ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΜΟΡΙΑΚΗΣ ΔΥΝΑΜΙΚΗΣ (MOLECULAR DYNAMICS) ΣΕ ΑΝΑΔΙΑΤΑΣΣΟΜΕΝΗ ΛΟΓΙΚΗ"

ΜΑΛΑΒΑΖΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

ΕΠΙΒΛΕΠΩΝ: Αναπληρωτής Καθηγητής Παπαευσταθίου Ιωάννης (ΗΜΜΥ)
Μέλος Επιτροπής: Καθηγητής Απόστολος Δόλλας (ΗΜΜΥ)
Μέλος Επιτροπής: Καθηγητής Διονύσιος Πνευματικάτος (ΗΜΜΥ)

Χανιά, Κρήτη
2016

Ευχαριστίες

Καταρχάς θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Ιωάννη Παπαευσταθίου, αρχικά για την εμπιστοσύνη που μου έδειξε και για τις συμβουλές του καθ' όλη την διάρκεια της διπλωματικής μου εργασίας και έπειτα για τις γνώσεις που έλαβα μέσα από αυτή την διαδικασία.

Επίσης θα ήθελα να ευχαριστήσω όλα τα μέλη του Microprocessor & Hardware Lab (MHL) και ιδιαίτερα τον Παύλο Μαλακωνάκη και τον Κωνσταντίνο Γεωργόπουλο, οι οποίοι με στήριξαν στην διπλωματική αυτή εργασία, με τις συμβουλές τους και τις γνώσεις τους και για την σημαντική τους βοήθεια όποτε την χρειαζόμουν.

Χανιά, Αύγουστος 2016
Μαλαβάζος Κωνσταντίνος

Περίληψη

Η συνεχής ανάπτυξη της επιστήμης των υπολογιστών τα τελευταία χρόνια έχει αλλάξει ριζικά τον τρόπο προσέγγισης των επιστημονικών προβλημάτων. Οι προσομοιώσεις σε υπολογιστή έχουν εξελιχθεί ως τον τρίτο πυλώνα της επιστήμης, γεφυρώνοντας το χάσμα ανάμεσα στην παραδοσιακές δύο διαδικασίες, θεωρία και πείραμα. Ένα τέτοιο παράδειγμα αποτελούν οι προσομοιώσεις Μοριακής Δυναμικής. Η προσομοίωση Μοριακής Δυναμικής βασίζεται στον υπολογισμό διαφόρων δυνάμεων. Οι σημαντικότερες και πιο χρονοβόρες για τον υπολογισμό τους είναι οι long-ranged non-bonded forces. Στο παρελθόν έχουν γίνει διάφορες προσπάθειες επιτάχυνσης αυτού του κομματιού της προσομοίωσης.

Η παρούσα διπλωματική εργασία στοχεύει να επιταχύνει την προσομοίωση Μοριακής Δυναμικής που προσφέρει το πακέτο NAMD 2.10 με την βοήθεια της τεχνολογίας των FPGAs. Το πακέτο αυτό προσφέρει από μόνο του ποικίλους τρόπους επιτάχυνσης της διαδικασίας, όπως με την βοήθεια του CHARM++ ή και μέσω GPU χρησιμοποιώντας CUDA. Για την επίτευξη του στόχου της διπλωματικής εργασίας αυτής χρειάστηκε η μετατροπή του κώδικα από CUDA σε C και έπειτα με την βοήθεια του εργαλείου Vivado HLS 2015.4 να δημιουργηθεί κώδικας περιγραφής υλικού κατάλληλος για την εκάστοτε επιθυμητή FPGA.

Πιο συγκεκριμένα, ο πολυνηματικός κώδικας CUDA που υλοποιούσε τις δυνάμεις αυτές μετατράπηκε σε ένα μονονηματικό κώδικα C ώστε το εργαλείο Vivado HLS να τον δεχτεί. Έπειτα στο Vivado HLS 2015.4 υλοποιήθηκαν μια σειρά από διαφορετικές σχεδιάσεις πάνω σε συγκεκριμένη FPGA και χρησιμοποιήθηκαν μια σειρά από directives που παρέχει το εργαλείο και δημιουργούν βέλτιστο κώδικα περιγραφής υλικού.

Τέλος, οι σχεδιάσεις δεν κατέβηκαν ποτέ πραγματικά σε FPGA αλλά έγιναν πάρα πολλές προσομοιώσεις του κώδικα με το εργαλείο. Τα αποτελέσματα έδειξαν πως το I/O δημιουργεί ιδιαίτερα προβλήματα και χωρίς αυτό το σύστημα μπορεί να πάρει μέχρι και 6,18x speedup σε σχέση με το Software με χρήση CHARM++, που εκτελέστηκε σε ένα σύστημα με CPU Intel Core i5-4690K (4 cores – 4 threads) στα 3.9 GHz και 8 GB RAM και λειτουργικό σύστημα Linux Ubuntu 15.04 64bit.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Εξισώσεων

Κατάλογος Γραφημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Οι αρκετές δεκαετίες συνεχούς ανάπτυξης του κλάδου της πληροφορικής έχουν αλλάξει ριζικά τον τρόπο προσέγγισης των επιστημονικών προβλημάτων. Οι προσομοιώσεις σε υπολογιστή έχουν εξελιχθεί ως τον τρίτο πυλώνα της επιστήμης, γεφυρώνοντας το χάσμα ανάμεσα στην παραδοσιακές δύο διαδικασίες, θεωρία και πείραμα [1]. Με τον συνδυασμό του σωστού υπολογιστικού μοντέλου ενός φυσικού συστήματος με το κατάλληλο υλικό και λογισμικό, είναι τώρα δυνατό να μελετηθεί η συμπεριφορά του συστήματος, χωρίς στην πραγματικότητα να εκτελούνται φυσικά πειράματα.

Οι προσομοιώσεις μοριακής δυναμικής (Molecular Dynamics: MD simulation) είναι ένα τέτοιο εργαλείο που μοντελοποιεί ένα πραγματικό σύστημα σε ατομικό επίπεδο και χρησιμοποιεί την κλασική μηχανική για να μελετήσει την συμπεριφορά του. Το MD λειτουργεί ως ένα εικονικό πείραμα και παρέχει μία προβολή του πειράματος του εργαστηρίου με περισσότερες λεπτομέρειες. Είναι ένα από τα πιο ευρέως χρησιμοποιούμενα εργαλεία στην υπολογιστική επιστήμη και στην μηχανική. Στην βιοϊατρική παρέχει μέχρι στιγμής πολλές ενδιαφέρουσες και σημαντικές πληροφορίες για την κατανόηση της λειτουργικότητας των βιολογικών συστημάτων [2, 3, 4, 5, 6, 7, 8, 9].

Ένα παράδειγμα τέτοιων ευρημάτων βρίσκεται σε μία μελέτη του Severin και λοιπών [78], όπου οι συγγραφείς μελέτησαν την μεθυλίωση του DNA, μια διαδικασία όπου το υδρογόνο άτομο στην 5η θέση μιας βάσης κυτοσίνης αντικαθίστανται από μία ομάδα μεθυλίου (CH3). Η μεθυλίωση διαδραματίζει σημαντικό ρόλο στην έκφραση του γονιδίου, το οποίο με τη σειρά του υπαγορεύει πως ένας ζωντανός οργανισμός προσαρμόζεται στην προστασία των ενδιαιτημάτων και σε ζωτικές λειτουργίες π.χ. διατροφικές συνήθειες. Ο τρόπος που γίνεται η μεθυλίωση ελέγχει την σύνδεση των πρωτεϊνών σε συγκεκριμένες θέσεις του DNA, παρεμποδίζοντας δύο γονίδια, και το οποίο μπορεί να προκαλέσει καρκίνο. Μέχρι πρόσφατα, οι επιστήμονες γνώριζαν μόνο δύο έμμεσες μεθόδους πώς η μεθυλίωση επιδρά στην γονιδιακή έκφραση. Στο έργο αυτό, με την βοήθεια του MD simulation, βρέθηκε ότι θα μπορούσε να υπάρχει ένας τρίτος τρόπος για το πώς η μεθυλίωση μπορεί να ρυθμίσει την γονιδιακή έκφραση πιο άμεσα, μέσω της αλλαγής των μηχανικών ιδιοτήτων του DNA. Ένα τέτοιο εύρημα μπορεί να είναι ζωτικής σημασίας για την κατανόηση της επιγενετικής με βάση την μεθυλίωση, μεταξύ των οποίων την κατανόηση του πώς το σώμα μας προσαρμόζεται στον τρόπο ζωής μας.

Η προσομοίωση στη Μοριακή Δυναμική είναι μία επαναληπτική διαδικασία, όπου προχωρά με χρονικά βήματα (timesteps). Κάθε timestep έχει δύο φάσεις: Υπολογισμός Δύναμης και Ενημέρωση Κίνησης. Οι δυνάμεις που επιδρούν σε κάθε σωματίδιο υπολογίζονται αρχικά χρησιμοποιώντας την κλασική μηχανική και έπειτα η κατάσταση κάθε σωματιδίου ενημερώνεται. Ένα timestep τυπικά αντιστοιχεί σε ένα ή λίγα femtoseconds (fs) του πραγματικού χρόνου. Οι προσομοιώσεις ΜΔ έτσι απαιτούν εκατομμύρια timesteps για την προσομοίωση μόλις λίγων νανοδευτερολέπτων (ns) σε πραγματικό χρόνο.

Μεταξύ όλων των υπολογισμών στην ΜΔ, ο υπολογισμός των δυνάμεων καταναλώνει το μεγαλύτερο μέρος του χρόνου εκτέλεσης. Αν και ο ακριβής υπολογισμός εξαρτάται από το φυσικό σύστημα, το μοντέλο δυνάμεων και άλλων παραμέτρων προσομοίωσης, περιλαμβάνει

και τον υπολογισμό των δυνάμεων που οφείλονται σε διάφορους δεσμούς (π.χ. δεσμούς υδρογόνου), όπως επίσης και των δυνάμεων Van der Waals (VdW) και των ηλεκτροστατικών (ή δυνάμεων Coulomb):

$$F_{total} = F_{bond} + F_{angle} + F_{dihedral} + F_{hydrogen} + F_{vanderWaals} + F_{electrostatic} \quad (1.1)$$

Δεδομένου ότι οι δεσμοί επηρεάζουν μόνο λίγα γειτονικά σωματίδια, η αξιολόγηση των συνδεδεμένων δυνάμεων (bonded forces), έχει υπολογιστική πολυπλοκότητα O(n) όπου n είναι ο συνολικός αριθμός των σωματιδίων στο σύστημα. Η φάση της ενημέρωσης της κίνησης των σωματιδίων ενός timestep είναι επίσης σχετικά απλή και διαρκεί μόνο O(n) χρόνο εκτέλεσης. Από την άλλη πλευρά, η υπολογιστική πολυπλοκότητα των μη-συνδεδεμένων δυνάμεων (non-bonded forces), είναι αρχικά $O(n^2)$, επειδή απαιτείται η αξιολόγηση των αλληλεπιδράσεων μεταξύ όλων των πιθανών ζευγών σωματιδίων. Αν και υπάρχουν διάφορες τεχνικές για να μειωθεί σημαντικά αυτή η πολυπλοκότητα, η ΜΔ παραμένει υπολογιστικά απαιτητική.

Η ίδια η ΜΔ, ευτυχώς, προσφέρει παραλληλισμό της διαδικασίας με χωρική διάσπαση (spatial decomposition) του φυσικού συστήματος. Ευρέως χρησιμοποιούμενα πακέτα ΜΔ μπορούν να επωφεληθούν πλήρως από αυτό και κατά συνέπεια, μπορούν να τρέξουν πλήρως σε συστήματα πολλαπλών CPU [10]. Προσομοιώσεις ΜΔ μπορεί εύκολα να περιλαμβάνουν εκατοντάδες πυρήνων CPU και αρκετούς μήνες εκτέλεσης [11, 12]. Ωστόσο, το μέγεθος και η χρονική πολυπλοκότητα που έχουν επιτευχθεί ακόμα και από τις μεγαλύτερες εκτελέσεις ΜΔ υστερούν σε πολλές τάξεις μεγέθους, συγκρινόμενες με το πραγματικό μέγεθος συστημάτων και γεγονότων που πρέπει να μελετηθούν. Για παράδειγμα, εκτιμάται ότι θα χρειαστούν περίπου 70 χρόνια σε ένα μόνο high-end πυρήνα CPU να προσομοιωθούν μόνο 100 ns ενός βιολογικού συστήματος ενός εκατομμυρίου ατόμων, όπως ο ιός Satellite Tobacco Mosaic (STMV) [11]. Αντίθετα, πολλά σημαντικά βιολογικά φαινόμενα, π.χ., αναδίπλωση πρωτεϊνών (protein folding) (βλέπε Σχήμα 1.1), πιστεύεται ότι συμβαίνουν από μερικά microseconds έως μερικά millisecond [13, 14]. Έτσι, η ταχύτερη εκτέλεση των ΜΔ είναι ένας σημαντικός στόχος για την επιστημονική κοινότητα.

Άλλη μία προσέγγιση για την ταχύτερη εκτέλεση των ΜΔ είναι με την χρήση GPU. Αν και οι GPUs αναπτύχθηκαν αρχικά για γρήγορη απόδοση των γραφικών του υπολογιστή, η τεράστια υπολογιστική ισχύ που προσφέρουν οι σύγχρονες GPUs, μαζί με το χαμηλό κόστος, κυρίως λόγω της παγκόσμιας αγοράς ηλεκτρονικών παιχνιδιών, αποτελούν μια βιώσιμη λύση για την επιστημονική υπολογιστική [15, 16, 17]. Η εισαγωγή των γλωσσών προγραμματισμού υψηλού επιπέδου για GPUs, όπως CUDA και OpenGL, επέτρεψε την ταχεία ανάπτυξη εφαρμογών της μοριακής δυναμικής με χρήση GPUs. Σχεδόν όλες τα διαθέσιμα στο κοινό πακέτα ΜΔ έχουν τώρα GPU-accelerated εκδόσεις [18]. Ενώ μερικοί από αυτούς έχουν επιτύχει σημαντικά speed-ups σε ορισμένες περιορισμένες συνθήκες, οι περισσότεροι έχουν προσπαθήσει να επιτευχθεί εύλογο speed-up για πολύπλοκες περιπτώσεις [19, 20]. Η πιο αξιοσημείωτη εργασία για επιτάχυνση με χρήση GPU της ΜΔ έγινε από την ομάδα NAMD στο Πανεπιστήμιο του Ιλλινόις, όπου επιτεύχθηκε 6x-7x ταχύτητα πάνω από επεξεργαστές [20, 21].

1.1 Αντικείμενο Διπλωματικής

Η διπλωματική εργασία που παρουσιάζεται, αφορά την επιτάχυνση των προσομοιώσεων Μοριακής Δυναμικής του πακέτου NAMD 2.10 με χρήση αναδιατασσόμενης λογικής (FPGA). Η επιτάχυνση επιτυγχάνεται απομονώνοντας το κρίσιμο κομμάτι (critical section) της προσομοίωσης, δηλαδή των μη-συνδεδεμένων δυνάμεων Van der Waals. Σκοπός είναι η εκμετάλλευση της παραλληλίας του αλγορίθμου που προσφέρει το πακέτο NAMD και η εισαγωγή του σε FPGA. Η χρήση της FPGA μας δίνει την δυνατότητα δημιουργίας πολλαπλού υλικού (hardware) το οποίο θα τρέχει παράλληλα. Για την δημιουργία κώδικα περιγραφής υλικού (VHDL, Verilog, SystemC) χρησιμοποιήθηκε το εργαλείο της Xilinx Vivado HLS 2015.4.

1.2 Δομή Διπλωματικής Εργασίας

Η δομή που ακολουθείται στη διπλωματική εργασία είναι η παρακάτω:

Στο *Κεφάλαιο 2* γίνεται μία εισαγωγή στην Μοριακή Δυναμική και στον τρόπο προσομοίωσης. Αναφέρονται παλαιότερες δημοσιεύσεις με παρόμοιες υλοποιήσεις και γίνεται μία αναφορά στα βασικά χαρακτηριστικά των FPGAs.

Στο *Κεφάλαιο 3* γίνεται η περιγραφή της υλοποίησης σε βήματα και η ανάλυση των προβλημάτων που αντιμετωπίστηκαν στο καθένα. Επίσης γίνεται και αναφορά κάποιων βασικών χαρακτηριστικών του εργαλείου Vivado HLS 2015.4 και της γλώσσας CUDA.

Στο *Κεφάλαιο 4* γίνεται η παρουσίαση των αποτελεσμάτων που προέκυψαν από το εργαλείο Vivado HLS, μαζί με τις επιδόσεις των διαφόρων αρχιτεκτονικών.

Στο *Κεφάλαιο 5* εξάγονται κάποια συμπεράσματα, γίνεται μία σύνοψη της διπλωματικής εργασίας και προτείνονται μελλοντικές επεκτάσεις.

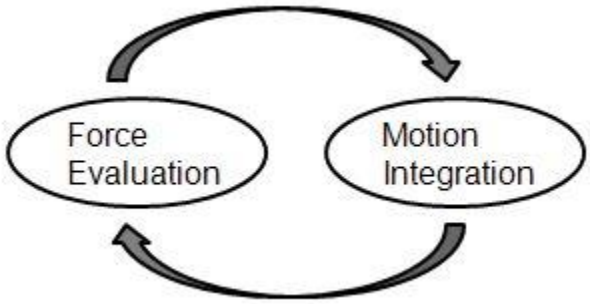
Κεφάλαιο 2

Σχετική Έρευνα

Σε αυτό το κεφάλαιο αναλύεται η προσομοίωση Μοριακής Δυναμικής, για την κατανόηση των εννοιών που θα αναφερθούν στα επόμενα κεφάλαια. Ακόμα, αναφέρονται δημοσιεύσεις με παρόμοιες hardware και software υλοποιήσεις βιολογικών μοντέλων. Τέλος, γίνεται αναφορά στα βασικά χαρακτηριστικά των FPGA.

2.1 Προσομοίωση Μοριακής Δυναμικής (ΜΔ)

Η προσομοίωση Μοριακής Δυναμικής είναι μία επαναληπτική διαδικασία όπου μοντελοποιεί την δυναμική των σωματιδίων εφαρμόζοντας τους νόμους της κλασικής μηχανικής [22, 23]. Η προσομοίωση προχωράει με timesteps, όπου το καθένα αναλύεται σε δύο φάσεις: υπολογισμός δυνάμεων και ενημέρωση κίνησης. Το σχήμα 2.1 δείχνει ένα ΜΔ timestep όπου οι δυνάμεις που επιδρούν σε κάθε σωματίδιο υπολογίζονται αρχικά και έπειτα η κατάσταση του κάθε σωματιδίου ενημερώνεται κατάλληλα.

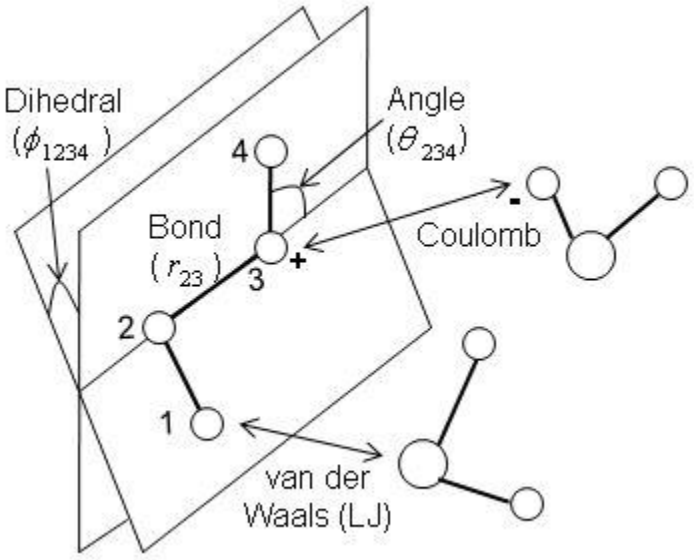


Εικόνα 2.0.1: *ΜΔ timestep*

Το άνω όριο του Δt, δηλαδή του διαστήματος μεταξύ δύο διαδοχικών timesteps, καθορίζεται από τη δόνηση των σωματιδίων και σε μία τυπική προσομοίωση, ένα timestep αντιστοιχεί σε ένα ή λίγα femtoseconds (fs) σε πραγματικό χρόνο. Αυτό σημαίνει ότι εκατομμύρια timesteps πρέπει να προσομοιωθούν για την μελέτη ενός εύλογου χρονικού διαστήματος ενός συγκεκριμένου συστήματος, με αποτέλεσμα να διαρκεί πολύ η προσομοίωση. Ο χρήστης παρέχει την αρχική κατάσταση (θέση, ταχύτητα, κλπ), το μοντέλο δύναμης και άλλες ιδιότητες του συστήματος, και ορισμένες παραμέτρους προσομοίωσης (τύπος προσομοίωσης, συχνότητα εξόδου κ.λπ.), και στη συνέχεια η ΜΔ προσομοιώνει τη δυναμική του συστήματος.
Αυτή την στιγμή, υπάρχουν διάφορα πακέτα ΜΔ (π.χ., AMBER [24], Desmond [25], GROMACS [26], LAMMPS [27], NAMD [28] κλπ.) που είναι ευρέως διαδεδομένα και χρησιμοποιούνται. Υποστηρίζουν διάφορους τύπους πεδίων δυνάμεων (π.χ., AMBER [29], CHARMM [30] κλπ.) και τύπους προσομοιώσεων. Αλλά ανεξάρτητα με το πακέτο ή το μοντέλο δυνάμεων που υποστηρίζουν, ο υπολογισμός δυνάμεων γενικά περιλαμβάνει και τις δυνάμεις Van der Waals, τις ηλεκτροστατικές (Coulomb), και διάφορες ακόμη bonded-forces, όπως φαίνεται και στο σχήμα 2.2 και στην εξίσωση 2.1.

$$F_{total} = F_{bond} + F_{angle} + F_{dihedral} + F_{hydrogen} + F_{vanderWaals} + F_{electrostatic}$$

Εξίσωση 2.1



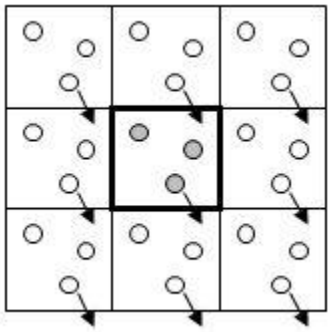
Εικόνα 2.0.2: *Δυνάμεις που δρουν στην ΜΔ*

Οι δυνάμεις Van der Waals και οι ηλεκτροστατικές δυνάμεις κατηγοριοποιούνται στις μη-συνδεδεμένες δυνάμεις (non-bonded forces) ενώ όλες οι υπόλοιπες στις συνδεδεμένες δυνάμεις (bonded forces). Όπως αργότερα θα δούμε στα 2.1.2 και 2.1.3 οι δυνάμεις αυτές χωρίζονται περαιτέρω σε Range-limited non-bonded forces και σε long-range non-bonded forces.

Οι συνδεδεμένες δυνάμεις (bonded forces) επηρεάζουν μερικά γειτονικά άτομα, μπορούν όλες να υπολογιστούν σε χρόνο O(n), όπου n είναι το μέγεθος της προσομοίωσης, δηλαδή ο αριθμός των σωματιδίων στο σύστημα. Από την άλλη πλευρά, οι non-bonded δυνάμεις έχουν εξ ’ορισμού πολυπλοκότητα $O(n^2)$. Παρ ’όλα αυτά υπάρχουν αρκετοί αλγόριθμοι και τεχνικές που μειώνουν την πολυπλοκότητα, μερικές από τις οποίες θα αναλυθούν αργότερα. Στις δυνάμεις Van der Waals, πρακτικά, η πολυπλοκότητα μπορεί να μειωθεί στο O(n) και οι ηλεκτροστατικές στο O(nlog(n)). Η ενημέρωση της κίνησης και κάποιων άλλων δυνάμεων παραμένουν στο O(n). Σε μία τυπική προσομοίωση, η οποία τρέχει σε έναν πυρήνα CPU, ο περισσότερος χρόνος καταναλώνεται στον υπολογισμό των non-bonded δυνάμεων. Στην παράλληλη εκτέλεση της προσομοίωσης, δηλαδή σε πολλαπλές CPU, η επικοινωνία ανάμεσα στους πυρήνες και η ανταλλαγή γίνεται ένας κυρίαρχος παράγοντας του χρόνου, καθώς ο αριθμός των επεξεργαστών αυξάνεται.

2.1.1 Periodic Boundary Condition (PBC) – Περιοδική Οριακή Συνθήκη (ΠΟΣ)

Οι οριακές συνθήκες παίζουν σημαντικό ρόλο στην ΜΔ προσομοίωση, δεδομένου ότι καθορίζουν το περιβάλλον του συστήματος προσομοίωσης. Ο αριθμός των σωματιδίων σε μία τυπική προσομοίωση ΜΔ είναι πολύ μικρότερο από αυτό ενός πραγματικού συστήματος. Ένα πραγματικό σύστημα έχει σωματίδια της τάξης του 10^{23} [11], ενώ ακόμη και η μεγαλύτερη σε μέγεθος προσομοίωση ΜΔ περιέχει μόνο λίγα εκατομμύρια σωματίδια. Μια κοινή πρακτική στη ΜΔ είναι η χρήση της περιοδικής οριακής συνθήκης (ΠΟΣ) [23, 31]. Αντί να θέτουμε ένα συγκεκριμένο όριο, θεωρούμε ότι το ίδιο "κουτί" προσομοίωσης ή κάθε μονάδα κελιού αντιγράφεται στα όρια της κάθε διάστασης. Ένα σωματίδιο στο σύστημα αλληλοεπιδρά όχι μόνο με τα σωματίδια εντός του καθορισμένου κουτιού προσομοίωσης, αλλά και με τα σωματίδια στα αναπαραγόμενα αντίγραφα. Όταν ένα σωματίδιο κινείται σε ένα αρχικό σύστημα, όλα τα αντίγραφα του κινούνται με ομοιόμορφο τρόπο. Όταν ένα σωματίδιο εγκαταλείπει το αρχικό κουτί προσομοίωσης, ένα άλλο σωματίδιο, ένα αντίγραφο του δηλαδή, εισέρχεται σε αυτό. Το σχήμα 2.3 δείχνει μια απλοποιημένη αναπαράσταση δύο διαστάσεων τέτοιων οριακών διελεύσεων. Δεδομένου ότι όλες οι εικόνες είναι απλώς επαναλήψεις του αρχικού σωματιδίου, κρατώντας τα δεδομένα της αρχικής κατάστασης του σωματιδίου είναι αρκετά για να τρέξει η προσομοίωση μόνο. Έτσι, η ΠΟΣ επιτρέπει σε μία προσομοίωση ΜΔ να εκτελεστεί χρησιμοποιώντας ένα σχετικά μικρό αριθμό των σωματιδίων κατά τέτοιο τρόπο, ώστε στα σωματίδια να ασκούνται δυνάμεις σαν να ήταν σε ένα πραγματικό διάλυμα.



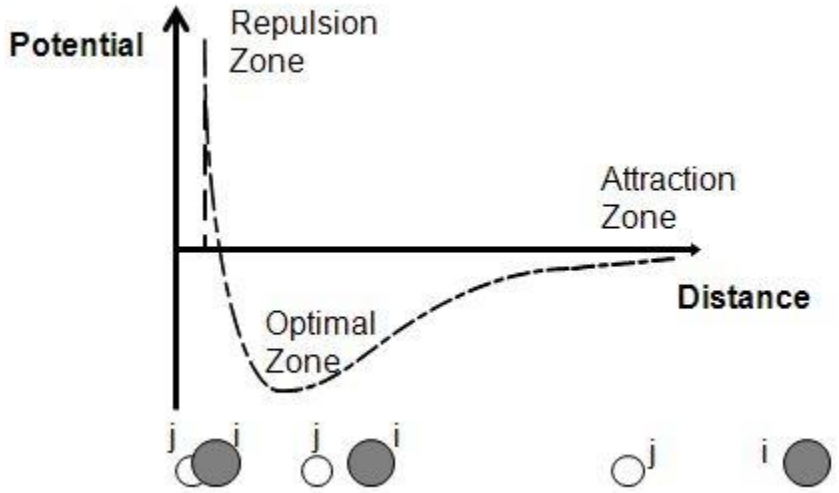
Εικόνα 2.0.3: *Αναπαράσταση 2 διαστάσεων όπου σωματίδια διασχίζουν τα όρια της προσομοίωσης με βάση την ΠΟΣ.*

2.1.2 Van der Waals (VdW) ή Lennard-Jones (LJ) Force

Οι Van der Waals (VDW) δυνάμεις είναι οι δυνάμεις έλξης (ή άπωσης) που λειτουργούν ανάμεσα σε ένα ζεύγος ατόμων που δεν είναι συνδεδεμένα [23]. Οι δυνάμεις αυτές προσεγγίζονται από το δυναμικό Lennard-Jones (LJ) όπως φαίνεται στο σχήμα 2.4. Η εξίσωση 2.2 δίνει την δύναμη LJ που δρα στο σωματίδιο i. Εδώ, τα ε_{ab} και σ_{ab} είναι οι παράμετροι που σχετίζονται με τους τύπους των σωματιδίων και r_{ji} είναι η απόσταση μεταξύ του σωματιδίου i και του σωματιδίου j.

$$\overrightarrow{F}_i(LJ) = \sum_{i \neq j} \frac{\epsilon_{ab}}{\sigma_{ab}^2} \left\{ 12 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{14} - 6 \left(\frac{\sigma_{ab}}{|r_{ji}|} \right)^{8} \right\} \overrightarrow{r_{ji}}$$

Εξίσωση 2.2



Εικόνα 2.0.4: *Δυναμική Lennard-Jones.*

Μια πλήρης αξιολόγηση των VDW ή LJ δυνάμεων απαιτεί αξιολόγηση των αλληλεπιδράσεων μεταξύ όλων των ζευγών σωματιδίων στο σύστημα. Η υπολογιστική πολυπλοκότητα είναι, ως εκ τούτου, O (n^2), όπου n είναι ο αριθμός των σωματιδίων στο σύστημα. Ένας κοινός τρόπος για να μειωθεί αυτή η πολυπλοκότητα είναι η εφαρμογή μιας τιμής-απόστασης αποκοπής. Η LJ δύναμη εξασθενεί γρήγορα ανάλογα με την απόσταση ενός ζεύγους σωματιδίων και συνήθως αγνοείται όταν τα δύο σωματίδια διαχωρίζονται πέρα από κάποια απόσταση αποκοπής. Μια τυπική απόσταση αποκοπής στη ΜΔ βιολογικών συστημάτων βρίσκεται μεταξύ 8-16 Angstroms. Για να εξασφαλιστεί η ομαλή μετάβαση στην απόσταση αποκοπής, μια πρόσθετη λειτουργία χρησιμοποιείται συχνά πάρα πολύ. Χρησιμοποιώντας μια απόσταση αποκοπής μόνη της δεν μειώνει την πολυπλοκότητα υπολογισμού της LJ δύναμης, επειδή όλα τα ζεύγη των σωματιδίων θα πρέπει ακόμα να ελεγχθούν για να διαπιστωθεί αν είναι εντός της απόστασης αποκοπής. Η πολυπλοκότητα μειώνεται σε O(n) με την χρήση τεχνικών όπως η μέθοδος cell-list ή neighbor-list, η οποία θα περιγραφεί στο 2.1.5.

2.1.3 Electrostatic or Coulomb Force – Ηλεκτροστατικές ή Coulomb Δυνάμεις

Οι ηλεκτροστατικές ή Coulomb δυνάμεις μεταξύ δύο φορτισμένων σωματιδίων δίνεται από την Εξίσωση 2.3. Εδώ το q_i και q_j είναι τα φορτία των σωματιδίων i και j αντίστοιχα και το r_{ij} είναι η απόσταση μεταξύ τους [23].

$$\vec{F}_i(CL) = q_i \sum_{i \neq j} \left(\frac{q_j}{|r_{ji}|^3} \right) \vec{r_{ji}}$$

Εξίσωση 2.3

Η Εξίσωση 2.3 έχει πολυπλοκότητα $O(n^2)$, επειδή απαιτεί αξιολόγηση όλων των πιθανών ζευγών σωματιδίων στο σύστημα. Αρκετοί αποδοτικοί αλγόριθμοι έχουν αναπτυχθεί για να υπολογίζουν τη δύναμη αυτή χωρίς υπολογίζονται όλα τα ζεύγη σωματιδίων [32, 33, 34, 35, 36]. Μερικά από αυτά θα παρουσιαστούν εδώ, με ιδιαίτερη έμφαση στην μέθοδο PME (Particle Mesh Ewald), δεδομένου ότι αυτή χρησιμοποιείται ευρέως στη ΜΔ.

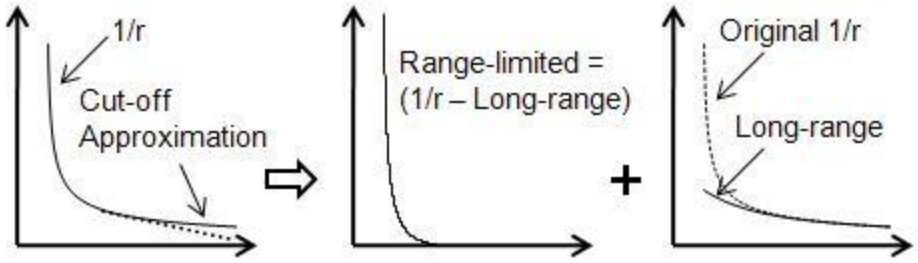
Ewald Summation: Η μέθοδος Ewald Summation (ή Ewald Sums) είναι ένας αλγόριθμος που χρησιμοποιείται για τον υπολογισμό των ηλεκτροστατικών δυνάμεων σε ένα σύστημα με περιοδικές οριακές συνθήκες [22]. Αναπτύχθηκε αρχικά το 1921 για υπολογίσει την ηλεκτροστατική ενέργεια ιοντικών κρυστάλλων [37]. Σε σχέση με τον αρχικό υπολογισμό που είναι $O(n^2)$, η μέθοδος Ewald Summation μειώνει την πολυπλοκότητα στο $O(n^{3/2})$.

Για ένα περιοδικό σύστημα n-σωματιδίων, η συνεισφορά Coulomb για δυναμική ενέργεια μπορεί να εκφραστεί χρησιμοποιώντας την εξίσωση 2.4.

$$E_i = \frac{1}{2} \sum_{n}^{'} \sum_{(i,j)=1}^{N} \frac{q_i q_j}{|r_{ji} + nL|}$$

Εξίσωση 2.4

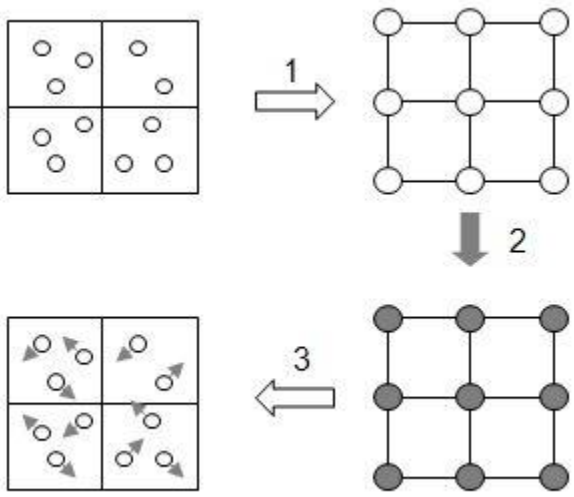
Το άκρο του αθροίσματος σημαίνει πως το άθροισμα βρίσκεται πάνω από τον αριθμό όλων των περιοδικών αντιγράφων και των σωματιδίων εκτός της *περίπτωσης i = j* εάν *n = 0*. Αυτό σημαίνει πως ένα σωματίδιο δεν αλληλοεπιδρά με τον εαυτό του αλλά με τα περιοδικά αντίγραφά του. Το L είναι η διάσταση του κελιού. Το ηλεκτροστατικό δυναμικό του συστήματος διαιρείται σε δύο κύρια τμήματα όπως φαίνεται στο σχήμα 2.5. Το range-limited τμήμα υπολογίζεται σε με πολυπλοκότητα O(n) και το long-range limited σε $O(n^{3/2})$.



Εικόνα 2.0.5: *Διάσπαση των ηλεκτροστατικών δυνάμεων σε range-limited και long-range limited.*

Particle Mesh Ewald (PME) Method: Το PME είναι μία αποτελεσματική τεχνική η οποία χρησιμοποιείται ευρέως για τον υπολογισμό του προτύπου Ewald Sums, λόγω της υπολογιστικής της αποδοτικότητας [33, 35]. Η τεχνική αυτή προσεγγίζει τα range ή τα long-range limited τμήματα του Ewald Sums με μια διακριτή συνέλιξη σε ένα πλέγμα παρεμβολής, χρησιμοποιώντας την διακριτή 3D Fast Fourier Transformation (FFT), μειώνοντας την υπολογιστική πολυπλοκότητα από $O\ (n^{3/2})$ σε $O(nlog(n))$. Χρειάζεται προσεκτική αξιολόγηση του συστήματος παρεμβολής και του μεγέθους του πλέγματος, ώστε να επιτευχθεί υψηλή ταχύτητα και ακρίβεια στην προσομοίωση. Η βασική διαδικασία της PME απεικονίζεται στο σχήμα 2.6 και αποτελείται από τρία στάδια όπως ακολουθούν [38]:

1. Ανάθεση των φορτίων των σωματιδίων στα σημεία του πλέγματος.
2. Υπολογισμός δυνάμεων με FFT και αντίστροφο FFT (rFFT).
3. Παρεμβολή των δυνάμεων πίσω στα σωματίδια.



Εικόνα 2.0.6: *Βήματα PME.*

Η πολυπλοκότητα του πρώτου και του τρίτου βήματος είναι O(n), ενώ εκείνη του δεύτερου είναι O(nlog(n)), η οποία κυριαρχεί στο συνολικό υπολογισμό. Αξίζει να σημειωθεί ότι μερικές φορές, για να εξοικονομηθεί χρόνος υπολογισμού, το τμήμα long-range limited του PME υπολογίζεται μόνο κάθε λίγα timesteps, π.χ., κάθε 4 timesteps. Ένα βελτιωμένο σύστημα του PME, που ονομάζεται Smooth PME (SPME) αναφέρεται στο [34].

2.1.4 Άλλες Δυνάμεις

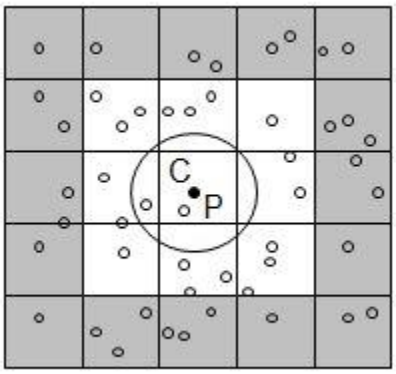
Εκτός των non-bonded δυνάμεων, οι συνδεδεμένες αλληλεπιδράσεις (π.χ. δεσμός, γωνία κλπ. όπως στο σχήμα 2.2) πρέπει να υπολογίζονται σε κάθε timestep. Οι αλληλεπιδράσεις αυτές έχουν υπολογιστική πολυπλοκότητα *O(n)* και καταναλώνουν μικρό τμήμα του χρόνου σε κάθε timestep. Τα συνδεδεμένα ζεύγη γενικά εξαιρούνται από τους υπολογισμούς των non-bonded δυνάμεων.

2.1.5 Cell-list vs. Neighbor-list

Το κάθε σωματίδιο πρέπει να αλληλοεπιδρά με μερικά από τα γειτονικά σωματίδια ώστε να υπολογιστούν οι range-limited non-bonded δυνάμεις. Αυτό απαιτεί μία διαδικασία η οποία πρέπει να λαμβάνει υπόψιν όλα τα γειτονικά σωματίδια κάθε σωματιδίου. Μία απλή προσέγγιση θα επαναλαμβανόταν για όλα τα σωματίδια στο σύστημα ώστε να βρεθούν οι γείτονες του καθενός. Αυτή είναι μία αναποτελεσματική εφαρμογή με πολυπλοκότητα $O(n^2)$. Παρακάτω αναλύονται δύο αποτελεσματικές μέθοδοι.

Cell-list: Στην cell-list μέθοδο [23, 39, 40], ένα κουτί προσομοίωσης αρχικά διαχωρίζεται σε μερικά κελιά, κυβικού σχήματος. Κάθε διάσταση τυπικά επιλέγεται να είναι ελαφρά μεγαλύτερη από την απόσταση αλληλεπίδρασης αποκοπής. Αυτό σημαίνει, για ένα σύστημα 3D (2D), που διέρχονται από μέσα τα σωματίδια του αρχικού κελιού, 26 για 3D (8 για 2D) γειτονικά κελιά θα είναι αρκετά. Εάν χρησιμοποιείται ο τρίτος νόμος του Νεύτωνα, τότε τα μισά μόνο από τα γειτονικά κελιά πρέπει να ελέγχονται. Αν η διάσταση του κελιού είναι μικρότερη από την απόσταση αποκοπής, τότε θα πρέπει να ελέγχεται μεγαλύτερος αριθμός κελιών.

Το κόστος κατασκευής cell-list κλιμακώνεται γραμμικά με τον αριθμό των σωματιδίων. Η σάρωση κάθε σωματιδίου στο σύστημα και η ανάθεσή του σε ένα αντίστοιχο κελί, μία φορά πριν από τον υπολογισμό των δυνάμεων, είναι επαρκής. Η πρόσθετη εργασία για την κατασκευή των cell-lists αποδίδει καλά, επειδή η πολυπλοκότητα του υπολογισμού δυνάμεων γίνεται τώρα *O(n).* Στο σχήμα 2.7 φαίνεται μια δισδιάστατη απεικόνιση του cell-list του σωματιδίου ‘P’ στο κελί ‘C’.



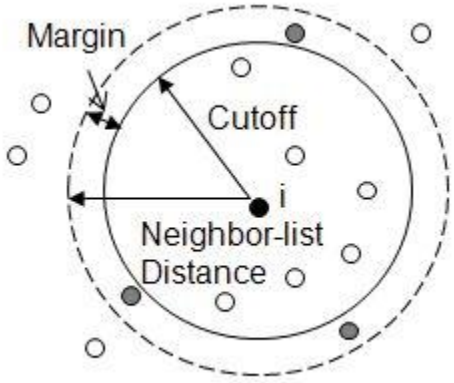
Εικόνα 2.0.7: *Δισδιάστατη απεικόνιση του cell-list του σωματιδίου 'P' στο κελί 'C'.*

Neighbor-list: Αν και στην μέθοδο cell-list η πολυπλοκότητα του υπολογισμού range-limited non-bonded δυνάμεων μειώνεται, εξακολουθείται να ελέγχονται πολλά περισσότερα σωματίδια από όσο χρειάζεται. Για κάθε σωματίδιο πρέπει μόνο να ελεγχθεί ο περιβάλλοντας όγκος του (4/3) x 3.14 x R_c^3, όπου R_c είναι η ακτίνα αποκοπής. Αλλά στην μέθοδο cell-list καταλήγουμε στον έλεγχο ενός όγκου 27 x R_c, το οποίο είναι 6 φορές μεγαλύτερος από αυτό που χρειάζεται. Αυτό μπορεί να βελτιωθεί με τη μέθοδο Neighbor-list [23, 41]. Στη μέθοδο αυτή, μία λίστα των πιθανών γειτονικών σωματιδίων διατηρείται για κάθε σωματίδιο και μόνο αυτή η λίστα ελέγχεται για τον υπολογισμό δυνάμεων. Ένα σωματίδιο περιλαμβάνεται στην Neighbor-list ενός άλλου σωματιδίου εάν η απόσταση μεταξύ τους είναι μικρότερη από $R_c + R_m$, όπου R_m είναι ένα μικρό περιθώριο. Το R_m επιλέγεται έτσι ώστε η Neighbor-list να περιλαμβάνει επίσης τα σωματίδια τα οποία δεν είναι ακόμα στην περιοχή αποκοπής, αλλά μπορεί να εισέλθουν σε αυτήν πριν η λίστα ενημερωθεί. Σε κάθε timestep, η κάθε Neighbor-list ελέγχεται πριν χρησιμοποιηθεί στον υπολογισμό των δυνάμεων. Οι neighbor-lists ενημερώνονται συνήθως τακτά ανά ένα σταθερό χρονικό διάστημα ή όταν ο αριθμός των μετατοπίσεων των σωματιδίων υπερβαίνουν μία προκαθορισμένη τιμή.

Μία neighbor-list μπορεί να κατασκευαστεί για όλα τα σωματίδια σε O(n) χρόνο χρησιμοποιώντας cell-list. Όσο η neighbor-list δεν ενημερώνεται πολύ συχνά, το οποίο είναι γενικά εύκολο να εξασφαλιστεί, αυτή η μέθοδος μειώνει τον υπολογισμό των range-limited δυνάμεων σημαντικά. Η εξοικονόμηση χρόνου εκτέλεσης που επιτυγχάνεται έρχεται με το κόστος του επιπλέον χώρου αποθήκευσης που απαιτείται για να σώσει τη neighbor-list του κάθε σωματιδίου. Για τους περισσότερους high-end επεξεργαστές, αυτό δεν είναι ένα σημαντικό πρόβλημα.

Όπως φαίνεται στο Σχήμα 2.8, για το σωματίδιο i, όλα τα σωματίδια εντός της απόστασης της neighbor-list, εκτός από το ίδιο το σωματίδιο, περιλαμβάνονται στην neighbor-list του και τα σωματίδια που βρίσκονται ανάμεσα στο περιθώριο. Παρόλο που η μείωση της συχνότητας

ενημέρωσης της neighbor-list (αυξάνοντας το περιθώριο-margin) θα μπορούσε να μειώσει το υπολογιστικό κόστος κατασκευής των neighbor-lists, θα οδηγούσε σε μείωση της αποτελεσματικότητας δεδομένου ότι θα προστίθονταν περισσότερα σωματίδια στη λίστα από ό,τι πραγματικά χρειάζονται.



Εικόνα 2.0.8: *Σφαίρα Neighbor-list.*

2.2 Παρόμοιες δουλειές

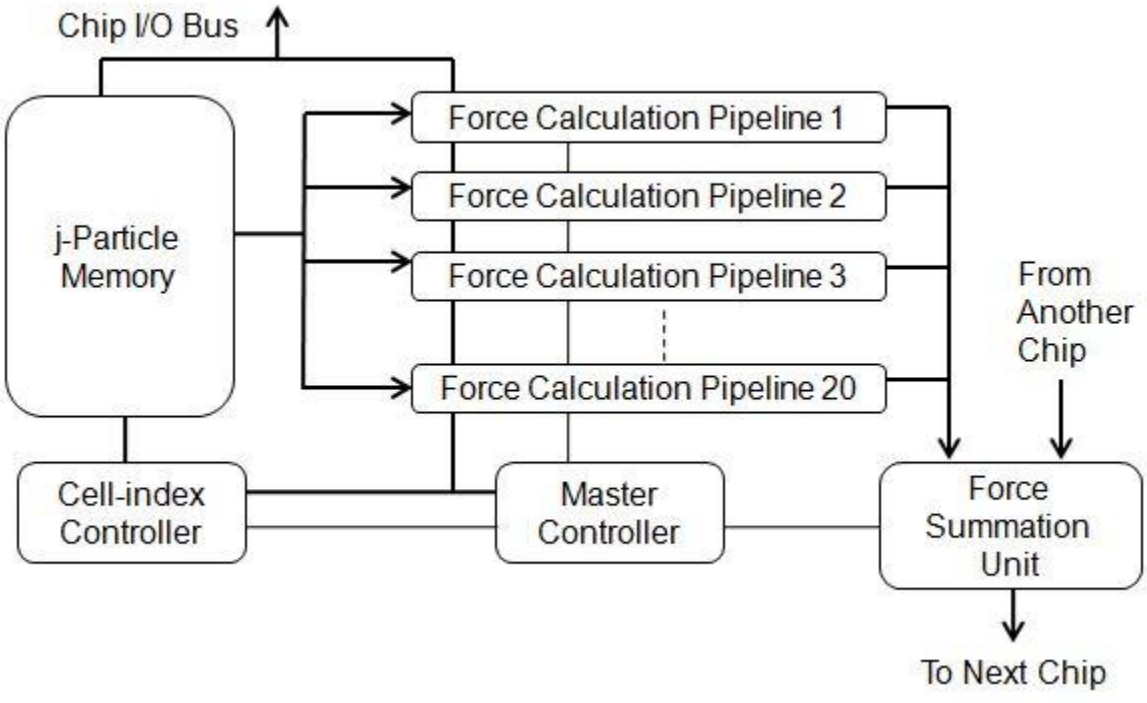
2.2.1 ASIC Acceleration

Η επιτάχυνση με την χρήση τεχνολογίας ASIC ξεκίνησε το 1990, όπου μία ομάδα Ιαπώνων ερευνητών δημιούργησαν την μηχανή ΜΔ, η οποία αποτελούταν custom επεξεργαστές, που ανέπτυξαν οι ίδιοι, και λειτουργούσαν παράλληλα για να προσομοιώσουν ένα σύστημα [8, 168]. Κάθε επεξεργαστής που ονομαζόταν (MODEL: MOlecular Dynamics processing ELement) είχε ένα ενσωματωμένο pipeline για τον υπολογισμό των non-bonded δυνάμεων. Οι δυνάμεις και τα virials υπολογίζονταν με αρκετά καλή ακρίβεια για πραγματικές προσομοιώσεις ΜΔ και μία ταχύτητα της τάξεως 50x επιτεύχθηκε σε σύγκριση τον επεξεργαστή UltraSPARC-I Sun Ultra-2 (200 MHz), για την προσομοίωση ενός μορίου πρωτεΐνης RAS p21 βυθισμένο σε μία σφαίρα νερού (13.258 σωματίδια). Παρακάτω αναλύονται άλλες δύο αρχιτεκτονικές σε τεχνολογία ASIC, οι οποίες είναι πιο πρόσφατες.

MD-GRAPE: Το σύστημα MDGRAPE-3, επίσης γνωστό ως “Protein Explorer”, είναι ένας ειδικού σκοπού peta-flops υπολογιστής με επιταχυντή υλικού για την κλασική προσομοίωση μοριακής δυναμικής [117, 141, 166, 167]. Αναπτύχθηκε στο RIKEN (RIkagaku KENkyujo: Institute of Physical and Chemical Research), στην Ιαπωνία το 2006, σε συνεργασία της SGI Ιαπωνίας και της Intel Ιαπωνίας. Η αρχιτεκτονική του είναι παρόμοια με αυτή των προηγουμένων, the GRAPE

(GRAvity PipE) system, το οποίο αναπτύχθηκε αρχικά για την προσομοίωση της βαρύτητας n-σωμάτων και έπειτα επεκτάθηκε στην επιτάχυνση προσομοιώσεων κλασικής ΜΔ [42, 43, 44, 45, 46, 47, 48, 49].

Το MDGRAPE-3 αποτελείται από 201 μονάδες των 24 MDGRAPE-3 chips, 64 παράλληλους servers με επεξεργαστές Intel Xeon 5000-series (codename Dempsey) και άλλους 37 παράλληλους servers με Intel Xeon στα 3.2 GHz (2 MB L2 caches). Κάθε MDGRAPE-3 board αποτελείται από 12 MDGRAPE-3 chips, το καθένα είχε 20 pipelines υπεύθυνα για τον υπολογισμό των non-bonded δυνάμεων. Οι υπόλοιποι υπολογισμοί τους αναλάμβαναν οι κεντρικοί υπολογιστές (host computers). Κάθε chip χωρούσε 32.768 σωματίδια με 165 GFlops στα 250 MHz (230 GFlops στα 350MHz). Μία από τις βασικές καινοτομίες στην MDGRAPE-3 chip, όπως φαίνεται στο Σχήμα 2.9, είναι η ικανότητα broadcasting των δεδομένων των σωματιδίων στα pipelines που υπολογίζουν τις non-bonded δυνάμεις, η οποία μειώνει σημαντικά τις απαιτήσεις εύρους ζώνης της μνήμης.



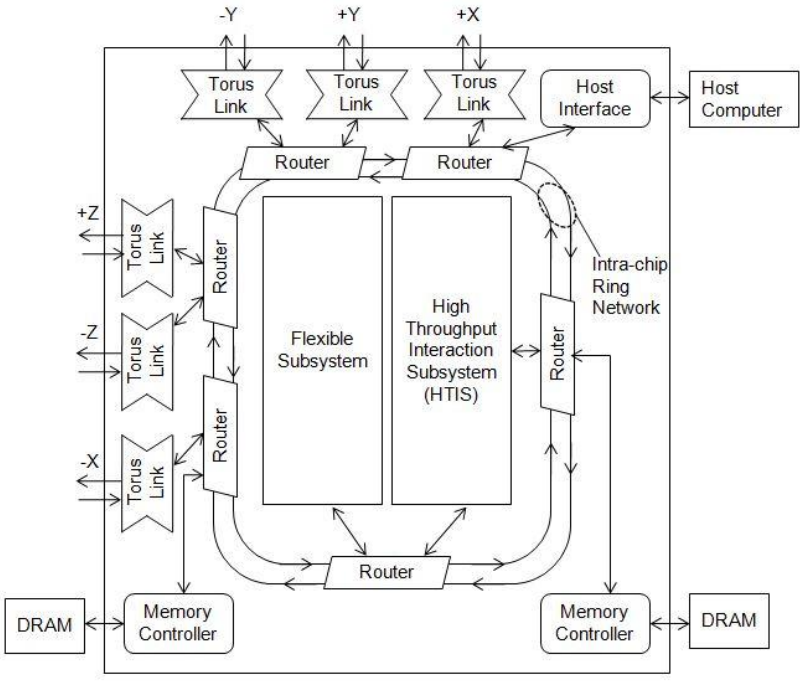
Εικόνα 2.9: *Block diagram of MDGRAPE-3 ASIC*

Η γενιά 130 nm του MDGRAPE-3 chip ήταν η ταχύτερη προσομοίωση ΜΔ σε LSI και ολόκληρο το σύστημα είχε κατανάλωση ισχύος 200 kilowatts ανά ώρα (19 Watt στα 350 MHz ανά chip).

Anton: Το Anton αναπτύχθηκε στο D. E. Shaw Research στην Νέα Υόρκη το 2008, και αποτελεί έναν ειδικού σκοπού υπερυπολογιστή σχεδιασμένο για προσομοιώσεις ΜΔ

βιομοριακών συστημάτων [50, 51]. Η έκδοση των 512 κόμβων φέρεται να έχει επιτύχει προσομοιώσεις της τάξεως των milliseconds, δύο τάξεις μεγέθους δηλαδή πάνω από τις προηγούμενες τεχνολογίες [52]. Όπως φαίνεται στο Σχήμα 2.10, κάθε κόμβος του Anton περιλαμβάνει ένα ASIC με δύο κύρια υπολογιστικά υποσυστήματα, το high-throughput interaction subsystem (HTIS) το οποίο υπολογίζει τις range-limited αλληλεπιδράσεις ζευγών χρησιμοποιώντας 32 pipelines 26-σταδίων, και το flexible subsystem, το οποίο αναλαμβάνει τους υπόλοιπους υπολογισμούς. Το flexible subsystem περιέχει οκτώ ειδικά σχεδιασμένα πυρήνες γεωμετρίας (GCS) για αριθμητικούς υπολογισμούς, τέσσερις επεξεργαστές Tensilica LX για τον έλεγχο της συνολικής ροής δεδομένων, και τέσσερα engines μεταφορά δεδομένων. Τα Anton ASICs υλοποιούνται σε τεχνολογία 90 nm και είναι χρονισμένα στα 485 MHz, με εξαίρεση των pipelines στην HTIS, τα οποία είναι χρονισμένα στα 970 MHz.

Ορισμένες από τις βασικές καινοτομίες στο Anton είναι η μέθοδος διαχωρισμού μιας ουδέτερης περιοχής, για τον υπολογισμό των range-limited non-bonded δυνάμεων, αναδιατάσσοντάς τα δεδομένα επικοινωνίας σε πραγματικό χρόνο. Η επικοινωνία με χαμηλό latency που επιτυγχάνεται στο Anton επιτρέπει πολύ γρήγορη end-to-end επικοινωνία μεταξύ των κόμβων στο λογισμικό (σε κάποιες εκατοντάδες nanoseconds), το οποίο αποδεικνύεται ότι είναι πολύ χρήσιμο για υπολογισμούς βασισμένους σε 3D FFT για long-range ηλεκτροστατικές δυνάμεις [53, 54, 55].



Εικόνα 2.10: *Block diagram of an Anton processing node*

GPU - NAMD (NAnoscale Molecular Dynamics): Το NAMD αναπτύχθηκε από την ομάδα Theoretical and Computational Biophysics Group of UIUC [6, 28, 56]. Είναι ένα πακέτο

προσομοίωσης το οποίο είναι ευρέως διαδεδομένο και έχει αναπτυχθεί για αρκετές πλατφόρμες [10, 57]. Μία από τα χαρακτηριστικά του είναι η έκδοση που υποστηρίζει επιτάχυνση με GPUs και επιτυγχάνει 6x με 7x speed-up σε σύστημα με τετραπύρηνο CPU/GPU, σε μεγάλες και περίπλοκες προσομοιώσεις [20, 21]. Το NAMD διατηρεί την αρχική δομή που λειτουργεί πάνω σε μία μόνο CPU και επιτυγχάνει επιτάχυνση στον υπολογισμό των range-limited non-bonded δυνάμεων πάνω στην GPU. Παρ' όλα αυτά η επιτάχυνση περιορίζεται από το νόμο του Amdahl. Για την χρήση του πακέτου με τη υποστήριξη GPU acceleration χρειάζεται μία high-end NVIDIA κάρτα γραφικών.

2.2.3 FPGA Acceleration

Early Work: Οι παλαιότερες αναφορές για προσομοιώσεις ΜΔ σε FPGAs ξεκινούν το 2003, όπου ο υπολογισμός ταχύτητας και θέσης με τον αλγόριθμο Velocity Verlet σχεδιάστηκε για FPGA [58]. Οι υπολογισμοί έγιναν με το πρότυπο IEEE 754 32-bit floating point arithmetic. Η απόδοση παρουσιάστηκε για δύο τύπους FPGA, την Altera Stratix ES1S80 που πέτυχε 5.69 GFLOPS και στην Xilinx Virtex-II Pro που πέτυχε αντίστοιχα 4.47 GFLOPS.

Το 2004 στο πανεπιστήμιο του Toronto δημιούργησαν μία πρώτη έκδοση ενός συστήματος ΜΔ πάνω στο Transmogrifier 3 (TM3) system [59, 60]. Το TM3 περιείχε τέσσερις συσκευές Virtex-E 2000E οι οποίες ήταν συνδεδεμένες η μία με την άλλη με 98-bit bi-directional buses. Κάθε FPGA ήταν συνδεδεμένη σε μία 256K x 64 bit external SRAM αφιερωμένη για την κάθε FPGA. Ο υπολογισμός των LJ δυνάμεων και του αλγορίθμου Velocity Verlet πραγματοποιήθηκαν υλοποιήθηκαν σε FPGAs. Η απόδοση που επιτεύχθηκε φέρεται να είναι μόλις 0.29x speed-up εκείνης του αρχικού αλγορίθμου-κώδικα που έτρεχε σε έναν υπολογιστή (PC) με επεξεργαστή Intel Pentium 4 στα 2.4 GHz, λόγω του περιορισμένου memory bandwidth και της χαμηλής ταχύτητας ρολογιού στα 26 MHz. Ένα speed-up της τάξεως του 20x είχε προβλεφθεί εάν η αρχιτεκτονική αυτή χρησιμοποιούσε πιο προηγμένες FPGAs με βελτιωμένη μνήμη.

Το 2008 ένα υψηλής απόδοσης ΜΔ σύστημα βασισμένο σε FPGAs αναπτύχθηκε στο CAAD Lab του πανεπιστημίου της Βοστώνης [38, 61, 62, 63, 64, 65]. Οι LJ και Coulomb δυνάμεις υλοποιήθηκαν πάνω σε ένα Annapolis Microsystems WildstarII-Pro board, το οποίο είχε δύο Xilinx Virtex-II Pro XC2VP70-5 FPGAs [66, 67]. Η σχεδίαση υποστήριζε μέχρι 32 τύπους ατόμων και 11.200 άτομα και επιτεύχθηκε 5.5x speed-up πάνω από έναν επεξεργαστή (2.8 GHz Xeon) για μία προσομοίωση ενός συστήματος 8000 σωματιδίων, χρησιμοποιώντας το πακέτο ΜΔ Protomol [110]. Το σύστημα αυτό μπορούσε να υποστηρίξει μέχρι και 256K σωματίδια χρησιμοποιώντας off-chip/on-board μνήμη [38]. Ένα από τα βασικά επιτεύγματα αυτής της εργασίας ήταν η επίτευξη ακρίβειας της προσομοίωσης συγκρίσιμη με εκείνη του λογισμικού χρησιμοποιώντας μια νέα λειτουργία αριθμητικής, ημι-κινητής υποδιαστολής.

Η σημαντικότερη δουλειά που έχει γίνει σε FPGA είναι από την διδακτορική διατριβή του Ashfaquzzaman Khan στο Πανεπιστήμιο της Βοστόνης το 2012 [71]. Στην διατριβή αυτή δημιουργήθηκε ένα ολοκληρωμένο σύστημα προσομοίωσης Μοριακής Δυναμικής

χρησιμοποιώντας το πακέτο NAMD 2.8. Στο σύστημα αυτό με 4 CPU cores και 4 FPGAs επιτεύχθηκε ταχύτητα 2.22x.

Maxwell: Ο Maxwell είναι ένας Computer Cluster βασισμένος σε FPGAs που αναπτύχθηκε από την FHPCA (FPGA High Performance Computing Alliance) στο EPCC (Edinburgh Parallel Computing Centre) του πανεπιστημίου του Εδιμβούργου [15]. Η αρχιτεκτονική του Maxwell περιλαμβάνει 32 blades τις οποίες τις φέρει η κεντρική IBM Blade Center. Κάθε blade περιλαμβάνει έναν επεξεργαστή Xeon και δύο Virtex-4 FX-100 FPGAs. Οι FPGAs συνδέονται με ένα γρήγορο υποσύστημα επικοινωνίας, το οποίο επιτρέπει συνολικά 64 FPGAs να συνδέονται μεταξύ τους σε ένα 8 x 8 σπειροειδή πλέγμα. Κάθε FPGA διαθέτει επίσης τέσσερις 256 MB DDR2 SDRAMs που συνδέονται με αυτές. Τα FPGAs συνδέονται με τον host μέσω ενός διαύλου PCI.

Το 2011, μια FPGA-accelerated έκδοση του LAMMPS αναφέρθηκε πως πρέπει να υλοποιηθεί σε Maxwell [69, 70]. Μόνο οι υπολογισμοί των range-limited non-bonded δυνάμεων (συμπεριλαμβανομένων των δυναμικών και των virials) υπολογίστηκαν στις FPGAs με 4 πανομοιότυπα pipelines σε κάθε FPGA. Η επιτάχυνση που σημειώθηκε ήταν 14x μόνο για τον kernel (εξαιρώντας την επικοινωνία των δεδομένων) σε δύο ή περισσότερους κόμβους του συστήματος του Maxwell, αν και οι αποδόσεις end-to-end ήταν εν τέλει χειρότερες από το software.

Η εργασία αυτή ουσιαστικά υλοποίησε τον εσωτερικό βρόγχο (inner loop) του υπολογισμού του neighbor-list σε ένα kernel για FPGA. Κάθε φορά που ένα σωματίδιο και η neighbor-list του στέλνεται στις FPGAs από τον host τότε υπολογίζονται οι αντίστοιχες δυνάμεις στις FPGAs αυτές. Αυτό δημιουργεί τεράστια ποσά δεδομένων για την επικοινωνία και το οποίο τελικά οδήγησε στο speed-down της FPGA-accelerated έκδοσης.

2.3 Βασικά Χαρακτηριστικά FPGA

Αυτή η ενότητα περιέχει μερικά βασικά χαρακτηριστικά των FPGAs. Στην υλοποίηση της διπλωματικής εργασίας χρησιμοποιήθηκε το εργαλείο Vivado HLS 2015.4 στο οποίο σαν επιλεγμένη FPGA ήταν σειράς 7 της εταιρείας Xilinx, η Zynq ZC 706.

2.3.1 Αρχιτεκτονική FPGA

Οι FPGAs (**Field-Programmable Gate Array**) είναι ένα ολοκληρωμένο κύκλωμα σχεδιασμένο να διαμορφώνεται-προγραμματίζεται μετά την δημιουργία του, εξ ’ου “field programmable”. Ο προγραμματισμός των FPGAs γίνεται με την βοήθεια γλωσσών περιγραφής υλικού (Hardware Description Language) παρόμοια με αυτές που χρησιμοποιούνται για ένα ASIC [72].

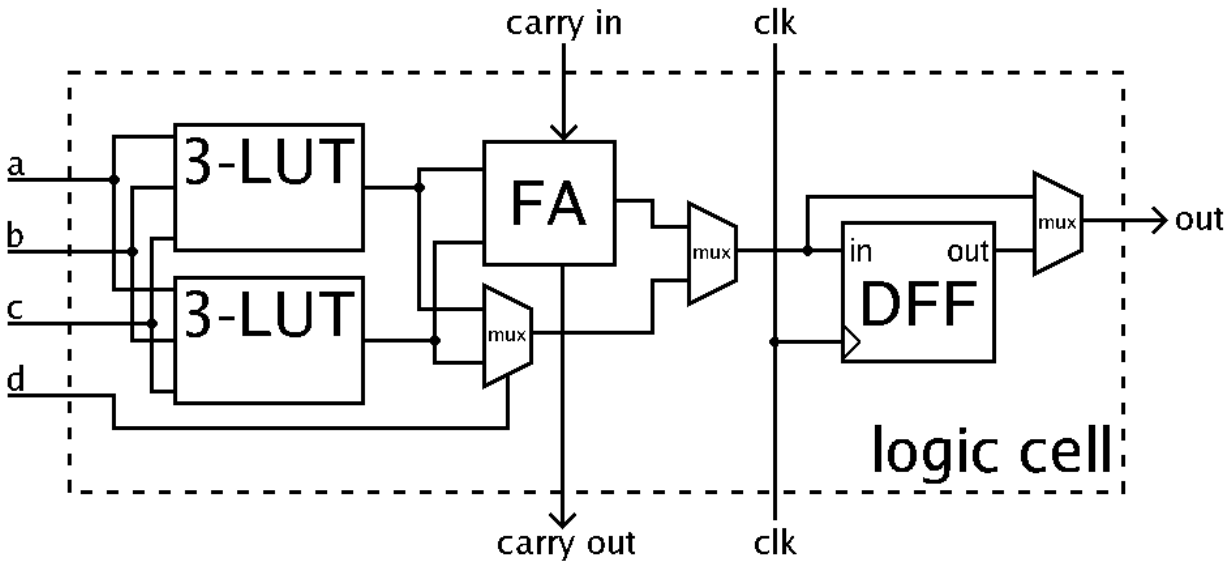
Οι FPGAs περιέχουν μια σειρά από blocks επαναπρογραμματιζόμενης λογικής, και μία ιεραρχία από επαναπροσδιοριζόμενες διασυνδέσεις που επιτρέπει στα blocks να διασυνδέονται ενσύρματα μεταξύ τους. Τα blocks λογικής μπορούν να προγραμματιστούν ώστε να εκτελούν περίπλοκες και συνδυαστικές λειτουργίες, ή να προγραμματιστούν σε απλές πύλες όπως AND, OR, XOR κλπ. Στις περισσότερες FPGAs, τα λογικά blocks μπορούν να περιέχουν μνήμες, οι οποίες μπορεί να αποτελούνται από απλά flip-flops ή ολόκληρα blocks μνημών. Τα

τελευταία χρόνια έχουν εξοπλιστεί με ειδικούς πολλαπλασιαστές και με on-chip μνήμες BRAMs [73, 74]. Όλα αυτά έχουν μετατρέψει τις FPGAs μία σημαντική λύση για επιτάχυνση επιστημονικών εφαρμογών όπως και η Μοριακή Δυναμική.

Σύγχρονες Αρχιτεκτονικές: Στις σύγχρονες αρχιτεκτονικές FPGA, κυριαρχεί η τάση να συνδυάζονται τα blocks λογικής και των διασυνδέσεων με ενσωματωμένους μικροεπεξεργαστές και σχετικά περιφερειακά ώστε να δημιουργήσουν ένα ολοκληρωμένο σύστημα προγραμματιζόμενου chip. Η σειρά FPGAs Xilinx Zynq-7000 All Programmable SoC, στις οποίες περιέχεται ένας διπύρηνος ARM Cortex-A9 MPCore στο 1 GHz ενσωματωμένο μέσα στο hardware των FPGAs, αποτελεί ένα παράδειγμα τέτοιας σύγχρονης αρχιτεκτονικής.

Τα logic blocks αποτελούνται από μερικά λογικά κελιά (που ονομάζονται ALM, LE, Slice κλπ.). Ένα τυπικό τέτοιο λογικό κελί (logic cell) αποτελείται από ένα Look Up Table (LUT) 4 εισόδων, έναν Full Adder (FA) και ένα D-Flip-Flop όπως φαίνεται στο σχήμα 2.11. Σε αυτό το σχήμα τα LUTs διαχωρίζονται σε δύο LUTs 3 εισόδων. Ο μεσαίος πολυπλέκτης (Multiplexer ή MUX) διαλέγει την λειτουργία του logic cell. Στην λειτουργία *normal* τα LUTs συνδυάζονται και δημιουργούν ένα LUT 4 εισόδων με την βοήθεια του αριστερού πολυπλέκτη. Στην λειτουργία arithmetic οι έξοδοι των LUTs τροφοδοτούν τον FA.

Configurable Logic Blocks: Το βασικότερο κοινό χαρακτηριστικό των FPGAs είναι μια σειρά logic blocks (τα οποία αποκαλούνται Configurable Logic Block, CLB, ή Logic Array Block, LAB, ανάλογα τον κατασκευαστή), I/O pads, και routing channels. Ο πολυπλέκτης στα δεξιά ρυθμίζει εάν οι έξοδοι θα είναι σύγχρονες ή ασύγχρονες.

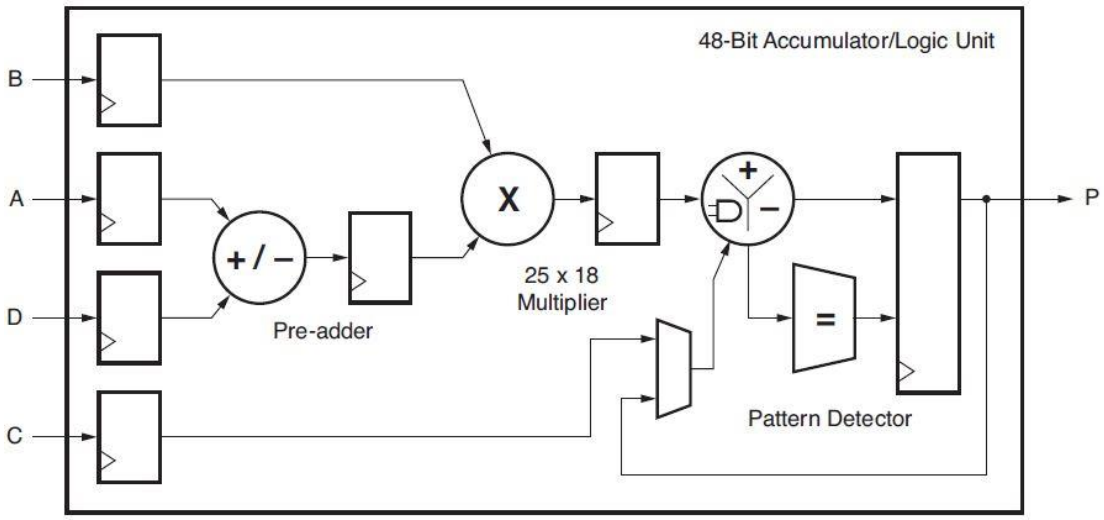


Εικόνα 2.11: *Απλοποιημένο Block Diagram ενός logic cell.*

Configurable Memory Interface: Οι FPGAs παρέχουν μία on-chip μνήμη, που ονομάζεται BRAM (Block-RAM), η οποία μπορεί να επαναπρογραμματιστεί και συνδέεται με μία ποικιλία BRAM Interface Controllers. Επίσης στις . Οι BRAMs δίνουν την δυνατότητα στους σχεδιαστές να προσπελαύνουν την τοπική μνήμη αυτή πολύ γρηγορότερα από τις DRAMs, οι οποίες είναι και

αυτές ενσωματωμένες στις FPGAs. Αυτές οι λειτουργίες δίνουν μεγάλη ευελιξία στους σχεδιαστές στον έλεγχο αποθήκευσης και μεταφοράς δεδομένων, το οποίο είναι ένα μείζον ζήτημα για πολλές εφαρμογές.

DSP Slices: Οι FPGAs είναι αποτελεσματικές για εφαρμογές digital signal processing (DSP) διότι μπορούν να υλοποιήσουν πλήρως παράλληλους αλγορίθμους [75]. Οι εφαρμογές DSP χρησιμοποιούν πολλούς δυαδικούς πολλαπλασιαστές που υλοποιούνται βέλτιστα σε ειδικές φέτες (slices) DSP. Όλες οι FPGAs της σειράς 7 έχουν πολλές ειδικές, full-custom, χαμηλής κατανάλωσης DSP slices, που συνδυάζουν υψηλή ταχύτητα με μικρό μέγεθος ενώ διατηρούν την ευελιξία σχεδιασμού του συστήματος. Οι DSP slices αυξάνουν την ταχύτητα και την αποτελεσματικότητα πολλών εφαρμογών πέρα των digital signal processing, όπως wide dynamic bus shifters, memory address generators, wide bus multiplexers, και memory-mapped I/O registers. Η βασική λειτουργικότητα των DSP48E1 φαίνεται στο Σχήμα 2.12.



Εικόνα 2.12: *Βασικό Block Diagram μίας DSP48E1 slice. [74]*

Κεφάλαιο 3

Στο κεφάλαιο αυτό θα παρουσιαστεί η υλοποίηση του αλγορίθμου με χρήση αναδιατασσόμενης λογικής. Η σχεδίαση και η υλοποίηση γινόταν σε βήματα, τα οποία θα αναλυθούν. Επίσης θα γίνει αναφορά στις δυσκολίες που αντιμετωπίστηκαν κατά την υλοποίηση και οι οποίες οδήγησαν εκ νέου στην σχεδίαση.

3.1 Ανάλυση των Βημάτων

Εδώ θα γίνει η ανάλυση των βημάτων κατά την υλοποίηση και οι δυσκολίες που προέκυψαν στο καθένα. Καθ' όλη την διπλωματική χρησιμοποιήθηκε το μοντέλο πρωτεΐνης ApoA1, το οποίο είναι διαθέσιμο στην κεντρική ιστοσελίδα του NAMD.

3.1.1 Βήμα 1: Μετατροπή κώδικα CUDA σε C.

Βασικά Χαρακτηριστικά CUDA: Η υλοποίηση του αλγορίθμου στο πακέτο NAMD 2.10 για την επιτάχυνση των long-range non-bonded δυνάμεων έχει γίνει σε διάφορες γλώσσες όπως C/C++, Tcl, CUDA, Charm++ κλπ. [76]. Στην διπλωματική αυτή επιλέχθηκε η μετατροπή του κώδικα γραμμένο για κάρτες γραφικών από την γλώσσα υψηλού επιπέδου CUDA στην γλώσσα C. Η επιλογή αυτή έγινε καθώς ο αλγόριθμος που ήταν γραμμένος για GPU ήταν πολύ πιο ευανάγνωστος και πιο δομημένος. Η επιλογή της γλώσσας C για την μετατροπή της CUDA έγινε, διότι είναι πρώτον μία από τις ισχυρότερες γλώσσες προγραμματισμού υψηλού επιπέδου και επίσης το εργαλείο Xilinx Vivado HLS δέχεται περιορισμένους τύπους γλωσσών, μία από αυτές είναι η C.

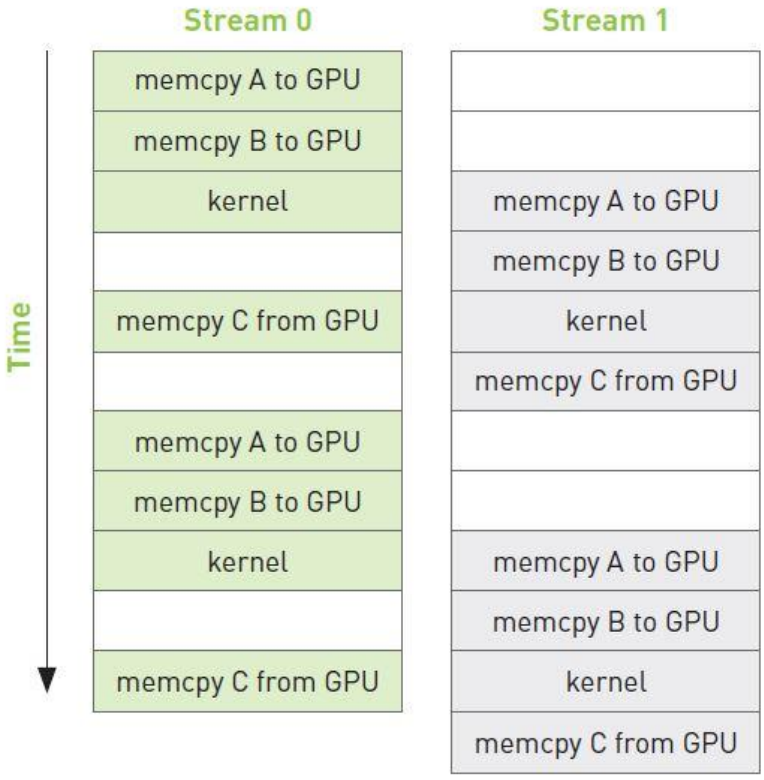
Η CUDA είναι μία γλώσσα προγραμματισμού υψηλού επιπέδου, η οποία αναπτύχθηκε από την εταιρεία NVIDIA, που κατασκευάζει GPUs και αποτελεί πρωτοπόρα εταιρεία στην πώληση εμπορικών GPUs.

Κύριο χαρακτηριστικό της, όπως και άλλων αντίστοιχων γλωσσών (π.χ. OpenCL), είναι η μαζική παραλληλία που προσφέρει στον προγραμματισμό. Οι μονάδες επεξεργασίας γραφικών είναι μαζικά παράλληλοι επεξεργαστές αριθμητικών υπολογισμών που προγραμματίζονται σε C με κάποιες επεκτάσεις. Ο προγραμματισμός αυτών των επεξεργαστών δεν απαιτεί κατανόηση των αλγορίθμων. Οι GPUs προσφέρουν την δυνατότητα πολυνηματικού προγραμματισμού (multithreading), σχετικά μικρές κρυφές μνήμες σε σχέση με τις CPU και σχεδίαση διασυνδέσεων μνήμης με έμφαση στο bandwidth. Όλα αυτά τα χαρακτηριστικά μετατρέπουν τις GPUs κατάλληλες για πολλούς παράλληλους υπολογισμούς σε επιστημονικά μοντέλα, ένα από αυτά είναι και το NAMD. Στο NAMD οι υπολογισμοί των long-ranged non-bonded δυνάμεων αποτελούν καθίστανται κατάλληλοι για την CUDA.

Αρχικός στόχος ήταν η μετατροπή του πολυνηματικού κώδικα της CUDA σε μονονηματικό κώδικα C. Για να επιτευχθεί αυτό χρειάστηκε η βαθύτερη κατανόηση της γλώσσας CUDA, και ειδικότερα των επεκτάσεων που προσέφερε σε σχέση με την C.

Στην πρωτεΐνη ApoA1 χρησιμοποιεί 248832 νήματα (threads), και 2 streams. Τα threads αυτά χωρίζονται σε 1944 blocks από threads και το κάθε block από 128 threads. Όλα αυτά τα

threads εκτελούν παράλληλα το ίδιο κομμάτι κώδικα, όπως επίσης και τα 2 streams. Το κάθε stream έχουν τον ίδιο αριθμό από threads και λειτουργούν παράλληλα [76]. Η χρήση τους βοηθά στον ακόμα μεγαλύτερη παραλληλισμό του κώδικα καθώς δίνουν την δυνατότητα εγγραφής στην μνήμη από το πρώτο stream ενώ την ίδια στιγμή το δεύτερο πραγματοποιεί υπολογισμούς και τούμπαλιν. Στο σχήμα 3.1 φαίνεται πως 2 streams διαχειρίζονται και συνεργάζονται στον ίδιο kernel.



Εικόνα 3.1: *Χρονοδιάγραμμα 2 streams που εκτελούν παράλληλα και ανεξάρτητα τον ίδιο kernel.*

Βασική Υλοποίηση C: Κατά την υλοποίηση του κώδικα C έπρεπε να ακολουθηθεί η φιλοσοφία μετατροπής του πολυνηματικού κώδικα σε μονονηματικό. Για να επιτευχθεί αυτό και τα 248832 νήματα και των 2 streams έπρεπε να παρομοιώνονται από ένα νήμα μόνο. Η προσομοίωση αυτή έγινε με την χρήση βρόγχων (loops). Ουσιαστικά όλες οι λειτουργίες που γίνονταν παράλληλα στην CUDA, στην C θα γίνονται γραμμικά, δηλαδή τα streams και τα threads θα εκτελεστούν το ένα μετά το άλλο.

Ο κώδικας της βασικής συνάρτησης ξεκινάει καλώντας ένα loop μεγέθους 2. Αυτό το loop θα προσομοιώσει την λειτουργία των streams. Όταν ολοκληρωθεί το πρώτου θα ξεκινήσει η λειτουργία του δεύτερου. Έπειτα ξεκινάει η προσομοίωση των blocks. Στο loop των streams ξεκινάει ένα δεύτερο loop, το οποίο έχει μέγεθος 1944 και προσομοιώνει με την σειρά του την λειτουργία των blocks. Μέσα σε αυτό το loop πλέον ξεκινάει ο βασικός κώδικας της συνάρτησης.

Σε κάθε block περιέχει με την σειρά του 128 threads. Τα threads αυτά προσομοιώνονται σε διακριτά σημεία μέσα στον κώδικα. Τα threads της CUDA, καθώς εκτελούν παράλληλα τον kernel, αποθηκεύουν και διαβάζουν από διάφορα σημεία της μνήμης της GPU. Για να αποφευχθούν λογικά και λειτουργικά σφάλματα στον κώδικα η CUDA παρέχει μία συνάρτηση, αντίστοιχη της συνάρτησης barrier() που παρέχει η βιβλιοθήκη <pthread.h> στην C, που ονομάζεται syncthreads(). Η syncthreads() παρέχει τον έλεγχο του συγχρονισμού στον προγραμματιστή, σταματώντας την λειτουργία των threads, μόλις φθάσουν σε αυτή και αφήνοντας ξανά την λειτουργία τους μόνο όταν έχουν φθάσει όλα σε αυτήν. Έτσι επιτυγχάνεται ο έλεγχος των δεδομένων που γράφουν τα threads πριν το syncthreads() και η ανάγνωση τους μετά από αυτό. Η έλλειψη του θα προκαλούσε προβλήματα καθώς τα threads, αν και εκτελούνται παράλληλα, κάποιο thread υπάρχει περίπτωση να διάβαζε από μία θέση μνήμης στην οποία δεν είχε προλάβει να εγγράψει κάποιο άλλο thread την σωστή τιμή.

Η υλοποίηση αυτών των λειτουργιών γίνεται με την χρήση loops μέσα στον κώδικα. Όταν ξεκινάει μία διαδικασία μέσα στον κώδικα, η οποία εγγράφει στην μνήμη δεδομένα, στον κώδικα της C ανοίγει ένα loop, το οποίο προσομοιώνει την λειτουργία των 128 threads του κάθε block, μεγέθους όσο και ο αριθμός αυτών των threads. Με αυτό το τρόπο εξασφαλίζεται η ομαλή αποθήκευση των δεδομένων, ώστε όταν χρειαστούν σε επόμενα σημεία του κώδικα να είναι έτοιμα προς ανάγνωση.

Επιπλέον, στην CUDA δημιουργούνται αρκετές τοπικές μεταβλητές. Οι μεταβλητές αυτές μπορεί να δημιουργούνται από το κάθε thread ξεχωριστά. Έτσι σε κάθε block δημιουργούνται 128 διαφορετικές μεταβλητές με το ίδιο όνομα. Αυτό επιτρέπεται στην CUDA να τις ξεχωρίζει ανάλογα από ποιο thread δημιουργήθηκαν, εφόσον είναι ανεξάρτητες μεταξύ τους. Στην C για να προσομοιωθεί αυτή η ιδιότητα δημιουργήθηκαν για κάθε τέτοιου είδους μεταβλητής ένας πίνακας μονοδιάστατος και μεγέθους 128, ώστε να αποθηκεύεται σε κάθε κελί η τιμή της μεταβλητής για το αντίστοιχο thread (κελί 0 αντιστοιχεί στο thread 0 κ.ο.κ.).

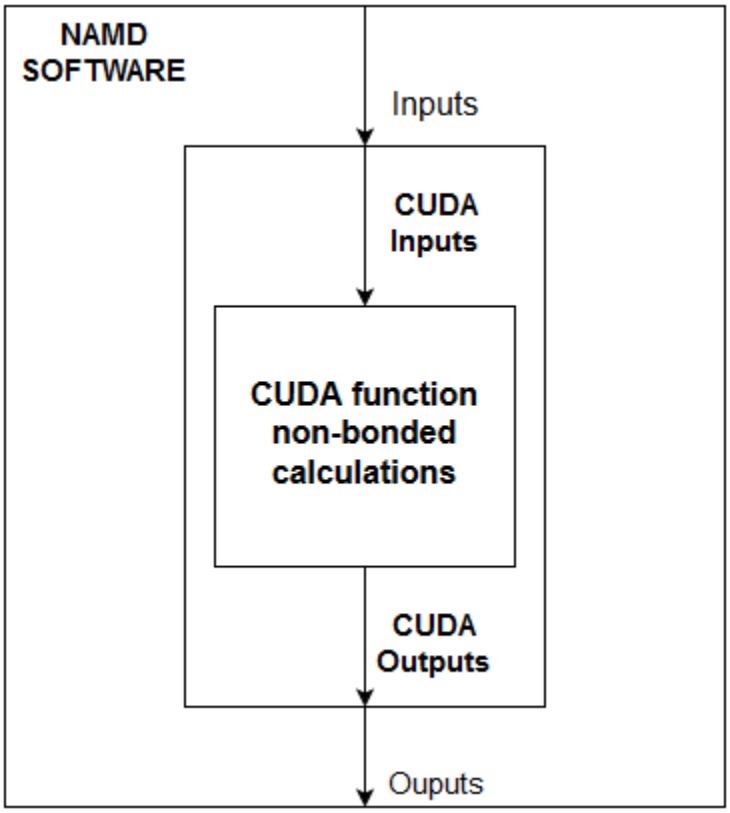
Η βασική συνάρτηση αυτή ανήκει σε ένα ολοκληρωμένο πρόγραμμα C, το οποίο δημιουργήθηκε για τον έλεγχο των αποτελεσμάτων και έπειτα χρησιμοποιήθηκε σαν testbench για το Vivado HLS, όπου θα αναλυθεί αργότερα σε αυτό το κεφάλαιο.

Οι είσοδοι της βασικής συνάρτησης διαβάζονται από αρχεία τα οποία έχουν δημιουργηθεί στην αρχή κατά την εκτέλεση της CUDA. Πιο συγκεκριμένα, ο αρχικός κώδικας της CUDA τροποποιήθηκε με κατάλληλο τρόπο, ώστε κατά την εκτέλεση του NAMD 2.10 με GPU acceleration με την πρωτεΐνη ApoA1, στο πρώτο timestep (από τα 500) της λειτουργίας να αποθηκεύονται οι είσοδοι για την βασική συνάρτηση της CUDA σε buffers και έπειτα να μεταφέρονται στην κεντρική μνήμη. Έπειτα οι buffers αυτοί γράφτηκαν σε αρχεία και από τα οποία κατά την εκτέλεση της C διαβάζονται στην αρχή του προγράμματος (συγκεκριμένα στην main), με κατάλληλες συναρτήσεις που δημιουργήθηκαν, και αποθηκεύονται εκ νέου σε buffers όπου αποτελούν εισόδους για την κεντρική συνάρτηση.

Μετά την επιτυχή εκτέλεση της βασικής συνάρτησης, το πρόγραμμα επιστρέφει στην main, όπου ελέγχονται οι έξοδοι με τις αντίστοιχες εξόδους της CUDA, οι οποίες επίσης εγγράφτηκαν σε αρχεία πριν την εκτέλεση του προγράμματος.

Όπως θα φανεί και στην παρουσίαση των αποτελεσμάτων στο κεφάλαιο 4, η πολυπλοκότητα του αλγορίθμου που επιτεύχθηκε είναι γραμμική. Όταν τροφοδοτούμε το πρόγραμμα με μικρότερη είσοδο τότε ο χρόνος εκτέλεσης του μειώνεται ανάλογα. Λόγω της

γραμμικότητας αυτής, μπορούμε εύκολα να προβάλλουμε τα αποτελέσματα (project) για μεγαλύτερη είσοδο. Αυτό θα βοηθήσει αργότερα στην εξαγωγή αποτελεσμάτων από το Vivado HLS. Η μείωση της εισόδου επιτυγχάνεται μειώνοντας το loop που προσομοιώνει το block size. Παρακάτω φαίνεται ένα block diagram του αρχικού κώδικα του NAMD.



Εικόνα 3.2: Block diagram Software NAMD

3.1.2 Βήμα 2: Εισαγωγή κώδικα C στο Vivado HLS.

Κύρια Χαρακτηριστικά: Το Vivado HLS είναι μία πλατφόρμα που δημιουργήθηκε από την εταιρεία Xilinx και σκοπός του είναι η αυτοματοποιημένη παραγωγή γλώσσας περιγραφής υλικού για ένα αλγόριθμο ανεπτυγμένο σε μία γλώσσα υψηλού επιπέδου. Το Vivado HLS παρέχει την δυνατότητα στον χρήστη να εκτελέσει τον κώδικα σε γλώσσα υψηλού επιπέδου, έπειτα να γίνει το synthesize (compile) της σχεδίασης και τέλος να γίνει προσομοίωση του κώδικα γλώσσας περιγραφής υλικού για την FPGA που έχει επιλεχθεί.

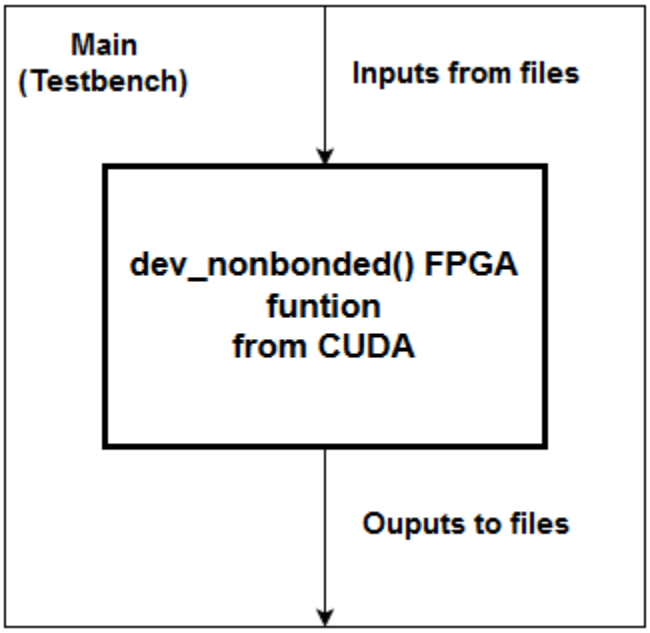
Στο Vivado HLS χρειάζεται να ορισθεί στην αρχή της δημιουργίας του project ένα αρχείο με μία βασική συνάρτηση, η οποία θα είναι αυτή που θα μεταφερθεί στο hardware. Έπειτα ορίζεται το testbench. Στο testbench υπάρχει όλο το υπόλοιπο κομμάτι του προγράμματος όπως για παράδειγμα η συνάρτηση main και οποιοσδήποτε άλλος κώδικας χρειάζεται για την εκτέλεση του προγράμματος. Επίσης, επιλέγεται η FPGA για την οποία θα γίνονται οι προσομοιώσεις του κώδικα και με το ρολόι με το οποίο θα δουλεύει. Τέλος, το εργαλείο παρέχει

μια σειρά directives με τα οποία ο κώδικας μπορεί να επιταχυνθεί, μειώνοντας τους συνολικούς κύκλους. Όλες αυτές οι πληροφορίες αναφέρονται εδώ για να γίνει κατανοητό το κείμενο παρακάτω, στο οποίο θα αναλυθούν οι σχεδιάσεις.

Ένα από τα μεγαλύτερα προβλήματα που έπρεπε να αντιμετωπιστούν ήταν η αδυναμία προσομοιώσεις του κώδικα σε όλες τις σχεδιάσεις για ολόκληρη την είσοδο. Πιο συγκεκριμένα, η προσομοίωση του κώδικα περιγραφής υλικού, που παράγεται κατά το synthesis, είναι μία πολύ αργή διαδικασία. Για τον λόγο αυτό για να προσομοιωθεί ο κώδικας χρησιμοποιήθηκε μικρότερη είσοδος. Η μικρότερη είσοδος σημαίνει ότι ο βρόγχος που προσομοιώνει την λειτουργία των blocks (μεγέθους 1944), μειώθηκε πολλές φορές σε μέγεθος από 1 έως και 12 (grid size). Αυτό παρήγαγε λάθος αποτελέσματα όπως είναι φυσικό, αλλά μας ενδιέφερε ο χρόνος μόνο εκτέλεσης στην FPGA.

Για την διπλωματική εργασία χρησιμοποιήθηκε η έκδοση του Vivado HLS 2015.4 και ως target devices (FPGAs) οι Zynq ZC 706 και κάποιες φορές η ZedBoard.

1η Υλοποίηση: Η αρχική υλοποίηση αποτελούσε την μεταφορά του κώδικα ως είχε στο Vivado HLS. Για testbench επιλέχθηκε η main και οι συναρτήσεις ανάγνωσης των αρχείων που έχουν τις εισόδους και οι συναρτήσεις εγγραφής σε αρχεία των εξόδων. Η βασική συνάρτηση που επιλέχθηκε, για να μεταφερθεί στην FPGA, ήταν η συνάρτηση που μετατράπηκε από CUDA σε C. Στην αρχική αυτή υλοποίηση δεν χρησιμοποιήθηκαν καθόλου directives, τα οποία παρέχει το εργαλείο. Λόγω των προβλημάτων με την προσομοίωση του κώδικα περιγραφής υλικού σε αυτή την σχεδίαση έγινε μόνο synthesis. Το synthesis έγινε για διάφορες τιμές του grid size. Όπως ήταν αναμενόμενο παρατηρήθηκε και εκεί μία γραμμικότητα στον τελικό αριθμό των κύκλων ανάλογο του μεγέθους του grid size. Παρακάτω φαίνεται το block diagram της υλοποίησης αυτής.



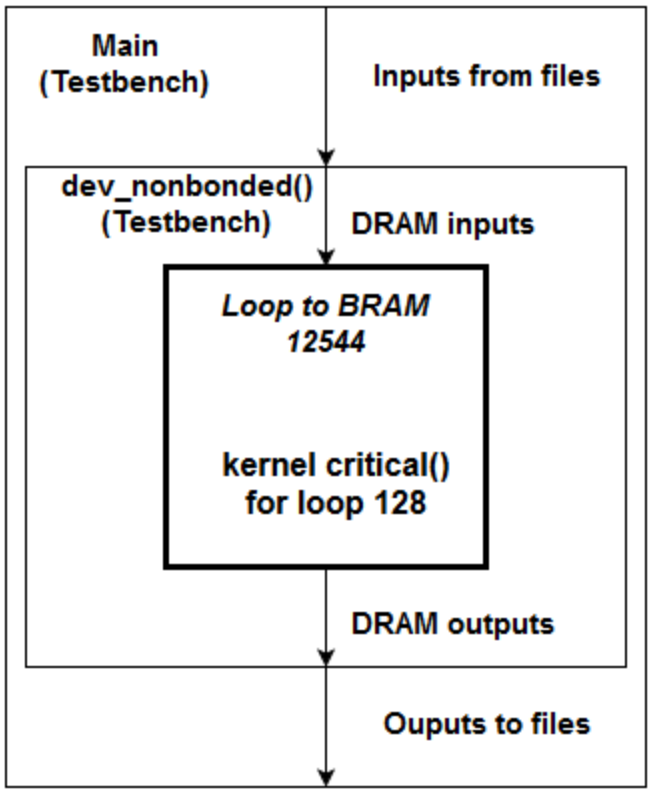
Εικόνα 3.3: Block diagram 1ης Υλοποίησης.

Στην υλοποίηση αυτή, έπειτα, χρησιμοποιήθηκαν directives. Αρχικά, τοποθετήθηκαν σε ολόκληρο τον κώδικα το directive του PIPELINE, σε όλους τους βρόγχους. Το synthesis γινόταν μόνο για grid size ίσο με 1 και 2. Η γραμμικότητα εξασφάλιζε την πληροφορία και για grid size ίσο με 1944. Τα pipelines μείωσαν αισθητά τους κύκλους κατά μερικά εκατομμύρια. Η τεχνική του pipelining διατηρήθηκε για όλη την σχεδίαση. Η δεύτερη μέθοδος που εφαρμόσθηκε ήταν η χρήση του directive LOOP UNROLLNG. Αρχικά τοποθετήθηκε μόνο στα δύο μεγάλα loops, δηλαδή στο stream και blocks. Η χρήση του μείωσε τους κύκλους κατά μία τάξη μεγέθους. Παρ' όλα αυτά, το ρολόι αυξήθηκε αρκετά. Έπειτα, το LOOP UNROLLING χρησιμοποιήθηκε σε όλα τα loops, χωρίς όμως να αλλάζει αισθητά τον αριθμό των κύκλων και χρησιμοποιούσε πολύ περισσότερους πόρους στην FPGA. Το LOOP UNROLLING διαχωρίζει τον βρόγχο όσο του factor που ορίζει ο χρήστης. Έτσι ένα loop το οποίο έχει μέγεθος 128 μπορεί με τον factor ίσο με 2 να διαχωριστεί σε 2 loops μεγέθους 64. Εκεί φυσικά χρειάζεται ιδιαίτερη προσοχή η εξαρτήσεις μεταξύ των επαναλήψεων. Το LOOP UNROLLING λόγω της φύσης του αυξάνει το υλικό και τους πόρους που δεσμεύονται για ένα loop ανάλογα τον factor που του ορίστηκε.

Η γραμμικότητα του αλγορίθμου οφείλεται στην μεταφορά του κώδικα από CUDA σε C. Το κύριο χαρακτηριστικό της CUDA είναι η παραλληλία που προσφέρει στον προγραμματισμό εφόσον δεν υπάρχουν εξαρτήσεις. Αυτή η ιδιότητα κληρονομήθηκε και στην C, έτσι ώστε όλα τα loops που προσομοιώνουν threads να μην έχουν εξαρτήσεις από την μία επανάληψη στην επόμενη. Αυτό σημαίνει ότι οι μεταβλητές και οι θέσεις μνήμης που εγγράφονται σε κάθε επανάληψη του κάθε τέτοιου βρόγχου να είναι ανεξάρτητα από τα επόμενα και τα προηγούμενα. Αυτή η ιδιότητα οδήγησε στην χρήση ενός ακόμα directive, του DEPENDENCE. Με αυτό το directive δίνεται η δυνατότητα να ορισθούν στην αρχή ενός βρόγχου οι εξαρτήσεις μέσα σε αυτόν. Στην περίπτωση των βρόγχων που προσομοιώνουν την λειτουργεία των 128 threads σε κάθε block, δεν υπάρχει καμία εξάρτηση και για τον λόγο αυτό οι ελάχιστοι κύκλοι μειώθηκαν ενώ ο μέγιστος αριθμός μειώθηκε και αυτός, αλλά όχι αρκετά. Τα αποτελέσματα παρουσιάζονται αναλυτικότερα στο κεφάλαιο 5 και συγκεκριμένα στο 4.2.2.

2η Υλοποίηση (Critical Section): Η πρώτη υλοποίηση δεν θα μπορούσε σε καμία περίπτωση να καταφέρει έναν χαμηλό χρόνο και άρα μια καλή απόδοση στην FPGA. Για την 2^{η} υλοποίηση έγινε περισσότερη μελέτη του κώδικα C. Συγκεκριμένα έγινε profiling αυτού του κώδικα και διαπιστώθηκε ότι ένα συγκεκριμένο loop καταναλώνει το 95% του συνολικού χρόνου. Στο loop αυτό υπολογίζονται πραγματικά οι long-ranged non-bonded δυνάμεις.

Για τη νέα σχεδίαση χρειάστηκε η αλλαγή του κώδικα C. Το critical κομμάτι αυτό του κώδικα μεταφέρθηκε σε μία νέα συνάρτηση η οποία καλείται πλέον μέσα στην αρχική βασική συνάρτηση. Η νέα συνάρτηση αποτελείται πλέον από 2 loops το ένα εμφωλευμένο στο άλλο. Το εξωτερικό loop είναι μεγέθους 128 και προσομοιώνει την λειτουργία των threads ενός block. Είναι σημαντικό να αναφερθεί επίσης πως η critical συνάρτηση καλείται πολλές φορές αφού και αυτή με την σειρά της ανήκει σε μία ιεραρχία από loops (συνολικά 4: stream, blocks, blocki, blockj). Το block diagram φαίνεται παρακάτω.



Εικόνα 3.4: Block Diagram 2ης Υλοποίησης

Αρχικά, ο νέος κώδικας μεταφέρθηκε στο Vivado HLS χωρίς directives. Παρατηρήθηκε μεγάλη μείωση των κύκλων κατά το synthesis σε σχέση με την αρχική σχεδίαση. Επίσης είναι σημαντικό να σημειωθεί ότι η αρχική βασική συνάρτηση στην 1η σχεδίαση πλέον αποτελεί κομμάτι του testbench. Το loop που υπολογίζει τις non-bonded δυνάμεις είναι το παρακάτω:

```
for (syncthreads=0; syncthreads<THREAD_SIZE; syncthreads++) {
   for ( j = 0; j < SHARED_SIZE; ++j ) {
                    /* actually calculate force */
                    …
   }
}
```

Στον παραπάνω κώδικα το βασικό loop που υπολογίζει τις non-bonded δυνάμεις συγκεκριμένα είναι αυτό με index ‘j’, δηλαδή το εμφωλευμένο. Το εξωτερικό loop τοποθετήθηκε κατά την μετατροπή του κώδικα από CUDA σε C και έχει σκοπό να προσομοιώσει την λειτουργία των 128 threads του κάθε block.

Έπειτα, ξεκίνησε η χρήση των directives. Το πρώτο directive που χρησιμοποιήθηκε ήταν το PIPELINE στο εσωτερικό loop. Η σχεδίαση με το PIPELINE ήταν 5 φορές ταχύτερη από αυτή χωρίς το pipeline. Τα αποτελέσματα που εμφανίζονται στο synthesis είναι πάντα για μία κλήση της critical. Στη συνέχεια, χρησιμοποιήθηκε το directive του LOOP UNROLLING με factor ίσο με 2, αλλά δεν έδωσε καμμιά βελτίωση. Ο λόγος οφειλόταν στην αδυναμία του εργαλείου να

πραγματοποιήσει το unrolling του loop καθώς έβλεπε εξαρτήσεις μεταξύ των επαναλήψεων. Αυτό φυσικά λύθηκε με την εισαγωγή του directive DEPENDENCE και δηλώνοντας τις εξαρτήσεις αυτές ως λανθασμένες. Στην ίδια αυτή σχεδίαση χρησιμοποιήθηκε και ένα επιπλέον directive, το ARRAY PARTITION. Το directive αυτό έχει την ιδιότητα να κατακερματίζει κάποιους buffers, που θα ορίσει ο προγραμματιστής. Ο κατακερματισμός μπορεί να γίνει με διάφορους τρόπους. Στην περίπτωση της σχεδίασης αυτής χρησιμοποιήθηκε ο κατακερματισμός των buffers totalev, totales, totalee σε 2 μέρη ο καθένας με την μέθοδο cyclic. Η χρήση όλων αυτών των directives μείωσε και άλλο τους κύκλους. Έπειτα δοκιμάστηκε η χρήση μόνο του pipelining, του unrolling και των false dependencies, χωρίς δηλαδή το partitioning των arrays. Το αποτέλεσμα αποδείχθηκε καλύτερο.

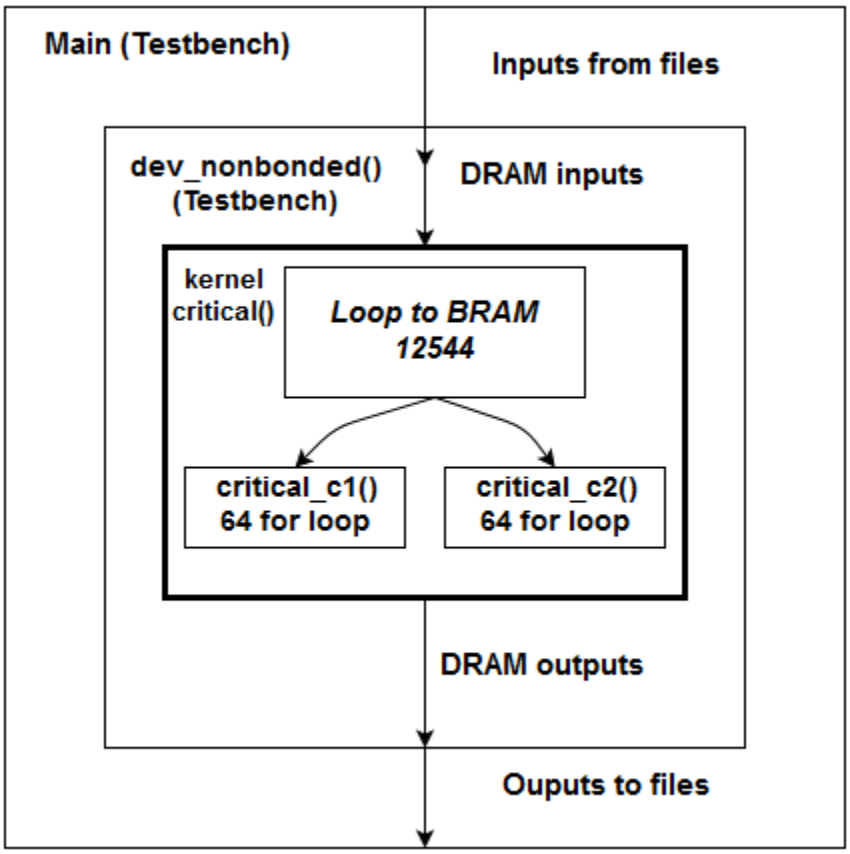
Για πρώτη φορά στην διπλωματική χρησιμοποιήθηκε και το C/RTL Co-simulation, κατά το οποίο γίνεται η προσομοίωση του κώδικα περιγραφής υλικού που δημιουργήθηκε κατά το βήμα του synthesis. Φυσικά η είσοδος δεν θα μπορούσε να ξεπεράσει έναν αριθμό για το grid size, καθώς το simulation είναι μία απίστευτα χρονοβόρα διαδικασία. Χρησιμοποιήθηκαν τρεις διαφορετικοί αριθμοί και άρα τρία διαφορετικά simulations, για 1, 2 και 4. Για την τιμή 1944 γίνεται προβολή του χρόνου από αυτό που είναι πιο κοντά δηλαδή για grid size ίσο με 4. Τα αποτελέσματα παρουσιάζονται στο 4.2.3.

3η Υλοποίηση (2 cores): Σε αυτή την υλοποίηση αφού μελετήθηκε εκ νέου ο αλγόριθμος αποφασίστηκε να διαχωριστεί σε 2 πυρήνες (cores). Το directive LOOP UNROLLING, ουσιαστικά κάνει παρόμοια δουλειά εάν δοθεί factor ίσο με 2. Παρόλαυτά η αυτοματοποιημένη διαδικασία αυτή έχει αρκετά προβλήματα και για τον λόγο αυτό προτιμήθηκε η χειροκίνητη. Κατά την διαδικασία αυτή, ο κώδικας μετατράπηκε ώστε η critical συνάρτηση να καλεί με την σειρά της δύο νέες συναρτήσεις οι οποίες μοιράζονται την δουλειά της critical. Η κάθε μία τέτοια συνάρτηση critical_c1 και critical_c2 έχουν το αρχικό κώδικα της critical αλλά με την διαφορά ότι εκτελούν το loop των 128 threads από 64 φορές το καθένα, δηλαδή μοιράζουν την εκτέλεση του στα 2. Όπως έχουμε ήδη επισημάνει δεν υπάρχει καμία εξάρτηση ανάμεσα στις επαναλήψεις και έτσι ο στόχος αυτής της σχεδίασης ήταν να εκτελούνται αυτές οι δύο συναρτήσεις παράλληλα. Για να επιτευχθεί αυτό έπρεπε και οι buffers και οι μεταβλητές που θα δεχόταν η κάθε συνάρτηση να είναι ανεξάρτητοι/ες, διότι αντίθετα το εργαλείο αδυνατεί να εκτελέσει τις δύο συναρτήσεις παράλληλα. Έτσι οι buffers, μεγέθους 128, που προκύπτουν από την προσομοίωση των threads διαχωρίστηκαν σε buffers μεγέθους 64 θέσεων και ο καθένας στάλθηκε κατάλληλα στην αντίστοιχη συνάρτηση του. Κάποιοι buffers δεν μπορούσαν να διαχωριστούν όμως καθώς θα έπρεπε να αλλάξει ριζικά ο αλγόριθμος και ο τρόπος ανάγνωσης τους , οπότε μεταφέρονταν ως έχουν, σε σχέση με το μέγεθος, τους στις δύο συναρτήσεις. Παρακάτω εμφανίζεται το loop μετά την μετατροπή στην critical_c1 και critical_c2:

```
for (syncthreads=0; syncthreads<THREAD_SIZE/2; syncthreads++) {
   for ( j = 0; j < SHARED_SIZE; ++j ) {
                    /* actually calculate force */
                    …
   }
}
```

Όπως γίνεται αντιληπτό και από τον κώδικα είναι ότι το εσωτερικό loop παραμένει ανέπαφο καθώς δεν μπορεί να διασπαστεί και το εξωτερικό χωρίζεται στα δύο. Στο κάθε core εσωτερικά με κατάλληλους factors όπου χρειάζεται εκτελείται ο κώδικας και για τα 128 threads.

Άλλη μία αλλαγή που πραγματοποιήθηκε ήταν η δημιουργία τοπικών buffers και μεταβλητών. Ο λόγος ήταν ώστε να μειωθεί όσο δυνατόν στο ελάχιστο το I/O. Στις προηγούμενες σχεδιάσεις οι είσοδοι χρησιμοποιούνται όπως έρχονται από την DRAM. Σε αυτή την σχεδίαση ουσιαστικά όλες οι είσοδοι μεταφέρονται τώρα στην BRAM η οποία είναι κατά πολύ ταχύτερη από την DRAM. Αυτό επιτεύχθηκε με την δημιουργία των buffers και των μεταβλητών στην critical συνάρτηση (ταυτόχρονα διαχωρίστηκαν οι buffers των 128 θέσεων) και έπειτα η αρχικοποίηση τους από την DRAM. Έτσι πλέον οι critical_c1 και critical_c2 θα αναζητούν μόνο από την BRAM. Επιπλέον οι buffers που ήταν αδύνατον να διαχωριστούν απλά αντιγράφτηκε ο καθένας σε δύο τοπικούς buffers στην BRAM ώστε να σταλούν ανεξάρτητα στις δύο συναρτήσεις. Το μέγιστο μέγεθος ήταν 12544 θέσεις. Παρακάτω βλέπουμε το block diagram της 3ης υλοποίησης.



Εικόνα 3.5: Block diagram 3ης υλοποίησης.

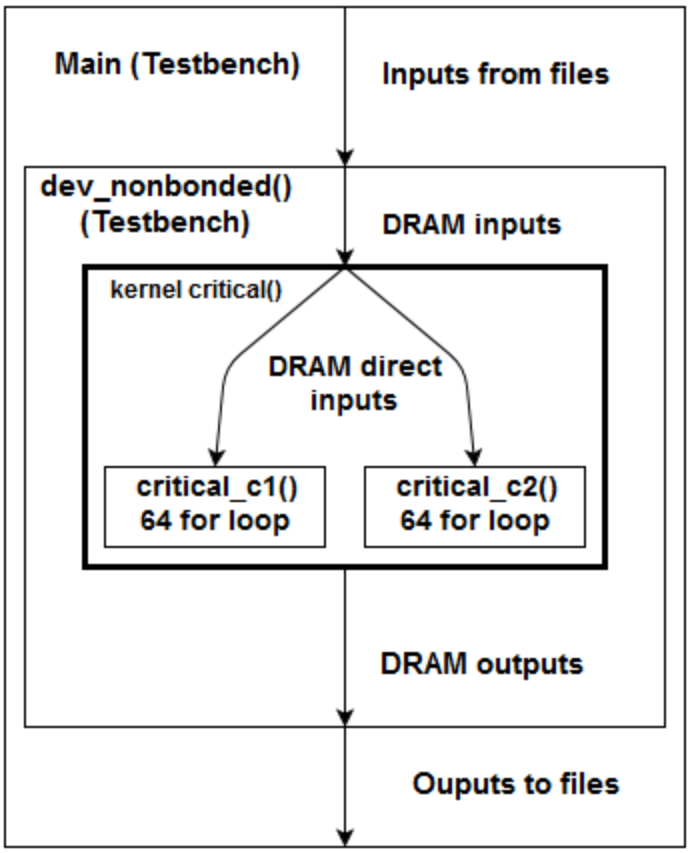
Η σχεδίαση αυτή όπως ήταν αναμενόμενο παρουσίασε μεγάλη βελτίωση σε σχέση με την προηγούμενη. Οι κύκλοι μειώθηκαν κατά 26%. Παρόλαυτά παρατηρήθηκε πως η σχεδίαση αυτή δεν θα μπορούσε ποτέ να μειωθεί κάτω από περίπου 12544 κύκλους (περίπου γιατί

χρειάζονται και μερικοί επιπλέον κύκλοι για είσοδο και έξοδο από loop, όπως και η είσοδος σε συναρτήσεις). Αυτό φυσικά συμβαίνει διότι το αρχικό loop αρχικοποίησης δεν μπορεί να μειωθεί κάτω από 12544 επαναλήψεις από την στιγμή που τοποθετούμε ολόκληρους buffers στην BRAM.

Η σχεδίαση αυτή λοιπόν έχει σκοπό να εκμεταλλευτεί την ανεξαρτησία μεταξύ των επαναλήψεων των loops και την ταχύτητα που προσφέρει η BRAM. Το τελευταίο όμως δημιουργεί ένα φράγμα στην σχεδίαση, ακόμα και αν υποθετικά διαχωριζόταν η σχεδίαση σε 128 διαφορετικά cores (όσο το μέγεθος και του loop), το οποίο φράγμα είναι περίπου 12544 κύκλοι. Τα αποτελέσματα παρουσιάζονται στην ενότητα 4.2.4.

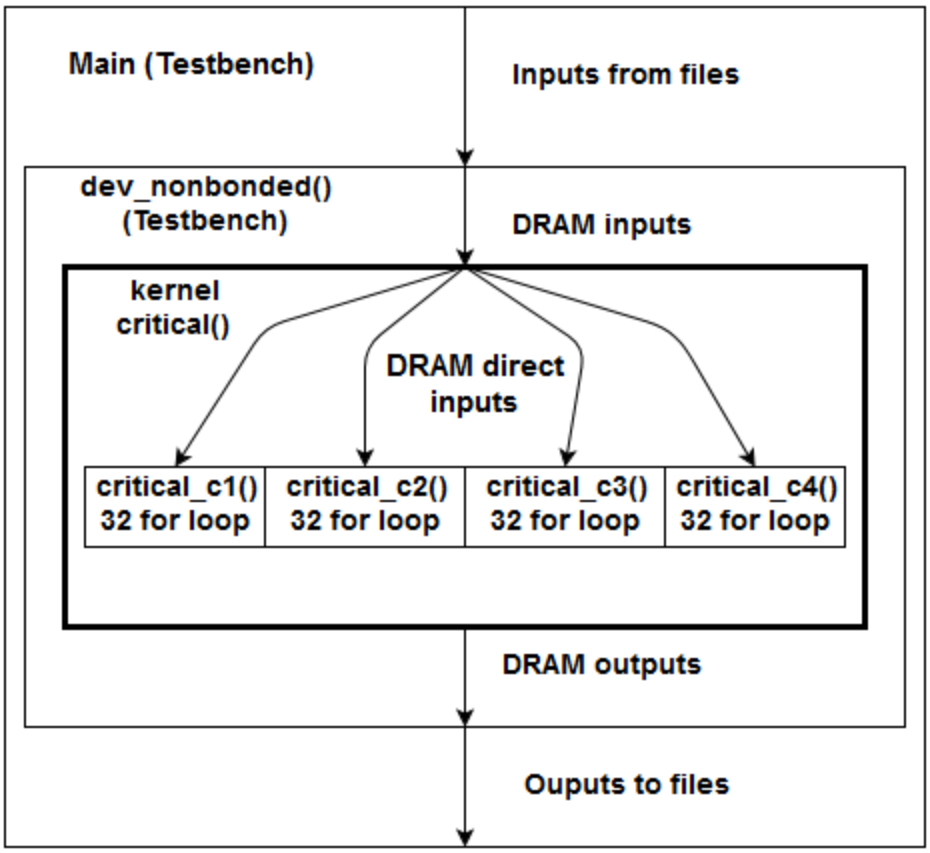
Αφαίρεση INTERFACE directives: Κατά τις σχεδιάσεις αυτές, χρησιμοποιήθηκαν εσφαλμένα και κάποια directives του εργαλείου. Τα directives αυτά, ανήκουν στην κατηγορία των INTERFACE και χρησιμοποιούνται για την επικοινωνία των cores με τον ARM επεξεργαστή και τις μνήμες της FPGA και η εσφαλμένη χρήση τους επηρέαζε την λειτουργία της σχεδίασης. Μετά την αφαίρεση τους η 3^{η} υλοποίηση αμέσως επιταχύνθηκε. Πιο συγκεκριμένα, οι συνολικοί κύκλοι κατέβηκαν στους 14735 με 2 cores. Τα αποτελέσματα παρουσιάζονται στην ενότητα 4.2.4, όπου γίνεται και μία προσπάθεια υπολογισμού του συνολικού χρόνου της σχεδίασης.

4η Υλοποίηση (bufferless 2 cores): Η νέα σχεδίαση αυτή περισσότερο σκοπό έχει να μετρήσει την επιτάχυνση που μπορεί να επιτευχθεί από τον kernel μόνο, που δημιουργήθηκε για την FPGA (computations only). Πιο συγκεκριμένα, οι μεγάλοι buffers, οι οποίοι δεν είναι δυνατόν να διαχωριστούν, δεν έρχονται πλέον από την BRAM, αλλά από την DRAM. Στη νέα σχεδίαση, για τον κάθε τέτοιον buffer πλέον, δημιουργούνται στο testbench δύο αντίγραφά του, ώστε να σταλούν στις critical_c1 και critical_c2 αντίστοιχα. Αυτό πλέον μειώνει δραματικά τους κύκλους, καθώς οι αρχικοποιήσεις, των buffers στην BRAM, πλέον περιορίζονται μόνο στους buffers, μεγέθους 128, των threads. Έτσι το αρχικό loop των αρχικοποιήσεων από 12544 επαναλήψεις χρειάζεται περίπου μόνο 128. Όπως ήταν φυσικό, οι συνολικοί κύκλοι μειώθηκαν δραματικά. Όπως γνωρίζουμε φυσικά αυτή η ταχύτητα που επιτεύχθηκε δεν αντανακλά την πραγματικότητα, καθώς η βασική συνάρτηση critical και με την σειρά τους οι critical_c1 και critical_c2, θα πρέπει πλέον να προσπελαύνουν την DRAM και όχι την BRAM. Τα αποτελέσματα παρουσιάζονται στην ενότητα 4.2.5. Παρακάτω παρατίθεται το block diagram της υλοποίησης αυτής.



Εικόνα 3.6: Block diagram 4ης Υλοποίησης.

5η Υλοποίηση (bufferless 4 cores): Η νέα σχεδίαση αυτή αποτελεί συνέχεια της προηγούμενης, καθώς ο kernel πλέον διαχωρίστηκε σε 4 πυρήνες, οι οποί μοιράζουν την δουλειά του υπολογισμού των long-ranged non-bonded δυνάμεων. Όπως έγινε και στις προηγούμενες σχεδιάσεις ο κάθε buffer των threads διαχωρίστηκε σε 4 τέσσερις νέους buffers με μέγεθος ο καθένας 32 θέσεων. Σκοπός της σχεδίασης είναι να εκμεταλλευτεί και άλλο την ανεξαρτησία του εξωτερικού loop, των 128 επαναλήψεων, και να το διαχωρίσει σε τέσσερις νέους πυρήνες, στον οποίο το loop αυτό θα εκτελεστεί για 32 επαναλήψεις μόνο. Και οι τέσσερις αυτές συναρτήσεις θα εκτελεσθούν παράλληλα και άρα θα υποδιπλασιαστεί θεωρητικά περίπου ο χρόνος εκτέλεσης σε σχέση με την προηγούμενη σχεδίαση. Και σε αυτήν την σχεδίαση οι μεγάλοι buffers παρέμειναν στην DRAM, οπότε γνωρίζουμε ότι οι χρόνοι είναι πλασματικοί και δείχνουν μόνο το χρόνο εκτέλεσης των υπολογισμών χωρίς το I/O. Η υλοποίηση αυτής της σχεδίασης απέδειξε και πρακτικά την αρχική σκέψη υποδιπλασιασμού του χρόνου. Τα αποτελέσματα παρουσιάζονται στο 4.2.6. Παρακάτω φαίνεται το block diagram της τελευταίας υλοποίησης.



Εικόνα 3.7: Block diagram 5ης υλοποίησης.

Κεφάλαιο 4

Το κεφάλαιο αυτό έχει ως στόχο την παρουσίαση και ανάλυση των αποτελεσμάτων της κάθε αρχιτεκτονικής που σχεδιάστηκε μέσα από τις προσομοιώσεις του Vivado HLS. Επίσης η κάθε αρχιτεκτονική θα συγκριθεί με το software και θα υπολογιστεί η κάθε δυνατή ταχύτητα που μπορούσε να προσφέρει. Η παρουσίαση των αποτελεσμάτων θα γίνει μέσα από ειδικά διαγράμματα.

4.1 Τρόπος επικύρωσης αποτελεσμάτων

Κατά την δοκιμή των αποτελεσμάτων, αρχικά στο επίπεδο του software που αναπτύχθηκε στην C, χρησιμοποιήθηκαν δύο πίνακες ως έξοδοι του συστήματος. Ο πρώτος πίνακας είναι ο forces, ο οποίος μεταφέρει την πληροφορία για τις δυνάμεις που ασκούνται σε ένα άτομο και αποτελείται από τέσσερις float αριθμούς. Ο δεύτερος είναι ο virials και υπολογίζεται από το virial theorem, το οποίο παρέχει μία εξίσωση που αφορά τη μέση διάρκεια του χρόνου της συνολικής κινητικής ενέργειας ⟨T⟩, ενός σταθερού συστήματος που αποτελείται από N σωματίδια, που δεσμεύονται από το δυναμικό των δυνάμεων, με εκείνη της συνολικής δυναμικής ενέργειας. Οι δύο buffers αυτοί παράγονται σταδιακά και ολοκληρώνονται αφού τελειώσουν και τα δύο streams. Τότε οι buffers επιστρέφονται στην main όπου και συγκρίνονται με κατάλληλες συναρτήσεις με τους αντίστοιχους buffers που έχουν παραχθεί από το software της CUDA. Αξίζει να σημειωθεί ότι τα αποτελέσματα διαφέρουν και ο λόγος είναι ότι η CUDA διαβάζει με διαφορετικό τρόπο τρεις buffers, τους force_table, energy_table και lj_table. Η απόκλιση στον buffer του forces είναι της τάξης του πρώτου με δεύτερου δεκαδικού ενώ στον virials είναι πολύ μεγαλύτερο και οφείλεται στις πολλές περισσότερες πράξεις που εκτελούνται πάνω του και σταδιακά αλλοιώνουν πολύ το αποτέλεσμα.

4.2 Παρουσίαση Αποτελεσμάτων

Σε αυτή την ενότητα θα γίνει η παρουσίαση των αποτελεσμάτων μέσω γραφημάτων και η θεωρητική ανάλυση τους. Κατά την παρουσίαση τους έγινε στις πιο σημαντικές από τις σχεδιάσεις μια σύγκριση στην ταχύτητα με τον αρχικό software του NAMD 2.10. Για την σύγκριση χρησιμοποιήθηκε μόνο ο χρόνος που επιτεύχθηκε με το CHARM++ για 1 πυρήνα. Αν και η μετατροπή δεν έγινε από γλώσσα CHARM++ αλλά από την CUDA, δεν χρησιμοποιούνται οι χρόνοι της καθώς ήταν πολύ καλύτεροι από την CHARM++. Πρωταρχικός στόχος ήταν η επιτάχυνση του κώδικα και επίτευξη μεγαλύτερης ταχύτητας από το CPU software γραμμένο σε CHARM++.

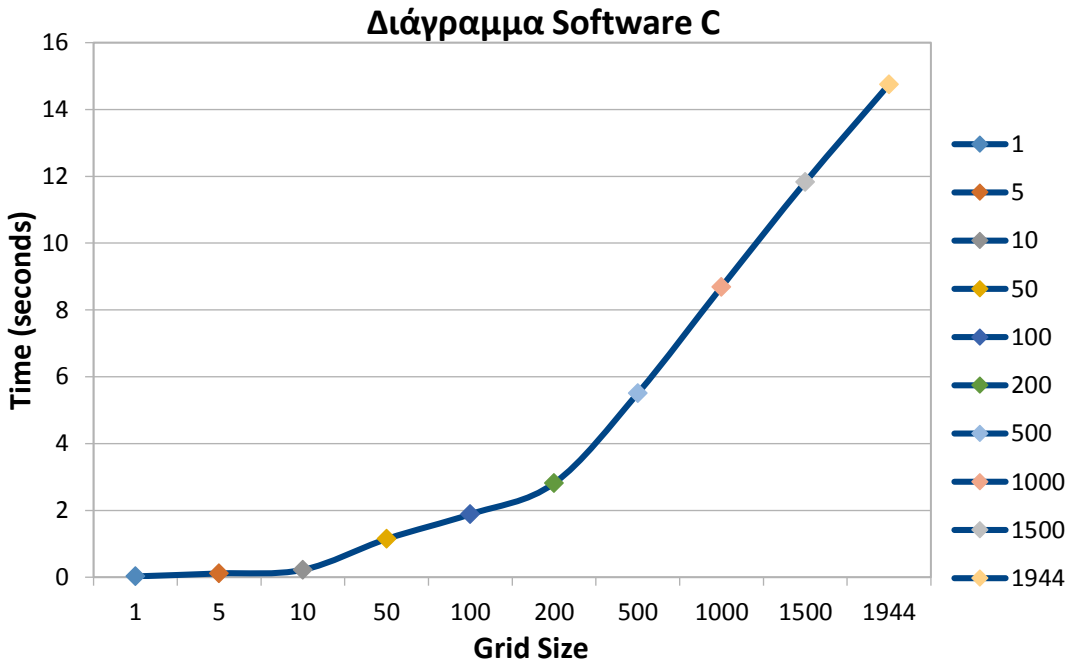
4.2.1 Αποτελέσματα ταχύτητας Software C.

Το πρόγραμμα που δημιουργήθηκε έπειτα την μετατροπή του κώδικα από CUDA σε C εκτελέστηκε σε έναν συμβατικό υπολογιστή με τα εξής χαρακτηριστικά: CPU Intel Core i5-4690K (4 cores – 4 threads) στα 3.9 GHz και 8 GB RAM και λειτουργικό σύστημα Linux Ubuntu 15.04

64bit. Το πρόγραμμα έτρεξε για διαφορετικό αριθμό εισόδων, πειράζοντας σε κάθε περίπτωση το grid_size. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα και στο παρακάτω διάγραμμα. Η CUDA έκδοση του κώδικα NAMD έτρεχε σε μία NVIDIA Ge-Force 980 GTX. **Ο χρόνος που χρειαζόταν σε ένα timestep να ολοκληρωθεί η διαδικασία στην CUDA (kernel-only) ήταν 20.1 ms ενώ με το CHARM++ ήταν 600 ms. Μετρώντας όμως και το I/O στην GPU ο χρόνος αυξανόταν στα 188.7 ms.**

Software C	
Grid Size (Stream x2)	Time (Seconds)
1	0.024196
5	0.112799
10	0.221513
50	1.1452
100	1.886024
200	2.813905
500	5.507171
1000	8.690295
1500	11.825162
1944	14.750039

Πίνακας 4.1: *Χρόνος εκτέλεσης C για διαφορετικό Grid Size.*



Γράφημα 4.1: *Χρόνος Εκτέλεσης C*

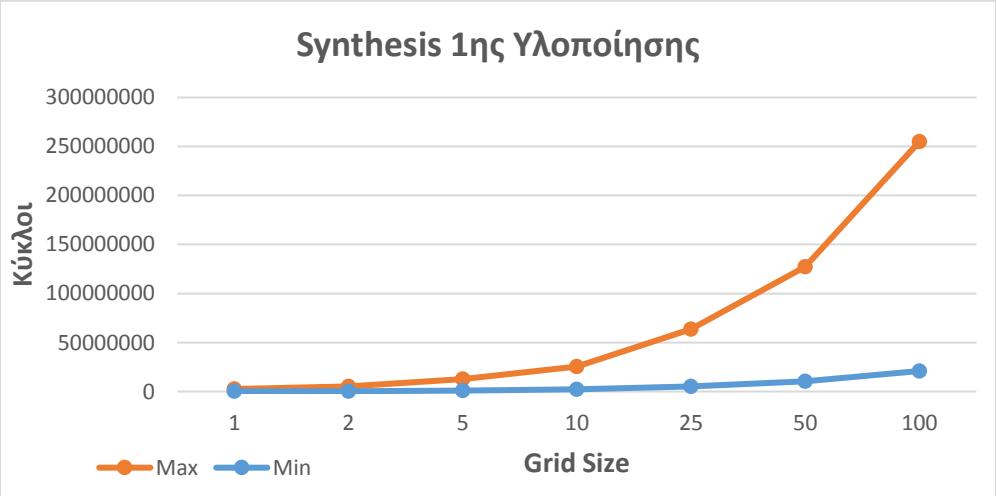
Στο διάγραμμα παρατηρείται ότι υπάρχει γραμμικότητα ως προς τον χρόνο εκτέλεσης και την είσοδο. Αυτό αποδεικνύει ότι ο κώδικας παράχθηκε από έναν αλγόριθμο με μεγάλη παραλληλία και η οποία μπορεί να εκμεταλλευτεί για να επιτευχθεί επιτάχυνση στην FPGA.

4.2.2 Αποτελέσματα 1ης Υλοποίησης από Vivado HLS

Κατά την πρώτη υλοποίηση λόγω του όγκου του προγράμματος ήταν αδύνατο να παρθούν αποτελέσματα με C/RTL Co-simulation, για αυτό τον λόγο παρακάτω παρουσιάζονται τα αποτελέσματα από διαφορετικές χρήσεις των directives μόνο μετά το synthesis. Στα πρώτα αποτελέσματα δεν έχει χρησιμοποιηθεί κανένα directive και φαίνονται οι ελάχιστοι και οι μέγιστοι κύκλοι όπως παράγονται από το synthesis του Vivado HLS. Στο γράφημα 4.2 δεν συμπεριλαμβάνεται η τιμή 1944, διότι οι αντίστοιχες τιμές των κύκλων είναι τεράστιες και οπότε δεν φαίνονται με ακρίβεια οι μικρότερες τιμές.

Vivado HLS			
Grid Size	Min Cycles	Max Cycles	Clock
1	209138	2548474	10.53
2	418274	5096946	10.53
5	1045682	12742362	10.53
10	2091362	25484722	10.53
25	5228402	63711802	10.53
50	10456802	127423602	10.53
100	20913602	254847202	10.53
1944	406560386	4954229570	10.53

Πίνακας 4.2: *Κύκλοι από το synthesis του Vivado HLS σε σχέση με το Grid Size.*



Γράφημα 4.2: *Μέγιστοι και Ελάχιστοι Κύκλοι όπως προκύπτουν από το Synthesis, χωρίς την τιμή του 1944.*

Η γραμμικότητα που παρατηρήθηκε στο software της C έχει αλλάξει και μοιάζει περισσότερο με εκθετική. Ο λόγος δεν είναι ότι ο αλγόριθμος άλλαξε αλλά το εργαλείο για μεγαλύτερη είσοδο θέτει πολύ παραπάνω κύκλους. Αυτό αργότερα εξαλείφεται με τις επόμενες αρχιτεκτονικές και με την χρήση των directives.

Στην ίδια σχεδίαση έπειτα χρησιμοποιήθηκαν directives. Η χρήση του κάθε directive και οι συνδυασμοί τους είχαν διαφορετικά αποτελέσματα. Όλα αυτά παρουσιάζονται στον παρακάτω Πίνακα 4.3. Για χάριν ευκολίας το synthesis έγινε μόνο για grid size ίσο με 1 και συγκρίθηκε με αυτό της σχεδίασης χωρίς directives.

Grid Size 1			
	Min	Max	Clock
1) *Pipelines Only*	12715	21171561	8.7
2) *Pipelines and Unroll (Blocks Stream)*	4523	2343859	10.98
3) *Pipelines και σε όλα τα loops unroll*	2635	2232841	7.1
4) *Pipeline και unroll σε λιγότερα loops (όπου δεν υπήρχαν dependencies)*	12691	2254969	7.1
5) *Pipeline και unroll και dependencies v2*	4567	2246333	7.1
6*) τα loops των streams και των blocks μέσα στον κώδικα v3*	5454	2185436	8.02

Πίνακας 4.3: *Κύκλοι για διαφορετικές περιπτώσεις χρήσης directives για grid size ίσο με 1.*

Γράφημα 4.3: *Ελάχιστοι κύκλοι ανά περίπτωση.*

Γράφημα 4.4: *Μέγιστοι κύκλοι ανά περίπτωση.*

Όπως παρατηρείται από τα παραπάνω γραφήματα η χρήση των directives όντως επιταχύνουν την σχεδίαση και συγκεκριμένα ο συνδυασμός τους. Στην 1η περίπτωση ειδικότερα η χρήση μόνο του directive PIPELINE, μειώνει τον ελάχιστο αριθμό των κύκλων αλλά αυξάνει κατά πολύ τον μέγιστο (μία τάξη μεγέθους μεγαλύτερο), όπως παρατηρείται από τους πίνακες 4.2 και 4.3. Έπειτα όμως στις επόμενες περιπτώσεις ο συνδυασμός και άλλων directives βοηθάει στο να μειωθούν και κάτω από την αρχική σχεδίαση.

Η 1η υλοποίηση δεν έφερε τα επιθυμητά αποτελέσματα, καθώς ο κώδικας που έπρεπε να μετατραπεί σε hardware ήταν ιδιαίτερα πολύπλοκος. Παρόλαυτά με την 1η υλοποίηση υπήρξε εξοικείωση με το εργαλείο Vivado HLS.

4.2.3 Αποτελέσματα 2ης Υλοποίησης από Vivado HLS

Κατά την 2η υλοποίηση με το profiling που έγινε πάνω στον κώδικα της C, παρατηρήθηκε ότι ο περισσότερος χρόνος καταναλώνεται σε ένα συγκεκριμένο βρόγχο του προγράμματος. Επιλέχθηκε λοιπόν, μόνο ο βρόγχος αυτός να μετατραπεί σε hardware. Τα αποτελέσματα φαίνονται στον πίνακα 4.4. Στον πίνακα αυτόν φαίνονται μόνο τα synthesis για grid size 1944 (max).

Synthesis		
CRITICAL SECTION ONLY	Min	Max
1) Χωρίς directives	8450	733442
2) Pipeline στο εσωτερικό loop	126082	126082
3) Περίπτωση 2 συν Loop Unrolling factor=2 (εξωτερικό loop)	126018	126018
4) Περίπτωση 3 συν dependencies και array partition	105602	105602
5) Εσωτερικό loop - Pipeline, Loop Unrolling, Dependencies	100471	100471

Πίνακας 4.4: *Ελάχιστοι και Μέγιστοι κύκλοι για κάθε περίπτωση χρήσης directives.*

Γράφημα 4.5: *Ελάχιστοι κύκλοι ανά περίπτωση.*

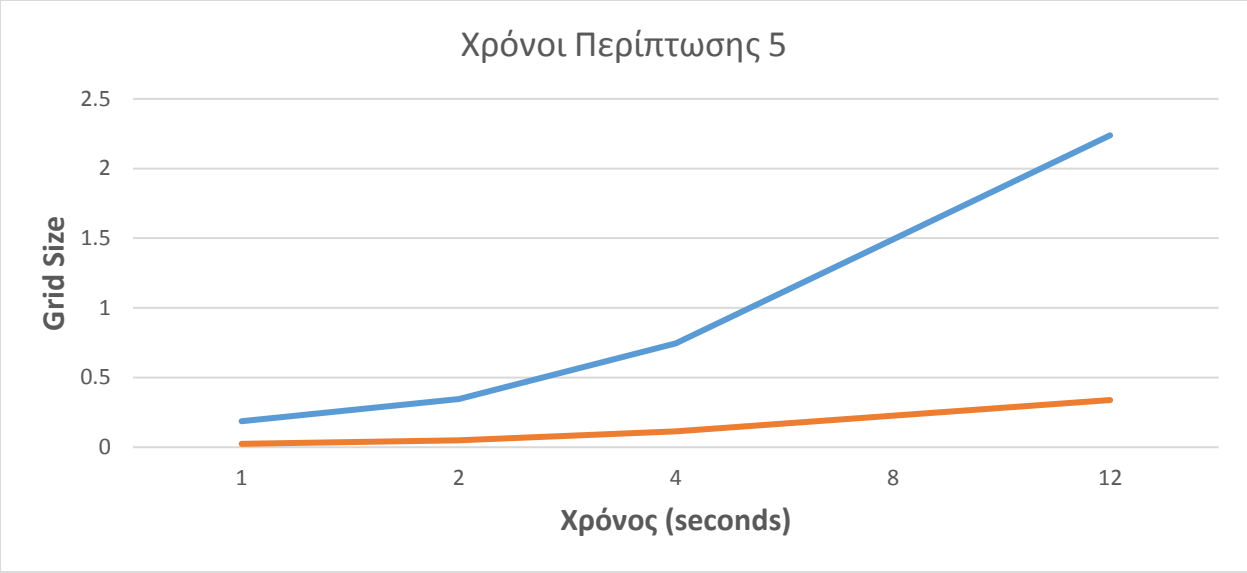
Γράφημα 4.6: *Μέγιστοι κύκλοι ανά περίπτωση.*

Η παρατήρηση του πίνακα και των γραφημάτων δείχνει πως με την χρήση των directives οι ελάχιστοι και μέγιστοι κύκλοι γίνονται ίδιοι σε κάθε περίπτωση, που δείχνει ότι ο κώδικας απλουστεύεται πολύ περισσότερο για το εργαλείο. Επίσης οι κύκλοι είναι πολύ λιγότεροι, αφού το synthesis δείχνει μόνο τους κύκλους ενός μόνο call του hardware.

Μετά τα αποτελέσματα του synthesis έγινε προσπάθεια προσομοίωσης του κώδικα με το C/RTL Co-simulation του Vivado HLS. Το simulation έγινε προφανώς μόνο στην τελευταία περίπτωση αφού εκεί παρατηρήθηκαν τα καλύτερα αποτελέσματα από το synthesis. Όπως είναι λογικό το simulation έγινε για μικρό grid size, καθώς αποτελεί μία πάρα πολύ χρονοβόρα διαδικασία. Τα grid sizes που χρησιμοποιήθηκαν είναι το 1, 2, 4, 8 και 12. Για την περίπτωση του grid size ίσο με 1944 έγινε προβολή (projection) από τα κοντινότερα αποτελέσματα, δηλαδή από grid size ίσο με 12. Τα αποτελέσματα φαίνονται παρακάτω.

Περίπτωση 5 – C/RTL Co-simulation	
Grid Size	Time (seconds)
1	0,1865
2	0,3454
4	0,7462
8	1,4924
12	2,2386
1944 expected	362,6532

Πίνακας 4.5: *Χρόνος ανά grid size και χρόνος που εκτιμάται για grid size 1944.*



Γράφημα 4.7: *Χρόνος σε δευτερόλεπτα για διαφορετικά grid sizes. Παραλείπεται το 1944 για να φανεί η γραμμικότητα.*

Σε αυτή την σχεδίαση παρατηρείται ότι ο χρόνος αυξάνεται γραμμικά με το grid size. Επίσης παραλείφθηκε το 1944 από το γράφημα για να φαίνονται καλύτερα τα αποτελέσματα. Από τον πίνακα 4.5 φαίνεται πως με την σχεδίαση αυτή ο τελικός χρόνος εκτέλεσης θα είναι 362,6532 seconds ή περίπου 6 λεπτά. Ο χρόνος αυτό φυσικά δεν είναι ικανοποιητικός αφού ο αντίστοιχος χρόνος για μία εκτέλεση του κώδικα είναι μόλις 10 ns και ο αντίστοιχος χρόνος της CPU είναι 0,5 seconds. Επίσης στην σχεδίαση αυτή παρατηρήθηκε τεράστια μείωση στους πόρους της FPGA. Πιο συγκεκριμένα για την BRAM υπάρχει μηδενική χρήση (0%), για τα DSP48E 3%, για FFs 4% και τέλος για LUT 11%.

4.2.4 Αποτελέσματα 3ης Υλοποίησης από Vivado HLS

Στην σχεδίαση αυτή εκμεταλλευόμενοι την γραμμικότητα του κώδικα, το critical section έσπασε σε 2 cores, οι οποίοι μοιράζονται την δουλειά. Επίσης γίνεται και εγγραφή όλων των εισόδων σε BRAM ώστε να εκμεταλλευτούμε την ταχύτητα της. Τα directives για το κάθε core παραμένουν ίδια από την προηγούμενη σχεδίαση. Το synthesis παρουσίασε τα παρακάτω αποτελέσματα για grid size πάντα ίσο με 1944. Επίσης ελέγχθηκε και σε άλλη FPGA την ZedBoard (Πίνακας 4.6). Το target clock που επιλέχθηκε και στις δύο περιπτώσεις είναι τα 5 ns, παρόλαυτά το εργαλείο δεν κατάφερε να φτάσει τον χρόνο αυτό για κάθε κύκλο.

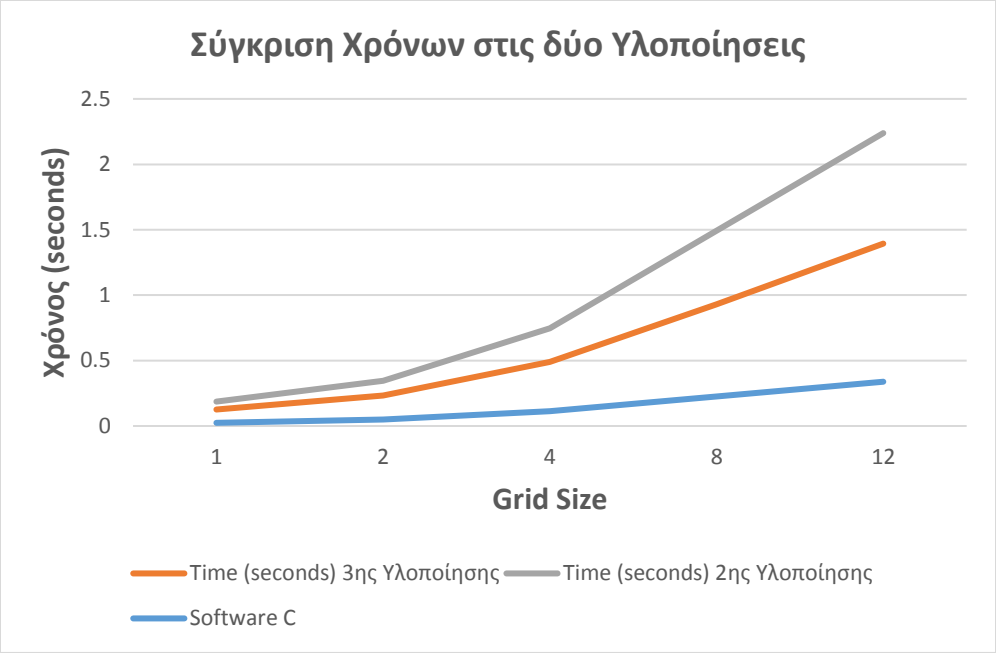
CRITICAL SECTION 2 CORES - Synthesis			
FPGA	**Min**	**Max**	**Clk (ns)**
Zynq ZC 706	74543	74543	8,7
ZedBoard	78646	78646	8,7

Πίνακας 4.6: *Κύκλοι για ένα call του critical section για δύο διαφορετικές FPGAs.*

Από τον παραπάνω πίνακα παρατηρείται μία μεγάλη μείωση των κύκλων για ένα call, σε σχέση με την προηγούμενη σχεδίαση. Επίσης φαίνεται πως η μείωση δεν είναι κοντά στο 50% όπως θα περιμέναμε θεωρητικά, αφού η σχεδίαση έσπασε σε δύο παράλληλα κομμάτια. Αυτό συμβαίνει λόγω των κύκλων που χρειάζεται στην αρχή του κάθε call της συνάρτησης για να εγγραφούν οι είσοδοι στην BRAM. Όπως αναφέρεται και στο προηγούμενο κεφάλαιο το loop αρχικοποίησης είναι 12544 επαναλήψεων. Η αντίστοιχη μείωση παρατηρείται και στους χρόνους που παράχθηκαν από το εργαλείο κατά το C/RTL Co-Simulation. Τα αποτελέσματα φαίνονται παρακάτω.

CRITICAL SECTION 2 CORES - C/RTL		
Grid Size	**Time (seconds) 3ης Υλοποίησης**	**Time (seconds) 2ης Υλοποίησης**
1	0.1259	0.1865
2	0.232915	0.3454
4	0.4891215	0.7462
8	0.92933085	1.4924
12	1.393996275	2.2386
1944 expected	225.8273966	362.6532

Πίνακας 4.7: *Χρόνος ανά grid size και χρόνος που εκτιμάται για grid size 1944. Παρατίθενται και τα αποτελέσματα της προηγούμενης υλοποίησης για σύγκριση.*



Γράφημα 4.8: *Σύγκριση χρόνων από C/RTL Co-Simulation για τις δύο υλοποιήσεις. Παραλείπονται οι τιμές για 1944.*

Όπως και στην προηγούμενη υλοποίηση παρατηρείται γραμμικότητα του χρόνου ως προς την μέγεθος της εισόδου και οι χρόνοι είναι αρκετά καλύτεροι. Παρόλαυτά για τον συνολικό χρόνο, δηλαδή για μέγεθος εισόδου ίση με το 1944, τα αποτελέσματα είναι ακόμα μη ικανοποιητικά.

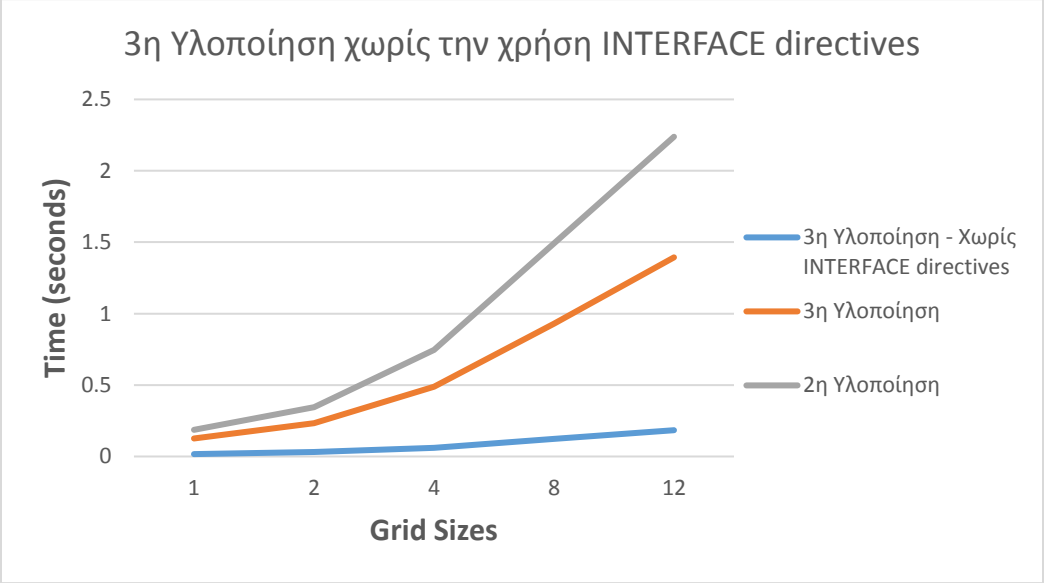
Η υλοποίηση αυτή ήταν πιο απαιτητική ως προς τους πόρους της FPGA. Πιο συγκεκριμένα ολόκληρη η σχεδίαση χρειάζεται 26% της BRAM, 2% των DSP48Es, 6% των FFs και 13% των LUTs. Τα αποτελέσματα αυτά είναι για την Zynq ZC 706. Με απλούς υπολογισμούς φαίνεται πως η σχεδίαση αυτή μπορεί να σπάσει για συνολικά 6 πυρήνες, μέχρι να εξαντληθούν οι πόροι της FPGA. Για τους πυρήνες αυτούς αν θεωρήσουμε ότι η σχεδίαση ακολουθεί την ίδια γραμμικότητα με αυτή με τους 2 πυρήνες, τότε ο ελάχιστος χρόνος που θα επιτευχθεί θεωρητικά είναι περίπου 100,63 δευτερόλεπτα ή περίπου 1,6 λεπτά. Ο χρόνος αυτός απέχει από τον χρόνο του software του NAMD 2.10 για CPU. Στην σχεδίαση αυτή ακόμα και με μία ιδανική FPGA, η οποία θα μπορούσε να χωρέσει 128 πυρήνες, ο ελάχιστος χρόνος υπολογίζεται, λόγω της γραμμικότητας του αλγορίθμου, στα 13,5 δευτερόλεπτα. Σε αυτή την περίπτωση υπάρχει ένα speed down ίσο με 27 σε σχέση με το CPU NAMD με CHARM++.

Χρήση Interface Directives: Κατά την διάρκεια υλοποίησης αυτής της σχεδίασης εντοπίστηκε ότι χρησιμοποιήθηκαν εσφαλμένα τα INTERFACE directives. Χωρίς αυτά οι συνολικοί κύκλοι έπεσαν στους 14735 και επίσης επιτεύχθηκε το ρολόι να είναι κάτω από το ρολόι που τέθηκε ως στόχος. Συγκεκριμένα σε όλες τι προηγούμενες σχεδιάσεις το ρολόι που επιτεύχθηκε ήταν αρκετά μεγαλύτερο από τον στόχο. Στην σχεδίαση αυτή στην FPGA Zynq ZC

706, ο στόχος ήταν τα 5 ns και επιτεύχθηκε 4.32 ns δηλαδή μέσα στο όριο. Παρακάτω θα παρουσιαστούν τα αποτελέσματα από το C/RTL Co-simulation.

Grid Size	Time (picoseconds)	Time (seconds)
1	33467882000	0.033467882
2	61604364000	0.061604364
4	123208729000	0.123208729
8	246417458000	0.246417458
12	369626186000	0.369626186
1944	2,97908E+13	29,79076725

Πίνακας 4.8: *Χρόνοι σε picoseconds και nanoseconds ανάλογα διαφορετικά Grid Sizes. Στην σχεδίαση αυτή δεν χρησιμοποιήθηκαν INTERFACE directives.*



Γράφημα 4.9: *Γραφική αναπαράσταση των χρόνων σε seconds σε σχέση με το grid size. Παραλείπονται οι τιμές για 1944. Στο γράφημα αυτό παρουσιάζονται ταυτόχρονα και οι προηγούμενες δύο σχεδιάσεις ξανά για διευκόλυνση στην σύγκριση τους.*

Η εσφαλμένη χρήση των directives δημιουργούσε μεγάλο πρόβλημα κατά την σχεδίαση, καθώς μείωνε την απόδοση της σχεδίασης αισθητά. Η σχεδίαση παρόλαυτά, δεν μπορεί πάλι να μειωθεί κάτω από τους 12544 κύκλους περίπου, καθώς όπως αναφέρθηκε, χρειάζονται για την αρχικοποίηση των buffers στην BRAM. Η παραπάνω σχεδίαση έγινε και για 4 πυρήνες. Το

αποτέλεσμα φυσικά δεν άλλαξε δραματικά. Η μείωση στους κύκλους όπως και ο χρόνος στο C/RTL Co-simulation ήταν μικρή. Συγκεκριμένα οι κύκλοι του κάθε call της βασικής συνάρτησης critical ήταν 13711 και ο συνολικός χρόνος για Grid Size ίσο με 1 ήταν 15493567500 ps ή 0.01549 seconds.

Οι πόροι που χρειάστηκαν για την σχεδίαση με τους 2 πυρήνες είναι οι εξής: 26% BRAM, 22% DSP48E, 7% FF και 18% LUT. Για την επόμενη σχεδίαση με τους 4 πυρήνες όπως ήταν λογικό οι πόροι που χρησιμοποιήθηκαν σχεδόν διπλασιάστηκαν: 53% BRAM, 44% DSP48E, 15% FF και 36% LUT.

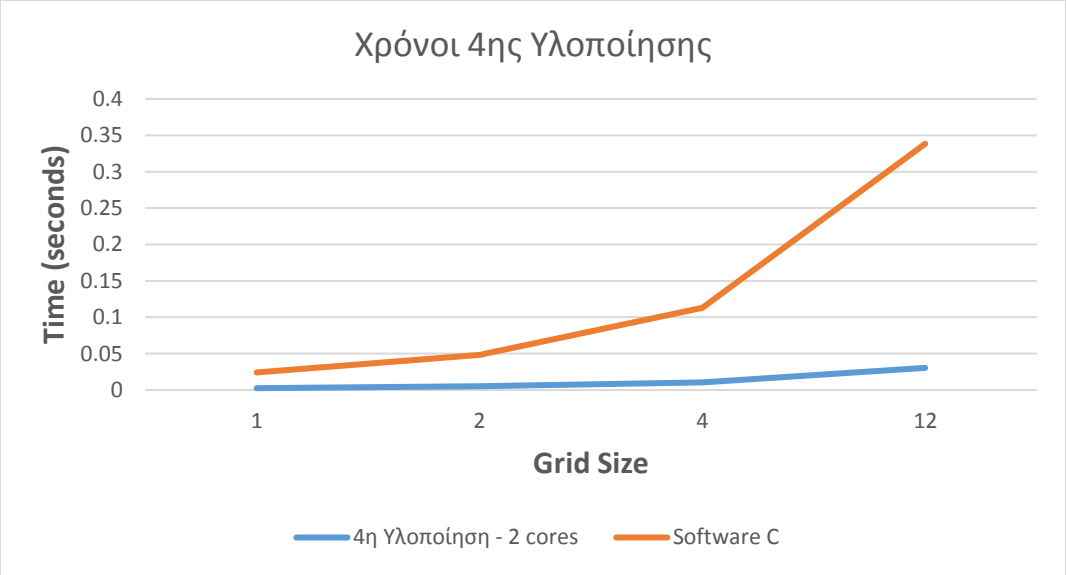
Οι παραπάνω σχεδιάσεις αποτελούν τις πιο ρεαλιστικές προσεγγίσεις, καθώς εκτελέστηκαν για αρκετά παραδείγματα Grid Sizes και επίσης λαμβάνουν υπόψιν τους και το I/O. Η εκτέλεση του NAMD 2.10 με CHARM++ για 1 πυρήνα εκτελέστηκε για 0,6 seconds, στο μηχάνημα που έχει ήδη αναφερθεί. Η τελική σχεδίαση με τους 4 πυρήνες υπολογίζεται, με Grid Size ίσο με 1944, στα 27,1 seconds. Η μείωση όπως προαναφέρθηκε δεν είναι μεγάλη και το speed-down είναι περίπου στο **41x** σε σχέση με το CPU NAMD με CHARM++. Σε σχέση με τον κώδικα της CUDA είναι **158x** αργότερο.

4.2.5 Αποτελέσματα 4ης Υλοποίησης από Vivado HLS

Η προηγούμενη υλοποίηση αποτελεί μία πιο πραγματική προσέγγιση του προβλήματος, καθώς λαμβάνει υπόψιν της και το πρόβλημα του I/O. Στην 4^{η} υλοποίηση οι μεγάλοι buffers που γινόταν η αρχικοποίηση τους στην BRAM, τώρα χρησιμοποιούνται απευθείας από την DRAM. Με αυτό τον τρόπο μετρούνται μόνο οι χρόνοι που χρειάζεται ο kernel για τον υπολογισμό των δυνάμεων. Παρακάτω παρουσιάζονται οι χρόνοι για διαφορετικά grid sizes.

Grid Size	Time (picoseconds)	Time (seconds)
1	2620607500	0,002620608
2	5241215000	0,005241215
4	10366067500	0,010366068
12	30494987500	0,030494988
1944 expected	4,94019E+12	4,940187975

Πίνακας 4.9: *Χρόνοι σε picoseconds και nanoseconds ανάλογα διαφορετικά Grid Sizes.*



Γράφημα 4.10: *Γραφική αναπαράσταση των χρόνων σε seconds σε σχέση με το grid size. Παραλείπεται η τιμή για 1944.*

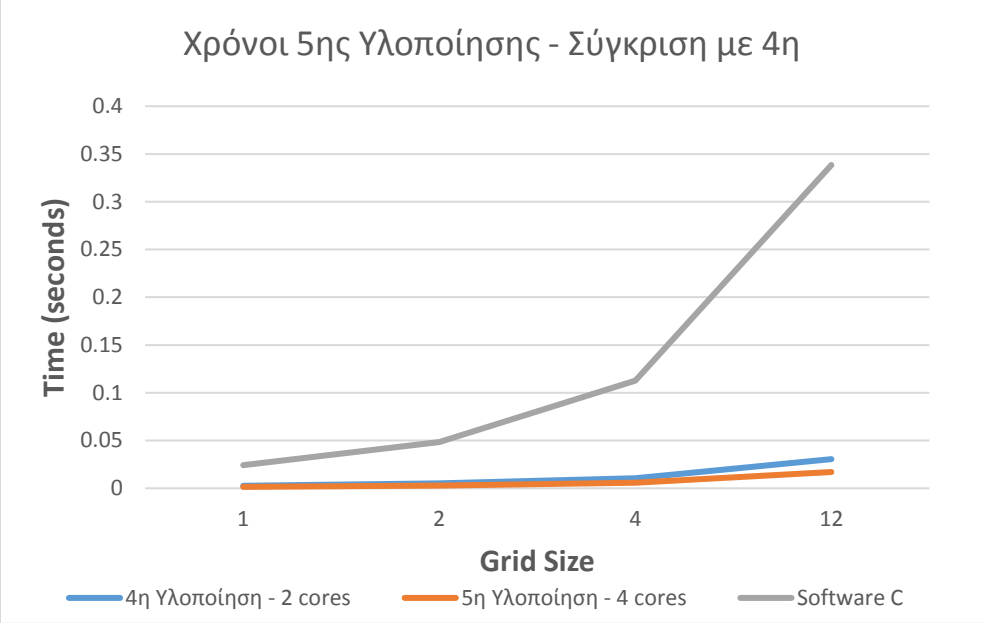
Η παρατήρηση των παραπάνω μετρήσεων οδηγούν στο συμπέρασμα ότι ο kernel έχει επιταχυνθεί κατά πολύ χωρίς το I/O. Αυτό φυσικά συμβαίνει καθώς πλέον δεν υπάρχει η αρχικοποίηση που περιόριζε την σχεδίαση τουλάχιστον στους 12544 κύκλους ανά κάλεσμα της βασικής συνάρτησης. Οι συνολικοί κύκλοι που χρειάζεται πλέον για κάθε τέτοιο κάλεσμα είναι 2319. Επίσης οι πόροι που χρειάζονται πλέον είναι λιγότεροι καθώς δεν χρησιμοποιείται τόσο ποσοστό της BRAM. Συγκεκριμένα, έχουμε 2% BRAM, 22% DSP48E, 7% FFs και 17% LUTs. Αυτό σημαίνει ότι στην Zynq ZC 706, μπορεί να διαχωριστεί και να μοιραστεί η δουλειά του kernel σε 8 πυρήνες μέχρι να εξαντληθούν οι πόροι της FPGA. Υποθέτοντας ότι διατηρείται θεωρητικά η γραμμικότητα του αλγορίθμου, τότε ο συνολικός χρόνος θα μειωθεί στα 1,225 δευτερόλεπτα. Όπως γίνεται αντιληπτό το μόνο που περιορίζει την σχεδίαση αυτή είναι οι πόροι της FPGA. Μία FPGA ελαφρώς μεγαλύτερη ώστε να μπορεί να χωρέσει τουλάχιστον 12 πυρήνες τότε ο χρόνος θα έπεφτε στα 0,30625 seconds. Αυτό σημαίνει ότι θα επιτευχθεί ένα speedup 1,6. Ο υποδιπλασιασμός του χρόνου θα επιτευχθεί στην επόμενη και τελευταία σχεδίαση και θα αποδείξει την γραμμικότητα του αλγορίθμου. Σε αυτή την υλοποίηση έχουμε **7x speed down** από τον CPU κώδικα και επίσης είναι **235x** αργότερο από την GPU.

4.2.6 Αποτελέσματα 5ης Υλοποίησης από Vivado HLS

Η τελευταία σχεδίαση αυτή υλοποιήθηκε για να αποδειχθεί ότι η 4^{η} υλοποίηση μπορεί να διαχωριστεί περεταίρω, όπως επίσης αποδεικνύει την γραμμικότητα του αλγορίθμου. Στην σχεδίαση αυτή οι δύο πυρήνες kernel έγιναν τέσσερις. Όπως και στην προηγούμενη σχεδίαση, οι μεγάλοι buffers διπλασιάστηκαν, που σημαίνει ότι πλέον υπάρχουν 4 αντίγραφα του καθενός και τα οποία στέλνονται κάθε φορά σαν είσοδοι στον κάθε πυρήνα. Οι μικρότεροι buffers που διατηρούν την πληροφορία για το κάθε thread διαχωρίστηκαν σε 4 πλέον buffers ο καθένας, μεγέθους ¼ του αρχικού μεγέθους (32 θέσεις). Παρακάτω φαίνονται τα αποτελέσματα από τους χρόνους εκτέλεσης για διαφορετικά grid sizes.

Grid Size	Time (picoseconds)	Time (seconds)
1	1463487500	0,001463488
2	2926975000	0,002926975
4	5788787500	0,005788788
12	17029387500	0,017029388
1944 expected	2,75876E+12	2,758760775

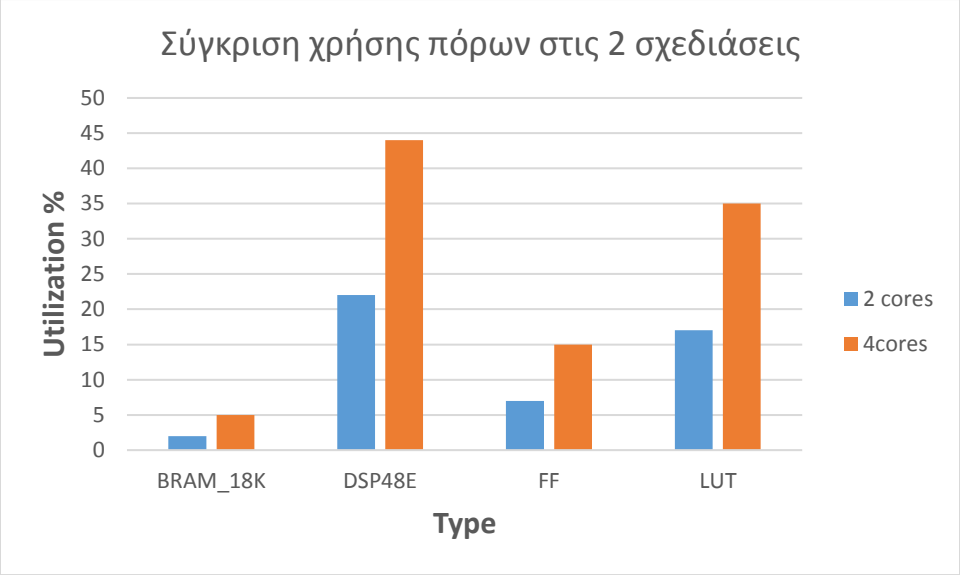
Πίνακας 4.10: *Χρόνοι σε picoseconds και nanoseconds ανάλογα διαφορετικά Grid Sizes.*



Γράφημα 4.11: *Γραφική αναπαράσταση των χρόνων σε seconds σε σχέση με το grid size. Σύγκριση ανάμεσα στις 2 σχεδιάσεις. Παραλείπεται η τιμή για 1944.*

Όπως αναμενόταν ο χρόνος της τελευταίας σχεδίασης με τους 4 πυρήνες είναι σχεδόν υποδιπλάσιος του αρχικού με τους 2. Αυτό φασικά αποδεικνύει και την γραμμικότητα του αλγορίθμου. Τα speed downs προφανώς υποδιπλασιάστηκαν σχεδόν αντίστοιχα και συγκεκριμένα για την CPU έκδοση του NAMD με CHARM++ υπάρχει speed down της τάξεως του **3.8x** και από την GPU έκδοση με CUDA **130x**.

Όσον αφορά τους πόρους, στην BRAM καταναλώνονται το 5%, στα DSP48Es 44%, στα FFs 15% και στα LUTs 35%. Όπως ήταν φυσιολογικό ο διπλασιασμός των πυρήνων έφερε και διπλασιασμό των πόρων. Όπως και στην προηγούμενη σχεδίαση στην Zynq ZC 706 αναμένεται να χωρέσουν 8 πυρήνες και ο χρόνος υπολογίζεται να φτάσει στα 1,37 seconds.



Γράφημα 4.12: *Γραφική αναπαράσταση της χρήσης των πόρων στην 4η υλοποίηση (2 cores).*

Με προβολή στους 8 πυρήνες, δηλαδή όσοι χωρούν στην Zynq ZC 706, όπως προαναφέρθηκε ο χρόνος θα έπεφτε στα **1,37 seconds**, όμως δημιουργείται μεγάλο πρόβλημα με το I/O. Συγκεκριμένα, σε κάθε call του critical(), δηλαδή της βασικής συνάρτησης που καλεί τους 8 πυρήνες, χρειάζεται 953 Kilobytes ως είσοδο για την συνάρτηση. Άρα για όλα ta calls της συνάρτησης αυτής χρειάζεται να υπολογιστεί η συνολική είσοδος, η οποία προκύπτει με το γινόμενο του μεγέθους της εισόδου επί όλα τα calls. Εν τέλει προκύπτει ότι η συνολική είσοδος είναι 1718 Terabytes. Άρα το bandwidth που χρειάζεται από την DRAM της FPGA είναι **1780.22235 Terabytes/1.37 sec** ή **1299.4 Terabytes/sec,** κάτι το οποίο φυσικά δεν υποστηρίζει η FPGA. Για τον λόγο αυτό η 4η και 5η υλοποίηση αποτελούν μία θεωρητική προσέγγιση του αλγορίθμου χωρίς να υπολογίζεται το I/O.

4.3 Συγκεντρωτικά Στοιχεία Διπλωματικής Εργασίας

Παρακάτω φαίνονται κάποια συγκεντρωτικά στοιχεία της διπλωματικής εργασίας αυτής σε πίνακες και γραφήματα.

Παρακάτω φαίνεται συγκεντρωτικά σε πίνακα οι πόροι κατανάλωσης της Zynq ZC 706 σε κάθε σχεδίαση.

Πόροι ανά σχεδίαση στην Zynq 706 ZC				
	BRAM_18K (1090)	**DSP48E (900)**	**FF (437200)**	**LUT (218600)**
2η Υλοποίηση	~0%	3%	4%	11%
3η Υλοποίηση	26%	2%	6%	13%
3η Υλοποίηση χωρίς directives	26%	22%	7%	18%
3η Υλοποίηση χωρίς directives 4 cores	53%	44%	15%	36%
4η Υλοποίηση χωρίς I/O	2%	22%	7%	17%
5η Υλοποίηση χωρίς I/O	5%	44%	15%	35%

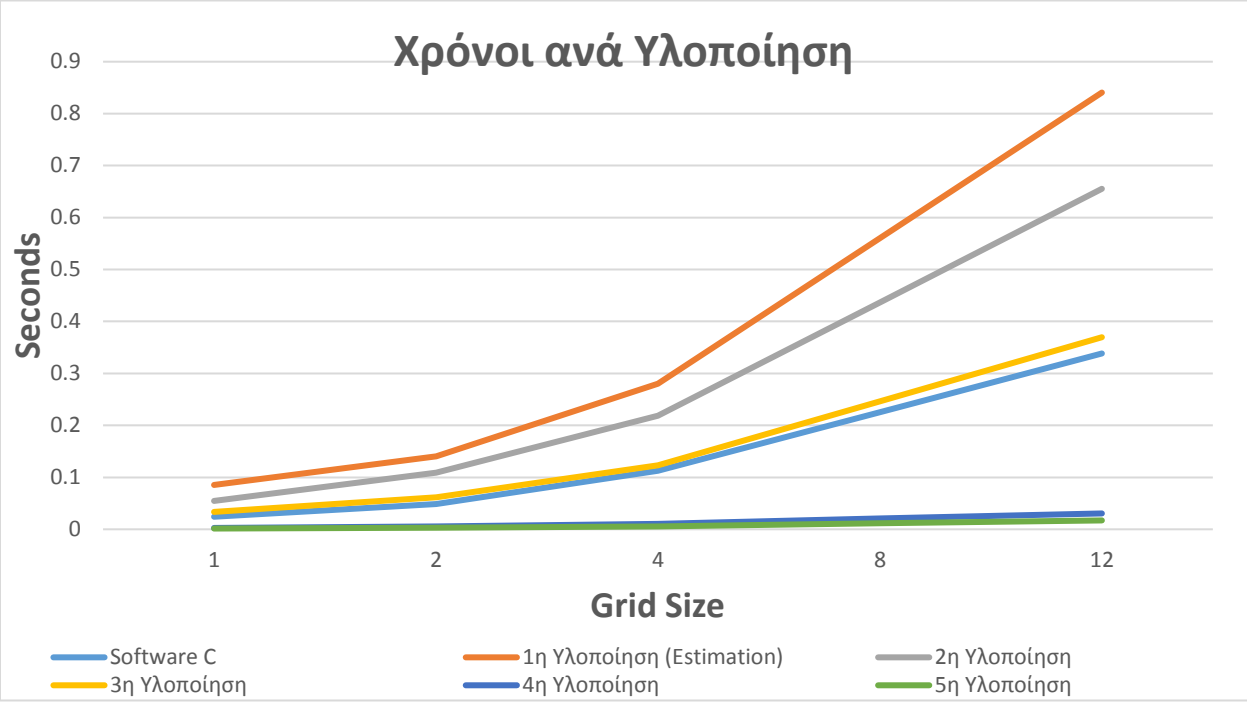
Πίνακας 4.11: Συγκεντρωτικός πίνακας για την κατανάλωση πόρων για όλες τις υλοποιήσεις για την Zynq ZC 706.

Παρακάτω φαίνεται συγκεντρωτικά σε πίνακα οι χρόνοι κάθε σχεδίασης.

Χρόνοι Υλοποιήσεων						
Grid Size	**Software C**	**1η Υλοποίηση (Estimation)**	**2η Υλοποίηση**	**3η Υλοποίηση**	**4η Υλοποίηση**	**5η Υλοποίηση**
1	0.024196	0.085358	0.054607444	0.033467882	0.002620608	0.00146349
2	0.048392	0.140097	0.109214888	0.061604364	0.005241215	0.00292698
4	0.112799	0.280194	0.218429777	0.123208729	0.010366068	0.00578879
8	0.225598	0.560388	0.436859553	0.246417458	0.020732135	0.01157758
12	0.338397	0.840582	0.65528933	0.369626186	0.030494988	0.01702939
1944	**14.750039**	**49.54**	**39.61077293**	**29.79076725**	**4.940187975**	**2.75876078**

Πίνακας 4.12: Συγκεντρωτικός Πίνακας χρόνων σε κάθε σχεδίαση.

Παρακάτω φαίνεται συγκεντρωτικά σε γράφημα οι χρόνοι κάθε σχεδίασης.



Γράφημα 4.13: Συγκεντρωτικό Γράφημα χρόνων σε κάθε σχεδίαση.

4.4 Σύγκριση αποτελεσμάτων με παλαιότερες δουλειές

Ο στόχος της διπλωματικής αυτής εργασίας αποτελεί μία πρώτη προσπάθεια προσέγγισης του πακέτου προσομοίωσης Μοριακής Δυναμικής NAMD από την πλευρά του GPU-accelerated κώδικα. Με βάση αυτό είναι αδύνατο να συγκριθεί με τις προηγούμενες δουλειές σε ASIC τεχνολογίες που έχουν γίνει στον τομέα των προσομοιώσεων Μοριακής Δυναμικής, καθώς αποτελούν πολύ παλαιές εργασίες και σε διαφορετικά πακέτα. Πολλοί μάλιστα από τους αλγορίθμους που χρησιμοποιήθηκαν, δημιουργήθηκαν καθαρά για τις σχεδιάσεις αυτές, και με πολύ συγκεκριμένες προσομοιώσεις. Για να πραγματοποιηθεί μία ορθή σύγκριση θα πρέπει να γίνει με βάση κάποια συγκεκριμένα χαρακτηριστικά. Τα χαρακτηριστικά αυτά είναι: 1) Χρήση ίδιου πακέτου προσομοιώσεων Μοριακής Δυναμικής, 2) κατά την παρουσίαση των αποτελεσμάτων να έχουν χρησιμοποιηθεί ίδιες προσομοιώσεις και 3) οι δύο εργασίες να μην απέχουν μεγάλο διάστημα χρονικά, καθώς αυτό θα αλλοίωνε τον σκοπό της σύγκρισης. Η μόνη τέτοια εργασία που πληροί τις προϋποθέσεις αυτές είναι η διδακτορική διατριβή του Ashfaquzzaman Khan στο Πανεπιστήμιο της Βοστόνης το 2012 [71]. Στην διατριβή του κατάφερε να δημιουργήσει ένα ολοκληρωμένο σύστημα με CPUs και FPGAs και χρησιμοποίησε το πακέτο NAMD 2.8 και για τα αποτελέσματα του χρησιμοποιήθηκε η ίδια προσομοίωση που χρησιμοποιήθηκε και στην παρούσα διπλωματική εργασία δηλαδή η

πρωτεΐνη ApoA1, η οποία δίνεται ως παράδειγμα προσομοίωσης του κώδικα από την σελίδα του NAMD [77].

Κατά τις προσομοιώσεις του δημιουργήθηκαν 4 pipelines για τον υπολογισμό των δυνάμεων σε κάθε Gidel FPGA στα 125 MHz. Το host μηχάνημα που χρησιμοποιήθηκε είναι ένα Dell Precision T5400 workstation με Intel Xeon CPU E5405 (Harpertown) στα 2GHz. Έχει τέσσερις πυρήνες σε τεχνολογία 45 nm και η κάθε μια έχει μία 32 KB L1 I-Cache και μία 32 KB L1 D-Cache. Επίσης έχουν δύο 6 MB L2 caches που τις μοιράζονται ανά δύο οι πυρήνες. Το workstation έχει συνολική μνήμη 2 GB. Το λειτουργικό που χρησιμοποιείται είναι Ubuntu 8.04 (Linux kernel 2.6.24). Επίσης χρησιμοποιήθηκε το Gidel ProcWizard (version 8.9) για να δημιουργηθεί το interface με την πλακέτα και το εργαλείο Altera Quartus II (version 9.1) για το compilation καιτο bit-stream generation [71]. Με βάση τα χαρακτηριστικά αυτά πέτυχαν τα αποτελέσματα που εμφανίζονται στον παρακάτω πίνακα.

	Kernel-only Runtime	Kernel-only Speedup	End-to-end Runtime	End-to-end Speed-up
1 CPU Core	~1437 ms	1x	1957 ms	1x
1 CPU Core + 1 FPGA	129 ms	11.14x	648 ms	3.02x
1 CPU Core + 4 FPGAs	53.25 ms	26.99x	580 ms	3.37x
Theoretical limit	0 ms	infinite	520 ms	3.76x

Πίνακας 4.13: *Speed-up using FPGAs over a single CPU core [71].*

Από τον παραπάνω πίνακα συμπεραίνεται ότι η αύξηση του αριθμού των FPGAs που χρησιμοποιούνται αυξάνει κατά πολύ το kernel-only speedup. Στην παρούσα διπλωματική εργασία θεωρώντας το I/O ότι δεν επηρεάζει την ταχύτητα και εξετάζοντας δηλαδή το Kernel-only Speedup σε σχέση πάντα με το CPU-NAMD (CHAMR++) έχουμε τα αποτελέσματα που εμφανίζονται στον επόμενο πίνακα.

	Kernel-only Runtime	Kernel-only Speedup
1 core	9,682768431	no speedup
2 cores	4,940187975	no speedup
4 cores	2,758760775	no speedup
128 cores (theoritical)	0,086211274	6,18x

Πίνακας 4.14: *Speedup για διαφορετικό αριθμό cores.*

Όπως φαίνεται η διδακτορική εργασία δεν έχει κάποιο κατώφλι ως προς το speedup που μπορεί να επιτύχει και το μόνο που την περιορίζει είναι ο αριθμός των FPGAs. Στην άλλη πλευρά στην παρούσα διπλωματική εργασία το κατώφλι είναι τα 128 cores καθώς μέχρι τόσο μπορεί να

διασπαστεί το εξωτερικό loop του critical kernel. Η σχεδίαση που λαμβάνει υπόψιν και το I/O μπορεί και αυτή προφανώς να διασπαστεί σε 128 cores, παρόλαυτά δεν θα επιτύχει μεγάλη ταχύτητα λόγω του περιορισμού των 12544 κύκλων κατά την αρχή της εκτέλεσης του critical section. Στα αποτελέσματα αυτά φυσικά πρέπει να λαμβάνουμε υπόψιν και την διαφορά στις τεχνολογίες ανάμεσα στις δύο εργασίες.

Κεφάλαιο 5

Σύνοψη και Συμπεράσματα

Στο κεφάλαιο αυτό πραγματοποιείται μία τελική αποτίμηση της διπλωματικής εργασίας. Θα παρουσιαστούν κάποια τελικά συμπεράσματα και έπειτα θα συζητηθούν προτάσεις για μελλοντική δουλειά, στην βελτίωση της σχεδίασης.

5.1 Συμπεράσματα

Στην διπλωματική εργασία αυτή υλοποιήθηκε ένας αλγόριθμος προσομοίωσης Μοριακής Δυναμικής πάνω σε αναδιατασσόμενη λογική. Ο αλγόριθμος ανήκει στην επιστημονική ομάδα του Πανεπιστημίου του Illinois με επικεφαλής τον James Phillips []. Για την επίτευξη του σκοπού αυτού, κατανοήθηκαν έννοιες πάνω στην Μοριακή Δυναμική και Μοριακή Βιολογία, όπως επίσης αποκτήθηκαν γνώσεις πάνω σε νέες τεχνολογίες. Οι τεχνολογίες αυτές, όπως CUDA και CHARM++, είχαν ήδη επιτύχει από μόνες τους μεγάλες ταχύτητες εκτέλεσης των προσομοιώσεων, ιδιαίτερα η CUDA. Η πρόκληση πλέον ήταν να επιτευχθεί μία αντίστοιχη ταχύτητα και με άλλες τεχνολογίες όπως η FPGAs. Ο αλγόριθμος του Πανεπιστημίου του Illinois αποτελεί έναν παραλληλοποιήσιμο αλγόριθμο, κάτι που μετατρέπει τις FPGAs ιδανικές να τον υποστηρίξουν. Στις τελικές σχεδιάσεις παρόλαυτά δεν επιτεύχθηκε η επιτάχυνση του κώδικα πάνω από τις ταχύτητες που επιτυγχάνει η CUDA και το CHARM++, για τις FPGAs που δοκιμάστηκαν.

Οι τελικές σχεδιάσεις αυτές έδειξαν πως οι FPGAs αποτελούν μία σχετικά νέα τεχνολογία η οποία αναπτύσσεται συνεχώς και χρειάζεται ακόμα χρόνος ώστε να φτάσουν τις τεχνολογίες πολύ ειδικού σκοπού όπως τις GPUs. Τα εργαλεία σχεδίασης και τα προβλήματα που προκύπτουν με το I/O δυσκολεύουν πολλές φορές τους σχεδιαστές. Παρόλαυτά η ευελιξία που προσφέρουν θα τις μετατρέψουν σε λίγα χρόνια σε μία από τις κυρίαρχες τεχνολογίες και για ακόμα περισσότερα επιστημονικά προβλήματα, όπως η Μοριακή Δυναμική.

5.2 Προβλήματα που αντιμετωπίστηκαν

Κατά την διπλωματική εργασία αντιμετωπίστηκαν μια σειρά από προβλήματα, από τα πρώτα κιόλας βήματα έως και τα τελευταία. Αρχικά υπήρξε μία δυσκολία στην εγκατάσταση του πακέτου NAMD 2.10 σε έναν οικιακό υπολογιστή, ώστε να ξεκινήσει μία πρώτη εξοικείωση με το πρόγραμμα. Η δυσκολία αυτή οφειλόταν στις ελλείψεις που παρουσίαζε ο οδηγός εγκατάστασης του [77], όπως επίσης και τα λάθη που είχε σε αρκετές περιπτώσεις, καθυστερώντας αρκετά την εκπόνηση της διπλωματικής εργασίας. Για τον λόγο αυτό έχει δημιουργηθεί ένας πλήρως οδηγός εγκατάστασης του πακέτου και εκτέλεσης του είτε με CPU είτε με GPU. Άλλη ένα σημαντικό πρόβλημα που οφείλεται στην ομάδα ανάπτυξης του NAMD,

είναι η έλλειψη documentation του κώδικα. Αυτό δυσκόλεψε αρκετά την εκπόνηση της εργασίας καθώς τις περισσότερες φορές ο κώδικας δεν ήταν κατανοητός. Αυτό το πρόβλημα αντιμετωπίζεται και στην διδακτορική διατριβή του Ashfaquzzaman Khan στο Πανεπιστήμιο της Βοστόνης το 2012 [79].

Κατά το στάδιο μετατροπής της γλώσσας CUDA σε C, επίσης αντιμετωπίστηκαν προβλήματα που οφειλόταν στην έλλειψη γνώσης για την CUDA και αντιμετωπίστηκαν ύστερα από αρκετή ενασχόληση και εξάσκηση πάνω στην γλώσσα. Χρησιμοποιήθηκαν κατά κόρον το βιβλίο των Jason Sanders και Edward Kandrot [80], όπως επίσης και το documentation της NVIDIA για CUDA [76]. Έπειτα στο τελευταίο πλέον στάδιο, δηλαδή την χρήση του εργαλείου Vivado HLS 2015.4, για την μετατροπή του κώδικα σε γλώσσα περιγραφής υλικού, χρησιμοποιήθηκαν τα tutorials UG871 [81]και UG902 [82]. Σε αυτά υπάρχει εκτενής περιγραφή για το εργαλείο και τις δυνατότητες του και περιγράφεται η κάθε λειτουργία του και ιδιαιτερότητα του, ειδικά για την χρήση των Directives και τρόπους προγραμματισμού σε γλώσσα που να γίνεται όσο τον δυνατόν πιο αντιληπτή από το εργαλείο. Σημαντική βοήθεια επίσης υπήρξε και από το community στα forums της NVIDIA [83] και της Xilinx [84], όπως επίσης και από τα μέλη του Microprocessor & Hardware Lab (MHL) του Πολυτεχνείου Κρήτης.

5.3 Μελλοντική Εργασία

Η μελέτη του αλγορίθμου έδειξε πως οι προσομοιώσεις Μοριακής Δυναμικής αποτελούν ένα κατάλληλο πεδίο ενασχόλησης, για την επιτάχυνση τους με FPGAs. Τα προβλήματα που προέκυψαν αποτελούν τροφή για μελλοντική εργασία.

Αρχικά, θα πρέπει ο κώδικας να γίνει πιο ευέλικτος ώστε να μπορούν να χρησιμοποιηθούν περισσότερα παραδείγματα και από άλλες πρωτεΐνες. Στον κώδικα C που δημιουργήθηκε χρησιμοποιήθηκαν κάποιες φορές fixed πίνακες ώστε να γίνεται πιο εύκολα έλεγχος και debugging του κώδικα. Για την επίτευξη αυτού χρειάζεται προφανώς και η υποστήριξη από το Πανεπιστήμιο του Illinois και της επιστημονικής ομάδας ανάπτυξης του NAMD.

Το critical section του κώδικα και με το οποίο αναλώθηκε περισσότερο η διπλωματική, το οποίο υπολογίζει τις long-ranged non-bonded δυνάμεις, είναι γραμμένο σε CUDA κάτι το οποίο περιόριζε αρκετά την μετατροπή του σε C. Η CUDA περιέχει κάποια χαρακτηριστικά τα οποία θα πρέπει να υλοποιηθούν στην C. Στην διπλωματική μεταφέρθηκαν αυτούσια, χωρίς όμως να αποτελεί την βέλτιστη λύση. Αυτό που χρειάζεται είναι μια πολύ βαθύτερη κατανόηση του αλγορίθμου και από πλευράς Μοριακής Δυναμικής ώστε να δημιουργηθεί ένας βέλτιστος κώδικας C. Σε αυτό το εγχείρημα φυσικά χρειάζεται η υποστήριξη της ερευνητικής ομάδας του Πανεπιστημίου του Illinois. Αυτό ίσως περιορίσει τα προβλήματα που υπήρχαν με τους μεγάλους buffers και τις καθυστερήσεις στο I/O, καθώς οι πίνακες αυτοί θα μπορούν πλέον να διαχωρίζονται σε μικρότερους χωρίς να χάνεται πληροφορία.

Η επιτάχυνση των προσομοιώσεων Μοριακής Δυναμικής δεν εξαρτώνται μόνο από τους αλγορίθμους, αλλά και από την τεχνολογία στην οποία εκτελούνται. Οι FPGAs είναι σχετικά νέα τεχνολογία που παρολαυτά χρησιμοποιείται όλο και περισσότερο σε επιστημονικά

προβλήματα. Παρόλαυτα χρειάζονται βελτιώσεις όσον αφορά το I/O, όπως επίσης και τα εργαλεία σχεδιασμού. Με πιο εξελιγμένα εργαλεία θα αντιλαμβάνονται το πρόβλημα καλύτερα και θα δημιουργούνται βέλτιστοι κώδικες περιγραφής υλικού, οι οποίοι θα διαχειρίζονται και τους πόρους και την ταχύτητα που προσφέρει η εκάστοτε FPGA.

Αναφορές

[1] President's Information Technology Advisory Committee (2005). Computational science: Ensuring america's competitiveness. National Coordination Office for Information Technology Research and Development. http://www.nitrd.gov.

[2] Adcock, S. A. and McCammon, J. A. (2006). Molecular dynamics: Survey of methods for simulating the activity of proteins. Chemical Reviews, 106(5):1589-1615. http://dx.doi.org/10.1021/cr040426m.

[3] Brooks, C. and Case, D. A. (1993). Simulations of peptide conformational dynamics and thermodynamics. Chemical Reviews, 93(7):2487-2502. http://dx.doi.org/10.1021/cr00023a008.

[4] DeMarco, M. L. and Daggett, V. (2004). From conversion to aggregation: Protofibril formation of the prion protein. Proceedings of the National Academy of Sciences of the United States of America, 101(8):2293-2298. http://dx.doi.org/10.1073/pnas.0307178101.

[5] Flower, D. R., Phadwal, K., Macdonald, I. K., Coveney, P. V., Davies, M. N., and Wan, S. (2010). T-cell epitope prediction and immune complex simulation using molecular dynamics: state of the art and persisting challenges. Immunome Research, 6(Suppl 2):S4. http://dx.doi.org/10.1186/1745-7580-6-S2-S4.

[6] Kalè, L., Skeel, R., Bhandarkar, M., Brunner, R., Gursoy, A., Krawetz, N., Phillips, J., Shinozaki, A., Varadarajan, K., and Schulten, K. (1999). NAMD2: Greater scalability for parallel molecular dynamics. Journal of Computational Physics, 151:283-312. http://dx.doi.org/10.1006/jcph.1999.6201.

[7] Karplus, M. and McCammon, J. A. (2002). Molecular dynamics simulations of biomolecules. Nature Structural Biology, 9(9):646-652. http://dx.doi.org/10.1038/nsb0902-646.

[8] Khalili-Araghi, F., Tajkhorshid, E., and Schulten, K. (2006). Dynamics of K+ ion conduction through Kv1.2. Biophysical Journal, 91(6):72-76. http://dx.doi.org/10.1529/biophysj.106.091926.

[9] Moraitakis, G., Purkiss, A. G., and Goodfellow, J. M. (2003). Simulated dynamics and biological macromolecules. Reports on Progress in Physics, 66(3):383. http://dx.doi.org/10.1088/0034-4885/66/3/203.

[10] Kumar, S., Huang, C., Zheng, G., Bohm, E., Bhatele, A., Phillips, J. C., Yu, H., and Kalè, L. V. (2008). Scalable molecular dynamics with NAMD on the IBM Blue Gene/L system. IBM Journal of Research and Development, 52(1-2):177-188. http://dx.doi.org/10.1147/rd.521.0177.

[11] Freddolino, P. L., Arkhipov, A. S., Larson, S. B., McPherson, A., and Schulten, K. (2006). Molecular dynamics simulations of the complete satellite tobacco mosaic virus. Structure, 14(3):437-449. http://dx.doi.org/10.1016/j.str.2005.11.014.

[12] Khalili-Araghi, F., Tajkhorshid, E., and Schulten, K. (2006). Dynamics of K+ ion conduction through Kv1.2. Biophysical Journal, 91(6):72-76. http://dx.doi.org/10.1529/biophysj.106.091926.

[13] Eaton, W. A., Muoz, V., Thompson, P. A., Chan, C.-K., and Hofrichter, J. (1997). Submillisecond kinetics of protein folding. Current Opinion in Structural Biology, 7(1):10-14. http://dx.doi.org/10.1016/S0959-440X(97)80003-6.

[14] Hagen, S. J., Hofrichter, J., Szabo, A., and Eaton, W. A. (1996). Diffusion-limited contact formation in unfolded cytochrome c: estimating the maximum rate of protein folding. Proceedings of the National Academy of Science of the United States of America, 93(21):11615-11617. http://dx.doi.org/10.1073/pnas.93.21.11615.

[15] NVIDIA (2012a). Fermi - the next generation CUDA architecture. http://www.nvidia.com/object/fermi_architecture.html.

[16] NVIDIA (2012b). High performance computing. http://www.nvidia.com/object/tesla_computing_solutions.html.

[17] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., and Phillips, J. (2008). GPU computing. Proceedings of the IEEE, 96(5):879-899. http://dx.doi.org/10.1109/JPROC.2008.917757.

[18] NVIDIA (2012c). Tesla bio workbench - enabling new science. http://www.nvidia.com/object/tesla_bio_workbench.html.

[19] GROMACS (2012). GROMACS installation instructions for GPUs. http://www.gromacs.org/Downloads/Installation_Instructions/GPUs.

[20] Stone, J. E., Phillips, J. C., Freddolino, P. L., Hardy, D. J., Trabuco, L. G., and Schulten, K. (2007). Accelerating molecular modeling applications with graphics processors. Journal of Computational Chemistry, 28(16):2618-2640. http://dx.doi.org/10.1002/jcc.20829.

[21] Phillips, J. C., Stone, J. E., and Schulten, K. (2008). Adapting a message driven parallel application to GPU-accelerated clusters. In Proceedings of the ACM/IEEE Conference on Supercomputing (SC), pages 8:1-8:9. http://dx.doi.org/10.1109/SC.2008.5214716.

[22] Allen, M. P. (2004). Introduction to molecular dynamics simulation. Computational Soft Matter: From Synthetic Polymers to Proteins, Lecture Notes, NIC Series, 23:1-28.

[23] Rapaport, D. C. (2004). The art of molecular dynamics simulation. Cambridge University Press, 2nd edition. http://dx.doi.org/10.1017/CBO9780511816581.

[24] Case, D. A., Cheatham, T. E., Darden, T., Gohlke, H., Luo, R., Jr., K. M. M., Onufriev, A., Simmerling, C., Wang, B., and Woods, R. J. (2005). The Amber biomolecular simulation programs. Journal of Computational Chemistry, 26(16):1668-1688. http://dx.doi.org/10.1002/jcc.20290.

[25] Bowers, K. J., Chow, E., Xu, H., Dror, R. O., Eastwood, M. P., Gregersen, B. A., Klepeis, J. L., Kolossvary, I., Moraes, M. A., Sacerdoti, F. D., Salmon, J. K., Shan, Y., and Shaw, D. E. (2006). Scalable algorithms for molecular dynamics simulations on commodity clusters. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC), pages 84:1-84:13. http://dx.doi.org/10.1145/1188455.1188544.

[26] Hess, B., Kutzner, C., van der Spoel, D., and Lindahl, E. (2008). GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. Journal of Chemical Theory and Computation, 4(3):435-447. http://dx.doi.org/10.1021/ct700301q.

[27] LAMMPS (2012b). LAMMPS molecular dynamics simulator. http://lammps.sandia.gov.

[28] Phillips, J. C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R. D., Kal_e, L., and Schulten, K. (2005). Scalable molecular dynamics with NAMD. Journal of Computational Chemistry, 26(16):1781-1802. http://dx.doi.org/10.1002/jcc.20289.

[29] Ponder, J. W. and Case, D. A. (2003). Force fields for protein simulations. Advances in Protein Chemistry, 66:27-85. http://dx.doi.org/10.1016/S0065-3233(03)66002-X.

[30] MacKerell, A. D., Banavali, N., and Foloppe, N. (2000). Development and current status of the CHARMM force _eld for nucleic acids. Biopolymers, 56(4):257-265. http://dx.doi.org/10.1002/1097-0282(2000)56:4%3C257::AID-BIP10029%3E3.0.CO;2-W.

[31] Makov, G. and Payne, M. C. (1995). Periodic boundary conditions in *ab initio* calculations. Physical Review B, 51(7):4014-4022. http://dx.doi.org/10.1103/PhysRevB.51.4014.

[32] Board, J. A. J., Humphres, C. W., Lambert, C. G., Rankin, W. T., and Toukmaji, A. Y. (1997). Ewald and multipole methods for periodic N-body problems. In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (PPSC), pages 1-8.

[33] Darden, T., York, D., and Pedersen, L. (1993). Particle Mesh Ewald: An N.log (N) method for Ewald sums in large systems. Journal of Chemical Physics, 98(12):10089-10092. http://dx.doi.org/10.1063/1.464397.

[34] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. (1995). A smooth particle mesh Ewald method. Journal of Chemical Physics, 103(19):8577{8593. http://dx.doi.org/10.1063/1.470117.

[35] Petersen, H. G. (1995). Accuracy and efficiency of the Particle Mesh Ewald method. Journal of Chemical Physics, 103(9):3668-3679. http://dx.doi.org/10.1063/1.470043.

[36] Skeel, R. D., Tezcan, I., and Hardy, D. J. (2002). Multiple grid methods for classical molecular dynamics. Journal of Computational Chemistry, 23(6):673-684. http://dx.doi.org/10.1002/jcc.10072.

[37] Ewald, P. P. (1921). Die berechnung optischer und elektrostatischer gitterpotentiale. Annalen der Physik, 369(3):253-287. http://dx.doi.org/10.1002/andp.19213690304.

[38] Gu, Y. (2008). FPGA acceleration of molecular dynamics simulations. PhD dissertation, Boston University, USA.

[39] Hockney, R., Goel, S., and Eastwood, J. (1974). Quiet high-resolution computer models of a plasma. Journal of Computational Physics, 14(2):148-158. http://dx.doi.org/10.1016/0021-9991(74)90010-2.

[40] Schofield, P. (1973). Computer simulation studies of the liquid state. Computer Physics Communications, 5(1):17-23. http://dx.doi.org/10.1016/0010-4655(73)90004-0.

[41] Verlet, L. (1967). Computer \Experiments" on classical uids. I. Thermodynamical properties of Lennard-Jones molecules. Physical Review, 159(1):98-103. http://dx.doi.org/10.1103/PhysRev.159.98.

[42] Ebisuzaki, T., Makino, J., Fukushige, T., Taiji, M., Sugimoto, D., Ito, T., and Okumura, S. K. (1993). GRAPE project: an overview. Publications of the Astronomical Society of Japan, 45:269-278.

[43] Fukushige, T., Taiji, M., Makino, J., Ebisuzaki, T., and Sugimoto, D. (1996). A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: MD-GRAPE. Astrophysical Journal, 468:51-61. http://dx.doi.org/10.1086/177668.

[44] Ito, T., Makino, J., Fukushige, T., Ebisuzaki, T., Okumura, S. K., and Sugimoto, D. (1993). A special-purpose computer for N-body simulations: GRAPE-2A. Publications of the Astronomical Society of Japan, 45:339-347.

[45] Komeiji, Y., Uebayasi, M., Takata, R., Shimizu, A., Itsukashi, K., and Taiji, M. (1997). Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer. Journal of Computational Chemistry, 18(12):1546-1563. http://dx.doi.org/10.1002/(SICI)1096-987X(199709)18:12%3C1546::AID-JCC11%3E3.0.CO;2-I.

[46] Narumi, T., Susukita, R., Ebisuzaki, T., Mcniven, G., and Elmegreen, B. (1999). Molecular Dynamics Machine: Special-purpose computer for molecular dynamics simulations. Molecular Simulation, 21:401-415. http://dx.doi.org/10.1080/08927029908022078.

[47] Narumi, T., Susukita, R., Furusawa, H., and Ebisuzaki, T. (2000a). 46 TFLOPS special-purpose computer for molecular dynamics simulations: WINE-2. In 5th International Conference on Signal Processing Proceedings (ICSP), volume 1, pages 575-582. http://dx.doi.org/10.1109/ICOSP.2000.894557.

[48] Narumi, T., Susukita, R., Koishi, T., Yasuoka, K., Furusawa, H., Kawai, A., and Ebisuzaki, T. (2000b). 1.34 TFLOPS molecular dynamics simulation for NaCl with a special-purpose computer: MDM. In ACM/IEEE Conference on Supercomputing (SC), pages 54:1{54:20. http://dx.doi.org/10.1109/SC.2000.10016.

[49] Sumanth, J., Swanson, D., and Jiang, H. (2003). Performance and cost effectiveness of a cluster of workstations and MD-GRAPE 2 for MD simulations. In Second International Symposium on Parallel and Distributed Computing, pages 244-249. http://dx.doi.org/10.1109/ISPDC.2003.1267670.

[50] Dror, R. O., Grossman, J. P., Mackenzie, K. M., Towles, B., Chow, E., Salmon, J. K., Young, C., Bank, J. A., Batson, B., Denero_, M. M., Kuskin, J. S., Larson, R. H., Moraes, M. A., and Shaw, D. E. (2010). Exploiting 162-nanosecond end-to-end communication latency on Anton. In Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 1-12. http://dx.doi.org/10.1109/SC.2010.23.

[51] Shaw, D. E., Denero_, M. M., Dror, R. O., Kuskin, J. S., Larson, R. H., Salmon, J. K., Young, C., Batson, B., Bowers, K. J., Chao, J. C., Eastwood, M. P., Gagliardo, J., Grossman, J. P., Ho, C. R., Ierardi, D. J., Kolossv_ary, I., Klepeis, J. L., Layman, T., McLeavey, C., Moraes, M. A., Mueller, R., Priest, E. C., Shan, Y., Spengler, J., Theobald, M., Towles, B., and Wang, S. C. (2007). Anton, a special-purpose machine for molecular dynamics simulation. In Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), pages 1-12. SIGARCH Computer Architecture News, 35(2):1-12. http://dx.doi.org/10.1145/1250662.1250664.

[52] Bowers, K. J., Lippert, R. A., Dror, R. O., and Shaw, D. E. (2010). Improved twiddle access for Fast Fourier Transforms. Transaction on Signal Processing, 58(3):1122-1130. http://dx.doi.org/10.1109/TSP.2009.2035984.

[53] Shaw, D. E., Dror, R. O., Salmon, J. K., Grossman, J. P., Mackenzie, K. M., Bank, J. A., Young, C., Denero_, M. M., Batson, B., Bowers, K. J., Chow, E., Eastwood, M. P., Ierardi, D. J., Klepeis, J. L., Kuskin, J. S., Larson, R. H., Lindor_-Larsen, K., Maragakis, P., Moraes, M. A.,

Piana, S., Shan, Y., and Towles, B. (2009). Millisecond-scale molecular dynamics simulations on Anton. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC), pages 39:1-39:11. http://dx.doi.org/10.1145/1654059.1654099.

[54] Dror, R., Grossman, J., Mackenzie, K., Towles, B., Chow, E., Salmon, J., Young, C., Bank, J., Batson, B., Denero_, M., Kuskin, J., Larson, R., Moraes, M., and Shaw, D. (2011). Overcoming communication latency barriers in massively parallel scienti_c computation. IEEE Micro, 31(3):8-19. http://dx.doi.org/10.1109/MM.2011.38.

[55] Young, C., Bank, J. A., Dror, R. O., Grossman, J. P., Salmon, J. K., and Shaw, D. E. (2009). A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC), pages 23:1{23:11. http://dx.doi.org/10.1145/1654059.1654083.

[56] Nelson, M. T., Humphrey, W., Gursoy, A., Dalke, A., Kal_e, L. V., Skeel, R. D., and Schulten, K. (1996). NAMD: a parallel, object-oriented molecular dynamics program. International Journal of High Performance Computing Applications, 10(4):251-268. http://dx.doi.org/10.1177/109434209601000401.

[57] Mei, C., Sun, Y., Zheng, G., Bohm, E. J., Kal_e, L. V., Phillips, J. C., and Harrison, C. (2011). Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime. In Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 61:1-61:11. http://dx.doi.org/10.1145/2063384.2063466.

[58] Wolinski, C., Trouw, F., and Gokhale, M. (2003). A preliminary study of molecular dynamics on recon_gurable computers. In International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pages 304-307.

[59] Azizi, N., Kuon, I., Egier, A., Darabiha, A., and Chow, P. (2004). Reconfigurable molecular dynamics simulator. In The 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 197-206. http://dx.doi.org/10.1109/FCCM.2004.48.

[60] Kuon, I., Azizi, N., Darabiha, A., Egier, A., and Chow, P. (2004). FPGA-based supercomputing: an implementation for molecular dynamics. In ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA), pages 253-253. http://dx.doi.org/10.1145/968280.968340.

[61] Gu, Y., Vancourt, T., and Herbordt, M. C. (2005). Accelerating molecular dynamics simulations with con_gurable circuits. In International Conference on Field Programmable Logic and Applications (FPL), pages 475-480. http://dx.doi.org/10.1109/FPL.2005.1515767.

[62] Gu, Y., Vancourt, T., and Herbordt, M. C. (2006a). Accelerating molecular dynamics simulations with con_gurable circuits. IEE Proceedings – Computers and Digital Techniques, 153(3):189-195. http://dx.doi.org/10.1049/ip-cdt:20050182.

[63] Gu, Y., Vancourt, T., and Herbordt, M. C. (2006b). Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In International Conference on Field Programmable Logic and Applications (FPL), pages 1-8. http://dx.doi.org/10.1109/FPL.2006.311190.

[64] Gu, Y., Vancourt, T., and Herbordt, M. C. (2006c). Integrating FPGA acceleration into the Protomol molecular dynamics code: Preliminary report. In IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 315-316. http://dx.doi.org/10.1109/FCCM.2006.52.

[65] Gu, Y., Vancourt, T., and Herbordt, M. C. (2008). Explicit design of FPGAbased coprocessors for short-range force computations in molecular dynamics simulations. Parallel Computing, 34(4-5):261-277. http://dx.doi.org/10.1016/j.parco.2008.01.007.

[66] Annapolis (2003). WILDSTAR-II Hardware Reference Manual. Annapolis Micro Systems Inc., USA.

[67] Annapolis (2006). WILDSTAR II PRO for PCI. Annapolis Micro Systems Inc., USA.

[68] Baxter, R., Booth, S., Bull, M., Cawood, G., Perry, J., Parsons, M., Simpson, A., Trew, A., McCormick, A., Smart, G., Smart, R., Cantle, A., Chamberlain, R., and Genest, G. (2007). Maxwell - a 64 FPGA supercomputer. In Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pages 287-294. http://dx.doi.org/10.1109/AHS.2007.71.

[69] Kasap, S. and Benkrid, K. (2011). A high performance implementation for molecular dynamics simulations on a FPGA supercomputer. In 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pages 375-382. http://dx.doi.org/10.1109/AHS.2011.5963962.

[70] Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics, 117(1):1-19. http://dx.doi.org/10.1006/jcph.1995.1039.

[71] MD. Ashfaquzzaman Khan (2012). Scalable Molecular Dynamics Simulation using FPGAs and Multicore processors. https://www.bu.edu/caadlab/Khan_dissertation.pdf

[72] FPGA Wikipedia website. https://en.wikipedia.org/wiki/Field-programmable_gate_array

[73] Altera (2012). Altera website. http://www.altera.com.

[74] Xilinx (2012). Xilinx website. http://www.xilinx.com.

[75] Xilinx 7 Series Documentation UG479. http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[76] CUDA Programming Documentation. http://docs.nvidia.com/cuda/cuda-c-programming-guide

[77] NAMD website. http://www.ks.uiuc.edu/Research/namd/

[78] Severin, P. M. D., Zou, X., Gaub, H. E., and Schulten, K. (2011). Cytosine methylation alters DNA mechanical properties. Nucleic Acids Research, 39(20):8740-8751. http://dx.doi.org/10.1093/nar/gkr578.

[79] MD. Ashfaquzzaman Khan (2012). Scalable Molecular Dynamics Simulation using FPGAs and Multicore processors. Chapter 6. https://www.bu.edu/caadlab/Khan_dissertation.pdf

[80] Jason Sanders, Edward Kandrot. Cuda By Example. An introduction to General-Purpose GPU Programming. http://www.physics.drexel.edu/~valliere/PHYS405/GPU_Story/CUDA_by_Example_Addison_Wesley_Jul_2010.pdf

[81] Vivado Design Suite Tutorial: High-Level Synthesis. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug871-vivado-high-level-synthesis-tutorial.pdf
[82] Vivado Design Suite User Guide. High-Level Synthesis. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf
[83] Xilinx Community Forums. https://forums.xilinx.com/
[84] NVIDIA Community Forums. https://devtalk.nvidia.com/

NAMD License

James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. Journal of Computational Chemistry, 26:1781-1802, 2005.