Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Παραλληλοποίηση της τοποθέτησης FPGA με βάση τον Simulated Annealing.** **Parallelizing FPGA Placement based on Simulated Annealing.**
Ονοματεπώνυμο Φοιτητή	**Γεώργιος Αθανασόπουλος**
Πατρώνυμο	**Δημήτριος**
Αριθμός Μητρώου	**ΜΠΣΠ/ 12002**
Επιβλέπων	**Μιχάλης Ψαράκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2016**

Περιεχόμενα

ABSTRACT

Οι συσκευές Field-Programmable Gate Arrays (FPGAs) επιτρέπουν την γρήγορη και φτηνή υλοποίηση λογικών κυκλωμάτων και αποτελούν μια ευρέως αποδεκτή μέθοδο σχεδιασμού κυκλωμάτων εδώ και αρκετές δεκαετίες. Η ραγδαία αύξηση της κλίμακας ολοκλήρωσης παρέχει την δυνατότητα να αυξηθεί η λογική που μπορεί να προγραμματιστεί σε μια FPGA, γεγονός που αυξάνει τις απαιτήσεις για εύρεση αλγορίθμων που θα είναι ταχύτεροι από αυτούς του παρελθόντος. Οι ερευνητικές προσπάθειες εστιάζουν στο στάδιο της τοποθέτησης (placement), το οποίο παίρνει έως και 50% του υπολογιστικού χρόνου με σκοπό να διατηρηθεί το πλεονέκτημα που παρουσιάζει αυτή η τεχνολογία να μπορεί να προγραμματιστεί γρήγορα και άμεσα. Τα περισσότερα εργαλεία CAD που εκτελούν την διαδικασία της τοποθέτησης στις FPGA είναι βασισμένα στην σειριακή εκτέλεση του αλγορίθμου Προσομοιωμένης Ανόπτησης (Simulated Annealing). Ο χρόνος εκτέλεσης του συγκεκριμένου αλγορίθμου αυξάνει εκθετικά όσο αυξάνεται η λογική στο κύκλωμα ενώ με τον ρυθμό αύξησης της εξομοιούμενης λογικής, αν δεν βρεθεί κάποια εναλλακτική θα καταστεί ασύμφορος. Γι αυτό, η έρευνα για εύρεση κατάλληλης λύσης που θα εκμεταλλεύεται την παράλληλη επεξεργασία παραμένει κρίσιμή.

Δυστυχώς οι περισσότερες προτάσεις που έχουν γίνει πάνω στην παράλληλη επεξεργασία, παρουσιάζουν μικρές τιμές επιτάχυνσης και η αποτελεσματικότητά τους εξαρτάται σε μεγάλο βαθμό από τμήματα τα οποία συνεχίζουν να εκτελούνται σειριακά. Σε αυτό το έγγραφο προτείνεται μια μέθοδος η οποία μπορεί να εκτελεστεί σε οποιοδήποτε πολυπύρηνο σύστημα αν και είναι αποδοτικότερη σε SIMD συστήματα ενώ παρέχονται παραδείγματα με χρήση CUDA C και της βιβλιοθήκης της NVidia για C++, Thrust. Στόχος της εργασίας είναι να εκμεταλλευτεί την δυνατότητα να εκτελείται ανά πάσα στιγμή ένας μεγάλος αριθμός νημάτων (threads) και των παράλληλων σχημάτων προγραμματισμού χωρίς να θυσιάζει μέρος της ποιότητας του τελικού κυκλώματος.

Field Programmable Gate Arrays (FPGAs) allow fast and cheap implementation of logic circuits and they consist a widely used method for design of electronic circuits for the past few decades. The growing market demands that more logic can potentially be packed into them especially now that the integration scale becomes smaller every few years. This unavoidably lead to the pursuit of a better programming method for the FPGAs as it is evident that the more the logic that is packed into them the more computational time is required in order to compile the hardware description program. The research trend at the moment is searching for a new algorithm that can potentially speed the process of placement of the logic blocks on the FPGA since that part of the program flow can take up to 50% of the total time spent in compilation. Most of the CAD programs responsible for the compilation process, implement a serial version of Simulated Annealing algorithm. The execution time of this algorithm increases exponentially the more logic is required to be implemented on the FPGA which threatens to take away some of the advantages of using an FPGA over an ASIC in the future. Fortunately the evolution of computational systems appears to be systems that rely on multiple processors that allow large throughput and that can work independently. Thus it is crucial that research focuses on parallel processing.

Unfortunately, most papers posted on this subject to day, show either underwhelming speedups or sacrifice quality for speed resulting in subpar implementations. This thesis proposes a method of Simulated Annealing parallelization that can be applied to any multicore/manycore system albeit optimized for SIMD systems. Examples are given that use NVidia's GPUs to serve as the means to increase the throughput using CUDA C language extension and the C++ library Thrust. Our goal is to propose an alternative version of the Simulated Annealing algorithm that takes advantage of parallel programming concepts without sacrificing much of the final product quality if any at all.

1. Εισαγωγή

Περίπου πριν δυόμιση δεκαετίες, οι συσκευές FPGA (Field Programmable Gate Arrays) εμφανίστηκαν σαν ιδέα και λόγω ορισμένων χαρακτηριστικών τους έγιναν ευρέως αποδεκτές ως μέσα υλοποίησης μέσων και μικρότερων κυκλωμάτων. Η περαιτέρω βελτίωση των κυκλωμάτων και η αύξηση της κλίμακας ολοκλήρωσης οδήγησε στην δραματική αύξηση της χρήσης τους για την επίλυση διαφόρων προβλημάτων, καθώς όλο και περισσότερη λογική προστίθεται σε αυτές μέρα με την ημέρα επιτρέποντας την ικανοποιητική εξομοίωση όλο και μεγαλύτερων και πολυπλοκότερων υλοποιήσεων.

Οι FPGAs παρουσιάζουν ορισμένα χαρακτηριστικά που τις καθιστά συχνά καλύτερες λύσεις σε σύγκριση με ένα chip ASIC (Application Specific Integrated Circuit). Καταρχάς, η παραγωγή ενός ASIC είναι μια διαδικασία πολυέξοδη για κάθε αρχιτεκτονική και για οποιοδήποτε σφάλμα παρουσιαστεί το κόστος αυτό αυξάνεται ενώ οι FPGAs μπορούν στις περισσότερες μορφές τους να επαναπρογραμματιστούν όσες φορές απαιτείται χωρίς καμία απολύτως επιβάρυνση. Επίσης η παραγωγή ASIC παίρνει 6-8 εβδομάδες σε αντίθεση με τις FPGA που σε μερικά λεπτά μπορούμε να τις προγραμματίσουμε γεγονός το οποίο μπορεί να αποτελέσει καθοριστικό παράγοντα στην επιλογή του ενός μέσου έναντι του άλλου καθώς όχι μόνο εξοικονομείται χρόνος κατά την ολοκλήρωση του κυκλώματος αλλά λαμβάνοντας υπ' όψιν το γεγονός ότι ο χρόνος ζωής ενός προϊόντος στην αγορά είναι περιορισμένος, αυτός ο επιπρόσθετος χρόνος μπορεί να βλάψει σημαντικά το κέρδος που μπορεί να αποφέρει ένα προϊόν.

Φυσικά οι FPGAs έχουν και μειονεκτήματα. Το βασικότερο μειονέκτημά τους είναι ότι λόγω του ότι δεν χρησιμοποιούν το βέλτιστο μήκος αγωγών και του ότι κάνουν προσομοίωση του κυκλώματος μέσω χρήσης κελιών μνήμης είναι κατά μέσο όρο έως και τρεις φορές πιο αργές [50] από ένα ASIC που περιλαμβάνει την ίδια λογική. Επίσης, στην αρχιτεκτονική SRAM την οποία θα αναλύσουμε αργότερα, οι μικροδιακόπτες που χρησιμοποιούνται για την δρομολόγηση αυξάνουν την καθυστέρηση της διέλευσης του ηλεκτρικού φορτίου καθώς ανεβάζουν την αντίσταση γεγονός που καθυστερεί περαιτέρω το κύκλωμα.

Επιπλέον, ένα ακόμη μειονέκτημά τους είναι ότι απαιτούν πολύ περισσότερο χώρο για να υλοποιήσουν την λογική του κυκλώματος καθώς το αποτέλεσμα δεν προκύπτει από την μοναδική μέθοδο διέλευσης του ηλεκτρικού φορτίου μέσα από το κύκλωμα αλλά από την εξομοίωση του αποτελέσματος της λογικής συνάρτησης με χρήση LUT(Look-Up Tables) τα οποία σε συνδυασμό με τις μεθόδους δρομολόγησης (routing methods) απαιτούν πολύ περισσότερο χώρο(από 20 έως 35 φορές μεγαλύτερο [50]) από ότι μερικές πύλες, γεγονός που με την σειρά του συνεπάγεται μεγαλύτερη κατανάλωση ενέργειας (7-14 φορές περισσότερή ενέργεια [50]) προς συντήρηση των μνημών και λειτουργίας του κυκλώματος.

Ο προγραμματισμός των FPGAs απαιτεί αρκετό χρόνο και αν δεν βελτιωθεί σύντομα, με την ραγδαία αύξηση της κλίμακας ολοκλήρωσης και την προσθήκη ολοένα και περισσότερων λογικών στοιχείων σε αυτές, κινδυνεύει να καταστεί ασύμφορος καθώς η ανάγκη για προσθήκη περισσότερης λογικής είναι μεγάλη. Το μεγαλύτερο τμήμα του υπολογιστικού χρόνου της διαδικασίας του προγραμματισμού τους απαιτείται για το στάδιο της τοποθέτησης των μπλοκ λογικής στην πλακέτα. Βελτιώνοντας αυτό το στάδιο επιτυγχάνουμε τις μεγαλύτερες επιταχύνσεις και σε αυτό εστιάζει η παρούσα μελέτη και πιο συγκεκριμένα στην επιτάχυνση εργαλείων που χρησιμοποιούν τον αλγόριθμο Simulated Annealing.

1.1 Συμβολή

Η συμβολή της παρούσας διατριβής στο θέμα της επιτάχυνσης του αλγορίθμου Simulated Annealing παρουσιάζεται σε δυο τομείς:

Πρόταση για παραγωγή ανεξάρτητων μεταξύ τους σετ κινήσεων: Ένα βασικό πρόβλημα κατά την παράλληλη εκτέλεση είναι η εξασφάλιση ότι οι εντολές που θα εκτελεστούν σε κάθε νήμα (thread) δεν θα κάνουν χρήση κοινών πόρων με άλλα νήματα όσο είναι δυνατόν. Στην περίπτωση μας, η παραγωγή ανεξάρτητων σετ κινήσεων επιτρέπει επιτάχυνση σχεδόν ίση με τον αριθμό των νημάτων που εκτελούνται ανά πάσα χρονική στιγμή για να αξιολογήσουν κάποια κίνηση το καθένα. Όταν όλα τα σετ είναι ανεξάρτητα είναι δυνατόν να εκτελείται μια κίνηση σε κάθε νήμα. Η πρότασή μας σε αυτόν τον τομέα παρέχει αυτή την εξασφάλιση ενώ διατηρεί μεγαλύτερο βαθμό τυχαιότητας από τις

υπόλοιπες προτάσεις που έχουν γίνει κατά καιρούς. Επίσης επιτρέπει την καλύτερη μετακίνηση λογικών μπλοκ από μία περιοχή σε κάποια άλλη βοηθώντας περαιτέρω στην αποφυγή απώλειας ποιότητας τελικής υλοποίησης.

Πλήρες πρόγραμμα Annealing (Annealing Schedule): Το Annealing Schedule τροποποιείται κατάλληλα ώστε να είναι αποδοτικό σε μια υλοποίηση παράλληλης επεξεργασίας. Παρατίθενται οι δομές που θεωρούμε ότι θα διευκολύνουν την διαδικασία και χρησιμοποιούμε για κριτήριο κόστους των αλγόριθμο Star [51] . Με χρήση σχημάτων υπολογισμού καταλλήλων για παράλληλη επεξεργασία λειτουργιών, υπολογίζουμε όλες τις απαραίτητες παραμέτρους με ελάχιστη ως καθόλου ανάγκη συγχρονισμού δεδομένων.

1.2 Οργάνωση Εργασίας

Η εργασία αυτή είναι δομημένη με τέτοιο τρόπο ώστε ακόμη και αν ο αναγνώστης έχει περιορισμένη γνώση τόσο στον τομέα FPGA όσο και στην παράλληλη επεξεργασία, να είναι δυνατόν να κατανοήσει την πρόταση που γίνεται αλλά και να γίνουν εμφανή τα πλεονεκτήματά της. Παρέχονται πληροφορίες τόσο για την αρχιτεκτονική μιας FPGA όσο και αρχών του παράλληλου προγραμματισμού.

Πιο συγκεκριμένα, στο κεφάλαιο 2 αναλύονται οι αρχιτεκτονικές των FPGA και ομαδοποιούνται βάση ενός συνόλου κριτηρίων. Με αυτόν τον τρόπο παρέχεται το θεωρητικό υπόβαθρο που απαιτείται για να κατανοηθεί αργότερα ο τρόπος με τον οποίο μοντελοποιείται η τοποθέτηση λογικών μπλοκ στην FPGA και να γίνει αξιολόγησή της. Στην συνέχεια εξηγείται ο ορισμός ενός βασικού λογικού στοιχείου (Basic Logic Element) και γίνεται ο διαχωρισμός του από το λογικό μπλοκ που αποτελεί το στοιχείο μονάδα που χρησιμοποιείται κατά το στάδιο της τοποθέτησης. Σε αυτό το κεφάλαιο επίσης αναφέρεται η ροή λειτουργίας ενός CAD (Computer Aided Design) προγράμματος και εμβαθύνουμε στο στάδιο της τοποθέτησης (placement) και της δρομολόγησης (routing) δίνοντας τις απαραίτητες πληροφορίες για να γίνουν αντιληπτοί οι περιορισμοί που προκύπτουν αργότερα σε επόμενο κεφάλαιο.

Μετά από το θεωρητικό υπόβαθρο, στο κεφάλαιο 3 εξετάζεται ο τρόπος με τον οποίο το πρόγραμμα πάνω στο οποίο στηρίχτηκε η εργασία αυτή [T-VPack 6.0] αντιμετωπίζει το πρόβλημα της τοποθέτησης των λογικών μπλοκ. Το συγκεκριμένο πρόγραμμα δανείζεται στοιχεία από προγενέστερες μελέτες αλλά έχει και κάποιες σημαντικές διαφορές οι οποίες το οδήγησαν στο να γίνει το ακαδημαϊκό πρότυπο. Τέλος αναφέρονται τρείς βασικές κατηγορίες τρόπων αντιμετώπισης του προβλήματος της τοποθέτησης στον παράλληλο προγραμματισμό.

Στο κεφάλαιο 4, η εργασία ασχολείται με τον παράλληλο προγραμματισμό. Αρχικά εστιάζει στο υλικό(hardware) το οποίο εκτελεί παράλληλα τις διεργασίες και πιο συγκεκριμένα την αρχιτεκτονική του. Κατανοώντας τις διαφορές μεταξύ πολυπύρηνων επεξεργαστών και καρτών γραφικών εμβαθύνουμε στις κάρτες γραφικών καθώς αυτές αποτελούν το μέσο που στοχεύει αυτή η μελέτη. Αναλύονται οι περιορισμοί που προκύπτουν κατά τον προγραμματισμό τους και επεξηγούνται συγκεκριμένα προγραμματιστικά μοτίβα που θα χρησιμοποιηθούν αργότερα για αποτελεσματική αξιολόγηση των πόρων που παρέχονται από τις GPU.

Στην συνέχεια στο κεφάλαιο 5 παρουσιάζεται η πρότασή μας σχετικά με την παραγωγή ανεξάρτητων κινήσεων καθώς επίσης και το κατάλληλο πεδίο εφαρμογής τους ώστε να μεγιστοποιηθεί τόσο η τυχαιότητα του συστήματος όσο και η μετακίνηση των στοιχείων πάνω στην FPGA. Στην συνέχεια αναφέρεται μια έκδοση του αλγορίθμου (Simulated Annealing) που προτείνεται ενώ παραθέτονται και συγκεκριμένες δομές δεδομένων που εκμεταλλεύονται τα προγραμματιστικά μοτίβα που αναφέρθηκαν στο προηγούμενο κεφάλαιο και οι οποίες κατά την άποψή μας θα βελτιώσουν τον χρόνο εκτέλεσης μιας τέτοιας υλοποίησης.

Τελειώνοντας, στο κεφάλαιο 6 συνοψίζουμε τα συμπεράσματα που προκύπτουν από την παραπάνω μελέτη και αναφέρονται κατευθύνσεις στις οποίες θα μπορούσαν μελλοντικές προσπάθειες να εστιάσουν ώστε να επεκτείνουν ή/και βελτιώσουν την εργασία αυτή.

2. Θεωρητικό υπόβαθρο

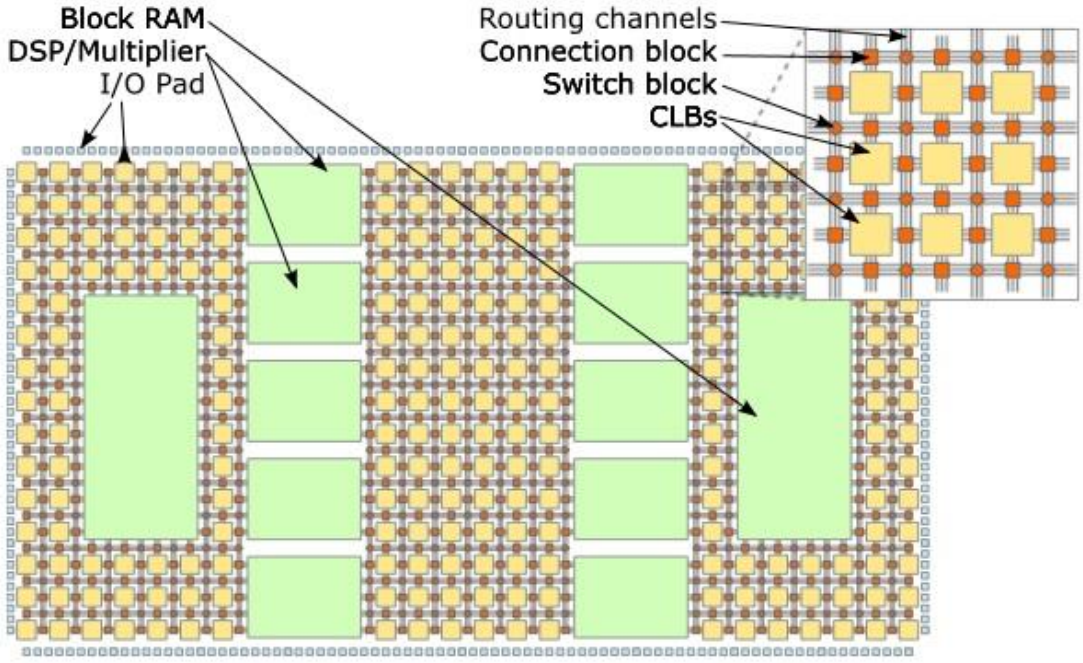
Για να κατανοηθεί η εργασία αυτή απαιτείται ένα θεωρητικό υπόβαθρο όσον αφορά τόσο την αρχιτεκτονική των FPGA όσο και της ροής λειτουργίας των προγραμμάτων που καθιστούν δυνατό τον προγραμματισμό τους.

2.1 Αρχιτεκτονικές FPGA

Μια συσκευή FPGA αποτελείται από τρία βασικά συστατικά:

- Προγραμματιζόμενα λογικά μπλοκ στα οποία υλοποιούνται οι λογικές συναρτήσεις. Τέτοια μπλοκ είναι συνήθως ένας συνδυασμός μπλοκ μνήμης με Flip-Flop.
- Μπλοκ Εισόδου/Εξόδου(I/O) τα οποία συνδέονται με τα λογικά μπλοκ και τα οποία αποτελούν τα σημεία επικοινωνίας του FPGA chip με οτιδήποτε βρίσκεται εκτός chip π.χ. LED SCREEN, διακόπτες, πλήκτρα κ.α.
- Προγραμματιζόμενο σύστημα δρομολόγησης (routing) το οποίο συνδέει τα λογικά μπλοκ μεταξύ τους καθώς επίσης και τα λογικά μπλοκ με μπλοκ εισόδου/εξόδου

Γενικά, μια FPGA είναι ένα σύνολο λογικών μπλοκ (Configurable Logic Blocks - CLBs) οργανωμένα σε ένα δισδιάστατο πλέγμα. Τα Μπλοκ I/O βρίσκονται στην περιφέρεια και μεταξύ των CLBs και των I/O υπάρχουν αγωγοί που προγραμματίζονται ώστε να συνδέσουν τα απαραίτητα τμήματα μεταξύ τους. Το ακόλουθο σχήμα παρουσιάζει μια τέτοια οργάνωση.



Εικ 2.1 Τυπική Αρχιτεκτονική FPGA.

Ανάλογα με τις ανάγκες, έχουν χρησιμοποιηθεί κατά καιρούς διαφορετικές αρχιτεκτονικές με πλεονεκτήματα και μειονεκτήματα η μια έναντι της άλλης.

2.1.1 Αρχιτεκτονικές FPGA με βάση την τεχνολογία προγραμματισμού

2.1.1.1 FPGAs Βασισμένες σε Static-Ram(SRAM) - SRAM Based FPGAs.

Η πιο διαδεδομένη αρχιτεκτονική είναι αυτή που στηρίζεται σε Static Ram Cells τα οποία αποθηκεύουν την λογική υπό την μορφή Look-Up Tables (LUT) και τα οποία είναι διαμοιρασμένα σε όλη την έκταση του chip. Τα LUTs συνδέονται μεταξύ τους μέσω αγωγών με την βοήθεια Multiplexers τα οποία καθορίζουν ποιοί αγωγοί χρησιμοποιούνται στην εκάστοτε υλοποίηση. Επειδή η συγκεκριμένη αρχιτεκτονική χρησιμοποιεί SRAM η οποία απαιτεί ενέργεια για να αποθηκεύσει τα δεδομένα και να τα διατηρήσει, είναι λογικό ότι κάθε φορά που η FPGA σταματάει να λειτουργεί, κατά την επανεκκίνηση χρειάζεται εκ νέου παραμετροποίηση.Σε αυτήν την αρχιτεκτονική υπάρχουν δυο επιλογές κατά τον προγραμματισμό.

Master mode: Η FPGA διαβάζει τα δεδομένα παραμετροποίησης από κάποια εξωτερική πηγή.(π.χ. ένα εξωτερικό FLASH memory chip). Σε ορισμένες περιπτώσεις αυτό το chip μπορεί να αντικατασταθεί από ενσωματωμένη FLASH μνήμη on-chip.

Slave mode: Σε αυτή την επιλογή η FPGA δέχεται τα δεδομένα παραμετροποίησης από κάποια άλλη συσκευή π.χ. JTAG ή μέσω κάποιας άλλης μεθόδου επικοινωνίας.

Το σημαντικότερο πλεονέκτημα αυτής της αρχιτεκτονικής είναι ότι παρέχει την δυνατότητα απεριόριστων σχεδόν επαναπρογραμματισμών. Το βασικότερο μειονέκτημά FPGAs αυτής της αρχιτεκτονικής είναι ότι λόγω του ότι χρησιμοποιούν volatile μνήμη SRAM απαιτείται ο επαναπρογραμματισμός τους κάθε φορά που βγαίνουν εκτός λειτουργίας καθώς επίσης και η συνεχής τροφοδότησή τους με ηλεκτρικό ρεύμα για να διατηρηθούν τα δεδομένα καθιστώντας τες , πιο ενεργοβόρες.

2.1.1.2 FPGAs Βασισμένες σε Flash Μνήμη - Flash Based FPGAs.

Μια εναλλακτική επιλογή είναι η χρήση non volatile μνήμης . Οι Flash-Based FPGAs χρησιμοποιούν FLASH ή EEPROM μνήμη. Αυτή η κατηγορία έχει το πλεονέκτημα ότι δεν χρειάζεται τροφοδότηση με ρεύμα για να διατηρήσει τα στοιχεία του κυκλώματος προς εξομοίωση όποτε δεν χρειάζεται επαναπρογραμματισμό κάθε φορά που τίθεται σε λειτουργία. Αυτό συνεπάγεται επίσης και μικρότερη κατανάλωση ενέργειας. Επίσης αυτού του είδους οι FPGAs είναι πιο συμπυκνωμένες κατά συνέπεια χρειάζονται λιγότερο χώρο για την υλοποίηση του ίδιου κυκλώματος. Το μειονέκτημα αυτής της αρχιτεκτονικής είναι ότι ο αριθμός των φορών που αυτού του τύπου οι FPGAs μπορούν να προγραμματιστούν δεν είναι πολύ μεγάλος.

2.1.1.3 FPGAs Βασισμένες σε Antifuse Τεχνολογία.

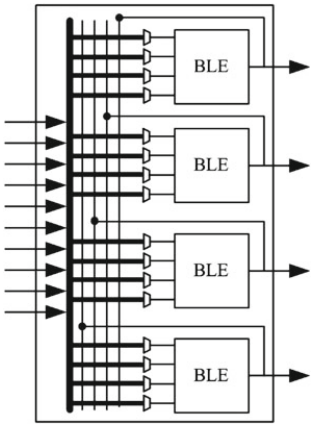
Οι Antifuse-Based FPGAs χρησιμοποιούν ασφάλειες οι οποίες καίγονται κατά τον προγραμματισμό τους και έτσι δημιουργούν το τελικό κύκλωμα. Τα πλεονεκτήματα της συγκεκριμένης μεθόδου είναι ότι έχουν καλύτερη ταχύτητα από τις άλλες δύο κατηγορίες, είναι πιο αποδοτικές από πλευράς χώρου που απαιτείται για την υλοποίηση τους και δεν στηρίζονται σε volatile τύπους μνήμης. Το μειονέκτημα αυτής της αρχιτεκτονικής είναι ότι μπορεί να προγραμματιστεί μόνο μία φορά καθιστώντας τες την λιγότερο προτιμώμενη επιλογή.

2.1.2 CLBs και αρχιτεκτονικές με βάση τα Logic Blocks

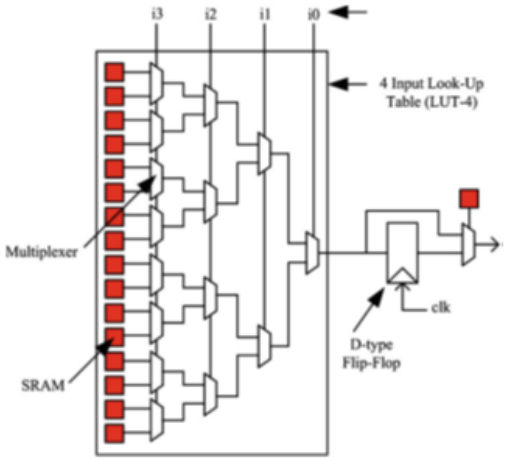
Ένα *CLB (Configurable Logic Block)* είναι το είναι το θεμελιώδες στοιχείο της FPGA που παρέχει τόσο την λογική όσο και την αποθηκευτική ικανότητα που απαιτείται για την υλοποίηση ενός κυκλώματος. Ένα CLB μπορεί να αποτελείται απλά από ένα transistor έως και ολόκληρο processor. Αν το βασικό κομμάτι λογικής της FPGA είναι ένα transistor απλά τότε θα απαιτείται μεγαλύτερη έκταση για να υλοποιηθεί οποιοδήποτε κύκλωμα και περισσότεροι αγωγοί και κατά συνέπεια το κύκλωμα γίνεται πολύ πιο αργό και ενεργοβόρο . Αν είναι ένας processor τότε ακόμη και για την πιο απλή λογική συνάρτηση θα υπάρχει κατασπατάληση πόρων συστήματος. Παρά το γεγονός ότι κατά καιρούς έχουν προταθεί και χρησιμοποιηθεί διάφορα CLBs αυτή η μελέτη επικεντρώνεται σε αυτά τα οποία βασίζονται σε *Look-up Tables (LUTs)*.

Ένα LUT είναι στην πραγματικότητα μια διάταξη memory cells και multiplexers. Τα κελιά μνήμης δίνουν σταθερό σήμα και ανάλογα με τις παραμέτρους που δίνονται στους multiplexers μόνο συγκεκριμένα σήματα επιτρέπεται να περάσουν κατά συνέπεια μπορεί με την βοήθειά τους να περιγραφεί στην μνήμη οποιαδήποτε λογική συνάρτηση. Ό αριθμός των κελιών μνήμης και κατά συνέπεια και των multiplexers παίζει καθοριστικό ρόλο στην ποιότητα του εξομοιωθέντος κυκλώματος. Ένα LUT-k έχει k εισόδους, μπορεί να περιγράψει οποιαδήποτε λογική συνάρτηση k εισόδων και έχει 2^k bits παραμετροποίησης κατά συνέπεια καταλαμβάνει ισάριθμα bits SRAM. Ως Βασικό Λογικό Στοιχείο (BLE - Basic Logic Element) ονομάζουμε το μικρότερο στοιχείο που αποτελείται από ένα LUT-k και ένα Flip-Flop.

Το κάθε CLB μπορεί να αποτελείται από ένα ή περισσότερα BLEs τα οποία είναι συνδεδεμένα μεταξύ τους με ξεχωριστό εσωτερικό δίκτυο δρομολόγησης που κάνει το κάθε BLE προσβάσιμο στα υπόλοιπα . Οι πιο διαδεδομένες αρχιτεκτονικές είναι αυτές που το CLB περιλαμβάνουν τέσσερα BLEs αν και υπάρχουν ορισμένες που έχουν έως και δέκα.



Εικ 2.3 CLB που αποτελείται από 4 BLEs.



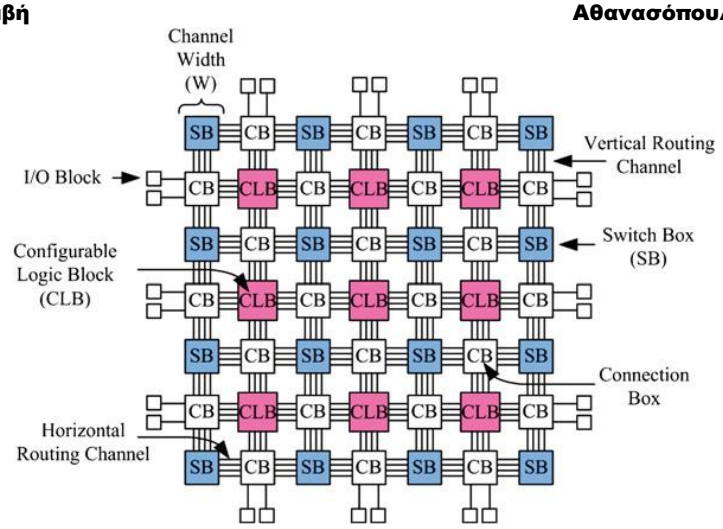
Εικ. 2.2 Ένα απλό Basic Logic Element με χρήση LUT-4

2.1.3 Αρχιτεκτονικές με βάση τον τρόπο δρομολόγησης(routing)

Όπως προαναφέρθηκε, σε μία FPGA τα προγραμματιζόμενα CLBs συνδέονται μεταξύ τους και με τα στοιχεία εισόδου/εξόδου I/Os μέσω ενός δικτύου αγωγών που διασταυρώνονται σε ορισμένα σημεία τα οποία έχουν προγραμματιζόμενους διακόπτες, οι οποίοι φροντίζουν να γίνει η δρομολόγηση. Εφόσον οι FPGAs πρέπει να δύνανται να υλοποιήσουν οποιοδήποτε κύκλωμα πρέπει το σύστημα δρομολόγησης να διαθέτει το απαραίτητο υλικό για να συνδέσει τόσο τα CLBs που είναι κοντά μεταξύ τους όσο και αυτά που βρίσκονται σε μεγάλη απόσταση. Παρόλο που οι περισσότερες συνδέσεις μεταξύ CLBs είναι συνήθως τοπικές ώστε να υπάρχει μικρότερη ενεργειακή κατανάλωση και μεγαλύτερη ταχύτητα λειτουργίας του κυκλώματος, υπάρχουν και αγωγοί μεγάλου μήκους που σκοπό έχουν την διασύνδεση μεταξύ πολύ απομακρυσμένων στοιχείων όπως π.χ. πιθανόν ένα CLB με ένα I/O. Για αυτόν τον λόγο έχουν χρησιμοποιηθεί δύο στρατηγικές που οδήγησαν σε δύο κατηγορίες αρχιτεκτονικών με βάση τον τρόπο δρομολόγησης.

2.1.3.1 Island-Style Routing Architecture

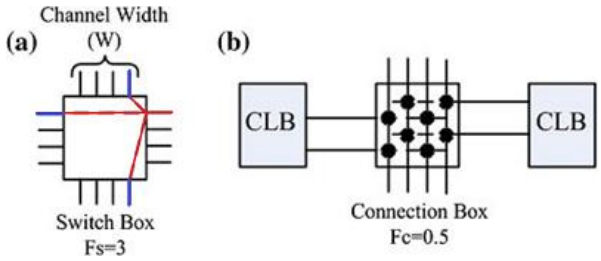
Ο πιο συνηθισμένος τύπος FPGA, τόσο για ακαδημαϊκή όσο και εμπορική χρήση και ο τύπος στον οποίο επικεντρώνεται αυτή η μελέτη, είναι ο Island-Style ή Mesh-Based. Το όνομά του προέρχεται από τον τρόπο που είναι δομημένα τα στοιχεία της FPGA. Τα CLBs είναι τοποθετημένα απέχοντας συγκεκριμένη απόσταση μεταξύ τους ενώ περιτριγυρίζονται από αγωγούς του δικτύου δρομολόγησης που τα συνδέει τόσο μεταξύ τους όσο και με τα στοιχεία εισόδου εξόδου που υπάρχουν στην περίμετρο του chip. Έτσι σχηματίζεται μια μορφή πλέγματος (Mesh-Based) ενώ τα CLBs μοιάζουν με νησιά περιτριγυρισμένα από αγωγούς όπως φαίνεται στην εικόνα 2.4.



Εικ. 2.4 Mesh-Based FPGA

Σε αυτή την αρχιτεκτονική το δίκτυο δρομολόγησης καταλαμβάνει το 80%-90% της έκτασης του chip και όπως βλέπουμε αποτελείται από τρία συστατικά.

- Οριζόντιους και κάθετους αγωγούς διαφορετικού μήκους οι οποίοι ενώνονται μεταξύ τους και παρέχουν την δυνατότητα σύνδεσης μεταξύ οποιονδήποτε δύο στοιχείων.
- Switch Boxes τα οποία περιλαμβάνουν προγραμματιζόμενους διακόπτες που ενώνουν τους αγωγούς με άλλους παρέχοντας την ευελιξία που απαιτείται για να δρομολογηθεί σχεδόν οποιοδήποτε κύκλωμα.
- Connection Boxes τα οποία συνδέουν τις ακίδες εισόδου και εξόδου των CLBs με το δίκτυο δρομολόγησης.



Εικ. 2.5 a) Switch Box με Fs=3, **b)** Connection Box με Fc=0.5

Όπως είναι εμφανές, το ποιά κυκλώματα μπορούν να υλοποιηθούν σε μια τέτοια αρχιτεκτονική εξαρτάται από ορισμένες παραμέτρους.

Μια από αυτές είναι το εύρος του καναλιού σε εκάστοτε σημείο του chip. Αυτό μπορεί να διαφέρει από σημείο σε σημείο . Οι κατασκευαστές FPGA έχουν φροντίσει, εφόσον οι περισσότερες συνδέσεις μεταξύ CLBs όταν γίνεται σωστή τοποθέτηση στοιχείων είναι κοντινές, να έχουν μικρού μήκους αγωγούς που να συνδέονται μέσω των switch boxes περισσότερους από ότι μεγαλύτερους. Φυσικά για τις περιπτώσεις που πρέπει να συνδεθεί κάποιο στοιχείο με κάποιο άλλο που βρίσκονται μακριά μεταξύ τους υπάρχουν αγωγοί οι οποίοι είναι μεγάλοι και δεν περνάνε από όλα τα switch boxes, γεγονός που τους καθιστά γρηγορότερη δίοδο για τέτοιες απομακρυσμένες συνδέσεις και το οποίο επιτρέπει μεγαλύτερη ταχύτητα στο τελικό κύκλωμα και μικρότερές απαιτήσεις σε χώρο. Όμως , αυτού του τύπου οι αγωγοί είναι δυνατόν να είναι ο λόγος που κάποιο κύκλωμα δεν είναι δρομολογήσιμο σε αυτήν την αρχιτεκτονική καθώς με αυτόν τον τρόπο μειώνεται η ευελιξία διασύνδεσης.

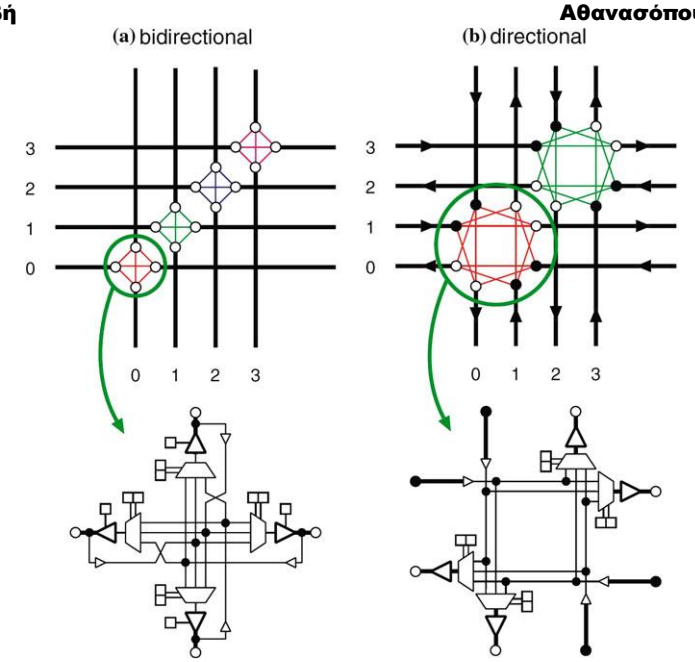
Ένας ακόμη παράγοντας που επηρεάζει την ποιότητα του κυκλώματος καθώς επίσης και την δυνατότητα δρομολόγησης ενός κυκλώματος είναι η δυνατότητα σύνδεσης κάθε αγωγού που εισέρχεται σε ένα Switch Box με άλλους αγωγούς που συνδέονται με αυτό. Η ευελιξία σύνδεσης αυτή συμβολίζεται με Fs και ισούται με τον απόλυτο αριθμό με τους οποίους μπορεί να συνδεθεί μέσω του συγκεκριμένου κουτιού κάθε αγωγός που εισέρχεται σε αυτό. π.χ. στην εικόνα 2.5a το Fs του κόκκινου αγωγού που εισέρχεται στο Switch Box από δεξιά ισούται με 3 καθώς μπορεί να συνδεθεί με τους τρεις μπλε αγωγούς (πάνω, κάτω και αριστερά). Επίσης, τα Switch Boxes έχουν δυο πιθανές μορφές. bi-directional και directional όπως φαίνεται στην εικόνα 2.6. Τα directional Switch Boxes είναι πιο γρήγορα και απαιτούν λιγότερο χώρο για την υλοποίηση κυκλωμάτων αλλά έχουν το μειονέκτημα ότι το εύρος του καναλιού πρέπει να είναι ζυγό και ότι το ηλεκτρικό ρεύμα περνάει μέσα από αυτά με συγκεκριμένη κατεύθυνση ενώ τα bi-directional χρησιμοποιούν tri-state buffers για να ορίσουν το προς τα πού θα κινηθεί το ρεύμα.

Η δυνατότητα μίας ακίδας του CLB να συνδεθεί με αγωγούς ορίζει το μέγεθος Fc το οποίο ισούται με τον αριθμό των αγωγών με τους οποίους δύναται να συνδεθεί η ακίδα προς το εύρος του καναλιού (συνολικός αριθμός αγωγών που διέρχεται από το συγκεκριμένο connection box). Έτσι στην εικόνα 2.5 b βλέπουμε ότι και τα δύο pins του κάθε CLB συνδέονται μόνο με δύο από τους τέσσερεις αγωγούς άρα Fc= 2/4 = 0.5 για το κάθε pin και εφόσον είναι οι ίδιοι αγωγοί και για τα δύο pins το συνολικό Fc είναι πάλι 0.5.

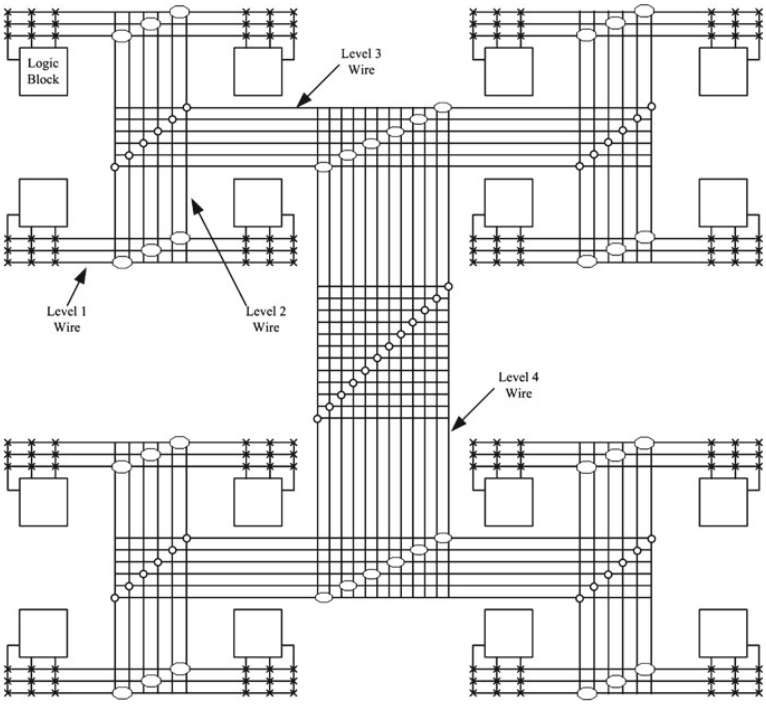
2.1.3.2 Αρχιτεκτονικές Ιεραρχικής Δρομολόγησης - Hierarchical Routing Architecture

Σε αυτήν την κατηγορία η διαφορά στην αρχιτεκτονική είναι ότι είναι πιο εύκολο να συνδεθούν μεταξύ τους δύο λογικά μπλοκ και η πολυπλοκότητα σύνδεσης αυξάνει όσο περισσότερα λογικά μπλοκ είναι συνδεδεμένα μεταξύ τους. Η μορφολογία τους είναι σαν δέντρο. Τα BLEs σχηματίζουν clusters για να παραχθεί ένα CLB και μετά με την ίδια διαδικασία σχηματίζεται cluster από clusters ιεραρχικά. Αρχικά, η σύνδεση μεταξύ BLEs γίνεται απευθείας με αγωγούς χωρίς την ύπαρξη διακοπτών στο ενδιάμεσο και δημιουργείται ένα cluster. Στην συνέχεια με την χρήση ενός switch box αυτά συνδέονται με άλλα ίδια clusters ενώ όλα μαζί πλέον σχηματίζουν ένα μεγαλύτερο cluster και ούτω καθεξής μέχρι που φτάνουμε στον υψηλότερο βαθμό ιεραρχίας στον οποίο υπέρ-clusters είναι ενωμένα μεταξύ τους.

Τέτοιου τύπου FPGAs είναι δυνατόν να εξασφαλίσουν την δρομολόγηση οποιουδήποτε κυκλώματος, όμως το μεγαλύτερο πρόβλημά τους είναι ότι απαιτούν πολύ χώρο για την υλοποίηση τους και κατά συνέπεια όσο αυξάνεται η λογική υπάρχει μεγαλύτερη πολυπλοκότητα και έτσι απώλεια ταχύτητας και κατανάλωση ενέργειας.



Εικ. 2.6 a) bidirectional καλωδίωση και το κύκλωμα που την υλοποιεί με tri-state buffers
b)(uni)directional καλωδίωση με συγκεκριμένες κατευθύνσεις ηλεκτρικού φορτίου



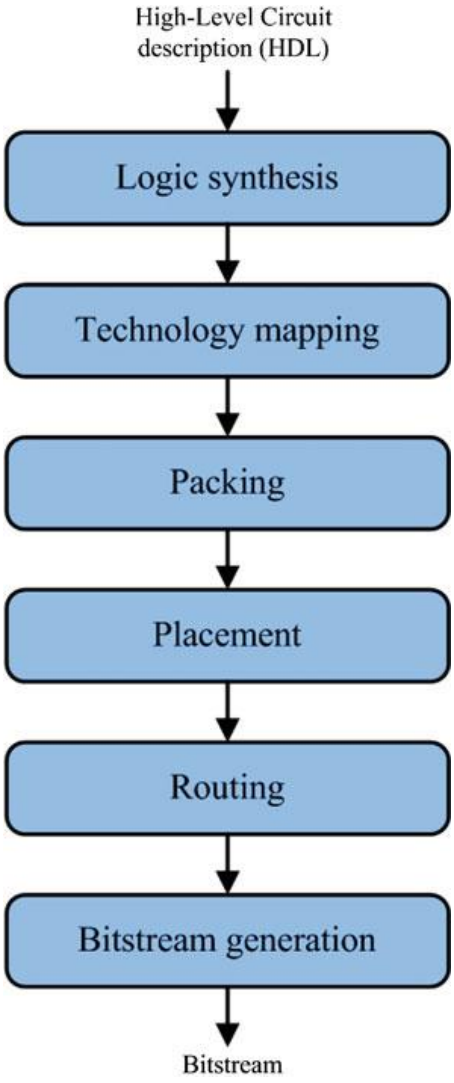
Εικ 2.7 Παράδειγμα Ιεραρχικής Αρχιτεκτονικής FPGA (Hierarchical Routing FPGA)

2.2 Εργαλεία CAD και ροή λειτουργίας τους.

Οι παράμετροι που πρέπει να οριστούν για την υλοποίηση ενός κυκλώματος σε FPGA είναι πάρα πολλές, πράγμα που καθιστά αυτή την διαδικασία αδύνατον να γίνει σε λογικό χρόνο από άνθρωπο. Γι αυτό, υπάρχουν τα CAD(Computer Aided Design) εργαλεία. Τα εργαλεία αυτά παίζουν πολύ σημαντικό ρόλο στην απόδοση του τελικού κυκλώματος καθώς ακόμη και μια τεχνολογικά ανώτερη από πλευράς σχεδιασμού και υλικού FPGA δύναται να χάσει τα πλεονεκτήματά της αν το CAD δεν μπορεί να εκμεταλλευτεί τα ειδικά αυτά χαρακτηριστικά που την κάνουν πιο αποδοτική από κάποια άλλη. Η ανάγκη για βελτίωση οδήγησε στην συνεχή έρευνα για εύρεση πιο αποδοτικών

αλγορίθμων των εργαλείων αυτών, ώστε η εκάστοτε υλοποίηση να απέχει όσο το δυνατόν λιγότερο από ένα αντίστοιχο ASIC chip.

Η ροή εκτέλεσης ενός τέτοιου προγράμματος αρχίζει συνήθως με την είσοδο κάποιου αρχείου γλώσσας περιγραφής υλικού (π.χ. Verilog, VHDL) και ολοκληρώνεται όταν η λογική του κυκλώματος προς υλοποίηση έχει μετατραπεί σε μια σειρά bits τα οποία προγραμματίζουν όλα τα προγραμματιζόμενα μέρη της FPGA. Η παραπάνω διαδικασία χωρίζεται στα ακόλουθα στάδια:



Εικ 2.8 Στάδια ροής λειτουργίας ενός CAD.

Η μελέτη αυτή επικεντρώνεται στην μελέτη του σταδίου της τοποθέτησης(Placement) αλλά θα γίνει μια μικρή αναφορά και στα υπόλοιπα στάδια.

2.2.1 Σύνθεση Λογικής - Logic Synthesis

Η υλοποίηση οποιουδήποτε κυκλώματος ξεκινάει με το στάδιο της Σύνθεσης Λογικής - Logic Synthesis. Σε αυτό το στάδιο, ένα αρχείο περιγραφικής γλώσσας (VHDL,Verilog) μετατρέπεται από register-transfer level περιγραφή σε ένα ιεραρχικό δίκτυο Boolean πυλών και Flip-Flops τα οποία αναπαριστούν την λογική του κυκλώματος προς υλοποίηση. Στην συνέχεια εφαρμόζονται τεχνικές βελτιστοποίησης που είναι άσχετες με την αρχιτεκτονική ή τις δυνατότητες της στοχευόμενης FPGA , οι οποίες απλοποιούν την λογική που προκύπτει , μειώνοντας όσο το δυνατόν περισσότερο τα στοιχεία τα οποία απαιτούνται για την υλοποίησή της σε κύκλωμα κατά συνέπεια και τον χώρο τον οποίο το κύκλωμα θα καταλαμβάνει στη συνέχεια.

2.2.2 Technology Mapping

Η έξοδος του ανωτέρω σταδίου είναι ένα δίκτυο λογικών πυλών, Flip-Flops και καλωδίωσης μεταξύ τους. Αυτό μπορεί να αναπαρασταθεί επίσης και με έναν κατευθυνόμενο άκυκλο γράφο (Directed Acyclic Graph- DAG). Κάθε κόμβος αντιστοιχεί σε μια λογική πύλη, Flip-Flop,είσοδο ή έξοδο του αναπαριστώμενου κυκλώματος και κάθε ακμή σε σύνδεση μεταξύ δύο τέτοιων στοιχείων. Ο στόχος αυτού του σταδίου είναι η εύρεση του καταλληλότερου συνδυασμού στοιχείων με βάση το πως έχουν περιγραφεί σε κάποια βιβλιοθήκη του προγράμματος, ο οποίος να μπορεί να περιγράψει επακριβώς την λογική του ανωτέρω γραφήματος. Η βιβλιοθήκη αυτή συνήθως περιλαμβάνει k-input LUTs και Flip-Flops. Οι αλγόριθμοι που χρησιμοποιούνται σε αυτό το στάδιο βελτιστοποιούν την υλοποίηση ως προς διάφορους παράγοντες όπως χώρο που καταλαμβάνεται και ενέργεια που καταναλώνεται από το προκύπτων κύκλωμα.

Το τελικό αποτέλεσμα αυτού του σταδίου είναι ένα δίκτυο από k-LUTs και Flip-Flops.

2.2.3 Clustering and Packing

Σε αυτό το στάδιο , μέσα από μια διαδικασία δύο επιπέδων, ορίζονται τα τελικά CLBs. Στο πρώτο επίπεδο, τα k-LUTs και τα Flip-Flops, συγχωνεύονται σε k-LogicBlocks (k-LBs) ενώ στο δεύτερο στάδιο τα k-LBs ομαδοποιούνται περαιτέρω ώστε να σχηματιστούν τα CLBs, πάντα σε συμφωνία με τους περιορισμούς που υπάρχουν λόγω των κατασκευαστικών προδιαγραφών της FPGA στην οποία πρόκειται να τοποθετηθούν.

Σε αυτό το στάδιο υπάρχουν τρείς τύποι αντιμετώπισης του προβλήματος.

- Top-down: Στις top down μεθόδους, το δίκτυο διαιρείται διαδοχικά ή τα Logic Blocks μετακινούνται επαναληπτικά με απώτερο σκοπό η δημιουργία clusters [3] [4][5][7][10]. Αυτή η μέθοδος παρέχει τα καλύτερα αποτελέσματα από τις τρείς.
- Depth-optimal: Αυτή η προσέγγιση έχει ως στόχο την μείωση των καθυστερήσεων ακόμη και αν αυτό προϋποθέτει επανάληψη λογικών τμημάτων [8][9].
- Bottom-up: Αποτελεί την συνήθως προτιμώμενη τακτική αντιμετώπισης καθώς παρέχει καλύτερους χρόνους εκτέλεσης και αρκετά καλές τιμές καθυστερήσεων κυκλώματος [1][2][6]. Επίσης , λόγω του τρόπου με τον οποίο λειτουργεί η συγκεκριμένη μέθοδος είναι ευκολότερη η ικανοποίηση περιορισμών όπως ο αριθμός ακίδων(pins) του κυκλώματος. Αρχικά επιλέγεται τυχαία κάποιο k-LUT και μετά με αυτό ως αρχικό συστατικό (seed) προστίθενται σε αυτό άλλα συστατικά με κριτήριο το να έχουν όσο μεγαλύτερο αριθμό κοινών ακίδων. Αυτή η μέθοδος χρησιμοποιείται και από το εργαλείο T-VPACK το οποίο έχει επιλεγεί για την υλοποίηση αυτής της εργασίας.

2.2.4 Τοποθέτηση - Placement

Το στάδιο αυτό της ροής λειτουργίας είναι το στάδιο κατά το οποίο ορίζεται η θέση την οποία θα καταλαμβάνει κάθε λογικό μπλοκ προς υλοποίηση πάνω στην FPGA. Σε ορισμένες περιπτώσεις, είναι απαραίτητο να διαμοιραστούν με τέτοιο τρόπο ώστε λαμβάνοντας υπ' όψιν το εύρος τον καναλιών να μην συνωστίζονται αργότερα (routability-driven placement) .Ο πιο συνήθης στόχος είναι να βρεθούν τα μπλοκ όσο το δυνατόν πλησιέστερα μεταξύ τους ώστε να ελαχιστοποιηθεί το μήκος των αγωγών που χρειάζεται κατά το στάδιο της δρομολόγησης - routing (wire-length driven placement) ή/και οι καθυστερήσεις στο κύκλωμα (timing-driven placement).

Υπάρχουν τεσσάρων ειδών placers:

- Placers βασισμένοι σε τεμαχισμό min-cut (partitioning-based)[50][51]
- Αναλυτικοί placers ακολουθούμενοι συνήθως από διαδικασία τοπικών βελτιώσεων[53-61]
- βασισμένα σε γενετικούς αλγορίθμους[64][65][66]
- Placers βασισμένοι στον αλγόριθμο προσομοιωμένης ανόπτησης(Simulated Annealing)

Οι min-cut(partitioning) μέθοδοι διαιρούν επαναληπτικά τον χώρο και μετακινούν ή ανταλλάζουν θέση μεταξύ των CLBs με κριτήριο την ελαχιστοποίηση τον τομών nets που γίνεται στα όρια του κάθε υπόχωρου. Αυτό έχει σαν αποτέλεσμα τα υψηλής συνδεσιμότητας CLBs να τείνουν να βρίσκονται στον ίδιο υπόχωρο. Το πλεονέκτημα αυτής της κατηγορίας placer είναι η ταχύτητα εκτέλεσης ωστόσο το γεγονός ότι δεν εστιάζουν καθόλου στην ελαχιστοποίηση του μήκους αγωγών ή/και στην βελτίωση του κρίσιμου μονοπατιού κάνουν πολύ συχνά την ποιότητα των κυκλωμάτων που προκύπτουν, να μην είναι ιδανική.

Στους αναλυτικούς(analytic) αλγορίθμους, γίνεται επίλυση ενός συστήματος εξισώσεων ώστε να βρεθούν θέσεις για το κάθε λογικό μπλοκ. Όμως επειδή οι πόροι που μπορούν να χρησιμοποιηθούν δεν είναι διαθέσιμοι το ίδιο σε όλα τα σημεία της FPGA πλακέτας, ακολουθεί μια διαδικασία εύρεσης κάποιας κατάλληλης λύσης με χρήση επαναληπτικού διαχωρισμού του χώρου τοποθέτησης. Στην συνέχεια, τα μπλοκ τα οποία έχουν τις ίδιες τιμές κατά την επίλυση των εξισώσεων διαμοιράζονται κατάλληλα στους υπόχωρους ώστε να υπάρχει τελικώς μόνο ένα μπλοκ το πολύ σε κάθε τμήμα. Υπάρχουν δύο κατηγορίες analytic αλγορίθμων. Οι force repelling αλγόριθμοι χρησιμοποιούν την συνδεσιμότητα μεταξύ των λογικών μπλοκ για να δημιουργήσουν απωθητικές δυνάμεις μεταξύ τους ώστε να αποφευχθεί η περίπτωση σύμπτωσης δύο μπλοκ στο χώρο, ενώ οι quadratic programming χρησιμοποιούν ένα γράφημα της λίστας πλεγμάτων (netlist) και προσπαθούν να ελαχιστοποιήσουν το τετραγωνισμένο συνολικό μήκος των αγωγών που απαιτείται για την υλοποίηση του κυκλώματος. Το πλεονέκτημα των analytic μεθόδων είναι ότι έχουν πολύ μικρότερους χρόνους εκτέλεσης σε σχέση με τους αλγορίθμους Simulated Annealing αλλά συχνά χάνουν σε ποιότητα αποτελέσματος.

Στο παρελθόν έγινε απόπειρα χρήσης γενετικών αλγορίθμων για την επίλυση του προβλήματος. Παρόλο που έχει γίνει κάποια εργασία σε αυτό το κομμάτι, η συγκεκριμένη κατηγορία δεν έχει το ίδιο καλή ποιότητα τελικού κυκλώματος με το VPR που στηρίζεται σε Simulated Annealing για τον ίδιο χρόνο εκτέλεσης ανεξαρτήτως αλγορίθμου.

Ο Simulated Annealing αλγόριθμος μιμείται εικονικά την διαδικασία annealing στην μεταλλουργία κατά την οποία μεταλλικές πλάκες θερμαίνονται και μετά με διαδοχικές κινήσεις τους δίνεται το σχήμα που είναι επιθυμητό κάθε φορά. Στην αρχικά πολύ υψηλή θερμοκρασία , η μεταλλική πλάκα μπορεί να παραμορφωθεί σε οποιοδήποτε σχήμα αλλά με την πάροδο του χρόνου μετά από διαδοχικές κινήσεις, η θερμοκρασία πέφτει σταδιακά κατά συνέπεια οι παραμορφώσεις που μπορούν να γίνουν είναι πιο μικρές και τοπικές [11]. Αντίστοιχα, στον Simulated Annealing αλγόριθμο, ορίζεται μια αρχική τιμή θερμοκρασίας ή οποία μειώνεται σταδιακά μετά από ένα συγκεκριμένο αριθμό κινήσεων και η οποία χρησιμοποιείται για να ρυθμιστεί ο βαθμός δυσκολίας αποδοχής μιας κίνησης.

Η εργασία αυτή επικεντρώνεται σε μεθόδους τοποθέτησης που στηρίζονται στον αλγόριθμο Simulated Annealing καθώς είναι σχετικά γρήγορες στην εύρεση καλής λύσης, συνήθως ξεπερνάνε σε ποιότητα αποτελέσματος τις άλλες μεθόδους στον ίδιο χρόνο εκτέλεσης, είναι ευέλικτες όπως θα εξηγηθεί παρακάτω και τυγχάνει η μέθοδος που υλοποιείται στο T-VPack 6.0 όπου είναι το εργαλείο που χρησιμοποιείται κατά κύριο λόγο από την ακαδημαϊκή κοινότητα σαν στάνταρ και σαν μέτρο σύγκρισης, να ανήκει σε αυτή την κατηγορία.

Αλγόριθμος Προσομοιωμένης Ανόπτησης - Simulated Annealing

Αρχικά, επιλέγεται μια τυχαία απεικόνιση των στοιχείων του κυκλώματος και ορίζεται μια αρχική θερμοκρασία η οποία είναι αρκετά μεγάλη. Για την αναπαράσταση του κυκλώματος χρησιμοποιείται συνήθως ένας πίνακας όπου έχει είτε κενά κελιά, είτε κελιά τα οποία περιέχουν τις πληροφορίες που απαιτούνται για να περιγραφεί ένα CLB ή ένα I/O block. Στην συνέχεια ορίζεται το μέγεθος R_{lim} και αρχικοποιείται. Το R_{lim} εξαρτάται από την τρέχουσα θερμοκρασία και παίζει ρόλο περιοριστικό στις κινήσεις που θα μπορούν να εξετασθούν.

Ο αλγόριθμος συνεχίζει κάνοντας έναν έλεγχο κατά πόσο ικανοποιείται ένα κριτήριο τερματισμού του αλγορίθμου και αν δεν έχει ικανοποιηθεί, τότε το σύστημα δοκιμάζει σειριακά να κάνει έναν συγκεκριμένο αριθμό κινήσεων ανά θερμοκρασία πριν την ενημερώσει σε κάποια νέα τιμή. Μια κίνηση αντιστοιχεί στην μετακίνηση ενός CLB ή I/O μπλοκ σε κάποια άλλη θέση εντός των ορίων του R_{lim}. Επιλέγεται τυχαία ένα λογικό μπλόκ και επίσης τυχαία επιλέγεται μια νέα θέση στην οποία πρόκειται να μετακινηθεί. Αν στην νέα θέση υπάρχει ήδη κάποιο άλλο λογικό μπλοκ τότε τα δύο αυτά ανταλλάζουν θέσεις. Στην συνέχεια υπολογίζονται τα νέα κόστη ώστε να αξιολογηθεί αν αυτή η κίνηση οδηγεί σε αποδοτικότερη κατάσταση με βάση κάποιο προκαθορισμένο κριτήριο.

Αν το κόστος της νέας κατάστασης είναι μικρότερο από αυτό της προηγούμενης τότε η νέα κίνηση γίνεται πάντα αποδεκτή. Αν είναι μεγαλύτερο , τότε και πάλι είναι πιθανόν να γίνει αποδεκτή η κίνηση παρόλο που θα οδηγήσει εκείνη την στιγμή σε χειρότερο placement. Αυτή η αποδοχή μη βέλτιστων κινήσεων ορισμένες φορές επιτρέπει την έξοδο από τα τοπικά ελάχιστα της συνάρτησης

κόστους κατά συνέπεια να επιτυγχάνεται συχνά καλύτερο τελικό αποτέλεσμα. Η πιθανότητα μια κίνηση να γίνει αποδεκτή προκύπτει από την σχέση $e^{-\Delta C/T}$ όπου ΔC είναι η μεταβολή του συνολικού κόστους και T η τρέχουσα θερμοκρασία. Αρχικά η θερμοκρασία είναι αρκετά υψηλή και όλες σχεδόν οι κινήσεις γίνονται αποδεκτές, όμως με την πάροδο του χρόνου η θερμοκρασία μειώνεται περιορίζοντας τον αριθμό των κινήσεων που θα γίνουν σε κάθε βήμα αλλά και την πιθανότητα να γίνουν αποδεκτές κινήσεις που έχουν μεγαλύτερο συνολικό κόστος από την προηγούμενη κατάσταση του κυκλώματος.

Αν κάποια κίνηση γίνει αποδεκτή τότε η καινούρια κατάσταση γίνεται η τρέχουσα κατάσταση ενώ αν δεν προκύψει έγκυρη κίνηση τότε η κατάσταση παραμένει η ίδια. Στην συνέχεια, αφού ενημερωθεί η τρέχουσα θερμοκρασία με βάση κάποια συνάρτηση μεταβολής και αφού υπολογιστεί το νέο R_{lim} , ο αλγόριθμος εκτελείται ξανά μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.

Σε μια τέτοια υλοποίηση, ο ρυθμός με τον οποίο μειώνεται η θερμοκρασία, το κριτήριο εξόδου το οποίο όταν πληρείται , τερματίζεται ο αλγόριθμος, ο αριθμός των κινήσεων που εκτελούνται ανά θερμοκρασία και η μέθοδος με την οποία προκύπτουν οι νέες κινήσεις , αποτελούν το πρόγραμμα annealing (annealing schedule). Με το κατάλληλο annealing schedule έχουμε πολύ καλά αποτελέσματα σε λογικό υπολογιστικό χρόνο. Κατά καιρούς έχουν προταθεί συγκεκριμένα annealing schedules τα οποία ενώ λειτουργούν πολύ αποδοτικά για συγκεκριμένα προβλήματα που καλούνταν να επιλύσουν, εν τέλει δεν ήταν το ίδιο αποδοτικά για άλλα προβλήματα π.χ. διαφορετικό benchmark ή διαφορετική αρχιτεκτονική. Επειδή το εργαλείο που χρησιμοποιείται σε αυτήν εδώ την μελέτη είναι ακαδημαϊκός placer, είναι υποχρεωτικό το annealing schedule να είναι ευέλικτο αρκετά ώστε να μπορεί να προσαρμοστεί και να είναι αποδοτικό σε πολλές διαφορετικές τεχνολογίες και κυκλώματα προς υλοποίηση.

Οι Huang, Romeo, Sangiovanni-Vincentelli [13] υλοποίησαν ένα annealing schedule στο οποίο η αρχική θερμοκρασία (InitialT) ορίζεται ίση με 20σ , όπου σ είναι η τυπική απόκλιση του κόστους ενός συγκεκριμένου αριθμού κινήσεων επί της αρχικής τοποθέτησης. Η νέα θερμοκρασία σε κάθε επανάληψη υπολογίζεται ως εξής:

$$T_{new} = T_{old}\, e^{-\lambda T_{old}/\sigma}$$

όπου λ είναι μια παράμετρος ελέγχου που συνήθως ορίζεται στην τιμή 0.7 και σ είναι η τυπική απόκλιση του αριθμού των κινήσεων που έγιναν δεκτές στην θερμοκρασία T_{old} . Τέλος, ο αλγόριθμος τερματίζει όταν η διαφορά του μεγίστου κόστους που έγινε αποδεκτό και του ελαχίστου είναι ίση με την μέγιστη αλλαγή κόστους που γίνεται από οποιαδήποτε κίνηση.

Σε μια άλλη προσέγγιση, οι Lam και Delosme [14] χρησιμοποίησαν το μέσο κόστος, την τυπική απόκλιση του κόστους και το ποσοστό των προτεινόμενων κινήσεων που έγιναν αποδεκτές στις τ πιο πρόσφατες κινήσεις (με τ συνήθως 100). Σε αυτό το annealing schedule η θερμοκρασία υπολογίζεται μετά από κάθε κίνηση. Η διαδικασία τερματίζει όταν στις τελευταίες $k\tau$ κινήσεις δεν υπήρξε καμία αλλαγή στο μέσο κόστος. Περαιτέρω αυτή η προσέγγιση ήταν η πρώτη που όρισε την παράμετρο R_{lim} η οποία σκοπό έχει να περιορίσει σε πιο τοπικό επίπεδο τις κινήσεις που είναι αποδεκτές. Αρχικά το R_{lim} είναι μεγάλο αρκετά ώστε να επιτρέπει σχεδόν οποιαδήποτε κίνηση πάνω στην πλακέτα. Κατά την διάρκεια της εκτέλεσης, το R_{lim} προσαρμόζεται ανάλογα με το ποσοστό των κινήσεων που γίνονται αποδεκτές κατά τέτοιο τρόπο ώστε περίπου 0.44 των κινήσεων να γίνονται αποδεκτές. Δυστυχώς η πολυπλοκότητα του αλγορίθμου τον καθιστά λιγότερο προτιμώμενο.

Επιπρόσθετα, καθώς οι δυνατότητες δρομολόγησης σε μια FPGA είναι πεπερασμένες , κατά καιρούς έχουν προταθεί annealing schedules τα οποία λαμβάνουν υπ' όψιν τέτοιους περιορισμούς. Στην δημοσίευση τους οι Ebeling et al [15] προτείνουν πέρα από το βασικό κριτήριο του περικλείοντος πλαισίου (bounding box) το οποίο θα αναλυθεί σε επόμενο κεφάλαιο, να χρησιμοποιείται και ένας άλλος παράγοντας που να υποδηλώνει πόσο συνωστισμένη είναι κάθε περιοχή.

Οι Nag και Rutenbar [16][17] προσπάθησαν να ενσωματώσουν το στάδιο της δρομολόγησης (routing) στο στάδιο τοποθέτησης. Σε αυτήν την προσέγγιση, κάθε αλλαγή στο κύκλωμα απαιτεί εκ νέου εύρεση μονοπατιού. Όσο πιο υψηλή η θερμοκρασία, τόσο λιγότερες προσπάθειες γίνονται και λιγότερες διαδρομές ελέγχονται. Όσο η θερμοκρασία ελαττώνεται τόσος περισσότερος χρόνος αφιερώνεται στην εύρεση έγκυρου μονοπατιού προς δρομολόγηση του τελικού κυκλώματος. Αν δεν βρεθεί μονοπάτι μετά από κάποια μεταβολή τότε το κόστος της κίνησης αυξάνεται υπερβολικά, καθιστώντας την κίνηση αδύνατο να ολοκληρωθεί. Η ποιότητα των υλοποιήσεων με αυτή την μέθοδο είναι άριστη αλλά ο χρόνος εκτέλεσης αυξάνεται ραγδαία με την αύξηση της λογικής προς υλοποίηση.

Η πολυπλοκότητα του συγκεκριμένου αλγορίθμου υπολογίζεται σε $O(n^3)$, όπου n ο αριθμός CLBs στο κύκλωμα.

Simulated Annealing Pseudocode

```
1: S = Random Placement();
2: T = InitialTemperature();
3: R_lim = InitialR_lim();
4: while (ExitCriterion () == FALSE)
5: {
6:         while (InnerLoopCriterion () == FALSE)
7:         {
8:                 S_new= GenerateViaMove(S, R_lim);
9:                 ΔC = Cost(S_new) - Cost(S);
10:                r = random (0,1);
11:                if(r<e^(-ΔC/T))
12:                {
13:                        S= S_new;
14:                }
15:        }
16:        T=UpdateTemp();
17:        R_lim=UpdateR_lim();
18:}
```

Πιν. 2.1 Ψευδοκώδικας Simulated Annealing

2.2.5 Δρομολόγηση - Routing

Η διαδικασία δρομολόγησης (Routing) έχεις ως στόχο την σύνδεση των πλέον τοποθετημένων CLBs και I/O pads μεταξύ τους, χρησιμοποιώντας τους υπάρχοντες πόρους που διαθέτει η κάθε αρχιτεκτονική FPGA. Η αναπαράσταση που συνήθως χρησιμοποιείται είναι ένα κατευθυνόμενο γράφημα του οποίου οι κόμβοι αντιστοιχούν στους αγωγούς και ακίδες (pins) των μπλοκ λογικής ενώ η πιθανές συνδέσεις μεταξύ τους αντιστοιχούν στα τόξα. Παρά το γεγονός ότι μελετήθηκε η χρήση μη κατευθυνόμενου γραφήματος, τελικά αποφασίστηκε ότι είναι απαραίτητο να είναι κατευθυνόμενο ώστε να μοντελοποιηθούν κατάλληλα στοιχεία του κυκλώματος όπως tri-sate buffers και πολυπλέκτες (multiplexers).

Μια σύνδεση μεταξύ CLBs ή/και I/O pads αντιστοιχεί στην εύρεση μονοπατιού στο ανωτέρω γράφημα, το οποίο να συνδέει τους αντίστοιχους κόμβους. Είναι λογικό λοιπόν, ότι όσο περισσότεροι πόροι χρησιμοποιούνται για να επιτευχθεί μια σύνδεση τόσο μικρότερη συχνότητα λειτουργίας του τελικού κυκλώματος μπορεί να επιτευχθεί. Κατά συνέπεια οι συνδέσεις καλό είναι να έχουν όσο το δυνατόν μικρότερο μήκος. Αυτό σε συνδυασμό με το γεγονός ότι οι FPGA έχουν περιορισμένο αριθμό τέτοιων πόρων, δύναται να οδηγήσει σε συνωστισμό σε ορισμένα μονοπάτια καθώς παραπάνω από δύο συνδέσεις μεταξύ στοιχείων του κυκλώματος είναι πιθανόν να χρησιμοποιεί αυτόν τον αγωγό για να έχει την μικρότερη δυνατή καθυστέρηση. Αυτό οδήγησε στην ανάγκη ύπαρξης κάποιου συστήματος αποφυγής κορεσμού των πόρων.

Σε ορισμένα κυκλώματα, δεν είναι δυνατόν πάντα να πραγματοποιήσουμε τις συνδέσεις μεταξύ δύο στοιχείων με χρήση του ελαχίστου απαραίτητου δικτύου αγωγών. Σε αυτές τις περιπτώσεις, προτεραιότητα έχουν οι συνδέσεις μεταξύ των στοιχείων που βρίσκονται πάνω στο κρίσιμο μονοπάτι (critical path). Το critical path είναι το μονοπάτι που συνδέει όλα τα στοιχεία του κυκλώματος τα οποία εκτελούν την πιο χρονοβόρα διαδικασία. Οι δρομολογητές που εφαρμόζουν τον παραπάνω κανόνα για την προτεραιότητα δέσμευσης πόρων, ονομάζονται timing-driven. Αντιθετα, αν ο σκοπός του δρομολογητή είναι απλά να βρεθεί κάποιος τρόπος να δρομολογηθούν όλες οι συνδέσεις μεταξύ των στοιχείων ανεξαρτήτως από την αποδοτικότητά τους, ο αλγόριθμος δρομολόγησης ανήκει στην κατηγορία των routability-driven.

Οι αλγόριθμοι δρομολόγησης χωρίζονται σε δύο κατηγορίες ανάλογα με τον τρόπο που γίνεται η δρομολόγηση. Οι combined global-detailed Routers [15] [25] [26] [27] [28] [29] [30] [31] καθορίζουν ένα πλήρες μονοπάτι σε ένα βήμα ενώ οι two-step routing algorithms στο πρώτο βήμα επιλέγουν ποιά κανάλια αγωγών και ποιοί ακροδέκτες (pins) του καθενός μπλοκ λογικής θα χρησιμοποιηθούν και στο δεύτερο βήμα οριστικοποιείται ο συγκεκριμένος αγωγός που θα

χρησιμοποιηθεί για την σύνδεση. Ο τρέχων επικρατέστερος αλγόριθμος δρομολόγησης και αυτός που χρησιμοποιείται από το εργαλείο στο οποίο εστιάζει η μελέτη είναι ο Pathfinder ο οποίος αναλύεται παρακάτω.

Αλγόριθμος Pathfinder

Ο αλγόριθμος Pathfinder[24] προσπαθεί να βρει το κατάλληλο δέντρο που ενώνει ένα CLB πηγή κάθε πλέγματος (net) με όλα τα CLBs και I/O pads με τα οποία συνδέεται μέσω ενός κατευθυνόμενου γραφήματος G(V,E). Το V αποτελεί το σύνολο των ακροδεκτών (pins) και των αγωγών ενώ το E είναι το σύνολο των ακμών του γραφήματος και αντιπροσωπεύει τις πιθανές συνδέσεις μεταξύ δύο στοιχείων του V.

Ο Pathfinder χρησιμοποιεί επαναλήψεις στην προσέγγιση του προβλήματος. Αρχικά κάθε πλέγμα δρομολογείται χωρίς να λαμβάνονται υπ' όψιν οι περιορισμοί της αρχιτεκτονικής. Για την δρομολόγηση χρησιμοποιείται ο αλγόριθμος εύρεσης ελαχίστου μονοπατιού Dijkstra [19]. Αναπόφευκτα όταν το κύκλωμα είναι μεγάλο, θα υπάρχει μεγαλύτερος αριθμός πλεγμάτων nets που διατρέχουν το ίδιο μονοπάτι από ότι επιτρέπεται από τους πόρους όπως αυτοί ορίζονται στην αρχιτεκτονική της FPGA. Στις επόμενες επαναλήψεις, λαμβάνεται υπ' όψιν και το πόσο συνωστισμένο είναι το κάθε κανάλι και ανάλογα με το αν είναι η όχι, προσαυξάνεται το κόστος σύνδεσης μέσω αυτού. Έτσι αν μια οδός είναι συνωστισμένη είναι πιο πιθανό μια σύνδεση να γίνει μέσω κάποιου εναλλακτικού μονοπατιού. Αν κατά την αναζήτηση νέων διαδρομών, οι πόροι που οδηγούν σε αυτές είναι περισσότερο συνωστισμένοι τότε είναι πιθανό η συγκεκριμένη σύνδεση να ακολουθήσει το μονοπάτι που είναι ήδη συνωστισμένο.

Το κόστος χρήσης ενός πόρου δρομολόγησης n κατά την διάρκεια μιας εκ των επαναλήψεων δίνεται από τον τύπο:

$$C_n=(b_n+h_n)p_n$$

όπου C_n το τελικό κόστος χρήσης του πόρου, b_n βασικό κόστος πριν τις προσαυξήσεις, h_n ένα επιπλέον βάρος το οποίο προκύπτει από το πόσο συνωστισμένος ήταν ο συγκεκριμένος πόρος σε προηγούμενες επαναλήψεις και p_n είναι μια τιμή ανάλογα με το πόσα πλέγματα nets χρησιμοποιούν τον συγκεκριμένο πόρο.

Στην αναζήτηση μονοπατιού χρησιμοποιείται ένα μέγεθος ακόμη ως παράμετρος. Πρόκειται για την καθυστέρηση του κρίσιμου μονοπατιού (critical path delay) . Η καθυστέρηση κρίσιμου μονοπατιού αντιστοιχεί στην μέγιστη καθυστέρηση που παρουσιάζεται για οποιαδήποτε έγκυρη σύνδεση με βάση το γράφημα μεταξύ ενός στοιχείου πηγής(source) και ενός στοιχείου καταναλωτή(sink). η μέγιστη συχνότητα λειτουργίας του εκάστοτε κυκλώματος είναι αντιστρόφως ανάλογη της καθυστέρησης αυτής. Για αυτόν τον λόγο χρησιμοποιείται σαν παράμετρος στον αλγόριθμο ακολούθως:

$$C_n=A_{ij}d_n + (1- A_{ij}) (b_n+h_n)p_n$$

όπου C_n το κόστος χρήσης του πόρου n, d_n η καθυστέρηση που προκύπτει από την χρήση του πόρου n και A_{ij} η κρισιμότητα του συγκεκριμένου πόρου η οποία δίνεται από την εξίσωση

$$A_{ij}=D_{ij}/D_{max}$$

Με D_{ij} συμβολίζουμε την μέγιστη καθυστέρηση οποιουδήποτε συνδυαστικού μονοπατιού που διέρχεται από το συγκεκριμένο net και me D_{max} την καθυστέρηση κρίσιμου μονοπατιού του κυκλώματος. Με αυτόν τον τρόπο επιτυγχάνεται, όταν μια σύνδεση βρίσκεται πάνω στο κρίσιμο μονοπάτι, το κόστος χρήσης του πόρου να είναι ίσο με το βασικό κόστος ενώ αν βρίσκεται μερικώς η καθόλου πάνω σε αυτό τότε το κόστος χρήσης του πόρου να προσαυξάνεται αναλόγως με το πόσο μεγάλο μέρος του κρίσιμου μονοπατιού μοιράζονται.

Χρονική Ανάλυση - Timing Analysis

Η χρονική ανάλυση χρησιμοποιείται ώστε να υπολογιστεί η ταχύτητα του κυκλώματος κατά την διάρκεια της δρομολόγησης και μετά το πέρας αυτής. Επιπλέον, μέσω της χρονικής ανάλυσης υπολογίζεται η χαλαρότητα (slack) για κάθε σύνδεση μεταξύ ενός μπλοκ λογικής πηγής και ενός καταναλωτή σήματος (sink) ώστε να αποφασιστεί ποιές συνδέσεις πρέπει να έχουν προτεραιότητα κατά την δρομολόγηση για την δέσμευση πόρων που βρίσκονται πάνω στην ελάχιστη διαδρομή.

Αρχικά το κύκλωμα αναπαριστάται με παρόμοια μέθοδο με την δρομολόγηση μέσω ενός κατευθυνόμενου γραφήματος. Στο γράφημα αυτό, οι κόμβοι αντιπροσωπεύουν τα pins εισόδου και εξόδου λογικών στοιχείων του κυκλώματος (π.χ. LUTs, registers, I/O pads). Οι συνδέσεις μεταξύ των παραπάνω στοιχείων αναπαριστούνται με ακμές στο κατευθυνόμενο γράφημα. Ακμές επίσης προστίθενται και μεταξύ των pins εισόδου των μπλοκ συνδυαστικής λογική και των εξόδων τους. Σε αυτές τις ακμές προσάπτουμε μια τιμή που αντιστοιχεί στην καθυστέρηση του υλικού μεταξύ των κόμβων. Τα pins εισόδου των καταχωρητών δεν ενώνονται με αυτά εξόδου τους.

Για να υπολογιστεί η τελική καθυστέρηση, το γράφημα διατρέχεται κατά πλάτος πρώτα ξεκινώντας από τα σημεία εισόδου (input pads, έξοδοι καταχωρητών) και σταδιακά υπολογίζεται ο χρόνος άφιξης όλων των κόμβων του κυκλώματος με βάση τον τύπο :

$$T_{arr}(i) = \max_{\forall j \in fanin(i)}\{T_{arr}(j) + delay(j,i)\}$$

όπου ο κόμβος i είναι ο κόμβος του οποίου ο χρόνος άφιξης είναι προς υπολογισμό και $delay(j,i)$ είναι η καθυστέρηση που αναγράφεται στην ακμή που συνδέει τον κόμβο j με τον κόμβο i. Η τελική καθυστέρηση του κυκλώματος ισούται με τον μέγιστο χρόνο άφιξης όλων των κόμβων.

Κατά την διάρκεια του σταδίου placement και routing, είναι συχνά χρήσιμο να γνωρίζουμε πόση καθυστέρηση μπορεί να προστεθεί σε μια σύνδεση του κυκλώματος πριν η καθυστέρηση κάποιας διαδρομής της οποίας μέλος είναι η συγκεκριμένη σύνδεση γίνει μεγαλύτερη από την τρέχουσα καθυστέρηση κρίσιμου μονοπατιού. Αυτή η καθυστέρηση που είναι δυνατόν να προστεθεί ονομάζεται χαλαρότητα (slack). Ο υπολογισμός της γίνεται με παραπλήσιο τρόπο με τον υπολογισμό της μέγιστης καθυστέρησης του κυκλώματος. Αρχικά υπολογίζεται η συνολική καθυστέρηση όλων των μονοπατιών τους κυκλώματος και ευρίσκεται η καθυστέρηση κρίσιμου μονοπατιού. Στην συνέχεια, ξεκινώντας από όλα τα σημεία εξόδου του γραφήματος (register inputs, output pads) ορίζεται ότι εκεί ο μέγιστος επιτρεπτός χρόνος T_{req} είναι ίσος με την μέγιστη καθυστέρηση και με χρήση οπισθοδιάδοσης υπολογίζεται η μέγιστη επιτρεπτή καθυστέρηση και η χαλαρότητα του κάθε κόμβου σύμφωνα με τους τύπους:

$$T_{req}(i) = \min_{\forall j \in fanout(i)}\{T_{req}(j) - delay(j,i)\}$$

$$slack(i,j) = T_{req}(j) - T_{arr}(i) - delay(i,j)$$

2.2.6 Παραγωγή Ρεύματος Bits - Bitstream Generation

Η ροή λειτουργίας ενός CAD συστήματος τελειώνει με το στάδιο της δημιουργίας Bitstream η οποία και προγραμματίζει τελικώς την FPGA. Μετά το πέρας της τοποθέτησης και δρομολόγησης το εργαλείο CAD παράγει ένα ρεύμα δεδομένων (bits) το οποίο λαμβάνοντας τα μοντελοποιημένα πλέον στοιχεία από τα στάδια technology mapping, packing και placement, αναθέτει στα κελιά μνήμης SRAM την τιμή 0 ή 1 ώστε το κάθε Look-Up Table να υλοποιεί το μέρος της λογικής του κυκλώματος που του αντιστοιχεί. Τα δεδομένα του σταδίου Routing χρησιμοποιούνται για να προγραμματιστούν τα SRAM bits των switch boxes και των connection boxes. Μόλις ολοκληρωθεί αυτή η διαδικασία, η FPGA είναι έτοιμη να εξομοιώσει το κύκλωμα προς υλοποίηση.

2.3 Περίληψη

Σε αυτό το κεφάλαιο αρχικά αναφέραμε γενικές πληροφορίες για τις FPGAs και επεξηγήσαμε τους όρους που απαιτούνται για να περιγραφούν οι αρχιτεκτονικές και τα λογικά μπλοκ τους. Στην συνέχεια, αναλύσαμε τα είδη αρχιτεκτονικών τους με βάση τον τρόπο που γίνεται ο προγραμματισμός

τους, με βάση τα λογικά μπλοκ και με βάση του τρόπου που υλοποιείται η δρομολόγηση σε αυτές. Ακολούθησε ο ορισμός των CAD εργαλείων καθώς και ο λόγος για τον οποίο είναι απαραίτητα ενώ αμέσως μετά αναλύθηκαν τα στάδια της ροής λειτουργίας ενός CAD. Εστιάσαμε κυρίως στο στάδιο της τοποθέτησης (placement) και τις κατηγορίες αλγορίθμων, ενώ συζητήσαμε τόσο το πως αντιμετωπίστηκε το πρόβλημα σε παλαιότερες προσεγγίσεις όσο και τα πλεονεκτήματα και μειονεκτήματά τους.

Στα επόμενα δύο κεφάλαια δίνονται οι απαραίτητες πληροφορίες που θα οδηγήσουν αργότερα στην δική μας πρόταση πάνω στο θέμα της τοποθέτησης (placement). Πιο συγκεκριμένα στο κεφάλαιο 3 περιγράφεται αρχικά με λίγα λόγια το εργαλείο CAD που χρησιμοποιείται για την υλοποίηση αυτής της μελέτης καθώς και πως αντιμετωπίζει το πρόβλημα του placement με καινοτομία στο οποίο στοχεύει αυτή η μελέτη. Το τέταρτο κεφάλαιο ασχολείται με τα βασικά του παράλληλου προγραμματισμού και της παράλληλης επεξεργασίας, αναλύει τους περιορισμούς που προκύπτουν και επεξηγεί ορισμένα προγραμματιστικά μοτίβα τα οποία βοηθούν στην μεγαλύτερη αξιοποίηση των πόρων παραλληλισμού.

3.Εργαλείο προς μελέτη και καινοτομίες

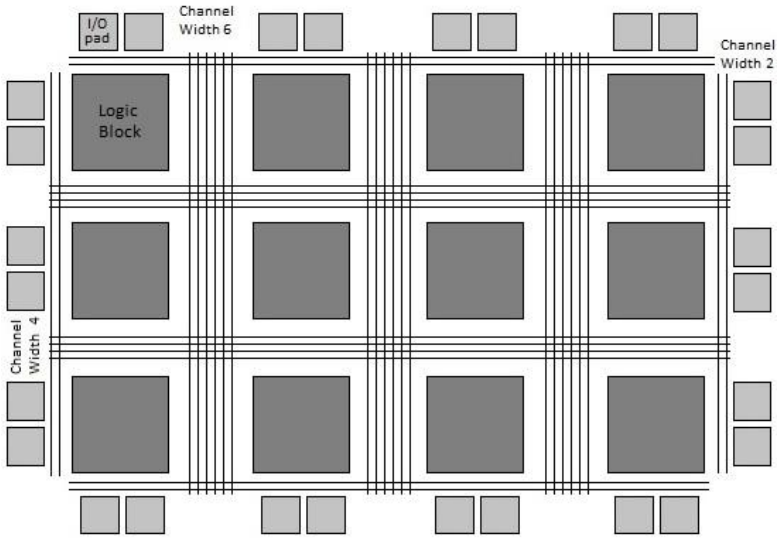
Το εργαλείο στο οποίο βασίστηκε η διατριβή αυτή για να μελετήσει την παραλληλοποίηση ενός αλγορίθμου τοποθέτησης, είναι το T-VPack, και πιο συγκεκριμένα το Versatile Place & Route (VPR) κομμάτι του, το οποίο είναι υπεύθυνο για την φάση της τοποθέτησης placement της ροής λειτουργίας. Οι Vaughn Betz et al. [12] παρουσιάζουν αρχικά το εργαλείο ως ευέλικτο αρκετά ώστε να μπορεί να χρησιμοποιηθεί σε πολλά διαφορετικά ήδη αρχιτεκτονικών FPGA. Το VPR στηρίζεται στον αλγόριθμο Simulated Annealing διατηρώντας τα καλύτερα στοιχεία από παλαιότερες προτάσεις και βελτιώνοντας σημαντικά ορισμένα προβλήματα που παρουσιάζονταν σε αυτές.

3.1 Επισκόπηση του εργαλείου VPR

Στο VPR, μια FPGA μοντελοποιείται ως ένα σετ από διακριτές θέσεις οι οποίες μπορούν να καταβληθούν από μπλοκ λογικής (Logic Blocks) ή μπλοκ εισόδου/εξόδου (I/O pads). Κατά την έναρξη της εκτέλεσής του δέχεται ένα αρχείο το οποίο περιγράφει την αρχιτεκτονική της FPGA την οποία στοχεύει. Αυτό το αρχείο πρέπει να ορίζει τρία πράγματα ώστε να εκμεταλλεύεται την ευελιξία που παρέχει το συγκεκριμένο εργαλείο:

- Τον αριθμό των ακίδων εισόδου και εξόδου των Logic Blocks.
- Τον αριθμό των I/O pads που χωράνε ανά γραμμή η στήλη της FPGA
- Τα σχετικά εύρη των διαφόρων καναλιών δρομολόγησης που διατρέχουν την FPGA

Επίσης οι διαστάσεις του πίνακα των Logic Blocks πρέπει να οριστούν στην γραμμή εντολών κατά την εκτέλεση αλλιώς επιλέγεται η μικρότερη δυνατή εκδοχή η οποία μπορεί να χωρέσει το κύκλωμα.



Εικ. 3.1 Μοντέλο αρχιτεκτονικής FPGA για το VPR.

Στην εικόνα 3.1 φαίνεται το μοντέλο που χρησιμοποιεί το VPR. Ο πίνακας λογικών μπλοκ έχει διαστάσεις 4x3 μπλοκ. Ο αριθμός των I/O pads τόσο για τις γραμμές όσο και για τις στήλες στην συγκεκριμένη αρχιτεκτονική είναι δύο και τα κανάλια μεταξύ των στοιχείων του πίνακα έχουν εύρος

σε ορισμένα σημεία δύο σε άλλα τέσσερα και σε άλλα έξι. Στο συγκεκριμένο μοντέλο τα I/O pads είναι πάντα περιμετρικά και κατά συνέπεια μπορούν να τοποθετηθούν μόνο εκεί.

Επίσης ο χρήστης μπορεί να ορίσει συγκεκριμένη θέση για το κάθε I/O pad ή να ρυθμίσει μέσω του αρχείου αρχιτεκτονικής να οριστούν τυχαία κλειδωμένες θέσεις για αυτά από το VPR. Αυτή η δυνατότητα του εργαλείου επιτρέπει την αξιολόγηση της τυχαίας επιλογής ακίδων σε διαφορετικές αρχιτεκτονικές. Τα προκαθορισμένης θέσης I/O pads εμφανίζονται πολύ συχνά, καθώς συχνά το κύκλωμα με το οποίο συνδέονται οι FPGA είναι συνήθως κατασκευασμένο πριν ολοκληρωθεί ο σχεδιασμός του κυκλώματος προς υλοποίηση σε αυτές. Σε τέτοιες περιπτώσεις, το εργαλείο placement πρέπει να λάβει υπ' όψιν του τις θέσεις με τις οποίες μια FPGA μπορεί να συνδεθεί με το εκάστοτε εξωτερικό κύκλωμα.

3.2 Προσαρμοζόμενο Annealing Schedule

Όπως προαναφέρθηκε, το VPR χρησιμοποιεί τον αλγόριθμο Simulated Annealing για την τοποθέτηση των στοιχείων του κυκλώματος προς υλοποίηση πάνω στην FPGA. Το πρόγραμμα Annealing που χρησιμοποιείται είναι δυνατόν να προσαρμοστεί σε διαφορετικές αρχιτεκτονικές ενώ έχει πάρει και αρκετά στοιχεία από παλαιότερες έρευνες. Ακολουθεί αναλυτικότερα η περιγραφή του:

3.2.1 Αρχική θερμοκρασία (InitialT)

Η αρχική θερμοκρασία υπολογίζεται με τον ίδιο τρόπο που την υπολογίζουν οι Huang et al [13]. Έστω ότι ο συνολικός αριθμός των μπλοκ και των I/O pads σε ένα κύκλωμα συμβολίζεται με N_{blocks}. Αρχικά τοποθετούνται όλα τα στοιχεία του κυκλώματος τυχαία και στην συνέχεια εκτελούνται N_{blocks} τυχαίες κινήσεις. Στην συνέχεια υπολογίζεται η τυπική απόκλιση του κόστους αυτών των διαφορετικών απεικονίσεων και ορίζεται η αρχική θερμοκρασία σε είκοσι φορές αυτήν την τυπική απόκλιση με συνέπεια όλες οι κινήσεις αρχικά να είναι αποδεκτές.

3.2.2 Αριθμός κινήσεων ανά θερμοκρασία

Ο αριθμός κινήσεων προς αξιολόγηση ανά θερμοκρασία υπολογίζεται με τον ίδιο τρόπο με τον οποίο προτείνεται από τους Swarz και Sechen [18].

$$MovesPerTemperature = InnerNum(N_{blocks})^{4/3}$$

Η συνήθης προκαθορισμένη τιμή για το InnerNum είναι 10 και μπορεί να οριστεί σε διαφορετική τιμή από την κονσόλα κατά την εκκίνηση. Ο αριθμός κινήσεων ανά θερμοκρασία είναι ανάλογος της ποιότητας του τελικού κυκλώματος και του χρόνου που απαιτείται για την ολοκλήρωση της διαδικασίας της τοποθέτησης. Ελαττώνοντας για παράδειγμα κατά 10% τον αριθμό κινήσεων έχουμε απώλεια ποιότητας κυκλώματος κατά 10% περίπου και εκτελείται έως και 10 φορές πιο γρήγορα.

3.2.3 Παράγοντας R_{limit}

Στην πρόταση των Lam και Delosme [14] έχει υλοποιηθεί ένας παράγοντας R_{limit} με σκοπό την ρύθμιση του αριθμού κινήσεων που γίνονται αποδεκτές α περίπου στο 44%. Αυτό επιτυγχάνεται με την χρήση αυτής της τιμής του α για τον υπολογισμό του παράγοντα R_{limit} ο οποίος είναι υπεύθυνος στο να περιορίζει την απόσταση των κινήσεων που εξετάζονται προς αποδοχή. Σε αυτή την υλοποίηση μόνο κινήσεις όπου τα λογικά μπλοκ που απέχουν μικρότερη απόσταση μεταξύ τους από R_{limit} εξετάζονται. Όσο μικρότερο το R_{limit} τόσο το α αυξάνεται καθώς οι κινήσεις που εξετάζονται είναι πολύ κοντινές με συνέπεια το κόστος τους να μην αλλάζει πολύ και να αυξάνεται η πιθανότητα αποδοχής τους. Αρχικά το R_{limit} ορίζεται σε μια τιμή που να επιτρέπει την οποιαδήποτε κίνηση σε ολόκληρη την FPGA και σταδιακά, κάθε φορά που ελαττώνεται η θερμοκρασία, προσαρμόζεται σε νέα τιμή λαμβάνοντας υπ' όψιν το ποσοστό των κινήσεων που έγιναν αποδεκτές στην προηγούμενη θερμοκρασία όπως φαίνεται παρακάτω:

$$R_{limit}^{new} = R_{limit}^{old} (1 - 0.44 + \alpha)$$

Στην συνέχεια αυτή η τιμή ελέγχεται αν είναι μεγαλύτερη από την μέγιστη διάσταση της FPGA αρχιτεκτονικής και αν είναι τότε η το καινούριο R_{limit} παίρνει την τιμή τησ μέγιστης διάστασης. Αντίστοιχα αν το R_{limit} προκύψει μικρότερο του ένα τότε ορίζεται στην τιμή ένα πράγμα που συμβαίνει στα τελευταία στάδια της διαδικασίας τοποθέτησης.

3.2.4 Ρύθμιση θερμοκρασίας

Μετά από αριθμό κινήσεων ίσο με *MovesPerTemperature* η θερμοκρασία ενημερώνεται σε κάποια νέα χαμηλότερη τιμή. Όταν η θερμοκρασία είναι πολύ υψηλή πρακτικά μεταφερόμαστε μεταξύ διαφορετικών απεικονίσεων με μικρό κέρδος από πλευράς κόστους. Το ίδιο συμβαίνει και στην περίπτωση που η θερμοκρασία είναι πολύ μικρή και οι κινήσεις έχουν καταντήσει εντελώς τοπικές αφού έχει επιτευχθεί σε μεγάλο βαθμό η καλή ποιότητα του τελικού κυκλώματος. Γι αυτό η θερμοκρασία σε αυτά τα στάδια πέφτει γρηγορότερα ενώ στα στάδια που είναι πιο αποδοτικά η θερμοκρασία πέφτει πιο αργά με συνέπεια να εκτελούνται περισσότερες κινήσεις σε αυτά. Με την χρήση μιας παραμέτρου γ καθορίζεται ο ρυθμός με τον οποίο ελαττώνεται η θερμοκρασία. Όταν το α είναι κοντά στο 0.44 το γ τείνει στο ένα ενώ όταν το α τείνει στο 0 ή στο 1 τότε το γ έχει πολύ χαμηλότερη τιμή. Η νέα θερμοκρασία ορίζεται από τον τύπο:

$$T_{new} = \gamma\, T_{old}$$

3.2.5 Κριτήριο εξόδου

Η διαδικασία της τοποθέτησης ολοκληρώνεται όταν η θερμοκρασία γίνει χαμηλότερη από ένα μικρό ποσοστό του μέσου κόστους ενός πλέγματος net καθώς σε αυτή την περίπτωση είναι πολύ μικρή η πιθανότητα να γίνει αποδεκτή μια νέα κίνηση. Έτσι ...

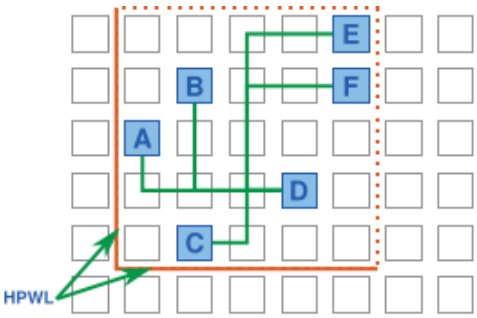
$$T < \varepsilon\ (\mathrm{Cost}/N_{nets})$$

όπου ε είναι ένας παράγοντας συνήθως μεταξύ του 0.05 και του 0.005 ενώ N_{nets} ο αριθμός των πλεγμάτων nets στο κύκλωμα προς υλοποίηση.

3.2.6 Κόστος

3.2.6.1 Μοντέλο Κόστους

Το VPR χρησιμοποιεί το μοντέλο προσέγγισης μήκους αγωγών Half Perimeter Wire Length (HPWL) ή αλλιώς Bounding Box Wire Length κατά το οποίο για 3 ή λιγότερα λογικά μπλοκ στο πλέγμα, το μήκος των αγωγών προς υλοποίησή του ισούται με την ημιπερίμετρο του ελαχίστου ορθογωνίου που μπορεί να περικλύσει τα λογικά μπλοκ στις θέσεις τους(Bounding Box). Επειδή το συγκεκριμένο μοντέλο υποεκτιμά το πραγματικό μήκος των αγωγών και χάνει σε ακρίβεια όσο περισσότερα από τρία λογικά μπλοκ αποτελούν το πλέγμα, στην αρχική μορφή του συγκεκριμένου μοντέλου προστέθηκε ο παράγοντας q(i). Αυτός ο παράγοντας παίρνει την τιμή ένα όταν τα λογικά μπλοκ στο πλέγμα είναι μέχρι τρία και αυξάνεται σταδιακά στην τιμή 2.79 για πλέγματα με 50 τερματικά.



Εικ 3.2 Μοντέλο HWPL για έξι τερματικά (A-F). Το Bounding Box φαίνεται με διακεκομμένη πορτοκαλί γραμμή ενώ η ημιπερίμετρος είναι επισημασμένη με συνεχή πορτοκαλί γραμμή.

3.2.6.2 Συνάρτηση Κόστους

Η συνάρτηση κόστους που χρησιμοποιείται στο VPR είναι σχεδιασμένη να λαμβάνει μερικώς υπ' όψιν το γεγονός ότι διαφορετικά σημεία στην FPGA μπορεί να έχουν διαφορετικό εύρος καναλιών προς δρομολόγηση και να προσπαθεί να αποφύγει την συμφόρηση στο μέλλον. Η σχέση η οποία υπολογίζει το κόστος είναι

$$\text{Cost}_{\text{linear congestion}} = \sum_{i=1}^{Nnets} \text{q(i)}[\frac{bbx(i)}{\text{Cav},x^{\beta}} + \frac{bby(i)}{\text{Cav},y^{\beta}}]$$

Το συνολικό κόστος ισούται με το άθροισμα για όλα τα πλέγματα του γινομένου των q του κάθε πλέγματος και του αθροίσματος των κλασμάτων του μήκους του περικλείοντος ορθογωνίου προς το μέσο εύρος καναλιού κατά μήκος του ορθογωνίου και του πλάτους του προς το μέσο εύρος καναλιού κατά πλάτος.

Αυτή η συνάρτηση κόστους φροντίζει ώστε να είναι πιο δύσκολο να γίνουν δεκτές κινήσεις οι οποίες θα δημιουργήσουν συμφόρηση σε περιοχές περιορισμένου εύρους καναλιών. Ο εκθέτης β επιτρέπει στον χρήστη να επηρεάσει το ποσοστό κατά το οποίο το μέσο εύρος παίζει ρόλο κατά πόσο μια κίνηση θα γίνει δεκτή ή όχι. Για τιμή β μηδέν η παραπάνω σχέση μετατρέπεται στην κανονική σχέση που χρησιμοποιείται κατά το HWPL μοντέλο ενώ για την τιμή ένα πειραματικά βρέθηκε ότι έχουμε τα καλύτερα αποτελέσματα από πλευράς ποιότητας.

Το C_{av} εξαρτάται μόνο από την χωρητικότητα των καναλιών (τα οποία είναι σταθερά και περιγράφονται στο αρχείο αρχιτεκτονικής που παρέχεται κατά την εκτέλεση) και από τις ελάχιστες και μέγιστες συντεταγμένες του περικλείοντος ορθογωνίου. Σε μια FPGA όπου όλα τα κανάλια έχουν το ίδιο εύρος, το C_{av} έχει σταθερή τιμή. Στην περίπτωση όμως που το εύρος των καναλιών διαφέρει από περιοχή σε περιοχή η συγκεκριμένη συνάρτηση κόστους επιτυγχάνει καλύτερα αποτελέσματα από πλευράς ποιότητας τελικού placement.

3.3 Παράλληλη εκτέλεση του αλγορίθμου Simulated Annealing

Η ανάγκη για μεγαλύτερη πολυπλοκότητα εξομοιούμενων κυκλωμάτων οδήγησε στην ραγδαία αύξηση της χωρητικότητας των FPGA σε λογική. Ειδικά μετά την επίτευξη όλο και μικρότερων κλιμάκων ολοκλήρωσης και με αφετηρία το 1982, όπου οι πύλες που διέθετε η τότε μέση FPGA ήταν 8192, σήμερα έχουμε φτάσει να ξεπερνάμε τα μερικά εκατομμύρια πύλες. Ο σειριακός Simulated Annealing αλγόριθμος είναι ένας αργός αλγόριθμος του οποίου ο χρόνος εκτέλεσης για μικρές υλοποιήσεις είναι ανεκτός. Παρόλα αυτά, καταντάει να είναι ασύμφορος όσο πιο πολύ αυξάνεται η λογική η οποία απαιτείται να χωρέσει στην FPGA. Με στόχο την επιτάχυνση της διαδικασίας, διάφορες έρευνες προσπάθησαν να επιλύσουν αυτό το πρόβλημα με την χρήση παράλληλης επεξεργασίας. Οι μέθοδοι αυτοί χωρίζονται σε τρεις βασικές κατηγορίες:

Ανεξάρτητα σύνολα (Independent Set Finding): Σε αυτές τις υλοποιήσεις οι πυρήνες αξιολογούν κινήσεις ταυτόχρονα και όταν ένας βρει μια έγκυρη κίνηση οι άλλοι σταματούν την αναζήτηση και απορρίπτουν τις κινήσεις που αξιολογούν εκείνη την στιγμή. Αυτή η μέθοδος παραλληλισμού δεν έχει σημαντικές απώλειες σε ποιότητα τελικής τοποθέτησης ενώ στους τέσσερις πυρήνες που δοκιμάστηκε αρχικά και δεδομένου ότι το ποσοστό των κινήσεων που γίνονται αποδεκτές ήταν μικρό, επιτεύχθηκε 3.5 φορές επιτάχυνση. Σε άλλες υλοποιήσεις αυτής της κατηγορίας χρησιμοποιείται ένας πυρήνας για να προτείνει κινήσεις σε ανεξάρτητα μεταξύ τους κελιά, ωστόσο η επιτάχυνση που επιτεύχθηκε ήταν πιο περιορισμένη καθώς απαιτείται συγχρονισμός μεταξύ των πυρήνων. Αυτού του τύπου οι αλγόριθμοι είναι μη ντετερμινιστικοί συνήθως. [32] [33] [34][41].

Τεμαχισμένη τοποθέτηση (Partitioned Placements): Σε αυτούς τους αλγορίθμους η αναπαράσταση της FPGA τεμαχίζεται σε κομμάτια και κάθε πυρήνας εκτελεί τοπική τοποθέτηση μόνο στα κομμάτια στα οποία του έχουν ανατεθεί. Αυτή η μέθοδος πολλές φορές στηρίζεται σε δεδομένα που μπορεί να έχουν αλλάξει κατά συνέπεια λειτουργούν με την λογική ότι μικρά λάθη είναι αποδεκτά. Περιστασιακά, ενημερώνουν τα δεδομένα τοποθέτησης ώστε τα λάθη που μπορεί να γίνουν να είναι μικρά και να μην καταλήξει το σύστημα να έχει σοβαρές αποκλίσεις σε ποιότητα. Οι καλύτερες υλοποιήσεις, κατά την διάρκεια της εκτέλεσης, αλλάζουν τα όρια ή την θέση των κομματιών που ανατίθενται σε κάθε πυρήνα ώστε να είναι δυνατή η μεταφορά στοιχείων από μια θέση που ανήκε σε ένα τμήμα σε θέση που ανήκε σε άλλο τμήμα, πράγμα το οποίο θα ήταν αδύνατο χωρίς αυτές τις μετατροπές. Η συγκεκριμένη κατηγορία, επιτυγχάνει τις καλύτερες επιταχύνσεις. Το

μειονέκτημα τις συγκεκριμένης προσέγγισης είναι συνήθως το γεγονός ότι υστερούν σε ποιότητα τελικής υλοποίησης καθώς παράγοντες όπως αργή μεταφορά στοιχείων από έναν υπόχωρο και μερικές φορές τα μη έγκυρα δεδομένα που χρησιμοποιούνται είναι δυνατόν να οδηγήσουν σε μη βέλτιστες καταστάσεις για τον ίδιο αριθμό κινήσεων ανά τοποθέτηση. Επίσης οι συγκεκριμένου τύπου αλγόριθμοι συνήθως είναι μη ντετερμινιστικοί και εξαρτώνται άμεσα από τον αριθμό πυρήνων. Χαρακτηριστικά παραδείγματα τέτοιων αλγορίθμων είναι οι [33] [35] [36] [37][42][43].

Υποθετικοί υπολογισμοί (Speculative Computation): Η κατηγορία αυτή εκμεταλλεύεται το δέντρο αποφάσεων για κάθε κίνηση κατά τον Simulated Annealing ώστε να παράγει έργο ανεξάρτητα με το αν μια κίνηση θα γίνει αποδεκτή η όχι. Πιο συγκεκριμένα ένας πυρήνας συνήθως ξεκινάει με την αξιολόγηση μίας κίνησης και άλλοι δύο συνεχίζουν ο ένας θεωρώντας ότι η κίνηση έγινε αποδεκτή και ο άλλος ότι απορρίφθηκε κ.ο.κ. Όταν ο πρώτος πυρήνας τελειώσει με την αξιολόγηση, τότε ανάλογα με αυτήν, η πρόοδος του ενός εκ των άλλων δύο απορρίπτεται και συνεχίζεται η εκτέλεση του αλγορίθμου από τον σωστό πυρήνα. Αυτή η μέθοδος καθιστά το αποτέλεσμα ίδιο με το αποτέλεσμα που θα έβγαινε από μια σειριακή εκτέλεση του αλγορίθμου χρησιμοποιώντας τα ίδια seeds για την παραγωγή τυχαίων αριθμών και κατά συνέπεια έχει και την ίδια ποιότητα τελικού κυκλώματος. Το μεγαλύτερο μειονέκτημα της συγκεκριμένης τεχνικής είναι ότι σε κάθε βήμα απορρίπτεται ένα μεγάλο μέρος έργου που έχει γίνει από τους πυρήνες για κάθε διαδρομή που δεν ακολουθήθηκε στο δέντρο αποφάσεων. Σε μη ιδανικές συνθήκες η απώλεια υπολογιστικού έργου μπορεί να ξεπεράσει και το πενήντα τοις εκατό. Επίσης, για να λειτουργήσουν χωρίς σφάλματα όλοι οι πυρήνες πρέπει να υπάρχει συγχρονισμός ο οποίος κοστίζει πολύ σε υπολογιστικό χρόνο και λόγω της αρχιτεκτονικής των SIMD συστημάτων (καρτών γραφικών) είναι αδύνατον να υλοποιηθούν σε αυτές. Οι παρακάτω αλγόριθμοι ανήκουν σε αυτήν την κατηγορία [38] [39] [40].

3.4 Περίληψη

Σε αυτό το κεφάλαιο παρουσιάστηκε το εργαλείο με το οποίο ασχολείται η μελέτη αυτή. Αναλύθηκαν όλα τα στοιχεία του προγράμματος Annealing που χρησιμοποιείται από αυτό ώστε να είναι κατανοητά αργότερα όταν θα χρησιμοποιηθούν και στην δικιά μας πρόταση καθώς επίσης και για να δοθεί έμφαση στις διαφορές στην προσέγγιση μεταξύ των αλλαγών που προτείνουμε και του αρχικού εργαλείου.

Στην συνέχεια έγινε μια γρήγορη αναφορά στις τρείς βασικές κατηγορίες αντιμετώπισης του θέματος της τοποθέτησης με χρήση παράλληλης επεξεργασίας.

4. Παράλληλη επεξεργασία

Ανέκαθεν ο άνθρωπος προσπαθούσε να μοντελοποιήσει τα πάντα γύρω του με βάση τις εμπειρίες και τις παρατηρήσεις του. Με αυτό τον τρόπο πολλές φορές μιμούμενος την φύση παρείχε λύσεις σε πολλά προβλήματα επιβίωσης του. Κάνοντας παρόμοιες παρατηρήσεις σύμφωνα με τον δικό του τρόπο επίλυσης προβλημάτων, κατέληξε αρχικά στο να αναπτύξει τεχνολογία η οποία λαμβάνει μια σειρά από εντολές και τις εκτελεί καταλήγοντας στο τελικό αποτέλεσμα όπως θα έκανε και ο ίδιος. Η υλοποίηση αυτού του τρόπου σκέψης οδήγησε στον σειριακό προγραμματισμό. Στην συνέχεια όντας σε ένα κοινωνικό περιβάλλον στο οποίο συνεργάζεται με άλλους συνανθρώπους του, έγινε αντιληπτό ότι αυτή η συνεργασία είναι ο λόγος που αρκετά θέματα που απασχολούσαν την επιβίωσή του σαν είδος, είχαν πλέον ξεπεραστεί πράγμα το οποίο θα ήταν αδύνατον αν ήταν μόνος του. Αντίστοιχα, στον προγραμματισμό ακολουθώντας παρόμοια μοτίβα σκέψης, οδηγηθήκαμε στην παράλληλη επεξεργασία.

Αρχικά, οι πρώτοι υπολογιστές λειτουργούσαν με βάση τον σειριακό προγραμματισμό. Όσο η τεχνολογία βελτιωνόταν ραγδαία, ήταν φτηνότερο να επενδύσουν οι εταιρίες όπως η Intel στην αύξηση της συχνότητας λειτουργίας ενός επεξεργαστή παρά να ερευνήσουν νέους αλγορίθμους για τα υπάρχοντα προβλήματα αλλά και μια νέα τεχνολογία που θα έδινε την δυνατότητα στο σύστημα να εκτελεί ταυτόχρονα παραπάνω από μία εντολές. Όμως η αύξηση των επιδόσεων των υπολογιστικών συστημάτων τα έκανε πιο δημοφιλή και προσιτά σε περισσότερο αριθμό χρηστών αυξάνοντας και τις ανάγκες για μεγαλύτερη υπολογιστική ισχύ. Έτσι προέκυψε ένας φαύλος κύκλος ο οποίος έφτασε στα όρια την προσέγγιση αυτή. Πιο συγκεκριμένα, τρία χαρακτηριστικά ώθησαν την τεχνολογία προς την κατεύθυνση της παράλληλης επεξεργασίας.

Παραλληλία σε επίπεδο εντολής (Instruction Level Parallelism- ILP) : Στα μονοπύρηνα συστήματα του παρελθόντος υπήρχε η δυνατότητα υπό συνθήκες να εκτελούνται παράλληλά παραπάνω από μια εντολές είτε εξαιτίας της διοχέτευσης (pipeline) είτε λόγω εκτέλεσης εντολών ασύγχρονα (out of order) στο υλικό. Παρά το γεγονός ότι αυτού του τρόπου ο παραλληλισμός εκτέλεσης έχει συμβάλει σημαντικά στις επιδόσεις των συστημάτων τόσο στο παρελθόν όσο και σήμερα, είναι γενικά αποδεκτό ότι έχει φτάσει στα όρια του.

Καθυστέρηση απόκρισης μνήμης (Memory Latency) : Ο χρόνος απόκρισης της μνήμης αποτελεί ένα πολύ σημαντικό παράγοντα στην τελική ταχύτητα εκτέλεσης ενός προγράμματος. Σε σχέση με το 2006 που η Intel κυκλοφόρησε στην αγορά πολυπύρηνο επεξεργαστή, η διαδικασία αυτή έχει βελτιωθεί πάρα πολύ. Αυτό οφείλεται στο ότι πλέον είναι δυνατόν να εκτελεστεί κάποια άλλη διεργασία όσο το αίτημα εξυπηρέτησης από την μνήμη αναμένει απάντηση αρκεί να έχουμε περισσότερες διεργασίες προς εκτέλεση από την δυνατότητα παραλληλίας που παρέχεται από τον αριθμό των επιπλέον πυρήνων και να μην απαιτούνται δεδομένα που προκύπτουν από την διευθέτηση της διεργασίας.

Κατανάλωση ενέργειας (Power Consumption) : Η διαρκής αύξηση της συχνότητας λειτουργίας των κυκλωμάτων απαιτεί όλο και μεγαλύτερη κατανάλωση ενέργειας. Ενέργεια που κατά την κατανάλωση της εκλύει θερμότητα. Τουλάχιστον με την τρέχουσα μέθοδο υλοποίησης υπολογιστικών συστημάτων είναι αδύνατον να επιτευχθούν συστήματα με μεγαλύτερη συχνότητα χωρίς την αναπόφευκτη υπερθέρμανση των κυκλωμάτων τους.

4.1 Σχεδιαστικές λεπτομέρειες παράλληλων συστημάτων

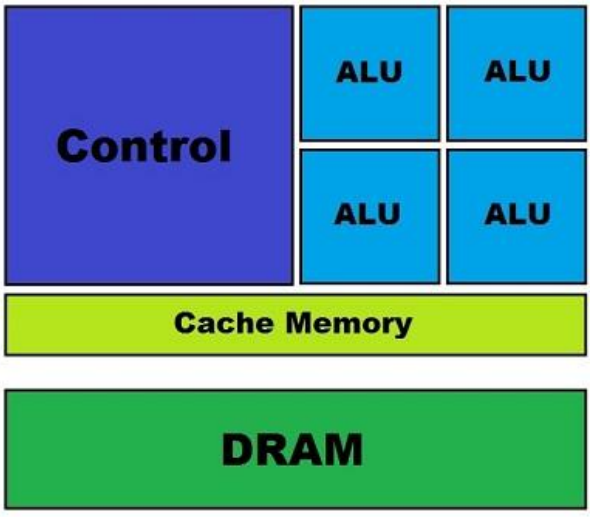
Η πλειοψηφία των προγραμμάτων ανήκει στην κατηγορία των σειριακών προγραμμάτων όπως έχουν περιγραφεί στο παρελθόν από τον Von Neumann. Η εκτέλεση αυτών των προγραμμάτων μπορεί να γίνει κατανοητή από τον άνθρωπο σαν μια ακολουθία ενεργειών που εκτελείται σειριακά και καταλήγει στην λύση του προβλήματος. Στο παρελθόν οι προγραμματιστές περίμεναν η επιτάχυνση να εξαρτάται από το υλικό και μόνο ενώ θεωρούταν απόλυτα φυσιολογικό ένα πρόγραμμα να εκτελείται ταχύτερα σε νεότερα συστήματα γεγονός το οποίο δεν είναι απαραίτητα ακριβές την σημερινή εποχή. Ένα σειριακό πρόγραμμα δεν είναι σχεδιασμένο να εκμεταλλεύεται επαρκώς την δυνατότητα που του παρέχεται από το υλικό για παραλληλία και χρειάζεται να τροποποιηθεί κατάλληλα. Μια μερική κατανόηση των παράλληλων συστημάτων είναι απαραίτητη για την εκμετάλλευση των δυνατοτήτων τους.

4.1.1 MIMD (Multiple Instructions Multiple Data)

Αύτη η κατηγορία, ως σκοπό έχει να επιταχύνει όσο γίνεται τα σειριακά προγράμματα προσθέτοντας παραπάνω από έναν πυρήνα στο σύστημα. Αυτή η κατηγορία διαθέτει μικρό αριθμό επεξεργαστικών μονάδων ενώ αφιερώνει ένα μεγάλο τμήμα της σε μια μονάδα ελέγχου η οποία μοιράζει τις εντολές κατάλληλα στην εκάστοτε επεξεργαστική μονάδα. Αυτού του είδους οι συσκευές έχουν την δυνατότητα να εκτελούν διαφορετικού τύπου εντολές σε διαφορετικές επεξεργαστικές μονάδες. Για παράδειγμα σε ένα τετραπύρηνο σύστημα σε κάποια χρονική στιγμή θα μπορούσε να εκτελείται ταυτόχρονα μια πρόσθεση, μια διαίρεση, μία αξιολόγηση boolean συνθήκης και ένας πολλαπλασιασμός. Οι σύγχρονοι επεξεργαστές υπολογιστών που κυκλοφορούν στο εμπόριο αυτή τησ στιγμή ανήκουν σε αυτήν την κατηγορία.

Το πλεονέκτημα των συστημάτων αυτής της κατηγορίας είναι ότι όσο περισσότεροι πυρήνες υπάρχουν και όσο το bus τους το επιτρέπει, τα σειριακά προγράμματα που εκτελούνται σε αυτούς θα παρουσιάζουν βελτίωση. Αυτό συμβαίνει διότι έχουν την δυνατότητα να εκτελούν διαφορετικού τύπου εντολές σε κάθε πυρήνα. Επίσης διαθέτουν πολύ εξελιγμένο και σύνθετο ελεγκτή μέρος του οποίου αποτελούν οι branch predictors οι οποίοι επιτυγχάνουν σε μεγάλο βαθμό την πρόβλεψη των διακλαδώσεων ώστε να μην χάνεται χρόνος κατά την αξιολόγηση των συνθηκών.

Τεχνικά το μεγαλύτερο μειονέκτημα αυτής της κατηγορίας είναι ότι διαθέτουν μόνο μνήμη cache και εκμεταλλεύονται την εξωτερική μνήμη του συστήματος. Αυτό έχει ως αποτέλεσμα αν δεν μπορεί να διοχετευθούν τα δεδομένα μέσω του διαύλου του συστήματος να καθυστερείται η απόκριση των μνημών .Επίσης, καταλαμβάνουν αρκετά μεγαλύτερο χώρο για την υλοποίησή τους και αυτό σε συνδυασμό με το μεγάλο κόστος τους περιορίζει τον αριθμό των πυρήνων που μπορεί να υπάρχουν σε ένα chip.



Εικ 4.1 Αρχιτεκτονική MIMD

4.1.2 SIMD (Single Instruction Multiple Data)

Σε ορισμένες περιπτώσεις όπως παρατηρήθηκε, δεν είναι απαραίτητο να γίνουν διαφορετικές ενέργειες για να επιλυθεί ένα πρόβλημα. Για παράδειγμα όταν θέλουμε να πολλαπλασιάσουμε έναν πίνακα με έναν αριθμό στην πράξη πολλαπλασιάζουμε κάθε στοιχείο του πίνακα με αυτόν τον αριθμό. Δηλαδή για διαφορετικά δεδομένα εκτελώντας την ίδια πράξη καταλήγουμε στο αποτέλεσμα. Αυτό δεν απαιτεί πλήρεις πυρήνες για να επιλυθεί σαν πρόβλημα. Είναι πολύ πιο αποδοτικό να χρησιμοποιήσουμε FPUs (Floating Point Units) και λιγότερο σύνθετους ελεγκτές (controllers) που θα τις συντονίζουν. Η αρχιτεκτονική αυτή εστιάζει στην μεγιστοποίηση του αριθμού των δεδομένων που μπορούν να επεξεργαστούν ταυτόχρονα και αποτελεί την βάση των σύγχρονων καρτών γραφικών GPUs.

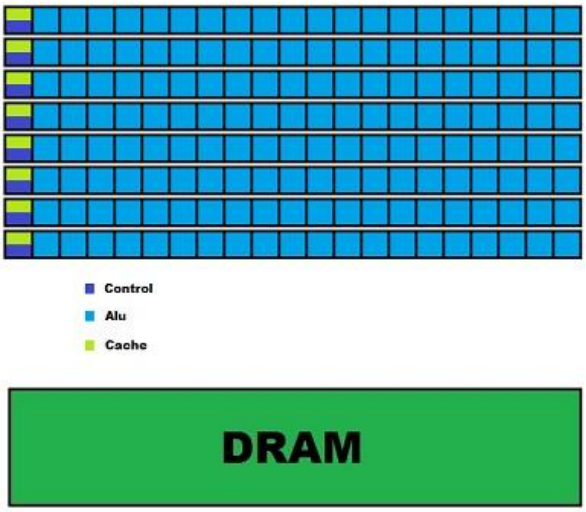
Οι κάρτες γραφικών διαθέτουν ξεχωριστή μνήμη για γρηγορότερη πρόσβαση στα δεδομένα και δυνατότητα μεταφοράς μεγαλύτερου όγκου δεδομένων ανά πάσα χρονική στιγμή ώστε να ελαχιστοποιηθεί ο χρόνος αδράνειας τον πόρων τους λόγω χρόνου απόκρισης μνήμης. Οι κάρτες γραφικών έχουν συνήθως μερικές χιλιάδες FPUs ώστε να μπορούν να υπολογίσουν ταχύτατα την τιμή των pixels ανά πάσα χρονική στιγμή. Πρόσφατα, μέσω επεκτάσεων γλωσσών όπως CUDA για τις

κάρτες της NVidia και OpenCL για άλλες, αυτή η υπολογιστική ισχύς μπορεί να χρησιμοποιηθεί ευκολότερα και για επίλυση άλλων αριθμητικών προβλημάτων. Αντιπρόσωποι της συγκεκριμένης κατηγορίας είναι ιδανικοί για επίλυση επιστημονικών προβλημάτων.

Φυσικά υπάρχουν και μεονεκτήματα. Για αρχή η συγκεκριμένη αρχιτεκτονική δεν μπορεί να εκτελέσει διαφορετικές εντολές σε διαφορετικές FPUs. Λόγω των πιο απλών controller που διαθέτει , χωρίς συντονισμό από κάποιον εξωτερικό παράγοντα π.χ. CPU δεν είναι δυνατόν να λειτουργήσει αποτελεσματικά. Οι ALU όπως φαίνονται στην εικόνα 4.2 εκτελούν ταυτόχρονα όλες την ίδια αριθμητική πράξη σε διαφορετικό σετ δεδομένων (εξ ου και Multiple Data).

Ένα ακόμη μειονέκτημά τους είναι λόγω του απλούστερου ελεγκτή που χρησιμοποιούν, δεν διαθέτουν branch predictors και απαιτείται από την CPU συνήθως να παγώσει όλα τα νήματα που παίρνουν την μια κατεύθυνση σε ένα branch μέχρι να υπολογιστούν τα υπόλοιπα και στην συνέχεια να υπολογιστούν αυτά ενώ τα υπόλοιπα περιμένουν.

Τέλος οι GPUs δουλεύουν σε χαμηλότερες συχνότητες από αυτές στις οποίες δουλεύουν οι CPUs.



Εικ 4.2 Αρχιτεκτονική SIMD

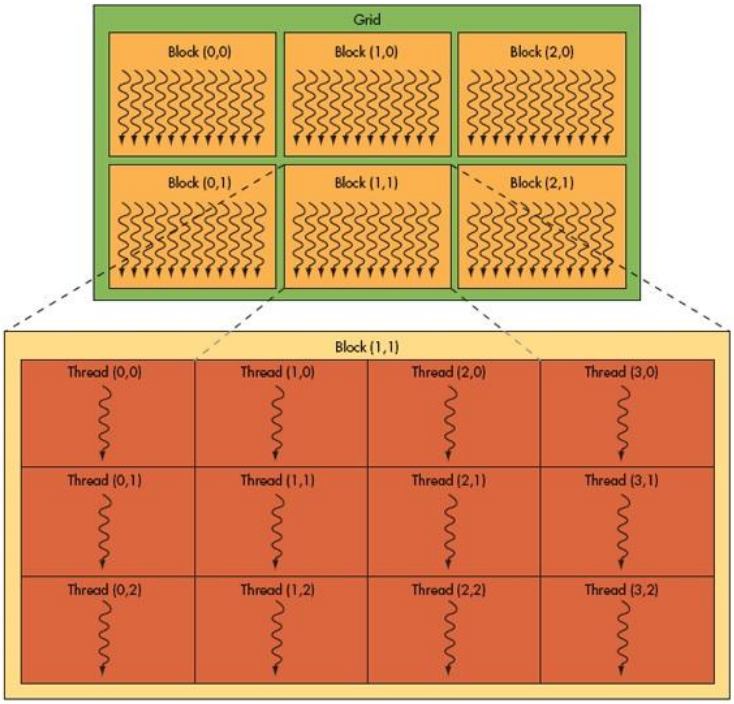
4.2 Διαφορές Παράλληλου Προγραμματισμού με χρήση GPU

Από την παραπάνω περιγραφή των αρχιτεκτονικών καταλαβαίνει κανείς ότι ο προγραμματισμός για παράλληλα συστήματα πρέπει να είναι διαφορετικός από τον σειριακό ώστε να εκμεταλλευτούμε τις δυνατότητες που μας παρέχονται. Παρακάτω αναλύουμε τον τρόπο λειτουργίας CUDA πολύ συνοπτικά ώστε να μπορέσουμε να δώσουμε παραδείγματα μέσω αυτού κατά την παρουσίαση του προτεινόμενου αλγορίθμου.

4.2.1 CUDA

Το μοντέλο προγραμματισμού CUDA επιτρέπει τον ταυτόχρονο προγραμματισμό τόσο της CPU όσο και της GPU. Η CPU εκτελεί τις εντολές ενός προγράμματος μέχρι την στιγμή στην οποία το πρόγραμμα δηλώνει ότι απαιτείται η δυνατότητα παραλληλίας που παρέχεται από την GPU. Η CPU ανά πάσα στιγμή μπορεί να δεσμεύσει μνήμη στην μνήμη της κάρτας γραφικών, να αντιγράψει δεδομένα από την μνήμη του συστήματος στην μνήμη της κάρτας και αντίστροφα καθώς επίσης και να ελευθερώσει δεσμευμένη μνήμη στην κάρτα γραφικών. Επίσης η CPU μπορεί ανά πάσα στιγμή να απαιτήσει από την κάρτα γραφικών να εκτελέσει κάποιον καθορισμένο αριθμό νημάτων στα οποία θα εκτελεστεί η ίδια ακολουθία εντολών όπως ορίζεται στο πρόγραμμα ως kernel.

Ένας kernel έχει την ίδια δομή με μια συνάρτηση στο πρόγραμμα με εξαίρεση το ότι εκτελείται στην κάρτα γραφικών και απαιτεί να οριστούν κάποιες επιπλέον παράμετροι (Grid, Block,Threads). Το σετ των δεδομένων εκτελείται σαν ένα Grid από Blocks, και κάθε μπλοκ εκτελεί ένα συγκεκριμένο αριθμό Threads όπως φαίνεται στην εικόνα 4.3.



Εικ 4.3. Ιεραρχία Grid->Block->Thread

Οι κάρτες γραφικών αποτελούνται από πολλαπλούς πολυεπεξεργαστές τους οποίους για συντομογραφία θα τους αναφέρουμε ως SMs (Streaming Multiprocessors). Κάθε SM αποτελείται από ένα απλό ελεγκτή, κάποια μνήμη και πάρα πολλές ALU (κάθε γραμμή της εικόνας 4.2). Κάθε ALU μπορεί να παίζει τον ρόλο ενός von Neumann επεξεργαστή ή να μην χρησιμοποιείται. Όλες οι ALU κάθε SM χρησιμοποιούν την κοινή μνήμη που διαθέτει ο SM ενώ διαθέτουν και ξεχωριστούς καταχωρητές (registers). Κάθε μπλοκ που απαιτείται να εκτελεστεί, το αναλαμβάνει ένας SM όπως ορίζεται από το υλικό της κάρτας γραφικών και τα threads εκτελούνται στις ALU. Επιπλέον σε κάθε SM μπορούν να εκτελούνται ανά κύκλο ρολογιού μέχρι τρία warps όπου ως warp ορίζεται ένα σύνολο 32 threads.Όταν κάποιος SM ολοκληρώσει το block στο οποίο δούλευε τότε αναλαμβάνει κάποιο νέο αν υπάρχει διαθέσιμο.

Όταν οι υπολογισμοί που ανατέθηκαν στην GPU φτάσουν σε πέρας, ο έλεγχος επιστρέφει στην CPU η οποία συνεχίζει την εκτέλεση του εναπομείναντος προγράμματος.

4.2.2 Περιορισμοί και Τρόποι βελτιστοποίησης.

Από το παραπάνω μοντέλο προκύπτουν ορισμένοι περιορισμοί. Κατ' αρχάς τα threads εκτελούν όλα τις ίδιες εντολές. Αυτό συνεπάγεται ότι αλγόριθμοι σαν αυτούς που ανήκουν στην κατηγορία των υποθετικών κινήσεων όπως αναφέρονται στο προηγούμενο κεφάλαιο, είναι αδύνατον να υλοποιηθούν. Η μόνη κοινή μνήμη που διαθέτουν τα threads είναι αυτή που υπάρχει σε κάθε SM. Το συμπέρασμα αυτού είναι η αδυναμία επικοινωνίας μεταξύ threads που ανήκουν σε διαφορετικά μπλοκ κατά την εκτέλεση του kernel παρά μόνο με χρήση της global μνήμης της κάρτας γραφικών και μόνο με χρήση συγχρονισμού τους. Η διαδικασία της ανάγνωσης και της ενημέρωσης της global μνήμης είναι αργή και καλό είναι να αποφεύγεται όσο είναι δυνατόν.

Όπως αναφέραμε προηγουμένως, τα threads εκτελούνται σε διαφορετικές ALU του ίδιου SM. Είναι φυσιολογικό λοιπόν η μία ALU να διαφέρει από την άλλη σε ταχύτητα λειτουργίας πρώτα και κύρια εξαιτίας του υλικού. Έτσι κάποιο thread που ξεκίνησε πρώτο μπορεί να τελειώσει μετά από άλλα που ξεκίνησαν αργότερα. Ο περιορισμός που προκύπτει είναι ότι τα δεδομένα που διαβάζονται αλλά και που γράφονται από κάθε thread πρέπει να είναι ανεξάρτητα μεταξύ τους. Αν τα δεδομένα δεν είναι ανεξάρτητα μεταξύ τους τότε παρουσιάζονται Read after Write (RAW) και Write after Write(WAW) προβλήματα. Για τις περιπτώσεις RAW μπορεί να χρησιμοποιηθεί συγχρονισμός όλων των threads μεταξύ της εντολής ανάγνωσης και της εντολής εγγραφής αλλά αυτό είναι μια πολύ αργή διαδικασία η οποία περιορίζει σημαντικά το κέρδος της παράλληλης επεξεργασίας. Το ίδιο συμβαίνει

και για περιπτώσεις WAW οι οποίες σε μερικές κάρτες γραφικών μπορούν να εκτελέσουν συγκεκριμένες προκαθορισμένες από το API atomic μεθόδους αλλά με σημαντικό κόστος σε χρόνο εκτέλεσης.

Η έλλειψη κάποιου μηχανισμού πρόβλεψης διακλάδωσης καθιστά συχνά τα σημεία αυτά στο πρόγραμμα λιγότερο αποτελεσματικά ακόμη και από τα αντίστοιχα που εκτελούνται στην CPU. Αυτό συμβαίνει διότι κατά την εκτέλεσή τους τα threads κάθε block, αρχικά συνεχίζουν την λειτουργία τους όσα ακολουθούν το ένα μονοπάτι και μόλις ολοκληρώσουν την εργασία τους, ο έλεγχος περνάει στην CPU που τα παγώνει και ορίζει ότι πρέπει να εκτελεστούν αυτά που παίρνουν το άλλο μονοπάτι και τα συγχρονίζει για να συνεχιστεί το πρόγραμμα.

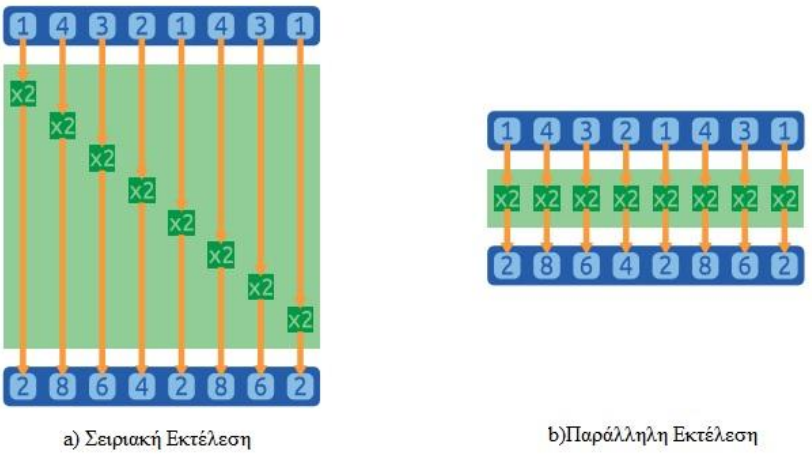
Ένας ακόμη τρόπος να βελτιωθεί το πρόγραμμα είναι η προσπάθεια να εκμεταλλευτεί η τοπικότητα των στοιχείων στην μνήμη ώστε να υπάρχει μεγαλύτερη ευστοχία της μνήμης cache. Είναι καλή πρακτική ειδικότερα στον προγραμματισμό με χρήση GPU να σχεδιάζουμε δομές και να εκτελούμε τις πράξεις κατάλληλα ώστε να ελαχιστοποιούμε τον αριθμό αναζήτησης στοιχείων από την μνήμη.

4.3 Προγραμματιστικά μοτίβα

Οι ανωτέρω περιορισμοί ώθησαν τους προγραμματιστές να αναπτύξουν προγραμματιστικά μοτίβα που μεγιστοποιούν την παραλληλία. Έτσι και η NVidia στην επέκταση γλώσσας C/C++ που έχει κατασκευάσει, έχει αυτοματοποιήσει ορισμένα από αυτά ώστε να λειτουργούν βέλτιστα στις κάρτες γραφικών της. Τα μοτίβα τα οποία είναι απαραίτητα για την υλοποίηση της εργασίας επεξηγούνται ακολούθως ενώ είναι διαθέσιμα για τους προγραμματιστές ως βιβλιοθήκη για την γλώσσα C++ με το όνομα Thrust [49].

4.3.1 Map

Πρόκειται ίσως για την πιο απλή μορφή παραλληλίας με χρήση πολλαπλών δεδομένων. Σε αυτό το μοτίβο, μια η περισσότερες πράξεις εκτελούνται στα δεδομένα μιας λίστας εισόδου και παράγεται έξοδος ίδιου μεγέθους με την λίστα εισόδου. Η στόχος των στοιχείων εξόδου μπορεί κάλλιστα να είναι η λίστα εισόδου. Για παράδειγμα, έστω ότι έχουμε ένα διάνυσμα n στοιχείων του οποίου θέλουμε να πολλαπλασιάσουμε το κάθε στοιχείο με τον αριθμό δύο και το τελικό αποτέλεσμα να αποθηκευτεί στην εκάστοτε θέση του διανύσματος η οποία χρησιμοποιείται σαν είσοδος. Η εικόνα 4.4 παρουσιάζει την σειριακή εκτέλεση και την παράλληλη εκτέλεση ενός τέτοιου μοτίβου σε ένα διάνυσμα 8 στοιχείων. Κάθε γραμμή αντιπροσωπεύει μία μονάδα χρόνου.



Εικ 4.4 α) Σειριακή εκτέλεση b) Παράλληλη εκτέλεση με χρήση του map προγραμματιστικού μοτίβου

4.3.2 Scan

Μια πολύ χρήσιμη λειτουργία είναι αυτή του scan. Εφαρμόζοντας αυτό το μοτίβο σε ένα διάνυσμα προκύπτει ένα νέο διάνυσμα του οποίο το κάθε στοιχείο αντιστοιχεί στο άθροισμα του στοιχείου με ίδιο δείκτη και των όσων βρίσκονται αριστερά του. Αυτή το μοτίβο δεν συναντάται στον σειριακό προγραμματισμό. Για παράδειγμα έστω το ακόλουθο διάνυσμα:

7 0 11 4 6 2 3 1

Μετά την εφαρμογή του το διάνυσμα που προκύπτει είναι το ακόλουθο...

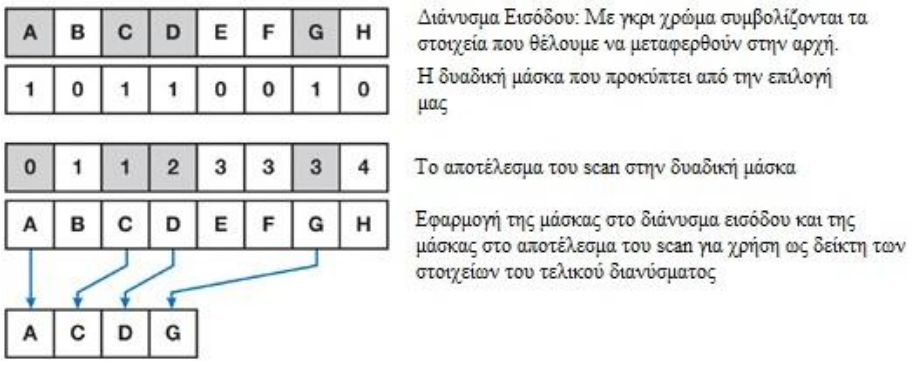
7 7 18 22 28 30 33 34

Το συγκεκριμένο μοτίβο χρησιμοποιείται εκτενώς κατά την εκτέλεση των παρακάτω μοτίβων παράλληλου προγραμματισμού.

4.3.3 Pack/Split

Λόγω έλλειψης μηχανισμού πρόβλεψης διακλάδωσης όπως είδαμε, η εκτέλεση ενός block μπορεί να είναι αργή λόγω του ότι πρώτα εκτελούνται τα threads που παίρνουν την μια κατεύθυνση και συγχρονίζονται και μετά τα υπόλοιπα τα οποία συγχρονίζονται επίσης. Στο σύνολο του προγράμματός μας επιθυμούμε για λόγους επιτάχυνσης να έχουμε όσο το δυνατόν λιγότερα blocks τα οποία να έχουν threads που ακολουθούν και τις δύο κατευθύνσεις. Γι αυτόν τον λόγο είναι χρήσιμο το μοτίβο pack/split. Σε αυτό το μοτίβο εφαρμόζοντας μια δυαδική μάσκα μπορούμε να μετακινήσουμε όλα τα δεδομένα που πληρούν μια προϋπόθεση σε μια διάταξη. Πιο συγκεκριμένα εφαρμόζουμε στην δυαδική μάσκα το μοτίβο του scan. Στην συνέχεια εφαρμόζουμε την μάσκα για να ορίσουμε ποιά στοιχεία του διανύσματος εισόδου θέλουμε να μετακινηθούν στην αρχή και ποιά όχι ενώ εφαρμόζουμε την μάσκα πάνω στο αποτέλεσμα του scan για να ορίσουμε την θέση στην οποία θα πρέπει να πάει κάθε στοιχείο. Το αποτέλεσμα είναι ένα διάνυσμα μήκους ίσου με του αριθμού των στοιχείων που μεταφέρονται και μπορεί εύκολα να υπολογιστεί από το άθροισμα των στοιχείων της δυαδικής μάσκας.

Αυτή η διαδικασία αποτελεί το pack.



Εικ 4.5 Προγραμματιστικό μοτίβο pack.

Αν μετά το πέρας του pack εφαρμόσουμε την NOT δυαδική μάσκα και προσθέσουμε στο αποτέλεσμα του scan της το άθροισμα των στοιχείων της αρχικής δυαδικής μάσκας τότε προκύπτει ένα διάνυσμα ίσου μήκους με το διάνυσμα εισόδου στο οποίο τα στοιχεία έχουν ταξινομηθεί στα στοιχεία που έχουν επιλεγεί από την μάσκα και στα υπόλοιπα.

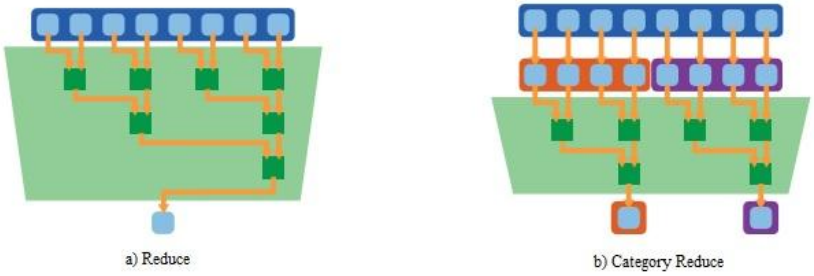
4.3.4 Reduce

Με αυτό το προγραμματιστικό μοτίβο επιτυγχάνουμε να εκτελέσουμε στο σύνολο ενός διανύσματος μια συγκεκριμένη πράξη με τρόπο γρηγορότερο από ότι στο σειριακό προγραμματισμό. Αυτή η πράξη μπορεί να είναι η εύρεση του αθροίσματος όλων των στοιχείων του, η εύρεση του

μεγίστου στοιχείου, ο πολλαπλασιασμός τους και λοιπά. Για παράδειγμα εάν χρησιμοποιούσαμε το reduce στο διάνυσμα [7 0 11 4 6 2 3 1] για την εύρεση του αθροίσματος των στοιχείων του, το αποτέλεσμα θα έβγαινε ως εξής:

7+	0	11+	4	6+	2	3+	1
	7	+	**15**		**8**	+	**4**
			22		+		**12**
							34

Στην παραλλαγή του συγκεκριμένου μοτίβου Category Reduce η πράξη εφαρμόζεται επαναληπτικά στο διάνυσμα εισόδου με την βασική διαφορά ότι τα δεδομένα του διανύσματος ανήκουν σε προκαθορισμένες κατηγορίες. Σε αυτήν την περίπτωση η πράξη εφαρμόζεται για κάθε κατηγορία και τα αποτελέσματα που προκύπτουν είναι ανά κατηγορία και ίσα με αυτές.



Εικ 4.6 Reduce και Category Reduce

4.4 Περίληψη.

Σε αυτό το κεφάλαιο αρχικά εξετάστηκαν οι τρόποι λειτουργίας των υπολογιστικών συστημάτων στο παρελθόν και τώρα, καθώς και οι συνθήκες οι οποίες οδήγησαν στην τελική επικράτηση της παράλληλης επεξεργασίας.

Στην συνέχεια αναλύθηκαν οι δύο κατηγορίες παράλληλων συστημάτων μαζί με τα πλεονεκτήματα και τα μειονεκτήματα που παρέχει η κάθε μια. Δόθηκε έμφαση στα SIMD συστήματα και στην χρήση των καρτών γραφικών GPUs και εξετάστηκαν ορισμένοι περιορισμοί που απορρέουν από την αρχιτεκτονική τους. Επίσης επισημάνθηκαν και ορισμένες λειτουργίες οι οποίες αν και μπορούν να εκτελεστούν καλό θα είναι να αποφεύγονται διότι αυξάνουν τον χρόνο εκτέλεσης.

Τέλος, έγινε μια σύντομη περιγραφή κάποιων προγραμματιστικών μοτίβων τα οποία υπάρχουν ήδη υλοποιημένα στην βιβλιοθήκη Thrust για C++ της NVidia με σκοπό να γίνει χρήση τους στο επόμενο κεφάλαιο στην πρότασή μας.

5. Παράλληλη τοποθέτηση με χρήση κάρτας γραφικών

Σε αυτό το κεφάλαιο, παρουσιάζουμε ένα μοντέλο για το στάδιο της τοποθέτησης το οποίο στηρίζεται σε GPU. Το μοντέλο αυτό ανήκει στην κατηγορία των αλγορίθμων τεμαχισμένης τοποθέτησης όπως αναλύθηκαν σαν κατηγορία στο τρίτο κεφάλαιο και στοχεύει στην ελαχιστοποίηση του μήκους αγωγών. Περιγράφουμε αναλυτικά τα στάδια λειτουργίας του και εξηγούμε με ποιόν τρόπο εξασφαλίζεται η τήρηση των περιορισμών που προκύπτουν λόγω της παράλληλης επεξεργασίας. Η συγκεκριμένη μέθοδος παρουσιάζει πολύ υψηλή παραλληλία η οποία αυξάνεται σε μεγάλο βαθμό όσο μεγαλύτερο είναι το κύκλωμα το οποίο προσπαθούμε να περιγράψουμε σε μια FPGA.

Τα στάδια που ακολουθούνται είναι :

1. Εκτέλεση InitialPlacement() μεθόδου του VPR
2. Δημιουργία πίνακα γειτονικότητας με χρήση του αρχείου netlist.net
3. Αξιολόγηση τρέχουσας τοποθέτησης και ενημέρωση διανύσματος GC
4. Τερματισμός αν το κριτήριο εξόδου έχει ικανοποιηθεί ως συνθήκη
5. Ενημέρωση T,Rlim με χρήση του αριθμού κινήσεων που εκτελέστηκαν και έγιναν αποδεκτές.
6. Εφαρμογή μεταβλητού πλαισίου στον χώρο τοποθέτησης
7. Εύρεση ανεξάρτητων μεταξύ τους κινήσεων με αυξημένη τυχαιότητα
8. Αξιολόγηση της κάθε κίνησης
9. Εφαρμογή των κινήσεων που κρίθηκαν αποδεκτές
10. Επιστροφή στο 3
11. Τοποθέτηση I/O pads

Τα στάδια τρία έως και δέκα επαναλαμβάνονται μέχρι να ικανοποιηθεί η συνθήκη εξόδου. Στην επόμενη παράγραφο περιγράφουμε πως επιτυγχάνεται η ανεξαρτησία των κινήσεων. Αμέσως μετά παρουσιάζεται το πλαίσιο επίδρασης πάνω στον χώρο τοποθέτησης καθώς επίσης και ο τρόπος με τον οποίο μεγιστοποιείται η μετακίνηση στοιχείων μεταξύ των blocks ενώ στην ενότητα 5.6 παρουσιάζουμε τον τρόπο αξιολόγησης της κάθε κίνησης αλλά και τον τρόπο υπολογισμού του συνολικού κόστους της τοποθέτησης. Οι δομές οι οποίες είναι απαραίτητες για την εκτέλεση της συγκεκριμένης μεθόδου περιγράφονται στην παράγραφο του αντίστοιχου σταδίου που τις απαιτεί για την λειτουργία του.

5.1 Αναπαράσταση CLBs και χώρου τοποθέτησης

Η αναπαράσταση που θα χρησιμοποιήσουμε για να περιγράψουμε τον αλγόριθμό μας είναι ένας δισδιάστατος πίνακας. Κάθε κελί του πίνακα αντιστοιχεί σε μία νησίδα (isle) της FPGA και είναι ικανό να χωρέσει ένα και μόνο CLB όπως αυτό περιγράφεται από το αρχείο τοποθέτησης που προκύπτει μετά από την εκτέλεση της μεθόδου InitialPlacement() του VPR. Ένα αρχείο placement έχει την ακόλουθη μορφή:

Netlist file: xor5.net Architecture file: sample.arch
Array size: 2 x 2 logic blocks

#block name	x	y	subblk	block number
#----------	--	--	------	----------
a	0	1	0	#0 -- NB: block number is a comment.
b	1	0	0	#1
c	0	2	1	#2
d	1	3	0	#3
e	1	3	1	#4
out:xor5	0	2	0	#5
xor5	1	2	0	#6
[1]	1	1	0	#7

Στην πρώτη γραμμή αναφέρονται τα ονόματα των αρχείων netlist και architecture για την συγκεκριμένη υλοποίηση. Αμέσως μετά ακολουθεί το μέγεθος του πίνακα στον οποίο τοποθετούνται

τα CLBs με μορφή m x n. Στην παραπάνω περίπτωση αυτός ο πίνακας είναι διαστάσεων 2x2. Στην συνέχεια ακολουθεί μια λίστα με πέντε πεδία.

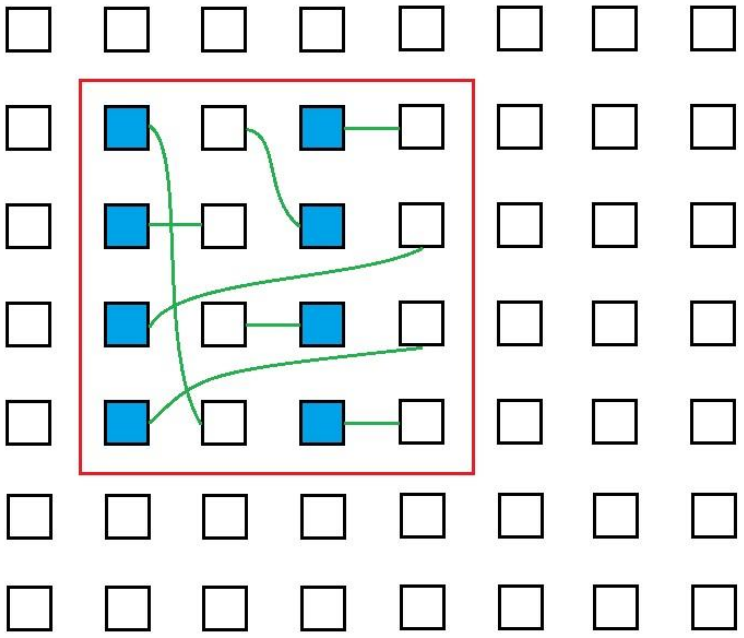
- block name: Σε αυτό το πεδίο βρίσκουμε το όνομα του κάθε CLB όπως αυτό έχει περιγραφεί.
- x : πρόκειται για την οριζόντια θέση του συγκεκριμένου CLB με block name πάνω στον πίνακα
- y : όμοια με το x αλλά για την κατακόρυφη θέση του
- subblk : Όπως περιγράψαμε στην παράγραφο 3.1 στον αρχείο αρχιτεκτονικής είναι απαραίτητο να ορίζεται ο αριθμός των I/O pads που χωράνε κατά μήκος και πλάτος μιας νησίδας. Όταν αυτός ό αριθμός είναι μεγαλύτερος από ένα χρησιμοποιείται το subblk για να ξεχωρίσει τα I/Os που έχουν κατά τα άλλα ίδιες συντεταγμένες.
- block number : πρόκειται για μία στήλη για σχόλια. Τα σχόλια στον συγκεκριμένο τύπο αρχείου αρχίζουν μετά το σύμβολο # το οποίο ισχύει μέχρι την χρήση κάποιου newline χαρακτήρα.
- Όλα τα blocks με θέση x ίση με μηδέν ή m+1 είναι υποχρεωτικά I/Os. Αντίστοιχα όλα τα blocks με θέση y ίση με μηδέν ή n+1 είναι επίσης I/Os. Όλα τα ενδιάμεσα είναι υποχρεωτικά είτε κενά ή καλυμμένα με CLBs.

Από το παραπάνω αρχείο μπορούμε να κατασκευάσουμε έναν πίνακα που αναπαριστά τα εξαρτήματα του κυκλώματος προς υλοποίηση πάνω στην FPGA και με βάση αυτόν τον πίνακα θα προτείνονται οι κινήσεις.

5.2 Εύρεση ανεξάρτητων κινήσεων

Για την επίτευξη της παραλληλίας χωρίς σφάλματα είναι απαραίτητο οι κινήσεις που θα εξετάζονται να είναι ανεξάρτητες μεταξύ τους. Δηλαδή κανένα κελί του πίνακα δεν πρέπει να παρουσιάζεται σε παραπάνω από ένα νήματα προς επεξεργασία κίνησης. Ορισμένες προσεγγίσεις αλγορίθμων τεμαχισμένης τοποθέτησης χρησιμοποιούν προκαθορισμένα μοτίβα. Αυτό περιορίζει μερικώς την ποιότητα του τελικού κυκλώματος. Η μέθοδος που προτείνεται σε αυτή την εργασία σκοπό έχει να μεγιστοποιήσει την τυχαιότητα.

Στην τεμαχισμένη τοποθέτηση αρχικά γίνεται υποδιαίρεση του χώρου εφαρμογής και κάθε υπόχωρος που προκύπτει ανατίθεται σε έναν πηρύνα. Εξετάζοντας την πρότασή μας σε χώρο ίσο με ένα block κατορθώνουμε να εξασφαλίσουμε την ανεξαρτησία των κινήσεων. Αν αυτό εξασφαλιστεί σε ένα block τότε αυτό είναι το ίδιο εύκολο να ισχύει και για την συνολική τοποθέτηση καθώς ο συνολικός χώρος επεξεργασίας είναι υπερσύνολο του ενός block.



Εικ 5.1 Παραγωγή ανεξάρτητων κινήσεων στο επισημασμένο 4x4 block.

Στην εικόνα 5.1 το σύνολο των τετραγώνων αντιπροσωπεύουν τον χώρο τοποθέτησης. Έστω ο υπόχωρός του που ορίζεται από την κόκκινη γραμμή. Ο υπόχωρος αυτός αποτελείται από 16 κελιά καθώς 4x4. Κάθε στοιχείο του υποχώρου μπορεί είτε να μην παίρνει μέρος σε κάποια κίνηση είτε να αποτελεί αφετηρία μιας κίνησης ή τελικό προορισμό. Η μέγιστη δυνατή παραλληλία παρουσιάζεται όταν όλα τα στοιχεία του υποχώρου παίρνουν μέρος σε μια και μόνο κίνηση. Άρα η ιδανική αντιμετώπιση του θέματος θα περιλαμβάνει οκτώ ανταλλαγές στοιχείων μεταξύ ζευγών κελιών όπου κάθε ανταλλαγή θα εκτελείται σε ένα και μόνο νήμα. Οι πιθανές περιπτώσεις όταν χρησιμοποιείται αυτή η διαδικασία είναι η ακόλουθες.

1. Τα δύο κελιά τα οποία ανταλλάσουν θέσεις είναι και τα δύο κενά. Σε αυτή την περίπτωση οι κινήσεις αγνοούνται.
2. Το ένα εκ των δύο κελιών είναι κενό. Σε αυτήν την περίπτωση το νήμα αξιολογεί την κίνηση του μη κενού μπλοκ και την αποδέχεται ή την απορρίπτει.
3. Τα δύο κελιά έχουν κάποιο τμήμα λογικής το καθένα. Το νήμα αξιολογεί και τις δύο κινήσεις και τις αποδέχεται ή τις απορρίπτει μαζί ανάλογα αν η κίνηση οδήγησε σε μικρότερο άθροισμα διαφορών κόστους.

Η παραγωγή των κινήσεων με χρήση της μεθόδου μας ορίζει ότι τα στοιχεία του υποχώρου του πίνακα τοποθέτησης με ζυγό δείκτη στηλών ανταλλάσουν θέση με μοναδικό τρόπο με στοιχεία με μόνο δείκτη. Αρχικά δεσμεύεται ένα διάνυσμα V με μήκος ίσο με τον συνολικό αριθμό νημάτων που απαιτούνται για την κάλυψη του υποχώρου στον οποίο εφαρμόζεται η μέθοδος. Στην περίπτωση της εικόνας 5.1, το block απαιτείται να έχει διαστάσεις 4x2. στην συνέχεια κάθε ένα από αυτά τα νήματα που προκύπτουν από αυτό το block γράφει στην θέση του διανύσματος που του αντιστοιχεί το αποτέλεσμα της ακόλουθης εξίσωσης.

$$V[\text{threadIdx.y}*\text{blockDim.x}+\text{threadIdx.x}] = 2\ (\text{threadIdx.y}*\text{blockDim.x}+\text{threadIdx.x})+1$$

όπου threadIdx.x και threadIdx.y είναι δύο μοναδικοί καταχωρητές για κάθε νήμα όπως εκφράζονται από την βιβλιοθήκη cuda.c και το blockDim.x εκφράζει την διάσταση x σε νήματα του block. Το παραπάνω αρχικοποιεί το διάνυσμα σε αύξουσα σειρά των μονών δεικτών του πίνακα που αντιπροσωπεύει τον υπόχωρο. Στην προκειμένη περίπτωση του παραδείγματος το διάνυσμα που προκύπτει είναι το ακόλουθο:

[1 3 5 7 9 11 13 15]

Στην συνέχεια κάθε νήμα παράγει έναν τυχαίο αριθμό ο οποίος έχει εύρος τιμών από μηδέν έως δύο και αποθηκεύεται σε καταχωρητή ξεχωριστό για κάθε νήμα μετά από επεξεργασία ως εξής:

$$p = (\text{threadIdx.x}\%2)*(\text{rand}()\%3)$$

Η τιμή λοιπόν του p για κάθε νήμα με δείκτη διάστασης x πολλαπλάσιο του δύο είναι μηδέν ενώ για κάθε δείκτη x περιττό η τιμή είναι οτιδήποτε από μηδέν έως δύο. Ο λόγος που δεν χρησιμοποιείται στο υπολογισμό του p η διάσταση y είναι γιατί το μέγεθος των block που θα χρησιμοποιήσουμε, στοχεύοντας στην μέγιστη παραλληλία και δυνατότητα μετακίνησης στοιχείων έχουν συγκεκριμένες διαστάσεις όπου το blockDim.x είναι πάντα ζυγός αριθμός. Κατά συνέπεια το αποτέλεσμα (threadIdx.y*blockDim.x)%2 θα είναι πάντα μηδέν. Επίσης, ορίζονται δύο ακόμη τυχαίες μεταβλητές d1 και d2 με εύρος από 0 έως το μισό μήκος του διανύσματος V μείον ένα.

Στην συνέχεια χρησιμοποιούνται δύο if-statements και κάθε μία ακολουθείται από τον συγχρονισμό όλων των νημάτων.

```
if(p==1)
{
        int temp;
        temp=V[(ty*blockDim.x+tx +2d1)%lengthV+ 1];
        V[(ty*blockDim.x+tx +2d1)%lengthV+ 1]=V [ty*blockDim.x+tx];
        V[ty*blockDim.x+tx]=temp;
}
__syncthreads();
```

Στο παραπάνω απόσπασμα κώδικα για συντομογραφία χρησιμοποιούμε tx και ty αντί για threadIdx.x και threadIdx.y αντίστοιχα. Σε αυτό το τμήμα, όλα τα στοιχεία του διανύσματος V με

άρτιο δείκτη και τιμή p στο νήμα που τους αντιστοιχεί ίση με 1 ανταλλάσουν θέση με κάποιο από τα στοιχεία με μονό δείκτη μετατοπισμένα δεξιά κατά δύο φορές d1 συν ένα με σκοπό να καταλήξουμε σε στοιχείο με μονό δείκτη και μετά τα νήματα συγχρονίζονται ώστε να ενημερωθούν όλες οι τιμές του διανύσματος V πριν συνεχίσουμε. Εφαρμόζουμε το ίδιο και για d2 για τα στοιχεία του διανύσματος με τιμή p ίση με δύο αλλά αυτή τη φορά η μετατόπιση γίνεται προς τα αριστερά. Τα στοιχεία με p μηδέν δεν μετακινούνται. Έτσι προκύπτει το τελικό διάνυσμα V το οποίο έχει ανακατεμένους τυχαία όλους τους μονούς αριθμούς.

Αν χρησιμοποιηθούν τα στοιχεία του συγκεκριμένου διανύσματος σαν δείκτης μπορούν να προκύψουν ανεξάρτητες κινήσεις μεταξύ ενός ζυγού και ενός μονού στοιχείου του πίνακα εφαρμογής. Παραδείγματος χάρη έστω το παρακάτω διάνυσμα V με p όπως δίνεται ακολούθως και d1 = 2, d2 = 3. Επίσης έστω ο πίνακας εφαρμογής A 4x4

[1 3 5 7 9 11 13 15]

μετά τα rand() τα p είναι τα ακόλουθα για κάθε νήμα

[1 0 2 0 0 0 1 0]

στην συνέχεια για d1 = 2

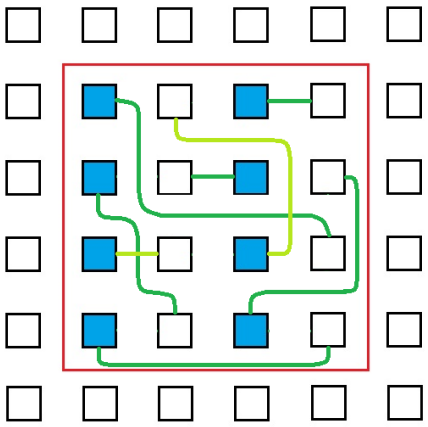
[11 3 5 13 9 1 7 15]

και για d2= 3

[11 3 13 5 9 1 7 15]

Οι προτεινόμενες κινήσεις θα είναι

A[0][0] <---> A[11/4][11%4] == A[2][3]
A[0][2] <---> A[3/4][3%4] == A[0][3]
A[1][0] <---> A[13/4][13%4] == A[3][1]
A[1][2] <---> A[5/4][5%4] == A[1][1]
A[2][0] <---> A[9/4][9%4] == A[2][1]
A[2][2] <---> A[1/4][1%4] == A[0][1]
A[3][0] <---> A[7/4][7%4] == A[1][3]
A[3][2] <---> A[15/4][15%4] == A[3][3]



Εικ 5.2 Αναπαράσταση παραδείγματος

5.3 Πλαίσιο εφαρμογής

Ένας πολύ σημαντικός παράγοντας όσον αφορά την ποιότητα του τελικού κυκλώματος είναι ο χώρος στον οποίο δραστηριοποιείται κάθε αλγόριθμος. Στην περίπτωση του δικού μας αλγορίθμου η

μέθοδος παραγωγής κινήσεων που περιγράφηκε νωρίτερα ισχύει μόνο για τα CLBs ενώ αγνοεί εντελώς τα I/O pads. Τα I/O τμήματα τοποθετούνται σειριακά καθότι είναι πολύ λίγα σε αριθμό και επηρεάζουν την ποιότητα του τελικού κυκλώματος σε πολύ μικρότερο βαθμό. Η μέθοδος εύρεσης της κατάλληλης θέσης για το κάθε εξάρτημα εισόδου/εξόδου επεξηγείται αργότερα. Έτσι από το αρχείο τοποθέτησης παίρνουμε μόνο τα στοιχεία που δεν ανήκουν στην πρώτη ή τελευταία γραμμή ή στήλη.

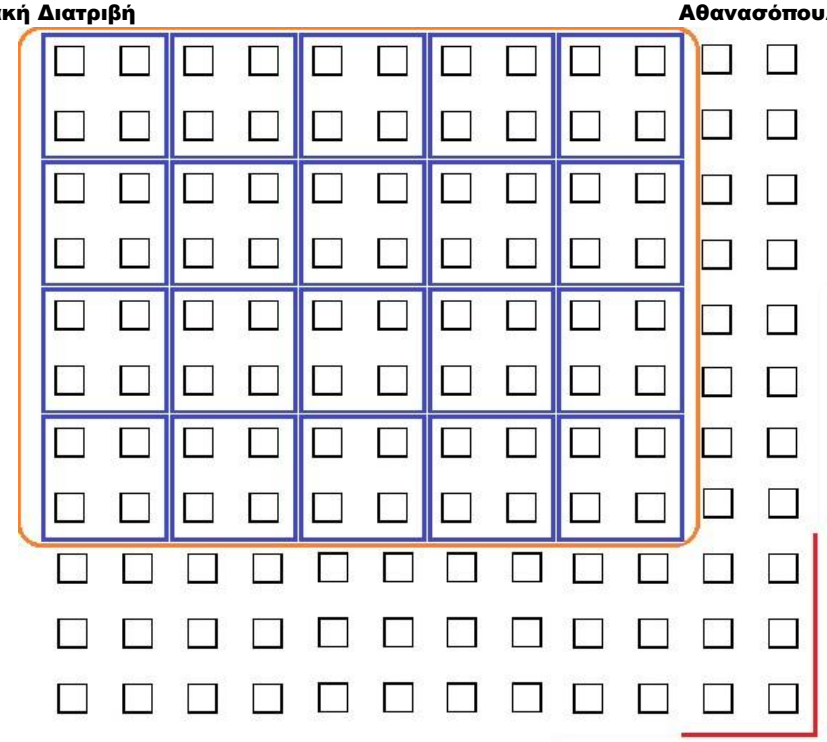
Όπως αναφέραμε στην παράγραφο 4.2.1 είναι απαραίτητο να δοθούν τρείς παράμετροι για να οριοθετήσουμε τον χώρο που ένας kernel θα ενεργήσει, καθώς επίσης και πως. Αν θεωρήσουμε ότι τα blocks είναι σταθερά σε διαστάσεις και θέση σε όλη την διάρκεια του προγράμματος τότε οι πιθανότητες είναι ότι το πρόγραμμά μας θα υλοποιήσει ένα κύκλωμα με κακή ποιότητα. Ο λόγος έγκειται στο γεγονός ότι τα blocks δεν επικοινωνούν μεταξύ τους. Αν λοιπόν οι υπόχωροι που ανατίθενται στους SMs είναι σταθεροί τότε το λογικό συμπέρασμα που προκύπτει είναι ότι δεν θα υπάρχει καθόλου μετακίνηση στοιχείων από τον έναν υπόχωρο σε κάποιον άλλο το οποίο περιορίζει τον βαθμό κατά τον οποίο μπορεί να βελτιωθεί μια τοποθέτηση.

Σε αυτό τον τομέα προτείνουμε δύο μεθόδους με τις οποίες μπορούμε να κάνουμε την τοποθέτηση όσο το δυνατόν πιο κοντά στην σειριακή εκτέλεση από πλευράς ποιότητας. Ανάλογα με το μοντέλο της GPU υπάρχει ένας μέγιστος αριθμός νημάτων που μπορούν να εκτελεστούν ανά block. Η κάρτα η οποία χρησιμοποιήθηκε σαν υλικό αναφοράς για την συγκεκριμένη εργασία είναι η NVidia GTX660. Η συγκεκριμένη κάρτα μπορεί να εκτελεί 1024 νήματα σε κάθε block ενώ ο μέγιστος αριθμός blocks που μπορούν να προγραμματιστούν είναι $2.147.483.647 = 2^{31}-1$.

Kernel παραγωγής ανεξάρτητων κινήσεων: Με τα παραπάνω στοιχεία υπόψη μας προτείνουμε κάθε block να επιδρά το μέγιστο σε ένα χώρο 44x44 δηλαδή σε 1936 κελιά του πίνακα. Ο θεωρητικός μέγιστος αριθμός νημάτων που μπορούν να εκτελεστούν ταυτόχρονα ανά block είναι 1024 και κάθε νήμα χρησιμοποιεί τα δύο κελιά τα οποία εξετάζει αν θα ήταν συμφέρον να αλλάξουν θέση. Συμπεραίνουμε λοιπόν ότι το μέγιστο πλήθος των στοιχείων που μπορούν να επεξεργαστούν ανά μπλοκ είναι 2048. Επιλέγουμε αυτό το μέγιστο μέγεθος block διότι για να λειτουργήσει η μέθοδος της προηγούμενης παραγράφου απαιτείται ζυγός αριθμός κελιών προς επεξεργασία. Στην θεωρία θα μπορούσαμε να χρησιμοποιήσουμε 32x32 νήματα για να φτάσουμε τα 2048 στοιχεία αλλά το block θα σταματούσε να αντιστοιχεί σε τετράγωνο υπόχωρο. Επιλέγουμε να θυσιάσουμε ένα μικρό ποσοστό νημάτων για να έχουμε καλύτερη κατανομή των στοιχείων προς μετακίνηση και εν δυνάμη να έχουμε μεγαλύτερο εύρος κίνησης οριζόντια και κατακόρυφα.

Το τελικό μέγεθος του κάθε μπλοκ επηρεάζεται και από την τιμή R_{lim} πού είδαμε ότι χρησιμοποιείται στο VPR για να ορίσει την μέγιστη απόσταση μεταξύ των στοιχείων προς εξέταση. Όσο αυτή η τιμή είναι μεγαλύτερη από 44 το μέγεθος παραθύρου προς επεξεργασία είναι 44x44. Όταν το R_{lim} ελαττωθεί και γίνει μικρότερο το 44 τότε το μέγεθος παραθύρου γίνεται 32x32, μετά 16x16 και τέλος 8x8 ως ελάχιστο όταν το R_{lim} γίνει μικρότερο από δεκαέξι. Ο λόγος της ύπαρξης αυτού του ελαχίστου είναι ώστε να υπάρχει τουλάχιστον ένα πλήρες warp να εκτελεστεί από κάθε SM και να διατηρηθεί το πλεονέκτημα που μας παρέχεται από την παράλληλη επεξεργασία.

Το δεύτερο τμήμα της πρότασής μας έχει να κάνει με τον αριθμό των blocks που θα προγραμματίζονται να εκτελεστούν και με ποιό στοιχείο ως αφετηρία. Σε αυτό το πρόβλημα επιλέγεται να χρησιμοποιηθεί ένα πλαίσιο του οποίου οι διαστάσεις χρησιμοποιούν ως μέτρο το ένα block. Πιο συγκεκριμένα στον χώρο θα εφαρμόζεται ένα πλέγμα grid διαστάσεων σε x και y ίσο με το ακέραιο τμήμα της διαίρεσης x_{max}/windowSize και y_{max}/windowSize ελαττωμένα κατά ένα με το windowSize να αντιπροσωπεύει την περιοχή στην οποία επιδρά ένα block . Επίσης αυτό το πλαίσιο θα έχει κάθε φορά μια τυχαία οριζόντια μετατόπιση από 0 έως x_{max} - windowSize*gridDim.x και μία ακόμη κατακόρυφη από 0 έως y_{max} - windowSize*gridDim.y. Αυτή η προσέγγιση επιτρέπει την μέγιστη μετακίνηση στοιχείων πάνω στον πίνακα αναπαράστασης ενώ εξασφαλίζει ότι το grid θα παραμένει μονίμως εντός του χώρου τοποθέτησης αποφεύγοντας να χρησιμοποιήσουμε κάποια branch εντολή για να επιβεβαιώσουμε ότι τα νήματα δεν θα βγαίνουν εκτός ορίων. Η εικόνα 5.3 δείχνει ένα τέτοιο παράδειγμα για windowSize 2x2 με μετατοπίσεις ίσες με μηδέν. Το εύρος των μετατοπίσεων φαίνονται με κόκκινη γραμμή στο κάτω δεξιά μέρος της εικόνας ενώ το πλαίσιο είναι επισημασμένο με πορτοκαλί γραμμή.



Εικ 5.3 Μετατοπιζόμενο πλαίσιο επίδρασης. Αριθμός blocks κατά x ίσος με 12/2=6-1= 5 και κατά y 11/2=5-1=4. Μέγιστη μετατόπιση d_x=12-(5x2) = 2 και d_y=11-(4x2)=3

5.4 Ο αλγόριθμος Star

Το VPR χρησιμοποιεί ως κριτήριο κόστους την ημιπερίμετρο του ελαχίστου παραλληλογράμμου που περικλείει τα CLBs του κάθε πλέγματος net. Το συγκεκριμένο κριτήριο είναι λιγότερο αποδοτικό για την παράλληλη επεξεργασία. Αυτό συμβαίνει διότι ανά πάσα στιγμή πρέπει να ελέγχουμε τα όρια του περικλείοντος ορθογωνίου διαδικασία που απαιτεί την χρήση μη συμφερόντων εντολών διακλάδωσης και διότι οποιαδήποτε κίνηση σε ένα πλέγμα δεν μετατοπίζει τα όρια του ελαχίστου παραλληλογράμμου, δεν προκαλεί μεταβολή στο κόστος ακόμη και αν στην πράξη μεταβάλει το μήκος των αγωγών που απαιτούνται.

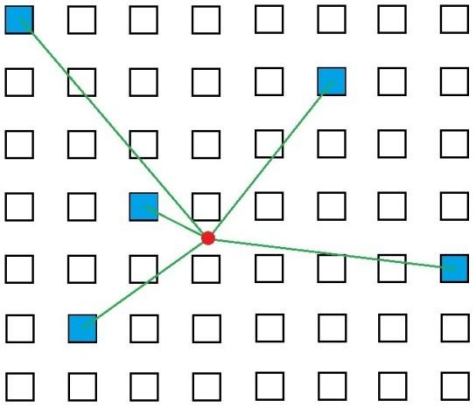
Από την άλλη, υπάρχει ο αλγόριθμος Star που ορίζει ότι κάθε μέλος ενός πλέγματος βρίσκεται σε καλύτερη θέση όσο πιο κοντά βρίσκεται στο κέντρο βάρους. Ως κέντρο βάρους ορίζεται το ιδεατό σημείο που προκύπτει βρίσκοντας τον μέσο όρο της διάστασης x και y αντίστοιχα από όλα τα CLBs μέλη του πλέγματος net.

$$x_{Cj} = \frac{1}{Rank_j} \sum_{\forall i \in Net(j)} x_i$$

$$y_{Cj} = \frac{1}{Rank_j} \sum_{\forall i \in Net(j)} y_i$$

όπου το j είναι κάποιο net και i ένα λογικό μπλοκ από όσα το αποτελούν. Το κόστος υπολογίζεται ως το άθροισμα της ευκλείδιας απόσταση του κάθε σημείου από αυτό το ιδεατό σημείο κέντρου βάρους.

$$Cost_j = \sum_{\forall i \in Net(j)} \left[(x_{Cj} - x_i)^2 + (y_{Cj} - y_i)^2 \right]$$



Εικ 5.4 Αναπαράσταση του Star. Το κόκκινο σημείο είναι το κέντρο βάρους.

5.5 Δομές δεδομένων και συμβολισμοί

Στην παράγραφο 5.1 είδαμε ότι ο τρόπος που χρησιμοποιείται για την αναπαράσταση του προβλήματος είναι ένας δισδιάστατος πίνακας με τα στοιχεία του μερικές φορές να μην έχουν τιμή ενώ άλλες φορές να παίρνουν σαν τιμή τον αριθμό του CLB που περιγράφουν όπως αυτά ορίζονται στο αρχείο '*.net' . Ο πίνακας V.I παρουσιάζει μια τέτοια αναπαράσταση . Στην συνέχεια με την βοήθεια αυτού του πίνακα ορίζουμε δύο διανύσματα $\overrightarrow{dx}$ και $\overrightarrow{dy}$ τα οποία χρησιμοποιούν ως δείκτη των αριθμό του κάθε τερματικού μείον ένα και σαν τιμή την θέση του συγκεκριμένου τερματικού ως προς x και y αντίστοιχα.

X Y	0	1	2	3
0	**5**	**11**	**2**	**7**
1	**3**	**10**	**15**	**9**
2	**4**	**1**	**12**	**14**
3	**13**	**6**	**16**	**8**

Πίνακας V.I

Τα $\overrightarrow{dx}$ και $\overrightarrow{dy}$ που προκύπτουν από τον πίνακα V.I είναι τα ακόλουθα:

$$\overrightarrow{dx} = \{\ 1,\ 2,\ 0,\ 0,\ 0,\ 1,\ 3,\ 3,\ 3,\ 1,\ 1,\ 2,\ 0,\ 3,\ 2,\ 2\ \}$$

$$\overrightarrow{dy} = \{\ 2,\ 0,\ 1,\ 2,\ 0,\ 3,\ 0,\ 3,\ 1,\ 1,\ 0,\ 2,\ 3,\ 2,\ 1,\ 3\ \}$$

Με βάση τα παραπάνω διανύσματα μπορούμε εύκολα να πούμε ότι το τερματικό 1 βρίσκεται στην θέση (1,2) στον πίνακα. Τα παραπάνω δύο διανύσματα θα χρησιμεύσουν στον υπολογισμό του κόστους και στην εύρεση του κέντρου βάρους.

Επιπλέον, θα κατασκευάσουμε μια δομή με χρήση του αρχείου netlist. Κατασκευάζουμε έναν πίνακα που κάθε στήλη του αντιστοιχεί σε ένα net του netlist και κάθε γραμμή σε ένα εκ των τερματικών του κυκλώματος που προσπαθούμε να υλοποιήσουμε με τέτοια σειρά όπως στα παραπάνω δύο διανύσματα. Όσα τερματικά ανήκουν στο net έχουν την τιμή ένα και όσα δεν ανήκουν έχουν την τιμή μηδέν. Δηλαδή ο πίνακας είναι της μορφής

$$A = \begin{bmatrix} \alpha_{00} & \alpha_{01} & & \alpha_{0n} \\ \alpha_{10} & \alpha_{11} & \cdots & \alpha_{1n} \\ \alpha_{20} & \alpha_{21} & & \alpha_{2n} \\ \vdots & & \ddots & \vdots \\ \alpha_{m0} & \alpha_{m1} & \cdots & \alpha_{mn} \end{bmatrix} \quad \text{όπου} \quad \alpha_{ij} \begin{cases} 1 & , CLB\ i \in netj \\ 0 & , \delta\varepsilon\nu\ \alpha\nu\acute{\eta}\kappa\varepsilon\iota\ \sigma\varepsilon\ j \end{cases}$$

Ονομάζουμε τον συγκεκριμένο πίνακα, πίνακα γειτονικότητας. Έστω από ένα αρχείο netlist προκύπτουν τα παρακάτω nets

net numer #	Components #
00	1,2,3,4,5
01	2,4,6
02	2,5,8
03	4,6,7,9
04	5,8,10
05	8,10,11,12,13

Ο πίνακας γειτονικότητας που προκύπτει είναι ο ακόλουθος...

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Ο λόγος που ο συγκεκριμένος πίνακας είναι πολύ χρήσιμος είναι διότι ανά πάσα στιγμή εφαρμόζοντας ως δυαδική μάσκα μια στήλη αυτού του πίνακα στα διανύσματα διαστάσεων $\overrightarrow{dx}$ και $\overrightarrow{dy}$ μπορούμε απευθείας να υπολογίσουμε το άθροισμα των θέσεων σε x και y όλων των τερματικών που ανήκουν στο συγκεκριμένο net ενώ με scan στην συγκεκριμένη στήλη παίρνουμε τον βαθμό συνδεσιμότητας του συγκεκριμένου net το οποίο επιλέγουμε να το αποθηκεύσουμε σε ένα διάνυσμα $\vec{r}$ με $\vec{r}_j = \vec{1}_r * G_{ij}$. Με αυτές τις παρατηρήσεις καταλαβαίνουμε ότι η εύρεση του κέντρου βάρους κάθε net είναι εύκολη και με ορισμένες μετατροπές είναι εύκολος και ο υπολογισμός του κόστους κάθε net.

Χρήσιμοι Συμβολισμοί

Πολλαπλασιασμός ανά στοιχείο πινάκων: Έστω οι ίδιων διαστάσεων πίνακες A, B και C

$$C = A \circ B \ \ => c_{ij} = a_{ij} * b_{ij}$$

Δυνάμεις ανά στοιχείο πινάκων: Έστω οι ίδιων διαστάσεων πίνακες A, B

$$B = A^{\circ 2} \Rightarrow b_{ij} = a_{ij}^2$$

$$B = A^{\circ(-1)} \Rightarrow b_{ij} = \frac{1}{a_{ij}}$$

$$B = A^{\circ\frac{1}{2}} \Rightarrow b_{ij} = \sqrt{a_{ij}}$$

Διανύσματα $\overrightarrow{1_r}$ και $\overrightarrow{1_c}$:

$\overrightarrow{1_r}$ = [1 1 ⋯ 1] m στοιχεία όπου m ο αριθμός των τερματικών του κυκλώματος

$$\overrightarrow{1_c} = \left.\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}\right\} m \text{ όπου n ο αριθμός των nets}$$

Παρατηρούμε ότι A$\overrightarrow{1_c}$ = $\sum a_{i*}$ δηλαδή προκύπτει ένα διάνυσμα του οποίου κάθε στοιχείο είναι το άθροισμα της αντίστοιχης γραμμής του πίνακα A

Αντίστοιχα $\overrightarrow{1_r}$A = $\sum \alpha_{*j}$ δηλαδή ένα διάνυσμα του οποίου κάθε στοιχείο είναι το άθροισμα της αντίστοιχης στήλης του πίνακα A.

m x n πίνακας X και Y: όπου X_{ij} και Y_{ij} είναι m*n πίνακες με $x_{ij} = \begin{cases} dx_i, \alpha\nu\ G_{ij} = 1 \\ 0\ ,\ \alpha\nu\ G_{ij} = 0 \end{cases}$ και αντίστοιχα $y_{ij} = \begin{cases} dy_i, \alpha\nu\ G_{ij} = 1 \\ 0\ ,\ \alpha\nu\ G_{ij} = 0 \end{cases}$

διάνυσμα κέντρων βάρους GC: Το διάνυσμα κέντρων βάρους προκύπτει με την σχέση $\overrightarrow{gc_j} = \overrightarrow{1_r}\, X_{*j} \circ \vec{r}_j^{\circ(-1)}$

5.6 Αξιολόγηση κινήσεων.

Η αξιολόγηση της κάθε κίνησης πρέπει να γίνεται από ένα νήμα σε λογικό χρονικό διάστημα. Αυτός ήταν ο βασικότερος λόγος που επιλέγηκε ο αλγόριθμος Star έναντι του αλγορίθμου Star+ . Η διαφορά κόστους που προκαλείται σε κάθε net από την μετακίνηση ενός CLB με χρήση των παραπάνω δομών είναι εύκολα υπολογίσιμη. Ο τύπος που μας δίνει το κόστος κάθε net j όπως είδαμε είναι

$$Cost_j = \sum_{\forall i \in Net(j)} \left[(x_{Cj} - x_i)^2 + (y_{Cj} - y_i)^2\right]$$

το οποίο μπορεί να χωριστεί σε δύο ξεχωριστά αθροίσματα των x και y. Το καθένα από αυτά μπορεί να γραφεί και ως εξής

$$Cost_{xj} = \sum (x_{Cj} - x_k)^2 + (x_{Cj} - x_i)^2$$

$$Cost_{yj} = \sum (y_{Cj} - y_k)^2 + (y_{Cj} - y_i)^2$$

όπου το k αντιστοιχεί σε όλα τα CLBs μέλη του net j με εξαίρεση το μετακινούμενο και i το μετακινούμενο CLB μέλος του net j. Η διαφορά κόστους για μετακίνηση από θέση (x_i, y_i) σε (x_i', y_i') μπορεί να υπολογιστεί τμηματικά με τους δύο παραπάνω τύπους ως εξής:

$$\Delta C_{xj} = \sum \left(x_{Cj} - x_k\right)^2 + \left(x_{Cj} - x_i'\right)^2 - \sum \left(x_{Cj} - x_k\right)^2 - \left(x_{Cj} - x_i\right)^2$$

$$\Delta C_{xj} = \left(x_{Cj} - x_i'\right)^2 - \left(x_{Cj} - x_i\right)^2$$

Αντίστοιχα

$$\Delta C_{yj} = \left(y_{Cj} - y_i'\right)^2 - \left(y_{Cj} - y_i\right)^2$$

Οι δύο παραπάνω εξισώσεις είναι πολύ εύκολο να υπολογιστούν και είναι ανεξάρτητες από τα υπόλοιπα CLB μέλη του net.Έτσι για τον υπολογισμό της διαφοράς κόστους κάθε νήμα πρέπει πρώτα να ανασύρει από τα περιεχόμενα του πίνακα αναπαράστασης την ταυτότητα των κελιών που προσπαθεί να μετακινήσει και μετά για κάθε ένα από αυτά να εκτελέσει τις ακόλουθες πράξεις για να υπολογίσει την διαφορά κόστους που παρουσιάζονται από αυτήν την αλλαγή:

$$\Delta C_x = \overrightarrow{1_r}\left[G_{ij} \circ \left[\left(\overrightarrow{GC_j} - x_i'\right)^2 - \left(\overrightarrow{GC_j} - x_i\right)^2\right]\right]$$

όπου i η ταυτότητα του block που μετακινείται. Αντίστοιχα υπολογίζεται και το ΔC_y. Αν τα κελιά προς ανταλλαγή θέσεων είναι κενά η κίνηση αγνοείται. Στην συνέχεια ο αλγόριθμος Simulated Annealing κρίνει με τα ίδια κριτήρια που χρησιμοποιούνται και στο VPR αν η κίνηση πρέπει να γίνει αποδεκτή η όχι. Αν γίνει τότε στα διανύσματα dx και dy τα CLBs που εμπλέκονται στην κίνηση ενημερώνονται κατάλληλα. Η διαφορά κόστους κάθε κίνησης αποθηκεύεται σε ένα διάνυσμα μήκους ίδιου με τον συνολικό αριθμό προτεινόμενων κινήσεων με βάση το blockId και το threadId.

5.7 Αριθμός κινήσεων που εκτελέστηκαν και αυτών που έγιναν αποδεκτές.

Για τον υπολογισμό του αριθμού κινήσεων που έγιναν αποδεκτές και του συνολικού αριθμού κινήσεων είναι απαραίτητο να ορίσουμε στην μνήμη της κάρτας γραφικών ένα διάνυσμα ακεραίων M μήκους ίσου με τον μέγιστο αριθμό κινήσεων που μπορεί να πραγματοποιηθούν με βάση το μέγεθος του χώρου τοποθέτησης. Ο μέγιστος αριθμός κινήσεων είναι ίσος με

$$swapTotal = \text{minBlockSize.x} * \text{minBlockSize.y} * maxNofBlocks$$

με

$$maxNofBlocks = \left(\frac{\text{x}_{\max}}{\text{minBlockSize.y}} - 1\right)\left(\frac{y_{max}}{\text{minBlockSize.y}} - 1\right)$$

όπως ορίστηκαν νωρίτερα στην παράγραφο 5.3. Στην συγκεκριμένη εργασία οι ελάχιστες διαστάσεις ενός block είναι 4x8.

Στην συνέχεια αν μια κίνηση γίνει αποδεκτή τότε το νήμα που την αξιολόγησε γράφει την τιμή τρία στο κελί που του αντιστοιχεί με βάση το blockId και το threadIdx του, ενώ αν η κίνηση είναι μη αποδεκτή τότε γράφεται η τιμή δύο. Αν και τα δύο μέρη της ανταλλαγής κάθε προτεινόμενης κίνησης είναι κενά τότε η κίνηση αυτή αγνοείται και δεν γράφεται τίποτε στο διάνυσμα αυτό.

Στο στάδιο πέντε της ροής εκτέλεσης όπως περιγράφεται στην αρχή του κεφαλαίου, εφαρμόζοντας reduce στο διάνυσμα που προκύπτει κάνοντας διαίρεση του κάθε στοιχείου του διανύσματος M με το δύο υπολογίζουμε των αριθμό τον κινήσεων που δοκιμάστηκαν στην προηγούμενη επανάληψη. Εφαρμόζοντας reduce στο διάνυσμα που προκύπτει από την εύρεση υπολοίπου διαίρεσης με το δύο για κάθε στοιχείο του διανύσματος M έχουμε το σύνολο των αποδεκτών κινήσεων. Τα δύο παραπάνω μεγέθη χρησιμοποιούνται για να ελεγχθεί αν πρέπει να μεταβούμε σε αλλαγή θερμοκρασίας και τον υπολογισμό των Rlim και α όταν πρέπει να μεταβούμε σε νέα θερμοκρασία.

Κατά το τέλος του σταδίου πέντε το διάνυσμα M αρχικοποιείται σε μηδενικό διάνυσμα.

5.8 Υπολογισμός συνολικού κόστους τοποθέτησης.

Ο υπολογισμός του συνολικού κόστους τοποθέτησης είναι υπολογίσιμος με τις δομές που έχουμε ορίσει πιο πριν. Πιο συγκεκριμένα το κόστος ενός net δίνεται από τον τύπο:

$$cost_j = \sum_{\forall i \in Net j} \left(gc_j - x_i\right)^2$$

$$cost_j = \sum x_i^2 - 2\sum \left(x_i gc_j\right) + \sum gc_j^2$$

$$cost_j = \sum x_i^2 - 2gc_j \sum x_i + \sum gc_j^2$$

$$cost_j = \sum x_i^2 - 2gc_j\, r_j gc_j + r_j gc_j^2$$

$$cost_j = \sum x_i^2 - r_j gc_j^2$$

Όμως $\sum x_i^2 = \overrightarrow{1_r}(X \circ X)_j$

$$cost_j = \overrightarrow{1_r}(X \circ X)_j - r_j gc_j^2$$

Το συγκεκριμένο άθροισμα υπολογίζεται αρχικά μια φορά και παίρνουμε το συνολικό κόστος της αρχικής τοποθέτησης. Στην επόμενη επανάληψη χρησιμοποιούμε reduce στα διανύσματα ΔC_x και ΔC_y αφού πρώτα εφαρμοστεί πάνω τους ως δυαδική μάσκα το διάνυσμα M%2 το οποίο εκφράζει μόνο τις κινήσεις που έχουν γίνει αποδεκτές. Το άθροισμα αυτό μπορεί να υπολογιστεί εύκολα εφαρμόζοντας reduce μέσω της βιβλιοθήκης Thrust για να επιταχύνουμε την διαδικασία με χρήση GPU. Αθροίζοντας το συνολικό κόστος όπως έχει υπολογιστεί στην τελευταία επανάληψη και την συνολική διαφορά κόστους που προκλήθηκε από τις αποδεκτές κινήσεις προκύπτει το τρέχον συνολικό κόστος τοποθέτησης.

5.9 I/O blocks

Τα I/O pads προγραμματίζονται στο κύκλωμα στο τέλος. Έχοντας το κέντρο βάρους κάθε net στο οποίο είναι μέλος κάθε I/O, μπορούμε να ψάχνουμε την ελάχιστη απόσταση από αυτό είτε στο x=0 ή στο xmax ή στο y=0 ή στο ymax. Άν αυτό είναι κατειλημμένο εκείνη την στιγμή τότε δοκιμάζουμε τις θέσεις γειτονικά σε αυτό το σημείο και επαναλαμβάνουμε την διαδικασία μέχρι να βρεθεί ελεύθερος χώρος. Αν και δεν παίζουν πολύ σημαντικό ρόλο στην ποιότητα του τελικού κυκλώματος όπως έχει παρατηρηθεί σε άλλες μελέτες , εξασφαλίζεται ότι θα βρίσκονται όσο το δυνατόν πιο κοντά στο κέντρο βάρους του net στο οποίο ανήκουν.

6. Συμπεράσματα

Σε αυτή την εργασία παρουσιάσαμε μια μέθοδο παραλληλίας της διαδικασίας τοποθέτησης(placement) σε μια FPGA χρησιμοποιώντας ως βάση το πρόγραμμα VPR. Η υλοποίηση της εφαρμογής δεν ολοκληρώθηκε λόγω έλλειψης χρόνου όμως αναμένεται να παρουσιάζει αύξηση του βαθμού παραλληλίας όσο μεγαλώνει ο αριθμός των πυρήνων που χρησιμοποιούνται με την προϋπόθεση ότι και το πρόβλημα αυξάνει αναλόγως. Οι κάρτες γραφικών οι οποίες εξετάστηκαν στο παράδειγμα μας είναι οι κάρτες της Nvidia οι οποίες υποστηρίζουν CUDA αλλά σαν αλγόριθμος θα μπορούσε να υλοποιηθεί και σε άλλες κάρτες γραφικών με τις κατάλληλες μετατροπές. Η μέθοδός μας χρησιμοποιεί επίσης και βελτιστοποιημένα παράλληλα μοτίβα για τον υπολογισμό των μεγεθών που απαιτούνται για την αξιολόγηση της κάθε κίνησης αλλά και για τον έλεγχο της διάρκειας εκτέλεσης του προγράμματος και είναι ντετερμινιστική.

Για την παράλληλη εκτέλεση της συγκεκριμένης εφαρμογής προτείναμε την χρήση μιας μεθόδου που αυξάνει σε μεγάλο βαθμό την τυχαιότητα των κινήσεων που εκτελούνται. Η μέθοδος αυτή παράγει πλήρως ανεξάρτητες μεταξύ τους κινήσεις ώστε να αποφευχθούν τυχόν ασυνέπειες λόγω Write After Read, Read After Write και Write After Write. Ο τρόπος παραγωγής ανεξάρτητων κινήσεων διαμοιράζει τον χώρο τοποθέτησης διασπώντας το πρόβλημα σε μικρότερα που είναι ευκολότερο να υπολογιστούν. Επίσης αντιμετωπίζουμε το πρόβλημα της μετακίνησης των στοιχείων από τον κάθε υπόχωρο που σχηματίζεται σε θέσεις εκτός του συγκεκριμένου υποχώρου.

Στην συνέχεια παρουσιάσαμε τον τρόπο με τον οποίο η κάθε κίνηση αξιολογείται και γίνεται αποδεκτή ή όχι καθώς επίσης και έναν τρόπο ώστε τα δεδομένα των νημάτων κάθε block συναθροίζονται για να δώσουν αρχικά τις αποδεκτές και απορριπτέες κινήσεις του συγκεκριμένου block και αργότερα τις συνολικές που θα χρησιμοποιηθούν για να υπολογιστεί η επόμενη θερμοκρασία και η στιγμή που θα γίνει αλλαγή θερμοκρασίας. Ο τρόπος αυτός προέκυψε από την μετατροπή ενός υπάρχοντος αλγορίθμου Star[51] σε αλγόριθμο παράλληλης εκτέλεσης. Επιπλέον η μέθοδος μας επιλέγει να τοποθετήσει τα στοιχεία εισόδου εξόδου τελευταία με σειριακό τρόπο αλλά με τρόπο που τα τοποθετεί σε όσο το δυνατόν πιο βέλτιστη θέση.

Τέλος με την βοήθεια των παράλληλων σχημάτων της βιβλιοθήκης για C++, Thrust υπολογίζεται το κόστος της συνολικής τοποθέτησης εφαρμόζοντας ξεχωριστό kernel για αύξηση της απόδοσης που μπορούμε να πάρουμε από την κάρτα γραφικών.

Στόχος μας καθ' όλη την διάρκεια του σχεδιασμού είναι να ελαχιστοποιηθούν τα branch statements και η ελαχιστοποίηση των κλήσεων στην global μνήμη αν και σε ορισμένες περιπτώσεις θυσιάζουμε μερικώς την εκμετάλλευση της τοπικότητας της μνήμης για να αυξηθεί η τυχαιότητα των κινήσεων.

6.1 Προοπτικές μελλοντικών επεκτάσεων

Πλήρης Υλοποίηση της προτεινόμενης μεθόδου και σύγκρισή της: Λόγω έλλειψης χρόνου δεν ήταν δυνατόν να ολοκληρωθεί η υλοποίηση της προτεινόμενης μεθόδου. Η πλήρης υλοποίηση της μεθόδου θα αποδείξει ότι ο βαθμός επιτάχυνσης είναι αρκετά μεγάλος για να κάνει την μέθοδο χρήσιμη πιθανώς στην ενσωμάτωσή της σε διάφορα εργαλεία CAD.

Υποστήριξη ετερογενών στοιχείων στην FPGA: Σε αρκετές FPGA του εμπορίου παρέχεται η δυνατότητα να χρησιμοποιήσουμε ετερογενή στοιχεία όπως μνήμες Flash, DSPs και διάφορα άλλα εξαρτήματα τα οποία είναι πάνω στο chip της FPGA. Στην περίπτωση χρήσης τέτοιων στοιχείων η αναπαράσταση που χρησιμοποιείται σε αυτή την εργασία αδυνατεί να τα περιγράψει και έτσι μπορεί να χρησιμοποιηθεί μόνο σε περιπτώσεις στις οποίες απουσιάζουν αυτά τα στοιχεία. Μια επέκταση του μοντέλου μας θα μπορούσε πιθανώς να αντιμετωπίσει αυτό το πρόβλημα στο μέλλον.

Χρήση διαφορετικού τρόπου πρότασης κινήσεων: Στην συγκεκριμένη εργασία ο στόχος μας ήταν η αύξηση της τυχαιότητας των κινήσεων και η μέτρηση του αντίκτυπου που μπορεί να έχει στο τελικό κύκλωμα. Για διαφορετικούς στόχους, θα μπορούσε ένα εναλλακτικό σύστημα να αποδώσει καλύτερα. Για παράδειγμα στο παρελθόν έχουν προταθεί μέθοδοι για πιο κατευθυνόμενες κινήσεις[52] που έχουν καλύτερο τελικό αποτέλεσμα από τις τυχαίες αλλά λόγω της αύξησης σε υπολογιστικό χρόνο που απαιτούν δεν έχουν γίνει το νέο πρότυπο. Μια ενδιαφέρουσα επέκταση θα μπορούσε να είναι η ενσωμάτωση τέτοιων κριτηρίων στην μέθοδό μας ώστε με χρήση της δυνατότητας παραλληλίας που προσφέρουν οι κάρτες γραφικών να ξεπεραστεί το πρόβλημα του μεγάλου χρόνου εκτέλεσης.

Timing-Driven τοποθέτηση: Το εύρος της εργασίας αφορά την τοποθέτηση με κριτήριο το ελάχιστο μήκος αγωγών. Αυτό σημαίνει ότι δεν δίνεται έμφαση στο κρίσιμο μονοπάτι. Μετατρέποντας τον αλγόριθμο κατάλληλα είναι δυνατόν να δοθούν βάρη τέτοια ώστε σε συνδυασμό με κάποιο αλγόριθμο Timing-Driven να αυξηθεί σημαντικά η ταχύτητα του προκύπτοντος κυκλώματος.

Μετατροπή ώστε να στοχεύει και άλλες πλατφόρμες και κάρτες γραφικών: Όπως αναφέρθηκε νωρίτερα η μέθοδος μπορεί να υλοποιηθεί σε διαφορετικές πλατφόρμες από MIMD έως SIMD. Παρόλα αυτά αποδίδει καλύτερα όταν χρησιμοποιείται σε SIMD συστήματα λόγω του γεγονότος ότι στα παραδείγματά μας έγινε προσπάθεια να αντιμετωπιστούν οι περιορισμοί που παρουσιάζονται στα SIMD συστήματα. Επίσης χρησιμοποιεί έτοιμες βιβλιοθήκες της NVidia για τον υπολογισμό των μεγεθών που απαιτούνται. Η επιλογή καρτών γραφικών που μας παρέχεται στην αγορά είναι τεράστια. Για τον λόγο αυτό παρουσιάζει ιδιαίτερο ενδιαφέρον η τροποποίηση της μεθόδου ώστε να καλύπτει και διαφορετικών τύπων κάρτες καθώς επίσης και η βελτιστοποίηση εκτέλεσης της σε MIMD συστήματα.

Εφαρμογή της μεθόδου και σε άλλα προβλήματα: ο αλγόριθμος Simulated Annealing βρίσκει εφαρμογή σε πολλά προβλήματα. Παρόλο που η εργασία αυτή επικεντρώνεται στην επίλυση του προβλήματος τοποθέτησης CLBs σε FPGA ο τρόπος αντιμετώπισής του είναι πιθανόν να αποτελεί την λύση και για διαφορετικές εφαρμογές όπως για παράδειγμα επίλυση προβλημάτων σαν του Πλανόδιου Πωλητή - Travelling Salesman.

Βιβλιογραφία

1. Marquart, A., Betz, V., Rose, J.: Using cluster-based logic block and timing-driven packing to improve FPGA speed and density. In: International Symposium on FPGA, Monterey, pp. 37–46 (1999)
2. Singh, A., Marek-Sadowska, M.: Efficient circuit clustering for area and power reduction in FPGAs. In: International Symposium on Field Programmable Gate Arrays, pp. 59–66 (2002)
3. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. 49, 291–307 (1970)
4. Fiduccia, C.M., Mattheyeses, R.M.: A linear-time heuristic for improving network partitions. In: Design Automation Conference, pp. 175–181 (1982)
5. Huang, D., Kahng, A.: When clusters meet partitions: new density based methods for circuit decomposition.In IEEE European Design and Test Conference, pp60–64 (1995)
6. Bozorgzadeh, E., et al.: Routability-driven packing: metrics and algorithms for clusterbased FPGAs. IEEE J. Circuits Syst. Comput. 13(1), 77–100 (2004)
7. Hagen, L., Kahng, A.: Combining problem reduction and adaptive multi-start: a new technique for superior iterative partitioning. In: IEEE Trans. CAD, pp.92–98 (1997)
8. Dehkordi, M., Brown, S.: The effect of cluster packing and node duplication control in delay driven clustering. In: IEEE International Conference on Field Programmable Technology (ICFPT), pp. 227–233 (2002)
9. Murgai, R., Brayton, R., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/area. IEEE International Conference on Computer Aided Design, pp6–9 (1991)
10. Bui, T., Chaudhuri, S., Leighton, T., Sipser, M.: Graph Bisection Algorithms with Good Average Behavior. Combinatorica (1987)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
12. Betz, V., Marquardt, A., Rose, J.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, New York (Jan 1999)
13. Huang, M., Romeo, F., Sangiovanni-Vincentelli, A.: An efficient general cooling schedule for Simulated Annealing (ICCAD 1986 pp381-384)
14. Lam, J., Delosme, J.: Performance of a New Annealing Schedule (DAC 1988 pp306-311)
15. Ebeling, C., McMurchie, L., Hauck, S.A. and Burns, S.: Placement and Routing Tools for the triptych FPGA. IEEE Trans on VLSI, Dec 1995 pp473-482
16. Nag, S. and Rutenbar, R.: Performance-Driven Simultaneous Place and Route for Row-Based FPGAs. DAC, 1994, pp301-307
17. Nag, S. and Rutenbar, R.: Performance-Driven Simultaneous Place and Route for Island-Style FPGAs. ICCAD, 1995, pp332-338
18. Swarz, W. and Sechen. C.: New Algorithms for the Placement and Routing of Macro Cells. ICCAD, 1990, pp336-339
19. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press, Cambridge (1990)
20. Marquart, A., Betz, V., Rose, J.: Using cluster-based logic block and timing-driven packing to improve FPGA speed and density. In: International Symposium on FPGA, Monterey, pp. 37–46 (1999)
21. Betz, V., Marquardt, A., Rose, J.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, New York (Jan 1999)
22. Betz, V., Rose, J.: VPR: a new packing placement and routing tool for FPGA Research. In: International Workshop on FPGA, pp. 213–222 (1997)
23. Luu, J., Kuon, I., Jamieson, P., Campbell, T., Ye, A., Fang, W.M., Rose, J.: VPR 5.0: FPGACAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In: FPGA, pp. 133–142 (Feb 2009)
24. McMurchie, L., Ebeling, C.: Pathfinder: a negotiation-based performance-driven router for FPGAs. In: International Workshop on Field Programmable Gate Array, pp. 111–117 (1995)

25. Alexander, M., Cohoon, J., Ganley, J. and Robins, G.: An Architecture Independent Approach to FPGA Routing Based on Multi-Weighted Graphs. European Design Automation Conf., 1994 pp259-264
26. Wu, Y.L., Marek-Sadowska, M.: An Efficient Router for 2-D Field Programmable Gate Arrays. European Design Automation Conf., 1994 pp412-416
27. Wu, Y.L., Marek-Sadowska, M.: Orthogonal Greedy Coupling - - A New Optimization Approach to 2-D FPGA Routing. DAC, 1995, pp568-573
28. Alexander, M.J., Robins, G.: New Performance-Driven FPGA Routing Algorithms. DAC, 1995, pp562-567
29. Lee, Y.S., Wu, A.: A Performance and Routability Driven Router for FPGAs Considering Path Delays. DAC, 1995, pp557-561
30. Frankle, J.: Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing. DAC, 1992, pp536-542
31. Placzewski, M.: Plane Parallel A* Maze Router and its Application to FPGAs. DAC, 1992, pp691-697
32. Kravitz, S. and Rutenbar, R.: Placement by simulated annealing on a multiprocessor. TCAD, pp. 534–549, Jul. 1987.
33. Haldar, M., Nayak, A., Choudhary, A. and Banerjee, P: Parallel algorithms for FPGA placement, GLSVLSI, (Chicago, IL, USA), pp. 86–94, 2000.
34. P. Banerjee, M. Jones, and J. Sargent, "Parallel simulated annealing algorithms for cell placement on hypercube multiprocessors," TPDS, pp. 91–106, Jan. 1990.
35. W. Sun and C. Sechen, "A loosely coupled parallel algorithm for standard cell placement," in ICCAD, (San Jose, CA, USA), pp. 137–144, 1994.
36. S. Kim, J. A. Chandy, S. Parkes, B. Ramkumar, and P. Banerjee, "ProperPLACE: A portable parallel algorithm for standard cell placement," in IPPS, (Canc´un, Mexico), pp. 932–941, 1994.
37. J. Chandy and P. Banerjee, "Parallel simulated annealing strategies for VLSI cell placement," in VLSID, (Bangalore, India), pp. 37–42, 1996
38. E. Witte, R. Chamberlain, and M. Franklin, "Parallel simulated annealing using speculative computation," TPDS, vol. 2, pp. 483–494, Oct. 1991.
39. J. Chandy, S. Kim, B. Ramkumar, S. Parkes, and P. Banerjee, "An evaluation of parallel simulated annealing strategies with application to standard cell placement," TCAD, vol. 16, pp. 398–410, Apr. 1997.
40. A. Ludwin , V. Betz, Efficient and Deterministic Parallel Placement for FPGAs, ACM Transactions on Design Automation of Electronic Systems (TODAES), v.16 n.3, p.1-23, June 2011
41. Rami Beidas, Alexander Choong, Jianwen Zhu: Parallelizing Simulated Annealing-Based Placement Using GPGPU, International Conference on Field Programmable Logic and Applications, 2010, vol. 00, no. , pp. 31-34
42. Christian Fobel, Gary Gr´ewal, Robert Collier, Deborah Stacey: GPU Approach to FPGA Placement based on Star+ . New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International
43. C. C. Wang and G. G. F. Lemieux: Scalable and deterministic timing-driven parallel placement for FPGAs. ACM/SIGDA International Symposium on Field Programmable Gate Arrays. ACM, 2011, pp. 153 - 162.
44. David B Kirk, Wen-mei W. Hwu: Programming Massively Parallel Processors - A Hands-on Approach. ISBN: 978-0-12-381472-2
45. Farooq, U., Marrakchi, Z., Mehrez, H.: Tree-Based Heterogeneous FPGA Architectures - Application Specific Exploration and Optimization. ISBN 978-1-4614-3593-8
46. Fobel, C.: Deterministic, Weak-scaling Parallelism for Wirelength- and Timing-driven FPGA Placement, Suitable for Multicore and Manycore Architectures
47. M. Xu et al. "Near-linear wirelength estimation for FPGA placement". In: Canadian Conference on Electrical and Computer Engineering. 2009, pp. 1198–1203. doi: 10.1109/CJECE.2009.5443860
48. www.udacity.com Nvidia Tutorial for Cuda C++
49. Nvidia Corp at www.nvidia.com for Cuda C Documentation, Thrust C++ Documentation, GFX Specifications

50. Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. IEEE Trans. CAD 26(2), 203–215 (2007)
51. Xu, M., Grewal, G., Areibi, S. : Starplace: A new analytic method for FPGA placement
52. K. Vorwerk, A. Kennings, and J.W. Greene. "Improving simulated annealing-based FPGA placement with directed moves". In: Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 28.2 (2009), pp. 179–192. doi: 10.1109/TCAD.2008.2009167.
53. Bo Hu and Malgorzata Marek-Sadowska. "FAR: fixed-points addition & relaxation based placement". In: Proceedings of the 2002 international symposium on Physical design. San Diego, CA, USA: ACM, 2002, pp. 161–166. doi: 10.1145/505388.505426
54. Kristofer Vorwerk and Andrew Kennings. "An improved multi-level framework for force directed placement". In: Proceedings of the conference on Design, Automation and Test in Europe - Volume 2. IEEE Computer Society, 2005, pp. 902–907. doi: 10.1109/DATE.2005.59.
55. Yonghong Xu and Mohammed A. S. Khalid. "QPF: efficient quadratic placement for FPGAs". In: International Conference on Field Programmable Logic and Applications.Vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 555–558. doi:10.1109/FPL.2005.1515784.
56. Padmini Gopalakrishnan, Xin Li, and Lawrence Pileggi. "Architecture-aware FPGA placement using metric embedding". In: Proceedings of the 43rd annual Design Automation Conference. San Francisco, CA, USA: ACM, 2006, pp. 460–465. doi: 10.1145/1146909.1147033.
57. Natarajan Viswanathan, Min Pan, and Chris Chu. "FastPlace: An efficient multilevel force-directed placement algorithm". In: Modern Circuit Placement. Ed. by Gi-Joon Nam and Jason Cong. Series on Integrated Circuits and Systems. Springer US, 2007, pp. 193–228. doi: 10.1007/978-0-387-68739-1_8.
58. William Swartz and Carl Sechen. "Timing driven placement for large standard cell circuits". In: Proceedings of the 32nd annual ACM/IEEE Design Automation Conference.San Francisco, California, United States: ACM Press, 1995, 211215. doi: 10.1145/217474.217531.
59. J.M. Kleinhans et al. "GORDIAN: VLSI placement by quadratic programming and slicing optimization". In: Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 10.3 (1991), pp. 356–365. doi: 10.1109/43.67789.
60. Jens Vygen. "Algorithms for large-scale flat placement". In: Proceedings of the 34th annual Design Automation Conference. Anaheim, California, United States: ACM, 1997, pp. 746–751. doi: 10.1145/266021.266360.
61. Georg Sigl, Konrad Doll, and Frank M. Johannes. "Analytical placement: A linear or a quadratic objective function?" In: Proceedings of the 28th ACM/IEEE Design Automation Conference. San Francisco, California, United States: ACM, 1991, pp. 427–432. doi:10.1145/127601.127707.
62. P. Maidee, C. Ababei, and K. Bazargan. "Timing-driven partitioning-based placement for island style FPGAs". In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24.3 (2005), pp. 395–406. doi: 10.1109/TCAD.2004.842812.
63. George Karypis et al. "Multilevel hypergraph partitioning: application in VLSI domain".In: Proceedings of the 34th annual Design Automation Conference. Anaheim, California, United States: ACM, 1997, pp. 526–529. doi: 10.1145/266021.266273.
64. Manuel Rubio-Solar et al. "A FPGA optimization tool based on a multi-island genetic algorithm distributed over grid environments". In: IEEE International Symposium on Cluster Computing and the Grid. Los Alamitos, CA, USA: IEEE Computer Society, 2008,pp. 65–72. doi: 10.1109/CCGRID.2008.96.
65. Meng Yang, A. Almaini, and Pengjun Wang. "FPGA placement optimization by two-step unified genetic algorithm and simulated annealing algorithm". In: Journal of Electronics (China) 23.4 (July 2006), pp. 632–636. doi: 10.1007/s11767-005-0198-3.

66. Peter Jamieson. “Revisiting genetic algorithms for the FPGA placement problem”. In:GEM. 2010, pp. 16–22.
67. Wikipedia. www.wikipedia.org
68. StackOverflow stackoverflow.com