα:



Εικόνα 3 Πριν την υλοποίηση memory manager (3 μεταφορές)

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7



Εικόνα 4 Μετά την υλοποίηση του memory manager (1 μεταφορά)

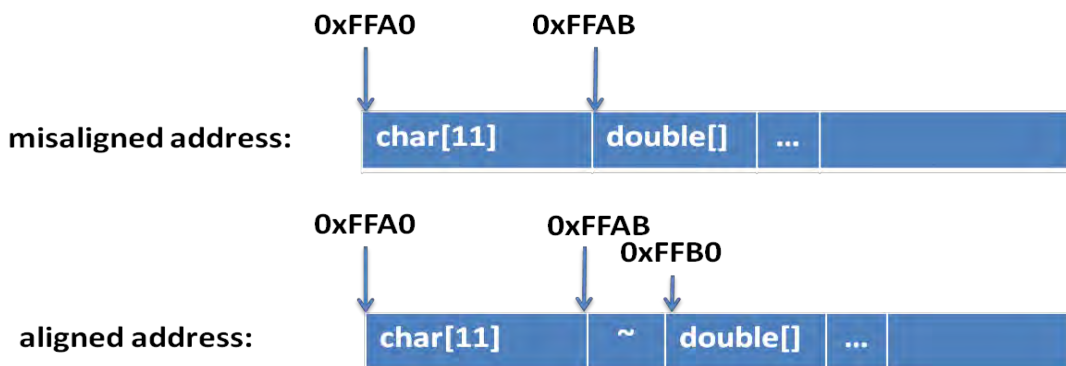
Aligned memory error

Στη μεταφορά την δεδομένων με τον memory manager είχαμε μεταβλητές διαφόρων τύπων, μεταξύ των οποίων και τύπου char για την αποθήκευση των οποίων χρειαζόμαστε 1byte. Επομένως μετά τη δέσμευση χώρου για έναν αριθμό chars υπάρχει περίπτωση η επόμενη διαθέσιμη διεύθυνση για την αποθήκευση του επόμενου τύπου δεδομένων (π.χ. double) να μην είναι πολλαπλάσια του 8 και κατά την προσπέλαση των δεδομένων να έχουμε misaligned address error.

Το πρόβλημα μπορεί να λυθεί με μια mask operation στα τελευταία 4 bits του δείκτη που χρησιμοποιείται για να δείχνει τη θέση του επόμενου προς αποθήκευση στοιχείου. Για παράδειγμα, αν η μεταβλητή memPtr έχει την misaligned διεύθυνση μνήμης 0xFFAB μετά τη mask operation

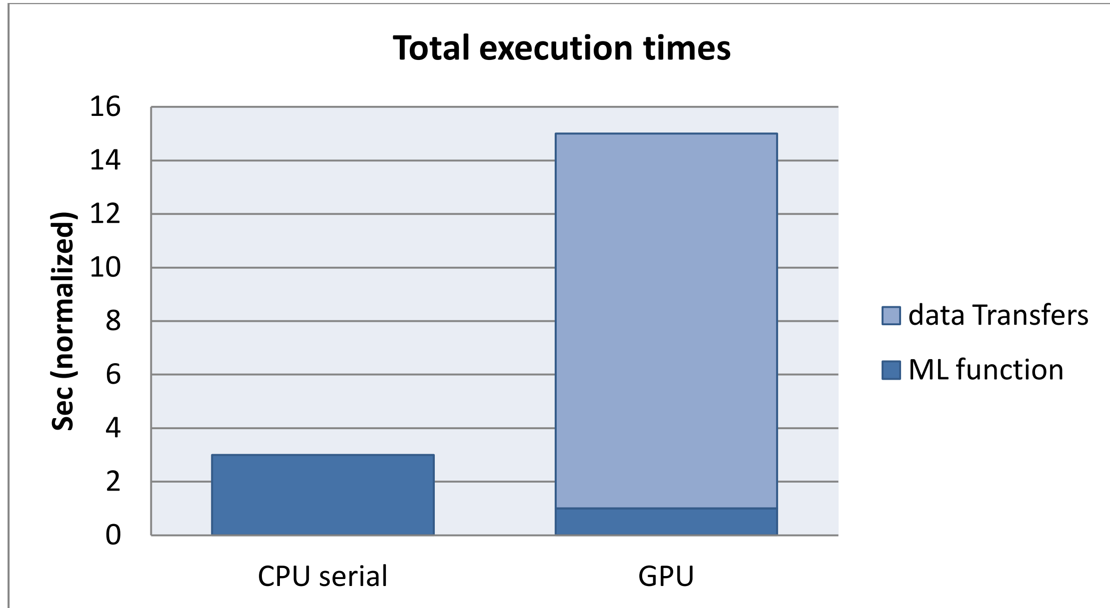
```
void *ptr = ((char *)memPtr+16) & ~ 0x0F;
```

η μεταβλητή ptr θα έχει τη διεύθυνση 0xFFB0 που είναι aligned.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Αποτελέσματα στο χρόνο εκτέλεσης



Ενώ έχουμε πετύχει ένα speedup 3x στο χρόνο εκτέλεσης της φυλογενετικής συνάρτησης ο συνολικός χρόνος της εφαρμογής είναι πιο αργός καθώς κάθε φορά πριν την εκτέλεση του kernel στην GPU έχουμε 2 μεταφορές δεδομένων από και προς τη GPU η καθυστέρηση των οποίων όχι μόνο εξαλείφει το speedup αλλά προσθέτει και overhead.

Άρα ο επόμενος στόχος είναι η ελαχιστοποίηση των δεδομένων που μεταφέρονται.

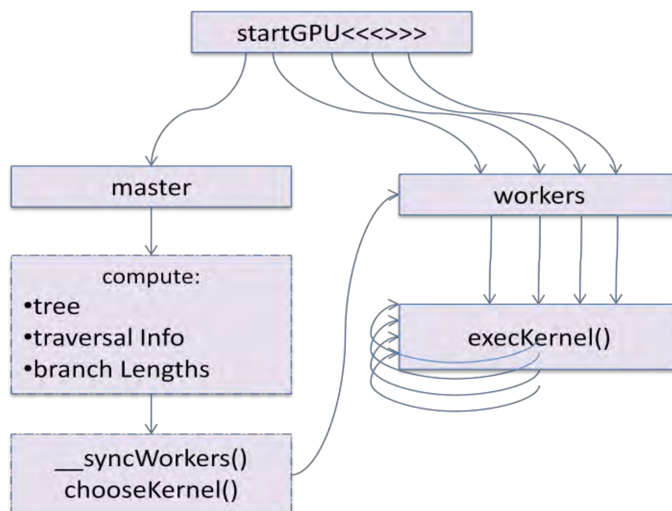
Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Μείωση πολλαπλών κλήσεων του device

Master – worker scheme on GPU

Για να μειώσουμε τις πολλαπλές κλήσεις των kernel από τον host και επομένως τις μεταφορές δεδομένων πρέπει να μεταφέρουμε μέρος του κώδικα που εκτελεί το νήμα της CPU στη GPU. Αυτό συνεπάγεται συγχρονισμό των νημάτων στην GPU τα οποία μέχρι τώρα εκτελούνταν ανεξάρτητα.

Το κομμάτι του master που θα μεταφέρουμε αρχικά διατρέχει το δέντρο με αναδρομικές κλήσεις συναρτήσεων και εξάγει τις πληροφορίες διαπέρασής του. Μόλις τα δεδομένα είναι έτοιμα ειδοποιεί τα worker threads για να υπολογίσουν το βαθμό πιθανοφάνειας. Έπειτα ελέγχει το βαθμό σε σχέση με ένα κριτήριο σύγκλησης και αποφασίζει αν εκτελεστεί ξανά η συνάρτηση πιθανοφάνειας από τους workers ή αν τους ειδοποιήσει να τερματίσουν την εκτέλεσή τους επιστρέφοντας τη ροή στη CPU. Δηλαδή έχει την ευθύνη της ενορχήστρωσης/συγχρονισμού της όλης διαδικασίας. Η λογική της εκτέλεσης στη GPU φαίνεται στο επόμενο διάγραμμα.



Παρακάτω φαίνεται ένα σχήμα υλοποίησης όπου το master thread εκτελεί βελτιστοποίηση των μηκών των διακλαδώσεων και καλεί τους workers για τον υπολογισμό του βαθμού πιθανοφάνειας:

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

```
/*
 * MASTER CODE
 */
_device__ void topLevelMakenewz()
{

    do{
      /*...computations...*/

        execKernel = NEWZCORE;
//set which kernel to execute

        if (tr_executeModel){
          globalBarrier();     //start
                             workers
          globalBarrier();       //end
                             workers
        }

        //reduction results
        dlnLdlz[0] = 0.0;
        d2lnLdlz2[0] = 0.0;
        for(i=0; i<alignLength; i++)
        {
          dlnLdlz[0] += d_dlnLdlz[i];
          d2lnLdlz2[0]              +=
                      d_d2lnLdlz2[i];
        }
              /*...computations...*/
    }
  while (!loopConverged);

  //broadcast to workers that kernel
   finished
  chkLast = 1;
  globalBarrier();
  endGPUexecution = TRUE;
  globalBarrier();

  b.blockFinish = 0;
}
```

```
/*
 * WORKERS CODE
 */
__device__ void parallelExecution()
{
    while(!endGPUexecution)
    {
      globalBarrier();

      if (chkLast)
        globalBarrier();

      //check   if   device   is   done.
        Return to host
      if (!endGPUexecution)
      {
        //choose kernel
        switch (execKernel)
        {
            case NEWZCORE:
            {
                parallelNewzCore();
                break;
            }
            case NEWVIEW:
            {
                parallelNewview();
                break;
            }
            case EVALUATE:
            {
                parallelEvaluate();
                break;
            }
            default:
                assert(0);
        }
        globalBarrier();
//calculations  done!  for  all  device
threads
      }
    }
}
```


Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Υλοποίηση GPU global barrier

Στο προγραμματιστικό μοντέλο CUDA, ο μόνος τρόπος συγχρονισμού όλων των νημάτων που τρέχουν στη συσκευή είναι με τον τερματισμό του πυρήνα εκτέλεσης. Δεν παρέχεται κάποια λειτουργία φράγματος(barrier) που να εγγυάται τη συνεπή εκτέλεση όλων των νημάτων.

Υλοποιήσαμε μια λειτουργία καθολικού φράγματος (global barrier), όπου τα νήματα που φτάνουν πρώτα στο φράγμα περιμένουν με ενεργή αναμονή μέχρι την άφιξη και του τελευταίου νήματος. Όπως φαίνεται παρακάτω στον κώδικα στον global barrier χρησιμοποιούνται οι:

- __syncthreads(): φράγμα που εγγυάται τον συγχρονισμό των νημάτων σε επίπεδο block
- attomicAdd(): εγγυάται την συνεπή αύξηση της τιμής ενός μετρητή που πειράζουν περισσότερα από ένα νήματα ταυτόχρονα.

```
/* global barrier
 * GTX 480 max dims: 120x128 (blocks, threads/block)
 */
__device__ void globalBarrier()
{
 int nofBlocks = gridDim.x*gridDim.y*gridDim.z;
 if (threadIdx.x == 0){
      int mySense = !(b.sense);
      int old = atomicAdd((int *)&b.blockFinish, 1);

      if (old == nofBlocks-1){
            b.blockFinish = 0;
            b.sense = !(b.sense);
      }
      else
      {
          while (mySense != b.sense);
      }
 }
 __syncthreads();
}
```

Εδώ αξίζει να σημειώσουμε ότι για τη λειτουργία του master thread χρησιμοποιούμε ένα ολόκληρο thread block. Συγκεκριμένα αν κάθε block έχει π.χ. 32 threads, στο block που δημιουργήθηκε για να τρέξει το master thread τερματίζουμε την εκτέλεση των 31 threads και το αυτό που μένει, εκτελεί τον κώδικα του master. Αυτό το κάνουμε λόγο της συνάρτησης __syncthreads() που υπάρχει στον global barrier η οποία έχει τον περιορισμό (σύμφωνα με το μοντέλο CUDA) όλα τα νήματα μέσα σε ένα block πρέπει να πέφτουν στην ίδια __syncthreads() (ακολουθώντας το ίδιο μονοπάτι). Ένα παράδειγμα λανθασμένης εκτέλεσης φαίνεται παρακάτω:

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

```
/* σε ένα warp με 32 threads
 * αν ο πίνακας count[] είναι αρχικοποιημένος στο 0
 * τελικά οι θέσεις 16-31 του count[] έχουν τιμή 0
 */
__global__ void syncWrongEx(int *count)
{
    if (threadIdx.x>15)
    {
        __syncthreads();
        count[threadIdx.x] = count[threadIdx.x%16];
    }
    else
    {
        count[threadIdx.x] = threadIdx.x;
        __syncthreads();
    }

}
```

Η ενεργή αναμονή γίνεται σε μία μεταβλητή στην global memory για να είναι ορατές οι αλλαγές από όλα τα νήματα. Επίσης η συσκευή μας GTX480 ενσωματώνει για αύξηση της απόδοσης δύο επίπεδα μνήμης cache L1, L2. H L2 cache είναι συνεπής (coherent) πράγμα που σημαίνει ότι αν αλλάξει η τιμή μιας μεταβλητής στην global memory η τιμή της στην cache μαρκάρεται ως άκυρη, με αποτέλεσμα η επόμενη προσπέλαση της μεταβλητής να γίνει από την global memory, διαβάζοντας τη σωστή τιμή και φέρνοντάς την ταυτόχρονα στην cache για μελλοντικές προσπελάσεις. H L1 cache δεν είναι συνεπής επομένως πρέπει να την απενεργοποιήσουμε στο compile-time για να είναι σωστή η λειτουργία του global barrier.

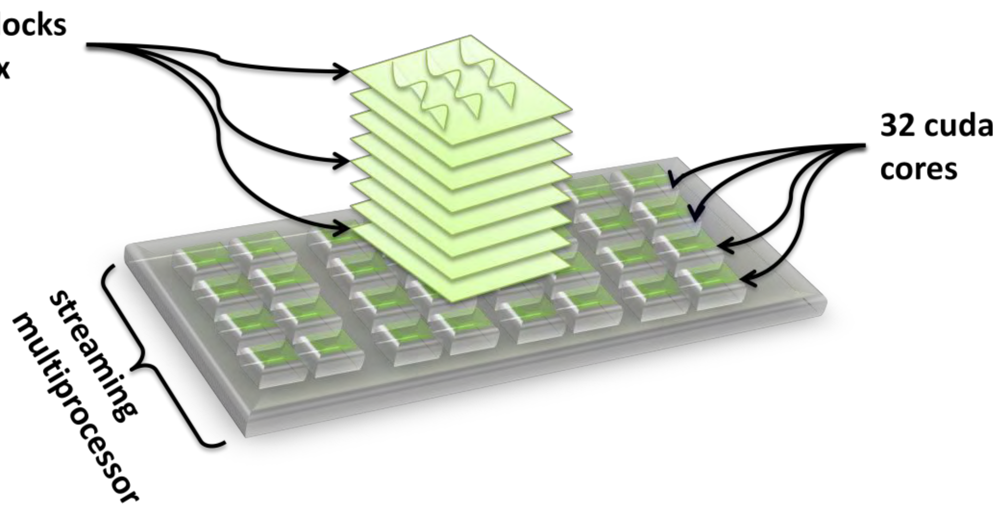
Η απενεργοποίηση της L1 cache ρίχνει την απόδοση για όλη την εφαρμογή. Μία λύση στη χρήση της είναι η απενεργοποίησή της μόνο για την μεταβλητή που διαβάζουμε από τη global memory και θέλουμε να είναι συνεπείς. Αυτό μπορούμε να το πετύχουμε ενσωματώνοντας κώδικα assembly για τις προσπελάσεις μνήμης που γίνονται στη συγκεκριμένη μεταβλητή. Ένα παράδειγμα φαίνεται παρακάτω:

```
__device__ __inline__ int ld_gbl_cg(const int *addr) {
  int return_value;
  asm("ld.global.cg.s32 %0, [%1];" : "=r"(return_value) : "l"(addr));
  return return_value;
}
```

Ένας ακόμη περιορισμός στη χρήση του global barrier λόγω hardware, έγκειται στον μέγιστο αριθμό block/thread που μπορούμε να δημιουργήσουμε. Το μοντέλο προγραμματισμού δεν μας εγγυάται παραχώρηση μονάδων εκτέλεσης σε όλα τα blocks/warps που έχουν δημιουργηθεί παρά μόνο όταν κάποια τελειώσουν την εκτέλεσή τους και ελευθερωθούν πόροι (λογικό αφού το μοντέλο έχει δημιουργηθεί με την υπόθεση ανεξαρτησίας μεταξύ των νημάτων). Άρα πρέπει κάθε φορά να δημιουργούμε τόσα νήματα όσα μπορούν να γίνουν mapping στο hardware με την προϋπόθεση ότι θα τους δοθεί κάποια στιγμή η σειρά εκτέλεσης. Έτσι αποκλείουμε τη περίπτωση του deadlock.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Σύμφωνα με τα παραπάνω, η αρχιτεκτονική Fermi, μπορεί να κάνει scheduling 8 blocks σε κάθε streaming multiprocessor. Επομένως στην GTX480, 15(SM)*8(blocks) = 120 blocks max. Επίσης κάθε multiprocessor μπορεί να έχει μέχρι και 1024 threads άρα 1024(threads)/120(blocks/SM) = 128(thread/block). Άρα οι μέγιστες διαστάσεις είναι 120*128 = 15360 threads τα οποία είναι αρκετά για την εκτέλεση της φυλογενετικής ανάλυσης.



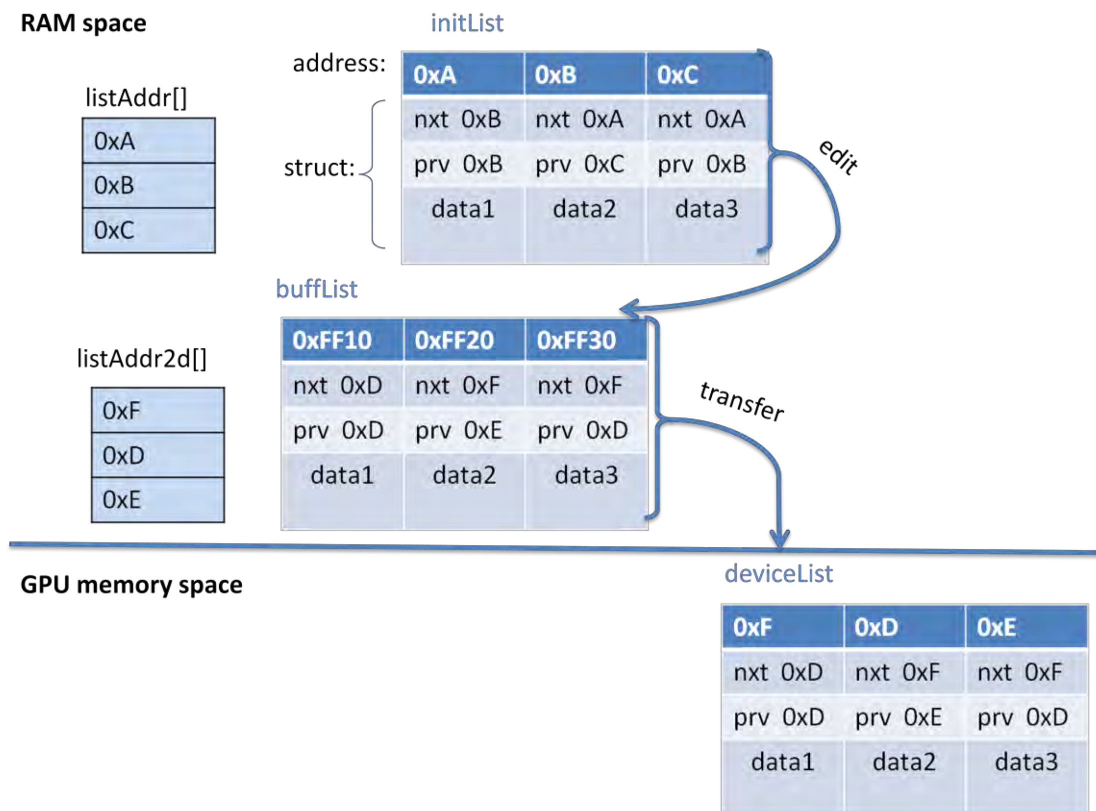
Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Μεταφορά φυλογενετικού δέντρου (διπλά διασυνδεμένη λίστα)

Το δέντρο εκτέλεσης είναι υλοποιημένο με μία διπλά διασυνδεμένη λίστα κάθε στοιχείο της οποίας έχει της πληροφορίες κάθε κόμβου. Το πλήθος των στοιχείων της λίστας και οι διευθύνσεις τους είναι γνωστά πριν την μεταφορά. Στόχος είναι χωρίς καμία αλλαγή στον κώδικα να μπορεί να γίνει προσπέλαση της λίστας από το master thread στη GPU. Επομένως ο αλγόριθμος υλοποίησης είναι ο εξής:

1. Αποθήκευση των διευθύνσεων της λίστας στον πίνακα listAddr[]
2. Δέσμευση χώρου στο device και αποθήκευση των διευθύνσεων στον host στον πίνακα listAddr2d[]
3. Δέσμευση χώρου buffList[] στον host για αποθήκευση της λίστας που θα στείλουμε στο device.
4. Για κάθε στοιχείο της λίστας ελέγχουμε τη θέση των δεικτών διασύνδεσης στον πίνακα listAddr() και βάζουμε στο αντίστοιχο στοιχείο λίστας του buffList[] το στοιχείο της ίδιας θέσης του list2Addr[]
5. Μεταφέρουμε το buffList[] στο χώρο που έχει δεσμευτεί στο device.

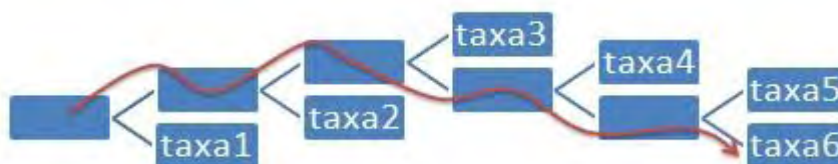
Στην επόμενη εικόνα φαίνεται ο αλγόριθμος, όπου initList είναι η αρχική λίστρα (δέντρο) στη RAM, buffList ένας buffer στη RAM όπου προετοιμάζουμε τη λίστα που θα στείλουμε στη GPU και deviceList η τελική λίστα που έχει η GPU. Βλέπουμε ότι η λογική διασύνδεση των δεδομένων στη GPU είναι ίδια με τη RAM.



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Διαχείριση αναδρομικών κλήσεων

Επί το πλείστον οι κάρτες γραφικών δεν υποστηρίζουν αναδρομή. Ο κύριος λόγος είναι ότι δεν μπορεί το κάθε thread να διατηρεί τη δική του stack. Στην GTX480 όμως η αναδρομή υποστηρίζεται εφόσον γίνει compile για sm_20 (compute capability). Το default stack size για κάθε thread είναι στο 1KB. Στον αλγόριθμό μας, η συνάρτηση που εκτελεί την αναδρομή απαιτεί 16 bytes στο stack για την αποθήκευση των τοπικών μεταβλητών και το μέγιστο μέγεθος του stack για ένα πλήρη μονοπάτι εκτέλεσης (χωρίς αναδρομή) είναι 5360 bytes. Επομένως, αν αυξήσουμε το μέγιστο μέγεθος του stack για κάθε thread σε 16384 bytes θα έχουμε 16384 – 5360 = 11024 bytes διαθέσιμα για αναδρομικές κλήσεις και 11024/16 = 689 μέγιστες συνεχόμενες κλήσεις. Ο αλγόριθμος διατρέχει αναδρομικά, δυαδικά δέντρα χωρίς ρίζα. Το μέγιστο μήκος ενός τέτοιο δέντρου (άρα και το μέγιστο μήκος αναδρομής) για N οργανισμούς-φύλλα είναι N-1, το οποίο σημαίνει ότι δεν έχουμε υπερχείλιση (stack overflow) όσο τρέχουμε τον αλγόριθμο για N<690 οργανισμούς.



Εικόνα 5 - Μέγιστο μήκος αναδρομής 5 σε δέντρο με 6 οργανισμούς

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Συμπεράσματα - Αποτελέσματα στο χρόνο εκτέλεσης

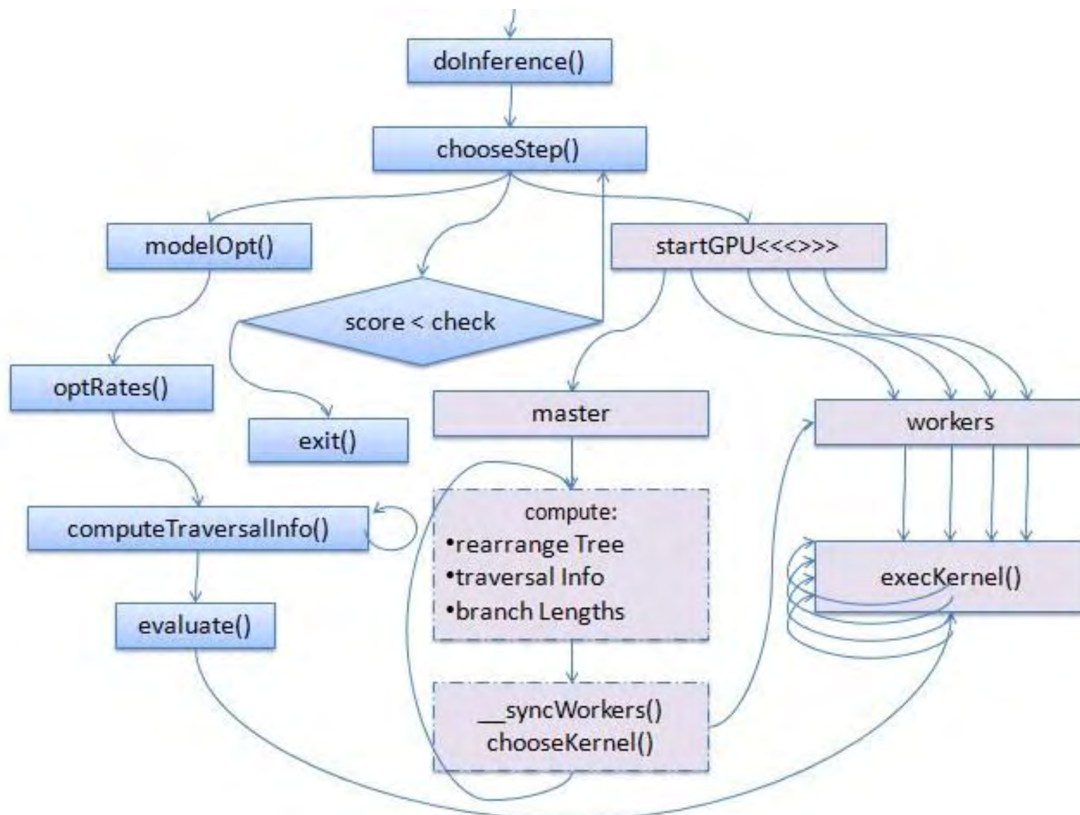
Όπως φαίνεται στο παρακάτω διάγραμμα με τη μεταφορά σχεδόν ολόκληρης της εφαρμογής στη GPU:

1. Το σχήμα master/worker δουλεύει σωστά στην κάρτα. Με το master thread συγκεκριμένα να: κατευθύνει την εκτέλεση, διατρέχει το δέντρο και υπολογίζει με αναδρομή τις πληροφορίες διαπέρασης (traversal Info), τα μήκη διακλαδώσεων (branch lengths) και εκτελεί την αναδιάταξή του (rearrange tree).

2. Οι μεταφορές δεδομένων έχουν μειωθεί στο 0,18% του συνολικού χρόνου καθώς τα μόνα δεδομένα που πρέπει να μεταφέρουμε είναι το αρχικό και τα ενδιάμεσα βέλτιστα δέντρα για να αποθηκευτούν σε αρχεία.

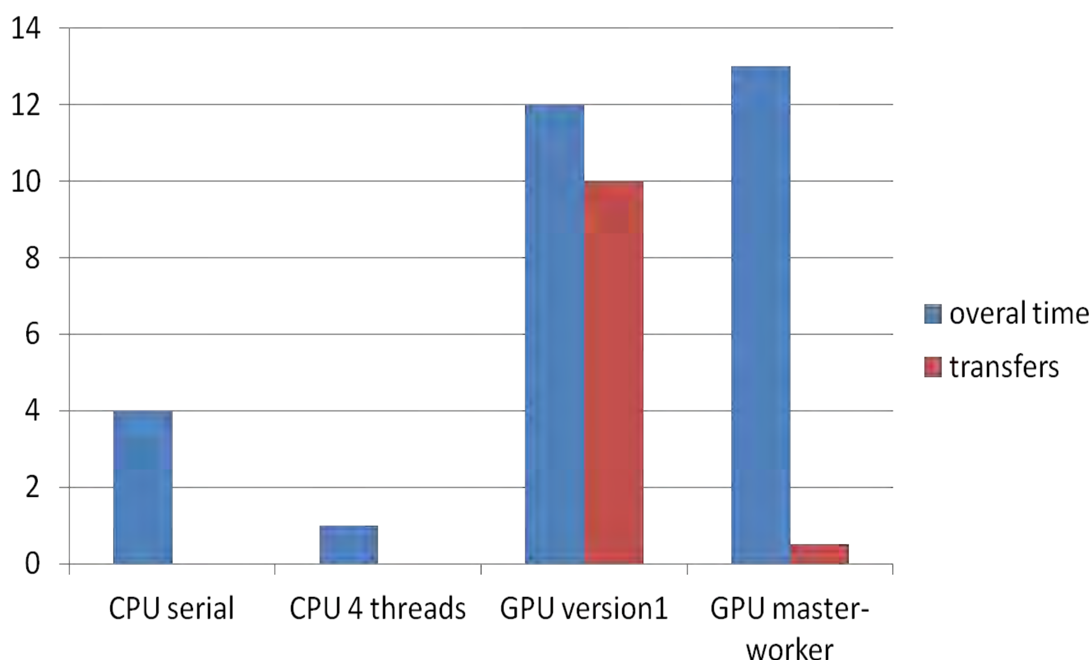
3. Ο συνολικός χρόνος παραμένει χειρότερος από τη σειριακή εκτέλεση πιθανώς λόγο του πυρήνα εκτέλεσης των workers ο οποίος χρειάζεται περεταίρω βελτιστοποίηση.

Τελική μορφή αλγορίθμου υβριδικής εκτέλεσης σε CPU-GPU:



Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Αποτελέσματα:



Αντίθετα με αυτό που περιμέναμε ο χρόνος εκτέλεσης στο σχήμα master-worker δεν βελτιώθηκε σε σχέση με πριν. Αυτό μπορεί να οφείλεται:

1. Αργή εκτέλεση του παράλληλου τμήματος που εκτελούν οι workers σε σχέση με την πρώτη έκδοση του kernel καθώς για την υλοποίηση του σχήματος άλλαξε ο κώδικας εκτέλεσης. Συγκεκριμένα οι περισσότερες από τις πράξεις των threads γίνονται με δομένα και αποτελέσματα στην global memory γεγονός που είναι υποψήφιο να επιφέρει μεγάλη καθυστέρηση. Εδώ μπορεί να υπάρξει βελτίωση με χρήση της shared memory ανά block και με αλλαγή των δομών αποθήκευσης για αποδοτικότερη μεταφορά δεδομένων.
2. Ένας ακόμα πιθανός παράγοντας είναι το μονοπάτι εκτέλεσης του master thread στη GPU να είναι βαρύ για μια μονάδα εκτέλεσης της κάρτας (CUDA core) και σε συνδυασμό με τη δρομολόγηση εκτέλεσης στα warp που είναι διαθέσιμα, να περνά αρκετός χρόνος περιμένοντας τους workers στο global barrier.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Τεχνικές μη δυνατές για εκμετάλλευση λόγω της δομής του αλγορίθμου

Streams: Ένα CUDA stream αντιπροσωπεύει μία σειρά ενεργειών που εκτελείτε ακολουθιακά. Πολλαπλά streams μπορούν να εκτελεστούν ταυτόχρονα. Η χρήση των CUDA streams επιτρέπει την ταυτόχρονη μεταφορά δεδομένων από και προς την GPU με την εκτέλεση του πυρήνα της εφαρμογής. Έτσι υπάρχει επικάλυψη των εναργειών με αποτέλεσμα τη μείωση του χρόνου εκτέλεσης. Ένα παράδειγμα με streams όπου επιτυγχάνεται περίπου 2x speedup σπάζοντας τις μεταφορές και την εκτέλεση του πυρήνα σε 3 κομμάτια ανεξάρτητα μεταξύ τους, φαίνεται παρακάτω:

Στη RAxML η εξάρτηση δεδομένων μεταξύ των εκτελέσεων της φυλογενετικής συνάρτησης δεν επιτρέπει τη χρήση τους, καθώς κάθε φορά πρέπει να ελέγχεται ο βαθμός πιθανοφάνειας της δοσμένης τοπολογίας πριν την απόφαση εκτέλεσης του επόμενου βήματος.

Concurrent kernel execution: Εκτέλεση περισσότερων του ενός kernel ταυτόχρονα. Η συσκευή μας έχει τη δυνατότητα να εκτελεί ταυτόχρονα περισσότερους του ενός kernels. Ομοίως εξαιτίας της εξάρτησης δεδομένων δεν μπορούμε να χρησιμοποιήσουμε αυτό το χαρακτηριστικό.

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7

Βιβλιογραφία

[1] Orchestrating the Phylogenetic Likelihood Function on Emerging Parallel Architectures. Stamatakis, Alexandros.

[2] Programming Massively Parallel Processors (David B. Kirk, Wen-mei W. Hwu)

[3] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol., 17:368–376, 1981.

[4] J. Shendure and H. Ji. Next-generation DNA sequencing. nature biotechnology, 26(10):1135–1145, 2008.

[5] F. Pratas, P. Trancoso, A. Stamatakis, L. Sousa: "Fine-grain parallelism using Multi-core, Cell/BE, and GPU systems: Accelerating the Phylogenetic Likelihood Function". Proceedings of ICPP 2009, accepted for publication, Vienna, Austria, September 2009. PDF

[6] J. Zhang, A. Stamatakis: "The Multi-Processor Scheduling Problem in Phylogenetics", accepted for publication at 11th IEEE HICOMB workshop (in conjunction with IPDPS 2012). PDF

[7] W. Pfeiffer, A. Stamatakis: "Hybrid MPI/Pthreads Parallelization of the RAxML Phylogenetics Code". Accepted for publication at HICOMB workshop, held in conjunction with IPDPS 2010, Atlanta, Georgia, April 2010. PDF

[8] Shucai Xiao and Wu-chun Feng, Department of Computer Science Virginia Tech "Inter-Block GPU Communication via Fast Barrier Synchronization"

[9] AWF Edwards, LL Cavalli-Sforza, VH Heywood, and J. McNeill. Phenetic and Phylogenetic classification. Systematics Association Publication, 6:67–76, 1963.

[10] Filip Blagojevic, Dimitrios S. Nikolopoulos, Alexandros Stamatakis, Christos D. Antonopoulos. "*RAxML-Cell: Parallel Phylogenetic Tree Inference on the Cell Broadband Engine*". Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS-07), Long Beach, CA, USA, March 2007. IEEE Computer Society Press, ISBN 1-4244-0910-1.

[11] RAxML Groups https://groups.google.com/forum/?fromgroups#!forum/raxml

[12] Nvidia Forums http://forums.nvidia.com.

[13] Exelixis Lab http://www.exelixis-lab.org/

[14] Wikipedia http://en.wikipedia.org/

[15] Developer tools/manuals/guides http://developer.nvidia.com/ CUDA C Programming Guide, cuda-memcheck ,cuda-gdb, Compute Visual Profiler User Guide

[16] Professional Programming Discussion http://stackoverflow.com/

Institutional Repository - Library & Information Centre - University of Thessaly
09/12/2017 03:35:36 EET - 137.108.70.7