

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Διπλωματική Εργασία

Μεταφορά και βελτιστοποίηση εφαρμογής
βιοπληροφορικής σε κάρτα γραφικών GPU,
χρησιμοποιώντας το μοντέλο παράλληλου
προγραμματισμού CUDA

Porting and optimization of a bioinformatics application
on a GPU using parallel programming model CUDA

Συγγραφέας: Θεοχαρίδης Κωνσταντίνος

Επιβλέπων καθηγητής_1: Αντωνόπουλος Χρήστος,
Επίκουρος Καθηγητής

Επιβλέπων καθηγητής_2: Μπέλλας Νικόλαος,
Αναπληρωτής Καθηγητής

Βόλος, Σεπτέμβριος 2012

Ευχαριστίες:

Να ευχαριστήσω τους καθηγητές μου, τον κύριο Αντωνόπουλο Χρήστο και τον κύριο Μπέλλα Νικόλαο για την συνεχή καθοδήγησή τους καθ' όλη την διάρκεια της υλοποίησης της διπλωματικής μου εργασίας. Επίσης, να ευχαριστήσω την οικογένεια μου και τους φίλους μου για την συνεχή στήριξή τους σε αυτή μου την προσπάθεια.

Περιεχόμενα

Κατάλογος Σχημάτων	5
Κατάλογος Συνοτομογραφιών	6
Περίληψη	7
Abstract	8
1 Εισαγωγή	9
1.1 Περιγραφή του προβλήματος και συμβολή της εργασίας	9
1.2 Διάρθρωση της διπλωματικής εργασίας	12
2 Η εφαρμογή FSA	13
2.1 Σκοπός εφαρμογής	13
2.2 Περιγραφή και συνοπτική ανάλυση μεθόδων	14
2.3 Στάδια παραλληλοποίησης	19
3 Το μοντέλο παράλληλου προγραμματισμού CUDA	21
3.1 Εισαγωγή	21
3.2 Βασικές έννοιες	22
3.3 Η αρχιτεκτονική Fermi της NVIDIA	23
4 Βελτιστοποίηση των συναρτήσεων Forward, Backward	30
4.1 Το Hidden Markov Model	30
4.2 Η συνάρτηση Forward	31
4.2.1 Φυσική σημασία υπολογισμών	31
4.2.2 Παραλληλισμός	32
4.2.3 Βελτιστοποίηση_1 – παραλληλία στοιχείων σε διαγώνιο	33
4.2.4 Βελτιστοποίηση_2 – χρήση shared memory	33
4.2.5 Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο	34
4.3 Η συνάρτηση Backward	35
4.3.1 Φυσική σημασία υπολογισμών	35
4.3.2 Παραλληλισμός	36
4.3.3 Βελτιστοποίηση_1 – παραλληλία στοιχείων σε διαγώνιο	37
4.3.4 Βελτιστοποίηση_2 – χρήση shared memory	37
4.3.5 Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο	38
4.4 Ο αλγόριθμος Forward – Backward	39
5 Βελτιστοποίηση της συνάρτησης Backward BaumWelch	40
5.1 Φυσική σημασία υπολογισμών	40
5.2 Παραλληλισμός	40
5.3 Βελτιστοποίηση	41

6	Μετρήσεις χρόνων σε GPU και σύγκριση με CPU	42
6.1	Σύστημα εκτέλεσης και αρχείο εισόδου	42
6.2	Γραφική παρουσίαση βελτιστοποιήσεων	43
6.3	Γραφική παρουσίαση speedup	51
7	Σύνοψη	52
7.1	Αναφορά κύριων σημείων	52
7.2	Μελλοντική επέκταση εφαρμογής	52
	Βιβλιογραφία	54

Κατάλογος Σχημάτων

2.1	Σύνοψη σταδίων λειτουργίας της FSA	14
2.2	Το HMM της FSA που χρησιμοποιείται για την σύγκριση των ζευγαριών	18
2.3	Γράφος POSET (αριστερή ευθυγράμμιση)	18
2.4	Γράφος POSET (δεξιά ευθυγράμμιση)	18
2.5	Οπτικοποίηση ενδεικτικής πρωτεϊνικής ευθυγράμμισης της FSA, με βάση το μέτρο της ακρίβειας, σε σχέση με μια πραγματική ευθυγράμμιση	19
2.6	Εξάρτηση δεδομένων σε “Forward”	20
2.7	Εξάρτηση δεδομένων σε “Backward”	20
3.1	Το μοντέλο CUDA	22
3.2	Σύνθεση CUDA core	24
3.3	Σύνθεση streaming multiprocessor	25
3.4	Διάγραμμα τμήματος αρχιτεκτονικής Fermi	26
3.5	Ιεραρχία μνήμης αρχιτεκτονικής Fermi	29
4.1	Το Hidden Markov Model	30
4.2.1	Εξάρτηση δεδομένων σε “Forward”	32
4.2.2	Εξάρτηση σε διαγωνίους από κάτω προς τα πάνω σε “Forward”	32
4.3.1	Εξάρτηση δεδομένων σε “Backward”	36
4.3.2	Εξάρτηση σε διαγωνίους από πάνω προς τα κάτω σε “Backward”	36
5.1	Η τεχνική του “sum_reduction”	41
6.1	Βελτιστοποίηση_1 (kernels)	43
6.2	Βελτιστοποίηση_1 (total exec. time)	44
6.3	Βελτιστοποίηση_2 (kernels)	45
6.4	Βελτιστοποίηση_2 (total exec. time)	46
6.5	Βελτιστοποίηση_3 (kernels)	47
6.6	Βελτιστοποίηση_3 (total exec. time)	48
6.7	Τελικά αποτελέσματα (kernels)	49
6.8	Τελικά αποτελέσματα (total exec. time)	50
6.9	Μεταβολή speedup (kernels)	51
7.1	Μελλοντική λύση (χρήση CUDA + MPI)	53

Κατάλογος Συντομογραφιών

FSA	<i>Fast Statistical Alignment</i>
GPU	<i>Graphics Processing Unit</i>
CPU	<i>Central Processing Unit</i>
OpenCL	<i>Open Computing Language</i>
CUDA	<i>Compute Unified Device Architecture</i>
HMM	<i>Hidden Markov Model</i>
POSET	<i>Partially Ordered Set</i>
GPGPU	<i>General Purpose Graphics Processing Unit</i>
SIMT	<i>Single Instruction Multiple Thread</i>
SM	<i>Streaming Multiprocessor</i>
MPI	<i>Message Passing Interface</i>

Περίληψη

Η εφαρμογή FSA, της οποίας η επιτάχυνση ευθυγράμμισης αποκλειστικά και μόνο πρωτεϊνικών ακολουθιών είναι και το αντικείμενο της μελέτης μας, είναι ένα πρόγραμμα ευθυγράμμισης βιολογικών ακολουθιών πρωτεϊνικής, RNA και DNA φύσης. Βασίζει την λειτουργία της σε ένα στατιστικό μοντέλο πιθανοτήτων και σημειώνει ικανοποιητική ακρίβεια στο αποτέλεσμα της, τόσο για ακολουθίες που δεν είναι ομόλογες, όσο και για μεγάλο πλήθος ακολουθιών, είτε μικρού είτε μεγάλου μεγέθους, ομόλογων ή μη.

Η στρατηγική επιλογής υποσυνόλου ζευγαριών μειώνει τις απαιτήσεις σε χρόνο εκτέλεσης και σε αποθηκευτικό χώρο, ενώ η στρατηγική εκμάθησης των εκάστοτε παραμέτρων του μοντέλου σύγκρισης του εκάστοτε ζευγαριού οδηγεί σε αποφυγή λαθών στην σύγκριση μη ομόλογων ακολουθιών. Επίσης, η FSA έχει επιλογή μέσω της οποίας μπορεί να χρησιμοποιηθεί συστάδα υπολογιστών για την επίτευξη παραλληλίας στον κώδικα προκειμένου να μειώσει σημαντικά τον χρόνο εκτέλεσής της.

Ωστόσο, οι υπολογισμοί που γίνονται είναι αρκετά χρονοβόροι για μεγάλου πλήθους ή μεγέθους πρωτεϊνικών ακολουθιών. Σκοπός της μελέτης που πραγματοποιείται είναι να μεταφερθεί η FSA πάνω στην κάρτα γραφικών (GPU) και να παρατηρηθεί κατά πόσο μπορεί να αξιοποιήσει την παραλληλία της, η οποία για την FSA στην παρούσα εργασία έγκειται στις μεθόδους σύγκρισης ανά ζευγάρι ακολουθιών. Με αυτόν τον τρόπο θα φανεί αν πράγματι η κάρτα μπορεί να βοηθήσει στην βελτιστοποίηση της απόδοσης της εφαρμογής.

Επίσης μέσω της εργασίας αυτής θα παρουσιαστούν διάφορες μορφές εξαρτήσεων δεδομένων και τεχνικές παραλληλοποίησης αυτών που μπορούν να εφαρμοστούν και από πολλές άλλες εφαρμογές. Πέρα από αυτό, θα μελετηθεί και κατά πόσο αποδοτικά μπορεί να ξεπεραστεί το πρόβλημα σειριακών τμημάτων κώδικα σε ένταξη παραλληλοποιήσιμων.

Ακόμη, θα παρουσιαστούν όλα τα βήματα και στάδια που υλοποιήθηκαν για την επίτευξη της τελικής απόδοσης. Με την μεταφορά μιας εφαρμογής πάνω στην κάρτα δεν επιτυγχάνεται κατευθείαν η μέγιστη απόδοση και τις περισσότερες φορές η αρχική απόδοση απέχει πολύ από την βέλτιστη. Θα αναλυθούν σταδιακά όλες οι βελτιστοποιήσεις που έγιναν για την επίτευξη της βέλτιστης απόδοσης.

Στο τέλος της διπλωματικής γίνεται σύγκριση των αποτελεσμάτων που προκύπτουν από την μεταφορά της FSA πάνω στην κάρτα γραφικών (GPU) σε σχέση με τα αντίστοιχα της αρχικής υλοποίησης στην κεντρική μονάδα επεξεργασίας (CPU). Το αρχείο εισόδου που χρησιμοποιήθηκε ήταν ενδεικτικού μεγέθους, αποτελούμενο από 73 πρωτεϊνικές ακολουθίες, κάθε μία μέσου μεγέθους 1452 αμινοξέων, για την εξαγωγή καλύτερων και πιο ολοκληρωμένων συμπερασμάτων. Πέραν αυτών, παρουσιάζεται και μια μελλοντική επέκταση της παρούσας εργασίας, η υλοποίηση της οποίας έχει τεράστιο ενδιαφέρον από πλευράς απόδοσης χρόνου εκτέλεσης.

Abstract

Application FSA (Fast Statistical Alignment) is used to align homologous and non-homologous protein, RNA and DNA biological sequences. The object of this undergraduate dissertation is to accelerate the alignment process exclusively among proteins. FSA depends its operation to a statistical probabilistic model and gives results with satisfactory accuracy. This is done not only for non-homologous sequences, but also for many sequences small or big length, homologous or not.

The strategy through which is chosen a subset of all sequence pairs, reduces the requirements about total execution time and capacity, whereas the machine learning strategy of the respective parameters of the comparison model of the respective pair, leads to a more accurate alignment during the comparison of non-homologous sequences. Furthermore, FSA has option for cluster computing in order to exploit parallelism achieving remarkably better total execution times.

However, the computations are very time-demanding for a big set or length of protein sequences. The aim is the porting and implementation of FSA on a GPU card using the parallel programming model CUDA in order to exploit the parallelism that is offered. This parallelism for this project is related to the comparison of each sequence pair. Via this process, can someone observe if there is any performance improvement that GPU can achieve.

What's more, this project presents various forms of data dependencies and some useful parallelism techniques that can be implemented by many other applications too. In addition to all these, this project studies how efficiently can be overcome the problem of the inevitable existence of sequential parts inside parts that can be parallelized.

Moreover, all the steps and stages that implemented for the achievement of the final performance will be presented. Initial port on GPU is not the best and many tries must be used in order to reach the peak performance. For this reason, all optimization stages will be presented with a sequential way to illustrate the whole optimization process from the initial port until the peak performance.

Finally, there are many graphs, one for each optimization stage that show the comparison to kernels and total execution times among CPU and GPU. The results that occurred are based on an input file that consists of 73 protein sequences, each with an average length of 1452 amino acid elements. The conclusions of this project are discussed and a crucial future improvement of FSA is planned.

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του προβλήματος και συμβολή της εργασίας

Η ευθυγράμμιση πρωτεϊνικών ακολουθιών είναι η προσπάθεια εντοπισμού συσχετιζόμενων, με εξελικτικό (evolutionarily), δομικό (structurally), ή λειτουργικό (functionally) τρόπο, θέσεων σε μια συλλογή αμινοξέων. Αν και το πρόβλημα της πρωτεϊνικής ευθυγράμμισης έχει μελετηθεί εδώ και αρκετές δεκαετίες, πολλές πρόσφατες έρευνες έχουν σημειώσει αξιόλογη πρόοδο βελτιώνοντας την ακρίβεια ή την κλιμάκωση πολλών προγραμμάτων ευθυγράμμισης. Σε ορισμένες περιπτώσεις, οι έρευνες αυτές έχουν συμβάλλει επιπρόσθετα και στην επέκταση των λειτουργιών των εκάστοτε προγραμμάτων ευθυγράμμισης.

Η πρωτεϊνική ευθυγράμμιση είναι μια κλασσική τεχνική στη Βιοπληροφορική για την οπτικοποίηση σχέσεων μεταξύ στοιχείων σε μια συλλογή εξελικτικά, δομικά ή λειτουργικά συσχετιζόμενων πρωτεϊνών. Έχοντας ως είσοδο ένα σύνολο αμινοξέων που απαρτίζουν τις πρωτεΐνες που πρόκειται να συγκριθούν, μια ευθυγράμμιση αναπαριστά τα στοιχεία για κάθε πρωτεΐνη σε μια απλή γραμμή, έτσι ώστε ισοδύναμα στοιχεία ανάμεσα σε δύο πρωτεΐνες να εμφανίζονται στην ίδια στήλη. Σε περίπτωση που δεν υπάρχει ισοδυναμία, εισάγεται μια παύλα ή ένα κενό (gap) όπως αποκαλείται, σε μία από τις δύο ή και στις δύο εκάστοτε θέσεις των στοιχείων των πρωτεϊνών που συνέβη αυτό, ανάλογα με τον αλγόριθμο σύγκρισης που χρησιμοποιείται.

Η ακριβής ερμηνεία και σημασία της προαναφερόμενης ισοδυναμίας εξαρτάται γενικά από το περιβάλλον μέσα στο οποίο μελετάται. Για έναν ερευνητή φυλογενετικών δένδρων, ισοδύναμα στοιχεία έχουν κοινή εξελικτική διαδρομή προγόνων. Για έναν ερευνητή της δομικής βιολογίας, ισοδύναμα στοιχεία αντιστοιχούν στην αποδοτική χρησιμοποίηση καθενός εκ των δύο στοιχείων σε θέσεις που ανήκουν σε ομόλογα πεδία ενός συνόλου πρωτεϊνών. Τέλος, για έναν ερευνητή της μοριακής βιολογίας, ισοδύναμα στοιχεία παίζουν παρόμοιους λειτουργικούς ρόλους στις πρωτεΐνες στις οποίες ανήκουν. Σε κάθε περίπτωση, μια ευθυγράμμιση παρέχει μια συνολική εικόνα των εκάστοτε κρυφών εξελικτικών, δομικών ή λειτουργικών περιορισμών που χαρακτηρίζουν μια οικογένεια πρωτεϊνών, με έναν συμπαγές και οπτικά διαισθητικό τρόπο.

Από τα προγράμματα ευθυγράμμισης που έχουν δημιουργηθεί, το πιο διαδεδομένο όλων, λόγω του ότι είναι εξακριβωμένα ελεγμένο για την ορθότητά του είναι το ClustalW που εμφανίστηκε το 1994. Ωστόσο, πάρα πολλά άλλα προγράμματα που έχουν βγει, επιτυγχάνουν καλύτερα αποτελέσματα σε ακρίβεια, ταχύτητα ή και στα δύο. Ο λόγος για τον οποίο δεν έχουν ακόμα καθιερωθεί ως προγράμματα ευρείας αποδοχής είναι εξαιτίας της δυσκολίας ελέγχου ορθότητας των προγραμμάτων

ευθυγράμμισης. Αυτό συμβαίνει διότι είναι αδύνατο να προβλεφθεί με απόλυτη βεβαιότητα η εξελικτική ιστορία αμινοξέων που αντιστοιχούν σε γονιδιώματα (πολύ μεγάλου μεγέθους ακολουθίες) υπαρχόντων οργανισμών. Οι συγκρίσεις επομένως των προγραμμάτων ευθυγράμμισης βασίζονται σε βάσεις δεδομένων δομικών πρωτεϊνικών ευθυγραμμίσεων. Κάθε τέτοια βάση δεδομένων όμως είναι ευάλωτη στην τροποποίησή της και επιπλέον ίσως δεν αντιπροσωπεύει τα ζητήματα προβλημάτων με τα οποία είναι πιο σχετικοί οι βιολόγοι. Το αποτέλεσμα είναι οι τελευταίοι να είναι αντιμέτωποι με μια πληθώρα προγραμμάτων ευθυγράμμισης που μπορούν να χρησιμοποιήσουν, αλλά συχνά δεν είναι ξεκάθαρο ποια προσέγγιση θα δώσει τα καλύτερα αποτελέσματα για τα καθημερινά προβλήματα που ψάχνουν να αντιμετωπίσουν. Αν σε όλα τα παραπάνω, λάβει κανείς υπόψη και τις διαφορετικές αρχιτεκτονικές πάνω στις οποίες υλοποιούνται και ελέγχονται τα παραπάνω προγράμματα, αλλά και την έλλειψη ανοιχτού λογισμικού, διαπιστώνει κανείς ότι είναι ακόμη πιο δύσκολο να δημιουργηθούν νέες ιδέες ή να συνδυαστούν οι ήδη υπάρχουσες για την παραγωγή προγραμμάτων με καθολικά συνεπή λειτουργία στις διάφορες βάσεις δεδομένων ελέγχου.

Λόγω των παραπάνω προβλημάτων λοιπόν, οι βιολόγοι επιλέγουν να χρησιμοποιούν το πρόγραμμα ClustalW. Το εν λόγω πρόγραμμα χρησιμοποιείται σε συνδυασμό με ένα λογισμικό το οποίο επιτρέπει χειροκίνητη τροποποίηση και έλεγχο των αποτελεσμάτων που προκύπτουν μετά την διαδικασία ευθυγράμμισης. Η λογική πίσω από το λογισμικό αυτό είναι ότι καταρτισμένοι ειδικοί είναι ικανοί να διορθώσουν τυχόν λανθασμένες ευθυγραμμίσεις, έχοντας απλά και μόνο μπροστά τους μια οπτική αναπαράσταση των δεδομένων εξόδου. Αυτή η προσπάθεια πολλές φορές αποδεικνύεται ότι είναι καλύτερη από την εύρεση κάποιου άλλου είδους προγράμματος / λογισμικού, το οποίο είναι απίθανο να βρει την σωστή ευθυγράμμιση για όλες τις περιπτώσεις δεδομένων ελέγχου.

Η ευθυγράμμιση πρωτεϊνικών ακολουθιών είναι επομένως ένα από τα πιο θεμελιώδη προβλήματα στην σύγκριση γονιδιωμάτων που παραμένει ακόμα ανεπίλυτο. Η σημασία επίλυσης της με έναν αποδοτικό τρόπο τόσο σε ορθότητα αποτελέσματος όσο και σε ταχύτητα είναι πολύ μεγάλη και άκρως επιτακτική αν σκεφτεί κανείς ότι η ευθυγράμμιση αποτελεί ένα από τα βασικότερα τμήματα όλων των βιολογικών εφαρμογών κάθε μορφής. Πάνω από 60 προγράμματα ευθυγράμμισης βρίσκονται στο Wikipedia και πολλά άλλα καινούρια δημοσιεύονται κάθε χρόνο. Παρόλα αυτά, πολλά και αρκετά δημοφιλή από αυτά, έχουν τις αδυναμίες τους. Δύο από τις βασικότερες είναι η ανικανότητα σύγκρισης μεγάλου, σε πλήθους ή όγκο, ακολουθιών εισόδου λόγω κακής χρήσης αποθηκευτικού χώρου από τους εκάστοτε αλγόριθμους, αλλά και η αδυναμία διαχωρισμού ακολουθιών μη ομόλογων μεταξύ τους με αποτέλεσμα την εσφαλμένη ευθυγράμμισή τους με το σύνολο των ομόλογων. Κάτι τέτοιο έχει σαν αποτέλεσμα η ανακριβής ευθυγράμμιση που προκύπτει να επιφέρει μεγάλα προβλήματα στα εκάστοτε τμήματα εφαρμογών που διαχειρίζονται τα αποτελέσματά της, με επακόλουθο την εσφαλμένη εκτέλεση της όλης εφαρμογής. Ένα χαρακτηριστικό παράδειγμα είναι η λανθασμένη πληροφορία που λαμβάνουν τα φυλογενετικά δένδρα για την ανάλυση των εξελικτικών σχέσεων που θέλουν να πραγματοποιήσουν.

Η παρούσα διπλωματική εργασία έχει ως στόχο την επιτάχυνση, όσον αφορά τον χρόνο εκτέλεσης, της εφαρμογής FSA (Fast Statistical Alignment). Η συγκεκριμένη εφαρμογή, η οποία δημοσιεύθηκε το 2009, είναι ένα πρόγραμμα πολλαπλής ευθυγράμμισης ακολουθιών. Οι ακολουθίες αυτές μπορεί να είναι πρωτεϊνικής, RNA ή DNA φύσης. Στην παρούσα εργασία θα επικεντρωθούμε στην μελέτη και επιτάχυνση σύγκρισης πρωτεϊνικών ακολουθιών και μόνο. Η εν λόγω εφαρμογή βελτιώνει την ακρίβεια των ήδη υπαρχόντων προσεγγίσεων σε βάσεις δεδομένων δομικά πρωτεϊνικών ευθυγραμμίσεων. Βασίζει την λειτουργία της σε ένα στατιστικό μοντέλο πιθανοτήτων και επιλύει τα δύο προβλήματα ευθυγράμμισης που αναφέρθηκαν παραπάνω.

Όλα τα προγράμματα ευθυγράμμισης έχουν μεγάλο υπολογιστικό φόρτο εργασίας κατά την διαδικασία σύγκρισης των ακολουθιών εισόδου. Συνήθως η σύγκριση αυτή έχει να κάνει με το εκάστοτε ζευγάρι και το άθροισμα των εκάστοτε χρόνων σύγκρισης ανά ζευγάρι οδηγεί σε μια αρκετά χρονοβόρα διαδικασία. Η εφαρμογή FSA έχει επιλογές που βοηθούν σημαντικά στην μείωση του συνολικού χρόνου εκτέλεσης. Τέτοιες είναι η επιλογή υποσυνόλου ζευγαριών προς σύγκριση και η ομοιόμορφη κατανομή ζευγαριών σε κόμβους συστάδας υπολογιστών. Κάτι τέτοιο είναι εφικτό, αφού με βάση τον αλγόριθμο που υιοθετείται από την FSA, κάθε σύγκριση ζευγαριού με κάθε άλλη είναι τελείως ανεξάρτητη. Παρόλα αυτά, για μεγάλο πλήθος ακολουθιών εισόδου μεσαίου μεγέθους ο χρόνος είναι αρκετός. Μετά το MAFFT, το FSA είναι το δεύτερο γρηγορότερο πρόγραμμα που υπάρχει, αλλά διαφέρει από το πρώτο κατά μία τάξη μεγέθους.

Η εργασία έχει ως σκοπό την μελέτη της απόδοσης στον χρόνο εκτέλεσης της εφαρμογής FSA, αν μεταφερθούν τα πιο χρονοβόρα τμήματα αυτής στην κάρτα γραφικών (GPU) και την σύγκριση των αποτελεσμάτων της με τα αντίστοιχα της κεντρικής μονάδας επεξεργασίας (CPU). Έχει παρατηρηθεί τα τελευταία χρόνια ότι η χρήση των καρτών γραφικών (GPU) σαν επιταχυντών, έχει συμβάλλει σημαντικά στην μείωση του χρόνου εκτέλεσης πολλών μεγάλων, υπολογιστικά χρονοβόρων και ζωτικής σημασίας εφαρμογών (μετεωρολογικές, βιολογικές, πολυμέσων κτλ). Οι δυνατότητες που προσφέρουν οι προαναφερόμενες κάρτες όσον αφορά τις μονάδες εκτέλεσης που μπορούν να τρέξουν παράλληλα πάνω σε αυτές είναι πολύ μεγαλύτερες σε σχέση με αυτές που παρέχει η κεντρική μονάδα επεξεργασίας (CPU) και αυξάνονται με την πάροδο του χρόνου. Έχουν αναπτυχθεί διάφορα μοντέλα παραλληλισμού για την αξιοποίησή τους, με πιο διαδεδομένα αυτό της CUDA και του OpenCL. Το μοντέλο CUDA ανήκει στην NVIDIA, ενώ αυτό του OpenCL ανήκει σε μια πληθώρα εταιριών κατασκευής υλικού, στις οποίες εντάσσεται και η NVIDIA. Στην παρούσα μελέτη μας, θα χρησιμοποιηθεί το μοντέλο της CUDA.

Αν και οι κάρτες γραφικών, μπορούν με αποδοτική χρήση αυτών, να επιτύχουν πολύ μεγάλη παραλληλοποίηση και άρα σημαντική μείωση του χρόνου εκτέλεσης, έχουν και αυτές τις αδυναμίες τους. Οι αδυναμίες αυτές έχουν να κάνουν με την επικοινωνία που γίνεται με την κάρτα όταν πραγματοποιείται μεταφορά δεδομένων προς και από αυτές στο τμήμα κώδικα που τρέχει σε CPU, αλλά και με τους περιορισμούς συγχρονισμού που επιφέρει η γεωμετρία των μονάδων παράλληλης εκτέλεσης. Επίσης πολλές εφαρμογές έχουν ένα σειριακό τμήμα κώδικα που δεν

μπορεί να παραλληλοποιηθεί και το οποίο αρκετές φορές αποτελεί αναπόσπαστο μέρος ενός τμήματος κώδικα που είναι παραλληλοποιήσιμος με αποτέλεσμα να πρέπει να επέλθει συγχρονισμός των μονάδων εκτέλεσης, κάτι που είναι ιδιαίτερα επίπονο χρονικά . Επομένως, το σημαντικό και το ουσιαστικό της όλης υπόθεσης είναι η διαφορά έναντι πλεονεκτημάτων και μειονεκτημάτων μεταφοράς κώδικα σε κάρτα γραφικών, να υπερτερεί όσο γίνεται περισσότερο προς την πλευρά των ικανοτήτων αυτής. Όσο μεγαλύτερη η διαφορά αυτή, τόσο μεγαλύτερη η προσφορά της κάρτας και τόσο μεγαλύτερη η επιτάχυνση της όλης εφαρμογής.

1.2 Διάρθρωση της διπλωματικής εργασίας

Στο *Κεφάλαιο 2* γίνεται περιγραφή της εφαρμογής FSA, ανάλυση των κύριων χαρακτηριστικών αυτής, καθώς και αναφορά των σταδίων που θα παραλληλοποιηθούν.

Στο *Κεφάλαιο 3* γίνεται περιγραφή του μοντέλου παράλληλου προγραμματισμού CUDA της NVIDIA που θα χρησιμοποιηθεί και ανάλυση της αρχιτεκτονικής Fermi.

Εν συνεχεία, στο *Κεφάλαιο 4* γίνεται αναφορά όλων των προσπαθειών που εφαρμόστηκαν για την παραλληλοποίηση των συναρτήσεων “Forward” και “Backward”.

Έπειτα, στο *Κεφάλαιο 5* γίνεται λόγος για την όλη μελέτη και τεχνική παραλληλοποίησης που αφορά την συνάρτηση “BackwardBaumWelch”.

Κατόπιν αυτών, στο *Κεφάλαιο 6* γίνεται η μέτρηση, η σύγκριση και ο σχολιασμός των αποτελεσμάτων που προκύπτουν από την GPU σε σχέση με αυτά της CPU.

Τέλος, στο *Κεφάλαιο 7* παρουσιάζονται τα κύρια συμπεράσματα που προέκυψαν από την παρούσα εργασία, καθώς και μελλοντικές επεκτάσεις αυτής.

Κεφάλαιο 2

Η εφαρμογή FSA

2.1 Σκοπός εφαρμογής

Η εφαρμογή FSA ψάχνει να μεγιστοποιήσει την αναμενόμενη ακρίβεια ευθυγράμμισης. Ψάχνει την ευθυγράμμιση που θα έχει την ελάχιστη αναμενόμενη απόσταση από την πραγματική ευθυγράμμιση των ακολουθιών εισόδου. Η πραγματική ευθυγράμμιση μεταχειρίζεται ως μια τυχαία μεταβλητή με την πιθανότητα κάθε πραγματικής ευθυγράμμισης να καθορίζεται μέσω ενός στατιστικού μοντέλου. Συγκεκριμένα, η χρησιμοποίηση μιας στατιστικά υποκινούμενης αντικειμενικής συνάρτησης για την προσέγγιση της ευθυγράμμισης με την αναμενόμενη ακρίβεια, επιτρέπει την οπτικοποίηση ευθυγραμμίσεων με βάση υπολογισμούς 5 διαφορετικών μέτρων ποιότητας ευθυγράμμισης. Αυτά τα μέτρα είναι η ακρίβεια (accuracy), η ευαισθησία (sensitivity), η ειδικότητα (specificity), η συνέπεια (consistency) και η βεβαιότητα (certainty). Βάσει αυτών, παρέχεται στους βιολόγους ένας ποσοτικός τρόπος σύγκρισης ποιότητας των εκάστοτε ευθυγραμμίσεων που προκύπτουν.

Η μέθοδος που υιοθετείται από την εφαρμογή FSA για την ευθυγράμμιση των ακολουθιών εισόδου είναι εύρωστη σε παραλλαγές παραμέτρων που έχουν να κάνουν με την εξέλιξη των οργανισμών. Φαινόμενα όπως αυτά των ακολουθιών με διαφορετικές εξελικτικές αποστάσεις και διαφορετικές βάσεις σύνθεσης επιλύονται επιτυχώς. Ενώ οι μέθοδοι των φυλογενετικών ευθυγραμμίσεων βασίζουν την λειτουργία τους σε μοντέλα δένδρων, μια υπολογιστικά χρονοβόρα διαδικασία, η μέθοδος της FSA χρησιμοποιεί ένα μοντέλο σύγκρισης ζευγαριών ακολουθιών, όπου η ανεξάρτητη φύση της εν λόγω σύγκρισης, του επιτρέπει υπολογισμό παραμέτρων, σχετικών με το εκάστοτε ζευγάρι, με αποτέλεσμα μοντέλο με μοντέλο σύγκρισης να μπορεί να διαφέρει.

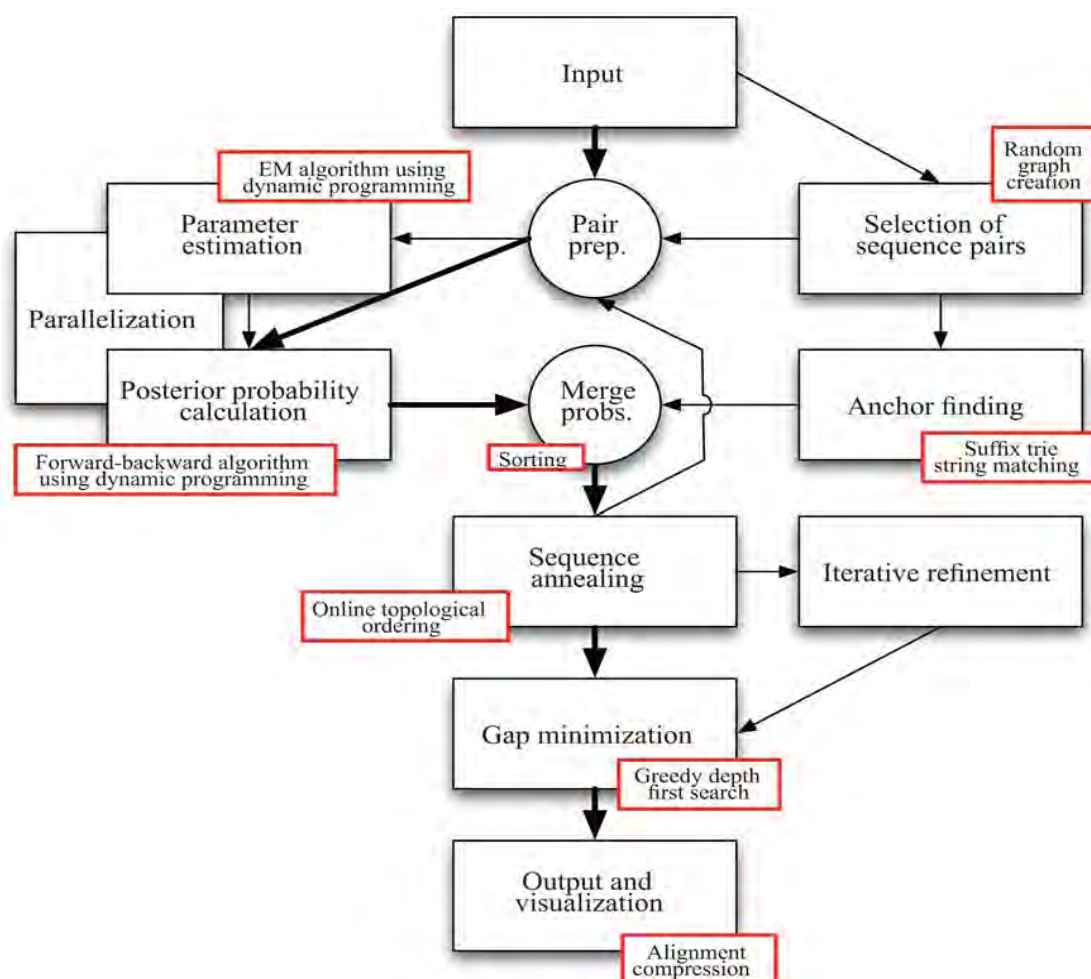
Πέραν αυτών, η εν λόγω εφαρμογή έχει ως στόχο την επιτυχή αντιμετώπιση ενός μεγάλου εύρους προβλημάτων ευθυγράμμισης που συναντούνται καθημερινά. Τα πραγματικά προβλήματα ευθυγράμμισης με τα οποία έρχονται αντιμέτωποι οι βιολόγοι καθημερινά έχουν να κάνουν με μεγάλου πλήθους ακολουθιών εισόδου, αλλά και ακολουθιών εισόδου μεγάλου μεγέθους, ομόλογων ή μη. Η κατάσταση γίνεται ακόμη πιο δύσκολη, όταν το προκύπτον αρχείο εισόδου αποτελεί συνδυασμό των δύο προαναφερόμενων ακολουθιών εισόδου. Πολλά προγράμματα έχουν σχεδιαστεί για την αντιμετώπιση ενός εκ των δύο προβλημάτων, ενώ άλλα καταφέρνουν μια επιτυχή αντιμετώπιση αυτών, σημειώνοντας όμως αρκετά αργούς χρόνους εκτέλεσης. Άλλα από αυτά πάλι ή σε ορισμένες περιπτώσεις και τα προηγούμενα, αποτυγχάνουν να διαχωρίσουν την ύπαρξη μη ομόλογων ακολουθιών μέσα στο σύνολο των ομόλογων, με επακόλουθο την εσφαλμένη ευθυγράμμισή τους. Στην πρώτη περίπτωση, η στρατηγική επιλογής υποσυνόλου

ζευγαριών προς σύγκριση, αλλά και η στρατηγική ομοιόμορφης κατανομής ζευγαριών σύγκρισης σε κόμβους συστάδας υπολογιστών της FSA, αντιμετωπίζουν επιτυχώς το πρόβλημα ταχύτητας εκτέλεσης σημειώνοντας μικρή μείωση της ακρίβειας ευθυγράμμισης. Αντίθετα, στη δεύτερη περίπτωση, η στρατηγική μηχανικής μάθησης παραμέτρων σύγκρισης που χρησιμοποιείται για το μοντέλο σύγκρισης του εκάστοτε ζευγαριού, συμβάλλει σημαντικά στον επιτυχή διαχωρισμό ομόλογων με μη ομόλογων ακολουθιών.

Τέλος, η εφαρμογή FSA δημιουργήθηκε έτσι ώστε να έχει έναν διατμηματικό χαρακτήρα ανάπτυξης και δόμησης. Αυτό σημαίνει ότι ενδεχόμενες μελλοντικές τροποποιήσεις σε διάφορες πτυχές της ευθυγράμμισης δεν θα επηρεάσουν τον κώδικα που αφορά άλλα στάδια ή ζητήματα της όλης διαδικασίας ευθυγράμμισης.

2.2 Περιγραφή και συνοπτική ανάλυση μεθόδων

Ο συνολικός τρόπος λειτουργίας της FSA συνοψίζεται στο παρακάτω γράφημα (Σχήμα 2.1):



Σχήμα 2.1: Σύνοψη σταδίων λειτουργίας της FSA

, όπου σε κόκκινο πλαίσιο αναγράφεται η μέθοδος που χρησιμοποιείται από το αντίστοιχο στάδιο λειτουργίας του προγράμματος. Επίσης η διαδρομή που σχηματίζεται αποκλειστικά και μόνο από τα έντονα μαύρα βέλη υποδεικνύει τον συνδυασμό σταδίων που αντιστοιχεί στον απλούστερο τρόπο εκτέλεσης του αλγορίθμου.

Η περιγραφή και η συνοπτική ανάλυση των σταδίων λειτουργίας που θα ακολουθήσει δεν περιλαμβάνει το στάδιο "*Anchor finding*" και την μέθοδό του "*Suffix trie string matching*", καθώς το συγκεκριμένο στάδιο έχει να κάνει με ενέργειες που εφαρμόζονται σε νουκλεοτίδια (RNA, DNA), ενώ η μελέτη μας όπως έχει αναφερθεί και νωρίτερα επικεντρώνεται αποκλειστικά και μόνο στην διαχείριση αμινοξέων (πρωτεΐνες). Πέραν αυτού, δεν συμπεριλαμβάνεται επίσης και το στάδιο "*Parallelization*", μέσω του οποίου υπάρχει επιλογή για ομοιόμορφη κατανομή των ζευγαριών σύγκρισης σε συστάδα υπολογιστών. Η αξιοποίηση της λογικής, που έχει η εν λόγω μορφή παραλληλοποίησης, από την κάρτα γραφικών, αποτελεί τον σημαντικότερο μελλοντικό στόχο της παρούσας εργασίας.

Input: Η είσοδος απαρτίζεται από τις πρωτεϊνικές ακολουθίες που πρόκειται να συγκριθούν. Αυτές οι ακολουθίες κανονικά πρέπει να είναι ομόλογες, αν και η FSA χειρίζεται με μεγάλη επιτυχία και την ύπαρξη μη ομόλογων μέσα σε αυτές. Το αρχείο εισόδου πρέπει να είναι μορφής FASTA.

Selection of sequence pairs (optional stage): Αν δεν καθοριστεί κάποιος αλγόριθμος επιλογής ζευγαριών, ο αριθμός των πιθανών ζευγαριών σύγκρισης είναι τετραγωνικός σε σχέση με το σύνολο των ακολουθιών που πρόκειται να συγκριθούν. Κάτι τέτοιο έχει σαν αποτέλεσμα πολύ μεγάλους χρόνους εκτέλεσης της διαδικασίας ευθυγράμμισης. Η εφαρμογή FSA ξεπερνά αυτό το πρόβλημα μειώνοντας τον αριθμό των ζευγαριών που πρόκειται να συγκριθούν. Αυτό επιτυγχάνεται χρησιμοποιώντας μια τυχαία προσέγγιση επιλογής γράφου εμπνευσμένη από την θεωρία τυχαίων γράφων "*Erdos-Renyi*". Η εν λόγω προσέγγιση μειώνει δραματικά το υπολογιστικό κόστος της διαδικασίας ευθυγράμμισης. Για την αξιοποίησή της απαιτείται το όρισμα **--fast** κατά την εκτέλεση του προγράμματος.

Pair prep.: Επιλέγεται το εκάστοτε ζευγάρι σύγκρισης.

Parameter estimation (optional stage): Πολλές φορές υπάρχει η ανάγκη για τον διαχωρισμό των απαιτήσεων κάθε ζευγαριού σύγκρισης. Για παράδειγμα αν τα κενά (gaps) σε μια πρωτεΐνη μετά τη φάση ενός σταδίου ευθυγράμμισής της με μια άλλη είναι πολύ περισσότερα, έναντι ενός άλλου ζευγαριού πρωτεϊνών, τότε η υιοθέτηση των ίδιων παραμέτρων μοντέλου σύγκρισης και για τα 2 ζευγάρια αποδεικνύεται συνήθως εσφαλμένη. Η εφαρμογή FSA ξεπερνά αυτό το πρόβλημα με την χρήση μιας στρατηγικής μηχανικής μάθησης των παραμέτρων σύγκρισης του εκάστοτε ζευγαριού, που έχει σαν αποτέλεσμα την δημιουργία ενός ξεχωριστού μοντέλου για κάθε ζευγάρι σύγκρισης. Αυτό επιτυγχάνεται μέσω ενός αλγορίθμου αναμενόμενης μεγιστοποίησης ("*Expectation Maximization algorithm*"), ο οποίος βασίζει την λειτουργία του στην τεχνική του δυναμικού προγραμματισμού (

“dynamic programming”). Μέσω αυτής της διαδικασίας υπολογίζονται οι πιθανότητες μετάβασης (transition, gap, indel) και οι πιθανότητες εκπομπής (emission, substitution, mismatch) του εκάστοτε μοντέλου σύγκρισης. Το εν λόγω μοντέλο είναι το γνωστό μοντέλο Markov κρυμμένων μεταβλητών (Hidden Markov Model) τριών ή πέντε καταστάσεων και εικονίζεται στο Σχήμα 2.2. Στην δική μας προσέγγιση χρησιμοποιείται αυτό των τριών καταστάσεων και για αυτό απαιτείται η χρήση του ορίσματος **--noindel2** κατά την εκτέλεση του προγράμματος. Να σημειωθεί ότι ο υπολογισμός των παραπάνω πιθανοτήτων γίνεται μέσω των συναρτήσεων *“Forward”* και *“BackwardBaumWelch”*, όπου και οι δύο βασίζουν την λειτουργία τους στην τεχνική του δυναμικού προγραμματισμού (*“dynamic programming”*).

Posterior probability calculation: Με την χρησιμοποίηση των κατάλληλων παραμέτρων του HMM (Hidden Markov Model) τριών καταστάσεων που προκύπτουν από το στάδιο *“Parameter Estimation”* για κάθε ζευγάρι σύγκρισης, υπολογίζονται οι μεταγενέστερες πιθανότητες (posterior probabilities) των εκάστοτε μεμονωμένων χαρακτήρων (αμινοξέα) που συμπεριλαμβάνονται στις πρωτεΐνες του αντίστοιχου ζευγαριού μέσω της προσέγγισης που βασίζεται στην απόσταση (distance-based approach). Οι πιθανότητες αυτές δείχνουν κατά πόσο δύο μεμονωμένοι χαρακτήρες ενός ζευγαριού είναι ισοδύναμοι και άρα πρέπει να ευθυγραμμιστούν ή δεν είναι ισοδύναμοι και συνεπώς δεν πρέπει να ευθυγραμμιστούν. Ο υπολογισμός των εν λόγω πιθανοτήτων γίνεται μέσω του γνωστού αλγορίθμου *“Forward - Backward”*, ο οποίος βασίζει και αυτός την λειτουργία του στην τεχνική του δυναμικού προγραμματισμού (*“dynamic programming”*). Οι συγκεκριμένες πιθανότητες συμπεριλαμβάνονται σε ένα αρχείο *.probs* που προκύπτει μετά την εκτέλεση του προγράμματος με την προϋπόθεση να περαστεί πριν την εκτέλεσή του το όρισμα **--gui**.

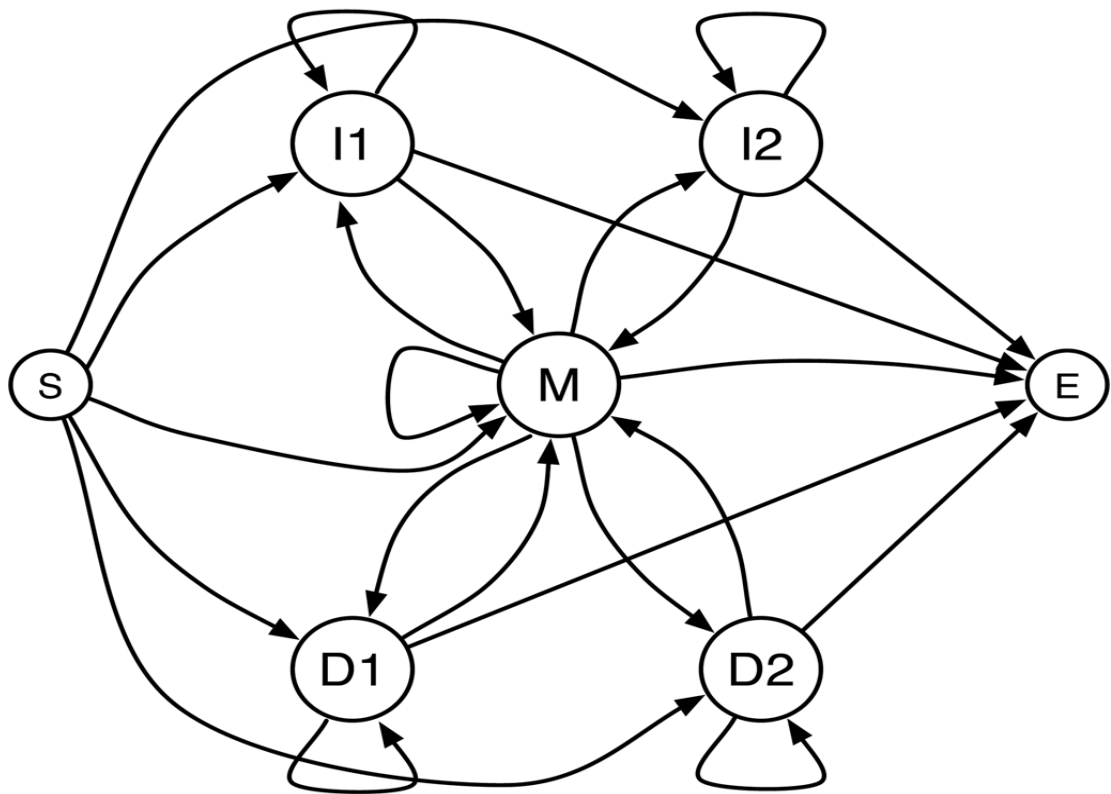
Merge probs.: Μετά την ολοκλήρωση του προαναφερόμενου σταδίου για όλα τα πιθανά ζευγάρια σύγκρισης, όλες οι μεταγενέστερες πιθανότητες (posterior probabilities) που προκύπτουν ταξινομούνται σύμφωνα με μια συνάρτηση βάρους στην οποία εφαρμόζεται ένας αλγόριθμος μορφής steepest-ascent.

Sequence annealing: Η προσέγγιση που βασίζεται στην απόσταση (distance-based approach) συμβάλλει σε μια πολλαπλή ευθυγράμμιση, χρησιμοποιώντας μόνο ανά ζευγάρι εκτιμήσεις του κατά πόσο ομόλογες είναι οι πρωτεΐνες που το απαρτίζουν. Αυτό γίνεται εφικτό μέσω της τεχνικής *“sequence annealing”*, η οποία κατασκευάζει πολλαπλές ευθυγραμμίσεις μέσω συγκρίσεις ζευγαριών. Η εν λόγω προσέγγιση ψάχνει να μεγιστοποιήσει την αναμενόμενη ακρίβεια ευθυγράμμισης κάνοντας χρήση ενός άπληστου (greedy) αλγορίθμου μορφής steepest-ascent. Η τεχνική *“sequence annealing”* αναζητά να βρει μια ευθυγράμμιση που θα έχει την ελάχιστη αναμενόμενη απόσταση από την πραγματική, όπου η πραγματική ευθυγράμμιση μεταχειρίζεται ως μια τυχαία μεταβλητή με την πιθανότητα κάθε πραγματικής ευθυγράμμισης να καθορίζεται μέσω ενός στατιστικού μοντέλου. Η ευθυγράμμιση που κατέχει την ελάχιστη αναμενόμενη απόσταση από την πραγματική, είναι και η ευθυγράμμιση με την μέγιστη αναμενόμενη ακρίβεια.

Iterative refinement (optional stage): Εξαιτίας της άπληστης φύσης της τεχνικής “sequence annealing”, η αναμενόμενη ακρίβεια ευθυγράμμισης η οποία υπολογίζεται μέσω αυτής, αποτελεί μια τοπικά βέλτιστη λύση, που τις περισσότερες φορές μπορεί να βελτιωθεί περαιτέρω για την επίτευξη της ολικά βέλτιστης λύσης (ευθυγράμμιση με την καλύτερη δυνατή ακρίβεια). Η στρατηγική συνεπώς της επαναληπτικής βελτίωσης (“iterative refinement”) προσπαθεί να επιτύχει αυτό ακριβώς το πράγμα. Όμως, το συγκεκριμένο στάδιο αποτελεί μια πολύ επίπονη χρονικά διαδικασία και συνήθως αποφεύγεται για την επεξεργασία μεγάλου μεγέθους προβλημάτων. Για την μη αξιοποίησή του, απαιτείται το πέρασμα του ορίσματος --**refinement 0** κατά την εκτέλεση του προγράμματος.

Gap minimization: Σε αυτό το στάδιο η FSA προσπαθεί να μειώσει όσο γίνεται τον αριθμό των φωνών που ένα κενό δημιουργείται. Να εξάγει δηλαδή μια καθολική ευθυγράμμιση (global alignment) με τον ελάχιστο αριθμό gap openings. Οι ευθυγραμμίσεις που φαίνονται στο Σχήμα 2.3 και Σχήμα 2.4 αντίστοιχα, είναι ισοδύναμες καθώς αντιστοιχούν στις ίδιες δηλώσεις ομολογίας, ωστόσο αυτή η οποία τελικώς εξάγεται είναι αυτή στο Σχήμα 2.4. Ουσιαστικά η λειτουργία αυτού του σταδίου είναι η αντίστοιχη κάθε φορά μετάβαση από Σχήμα 2.3 σε Σχήμα 2.4. Να σημειωθεί ότι δύο ισοδύναμες ευθυγραμμίσεις μπορούν να αναπαριστώνται από τον ίδιο γράφο μορφής POSET, ο οποίος αντιστοιχεί στην πιο χρονοβόρα διαδικασία εύρεσης των gap openings. Η σημαντικότητα του εν λόγω σταδίου έγκειται στις απαιτήσεις που έχει η ανάλυση σύγκρισης γονιδιωμάτων. Η τελευταία προϋποθέτει την χρήση μιας ευθυγράμμισης για την εξαγωγή εξελικτικών παραμέτρων, όπως είναι η συχνότητα των gap openings σε τμήματα ακολουθιών (συνήθως αρκετά ή πολύ μεγάλου μεγέθους). Κάτι τέτοιο θέλει να το κάνει με τον πιο οικονομικό τρόπο από άποψης χρόνου, συνεπώς προτιμάται η ευθυγράμμιση που επιφέρει το στάδιο “Gap minimization”.

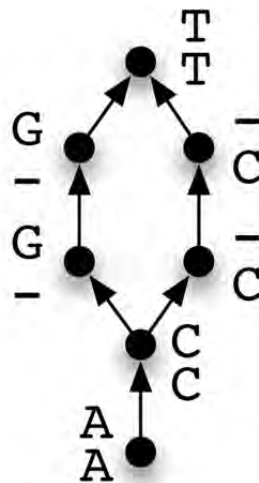
Output and visualization: Η έξοδος αντιστοιχεί σε μια καθολική ευθυγράμμιση (global alignment) των ακολουθιών εισόδου, η οποία είναι ένα τοπικό βέλτιστο της αναμενόμενης ακρίβειας που επιτυγχάνεται μέσω του στατιστικού μοντέλου που χρησιμοποιείται. Η οπτικοποίηση της ποιότητάς της μπορεί να γίνει μέσω ενός βοηθητικού γραφικού περιβάλλοντος αναπαράστασης (GUI) που συμπεριλαμβάνει η εφαρμογή. Η ποιότητα μιας ευθυγράμμισης εκτιμάται σύμφωνα με πέντε διαφορετικά μέτρα, τα οποία είναι η ακρίβεια (accuracy), η ευαισθησία (sensitivity), η ειδικότητα (specificity), η συνέπεια (consistency) και η βεβαιότητα (certainty). Στο Σχήμα 2.5 μπορεί να φανεί μια τέτοια οπτικοποίηση ανάμεσα σε ένα ενδεικτικό σύνολο ακολουθιών προς σύγκριση. Τέλος, να αναφερθεί ότι το αρχείο εξόδου είναι συνήθως της μορφής multi-FASTA, αν και η μορφή Stockholm μπορεί εξίσου να χρησιμοποιηθεί.



Σχήμα 2.2: Το HMM της FSA που χρησιμοποιείται για την σύγκριση των ζευγαριών



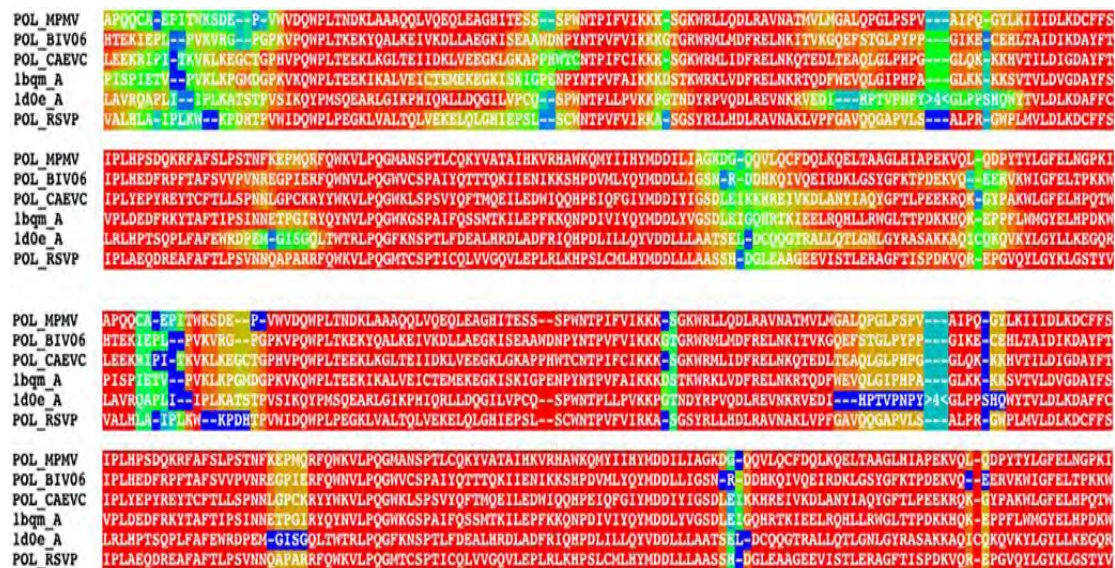
Σχήμα 2.3



Γράφος POSET



Σχήμα 2.4



Σχήμα 2.5: Οπτικοποίηση ενδεικτικής πρωτεϊνικής ευθυγράμμισης της FSA, με βάση το μέτρο της ακρίβειας, σε σχέση με μια πραγματική ευθυγράμμιση

Να σημειωθεί ότι στο παραπάνω σχήμα, λόγω μεγάλου μεγέθους των ακολουθιών και λόγω έλλειψης χώρου, η πρώτη ευθυγράμμιση (της FSA) αντιστοιχεί σε συγχώνευση των δύο πρώτων σχημάτων και η δεύτερη ευθυγράμμιση (πραγματική) αντιστοιχεί σε συγχώνευση των δύο τελευταίων σχημάτων.

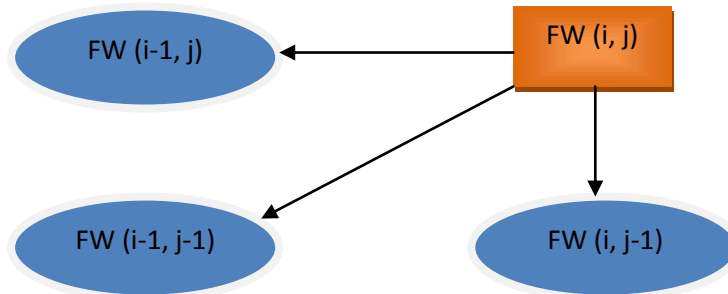
2.3 Στάδια παραλληλοποίησης

Μετά από το profiling της εφαρμογής FSA μέσα από το πρόγραμμα VTune, βγήκε το συμπέρασμα ότι τα πιο χρονοβόρα τμήματα αυτής έχουν να κάνουν με τις συναρτήσεις που απαρτίζουν τα στάδια “Parameter estimation” και “Posterior probability calculation”. Αξίζει να σημειωθεί ότι το στάδιο του “Sequence annealing” είναι ακολουθιακό και ερευνάται η δόμησή του με τέτοιο τρόπο έτσι ώστε να μπορεί να εισαχθεί με κάποιο τρόπο παραλληλισμός μέσα σε αυτό.

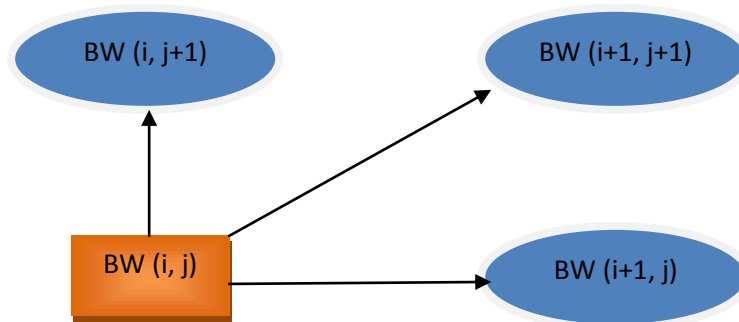
Πιο συγκεκριμένα, οι συναρτήσεις “Forward” και “BackwardBaumWelch” αντιστοιχούν στο στάδιο “Parameter estimation”, ενώ οι συναρτήσεις “Forward” και “Backward” αντιστοιχούν στο στάδιο “Posterior probability calculation”.

Ο παραλληλισμός σε “Forward” και “Backward” έγκειται στις διαγωνίους ενός διδιάστατου πίνακα διαστάσεων ανάλογων των μεγεθών των ακολουθιών που απαρτίζουν το εκάστοτε ζευγάρι σύγκρισης. Ο υπολογισμός κάθε επόμενης διαγωνίου είναι εφικτός μόνο όταν ολοκληρωθεί ο υπολογισμός όλων των στοιχείων / δεδομένων που ανήκουν στην προηγούμενη διαγώνιο. Στη “Forward” κάθε στοιχείο της διαγωνίου εξαρτάται από το αριστερά του, το κάτω του και το

κάτω αριστερά του (Σχήμα 2.6), ενώ στη “Backward” κάθε στοιχείο της διαγωνίου εξαρτάται από το δεξιά του, το πάνω του και το πάνω δεξιά του (Σχήμα 2.7). Οι διαγώνιοι σε “Forward” σαρώνουν τον πίνακα των δεδομένων από κάτω προς τα πάνω, ενώ οι διαγώνιοι σε “Backward” σαρώνουν τον πίνακα των δεδομένων από πάνω προς τα κάτω.



Σχήμα 2.6: Εξάρτηση δεδομένων σε “Forward”



Σχήμα 2.7: Εξάρτηση δεδομένων σε “Backward”

Οι παραπάνω μορφές εξάρτησης δεδομένων, βάση των οποίων ο υπολογισμός του εκάστοτε πίνακα εκτείνεται κατά μήκος των διαγωνίων αυτού, ανήκουν στην ομάδα των εφαρμογών “wavefront data dependencies & computation” και είναι πολύ συνηθισμένες στον πραγματικό κόσμο.

Ο παραλληλισμός σε “BackwardBaumWelch” έγκειται στην ανεξαρτησία ορισμένων εντολών που εκτελούνται από κάθε στοιχείο ξεχωριστά και στην εφαρμογή αποδοτικών αλγορίθμων πρόσθεσης στα αποτελέσματα που προκύπτουν από την εκτέλεση των προαναφερόμενων εντολών. Για την παράλληλη υλοποίηση των αθροισμάτων εφαρμόζεται η πολύ διαδεδομένη τεχνική του “reduction”, ενώ με τον όρο στοιχείο, αναφέρεται κάθε στοιχείο που ανήκει σε δισδιάστατο πίνακα διαστάσεων ανάλογων των μεγεθών των ακολουθιών που απαρτίζουν το εκάστοτε ζευγάρι σύγκρισης. Αξίζει να σημειωθεί ότι τμήμα εντολών που εκτελείται από κάθε στοιχείο έχει ακολουθιακή και αναδρομική εξάρτηση με τμήμα εντολών άλλων στοιχείων και επιτάσσει την διαχείρισή του στην πλευρά της CPU για την αποφυγή πολύ συχνού και επίπονου χρονικά συγχρονισμού μεταξύ των στοιχείων στη GPU. Η εν λόγω τεχνική μπορεί να αποδειχθεί πολύ χρήσιμη σε άλλες εφαρμογές όμοιων προβλημάτων παραλληλοποίησης.

Κεφάλαιο 3

Το μοντέλο παράλληλου προγραμματισμού CUDA

3.1 Εισαγωγή

Οι κάρτες γραφικών αρχικά σχεδιάστηκαν σαν συσκευές για την επεξεργασία και την οπτικοποίηση γραφικών του υπολογιστή. Όταν το 1990 το υλικό (hardware) έγινε προγραμματίσιμο, η εταιρία κατασκευής υλικού NVIDIA εκμεταλλεύτηκε αυτή την ευκαιρία και εφάρμοσε τη νέα αυτή ικανότητα του hardware πάνω στις κάρτες γραφικών. Στη συνέχεια, το 1999, επινόησε τον όρο GPU (Graphics Processing Unit) και μια νέα εποχή, αποκαλούμενη ως GPGPU (General Purpose GPU), μόλις είχε ξεκινήσει. Η πρώτη επίσημη λύση της εν λόγω εταιρίας για υποστήριξη κώδικα εκτέλεσης σε GPUs δημοσιεύθηκε το 2006, όπου έγινε και η αποκάλυψη του μοντέλου παράλληλου προγραμματισμού CUDA (Compute Unified Device Architecture).

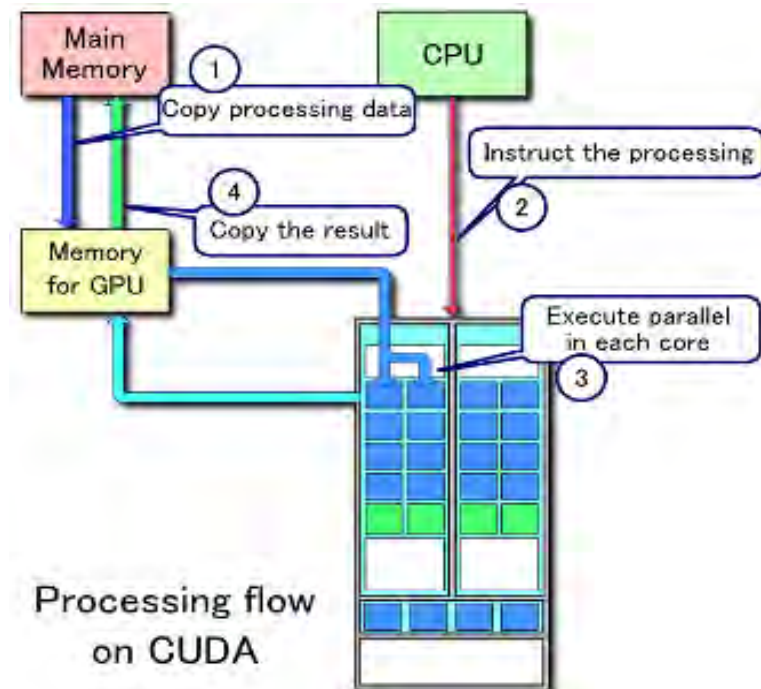
Με την χρήση των δυνατοτήτων παράλληλης εκτέλεσης που προσφέρει η αρχιτεκτονική των GPUs, το εν λόγω μοντέλο μπορεί να αυξήσει πολύ έντονα την απόδοση των υπολογισμών που πραγματοποιούνται στην κάρτα γραφικών, σημειώνοντας σημαντικά μικρότερους χρόνους εκτέλεσης αυτών. Σε αυτό συμβάλλει η επικοινωνία που γίνεται ανάμεσα σε GPU και CPU (Central Processing Unit), έτσι ώστε να ανατεθεί η επίπονη χρονικά επεξεργασία των δεδομένων από την CPU στην κάρτα γραφικών. Μέσω αυτού, παρατηρείται και αποδεικνύεται ότι είναι προτιμότερη η ύπαρξη πολλών παράλληλων μονάδων εκτέλεσης χαμηλής ταχύτητας επεξεργασίας (GPU threads) από εκείνη των ελάχιστων παράλληλων μονάδων εκτέλεσης υψηλής ταχύτητας επεξεργασίας (CPU threads).

Η συγκεκριμένη παρατήρηση έχει φανεί πολύ χρήσιμη στο πέρασμα των χρόνων και πλέον στις μέρες μας το μοντέλο παράλληλου προγραμματισμού CUDA χρησιμοποιείται ευρέως για την επιτάχυνση των υπολογισμών πολλών και πολύ σημαντικών πεδίων εφαρμογών. Κάποια από αυτά τα πεδία είναι:

- ✓ Βιολογία
- ✓ Μετεωρολογία
- ✓ Πολυμέσα
- ✓ Οικονομία
- ✓ Ενέργεια

Η πρώτη σειρά καρτών γραφικών της NVIDIA, η οποία σχεδιάστηκε για το μοντέλο CUDA, ήταν αυτή των G8x. Έκτοτε, όλες οι νεότερες σειρές καρτών που βγάζει η εν λόγω εταιρία, συμπεριλαμβάνει αυτές των GeForce, Tesla, Quadro. Στο Σχήμα 3.1

αναπαρίσταται το βασικό μοτίβο εκτέλεσης που χαρακτηρίζει μια εφαρμογή που κάνει χρήση του μοντέλου CUDA.



Σχήμα 3.1: Το μοντέλο CUDA

Συνοπτικά η διαδικασία που ακολουθείται είναι η εξής:

- ✓ Τα δεδομένα προς επεξεργασία μεταφέρονται από την *RAM* στην καθολική μνήμη (*global memory*) της GPU
- ✓ Η CPU παραδίδει την εκτέλεση στην GPU
- ✓ Η GPU εκτελεί τον κώδικα παράλληλα
- ✓ Το προκύπτον αποτέλεσμα επιστρέφει στην *RAM*

3.2 Βασικές έννοιες

Στο μοντέλο CUDA υφίστανται οι παρακάτω βασικές έννοιες:

Με τον όρο *threads* αναφερόμαστε στις μονάδες παράλληλης εκτέλεσης / νήματα.

Με τον όρο *blocks* αναφερόμαστε σε ένα σύνολο από *threads*.

Με τον όρο *grid* κάνουμε λόγο για ένα σύνολο από *blocks*.

Ο αριθμός των διαστάσεων ενός *grid* μπορεί να είναι μέχρι τρεις και μας δείχνει τον αριθμό των *blocks* που δεσμεύονται για κάθε διάσταση.

Επίσης, ο αριθμός των διαστάσεων ενός block μπορεί να είναι και αυτός μέχρι τρεις και μας δείχνει τον αριθμό των threads που δεσμεύονται για κάθε διάσταση.

Τα threads που ανήκουν στο ίδιο block μπορούν να μοιράσουν δεδομένα μεταξύ τους μέσω μια γρήγορης κοινής μνήμης που υπάρχει ανάμεσά τους (*shared memory*) και για την οποία θα γίνει λόγος στη συνέχεια. Επίσης για την ίδια κατηγορία threads, υπάρχουν κατάλληλα φράγματα (*barriers*) μέσω των οποίων είναι εφικτός ο συγχρονισμός της παράλληλης εκτέλεσής τους. Αντίθετα, threads μεταξύ διαφορετικών blocks, μπορούν να ανταλλάξουν δεδομένα μόνο μέσω μιας αργής κοινής μνήμης (*global memory*) και για την οποία θα γίνει λόγος στη συνέχεια. Για τον συγχρονισμό της δεύτερης κατηγορίας threads δεν υπάρχουν έτοιμες ρουτίνες διαχείρισης που να προσφέρει το μοντέλο και απαιτείται παρέμβαση του προγραμματιστή για την υλοποίηση δικών του ρουτινών συγχρονισμού που θα του αποδώσουν το ζητούμενο αποτέλεσμα.

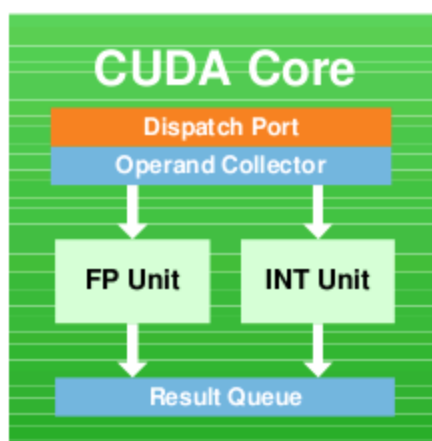
Το μοντέλο CUDA χρησιμοποιεί μια επέκταση των γλωσσών C και C++, επιτρέποντας με αυτόν τον τρόπο τον προγραμματισμό συναρτήσεων που μπορούν να εκτελεστούν στη GPU. Αυτές οι συναρτήσεις αποκαλούνται *kernels* και καλούνται μέσω μιας μορφής διαμόρφωσης εκτέλεσης που το ελάχιστο που μπορεί να ορίσει είναι ο αριθμός των blocks μέσα στο grid (*B*) και ο αριθμός των threads μέσα στα blocks (*T*). Η εν λόγω μορφή διαμόρφωσης εκτέλεσης περιγράφεται σαν το ζευγάρι $\langle\langle\langle B, T \rangle\rangle\rangle$. Μετά την κλήση, ο kernel εκτελείται παράλληλα με το κάθε thread να τρέχει τον ίδιο κώδικα kernel πάνω σε διαφορετικά δεδομένα. Αυτός ο τρόπος εκτέλεσης είναι γνωστός ως SIMT (*single-instruction multiple-thread*) και θα αναλυθεί στη συνέχεια. Τα threads και τα blocks διαχωρίζονται μεταξύ τους μέσω κατάλληλων προσδιοριστικών που προσφέρει το CUDA runtime και τα οποία είναι τα *threadIdx* και *blockIdx* αντίστοιχα. Ερμηνεύονται ως ενσωματωμένες μεταβλητές του kernel και ο προσδιορισμός των τριών διατάσεων τους γίνεται και για τα δύο με *.x*, *.y*, *.z*.

3.3 Η αρχιτεκτονική Fermi της NVIDIA

Είναι πολύ σημαντικό να αντιληφθεί κάποιος τον τρόπο με τον οποίο εξελίσσεται η λειτουργία της GPU από το χαμηλότερο έως το υψηλότερο επίπεδό της. Όσο καλύτερα τον κατανοήσει, τόσο καλύτερα θα μπορέσει να αξιοποιήσει και να προγραμματίσει το hardware της εκάστοτε κάρτας γραφικών που διαθέτει. Η συζήτηση που θα ακολουθήσει θα επικεντρωθεί στην αρχιτεκτονική Fermi και πιο συγκεκριμένα στο μοντέλο GeForce GTX480, το οποίο είναι και το μοντέλο που χρησιμοποιείται από την εφαρμογή FSA για τις μετρήσεις και τις παρατηρήσεις της παρούσας εργασίας.

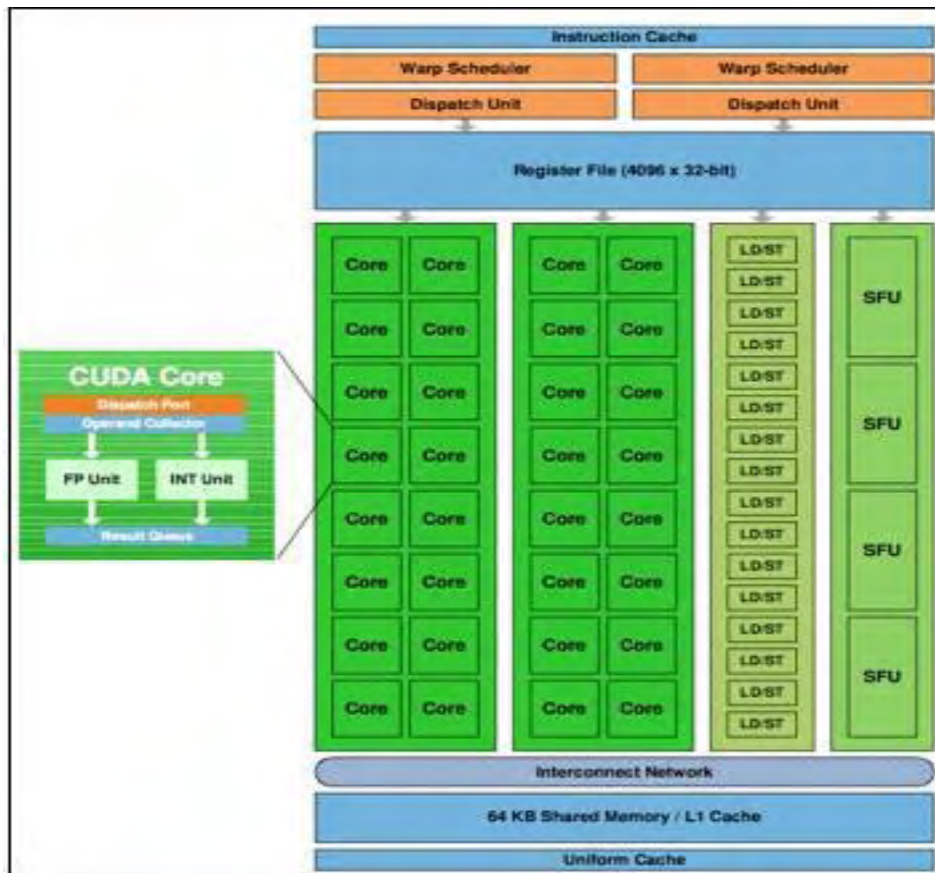
Το κυρίως μπλοκ σύνθεσης της αρχιτεκτονικής Fermi φαίνεται στο Σχήμα 3.2. Αυτό το επίπεδο σύνθεσης αποκαλείται επεξεργαστής CUDA (*CUDA core*). Οι *CUDA cores* είναι πολύ απλοί στη λειτουργία και στην κατανόησή τους. Έχουν μια μονάδα σειριακής επεξεργασίας αριθμών κινητής υποδιαστολής (*FPU*), μια μονάδα σειριακής επεξεργασίας ακεραίων αριθμών (*INTU*), κάποια λογική για αποστολή

εντολών και τελεστών στις παραπάνω μονάδες και μια δομή αποθηκευτικού χώρου μορφής σειράς για τα εκάστοτε αποτελέσματα που επιφέρει η επεξεργασία των παραπάνω μονάδων. Αντίθετα με τις επιπρόσθετες δυνατότητες που έχουν οι επεξεργαστές της CPU, οι CUDA cores δεν έχουν μονάδες φόρτωσης και αποθήκευσης για πρόσβαση σε μνήμη, δεν έχουν δικό τους αρχείο καταχωρητών και δεν διαθέτουν και κρυφές μνήμες μορφής L1 (*L1 caches*). Οι εν λόγω επεξεργαστές δεν σχεδιάστηκαν για την αποδοτική εκτέλεση ακολουθιακών εφαρμογών, αλλά για την αποδοτική αξιοποίηση παραλληλοποιησιμων τμημάτων κώδικα που επιφέρει η συνεργασία και η ταυτόχρονη εκτέλεσή τους.



Σχήμα 3.2: Σύνθεση CUDA core

Πολλαπλοί CUDA cores συσσωρεύονται σε ένα υψηλότερο επίπεδο διαστρωμάτωσης ως πολυεπεξεργαστές ροής (*streaming multiprocessors*). Η μορφή κάθε τέτοιου *streaming multiprocessor* (SM) αναπαρίσται στο Σχήμα 3.3. Κάθε SM περιέχει 32 CUDA cores, οι οποίοι μοιράζονται όλοι μαζί από κοινού τους καταχωρητές, τις κρυφές μνήμες, την τοπική μνήμη και τις μονάδες φόρτωσης και αποθήκευσης του SM. Οι ειδικές μονάδες λειτουργίας (SFUs) του SM χειρίζονται σύνθετες μαθηματικές λειτουργίες, όπως τετραγωνικές ρίζες, ημίτονα, συνημίτονα κτλ. Με βάση τα παραπάνω μπορεί να γίνει εύκολα κατανοητό γιατί ένας CUDA core δεν χρειάζεται να έχει δικές του κρυφές μνήμες ή δικές του μονάδες φόρτωσης και αποθήκευσης. Οι 32 CUDA cores που διαθέτει κάθε SM λειτουργούν ως μια ομάδα που μοιράζεται τους ίδιους πόρους και έχει σχεδιαστεί να διαχειρίζεται ταυτόχρονα 32 ίδιες εντολές διαφορετικών δεδομένων από ένα σύνολο 32 νημάτων, το οποίο η NVIDIA αποκαλεί *warp*. Επιπρόσθετα, στις νέες αρχιτεκτονικές, δύο διαφορετικά *warps* μπορούν να εκτελεστούν ταυτόχρονα στον ίδιο SM, καθώς τώρα ο χρονοπρογραμματιστής (*scheduler*) του hardware της GPU μπορεί να εισάγει δύο διαφορετικές εντολές ανά χτύπο ρολογιού.

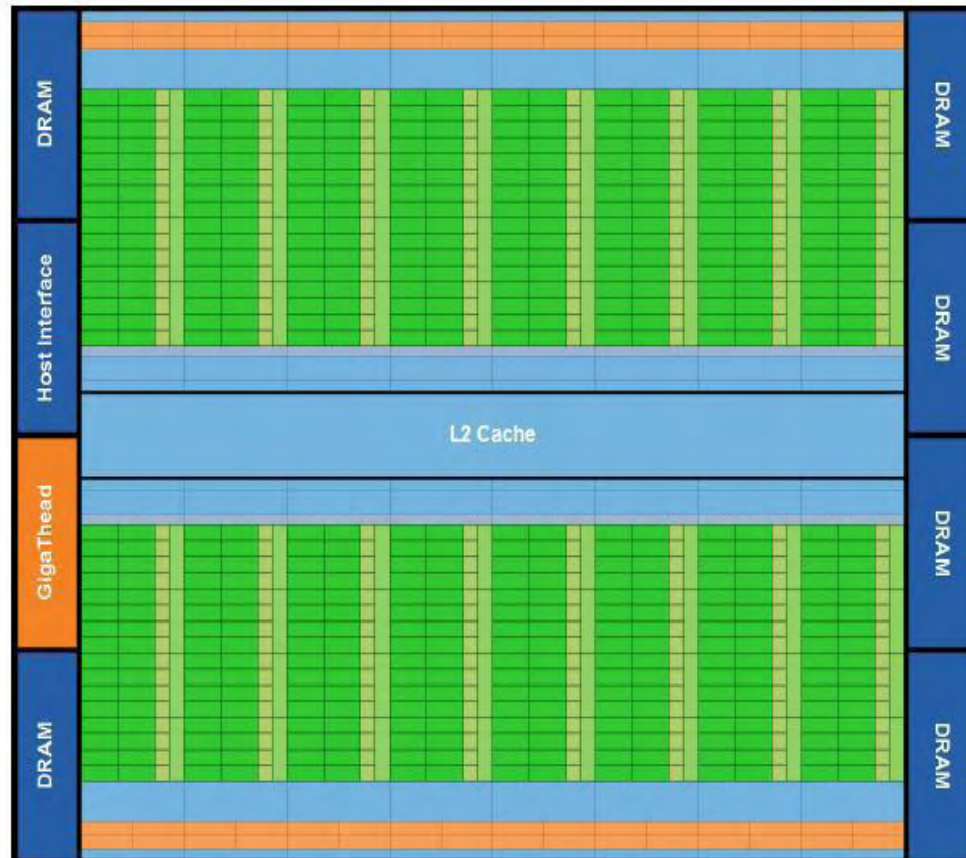


Σχήμα 3.3: Σύνθεση streaming multiprocessor

Η λειτουργία ενός SM μπορεί να συνοψιστεί με τον ακόλουθο τρόπο. Κάθε SM μπορεί να διαχειριστεί μέχρι 48 warps, όπου κάθε warp όπως αναφέρθηκε αποτελείται από μια ομάδα 32 νημάτων. Επομένως, κάθε SM μπορεί να διαχειριστεί μέχρι 1.536 νήματα και στην περίπτωση της κάρτας GTX480, η οποία έχει 15 SM, συνολικά μπορούν να υπάρξουν 23.040 παράλληλα νήματα. Όπως είναι φυσικό με όσα αναφέρθηκαν προηγουμένως, δεν μπορεί να υπάρξει ταυτόχρονη εκτέλεση και των 23.040 νημάτων, αλλά χρειάζεται να υπάρχουν ανά πάσα στιγμή όσο το δυνατόν περισσότερα ενεργά / απασχολημένα νήματα για την επίτευξη μεγάλου παραλληλισμού. Κάτι τέτοιο πρέπει να συμβαίνει διότι όταν ένα warp εκτελεί μια εντολή πρόσβασης σε μνήμη και περιμένει το αποτέλεσμα της, ο χρονοπρογραμματιστής νημάτων (thread scheduler) την ίδια στιγμή επιλέγει ένα άλλο έτοιμο προς εκτέλεση warp και με αυτόν τον τρόπο κρύβεται / καλύπτεται η καθυστέρηση πρόσβασης. Όλη αυτή η διαδικασία έχει σαν αποτέλεσμα να εκτελείται ανά πάσα στιγμή, ταυτόχρονα, ο μέγιστος αριθμός νημάτων που μπορεί να υποστηρίξει η εκάστοτε GPU. Στην συγκεκριμένη περίπτωση, με την χρήση της GTX480, αυτός ο αριθμός είναι 480 νήματα ($15 \text{ SM} * (32 \text{ CUDA cores} / \text{SM})$).

Το τελευταίο και υψηλότερο επίπεδο σύνθεσης φαίνεται στο Σχήμα 3.4. Εύκολα μπορεί να διακρίνει κάποιος τους 16 SMs (αν και η GTX480 χρησιμοποιεί τους 15 από αυτούς) και την διαμοιρασμένη από κοινού ανάμεσά τους L2 cache μεγέθους 768KB. Πέρα από αυτά, αντιληπτά είναι επίσης τα έξι 64-bit DRAM chipsets

μεγέθους 256MB το καθένα, σαν σύνολο γνωστά ως *global memory*, η διεπαφή Host (Host Interface (PCI Express)), αλλά και το υλικό χρονοπρογραμματισμού νημάτων GigaThread. Το εν λόγω υλικό είναι υπεύθυνο για την διαχείριση χιλιάδων νημάτων ταυτόχρονης εκτέλεσης, αλλά και για γρήγορες εναλλαγές περιβάλλοντος (context switches) ανάμεσα σε γραφικά και εφαρμογές υπολογιστικής φύσης.



Σχήμα 3.4: Διάγραμμα τμήματος αρχιτεκτονικής Fermi

Αντιλαμβάνοντας κανείς τον τρόπο με τον οποίο λειτουργεί μια GPU, μπορεί να συμπεράνει ότι είναι τελείως διαφορετικός από αυτόν μιας CPU. Μια CPU μπορεί να επικοινωνήσει απευθείας με την κύρια μνήμη του συστήματος (RAM) μέσω ενός ενσωματωμένου σε αυτή ρυθμιστή μνήμης (IMC). Αντίθετα, η GPU έχει πρόσβαση στην παραπάνω μνήμη μόνο μέσω του Host Interface και συγκεκριμένα μέσω του PCI Express, γεγονός που προκαλεί μεγάλη καθυστέρηση και αποτελεί πρόβλημα. Εξαιτίας αυτού, υπήρχε έντονα η ανάγκη για την εύρεση ενός πιο αποδοτικού τρόπου μεταφοράς δεδομένων.

Για αυτόν τον λόγο, οι σχεδιαστές εγκατέστησαν διάφορα είδη μνημών, εντός και εκτός του chipset της κάρτας γραφικών, για την επίτευξη αποδοτικότερων χρόνων πρόσβασης.

Στο Σχήμα 3.5 φαίνεται η βασική ιεραρχία της εν λόγω μνήμης, η οποία εξοικονομεί σημαντική ποσότητα χρόνου πρόσβασης. Το υψηλότερο επίπεδο της εν λόγω ιεραρχίας είναι η *DRAM* (*global memory*), η οποία στο σύνολό της είναι χωρητικότητας μεγαλύτερης του 1GB. Τα εκάστοτε ζητούμενα δεδομένα μεταφέρονται αρχικά από την *RAM* στην *DRAM* και η όλη μετέπειτα επεξεργασία πρόσβασής τους γίνεται αποκλειστικά μέσω της *DRAM*, χωρίς καμία επιπλέον εμπλοκή της *RAM*. Επιπλέον, όπως έχει ήδη αναφερθεί, η *global memory* αποτελείται από έξι 64-bit *DRAM* chipsets, τα οποία όμως είναι εγκατεστημένα στο PCB της κάρτας γραφικών και όχι μέσα στο chipset αυτής, γεγονός που μειώνει την ταχύτητα πρόσβασης. Αυτό συμβαίνει παρά του ότι η *DRAM* χρησιμοποιεί τεχνολογία DDR5 που συμβάλλει σε αρκετά γρήγορους χρόνους πρόσβασης και παρά του ότι είναι διαμοιρασμένη ανάμεσα σε όλους τους SMs.

Δύο επιπλέον μνήμες, στο επίπεδο αυτό, που προσφέρει το μοντέλο CUDA και είναι αρκετά πιο γρήγορες από την *global memory*, είναι η *constant memory* χωρητικότητας 64KB και η *texture memory* χωρητικότητας πολλών MB. Το κοινό χαρακτηριστικό και των δύο είναι ότι μπορούν να χρησιμοποιηθούν μόνο για ανάγνωση δεδομένων και όχι για εγγραφή αυτών, καθώς και το ότι μέρος των δεδομένων τους μπορεί να μεταφερθεί σε πολύ γρήγορες κρυφές μνήμες (*cached memories*). Το κόστος ανάγνωσης της *constant memory* μεγαλώνει ανάλογα με τον αριθμό των διαφορετικών διευθύνσεων που διαβάζονται από όλα τα νήματα. Η συγκεκριμένη ανάγνωση παρόλα αυτά μπορεί να είναι τόσο γρήγορη όσο η ανάγνωση από ένα καταχωρητή, με την προϋπόθεση όλα τα νήματα ενός μισού warp (*half-warp*) να διαβάζουν την ίδια διεύθυνση. Από την άλλη, το μέγεθος της *texture memory* σχετίζεται με το εκάστοτε μέγεθος της *global memory* και η ανάγνωση δεδομένων από αυτήν γίνεται μέσω κατάλληλων μεθόδων, ειδικά κατασκευασμένων για την διαχείριση της εν λόγω μνήμης.

Το αμέσως χαμηλότερο επίπεδο στο Σχήμα 3.5 περιλαμβάνει την κρυφή μνήμη μορφής L2 (*L2 cache memory*). Η L2 κατέχει κεντρική θέση στο chipset της GPU και αυτό είναι πολύ σημαντικό αν σκεφτεί κανείς ότι η διαμοιραζόμενη φύση της ανάμεσα στους SMs, καθιστά πολύ εύκολη την από κοινού πρόσβασή τους στην εν λόγω μνήμη. Η χωρητικότητά της είναι μόλις 768KB, σημαντικά μικρότερη της *global memory*, αλλά είναι πολύ πιο γρήγορη από αυτή. Η L2 συνήθως χρησιμοποιείται για την αποθήκευση δεδομένων μικρής χωρητικότητας, τα οποία χρησιμοποιούνται συχνά από τους SMs, με στόχο την αποφυγή επικοινωνίας με την *global memory*.

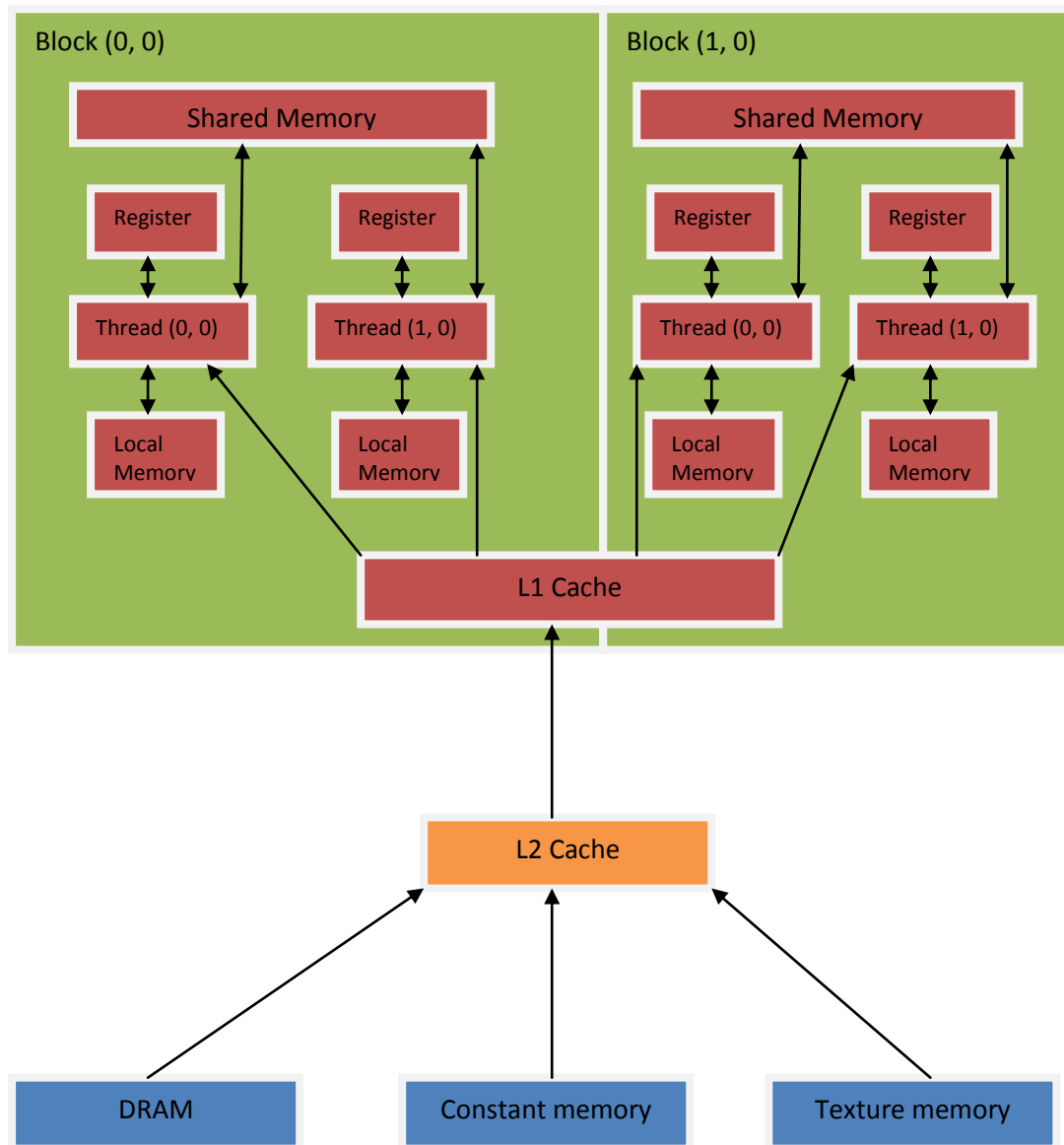
Αν και η ταχύτητα που προσφέρει η L2 είναι σημαντικά μεγάλη, η GPU διαθέτει και άλλα είδη μνήμης που βρίσκονται σε ακόμα χαμηλότερο επίπεδο, όπως απεικονίζεται στο Σχήμα 3.5. Αυτό το επίπεδο έχει να κάνει με ένα σύνολο μνημών και παρακάτω ακολουθεί μια συνοπτική ανάλυσή τους πάντα με βάση το μοντέλο κάρτας GTX480.

- ✓ *local memory*: Κάθε νήμα έχει την δικιά του *local memory* μεγέθους 16KB και η πρόσβαση σε αυτή μπορεί να γίνει και για ανάγνωση και για εγγραφή δεδομένων. Τμήμα δεδομένων αυτής δεν μπορεί να μεταφερθεί σε γρήγορες

μνήμες (not cached) και η πρόσβαση σε αυτή είναι το ίδιο ακριβή με την πρόσβαση στην *global memory*. Η διαφορά είναι ότι οι προσβάσεις δεδομένων στην *local memory* είναι πάντα ευθυγραμμισμένες (coalesced). Χρησιμοποιείται συνήθως για την αποθήκευση μεγάλων δομών δεδομένων που διαχειρίζεται κάθε νήμα.

- ✓ registers: Κάθε νήμα έχει τον δικό του αριθμό καταχωρητών, όπου ο αριθμός αυτός καθορίζεται από την διαμοίραση των πόρων που θα επιφέρει η γεωμετρία εκτέλεσης του εκάστοτε kernel. Η μνήμη αυτή είναι συνήθως περιορισμένου μεγέθους, αλλά αποτελεί το γρηγορότερο είδος μνήμης στη GPU και μπορεί να χρησιμοποιηθεί και για ανάγνωση και για εγγραφή δεδομένων.
- ✓ shared memory: Κάθε μπλοκ νημάτων έχει την δικιά του *shared memory* μεγέθους 48KB και η πρόσβαση σε αυτή μπορεί να γίνει και για ανάγνωση και για εγγραφή δεδομένων. Ένα νήμα ενός block δεν μπορεί να έχει πρόσβαση στην *shared memory* ενός άλλου block. Η εν λόγω μνήμη είναι διαμοιρασμένη σε ομοιόμορφου μεγέθους τμήματα (banks), στα οποία μπορεί να υπάρξει πρόσβαση ταυτόχρονα από κάθε νήμα. Η πρόσβαση στη συγκεκριμένη μνήμη είναι το ίδιο γρήγορη όπως η πρόσβαση σε έναν καταχωρητή με την προϋπόθεση να μην δημιουργούνται συγκρούσεις κατά την πρόσβαση των banks (bank conflicts). Χρησιμοποιείται κυρίως για τον συγχρονισμό των δεδομένων ανάμεσα σε νήματα διαφορετικών blocks.
- ✓ L1 cache: Κάθε streaming multiprocessor έχει την δικιά του *L1 cache* χωρητικότητας 64KB. Αποτελεί μαζί με τους *registers* την πιο γρήγορη μορφή μνήμης σημειώνοντας μικρή καθυστέρηση (latency) και μεγάλο εύρος ζώνης (bandwidth). Οι 32 CUDA cores κάθε SM διαβάζουν από κοινού την εν λόγω μνήμη και δεν μπορεί ένας CUDA core ενός SM να έχει πρόσβαση στην L1 ενός άλλου core διαφορετικού SM.

Να σημειωθεί ότι στο παρακάτω σχήμα έχει παραλειφτεί η επικοινωνία των threads με το επίπεδο των καθολικών, ανάμεσα στα threads μνημών, *DRAM*, *constant memory* και *texture memory* για λόγους απλότητας του σχήματος. Επίσης, ενδεικτικά έχουν παρουσιαστεί δύο blocks για να φανεί καλύτερα η ανεξάρτητη φύση που έχουν ανά block, οι μνήμες *shared*, *register* και *local*. Για την *L1 cache* έχει θεωρηθεί ότι το μέγεθος των blocks είναι τέτοιο έτσι ώστε να ανήκουν στον ίδιο SM. Τέλος, με αμφίδρομο βελάκι σημειώνεται η επικοινωνία για ανάγνωση και εγγραφή, ενώ με μονό σημειώνεται η επικοινωνία αποκλειστικά και μόνο για ανάγνωση.



Σχήμα 3.5: Ιεραρχία μνήμης αρχιτεκτονικής Fermi

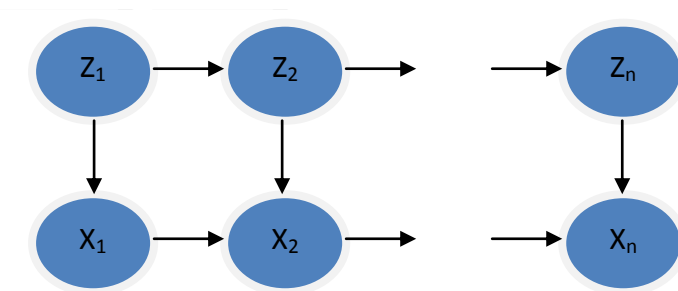
Κεφάλαιο 4

Βελτιστοποίηση των συναρτήσεων Forward, Backward

4.1 To Hidden Markov Model

Οι συναρτήσεις “Forward” και “Backward”, αλλά και η συνάρτηση “BackwardBaumWelch”, για την οποία γίνεται λόγος στο επόμενο κεφάλαιο όσον αφορά την βελτιστοποίησή της, πραγματοποιούν τους υπολογισμούς τους βάσει του γνωστού HMM.

Ένα Hidden Markov Model (HMM) έχει την μορφή που φαίνεται στο Σχήμα 4.1.



Σχήμα 4.1: To Hidden Markov Model

Το εν λόγω μοντέλο απαρτίζεται από:

Τυχαίες μεταβλητές Z_1, Z_2, \dots, Z_n Z (προσπαθούμε να τις βρούμε)

Τυχαίες μεταβλητές X_1, X_2, \dots, X_n X (γνωστά δεδομένα εκ των προτέρων)

Οι μεταβλητές Z_1, Z_2, \dots, Z_n μπορούν να πάρουν διακριτές τιμές, ενώ οι μεταβλητές X_1, X_2, \dots, X_n μπορούν να πάρουν διακριτές, πεπερασμένες, R, R^d τιμές.

Οι παράμετροι ενός HMM είναι:

✓ Transition probs: $\mathbf{T}(i, j) = P(Z_{k+1} = j \mid Z_k = i)$, όπου $i, j \in \{1, \dots, m\}$

✓ Emission probs: $\mathbf{E}_i(\mathbf{x}) = P(\mathbf{x} \mid Z_k = i)$, όπου $i \in \{1, \dots, m\}, \mathbf{x} \in X$
(ή)

$\mathbf{E}_i(\mathbf{x}) = P(X_k = \mathbf{x} \mid Z_k = i)$ αν το X διακριτό σύνολο τιμών

✓ Initial prob: $\mathbf{\Pi}(i) = P(Z_1 = i)$, όπου $i \in \{1, \dots, m\}$

Μία ενδεικτική και γνωστή χρήση του HMM είναι η αναγνώριση γραφικού χαρακτήρα (handwriting recognition). Για παράδειγμα αν το X_1 αντιστοιχεί σε κάποιο γράμμα που έγραψε κάποιος, το Z_1 είναι το γράμμα που προκύπτει βάσει του μοντέλου. Υπολογίζεται δηλαδή η πιθανότητα $P(Z_1 | X_1)$ δεδομένων των παραμέτρων του HMM που αναφέρθηκαν παραπάνω και ερμηνεύεται αναλόγως.

4.2 Η συνάρτηση Forward

4.2.1 Φυσική σημασία υπολογισμών

Η φυσική σημασία των υπολογισμών που πραγματοποιεί η συνάρτηση “Forward” είναι η εύρεση των πιθανοτήτων $P(Z_k, X_{1:k})$ (posterior probs), όπου $X_{1:k} = (X_1, \dots, X_k)$. Κάθε τέτοια πιθανότητα δείχνει κατά πόσο δύο στοιχεία είναι όμοια και άρα μπορούν να ευθυγραμμιστούν, ή δεν είναι και συνεπώς δεν μπορούν (gap).

Αν θέσουμε $a_k(Z_k) = P(Z_k, X_{1:k})$, έχουμε την παρακάτω αναδρομική μορφή εξάρτησης των υπολογισμών, που προκύπτει μέσω του HMM (Σχήμα 4.1):

$$a_k(Z_k) = \text{sum}(P(X_k | Z_k) * P(Z_k | Z_{k-1}) * a_{k-1}(Z_{k-1})), \text{ από } Z_{k-1} = 1 \text{ έως } m, \text{ όπου } k = 2, \dots, n$$

$$a_1(Z_1) = P(Z_1, X_1) = P(Z_1) * P(X_1 | Z_1)$$

, όπου $P(X_k | Z_k)$ emission probs, $P(Z_k | Z_{k-1})$ transition probs και $P(Z_1)$ initial prob.

Κάθε στοιχείο του δισδιάστατου πίνακα (FW_table) που γεμίζει η “Forward” και αντιστοιχεί σε μια πιθανότητα $P(Z_k, X_{1:k})$, εκτελεί τους παρακάτω υπολογισμούς (ενδεικτικά παρουσιάζεται η μορφή υπολογισμών του στοιχείου (2, 3)):

$$\text{FW_table}(2, 3)[2] = \text{add_func}(\text{FW_table}(2, 3)[2], \text{FW_table}(2, 2)[1], T[5], E[0]);$$

$$\text{FW_table}(2, 3)[2] = \text{add_func}(\text{FW_table}(2, 3)[2], \text{FW_table}(2, 2)[2], T[11], E[0]);$$

$$\text{FW_table}(2, 3)[0] = \text{add_func}(\text{FW_table}(2, 3)[0], \text{FW_table}(1, 3)[1], T[4], E[1]);$$

$$\text{FW_table}(2, 3)[0] = \text{add_func}(\text{FW_table}(2, 3)[0], \text{FW_table}(1, 3)[0], T[8], E[1]);$$

$$\text{FW_table}(2, 3)[1] = \text{add_func}(\text{FW_table}(2, 3)[1], \text{FW_table}(1, 2)[1], T[3], E[2]);$$

$$\text{FW_table}(2, 3)[1] = \text{add_func}(\text{FW_table}(2, 3)[1], \text{FW_table}(1, 2)[0], T[7], E[2]);$$

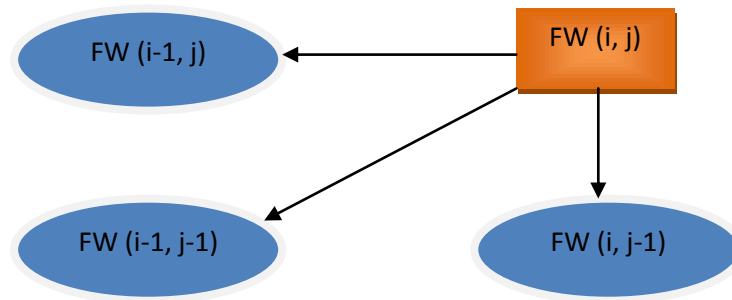
$$\text{FW_table}(2, 3)[1] = \text{add_func}(\text{FW_table}(2, 3)[1], \text{FW_table}(1, 2)[2], T[10], E[2]);$$

Όπως μπορεί να γίνει εύκολα αντιληπτό, η μορφή υπολογισμών που συμβάλλουν στην εύρεση του στοιχείου (2, 3) αντιστοιχεί στην αναδρομική μορφή εξάρτησης που προκύπτει μέσω του HMM, όπως παρουσιάστηκε νωρίτερα. Αυτό βέβαια συμβαίνει για κάθε στοιχείο του δισδιάστατου πίνακα FW_table.

Να σημειωθεί ότι ο πίνακας $T[]$ αντιστοιχεί στις transition probs, ο πίνακας $E[]$ στις emission probs, ενώ το $a_{k-1}(Z_{k-1})$ στα εκάστοτε στοιχεία του FW_table (που χρησιμοποιούνται για ανάγνωση), εκτός του στοιχείου προς εύρεση (που χρησιμοποιείται για εγγραφή).

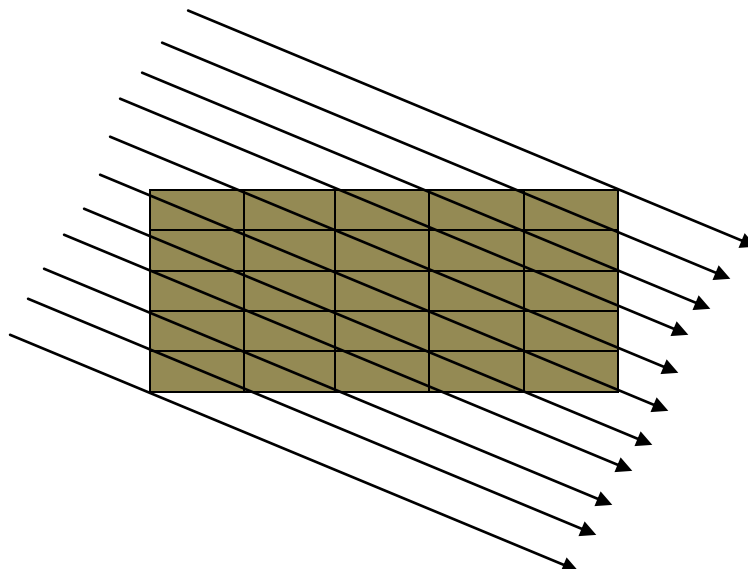
4.2.2 Παραλληλισμός

Ο παραλληλισμός στη “Forward” βρίσκεται στις διαγωνίους του δισδιάστατου πίνακα FW_table που υπολογίζεται μέσω αυτής. Συγκεκριμένα κάθε στοιχείο του εν λόγω πίνακα έχει την μορφή εξάρτησης που φαίνεται στο Σχήμα 4.2.1.



Σχήμα 4.2.1: Εξάρτηση δεδομένων σε “Forward”

Ο υπολογισμός του πίνακα FW_table εκτείνεται από κάτω προς τα πάνω και εντάσσεται στην γνωστή κατηγορία “*wavefront data dependencies & computation*”. Σύμφωνα με την εν λόγω κατηγορία, η εύρεση των στοιχείων της επόμενης διαγωνίου είναι εφικτή μόνο όταν ολοκληρωθεί ο υπολογισμός των στοιχείων που βρίσκονται στην προηγούμενη διαγώνιο. Χαρακτηριστικό είναι το Σχήμα 4.2.2.



Σχήμα 4.2.2: Εξάρτηση σε διαγωνίους από κάτω προς τα πάνω σε “Forward”

4.2.3 Βελτιστοποίηση_1 – παραλληλία στοιχείων σε διαγώνιο

Κάθε thread αναλαμβάνει τον υπολογισμό ενός δισδιάστατου στοιχείου 3 πεδίων του πίνακα *FW_table*. Το μέγεθος του block που χρησιμοποιείται είναι 512 threads και ο αριθμός αυτών είναι $(\text{συνολικός αριθμός threads σε διαγώνιο} / 512) + \text{padding}$, σε περίπτωση που η διαίρεση δώσει μη ακέραιο αριθμό. Επίσης, το grid είναι μονοδιάστατο, πράγμα που σημαίνει ότι υπάρχουν blocks μόνο στην διάσταση x. Επιλέχθηκε μονοδιάστατο grid, διότι είναι αρκετό για τον χειρισμό ακόμη και πολύ μεγάλων ακολουθιών εισόδου.

Τα blocks σαρώνουν σειριακά τα στοιχεία της διαγωνίου και πολλά από αυτά χρειάζεται να κάνουν άσκοπους υπολογισμούς, ειδικότερα στις αρχικές και στις τελευταίες διαγωνίους του πίνακα, για να μην προκαλούνται *divergencies* μέσα στα *warps*. Ο πίνακας *FW_table* γεμίζει με την χρήση της τεχνικής του δυναμικού προγραμματισμού, κατά την οποία χρησιμοποιούνται αποτελέσματα που υπολογίστηκαν πρόσφατα.

Για να επιτευχθεί συγχρονισμός ανάμεσα σε 2 διαδοχικές διαγωνίους, χρειάζεται κατάλληλη ρουτίνα υλοποίησης *global_barrier*, βάσει της οποίας θα διαβεβαιώνεται ότι έχουν ολοκληρώσει όλα τα blocks τους υπολογισμούς τους στην εκάστοτε προηγούμενη διαγώνιο και πλέον είναι εφικτή η έναρξη των υπολογισμών τους που αφορούν την εκάστοτε επόμενη διαγώνιο. Η ύπαρξη *global_barrier* είναι ιδιαίτερα επίπονη χρονικά, αλλά η μη χρησιμοποίησή της δεν μπορεί να αποφευχθεί.

4.2.4 Βελτιστοποίηση_2 – χρήση shared memory

Με την χρησιμοποίηση της *shared memory* μπορεί να μειωθεί ο αριθμός των προσβάσεων στην *global memory* ανά thread από 14 σε 6. Ενδεικτικά το thread που αναλαμβάνει το στοιχείο (2, 3) εκτελεί τους παρακάτω υπολογισμούς:

```
FW_table(2, 3)[2] = add_func( FW_table(2, 3)[2], FW_table(2, 2)[1], T[5] , E[0] );  
FW_table(2, 3)[2] = add_func( FW_table(2, 3)[2], FW_table(2, 2)[2], T[11], E[0] );
```

```
FW_table(2, 3)[0] = add_func( FW_table(2, 3)[0], FW_table(1, 3)[1], T[4], E[1] );  
FW_table(2, 3)[0] = add_func( FW_table(2, 3)[0], FW_table(1, 3)[0], T[8], E[1] );
```

```
FW_table(2, 3)[1] = add_func( FW_table(2, 3)[1], FW_table(1, 2)[1], T[3] , E[2] );  
FW_table(2, 3)[1] = add_func( FW_table(2, 3)[1], FW_table(1, 2)[0], T[7] , E[2] );  
FW_table(2, 3)[1] = add_func( FW_table(2, 3)[1], FW_table(1, 2)[2], T[10], E[2] );
```

Είναι προφανές ότι αριστερά των ισοτήτων χρειάζονται 7 προσβάσεις *global memory* για εγγραφή του στοιχείου (2, 3) και δεξιά των ισοτήτων χρειάζονται 7 προσβάσεις *global memory* για ανάγνωση του στοιχείου (2, 3).

Με την χρήση της *shared memory* οι υπολογισμοί μετασχηματίζονται ως εξής:

```
// copy values from global to shared memory

s_data[0] = FW_table(2, 3)[0];
s_data[1] = FW_table(2, 3)[1];
s_data[2] = FW_table(2, 3)[2];

// execute the computations using shared memory

s_data[2] = add_func( s_data[2], FW_table(2, 2)[1], T[5] , E[0] );
s_data[2] = add_func( s_data[2], FW_table(2, 2)[2], T[11], E[0] );

s_data[0] = add_func( s_data[0], FW_table(1, 3)[1], T[4], E[1] );
s_data[0] = add_func( s_data[0], FW_table(1, 3)[0], T[8], E[1] );

s_data[1] = add_func( s_data[1], FW_table(1, 2)[1], T[3] , E[2] );
s_data[1] = add_func( s_data[1], FW_table(1, 2)[0], T[7] , E[2] );
s_data[1] = add_func( s_data[1], FW_table(1, 2)[2], T[10], E[2] );

// copy values from shared to global memory

FW_table(2, 3)[0] = s_data[0];
FW_table(2, 3)[1] = s_data[1];
FW_table(2, 3)[2] = s_data[2];
```

Όπως μπορεί να γίνει εύκολα αντιληπτό, οι μοναδικές προσβάσεις ανά thread με την *global memory* υφίστανται μόνο κατά την επικοινωνία αυτής με την *shared*.

Να σημειωθεί ότι η γεωμετρία εκτέλεσης του kernel και η ύπαρξη του *global_barrier* κατά το στάδιο της βελτιστοποίησης_2 δεν αλλάζει.

4.2.5 Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο

Κάθε thread αναλαμβάνει τώρα τον υπολογισμό ενός πεδίου ενός δισδιάστατου στοιχείου 3 πεδίων του πίνακα *FW_table*. Συγκεκριμένα, τα threads χωρίζονται σε 3 ομάδες, όπου η πρώτη υπολογίζει το πεδίο [2] των στοιχείων, η δεύτερη το πεδίο [0] των στοιχείων και η τρίτη το πεδίο [1] των στοιχείων. Για να γίνει όμως αυτό χωρίς να υπάρχουν *divergencies* μέσα στα *warps*, πρέπει το μέγεθος του block που θα επιλεγεί να είναι πολλαπλάσιο του 32 και να διαιρείται ακριβώς με το 3. Με αυτόν τον τρόπο επιτυγχάνεται συνολικός υπολογισμός (*block_size* / 3) στοιχείων ανά block.

Το μέγεθος του block που χρησιμοποιείται είναι 480 threads και ο αριθμός αυτών είναι ((3 * συνολικός αριθμός στοιχείων σε διαγώνιο) / 480) + padding , σε περίπτωση που η διαίρεση δώσει μη ακέραιο αριθμό. Το grid παραμένει

μονοδιάστατο, καθώς η μνήμη που δεσμεύεται παραπάνω στην *global memory* για λογαριασμό του *FW_table* είναι ελάχιστη. Τέλος, η ύπαρξη του *global_barrier* παραμένει και αυτή ως έχει.

4.3 Η συνάρτηση Backward

4.3.1 Φυσική σημασία υπολογισμών

Η φυσική σημασία των υπολογισμών που πραγματοποιεί η συνάρτηση “*Backward*” είναι η εύρεση των πιθανοτήτων $P(X_{k+1:n} | Z_k)$ (*posterior probs*), όπου $X_{k+1:n} = (X_{k+1}, \dots, X_n)$. Κάθε τέτοια πιθανότητα δείχνει κατά πόσο δύο στοιχεία είναι όμοια και άρα μπορούν να ευθυγραμμιστούν, ή δεν είναι και συνεπώς δεν μπορούν (*gap*).

Αν θέσουμε $\beta_k(Z_k) = P(X_{k+1:n} | Z_k)$, έχουμε την παρακάτω αναδρομική μορφή εξάρτησης των υπολογισμών, που προκύπτει μέσω του HMM (Σχήμα 4.1):

$$\beta_k(Z_k) = \text{sum}(\beta_{k+1}(Z_{k+1}) * P(X_{k+1} | Z_{k+1}) * P(Z_{k+1} | Z_k)), \text{ από } Z_{k+1} = 1 \text{ έως } m, \text{ όπου } k = 1, \dots, n - 1$$

$$\beta_n(Z_n) = 1, \quad Z_n$$

, όπου $P(X_{k+1} | Z_{k+1})$ emission probs, $P(Z_{k+1} | Z_k)$ transition probs και $\beta_n(Z_n)$ initial prob.

Κάθε στοιχείο του δισδιάστατου πίνακα (*BW_table*) που γεμίζει η “*Backward*” και αντιστοιχεί σε μια πιθανότητα $P(X_{k+1:n} | Z_k)$, εκτελεί τους παρακάτω υπολογισμούς (ενδεικτικά παρουσιάζεται η μορφή υπολογισμών του στοιχείου (2, 3)):

```
BW_table(2, 3)[2] = mul_func( BW_table(2, 4)[2], T[11], E[0] );
BW_table(2, 3)[2] = add_func( BW_table(2, 3)[2], BW_table(3, 4)[1], T[10], E[2] );

BW_table(2, 3)[0] = mul_func( BW_table(3, 3)[0], T[8], E[1] );
BW_table(2, 3)[0] = add_func( BW_table(2, 3)[0], BW_table(3, 4)[1], T[7], E[2] );

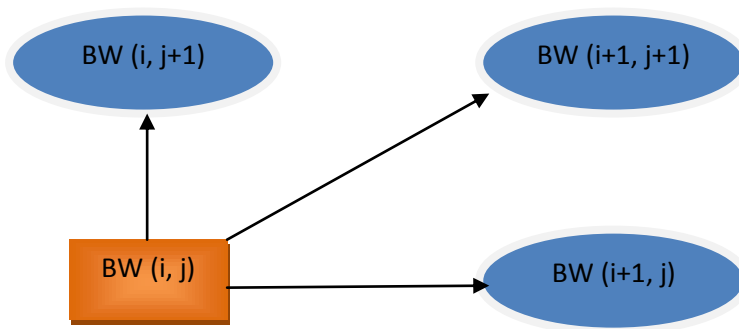
BW_table(2, 3)[1] = mul_func( BW_table(2, 4)[2], T[5], E[0] );
BW_table(2, 3)[1] = add_func( BW_table(2, 3)[1], BW_table(3, 3)[0], T[4], E[1] );
BW_table(2, 3)[1] = add_func( BW_table(2, 3)[1], BW_table(3, 4)[1], T[3], E[2] );
```

Όπως μπορεί να γίνει εύκολα αντιληπτό, η μορφή υπολογισμών που συμβάλλουν στην εύρεση του στοιχείου (2, 3) αντιστοιχεί στην αναδρομική μορφή εξάρτησης που προκύπτει μέσω του HMM, όπως παρουσιάστηκε νωρίτερα. Αυτό βέβαια συμβαίνει για κάθε στοιχείο του δισδιάστατου πίνακα *BW_table*.

Να σημειωθεί ότι ο πίνακας *T[]* αντιστοιχεί στις transition probs, ο πίνακας *E[]* στις emission probs, ενώ το $\beta_{k+1}(Z_{k+1})$ στα εκάστοτε στοιχεία του *BW_table* (που χρησιμοποιούνται για ανάγνωση), εκτός του στοιχείου προς εύρεση (που χρησιμοποιείται για εγγραφή).

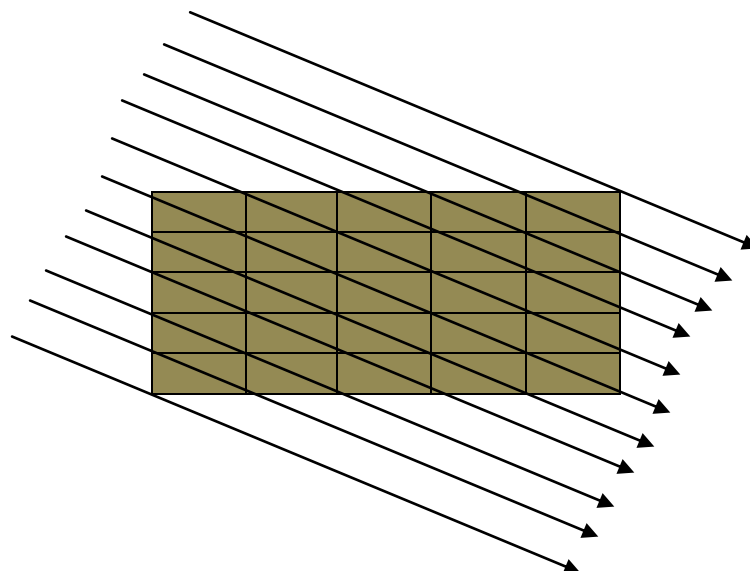
4.3.2 Παραλληλισμός

Ο παραλληλισμός στη “Backward” βρίσκεται στις διαγωνίους του δισδιάστατου πίνακα BW_table που υπολογίζεται μέσω αυτής. Συγκεκριμένα κάθε στοιχείο του εν λόγω πίνακα έχει την μορφή εξάρτησης που φαίνεται στο Σχήμα 4.3.1.



Σχήμα 4.3.1: Εξάρτηση δεδομένων σε “Backward”

Ο υπολογισμός του πίνακα BW_table εκτείνεται από πάνω προς τα κάτω και εντάσσεται στην γνωστή κατηγορία “wavefront data dependencies & computation”. Σύμφωνα με την εν λόγω κατηγορία, η εύρεση των στοιχείων της επόμενης διαγωνίου είναι εφικτή μόνο όταν ολοκληρωθεί ο υπολογισμός των στοιχείων που βρίσκονται στην προηγούμενη διαγώνιο. Χαρακτηριστικό είναι το Σχήμα 4.3.2.



Σχήμα 4.3.2: Εξάρτηση σε διαγωνίους από πάνω προς τα κάτω σε “Backward”

4.3.3 Βελτιστοποίηση_1 – παραλληλία στοιχείων σε διαγώνιο

Κάθε thread αναλαμβάνει τον υπολογισμό ενός δισδιάστατου στοιχείου 3 πεδίων του πίνακα BW_table. Το μέγεθος του block που χρησιμοποιείται είναι 512 threads και ο αριθμός αυτών είναι $(\text{συνολικός αριθμός threads σε διαγώνιο} / 512) + \text{padding}$, σε περίπτωση που η διαίρεση δώσει μη ακέραιο αριθμό. Επίσης, το grid είναι μονοδιάστατο, πράγμα που σημαίνει ότι υπάρχουν blocks μόνο στην διάσταση x. Επιλέχθηκε μονοδιάστατο grid, διότι είναι αρκετό για τον χειρισμό ακόμη και πολύ μεγάλων ακολουθιών εισόδου.

Τα blocks σαρώνουν σειριακά τα στοιχεία της διαγωνίου και πολλά από αυτά χρειάζεται να κάνουν άσκοπους υπολογισμούς, ειδικότερα στις αρχικές και στις τελευταίες διαγωνίους του πίνακα, για να μην προκαλούνται divergencies μέσα στα warps. Ο πίνακας BW_table γεμίζει με την χρήση της τεχνικής του δυναμικού προγραμματισμού, κατά την οποία χρησιμοποιούνται αποτελέσματα που υπολογίστηκαν πρόσφατα.

Για να επιτευχθεί συγχρονισμός ανάμεσα σε 2 διαδοχικές διαγωνίους, χρειάζεται κατάλληλη ρουτίνα υλοποίησης *global_barrier*, βάσει της οποίας θα διαβεβαιώνεται ότι έχουν ολοκληρώσει όλα τα blocks τους υπολογισμούς τους στην εκάστοτε προηγούμενη διαγώνιο και πλέον είναι εφικτή η έναρξη των υπολογισμών τους που αφορούν την εκάστοτε επόμενη διαγώνιο. Η ύπαρξη *global_barrier* είναι ιδιαίτερα επίπονη χρονικά, αλλά η μη χρησιμοποίησή της δεν μπορεί να αποφευχθεί.

4.3.4 Βελτιστοποίηση_2 – χρήση shared memory

Με την χρησιμοποίηση της *shared memory* μπορεί να μειωθεί ο αριθμός των προσβάσεων στην *global memory* ανά thread από 11 σε 3. Ενδεικτικά το thread που αναλαμβάνει το στοιχείο (2, 3) εκτελεί τους παρακάτω υπολογισμούς:

```
BW_table(2, 3)[2] = mul_func( BW_table(2, 4)[2], T[11], E[0] );  
BW_table(2, 3)[2] = add_func( BW_table(2, 3)[2], BW_table(3, 4)[1], T[10], E[2] );
```

```
BW_table(2, 3)[0] = mul_func( BW_table(3, 3)[0], T[8], E[1] );  
BW_table(2, 3)[0] = add_func( BW_table(2, 3)[0], BW_table(3, 4)[1], T[7], E[2] );
```

```
BW_table(2, 3)[1] = mul_func( BW_table(2, 4)[2], T[5], E[0] );  
BW_table(2, 3)[1] = add_func( BW_table(2, 3)[1], BW_table(3, 3)[0], T[4], E[1] );  
BW_table(2, 3)[1] = add_func( BW_table(2, 3)[1], BW_table(3, 4)[1], T[3], E[2] );
```

Είναι προφανές ότι αριστερά των ισοτήτων χρειάζονται 7 προσβάσεις *global memory* για εγγραφή του στοιχείου (2, 3) και δεξιά των ισοτήτων χρειάζονται 4 προσβάσεις *global memory* για ανάγνωση του στοιχείου (2, 3).

Με την χρήση της *shared memory* οι υπολογισμοί μετασχηματίζονται ως εξής:

```
// execute the computations using shared memory

s_data[2] = mul_func( BW_table(2, 4)[2], T[11], E[0] );
s_data[2] = add_func( s_data[2], BW_table(3, 4)[1], T[10], E[2] );

s_data[0] = mul_func( BW_table(3, 3)[0], T[8], E[1] );
s_data[0] = add_func( s_data[0], BW_table(3, 4)[1], T[7], E[2] );

s_data[1] = mul_func( BW_table(2, 4)[2], T[5], E[0] );
s_data[1] = add_func( s_data[1], BW_table(3, 3)[0], T[4], E[1] );
s_data[1] = add_func( s_data[1], BW_table(3, 4)[1], T[3], E[2] );

// copy values from shared to global memory

BW_table(2, 3)[0] = s_data[0];
BW_table(2, 3)[1] = s_data[1];
BW_table(2, 3)[2] = s_data[2];
```

Όπως μπορεί να γίνει εύκολα αντιληπτό, οι μοναδικές προσβάσεις ανά thread με την *global memory* υφίστανται μόνο κατά την επικοινωνία αυτής με την *shared*.

Να σημειωθεί ότι η γεωμετρία εκτέλεσης του kernel και η ύπαρξη του *global_barrier* κατά το στάδιο της βελτιστοποίησης_2 δεν αλλάζει.

4.3.5 Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο

Κάθε thread αναλαμβάνει τώρα τον υπολογισμό ενός πεδίου ενός δισδιάστατου στοιχείου 3 πεδίων του πίνακα *BW_table*. Συγκεκριμένα, τα threads χωρίζονται σε 3 ομάδες, όπου η πρώτη υπολογίζει το πεδίο [2] των στοιχείων, η δεύτερη το πεδίο [0] των στοιχείων και η τρίτη το πεδίο [1] των στοιχείων. Για να γίνει όμως αυτό χωρίς να υπάρχουν *divergencies* μέσα στα *warps*, πρέπει το μέγεθος του block που θα επιλεγεί να είναι πολλαπλάσιο του 32 και να διαιρείται ακριβώς με το 3. Με αυτόν τον τρόπο επιτυγχάνεται συνολικός υπολογισμός (*block_size* / 3) στοιχείων ανά block.

Το μέγεθος του block που χρησιμοποιείται είναι 480 threads και ο αριθμός αυτών είναι ((3 * συνολικός αριθμός στοιχείων σε διαγώνιο) / 480) + padding), σε περίπτωση που η διαίρεση δώσει μη ακέραιο αριθμό. Το grid παραμένει μονοδιάστατο, καθώς η μνήμη που δεσμεύεται παραπάνω στην *global memory* για λογαριασμό του *BW_table* είναι ελάχιστη. Τέλος, η ύπαρξη του *global_barrier* παραμένει και αυτή ως έχει.

4.4 Ο αλγόριθμος Forward – Backward

Κατά το στάδιο “*Posterior probability calculation*” της εφαρμογής FSA, εκτελείται ο αλγόριθμος “*Forward – Backward*”. Ο εν λόγω αλγόριθμος υπολογίζει τις πιθανότητες $P(Z_k | X)$, όπου $X = (X_1, \dots, X_n)$. Συγκεκριμένα, αν αναλυθούν οι προαναφερόμενες πιθανότητες προκύπτει:

$$P(Z_k | X) \propto P(Z_k, X) = P(X_{k+1:n} | Z_k, X_{1:k}) * P(Z_k, X_{1:k}) = P(X_{k+1:n} | Z_k) * P(Z_k, X_{1:k}), \text{ όπου}$$

- ✓ $k = 1, \dots, n$
- ✓ \propto : ανάλογο της πιθανότητας $P(Z_k, X)$ ως συνάρτηση του Z_k
- ✓ $P(X_{k+1:n} | Z_k)$: οι πιθανότητες που υπολογίζει η συνάρτηση “*Backward*” (βλ. 4.3)
- ✓ $P(Z_k, X_{1:k})$: οι πιθανότητες που υπολογίζει η συνάρτηση “*Forward*” (βλ. 4.2)

Κεφάλαιο 5

Βελτιστοποίηση της συνάρτησης BackwardBaumWelch

5.1 Φυσική σημασία υπολογισμών

Η φυσική σημασία των υπολογισμών που πραγματοποιεί η συνάρτηση “BackwardBaumWelch” είναι η εύρεση των πινάκων transition_table και emission_table που περιλαμβάνουν τις πιθανότητες μετάβασης (transition, gap, indel) και εκπομπής (emission, substitution, mismatch) αντίστοιχα, του εκάστοτε HMM (Σχήμα 4.1) για το εκάστοτε ζευγάρι σύγκρισης. Για τον υπολογισμό των εν λόγω πινάκων απαιτείται, πέραν όλων των άλλων, η εύρεση των πιθανοτήτων $P(X_{k+1:n} | Z_k)$ (posterior probs), όπου $X_{k+1:n} = (X_{k+1}, \dots, X_n)$. Οι συγκεκριμένες πιθανότητες είναι αυτές που υπολογίζει η συνάρτηση “Backward”, η οποία αποτελεί τμήμα της συνάρτησης “BackwardBaumWelch”.

5.2 Παραλληλισμός

Το τμήμα εντολών που εκτελεί κάθε thread μέσα στην συνάρτηση “BackwardBaumWelch” μπορεί να διαχωριστεί σε 3 τμήματα ως εξής:

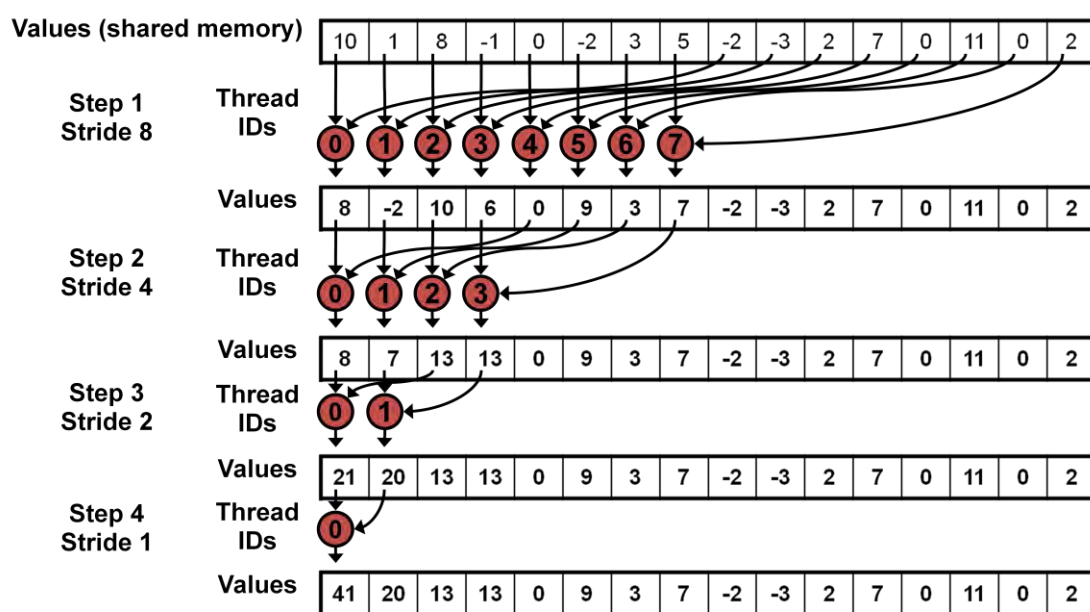
- ✓ Τμήμα 1: Ακολουθιακή και αναδρομική μορφή εξάρτησης δεδομένων που χρησιμοποιούνται μόνο για ανάγνωση, μέσω της χρήσης της συνάρτησης “Backward” για την εύρεση των πιθανοτήτων $P(X_{k+1:n} | Z_k)$, όπου $X_{k+1:n} = (X_{k+1}, \dots, X_n)$.
- ✓ Τμήμα 2: Ανεξάρτητες εντολές, ανά thread, τύπου πολλαπλασιασμού και πρόσθεσης. Μέσω αυτών υπολογίζεται ο πίνακας transition_table. Τα εν λόγω αθροίσματα έχουν να κάνουν σε σταθερές θέσεις του πίνακα transition_table.
- ✓ Τμήμα 3: Υπολογισμός πίνακα emission_table μέσω αθροισμάτων σε απροσδιόριστες εκ των προτέρων θέσεις αυτού. Η απροσδιοριστία έχει να κάνει με τη μη γνώση των στοιχείων των ακολουθιών εισόδου.

Ο παραλληλισμός συνεπώς της συνάρτησης “BackwardBaumWelch” έγκειται στο Τμήμα_2 αυτής. Συγκεκριμένα, τα αθροίσματα του εν λόγω τμήματος μπορούν να υλοποιηθούν με πολύ αποδοτικό τρόπο παράλληλα μέσω της γνωστής τεχνικής του “sum_reduction”, η οποία παρουσιάζεται στο Σχήμα 5.1.

Τα υπόλοιπα τμήματα λόγω της ακολουθιακής τους φύσης υλοποιούνται αναγκαστικά στην πλευρά της CPU (host side). Ο read_table περιέχει τις πιθανότητες που έχει υπολογίσει η συνάρτηση “Backward” (Τμήμα_1), ενώ ο emission_table στην GPU (Τμήμα_3) περιέχει τις τιμές που αθροίζονται κατά ακολουθιακό τρόπο στην πλευρά της CPU για την ολοκλήρωση του υπολογισμού

του `emission_table`. Για το εν λόγω άθροισμα προτιμάται η CPU, καθώς ένα thread στην CPU είναι πολύ πιο γρήγορο από ένα thread στην GPU. Το εν λόγω άθροισμα στην GPU δεν μπορεί να γίνει παράλληλα, γιατί δεν υπάρχει γνώση για το ποιες θέσεις του πίνακα `emission_table` γεμίζονται από ποια threads. Οι θέσεις αυτές εξαρτώνται από τα στοιχεία των ακολουθιών του εκάστοτε ζευγαριού σύγκρισης, τα οποία είναι αδύνατον να γνωρίζονται εκ των προτέρων.

Parallel Reduction: Sequential Addressing



Sequential addressing is conflict free

14

Σχήμα 5.1: Η τεχνική του *"sum_reduction"*

5.3 Βελτιστοποίηση

Το τμήμα εντολών που αναφέρθηκε παραπάνω βρίσκεται μέσα σε διπλό for που δείχνει τον συνολικό αριθμό στοιχείων. Κάθε thread αναλαμβάνει τον υπολογισμό ενός διδιάστατου στοιχείου 3 πεδίων. Το μέγεθος του block που χρησιμοποιείται είναι 512 threads και πρέπει να είναι δύναμη του δύο λόγω της τεχνικής *"sum_reduction"*. Ο συνολικός αριθμός blocks είναι $((iLen1 * iLen2) / BLOCK_SIZE) + padding$, όπου $iLen1$, $iLen2$ τα μήκη των ακολουθιών 1 και 2 αντίστοιχα του εκάστοτε ζευγαριού σύγκρισης. Το `padding` χρησιμοποιείται σε περίπτωση που η διαίρεση δώσει μη ακέραιο αριθμό. Τα blocks σαρώνουν σειριακά τον διδιάστατο πίνακα των στοιχείων, διαστάσεων ανάλογων του εκάστοτε ζευγαριού σύγκρισης, από κάτω προς τα πάνω. Τέλος, επιλέχθηκε grid 2 διαστάσεων, καθώς για μεγάλες ακολουθίες, τα 65535 blocks που διαθέτει η x διάσταση δεν επαρκούν και δεν χρειάζεται κάποιος έλεγχος για warp divergencies.

Κεφάλαιο 6

Μετρήσεις χρόνων σε GPU και σύγκριση με CPU

6.1 Σύστημα εκτέλεσης και αρχείο εισόδου

Το σύστημα στο οποίο πραγματοποιήθηκαν οι μετρήσεις που θα ακολουθήσουν αποτελείται από:

- ✓ επεξεργαστή intel i7 860
- ✓ 8GB RAM
- ✓ κάρτα γραφικών GeForce GTX480

Οι προδιαγραφές της GeForce GTX480 είναι:

- ✓ Core clock: 700MHz
- ✓ GPU clock: 1400MHz
- ✓ Memory clock: 1848MHz DDR5
- ✓ Memory capacity: 1536MB

Το λειτουργικό σύστημα που χρησιμοποιήθηκε ήταν το Ubuntu 12.04 LTS. Ο compiler για τα αρχεία που εκτελούνταν στη CPU ήταν ο g++, ενώ για την GPU ήταν ο nvcc. Η σύνδεση μεταξύ των αρχείων CPU με GPU γινόταν μέσω του g++. Ο τρόπος εκτέλεσης της εφαρμογής εμπεριείχε τα ορίσματα `--fast --noindel2 --refinement 0`.

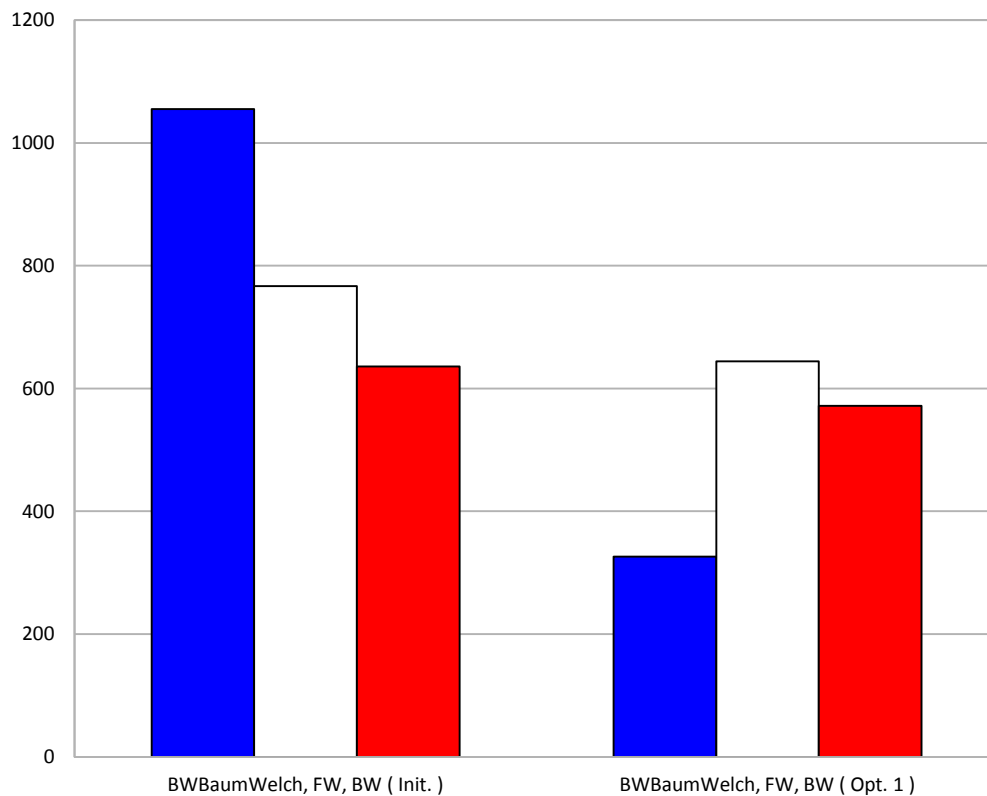
Το αρχείο εισόδου που χρησιμοποιήθηκε ήταν το *“ex2_protein.fasta”*, το οποίο περιέχει 73 ακολουθίες πρωτεϊνικής φύσεως. Το μέσο ζευγάρι σύγκρισης του εν λόγω αρχείου είναι (1417, 1460) και με βάση αυτό θα παρουσιαστούν οι μετρήσεις χρόνων όλων των σταδίων βελτιστοποίησης. Το μικρότερο ζευγάρι σύγκρισης είναι (1182, 1199), ενώ το μεγαλύτερο ζευγάρι σύγκρισης είναι (1949, 2205). Τα συγκεκριμένα ζευγάρια θα χρησιμοποιηθούν μαζί με το μέσο για να δείξουν την μεταβολή του speedup ανά συνάρτηση, καθώς αυξάνεται το μέγεθος του ζευγαριού.

Η βελτιστοποίηση, όπως έχει τονιστεί και νωρίτερα, γίνεται ανά ζευγάρι σύγκρισης (kernel). Όσο γρηγορότερα γίνεται η εν λόγω σύγκριση, τόσο μικρότερος ο συνολικός χρόνος εκτέλεσης που προκύπτει (total execution time).

Επίσης, να αναφερθεί ότι η βελτιστοποίηση που έγινε για την συνάρτηση *“BackwardBaumWelch”* είναι μία και μοναδική, ενώ για τις συναρτήσεις *“Forward”*, *“Backward”* υπάρχουν τρία στάδια βελτιστοποιήσεων. Παρόλα αυτά, σε κάθε γράφημα παρουσιάζεται και η βελτιστοποίηση της *“BackwardBaumWelch”*, αν και δεν αλλάζει, για λόγους αισθητικής και ομοιομορφίας των γραφημάτων.

6.2 Γραφική παρουσίαση βελτιστοποιήσεων

Στο Σχήμα 6.1 παρουσιάζεται η Βελτιστοποίηση_1 σε σχέση με την αρχική υλοποίηση, όσον αφορά τους kernels. Να σημειωθεί ότι σαν Βελτιστοποίηση_1 εννοείται η συγχώνευση των “Βελτιστοποίηση_1 – παραλληλία στοιχείων σε διαγώνιο (βλ. 4.2.3 & 4.3.3)” και “Βελτιστοποίηση (βλ. 5.3)”.



Σχήμα 6.1: Βελτιστοποίηση_1 (kernels)

➤ Initial (CPU)

- ✓ BWBaumWelch: 1,055 sec
- ✓ Forward: 0,767 sec
- ✓ Backward: 0,636 sec

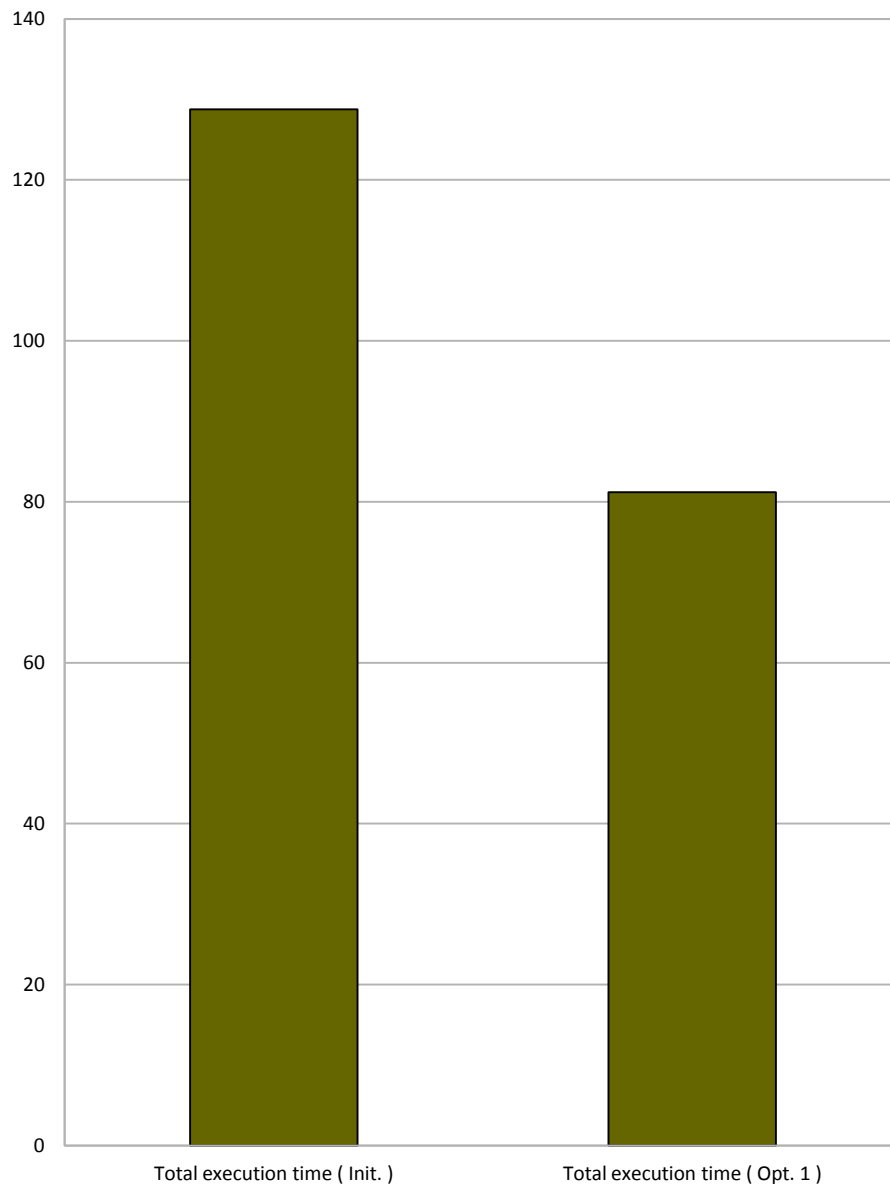
➤ Optimization 1 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,644 sec
- ✓ Backward: 0,572 sec

➤ Speedup

- ✓ BWBaumWelch: 3,236
- ✓ Forward: 1,191
- ✓ Backward: 1,112

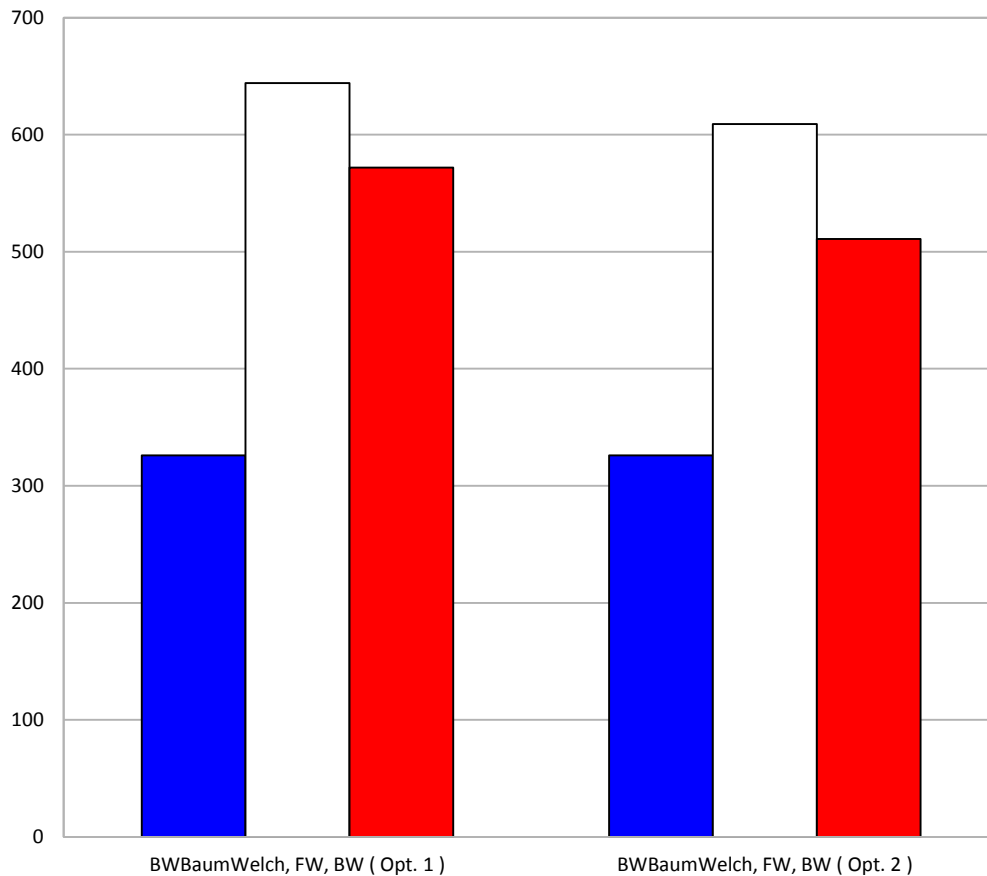
Έπειτα, στο Σχήμα 6.2 παρουσιάζεται η Βελτιστοποίηση_1 σε σχέση με την αρχική υλοποίηση, όσον αφορά τους τελικούς χρόνους εκτέλεσης.



Σχήμα 6.2: Βελτιστοποίηση_1 (total exec. time)

- Initial (CPU)
 - ✓ Total execution time: 128,756 min
- Optimization 1 (GPU)
 - ✓ Total execution time: 81,189 min
- Speedup
 - ✓ Total execution time: 1,586

Στο Σχήμα 6.3 παρουσιάζεται η Βελτιστοποίηση_2 σε σχέση με την Βελτιστοποίηση_1, όσον αφορά τους kernels. Να σημειωθεί ότι σαν Βελτιστοποίηση_2 εννοείται η συγχώνευση των “Βελτιστοποίηση_2 – χρήση *shared memory* (βλ. 4.2.4 & 4.3.4)” και “Βελτιστοποίηση (βλ. 5.3)”.



Σχήμα 6.3: Βελτιστοποίηση_2 (kernels)

➤ Optimization 1 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,644 sec
- ✓ Backward: 0,572 sec

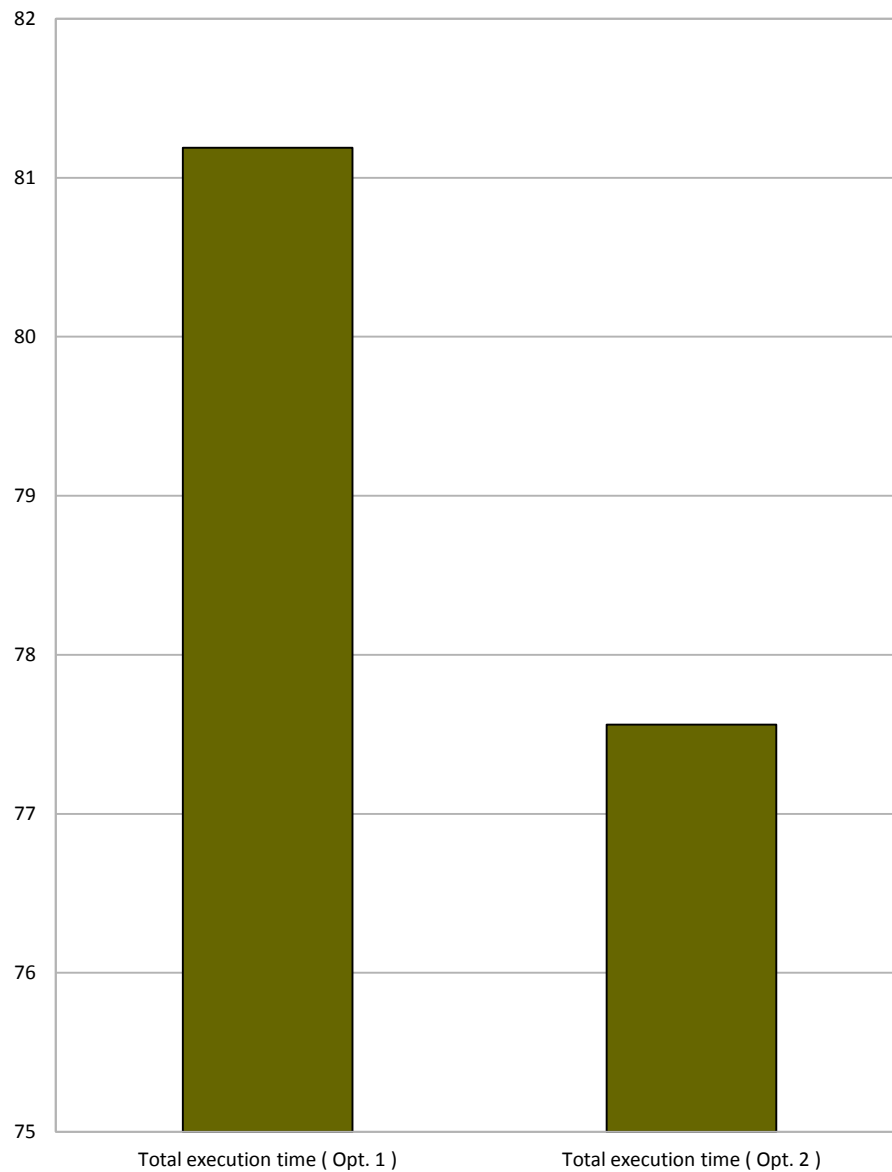
➤ Optimization 2 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,609 sec
- ✓ Backward: 0,511 sec

➤ Speedup

- ✓ BWBaumWelch: 1
- ✓ Forward: 1,057
- ✓ Backward: 1,119

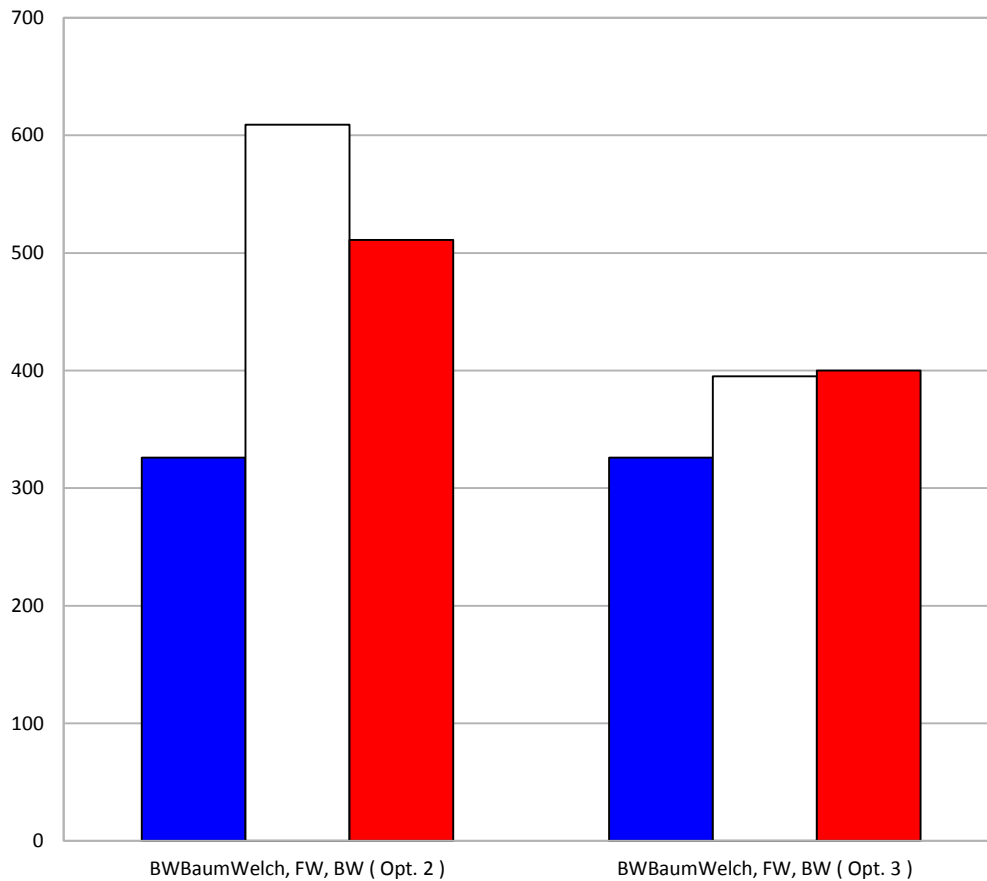
Στη συνέχεια, στο Σχήμα 6.4 παρουσιάζεται η Βελτιστοποίηση_2 σε σχέση με την Βελτιστοποίηση_1, όσον αφορά τους τελικούς χρόνους εκτέλεσης.



Σχήμα 6.4: Βελτιστοποίηση_2 (total exec. time)

- Optimization 1 (GPU)
 - ✓ Total execution time: 81,189 min
- Optimization 2 (GPU)
 - ✓ Total execution time: 77,56 min
- Speedup
 - ✓ Total execution time: 1,047

Στο Σχήμα 6.5 παρουσιάζεται η Βελτιστοποίηση_3 σε σχέση με την Βελτιστοποίηση_2, όσον αφορά τους kernels. Να σημειωθεί ότι σαν Βελτιστοποίηση_3 εννοείται η συγχώνευση των “Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο (βλ. 4.2.5 & 4.3.5)” και “Βελτιστοποίηση (βλ. 5.3)”.



Σχήμα 6.5: Βελτιστοποίηση_3 (kernels)

➤ Optimization 2 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,609 sec
- ✓ Backward: 0,511 sec

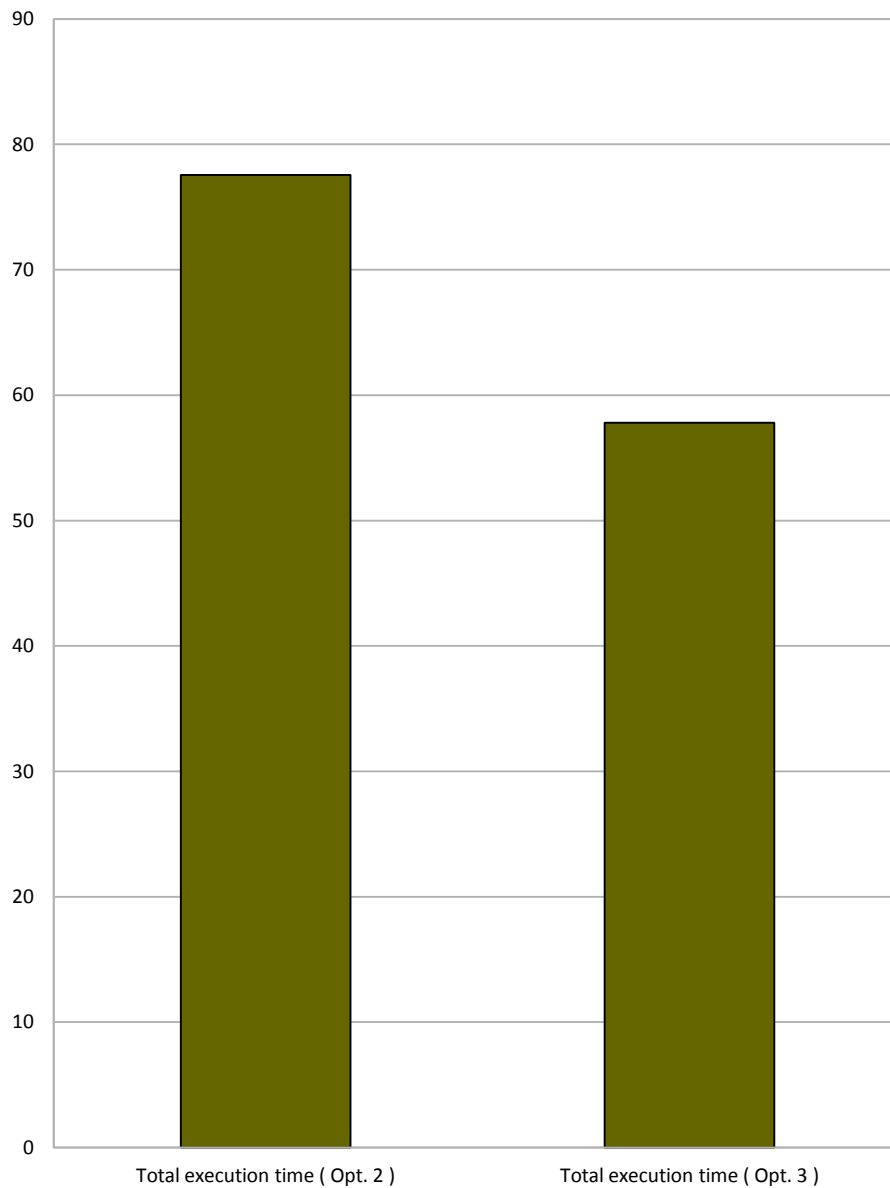
➤ Optimization 3 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,395 sec
- ✓ Backward: 0,4 sec

➤ Speedup

- ✓ BWBaumWelch: 1
- ✓ Forward: 1,542
- ✓ Backward: 1,277

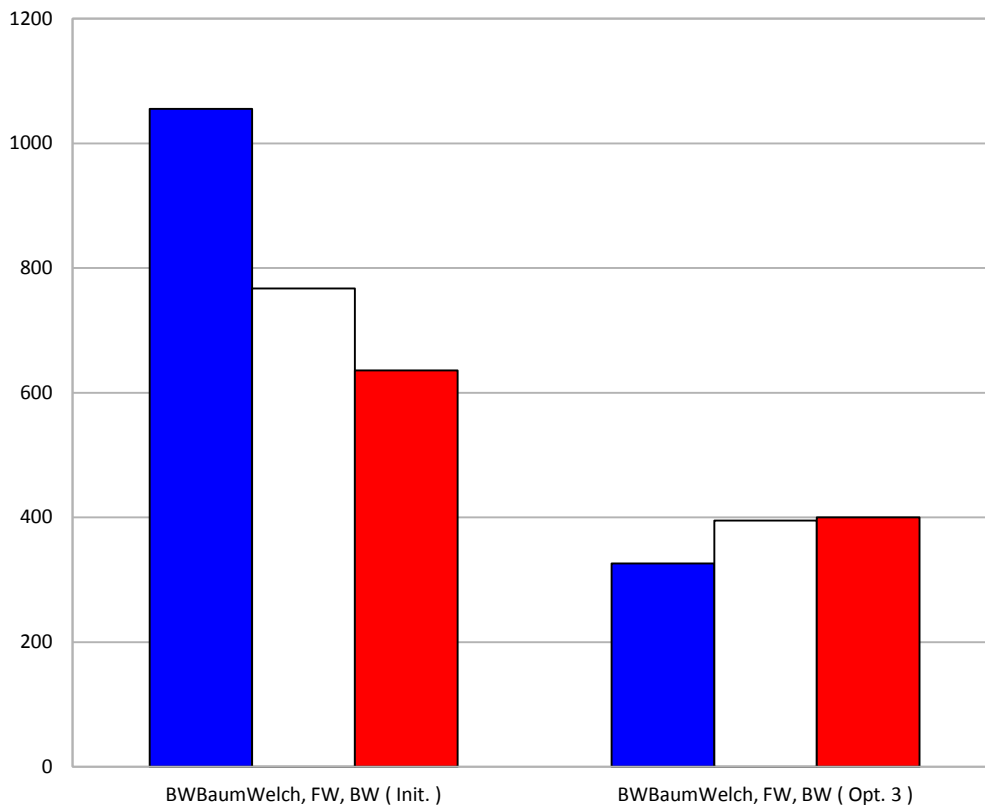
Έπειτα, στο Σχήμα 6.6 παρουσιάζεται η Βελτιστοποίηση_3 σε σχέση με την Βελτιστοποίηση_2, όσον αφορά τους τελικούς χρόνους εκτέλεσης.



Σχήμα 6.6: Βελτιστοποίηση_3 (total exec. time)

- Optimization 2 (GPU)
 - ✓ Total execution time: 77,56 min
- Optimization 3 (GPU)
 - ✓ Total execution time: 57,806 min
- Speedup
 - ✓ Total execution time: 1,342

Στο Σχήμα 6.7 παρουσιάζεται η Βελτιστοποίηση_3 σε σχέση με την αρχική υλοποίηση, όσον αφορά τους kernels. Να σημειωθεί ότι σαν Βελτιστοποίηση_3 εννοείται η συγχώνευση των “Βελτιστοποίηση_3 – παραλληλία πεδίων σε διαγώνιο (βλ. 4.2.5 & 4.3.5)” και “Βελτιστοποίηση (βλ. 5.3)”. Ουσιαστικά, το γράφημα που ακολουθεί αναπαριστά τα τελικά αποτελέσματα που προκύπτουν, με την μεταφορά της FSA στην GPU, όσον αφορά τους kernels.



Σχήμα 6.7: Τελικά αποτελέσματα (kernels)

➤ Initial (CPU)

- ✓ BWBaumWelch: 1,055 sec
- ✓ Forward: 0,767 sec
- ✓ Backward: 0,636 sec

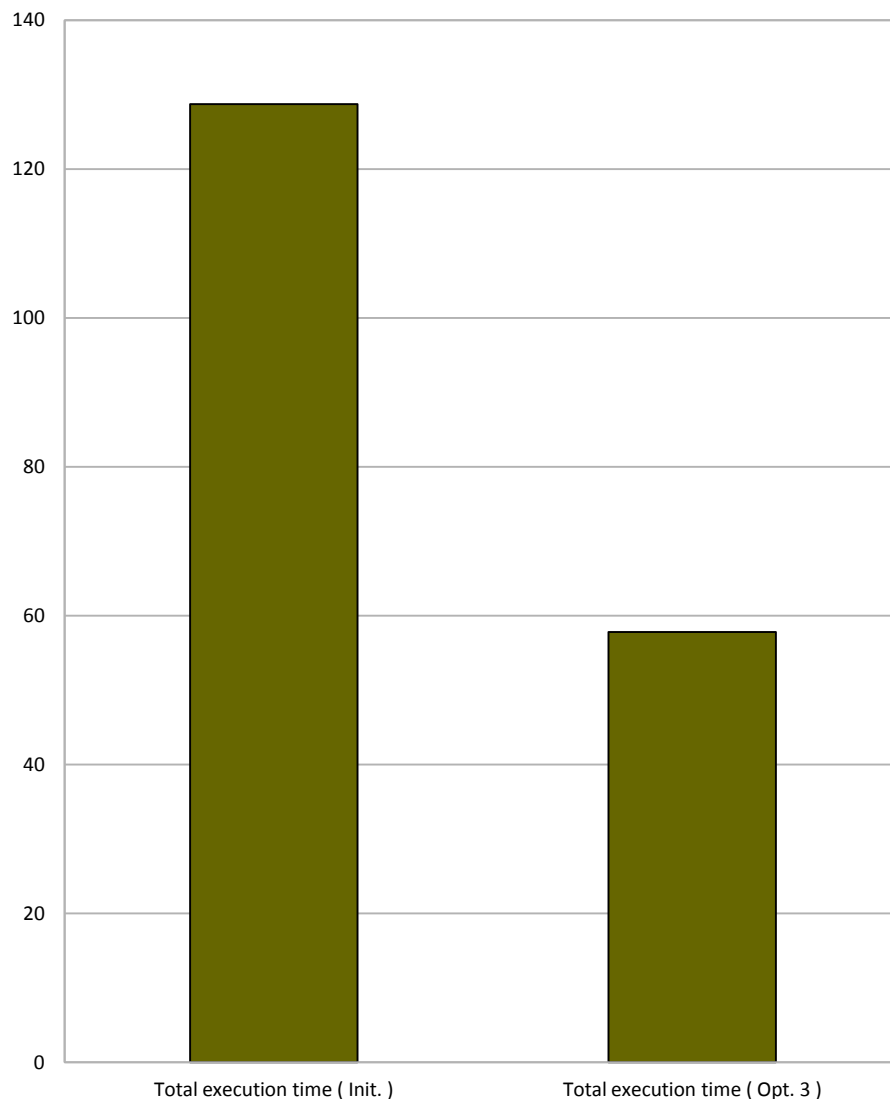
➤ Optimization 3 (GPU)

- ✓ BWBaumWelch: 0,326 sec
- ✓ Forward: 0,395 sec
- ✓ Backward: 0,4 sec

➤ Speedup

- ✓ BWBaumWelch: 3,236
- ✓ Forward: 1,942
- ✓ Backward: 1,59

Στη συνέχεια, στο Σχήμα 6.8 παρουσιάζεται η Βελτιστοποίηση_3 σε σχέση με την αρχική υλοποίηση, όσον αφορά τους τελικούς χρόνους εκτέλεσης. Ουσιαστικά, το γράφημα που ακολουθεί αναπαριστά τα τελικά αποτελέσματα που προκύπτουν, με την μεταφορά της FSA στην GPU, όσον αφορά τους τελικούς χρόνους εκτέλεσης.

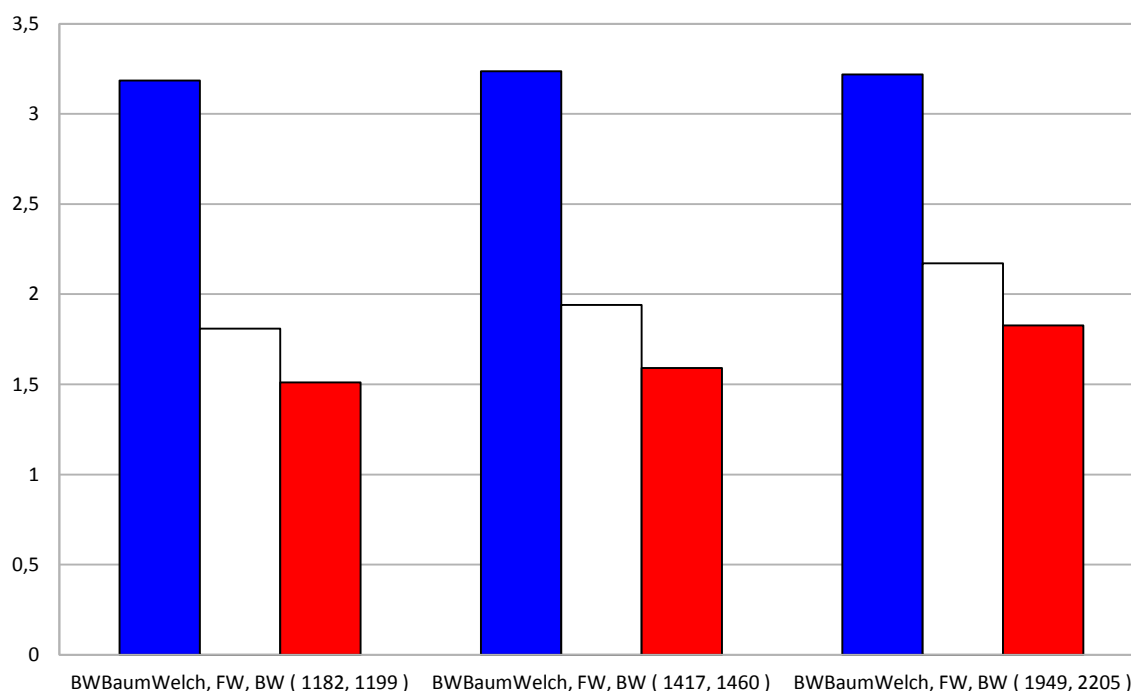


Σχήμα 6.8: Τελικά αποτελέσματα (total exec. time)

- Initial (CPU)
 - ✓ Total execution time: 128,756 min
- Optimization 3 (GPU)
 - ✓ Total execution time: 57,806 min
- Speedup
 - ✓ Total execution time: 2,227

6.3 Γραφική παρουσίαση speedup

Στο Σχήμα 6.9 παρουσιάζεται η μεταβολή του speedup, όσον αφορά τις συναρτήσεις που μεταφέρθηκαν στην GPU (kernels), για διαφορετικά μεγέθη του ζευγαριού σύγκρισης. Η εν λόγω μεταβολή έχει να κάνει με την πιο βελτιστοποιημένη έκδοση της εφαρμογής FSA. Με αυτόν τον τρόπο μπορεί να βγάλει κανείς πολύ χρήσιμα συμπεράσματα σχετικά με την κλιμάκωση και απόδοση της εφαρμογής για προβλήματα μεγαλύτερου μεγέθους.



Σχήμα 6.9: Μεταβολή speedup (kernels)

Με βάση το παραπάνω γράφημα λοιπόν, παρατηρείται ότι οι kernels *“Forward”* και *“Backward”* είναι γρηγορότεροι όσο αυξάνεται το μέγεθος του ζευγαριού σύγκρισης.

Αντίθετα, ο kernel *“BackwardBaumWelch”* είναι λίγο πολύ ανεξάρτητος από το μέγεθος του ζευγαριού σύγκρισης. Κάτι τέτοιο μπορεί να εξηγηθεί λόγω της αναπόφευκτης ύπαρξης των ακολουθιακών τμημάτων που περιλαμβάνει.

Συνεπώς, το συμπέρασμα είναι ότι για αρχεία εισόδου μεγαλύτερου μεγέθους ακολουθιών, επιτυγχάνεται μεγαλύτερο speedup και ως προς τους kernels και ως προς τον συνολικό χρόνο εκτέλεσης της εφαρμογής FSA.

Κεφάλαιο 7

Σύνοψη

7.1 Αναφορά κύριων σημείων

Η ευθυγράμμιση βιολογικών ακολουθιών αποτελεί αναπόσπαστο τμήμα πολλών, αν όχι όλων, των βιολογικών εφαρμογών που υπάρχουν σήμερα. Είναι πολύ σημαντική και πολύ χρονοβόρα διαδικασία επίσης. Η επιτάχυνσή της κρίνεται συνεπώς άκρως επιτακτική και πολύ χρήσιμη, λόγω του τεράστιου και ολοένα αυξανόμενου μεγέθους των βιολογικών βάσεων δεδομένων.

Η εφαρμογή FSA είναι ένα πρόγραμμα ευθυγράμμισης ακολουθιών πρωτεϊνικής, RNA και DNA φύσης, ομόλογων ή μη. Η παρούσα εργασία ασχολήθηκε αποκλειστικά και μόνο με την μελέτη των πρωτεϊνών. Οι μέθοδοι επιλογής υποσυνόλου ζευγαριών προς σύγκριση με μικρή μείωση της ακρίβειας και μηχανικής μάθησης των παραμέτρων σύγκρισης του εκάστοτε HMM, την έκαναν ιδιαίτερα ενδιαφέρουσα προς μελέτη και μεταφορά της στην GPU (μοντέλο CUDA) για την βελτιστοποίησή της.

Η εν λόγω βελτιστοποίηση είχε να κάνει με το εκάστοτε ζευγάρι σύγκρισης, όπου παραλληλοποιήθηκαν συναρτήσεις υπολογισμών “wavefront” μορφής (“Forward”, “Backward”), αλλά και συναρτήσεις που εμπειρεύχαν αναπόφευκτα ακολουθιακά τμήματα μέσα τους (“BackwardBaumWelch”). Με αυτόν τον τρόπο έγιναν ταυτόχρονα και φανερές οι δυσκολίες που αντιμετωπίζουν οι κάρτες γραφικών για την εκμετάλλευση της παραλληλίας που προσφέρουν. Επίσης, οι τεχνικές που παρουσιάστηκαν, μπορούν κάλλιστα να υιοθετηθούν και από άλλα προβλήματα παρόμοιας μορφής.

Για αρχείο εισόδου 73 πρωτεϊνικών ακολουθιών, μέσου μεγέθους ακολουθίας 1452 αμινοξέων, σημειώθηκε επιτάχυνση 2,227. Επίσης, παρατηρήθηκε ότι όσο μεγαλύτερου μεγέθους είναι οι ακολουθίες που πρόκειται να ευθυγραμμιστούν, τόσο μεγαλύτερη επιτάχυνση σημειώνεται και τόσο μικρότεροι είναι οι συνολικοί χρόνοι εκτέλεσης της εφαρμογής που προκύπτουν.

7.2 Μελλοντική επέκταση εφαρμογής

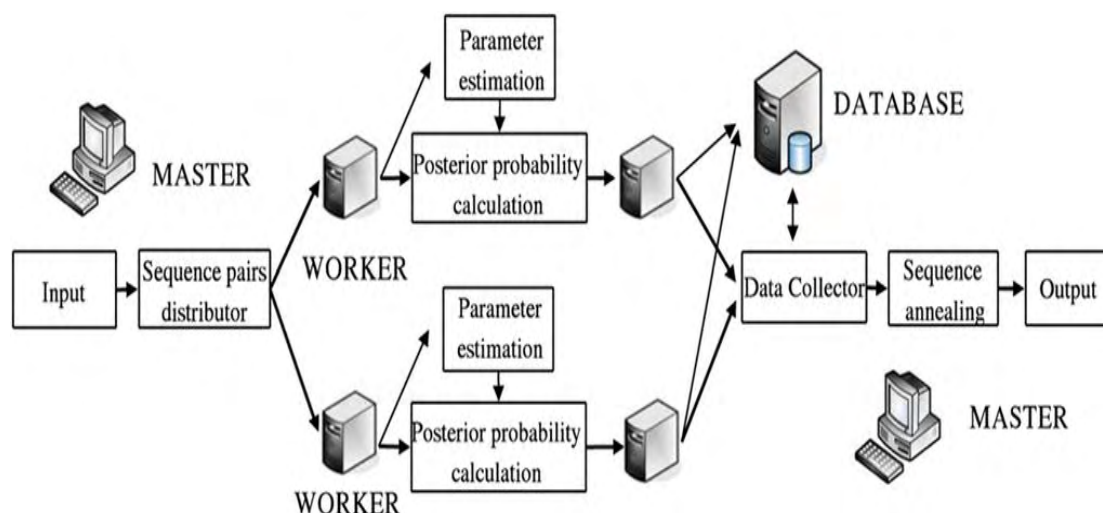
Η προσέγγιση της παρούσας εργασίας, όσον αφορά τον παραλληλισμό, έχει ένα βασικό πλεονέκτημα και ένα βασικό μειονέκτημα.

Το πλεονέκτημα είναι ότι επειδή κάθε φορά η βελτιστοποίηση έχει να κάνει με το εκάστοτε ζευγάρι σύγκρισης, είναι πολύ δύσκολο να υπάρξει πρόβλημα με το μέγεθος της *global memory*, ακόμα και για πολύ απαιτητικές ως προς το μέγεθός

τους ακολουθίες εισόδου. Ακριβώς επειδή η όλη διαδικασία δέσμευσης μνήμης και υπολογισμών στην GPU περιορίζεται στο εκάστοτε ζευγάρι σύγκρισης και όταν ολοκληρωθεί η φάση των υπολογισμών, αποδεσμεύεται η μνήμη που είχε νωρίτερα δεσμευθεί, η *global memory* έχει να κάνει κάθε φορά με το μέγεθος ενός ζευγαριού και όχι περισσοτέρων ή όλων, όπως της επιτρέπει θεωρητικά κάλλιστα η FSA.

Από την άλλη πλευρά, το μειονέκτημα είναι ότι δεν γίνεται εκμετάλλευση της ανεξάρτητης φύσης των ζευγαριών σύγκρισης. Κάθε ζευγάρι μπορεί να συγκριθεί και να υπολογιστούν οι πιθανότητες των στοιχείων του ανεξάρτητα από κάθε άλλο ζευγάρι. Με άλλα λόγια, τα αποτελέσματα της σύγκρισης του ενός δεν επηρεάζουν την ορθότητα των αποτελεσμάτων της σύγκρισης του άλλου. Υπάρχει δηλαδή ένας εμφανέστατος παραλληλισμός, ο οποίος με κάποιο τρόπο θα πρέπει να εκμεταλλευθεί.

Αν υποθετικά, κάθε block της GPU αναλάμβανε ένα ζευγάρι, για μεγάλες ακολουθίες εισόδου και η *global memory* δεν θα επαρκούσε και δεν θα μπορούσε να αξιοποιηθεί αποδοτικά ο παραλληλισμός που έχει κάθε ζευγάρι σύγκρισης. Επομένως, η λύση που θα αξιοποιούσε πολύ αποδοτικά όλες τις μορφές παραλληλισμού που διαθέτει η εφαρμογή FSA εικονίζεται στο Σχήμα 7.1.



Σχήμα 7.1: Μελλοντική λύση (χρήση CUDA + MPI)

Βάσει αυτής, αυτό που θα γίνεται θα είναι ουσιαστικά ένας ομοιόμορφος διαμοιρασμός ζευγαριών σε συστάδα υπολογιστών (cluster computing) που επικοινωνούν μεταξύ τους μέσω δικτύου και διαθέτουν ο κάθε ένας από αυτούς GPU NVIDIA graphic cards. Με αυτόν τον τρόπο θα αξιοποιείται και ο παραλληλισμός κάθε ζευγαριού σύγκρισης στην GPU (χρήση CUDA), αλλά και ο παραλληλισμός της ανεξάρτητης φύσης των ζευγαριών σύγκρισης (χρήση MPI).

Βιβλιογραφία

- [1] Chuong B. Do, Kazutaka Katoh, “Protein Multiple Sequence Alignment”
- [2] Robert K. Bradley, Adam Roberts, Michael Smoot, Sudeep Juvekar, Jaeyoung Do, Colin Dewey, Ian Holmes, Lior Pachter, “Fast Statistical Alignment”
- [3] Robert K. Bradley, Adam Roberts, Michael Smoot, Sudeep Juvekar, Jaeyoung Do, Colin Dewey, Ian Holmes, Lior Pachter, “Fast Statistical Alignment: Text S1”
- [4] “CUDA”
- [5] Tom R. Halfhill, “Looking Beyond Graphics”
- [6] “Intel VTune Performance Analyzer to Optimize Software on Intel Core i7 Processors”
- [7] “Reduction SDK 5.0 NVIDIA”
- [8] “Youtube (ML 14.4) Hidden Markov models (HMMs) (part 1)”,
“Youtube (ML 14.5) Hidden Markov models (HMMs) (part 2)”,
“Youtube (ML 14.6) Forward – Backward algorithm for HMMs”,
“Youtube (ML 14.7) Forward algorithm (part 1)”,
“Youtube (ML 14.8) Forward algorithm (part 2)”,
“Youtube (ML 14.9) Backward algorithm”
- [9] Ashwin M. Aji, Wu-chun Feng, “Accelerating Data-Serial Applications on Data-Parallel GPGPUs: A Systems Approach”
- [10] Shucai Xiao, Wu-chun Feng, “Inter-Block GPU Communication via Fast Barrier Synchronization”