Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Επιτάχυνση διανυσματικού πολλαπλασιασμού πινάκων με χρήση FPGA και HLS

Σιδηρόπουλος Στέφανος

Τριμελής Επιτροπή:

Διονύσιος Ν.Πνευματικάτος(Καθηγητής),

Ιωάννης Παπαευσταθίου(Αναπ. Καθηγητής),

Γρηγόριος Χρυσός(Δρ.)

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή

i. Γενικές πληροφορίες

Τα τελευταία χρόνια υπάρχει η ανάπτυξη των παράλληλων μικροεπεξεργαστών ως τάση της ανάπτυξης υπολογιστικών συστημάτων, γεγονός το οποίο ορίζει την τάση και στην ανάπτυξη του λογισμικού. Αυτή η τάση ξεκίνησε το 2005 όταν η Intel ακολούθησε το παράδειγμα της IBM η οποία ανακοίνωσε πως στο εξής η υψηλή απόδοση των επεξεργαστών θα βασίζεται στους πολλαπλούς επεξεργαστές ή πυρήνες. Έτσι γεννήθηκε η λέξη «multicore» η οποία συνδέθηκε και με την προσπάθεια διπλασιασμού των πυρήνων σε κάθε καινούρια γενιά ημιαγωγών επεξεργαστών. Οι πολλαπλοί πυρήνες είναι προφανές πως βοηθούν στην απόδοση εκτέλεσης παράλληλων διεργασιών , χωρίς όμως να αυξάνεται η απόδοση της κάθε διεργασίας ως ανεξάρτητη οντότητα, ενώ η προσέγγιση σχεδιασμού του λογισμικού ήταν αναγκαίο να αλλάξει με βάση την προσαρμογή του τρόπου σχεδιασμού από ακολουθιακά σε παράλληλα προγράμματα. Η ιδέα του παραλληλισμού υπήρχε σε ερευνητικό κομμάτι τουλάχιστον τρεις δεκαετίες όμως τελικά επικρατούσε πάντα η ιδέα του ενός πυρήνα. Πλέον η επικράτηση του παραλληλισμού είναι αποτέλεσμα λύσεων από ακόμα μεγαλύτερα προβλήματα τα οποία οδήγησαν τελικά στην αποδοτική εκμετάλλευση της σιλικόνης σε παραδοσιακές δομές μονοπύρηνων αρχιτεκτονικών.

Η κεντρική ιδέα ήταν πως η αξιοποίηση της γνώσης που υπήρχε σε επίπεδο παραλληλισμού, καθώς και η παραγωγή προγραμμάτων τα οποία θα λειτουργούσαν αποδοτικά σε ένα μεγάλο πλήθος παράλληλων υπολογιστών, θα μπορούσε να αποτελέσει τον ακρογωνιαίο λίθο για την ανάπτυξη παράλληλων εφαρμογών στο μέλλον. Μια από τις μεγαλύτερες προκλήσεις στον παραλληλισμό είναι πως δεν μπορεί να δημιουργηθεί ένας καθολικός τρόπος παραλληλισμού ο οποίος να έχει την μεγαλύτερη απόδοση. Το γεγονός αυτό οδηγεί στην ανάγκη να θεμελιωθούν κανόνες αποδοτικής παραλληλοποίησης σε ένα πιο γενικό επίπεδο και όχι σε συγκεκριμένα προγράμματα. Έτσι πυροδοτήθηκε ο ορισμός ενός αριθμού «Νάνων». Οι νάνοι είναι υπολογιστικά πρότυπα υπολογισμού και επικοινωνίας, τα οποία είναι κοινά σε έναν μεγάλο αριθμό εφαρμογών. [1]

Έχοντας ως γνώμονα την αύξηση της απόδοσης στα υπολογιστικά συστήματα υπήρξε και ανάπτυξη στον τομέα των FPGA (Field-Programmable Gate Array). Οι FPGA είναι ολοκληρωμένα κυκλώματα τα οποία παρέχουν την δυνατότητα να επαναπρογραμματιστούν κυρίως με την χρήση γλωσσών περιγραφής υλικού (Hardware Description Language).

Αποτελούνται από λογικά πεδία (Logic Block) τα οποία περιέχουν λογικές πύλες και επιτρέπουν τον συνδυασμό τους παράγοντας απλά ή σύνθετα τελικά κυκλώματα. Κύριος παραγωγός σήμερα είναι η εταιρία «Xilinx» που ιδρύθηκε το 1984, η οποία ανέπτυξε την συγκεκριμένη τεχνολογία και ενώ αρχικά έβρισκε εφαρμογή στην διασύνδεση μεταξύ άλλων ολοκληρωμένων κυκλωμάτων λόγω της έλλειψης χωρητικότητας, πλέον οι FPGAs αποτελούν μηχανές επεξεργασίας δεδομένων.

ii. Συνεισφορά εργασίας

Στην παρούσα διπλωματική εργασία γίνεται αρχικά απεικόνιση δύο Νάνων σε τεχνολογία αναδιατασσόμενης λογικής με τη χρήση HLS(High-Level Synthesis). Επίσης γίνεται επιτάχυνση των δύο νάνων με τη χρήση FPGA(Field Programmable Gate Array) και του εργαλείου «Vivado» το οποίο επιτρέπει την εισαγωγή directives για την πραγματοποίηση της επιτάχυνσης όπως θα αναλυθεί στην συνέχεια της εργασίας. Η διαδικασία αφορά τους νάνους «Sparse Linear Algebra»(Αραιά γραμμική άλγεβρα), ο οποίος υλοποιεί πολλαπλασιασμό αραιών σε στοιχεία πινάκων και «Dense Linear Algebra »(Πυκνή Γραμμική άλγεβρα), ο οποίος υλοποιεί πολλαπλασιασμό πυκνών σε στοιχεία πινάκων. Η επιτάχυνση των «Νάνων» είναι σημαντική διότι κατά συνέπεια επιταχύνονται και οι εφαρμογές οι οποίες χρησιμοποιούν τον εκάστοτε «Νάνο».

Συνοψίζοντας στην συγκεκριμένη εργασία επιτυγχάνονται τα εξής:

- Απεικόνιση δύο από τους Νάνους του Berkelay [4], «Sparse Linear Algebra» (Αραιά γραμμική άλγεβρα) και «Dense Linear Algebra» (Πυκνή Γραμμική άλγεβρα), σε αναδιατασσόμενη λογική με τη χρήση High Level Synthesis εργαλείου.
- Επιτάχυνση των εφαρμογών, πολλαπλασιασμός αραιών και πυκνών πινάκων, που ανήκουν στους παραπάνω Νάνους με τη χρήση HLS εργαλείων και κατάλληλων directives.
- Έρευνα και αξιολόγηση των directives για αλγορίθμους οι οποίοι χρησιμοποιούν δομές πινάκων και εμφωλευμένους βρόγχους.
- Εξαγωγή συμπερασμάτων σχετικά με τη χρήση των directives για εφαρμογές που ανήκουν στις συγκερκιμένες 2 κατηγορίες νάνων που επικεντρώθηκε η συγκεκριμένη διπλωματική εργασία.

Κεφάλαιο 2: Σχετική Εργασία

i. Οι νάνοι

Οι νάνοι βασίστηκαν στην αρχική δουλειά του Phil Colella, ο οποίος όρισε εφτά αριθμητικές μεθόδους που θεωρούσε πως θα ήτανε πολύ σημαντικές για μία πληθώρα εφαρμογών στις θετικές επιστήμες και την μηχανική, τουλάχιστον για την επόμενη δεκαετία. Οι εφτά νάνοι περιγράφονται στην συνέχεια μαζί με μερικά παραδείγματα εφαρμογής τους.

Dense Linear Algebra: Τα δεδομένα αποτελούνται από πλειοψηφία μη μηδενικών δεδομένων σε πίνακες ή διανύσματα, τα οποία πολλαπλασιάζονται μεταξύ τους ($1^{ου}$ επιπέδου θεωρείται ο πολλαπλασιασμός διανύσματος-διανύσματος, $2^{ου}$ επιπέδου ο πολλαπλασιασμός πίνακα-διανύσματος και $3^{ου}$ επιπέδου ο πολλαπλασιασμός πίνακα-πίνακα). Γενικά τέτοιου είδους εφαρμογές χρησιμοποιούν κάποιον βηματισμό για να διαβάζουν δεδομένα από τις γραμμές και τις στήλες των πινάκων. Κάποιες χαρακτηριστικές εφαρμογές είναι στους τριγωνικούς πίνακες και στους συμμετρικούς πίνακες με επιρροή στην κατασκευή γραφικών.

Sparse Linear Algebra: Το σύνολο των δεδομένων εμπεριέχει πολλά μηδενικά στοιχεία. Τα μη μηδενικά στοιχεία αποθηκεύονται με τρόπο ώστε η πρόσβαση σε αυτά να είναι πολύ πιο γρήγορη από την απλή προσπέλαση των πινάκων. Λόγω της ιδιομορφίας των δεδομένων η πρόσβαση σε αυτά γίνονται με αναγνώσεις και αποθηκεύσεις οι οποίες βασίζονται σε δείκτες. Χαρακτηριστικές εφαρμογές στην κλίση διανυσμάτων με επιρροή στην κατασκευή γραφικών.

Spectral Methods: Τα δεδομένα αφορούν το πεδίο της συχνότητας και όχι του χρόνου. Συνήθως οι εφαρμογές φασμάτων χρησιμοποιούν προσθέσεις και πολλαπλασιασμούς, οι οποίοι προέρχονται από πολλά στάδια υπολογισμών και μέσω ενός προτύπου επικοινωνίας μεταξύ των σταδίων συνδυάζονται. Χαρακτηριστικές εφαρμογές βρίσκει στον μετασχηματισμό Fourier και στην ψηφιακή επεξεργασία σήματος.

N-Body Methods: Σχετίζεται με τις αλληλεπιδράσεις μεταξύ διακριτών σημείων. Παραλλαγές του περιλαμβάνουν τις σχέσεις σωματιδίου-σωματιδίου όπου κάθε σωματίδιο εξαρτάται από όλα τα υπόλοιπα οδηγώντας σε O(N) πολυπλοκότητα υπολογισμών, καθώς και την ιεραρχική δομική σχέση μεταξύ των σωματιδίων στην οποία εμπλέκονται δυνάμεις ή δυναμικά διατηρώντας την πολυπλοκότητα σταθερή ή μειώνοντάς της σε O(NlogN).

Structured Grids: Αναπαρίσταται από ένα πλέγμα, στο οποίο τα σημεία έχουν εννοιολογική σχέση και ενημερώνονται εννοιολογικά μαζί. Έχει μεγάλη χωρική τοπικότητα(spatial

locality) και οι ενημερώσεις μπορεί να γίνονται σε μία συγκεκριμένη περιοχή ή μεταξύ δύο εκδοχών του πλέγματος. Επίσης το πλέγμα μπορεί να διαιρεθεί σε μικρότερα πλέγματα ή περιοχές ενδιαφέροντος και η μετάβαση μεταξύ των σημείων μπορεί να γίνει δυναμικά. Χαρακτηριστικές εφαρμογές σε πολυπλεγματικούς κλιμακωτούς πεντοδιαγώνιους πίνακες.

Unstructured Grids: Αποτελείται από ένα ακανόνιστο πλέγμα, όπου οι θέσεις των στοιχείων του συνήθως προκύπτουν από χαρακτηριστικά της εφαρμογής. Οι θέσεις των στοιχείων του πλέγματος καθώς και οι μεταξύ τους σχέσεις πρέπει να είναι αυστηρά ορισμένες. Οι ενημερώσεις γίνονται εννοιολογικά και εμπεριέχουν πολλές αναφορές μνήμης, καθώς για να γίνει μια ενημέρωση σε οποιοδήποτε σημείο θα πρέπει πρώτα να αναλυθεί η λίστα των γειτονικών του στοιχείων και να διαβαστούν τιμές από αυτά τα γειτονικά σημεία. Χαρακτηριστικές εφαρμογές σε διανυσματικούς υπολογιστές συγκέντρωσης/διασποράς.

Monte Carlo: Σχετίζεται με στατιστικούς υπολογισμούς αποτελεσμάτων που προκύπτουν από επαναληπτικές τυχαίες δοκιμές οι οποίες είναι άμεσα παραλληλοποιήσιμες (Embarrassingly parallel). Χαρακτηριστικές εφαρμογές σε άμεσα παραλληλοποιήσημους αλγόριθμους.

Οι νάνοι αποτελούν μία μέθοδο για την ανάδειξη κοινών απαιτήσεων διαφορετικών κατηγοριών εφαρμογών, σε ένα πιο αφηρημένο επίπεδο χωρίς να λαμβάνονται υπόψιν απαιτήσεις για μεμονωμένες εφαρμογές. Οι εφτά αρχικοί νάνοι είχαν οριστεί από τον Phil Colella σύμφωνα με κριτήρια τα οποία αναφέρθηκαν προηγουμένως, ωστόσο στην συνέχεια γεννήθηκε η επιθυμία σε ερευνητές του πανεπιστημίου του Berkeley να υπάρχει εφαρμογή τους σε περισσότερα επιστημονικά πεδία. Αυτό οδήγησε σε δύο βασικές ερωτήσεις: Πόσο ικανοποιητικά καλύπτουν οι υπάρχοντες νάνοι ένα μεγαλύτερο φάσμα εφαρμογών και ποιοι νάνοι πρέπει να προστεθούν για να καλύψουν τα κενά που υπάρχουν. Όπως έχει προαναφερθεί οι νάνοι χαρακτηρίζονται από ένα αφηρημένο επίπεδο ανάλυσης των εφαρμογών. Κατά την διάρκεια του χρόνου μπορεί να εμφανίζονται διαφορετικά προβλήματα, με διαφορετικές απαιτήσεις, διαφορετικές εκδοχές νάνων που τελικά λόγω της αύξησης της διαφορετικότητάς τους να ορίζονται ως καινούριοι νάνοι. Αυτό όσο δεν πολλαπλασιάζονται δραματικά οι νάνοι είναι αποδεκτό. Χαρακτηριστικό παράδειγμα αποτελεί αυτό του unstructured grid το οποίο θα μπορούσε να αντιμετωπιστεί και με εφαρμογή του Sparse matrix, ωστόσο θα αγνοούσε πολλές ιδιομορφίες της εφαρμογής. Σύμφωνα με τα παραπάνω λοιπόν, οι ερευνητές στο Berkeley πρόσθεσαν στου υπάρχοντες Νάνους τους τέσσερις παρακάτω:

Combinational logic: Σχετίζεται με την παράλληλη εκτέλεση απλών διεργασιών που έχουν να κάνουν με μεγάλα δεδομένα, συχνά εκμεταλλευόμενο τον bit-level παραλληλισμό.

Χαρακτηριστικό παράδειγμα εφαρμογής οι κώδικες ελέγχου κυκλικού πλεονασμού(Cyclic Redundancy Codes).

Graph Traversal: Σχετίζεται με την προσπέλαση μιας σειράς αντικειμένων συλλέγοντας στοιχεία από αυτά που θα μπορούσαν να χρησιμοποιηθούν στην αναζήτηση. Χρησιμοποιείται συχνά στην αναζήτηση στοιχείων σε πίνακα με μικρό υπολογιστικό κόστος.

Graphical Models: Εφαρμόζεται σε εφαρμογές που αποτελούνται από γράφους οι οποίοι απεικονίζουν τυχαίες μεταβλητές ως κόμβους και δεσμευμένες εξαρτήσεις ως ακμές. Χαρακτηριστικά παραδείγματα χρήσης είναι τα δίκτυα Bayes, καθώς και τα κρυφά Μαρκοβιανά μοντέλα(Hidden Markov Model).

Finite State Machines: Απεικονίζουν ένα διασυνδεδεμένο σύνολο καταστάσεων το οποίο θα μπορούσε να χρησιμοποιηθεί για προσπέλαση. Κάποιες μηχανές πεπερασμένων καταστάσεων θα μπορούσαν να διασπαστούν και να εκτελεστούν παράλληλα.

Οι αναπτυσσόμενες τάσεις της τεχνολογίας επίσης παίξανε καθοριστικό ρόλο στον ορισμό επιπλέον νάνων. Μία πολλά υποσχόμενη τάση ήταν η μηχανική μάθηση, η οποία με βάσει στατιστικά αποτελέσματα εκπαιδεύει συστήματα προβλέψεων και λήψης αποφάσεων. Το πλήθος των δεδομένων για την εκπαίδευση του συστήματος οδήγησε στον ορισμό δύο επιπλέον νάνων από τους ερευνητές του πανεπιστημίου.

Dynamic programming: Σχετίζεται με την επίλυση προβλημάτων διασπώντας το αρχικό πρόβλημα σε μικρότερα. Ο συνδυασμός των λύσεων των επιμέρους προβλημάτων οδηγούν στην συνολική λύση του προβλήματος. Παράδειγμα εφαρμογής είναι τα προβλήματα βελτιστοποίησης όπου το καλύτερο δυνατό αποτέλεσμα μπορεί να προέλθει από τον τα καλύτερα δυνατά αποτελέσματα των υποπροβλημάτων του.

Backtrack and Branch-and-Bound: Σχετίζεται με εκτεταμένες αναζητήσεις σε προβλήματα βελτιστοποίησης μεγάλου αριθμού και ακαθόριστου τύπου δεδομένων. Συνήθως χρειάζεται κάποια υποβοήθεια για τον αποκλεισμό περιοχών αναζήτησης στις οποίες δεν υπάρχουν βέλτιστες λύσεις. Ο αλγόριθμος βασίζεται στην μέθοδο «Διαίρει και βασίλευε» όπου ο αρχικός χώρος αναζήτησης διασπάται σε μικρότερους και τα όρια της κάθε περιοχής βρίσκονται έπειτα από ανακατασκευές των περιοχών.

Τεχνολογικός τομέας όπου οι νάνοι βρίσκουν μεγάλη εφαρμογή αποτελούν επίσης οι βάσεις δεδομένων. Στον τομέα αυτό υπάρχει η τάση να γίνονται όλο και μεγαλύτερες οι βάσεις καθώς τα δεδομένα πολλές φορές προέρχονται από το διαδίκτυο , στο οποίο τα δεδομένα συνεχώς αυξάνονται. Αυτό έχει ως αποτέλεσμα να εξελίσσονται οι μέθοδοι αναζήτησης. Μεγάλη εφαρμογή γνωρίζει ο αλγόριθμος «MapReduce» ο οποίος χρησιμοποιήθηκε με

επιτυχία και από την Google. Ο «MapReduce» αποτελεί μία πιο γενική εκδοχή του ήδη ορισμένου νάνου «Monte Carlo» χωρίς όμως να αποτελέσει τελικά ξεχωριστό νάνο.

Άλλος τομέας στον οποίον βρίσκουν μεγάλη εφαρμογή οι νάνοι είναι αυτός των γραφικών και των παιχνιδιών. Η χρήση διανυσμάτων για την κατασκευή σχημάτων και στην συνέχεια η ένωση και η κίνησή τους με χρήση κινηματικών εξισώσεων για την δημιουργία των γραφικών, αποτελούν ιδανική περίπτωση για την εφαρμογή ορισμένων νάνων. Παραδείγματα αποτελούν το «Sparse-Linear Algebra» για περιπτώσεις αντίστροφων κινηματικών, το «Graph Traversal» για τον εντοπισμό συγκρούσεων σε δίκτυα, καθώς και το «Finite State» για ανταπόκριση στις συγκρούσεις. Παρ' όλα αυτά δεν ορίστηκε ούτε από αυτόν τον τομέα κάποιος επιπλέον νάνος.[1, 11]

ii. Acceleration attempts των νάνων

Στα πλαίσια της ανάπτυξης του τομέα υπήρξε ανάπτυξη στην κατεύθυνση της παραγωγής εργαλείων HLS (High level synthesis) τα οποία να παράγουν το RTL(Register transfer level) μιας FPGA [13]. Η επιλογή τέτοιου είδους εργαλείων αντιμετωπίζονταν με σκεπτικισμό καθώς μπορεί να υπήρχε έλλειψη αποδοτικής σχεδίασης σε τέτοιου είδους αυτοματισμούς. Ωστόσο για να αποδειχθεί το αντίθετο χρησιμοποιήθηκαν δύο πειράματα από την Berkeley Design Technology Incorporation (BDTI), μία εφαρμογή ανάλυσης κίνησης σε video και μία εφαρμογή ασύρματου δέκτη. Και στις δύο η υλοποίηση με HLS και FPGA εξήγαγε απόδοση περίπου 40 φορές καλύτερη σε σχέση με την υλοποίηση σε έναν DSP επεξεργαστή και μία RTL η οποία δεν είχε προκύψει από αυτοματοποιημένο εργαλείο, ενώ η σχέση απόδοσης – κόστους ήταν περίπου 30 φορές καλύτερη με την χρήση HLS και FPGA, δίνοντας μία απόδειξη πως τα σύγχρονα εργαλεία HLS παρέχουν σχεδίαση υψηλής απόδοσης σε ορισμένες εφαρμογές. Γενικότερα προέκυψε πως είναι το ίδιο αποτελεσματικά όσο αναφορά την αποδοτικότητα χωρίς όμως να χρειάζονται γνώσεις για σχεδίαση RTL με την χρήση αυτών των εργαλείων και σε λιγότερο χρόνο [2, 11, 12].

Επίσης αντικείμενο προς μελέτη υπήρξε η χρήση της OpenCL, καθώς και της SOpenCL [10], για την επιτάχυνση αλγορίθμων με σημείο αναφοράς και πάλι τους νάνους, σε συνδυασμό με την χρήση FPGAs. Συγκεκριμένα χρησιμοποιήθηκαν οι νάνοι «N-Body» και «Structured Grid», όπως έχουν περιγραφεί παραπάνω. Κομμάτια της μελέτης ήταν αρχικά η αξιολόγηση της διαφοράς αποδοτικότητας μεταξύ πάγιων και επαναδιαμορφοποιήσιμων αρχιτεκτονικών με την χρήση OpenCL ενώ απόρροια του κομματιού αυτού αποτελεί η μελέτη για την παραγωγή ευρετικών μεθοδολογιών ώστε να μικρύνει το χάσμα μεταξύ των δύο ειδών

αρχιτεκτονικών. Οι μεθοδολογίες αυτές αποτελούνται από συμπεράσματα τα οποία αφορούν τον τρόπο επεξεργασίας των δεδομένων, τρόπους μεταφοράς δεδομένων, καθώς και διαχείριση μνήμης. Επίσης έγινε χρήση OpenCL σε συνδυασμό με FPGAs για την επιτάχυνση του δεύτερου νάνου και την αξιολόγηση των αποτελεσμάτων [3].

Η απόδοση μεταξύ πάγιων και επαναδιαμορφοποιήσιμων αρχιτεκτονικών έχει απασχολήσει και άλλη μελέτη στην οποία γίνεται χρήση OpenCL και FPGAs για την επιτάχυνση των νάνων με σκοπό να ανακαλυφτούν τα υπέρ και τα κατά των αποτελεσμάτων επιτάχυνσης για μία σειρά νάνων. Η σειρά αυτή αποτελείται από τους νάνους «N-Body», «Dynamic Programming» [14], «Structured Grid» και «Graph Traversal» ενώ η κάποιες σειρές αρχιτεκτονικών στις οποίες γίνονται οι αξιολογήσεις είναι οι «Opteron 6272», «A8-3850», «A10-5800K», «HD6550D», «HD7970», «HD7660D», «Xeon Phi P1750», «FPGA_C1», «FPGA_C2» και «FPGA_C3» [4]. Παρόμοια εργασία έχει πραγματοποιηθεί και για μία σειρά από GPU πλατφόρμες όπως για παράδειγμα AMD HD5450, AMD HD5870, NVIDIA GT520, και NVIDIA C2050 [8]. Επίπλεον αξιολόγηση με τον ίδιο τρόπο έχει πραγματοποιηθεί και για τον νάνο «Sparse Linear Algebra» όπου μεγαλύτερη απόδοση παρατηρήθηκε στις αρχιτεκτονικές οι οποίες είναι ενδεικτικές για Fused Multiply-Add(FMA) εντολές [9].

Κεφάλαιο 3: Vivado HLS

Η επιτάχυνση ενός αλγορίθμου είναι σημαντική καθώς έτσι βελτιώνεται και η συνολική απόδοση μιας εφαρμογής που χρησιμοποιεί τον αλγόριθμο. Ένας τρόπος επιτάχυνσης είναι η απεικόνιση και η παραλληλοποίηση του αλγορίθμου με την χρήση HLS(High level synthesis). Το 2009 η Berkeley Design Technology Incorporation δημοσίευσε στην αγορά ένα σύγχρονο εργαλείο HLS. Τέτοιου είδους εργαλεία παίρνουν ως είσοδο την αναπαράσταση μιας εφαρμογής σε μορφή υψηλού επιπέδου κώδικα, όπως C και Matlab και παράγουν το RTL(Register Transfer Level) για μία FPGA. Έτσι η HLS επιτρέπει την παραλληλοποίηση σημείων της εφαρμογής, όπως για παράδειγμα παραλληλοποίηση βρόγχων, καθώς επίσης και συγχρονισμό σε μία σχεδίαση , δουλεύοντας όμως σε μία υψηλού επιπέδου γλώσσα. Ως αποτέλεσμα, μειώνεται η εμπλοκή του μηχανικού στην σχεδίαση του RTL, γεγονός το οποίο μειώνει τα σφάλματα αλλά μπορεί και να αφήσει ανεκμετάλλευτο κάποιο ποσοστό της βελτιστοποίησης στην απόδοση που μπορεί να γίνει.

i. Directives και ανάλυση των directives

Το Vivado είναι εργαλείο της εταιρείας Xilinx, η οποία είναι από τις μεγαλύτερες εταιρείες παραγωγής FPGA παγκοσμίως. Το συγκεκριμένο εργαλείο δέχεται ως είσοδο μία υψηλού επιπέδου γλώσσα προγραμματισμού, παρόμοια με την C και την C++, με ορισμένους περιορισμούς οι οποίοι αναλύονται σε επόμενο κεφάλαιο και παράγει το RTL. Για την επιτάχυνση του αλγορίθμου χρησιμοποιούνται μέθοδοι οι οποίο ονομάζονται «directives» και τα ονόματα και η λειτουργία των οποίων περιγράφονται στην συνέχεια. Τα συγκεκριμένα directives και η περιγραφή τους προέρχονται από το εγχειρίδιο της Xilinx για το εργαλείο Vivado [5].

ALLOCATION: Το directive αυτό ορίζει τον αριθμό των πράξεων πυρήνων και των συναρτήσεων που θα χρησιμοποιηθούν περιορίζοντας τον αριθμό των πόρων σε επίπεδο RTL, ίσως με αύξηση της απόδοσης. Δέχεται ως παραμέτρους τον μέγιστο αριθμό των «block» στο RTL, καθώς και την πράξη ή την συνάρτηση που αυτά υλοποιούν.

ARRAY_MAP: Το συγκεκριμένο directive συνδυάζει μικρότερους πίνακες σε μία και μοναδική δομή πίνακα με οριζόντιο ή κατακόρυφο προσανατολισμό καθώς και με κάποιο επιθυμητό offset.

ARRAY_PARTITION: Το directive αυτό διασπά έναν πίνακα σε μικρότερους πίνακες ή σε ανεξάρτητα στοιχεία. Περιέχει τρεις επιλογές, την «type», την «factor» και «dimension». Η «type» έχει να κάνει με τον τύπο της διάσπασης και μπορεί να είναι «block» όπου ο αρχικός πίνακας χωρίζεται σε μικρότερους διατηρώντας τα διαδοχικά blocks του αρχικού, «cyclic» όπου ο αρχικός πίνακας χωρίζεται σε μικρότερους σε κυκλικά στοιχεία σε σχέση με τον αρχικό, καθώς και «complete» όπου ο αρχικός πίνακας διασπάται σε ανεξάρτητα στοιχεία. Ακόμη, η επιλογή «factor» ορίζει το σε πόσους υποπίνακες θα χωριστεί ο αρχικός πίνακας, ενώ η «dimension» αφορά την διάσταση στην οποία θα γίνει η διάσπαση αν είναι πολυδιάστατος ο πίνακας.

ARRAY_RESHAPE: Το directive αυτό συνδυάζει το array partitioning με το vertical array mapping, ώστε να δημιουργήσει έναν καινούριο πίνακα με λιγότερα στοιχεία αλλά μεγαλύτερες λέξεις. Επειδή είναι ένας συνδυασμός του array partitioning έχει τις ίδιες επιλογές με αυτό το directive.

CLOCK: Το vivado υποστηρίζει την χρήση πολλαπλών ρολογιών. Με το directive αυτό μπορεί να εφαρμοστεί ένα διαφορετικό ρολόι με διαφορετική περίοδο σε κάποια συνάρτηση. Σκοπός είναι να μειωθεί η περίοδος για την συνάρτηση σε περίπτωση που αυτή είναι πολύ μεγάλη για τις διεργασίες που γίνονται σε κάθε περίοδο.

DATAFLOW: Το συγκεκριμένο directive αναλύει την ροή δεδομένων μεταξύ βρόγχων ή συναρτήσεων ώστε να δημιουργήσει κανάλια προώθησης δεδομένων όπου αυτό είναι εφικτό, προσφέροντας την δυνατότητα βρόγχοι και συναρτήσεις να εκκινηθούν πριν παραχθούν τα

δεδομένα από του προηγούμενους, προσφέροντας έτσι ένα μεγαλύτερο επίπεδο παραλληλισμού βελτιώνοντας το throughput και μειώνοντας το latency. Οι εξαρτήσεις μεταξύ των δεδομένων όπως για παράδειγμα σημεία ανάγνωσης και εγγραφής δεδομένων σε έναν βρόγχο μπορούν να μειώσουν την απόδοση που προσφέρει το directive.

DEPENDENCE: Το directive αυτό χρησιμοποιείται για να προσδιορίσει εξαρτήσεις μεταξύ στοιχείων (μεταβλητές, πίνακες, δείκτες) στην ίδια επανάληψη ενός βρόγχου ή σε διαφορετικές επαναλήψεις ενός βρόγχου. Ο χρήστης μπορεί να ορίσει σε ποιες μεταβλητές υπάρχουν εξαρτήσεις τις οποίες το εργαλείο απέτυχε να εντοπίσει, καθώς και αν είναι εντός της ίδιας ή διαφορετικής επανάληψης του βρόγχου και προς ποια κατεύθυνση (ανάγνωση-εγγραφή, εγγραφή-ανάγνωση, εγγραφή-εγγραφή).

EXPRESSION_BALANCE: Κάποιες φορές οι ακολουθίες των πράξεων σε μία έκφραση δημιουργούν μια μακριά αλληλουχία πράξεων, γεγονός το οποίο μπορεί να αυξήσει το latency. Ενεργοποιώντας το directive αυτό, επιτρέπεται στο vivado να κάνει μια ανακατανομή των πράξεων ώστε να προκύψει μια μικρότερη αλληλουχία.

FUNCTION_INSTANTIATE: Το directive αυτό χρησιμοποιείται όταν υπάρχουν πολλές υλοποιήσεις για την ίδια συνάρτηση. Προεπιλεγμένα όλες οι υλοποιήσεις του ίδιου επιπέδου ιεραρχίας, χρησιμοποιούν την ίδια υλοποίηση RTL. Με το συγκεκριμένο directive μπορεί η κάθε υλοποίηση να χρησιμοποιεί ένα μοναδικό RTL επιτρέποντας έτσι πιο στοχευμένη βελτιστοποίηση.

INLINE: Με το συγκεκριμένο directive σταματάει το vivado να θεωρεί μία συνάρτηση ως ξεχωριστή οντότητα στην ιεραρχία. Με αυτόν τον τρόπο, σε μερικές περιπτώσεις οι πράξεις μέσα στην συνάρτηση βελτιστοποιούνται με γειτονικές πράξεις. Ο χρήστης μπορεί να επιλέξει την περιοχή στης οποίας τις συναρτήσεις θα εφαρμοστεί το directive, καθώς και το επίπεδο ιεραρχίας στο οποίο θα εφαρμοστεί.

INTERFACE: Το directive αυτό χρησιμοποιείται ώστε να αποδοθεί ο τρόπος με τον οποίον θα δημιουργηθούν οι RTL πόρτες κατά την διαδικασία της σύνθεσης, το οποίο

χρησιμοποιείται κυρίως για τον τρόπο επικοινωνίας των δεδομένων σε επίπεδο προγραμματισμού του ολοκληρωμένου.

LATENCY: Το vivado έχει ως στόχο πάντα το καλύτερο δυνατό αποτέλεσμα για το latency ενός αλγορίθμου. Το συγκεκριμένο directive μπορεί να ορίσει ως μέγιστο ή ελάχιστο όριο latency για ένα κομμάτι κώδικα. Αν το latency είναι μικρότερο από το ελάχιστο ή το μεγαλύτερο όριο τότε το τίθεται ίσο με αυτά. Αν είναι μεγαλύτερο από το ελάχιστο ή το μέγιστο τότε δεν υπάρχει κάποια ενέργεια, καθώς δεν μπορεί να γίνει κάποια βελτιστοποίηση παραπάνω από το συγκεκριμένο directive.

LOOP_FLATTEN: Το directive αυτό χρησιμοποιείται σε εμφωλευμένους βρόγχους ώστε να τους ενοποιήσει σε έναν μοναδικό βρόγχο. Στην υλοποίηση RTL η μετάβαση μεταξύ των βρόγχων κοστίζει μία περίοδο του ρολογιού, έτσι ενοποιώντας τους βρόγχους αυτή η καθυστέρηση εξαλείφεται. Το directive αυτό εφαρμόζεται στον εσωτερικό βρόγχο. Για να μπορέσει να εφαρμοστεί θα πρέπει οι εμφωλευμένοι βρόγχοι να είναι «Perfect nests» ή «Semi-perfect nests». Για να είναι Perfect loop nests θα πρέπει:

- Μόνο ο εσωτερικός βρόγχος να έχει σώμα.
- Να μην υπάρχει λογική συνθήκη μεταξύ των βρόγχων.
- Τα όρια όλων των βρόγχων να είναι σταθερές.

Για να είναι Semi-perfect loop nests θα πρέπει:

- Μόνο ο εσωτερικός βρόγχος να έχει σώμα.
- Να μην υπάρχει λογική συνθήκη μεταξύ των βρόγχων.
- Τα όρια του εξωτερικού βρόγχου να ορίζονται από μεταβλητή.

LOOP_MERGE: Σαυτό το directive ενώνονται όλοι οι βρόγχοι σε έναν. Εάν τα όρια των αρχικών βρόγχων είναι σταθερά θα πρέπει όλοι οι βρόγχοι να έχουν τα ίδια όρια, ενώ αν είναι μεταβλητά, χρησιμοποιείται για τον τελικό βρόγχο το μέγιστο όριο των αρχικών. Βρόγχοι με μεταβλητά και σταθερά όρια δεν μπορούν να συγχωνευτούν , καθώς και βρόχοι που περιέχουν δομές FIFO. Η τεχνική αυτή έχει ως αποτέλεσμα να μειώνονται οι παλμοί του ρολογιού που χρειάζονται για την εκτέλεση όλων των βρόγχων και κατ’ επέκταση και το

latency. Αυτό συμβαίνει επειδή εξοικονομούνται παλμοί στην μετάβαση από τον έλεγχο για επανάληψη, στο σώμα του βρόγχου.

LOOP_TRIPCOUNT: Το directive αυτό χρησιμοποιείται για να δώσει τον μέγιστο, τον ελάχιστο ή τον μέσο αριθμό επαναλήψεων ενός βρόγχου και συνεπώς είναι συνδεδεμένο με το latency του βρόγχου. Χρησιμοποιείται κυρίως σε βρόγχους των οποίων τα όρια είναι μεταβλητά για να οριστεί ένα σημείο αναφοράς στον αριθμό των επαναλήψεων.

OCCURRENCE: Το directive αυτό χρησιμοποιείται όταν έχει γίνει pipelining σε κάποια συνάρτηση ή κάποιον βρόγχο, ώστε να απομονώσει κάποια κομμάτια του κώδικα τα οποία θα εκτελούνται με χαμηλότερο ρυθμό και άρα μπορούν να γίνουν pipelined με έναν μικρότερο ρυθμό και να μοιραστούν με το top-level pipelining.

PIPELINE: Το directive αυτό ενεργοποιεί το pipelining εντός μίας συνάρτησης ή ενός βρόγχου. Κατά την διαδικασία του pipelining εισάγονται νέα δεδομένα(γίνονται καινούριες αναγνώσεις) και επεξεργάζονται, πριν τελειώσει η επεξεργασία και εγγραφή των προηγούμενων, όπου αυτό επιτρέπεται. Περιέχει την επιλογή «flushing» με την οποία περνάει σε επόμενο επίπεδο του pipelining όταν βρεθεί κάποιο stall στα νέα δεδομένα, καθώς και την επιλογή «loop rewinding» η οποία μπορεί να εφαρμοστεί μόνο σε βρόγχους και προκαλεί συνεχές pipelining μεταξύ των επαναλήψεων χωρίς να παύση μεταξύ τους. Για να μπορεί να βελτιώσει την απόδοση αυτή η επιλογή πρέπει να εφαρμοστεί σε έναν και μοναδικό βρόγχο ή σε perfect loop nest.

PROTOCOL: Το directive αυτό απενεργοποιεί την χρήση του ρολογιού σε ένα επιλεγμένο σημείο του κώδικα.

RESET: Το directive αυτό προσθέτει ή αφαιρεί σήματα reset σε στατικές ή καθολικές μεταβλητές.

RESOURCE: Με το directive αυτό χρησιμοποιείται για να οριστεί ποιος πυρήνας θα χρησιμοποιηθεί για την υλοποίηση μιας RTL. Ο πυρήνας μπορεί να είναι μια μεταβλητή, ένας πίνακας ή ένα όρισμα συνάρτησης. Δέχεται τρείς επιλογές, την «core» όπου ορίζεται ο τύπος του πυρήνα που θα χρησιμοποιηθεί, την «latency» όπου ορίζεται το latency του

πυρήνα, καθώς και το «port_map» το οποίο αντιστοιχεί τις θύρες του IP generationflow με τις θύρες του αντάπτορα.

STREAM: Από προεπιλογή του vivado, οι πίνακες υλοποιούνται ως δομές RAM. Όταν όμως η ανάγνωση και εγγραφή των δεδομένων στον πίνακα γίνεται με μία ακολουθιακή μορφή τότε είναι πιο αποδοτικό να γίνεται υλοποίηση του πίνακα ως FIFO και όχι ως RAM. Αυτό επιτυγχάνεται με τον συγκεκριμένο directive. Υπάρχουν ακόμη οι επιλογές «depth» και «dimension» στο συγκεκριμένο directive. Η πρώτη αλλάζει την προεπιλεγμένη τιμή του FIFO depth, ενώ η δεύτερη ορίζει σε πολυδιάστατους πίνακες, σε ποια διάσταση θα εφαρμοστεί το directive.

UNROLL: Το συγκεκριμένο directive εφαρμόζεται σε βρόγχους και δημιουργεί αντίγραφα του σώματός τους, τα οποία λειτουργούν παράλληλα. Έτσι μειώνεται ο αριθμός των επαναλήψεων του βρόγχου, με αντίτιμο το υλικό που θα πρέπει να δημιουργηθεί. Υπάρχει η επιλογή «unroll factor» με την οποία ρυθμίζεται ο αριθμός των αντιγράφων, η επιλογή «skip_exit_check» όπου εφαρμόζεται σε βρόγχους με σταθερό αριθμό επαναλήψεων ώστε να μην ελέγχεται η συνθήκη εξόδου, καθώς και η επιλογή «region», όπου μπορεί να καθοριστεί μια ολόκληρη περιοχή στης οποίας θα γίνει εφαρμογή του directive σε όλους τους βρόγχους.

Τα παραπάνω αποτελούν μια σειρά από directives που συνθέτουν την πλειοψηφία των directive του εργαλείου Vivado της Xilinx. Εφαρμόζονται σε διάφορα σημεία ενός κώδικα ανάλογα με τις ανάγκες και εμπλέκονται με την επεξεργασία και μεταφορά δεδομένων, καθώς και την διαχείριση μνήμης. Τα direcitves παίζουν καθοριστικό ρόλο καθώς επιτρέπουν τις αλλαγές σε επίπεδο συνδεσμολογίας υλικού χωρίς ο μηχανικός να πρέπει να έχει πλήρη γνώση της αρχιτεκτονικής σε επίπεδο RTL. Έχουν την μορφή εντολής η οποία πραγματοποιεί αλλαγές στην FPGA με σκοπό την βελτίωση της αποδοτικότητας μέσω ενός προκαθορισμένου και τυποποιημένου τρόπου που έχει να κάνει με την εντολή. Με αυτόν τον σκοπό χρησιμοποιούνται τα directives και στην συγκεκριμένη διπλωματική εργασία, ώστε να βελτιωθεί η απόδοση των υπό εξέταση νάνων.

Κεφάλαιο 4: Αρχιτεκτονική

Στο κεφάλαιο αυτό παρουσιάζεται η διαδικασία της βελτίωσης της απόδοσης κάποιου αλγορίθμου και συγκεκριμένα στην διπλωματική αυτή , των αλγορίθμων των δύο νάνων. Η διαδικασία ξεκινάει λαμβάνοντας τον κώδικα αναφοράς σε C/C++ και αναλύει όλα τα στάδια μέχρι την παραγωγή του RTL στην βελτιωμένη από πλευράς απόδοσης μορφή για μία FGPA.

Πιο συγκεκριμένα στο κεφάλαιο αυτό γίνεται περιγραφή του αλγορίθμου των νάνων, σχετικά με τον κώδικά τους και το πώς αυτοί λειτουργούν . Στην συνέχεια γίνεται ένας διαχωρισμός μεταξύ των directives τα οποία μπορούν ή όχι να εφαρμοστούν στους συγκεκριμένους αλγορίθμους με βάση την δομή των αλγορίθμων και σε συνδυασμό με τη φύση του εκάστοτε directive. Έπειτα από τα directives τα οποία με βάση την λειτουργία τους μπορούν να επιφέρουν αύξηση της αποδοτικότητας του αλγορίθμου γίνεται διαχωρισμός σε αυτά τα οποία επιδρούν με τον αναμενόμενο τρόπο αυξάνοντας την απόδοση και σε αυτά τα οποία δεν πετυχαίνουν τον στόχο. Τέλος για τα directives που επιφέρουν τα αναμενόμενα αποτελέσματα, δοκιμάζονται κάποιοι συνδυασμοί ώστε να αξιολογηθεί η συμπεριφορά στην ταυτόχρονη εφαρμογή τους, καθώς και να ανακαλυφθεί η καλύτερη δυνατή απόδοση με βάση την συμπεριφορά των directives μεταξύ τους. Τα directives έχουν επιλεχθεί και έχουν χρησιμοποιηθεί με βάση την λειτουργία τους η οποία περιγράφεται στα εγχειρίδια της Xilinx για το εργαλείο Vivado [5, 6, 7].

i. Sparse Matrix Multiplication

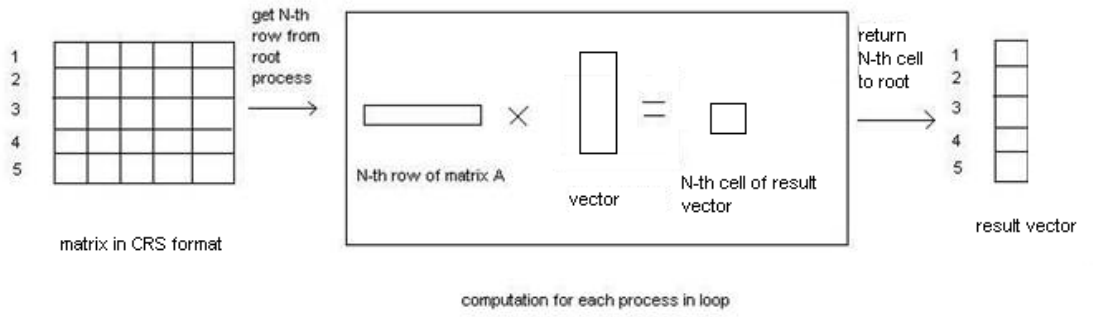
1. Περιγραφή του application – Dataflow του αλγόριθμου

Ο αλγόριθμος του Sparse-Matrix Multiplication σχετίζεται με τον πολλαπλασιασμό αραιών πινάκων, δηλαδή πινάκων που έχουν ως πλειοψηφία μηδενικά στοιχεία. Παραλλαγή του αλγορίθμου θα μπορούσε να χρησιμοποιηθεί και για την αφαίρεση ή των πολλαπλασιασμό τέτοιου είδους πινάκων. Ο συγκεκριμένος αλγόριθμος υποστηρίζει τέτοιου είδους πράξεις

μεταξύ διανύσματος-πίνακα, πίνακα-διανύσματος, διανύσματος-διανύσματος. Υπάρχουν διαφορετικές μορφές αποθήκευσης τέτοιου είδους πινάκων, ωστόσο στον κώδικα αναφοράς χρησιμοποιείται η μορφή αποθήκευσης συμπιεσμένων γραμμών(Compressed Row Storage).

Αλγόριθμος

Η αποθήκευση πινάκων σε μορφή συμπιεσμένων γραμμών γίνεται με την βοήθεια τριών υποπινάκων, οι οποίοι στον κώδικά αναφοράς αποτελούνται από τον val[] στον οποίο αποθηκεύονται τα μη μηδενικά στοιχεία του πίνακα, τον col_ind[] στον οποίον αποθηκεύονται αριθμοί οι οποίοι είναι οι δείκτες των στηλών στις οποίες βρίσκονται τα αντίστοιχα μη-μηδενικά στοιχεία, και του row_ptr[] στον οποίον αποθηκεύεται το πλήθος των μη-μηδενικών στοιχείων εώς την εκάστοτε γραμμή του πίνακα. Έτσι για κάθε γραμμή του πίνακα στον πρώτο βρόγχο, υπολογίζεται το σύνολο των μη-μηδενικών στοιχείων για την εκάστοτε γραμμή στον δεύτερο βρόγχο και τελικά υπολογίζεται το γινόμενο με το διάνυσμα πολλαπλασιασμού. Στον αλγόριθμο αναφοράς έχει επιλεχθεί ο πολλαπλασιασμός πίνακα-διανύσματος, όπου το διάνυσμα είναι ο πίνακας vector[] και το αποτέλεσμα αποθηκεύεται στον πίνακα result[].

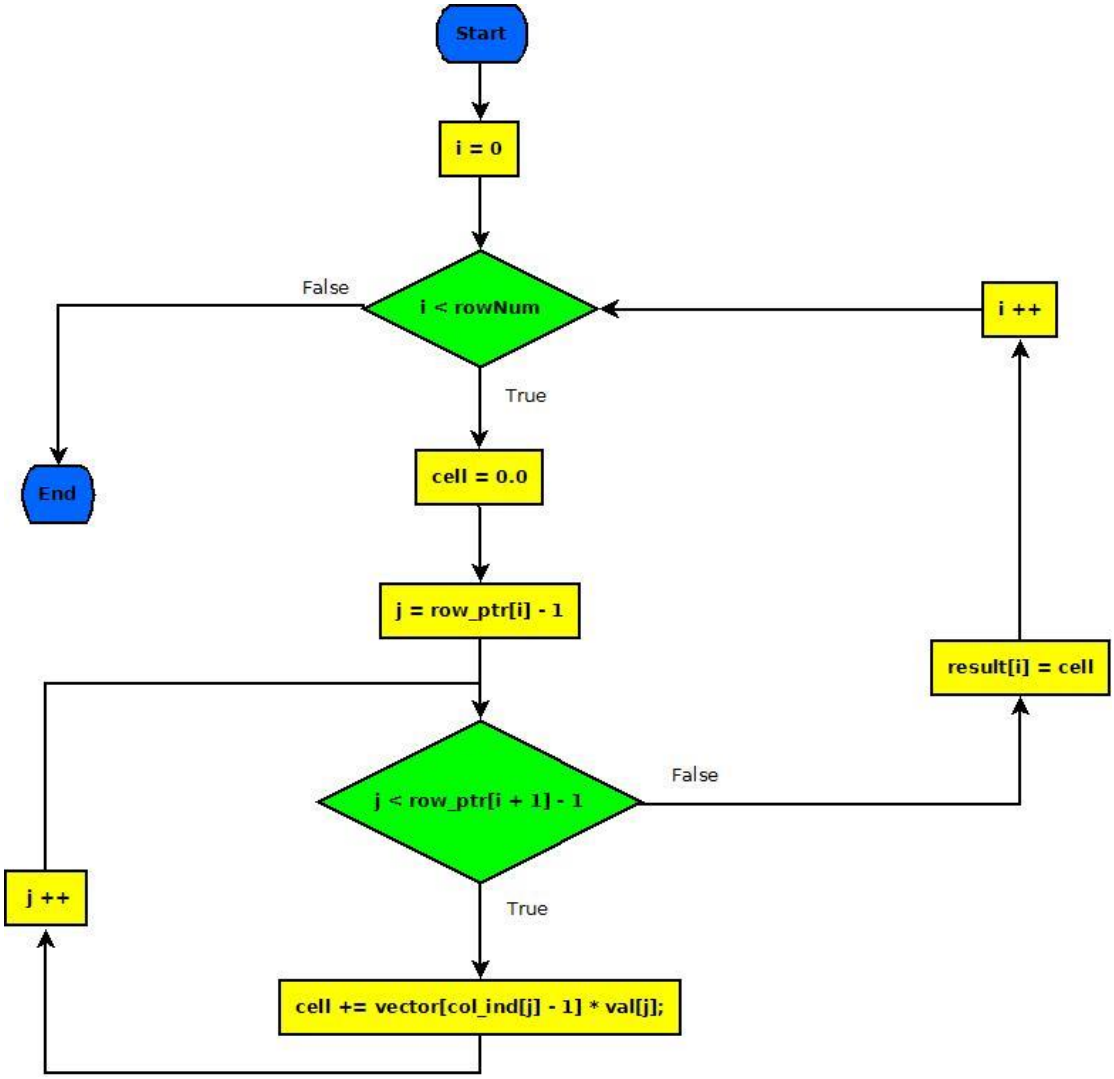


Εικόνα 4.1: Διαδικασία υπολογισμού στοιχείων στον βρόγχο

Παρατίθενται ο αλγόριθμος και το διάγραμμα ροής του αλγόριθμου.

```
void Solver(int val[N1], int col_ind[N1], int row_ptr[N2+1], int vector[N2],
        int rowNum, int result[N2]){



    int i;
    double cell;
    // Loop for number of rows in matrix
    OUTER_LOOP:for (i = 0; i < rowNum; i ++ )
    {

        cell = 0.0;
        // Loop for number of non-zero elements in current row
        INNER_LOOP:for (int j = row_ptr[i] - 1; j < row_ptr[i + 1] - 1 ; j ++ )
        {

            cell += vector[col_ind[j] - 1] * val[j];

        }


        result[i] = cell;

    }



}
```

Εικόνα 4.2: Κώδικας Πολλαπλασιασμού Αραιών Συμπιεσμένων Πινάκων



Εικόνα 4.3: Διάγραμμα Ροής αλγορίθμου.

2. Αλγόριθμος χωρίς directives

Αρχικά πραγματοποιήθηκε μετατροπή από τον κώδικα αναφοράς ο οποίος ήταν στην γλώσσα C++, στο vivado χωρίς καμία χρήση directives. Έγιναν μετατροπές από τον κώδικα αναφοράς στην δυναμική δέσμευση μνήμης για τους πίνακες η οποία δεν υποστηρίζεται στο vivado και έτσι το μέγεθος των πινάκων ορίζεται εξ ’αρχής στις δηλώσεις μεταβλητών. Δημιουργήθηκε ένα αρχείο «SA_Solver.c» στο οποίο εφαρμόζονται αργότερα τα directives, ένα αρχείο «SA_Solver_test_bench.c» το οποίο χρησιμοποιείται για να δίνει τα δεδομένα στο «SA_Solver.c» καθώς και ένα αρχείο «SA_Solver.h» στο οποίο γίνεται ορισμός της

συνάρτησης «SA_Solver.c» καθώς και των εισόδων-εξόδων αυτής. Τα δεδομένα λαμβάνονται από το αρχείο «smallcontent.txt» και αποθηκεύονται στο «result.txt»

Αρχικά το vivado δεν μπορεί να υπολογίσει το latency χωρίς directives, καθώς τα όρια των βρόγχων στον «SA_Solver.c» είναι μεταβλητά. Έτσι δεν μπορεί να υπολογίσει το πόσο υλικό πρέπει να δημιουργηθεί και κατά συνέπεια πόσοι κύκλοι απαιτούνται για τον υπολογισμό του αποτελέσματος, κάτι το οποίο αντιμετωπίζεται στην συνέχεια.

Σκοπός της διαδικασίας είναι η βελτίωση του χρόνου εκτέλεσης του αλγορίθμου, το οποίο εξαρτάται από το χρόνο του κάθε παλμού του ρολογιού καθώς και από τον αριθμό κύκλων του ρολογιού ως την παραγωγή αποτελεσμάτων από τον αλγόριθμο, το οποίο ονομάζεται «latency». Αρχικά απορρίπτονται τα directives τα οποία εξ' ορισμού δεν μπορούν να χρησιμοποιηθούν στον συγκεκριμένο αλγόριθμο λόγω της λειτουργίας τους και στα οποία δικαιολογείται την απόρριψή τους.

OCCURRENCE: Στον συγκεκριμένο αλγόριθμο δεν μπορεί να χρησιμοποιηθεί καθώς δεν υπάρχουν διαφορετικού επιπέδου pipelined κομμάτια, τα οποία να έχουν διαφορετικό ρυθμό.

RESET: Δεν μπορεί να χρησιμοποιηθεί στον συγκεκριμένο αλγόριθμο καθώς δεν υπάρχουν ούτε στατικές ούτε καθολικές μεταβλητές.

FUNCTION_INSTANTIATE: Δεν μπορεί να χρησιμοποιηθεί στον συγκεκριμένο αλγόριθμο καθώς δεν υπάρχουν πολλαπλές υλοποιήσεις για την ίδια συνάρτηση.

LOOP_MERGE: Στον εξεταζόμενο αλγόριθμο δεν μπορεί να εφαρμοστεί, καθώς απαιτεί βρόγχους οι οποίοι δεν είναι εμφωλευμένοι μεταξύ τους ενώ στον αλγόριθμο υπάρχει μόνο εμφωλευμένος βρόγχος.

INLINE: Στον συγκεκριμένο αλγόριθμο υπάρχει μία συνάρτηση και δεν υπάρχει ιεραρχία επιπέδων, συνεπώς δεν μπορεί να βελτιώσει την απόδοση του αλγορίθμου.

ALLOCATION: Στον συγκεκριμένο αλγόριθμο υπάρχει μόνο μία υλοποίηση της συνάρτησης Solver(), καθώς και ένα μοναδικό πολλαπλασιασμού και ένα μοναδικό σημείο

πρόσθεσης. Συνεπώς το συγκεκριμένο directive δεν μπορεί να χρησιμοποιηθεί για εξοικονόμηση πόρων

3. Πειραματική διαδικασία εφαρμογής directives

Στην συνέχεια παρουσιάζονται τα directives τα οποία μπορούν να εφαρμοστούν τεχνικά στον αλγόριθμο ωστόσο η εφαρμογή τους δεν παρουσιάζει καμία μεταβολή στην απόδοση για τους λόγους που αναλύονται. Σ' αυτόν τον αλγόριθμο όσα directives επηρεάζουν θετικά την απόδοση διατηρούνται και τα επόμενα εφαρμόζονται επιπλέον. Αυτό γίνεται με σκοπό να διαπιστωθεί αν κάποιος λάθος συνδυασμός directives μπορεί να δώσει οδηγήσει στην μείωση της απόδοσης.

ARRAY_MAP: Το συγκεκριμένο directive συνδυάζοντας τους μικρότερους πίνακες σε έναν μεγαλύτερο δεν επηρεάζει τον ρυθμό αναγνώσεων-εγγραφών αφού αυτές γίνονται μία φορά σε κάθε επανάληψη του βρόγχου και συνεπώς δεν μπορεί να επηρεάσει τον τρόπο των υπολογισμών και κατ' επέκταση και την απόδοση.

DEPENDENCE: Το εργαλείο Vivado γενικά εντοπίζει τις εξαρτήσεις μεταξύ των μεταβλητών αυτόματα. Στον συγκεκριμένο αλγόριθμο εκτελείται μία γραμμή κώδικα στην οποία όλες οι μεταβλητές είναι διαφορετικές, συνεπώς δεν μπορεί να υπάρχει κάποιου είδους εξάρτησης μεταβλητών μεταξύ των επαναλήψεων και η απόδοση δεν επηρεάζεται.

EXPRESSION_BALANCE: Στον συγκεκριμένο αλγόριθμο εκτελούνται πράξεις μέσα σε μία γραμμή κώδικα και δεν υπάρχουν εξαρτήσεις μεταβλητών ώστε να δημιουργηθεί κάποια μεγάλη αλληλουχία πράξεων. Συνεπώς αυτό το directive δεν επηρεάζει τον αλγόριθμο.

LATENCY: Το εργαλείο Vivado έχει ως στόχο πάντα το καλύτερο δυνατό αποτέλεσμα για το latency ενός αλγορίθμου. Στον συγκεκριμένο αλγόριθμο δεν υπάρχουν περιττοί ή υπολογιστικά ανενεργοί κύκλοι, συνεπώς ακόμα και αν οριστεί μέγιστο όριο για το latency δεν μπορεί να υπάρξει κάποια βελτίωση.

Ακολουθούν τα directives τα οποία είχαν επίπτωση στο latency του αλγορίθμου, είτε θετική οπότε είναι υποψήφια προς χρήση στην τελική φάση συνδυασμού, είτε αρνητική οπότε και απορρίπτονται.

LOOP_TRIPCOUNT: Στον αλγόριθμο τίθεται πρώτο το συγκεκριμένο directive για να ξεπεραστεί το πρόβλημα των μεταβλητών ορίων των βρόγχων ώστε να υπάρχει μία τιμή αναφοράς για το latency. Παρατηρείται πως το άνω όριο του «OUTER_LOOP» μπορεί να τεθεί ίσο με στο 362 καθώς η μεταβλητή «rowNum» του ορίου του, είναι ο αριθμός των γραμμών του πίνακα τιμών από τον οποίον εισάγονται τα δεδομένα, και είναι γνωστό ότι έχει 362 γραμμές. Στον «INNER_LOOP» όπου στο όριό του εμπλέκεται ο row_ptr[] τίθεται επίσης όριο στο 362 καθώς ο πίνακας είναι τετραγωνικός. Αυτό εξάγει ένα latency το οποίο χρησιμοποιείται ως σημείο αναφοράς.

CLOCK: Σκοπός του συγκεκριμένου directive είναι να μειωθεί η περίοδος για την συνάρτηση σε περίπτωση που αυτή είναι πολύ μεγάλη για τις διεργασίες που γίνονται σε κάθε περίοδο. Στον αλγόριθμο αρχικά υπήρχε ένα ρολόι για όλες τις συναρτήσεις με περίοδο T1=10ns και με latency=1.983.046, υπολογίζοντας προκύπτει πως για την εκτέλεσή του αλγορίθμου χρειάζεται t1=1.983.046*10ns =19.830.460ns. Χρησιμοποιώντας το συγκεκριμένο directive προστέθηκε ένα καινούριο ρολόι στην συνάρτηση Solver() με περίοδο T2=7ns εξάγοντας αυξημένο latency αλλά μειωμένο χρόνο εκτέλεσης του αλγορίθμου. Στην συνέχεια μειώθηκε ακόμη περισσότερο η περίοδος του ρολογιού σε T3=6ns εξάγοντας latency και πάλι αυξημένο αλλά ακόμη πιο μειωμένο χρόνο εκτέλεσης του. Τελικά η μείωση φτάνει σε περίοδο T5= 4ns ακόμη πιο μειωμένο χρόνο εκτέλεσης του αλγορίθμου. Συνεχίζοντας την μείωση παρατηρείται πως ο εκτιμώμενος χρόνος του ρολογιού(Estimated = 3.14ns) είναι μεγαλύτερος από τον στόχο(Target = 3.00ns), γεγονός το οποίο θα δημιουργήσει πρόβλημα στην FPGA και απορρίπτεται. Παρατηρείται πως η περίοδος ρολογιού ήταν διπλάσια από αυτήν που χρειάζεται για να εκτελεστούν οι πράξεις του αλγορίθμου στο κύκλωμα και έτσι μειώνοντάς την μειώνουμε και τον χρόνο εκτέλεσης.

DATAFLOW: Το συγκεκριμένο directive αναλύει την ροή δεδομένων μεταξύ βρόγχων ή συναρτήσεων ώστε να δημιουργήσει κανάλια προώθησης δεδομένων όπου αυτό είναι εφικτό, προσφέροντας έτσι ένα μεγαλύτερο επίπεδο παραλληλισμού βελτιώνοντας το throughput και μειώνοντας το latency. Οι εξαρτήσεις μεταξύ των δεδομένων όπως για παράδειγμα σημεία ανάγνωσης και εγγραφής δεδομένων σε έναν βρόγχο μπορούν να μειώσουν την απόδοση που προσφέρει το directive. Στον αλγόριθμο που μελετάται εφαρμόστηκε στο «INNER_LOOP» της Solver() χωρίς κάποιο αποτέλεσμα επειδή πιθανόν να μην υπήρχε κάποια εξάρτηση μεταξύ των δεδομένων, στο «OUTER_LOOP» της Solver() ανεβάζοντας το latency, προφανώς λόγω κάποια διένεξης με την αρχικοποίηση της μεταβλητής «cell» , της εγγραφής στον πίνακα result[] ή των δεδομένων που παράγονται από τον εμφωλευμένο βρόγχο.

LOOP_FLATTEN: Στον συγκεκριμένο αλγόριθμο χωρίς directives και οι δύο βρόγχοι είχαν μεταβλητά όρια οπότε δεν μπορούσαν να κατηγοριοποιηθούν σε κάποια από τις ομάδες “Perfect” ή “Semi-Perfect nested loops”. Εφαρμόζοντας όμως το «loop tripcount» τέθηκε άνω όριο και στους δύο βρόγχους και μπορεί πλέον να θεωρηθεί ότι τα άκρα τους είναι σταθερά και ότι είναι perfect nests. Στον αλγόριθμο εφαρμόστηκε το directive στον «INNER_LOOP» χωρίς κάποιο αποτέλεσμα τελικά όμως, επειδή το Vivado δεν μπόρεσε λόγω του Tripcount να εκμεταλλευτεί την ενοποίηση των ελέγχων των συνθηκών των βρόγχων στο οποίο στηρίζεται το συγκεκριμένο directive.

PIPELINE: Στον συγκεκριμένο αλγόριθμο εφαρμόστηκε αρχικά το pipeline στον «OUTER_LOOP» της συνάρτησης Solver(), χωρίς κάποιο αποτέλεσμα επειδή δεν μπόρεσε να γίνει κάποιο pipline μεταξύ των μεταβλητών του βρόγχου αυτού σε συνδυασμό με τα δεδομένα που παρήγαγε ο «INNER_LOOP». Στην συνέχεια αφαιρέθηκε από τον «OUTER_LOOP» και προστέθηκε στον «INNER_LOOP» της ίδιας συνάρτησης. Εκεί προέκυψε latency με βελτιωμένη απόδοση σε μεγάλο βαθμό. Ενεργοποιώντας την επιλογή «flushing» του directive, το vivado ενεργοποιεί την εισαγωγή δεδομένων σε χαμηλότερα επίπεδα του pipelining όταν βρεθεί κάποιο stall. Ενεργοποιώντας την επιλογή εξάγονται τα ίδια αποτελέσματα με πριν, γεγονός που μας επιτρέπει να γίνει η υπόθεση σύμφωνα και με την δομή του κώδικα, πως δεν υπάρχει κάποιο stall. Στην συνέχεια απενεργοποιείται το «flushing» και ενεργοποιείται το «rewinding» στον «inner_loop». Αυτό είναι μια επιλογή η οποία επιτρέπει συνεχόμενο pipelining βρόγχου χωρίς διακοπή μεταξύ των επαναλήψεων, έχει νόημα ωστόσο να εφαρμοστεί σε μη εμφωλευμένους βρόγχους ή σε perfect loop nests. Στον συγκεκριμένο αλγόριθμο θα το εφαρμοστεί, καθώς όπως αναφέρθηκε οι εμφωλευμένοι βρόγχοι θεωρούνται perfect nests μετά την εφαρμογή του loop tripcount. Εφαρμόζοντάς το

όμως εξάγεται αυξημένο latency γεγονός που οφείλεται και πάλι στο ότι οι εμφωλευμένοι βρόχοι έχουν μετατραπεί σε perfect-nested με την χρήση του tripcount.

STREAM: Από προεπιλογή του vivado, οι πίνακες υλοποιούνται ως δομές RAM. Όταν όμως η ανάγνωση και εγγραφή των δεδομένων στον πίνακα γίνεται με μία ακολουθιακή μορφή τότε είναι πιο αποδοτικό να γίνεται υλοποίηση του πίνακα ως FIFO και όχι ως RAM. Αυτό επιτυγχάνεται με τον συγκεκριμένο directive. Στον εξεταζόμενο αλγόριθμο οι πίνακες είναι μονοδιάστατοι, συνεπώς δεν υπάρχει νόημα στην επιλογή «dimension». Έγινε δοκιμή σε τέσσερις πίνακες της συνάρτησης Solver() όπως παρουσιάζεται στη συνέχεια. Στην αρχή γίνεται δοκιμή στον πίνακα val[] όπου φαίνεται πως αυξάνεται το latency καθώς και το estimated του ρολογιού σε μη ανεκτά επίπεδα, αφού estimated+uncertainty>Target. Επίσης είναι φανερό πως ο πίνακας val[] υλοποιείται πλέον ως FIFO γεγονός που προκάλεσε την αύξηση του latency και την αύξηση του estimated του ρολογιού αφού αναγκάστηκε να γίνει η προσπέλαση του πίνακα σε ακολουθιακή μορφή.

Στην συνέχεια αφαιρείται το directive από τον val[] και το προστίθεται στον πίνακα row_ptr[] όπου φαίνεται πως μειώνεται το latency, όμως το estimated του ρολογιού αυξάνεται σε μη ανεκτά επίπεδα πάλι, αφού estimated+uncertainty>Target. Επίσης είναι φανερό πως ο πίνακας row_ptr[] υλοποιείται πλέον ως FIFO γεγονός που προκάλεσε την αύξηση του latency και την αύξηση του estimated του ρολογιού αφού αναγκάστηκε να γίνει η προσπέλαση του πίνακα σε ακολουθιακή μορφή.

Στην συνέχεια αφαιρείται το directive από τον row_ptr[] και προστίθεται στον πίνακα col_ind[] όπου και πάλι μειώνεται το latency, όμως το estimated του ρολογιού αυξάνεται σε μη ανεκτά επίπεδα πάλι, αφού estimated+uncertainty>Target. Και πάλι ο πίνακας col_ind[] υλοποιείται πλέον ως FIFO γεγονός που προκάλεσε την αύξηση του latency και την αύξηση του estimated του ρολογιού αφού αναγκάστηκε να γίνει η προσπέλαση του πίνακα σε ακολουθιακή μορφή.

Στην συνέχεια αφαιρείται το directive από τον col_ind[] και προστίθεται στον πίνακα vector[] όπου το latency παραμένει σταθερό καθώς και πως το estimated του ρολογιού αυξάνεται σε μη ανεκτά επίπεδα πάλι, αφού estimated+uncertainty>Target. Επίσης φαίνεται πως ο πίνακας vector[] υλοποιείται πλέον ως FIFO, το οποίο προκάλεσε την αύξηση του latency προφανώς λόγω μη ακολουθιακής προσπέλασης των δεδομένων του.

UNROLL: Αρχικά εφαρμόστηκε στο «OUTER_LOOP» της συνάρτησης Solver(), με unroll_factor ίσο με 2, όπου παρατηρείται πως το latency μειώθηκε, όμως το estimated του

ρολογιού είναι πάνω από το target γεγονός που κάνει απαγορευτική τη χρήση του directive. Το estimated αυξήθηκε τόσο γιατί παράχθηκε το αντίγραφο ενός ολόκληρου βρόγχου(του «INNER_LOOP») ο οποίος εκτελούνταν δύο φορές πλέον σε κάθε περίοδο. Στην συνέχεια αφαιρέθηκε το directive από το «OUTER_LOOP» και προστέθηκε στο «INNER_LOOP», με unroll_factor ίσο με 2, εξάγοντας μία μείωση του latency, χωρίς αύξηση του estimated του ρολογιού οπότε και θα χρησιμοποιηθεί αυτό το directive στην τελική φάση βελτιστοποίησης. Η αύξηση της απόδοσης οφείλεται στο ότι πλέον υπολογίζονται σε κάθε περίοδο δύο στοιχεία του πίνακα με κόστος την παραγωγή υλικού.

ARRAY_PARTITION: Το directive αυτό διασπά έναν πίνακα σε μικρότερους πίνακες ή σε ανεξάρτητα στοιχεία. Στον εξεταζόμενο αλγόριθμο εφαρμόστηκε στους τέσσερις πίνακες δεδομένων της συνάρτησης Solver(). Δοκιμάστηκαν όλοι οι τύποι διάσπασης με επιλογή «factor» ίσο με 2 ώστε να υπάρχει μια εικόνα του πως συμπεριφέρεται ο αλγόριθμος με το συγκεκριμένο directive.

Έτσι εφαρμόζεται για τον πίνακα val[] με «type» ίσο με block και «factor» ίσο 2 όπου παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Για τον πίνακα col_ind[] με «type» ίσο με block και «factor» ίσο με 2 όπου παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Για τον πίνακα row_ptr[] με «type» ίσο με block και «factor» ίσο με 2 όπου το latency αυξήθηκε σε όλες τις περιπτώσεις, δηλαδή εφαρμόζοντας τύπο block, cyclic και complete τόσο στην πρώτη όσο και στην δεύτερη διάσταση.

Για τον πίνακα vector[] με «type» ίσο με block και «factor» ίσο με 2 όπου παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Η αύξηση του latency πιθανόν να προκλήθηκε λόγω αύξησης των πινάκων και κατ' επέκταση της αλλαγής διευθυνσιοδότησης για την προσπέλαση των στοιχείων τους.

ARRAY_RESHAPE: Το directive αυτό συνδυάζει το array partitioning με το vertical array mapping. Καθώς είναι ένας συνδυασμός του array partitioning έχει τις ίδιες επιλογές με εκείνο το directive. Έτσι εξετάστηκαν οι ίδιες περιπτώσεις για να δούμε πως επηρεάζει το συγκεκριμένο directive την αποδοτικότητα του αλγορίθμου.

Έτσι για τον πίνακα val[] με «type» ίσο με block και «factor» ίσο με 2 όπου παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε λόγω error, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Για τον πίνακα col_ind[] με «type» ίσο με block και «factor» ίσο με 2 όπου παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε λόγω error, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Για τον πίνακα row_ptr[] με «type» ίσο με block και «factor» ίσο με 2 παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε λόγω error, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Για τον πίνακα vector[] με «type» ίσο με block και «factor» ίσο με 2 παρατηρείται αύξηση του latency. Στην συνέχεια εφαρμόστηκε «type» ίσο με complete το οποίο δεν ολοκληρώθηκε λόγω error, καθώς και «type» ίσο με cyclic με «factor» ίσο με 2 όπου δεν παρατηρήθηκε αλλαγή σε σχέση με πριν εφαρμοστεί το directive.

Η αύξηση του latency πιθανόν να προκλήθηκε λόγω αύξησης των πινάκων και κατ' επέκταση της αλλαγής διευθυνσιοδότησης για την προσπέλαση των στοιχείων τους και σ' αυτήν την περίπτωση, καθώς το συγκεκριμένο directive χρησιμοποιεί και «array_partiotion»

Στην συνέχεια γίνεται μια δοκιμή για τα directives που στην πρώτη φάση επηρέαζαν την αποδοτικότητα του αλγορίθμου αρνητικά. Αυτό έγινε για να γίνει ταυτοποιηθεί πως όντως την επηρεάζουν αρνητικά και ότι μπορεί κάποιος λάθος συνδυασμός να δώσει αρνητικά αποτελέσματα. Τα directives αυτά που εξετάστηκαν ξανά είναι τα dataflow, loop_flatten (δεν παρουσίασε καμία μεταβολή της αποδοτικότητας στη πρώτη φάση, αλλά είναι σχεδιασμένο για εμφωλευμένους βρόγχους), array_partition και array_reshape .

Για να υπάρχει ένα μέτρο σύγκρισης αρχικά εφαρμόστηκε «loop_tripcount» directive στο «INNER_LOOP» και «OUTER_LOOP» με μέγιστο ρυθμό επαναλήψεων ίσο με 362 για λόγους που εξηγήθηκαν παραπάνω.

Στην συνέχεια εφαρμόζεται ξανά «flatten_loop» directive σε «INNER_LOOP» και «OUTER_LOOP» χωρίς κάποια μεταβολή στην απόδοση. Έτσι αποκλείεται τελικά αυτό το directive και εφαρμόζεται το «Dataflow» directive στο “INNER_LOOP» όπου προκύπτει error message: “Control flows are not supported in dataflow INNER_LOOP of function ‘Solver’ ”. Στην συνέχεια αφαιρείται το directive από το «INNER_LOOP» και προστίθεται στο «OUTER_LOOP» όπου προκύπτει latency αυξημένο σε σχέση με πριν, συνεπώς απορρίπτεται και αυτό το directive τελικά .

Έπειτα εφαρμόστηκε «array_partition» σε κάθε έναν από τους πίνακες val[], col_ind[], row_ptr[], vector[] ξεχωριστά, με «type» ίσο με block και factor ίσο με 2 όπου για όλες τις περιπτώσεις το latency ήταν αυξημένο και συνεπώς η απόδοση μειωμένη.

Παρατηρούμε πως το latency είναι αυξημένο σε όλες τις περιπτώσεις, συνεπώς απορρίπτεται και αυτό το directive.

Στην συνέχεια εφαρμόστηκε «array_reshape» με την ίδια μέθοδο και τις ίδιες μεταβλητές στους ίδιους πίνακες όπου για όλες τις περιπτώσεις το latency ήταν αυξημένο και άρα η απόδοση μειωμένη.

Συνεπώς απορρίφθηκαν όλα τα directives της επανεξέτασης και έτσι αυτά που απέμειναν για την τελική φάση είναι τα «loop_tripcount», «clock», «pipline», «stream» και «unroll». Το «loop_tripcount» χρησιμοποιείται σε όλες τις περιπτώσεις, ώστε να απαλειφθούν τα ερωτηματικά από τους χρόνους και να υπάρχει ένα σημείο αναφοράς

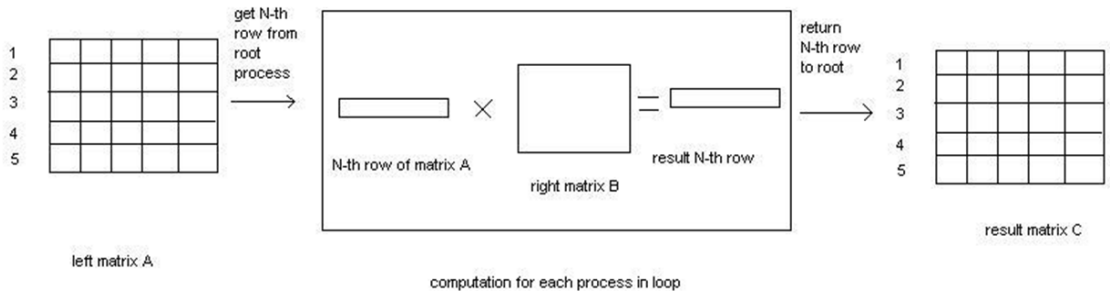
ii. Dense-Matrix Multiplication

1. Περιγραφή του application – Dataflow του αλγόριθμου

Ο αλγόριθμος υπολογίζει τον πολλαπλασιασμό κοινών δομών πινάκων(διάνυσμα-πίνακας, πίνακας-διάνυσμα, διάνυσμα - διάνυσμα, πίνακας-πίνακας) χωρίς την χρήση κάποιου είδους δείκτη, όπως για παράδειγμα πίνακες με μη μηδενικά στοιχεία, τριγωνικοί πίνακες και πίνακες ζωνών. Στον κώδικα αναφοράς πραγματοποιείται πολλαπλασιασμός δύο τετραγωνικών πινάκων ίσων διαστάσεων.

Αλγόριθμος

Για την πραγματοποίηση του πολλαπλασιασμού χρησιμοποιούνται τρεις βρόγχοι, ένας για την προσπέλαση του αριθμού των γραμμών του αριστερού πίνακα, ένας για την προσπέλαση του αριθμού των στηλών στον δεξιό πίνακα, καθώς και ένας για την προσπέλαση του αριθμού στηλών του πρώτου πίνακα-γραμμών του δεύτερου πίνακα που είναι ίσος.



Εικόνα 4.4: Διαδικασία υπολογισμού στοιχείων στον βρόγχο

Πραγματοποιείται μία αναστροφή πίνακα πριν από τους βρόγχους, η οποία είχε ως σκοπό να τονίσει κάποια θέματα σχετικά με βελτιστοποιήσεις του μεταγλωτιστή που απασχολούσαν τους συγγραφείς του κώδικα αναφοράς. Θα μπορούσε να είχε παραληφθεί η αναστροφή χρησιμοποιώντας στον πιο εσωτερικό βρόγχο την γραμμή κώδικα:

```
result[i][j]  = result[i][j] + lgrid[i][k] * rgrid[k][j];
```

Αντί για την:

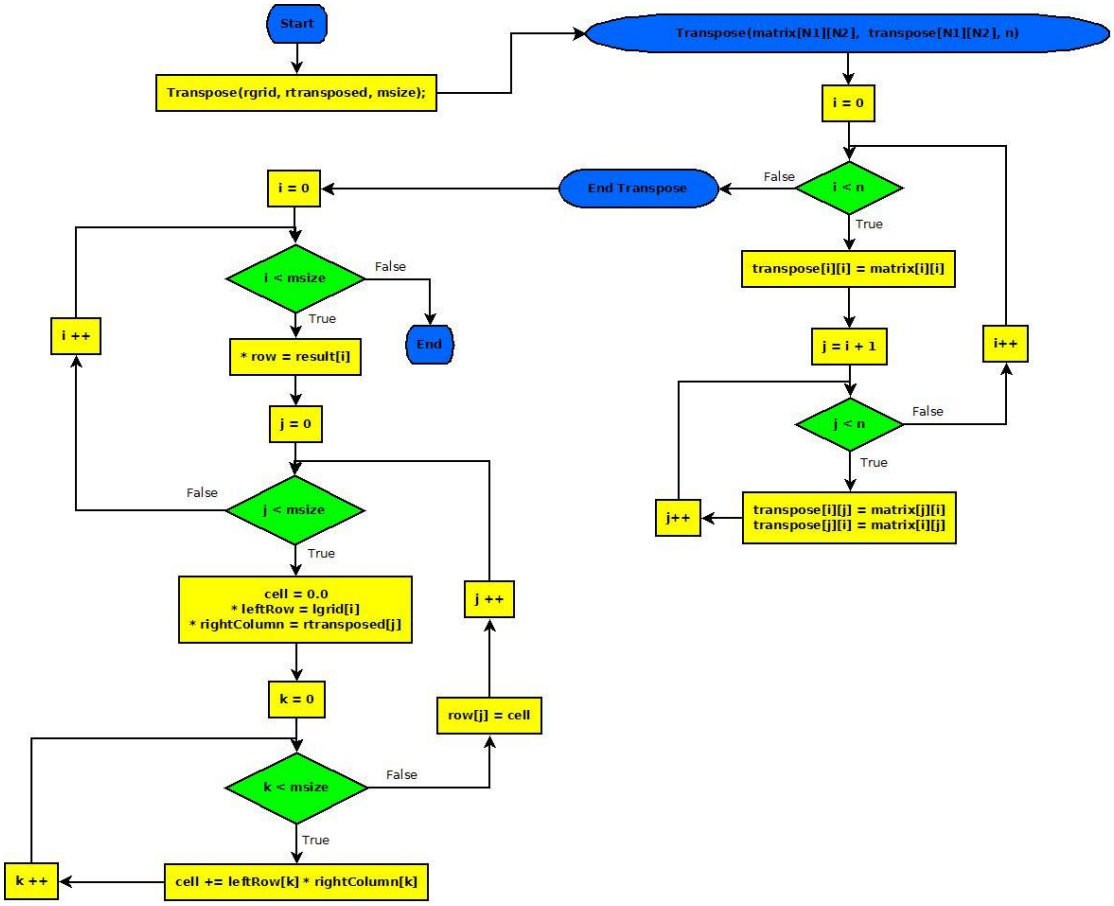
```
Cell += lefRow[k] * rightColumn[k];
```

Ωστόσο επιλέχθηκε να διατηρηθεί για να μην γίνουν παρεμβάσεις στον κώδικα αναφοράς οι οποίες δεν σχετίζονται με την αποδοτικότητα από directives, καθώς και για να υπάρχει δευτερεύουσα συνάρτηση στον αλγόριθμο προς πειραματισμό και εξαγωγή συμπερασμάτων.

Παρατίθενται ο αλγόριθμος και το διάγραμμα ροής του αλγόριθμου.

```
int Transpose(int matrix[N1][N2], int transpose[N1][N2], int n)
{
  LOOP4: for (int i = 0; i < n; i++)
    {
        transpose[i][i] = matrix[i][i];
    LOOP5: for (int j = i + 1; j < n; j++)
        {
            transpose[i][j] = matrix[j][i];
            transpose[j][i] = matrix[i][j];
        }
    }

return(0);
}


// Serial based method for matrix-matrix multiplication.
void Solver(int msize, int lgrid[N1][N2], int rgrid[N1][N2], int result[N1][N2])
{

    double rtransposed[229][229];


    Transpose(rgrid, rtransposed, msize);


     // Loop for number of rows in left matrix.
    LOOP1: for (int i = 0; i < msize; i ++ )
    {
      double* row = result[i];
       int k;

       // Loop for number of columns in right matrix.
       LOOP2: for (int j = 0; j < msize; j ++ )
        {
            double cell = 0.0;
            double* leftRow = lgrid[i];
            double* rightColumn = rtransposed[j];


           // Loop for number of columns in left matrix and rows in right matrix.
          LOOP3:   for (int k = 0; k < msize; k ++ )
            {
                cell += leftRow[k] * rightColumn[k];


            }
            row[j] = cell;
        }
    }

}
```

Εικόνα 4.5: Κώδικας Πολλαπλασιασμού Πυκνών Πινάκων



Εικόνα 4.6: Διάγραμμα Ροής αλγορίθμου.

2. Αλγόριθμος χωρίς directive

Η διαδικασία που ακολουθήθηκε στον δεύτερο αλγόριθμο είναι η ίδια με την διαδικασία του πρώτου αλγορίθμου. Αρχικά έγινε μετατροπή του κώδικα αναφοράς, οποίος ήταν στην γλώσσα προγραμματισμού C++, σε κώδικα αποδεκτό από το vivado. Η μετατροπή αυτή περιελάβανε και πάλι την απομάκρυνση τμημάτων κώδικα που είχαν να κάνουν με δυναμική δέσμευση μνήμης και τον ορισμό του μεγέθους των πινάκων από την αρχή στις δηλώσεις μεταβλητών. Δημιουργήθηκε ένα αρχείο «DLA_Solver.c» στο οποίο εφαρμόζονται αργότερα τα directives, ένα αρχείο «DLA_Solver_test.c» το οποίο χρησιμοποιείται για να δίνει τα δεδομένα στο «DLA_Solver.c» καθώς και ένα αρχείο «DLA_Solver.h» στο οποίο γίνεται ορισμός

της συνάρτησης «DLA_Solver.c» καθώς και των εισόδων-εξόδων αυτής. Τα δεδομένα λαμβάνονται από το αρχείο «smallcontent.txt» και αποθηκεύονται στο «result.txt»

Το vivado αρχικά δεν μπορεί να υπολογίσει το latency χωρίς directives, καθώς τα όρια των βρόγχων στον «DLA_Solver.c» είναι μεταβλητά, κάτι το οποίο θα πρέπει να λυθεί στην συνέχεια όπως και στον πρώτο αλγόριθμο.

Πριν γίνει η εφαρμογή των πρώτων directives που έχουν να κάνουν με τους βρόγχους, γίνεται δοκιμή του directive «clock» ώστε να βρεθεί μέχρι ποιο σημείο μπορεί να μειωθεί η περίοδος του ρολογιού πριν προστεθεί οποιοδήποτε άλλο directive. Κατόπιν δοκιμών εξάγεται το συμπέρασμα πως η περίοδος του ρολογιού όταν δεν υπάρχουν directives μπορεί να μειωθεί μέχρι T=4ns , καθώς αν μειωθεί περαιτέρω ισχύει ότι Estimated+Uncertainty>Target το οποίο δεν είναι αποδεκτό.

3. Πειραματική διαδικασία εφαρμογής directives

Σκοπός της διαδικασίας είναι η βελτίωση του χρόνου εκτέλεσης του αλγορίθμου κάτι το οποίο εξαρτάται από το χρόνο του κάθε παλμού του ρολογιού, καθώς και από τον αριθμό κύκλων του ρολογιού ως την παραγωγή αποτελεσμάτων από τον αλγόριθμο, το οποίο ονομάζεται latency. Στο παρόν κεφάλαιο γίνεται μια διαλογή των directives που μπορούν να εφαρμοστούν και έχουν κάποιο αποτέλεσμα στον συγκεκριμένο αλγόριθμο, ενώ στο επόμενο κεφάλαιο γίνονται δοκιμές για να βρεθεί ο καλύτερος δυνατός συνδυασμός των directives που απέμειναν από την πρώτη φάση. Συνεπώς στο παρόν κεφάλαιο αντιμετωπίζονται περισσότερο ποιοτικά τα αποτελέσματα. Η διαφοροποίηση που υπάρχει σε σχέση με τον πρώτο αλγόριθμο, είναι πως το κάθε directive που δοκιμάστηκε αφαιρέθηκε πριν δοκιμαστεί το επόμενο, ώστε να είναι ακόμη πιο δομημένη η αξιολόγηση των directives αφού φάνηκε πως δεν προκύπτουν αρνητικά αποτελέσματα στην απόδοση από λάθος συνδυασμό directives.

Αρχικά παρατίθενται τα directives τα οποία απορρίπτονται επειδή εξ’ ορισμού δεν μπορούν να χρησιμοποιηθούν στον συγκεκριμένο αλγόριθμο λόγω της λειτουργίας τους δικαιολογώντας την απόρριψή τους.

LOOP FLATTENING: Το directive αυτό δεν μπορεί να χρησιμοποιηθεί ούτε στον «LOOP3», ούτε στον «LOOP5» καθώς υπάρχει σώμα και στους εξωτερικούς βρόγχους («LOOP2» και «LOOP4»).

LOOP MERGING: Δεν μπορεί να χρησιμοποιηθεί καθώς οι βρόγχοι που υπάρχουν είναι εμφωλευμένοι και δεν γίνεται να γίνει κάποια ένωση βρόγχων με αυτό το directive καθώς απαιτεί μη εμφωλευμένους βρόγχους.

RESET: Το directive δεν μπορεί να εφαρμοστεί καθώς δεν υπάρχουν στατικές ή καθολικές μεταβλητές στον κώδικα.

FUNCTION_INSTANTIATE: Το directive αυτό δεν μπορεί να χρησιμοποιηθεί καθώς δεν υπάρχει συνάρτηση στον αλγόριθμο που να έχει παραπάνω από μία υλοποιήσεις.

OCCURRENCE: Στον συγκεκριμένο αλγόριθμο αυτό δεν μπορεί να χρησιμοποιηθεί καθώς δεν υπάρχουν διαφορετικού επιπέδου pipelined κομμάτια, τα οποία να έχουν διαφορετικό ρυθμό.

PROTOCOL: Το directive δεν μπορεί να χρησιμοποιηθεί, καθώς το ρολόι χρησιμοποιείται για να οργανώνει τις εγγραφές και τις αναγνώσεις των πινάκων.

ALLOCATION: Στον συγκεκριμένο αλγόριθμο υπάρχει μόνο μία υλοποίηση της συνάρτησης Solver(), καθώς και ένα μοναδικό πολλαπλασιασμού και ένα μοναδικό σημείο πρόσθεσης. Συνεπώς το συγκεκριμένο directive δεν μπορεί να χρησιμοποιηθεί για εξοικονόμηση πόρων.

Αρχικά χρησιμοποιείται και πάλι το directive «LOOP_TRIPCOUNT» για να εξαχθούν κάποια αποτελέσματα στο latency ώστε να υπάρχει ένα σημείο αναφοράς για τα directives όσο αναφορά τις επιπτώσεις τους στην απόδοση. Επειδή όπως αναφέρθηκε η μορφή των αποτελεσμάτων , στα οποία δεν υπάρχει προσδιορισμός latency προκύπτει από τα μεταβλητά όρια των βρόγχων, χρησιμοποιείται το «LOOP_TRIPCOUNT» directive για να οριστεί ένα μέγιστο πλήθος επαναλήψεων. Συγκεκριμένα δόθηκε μέγιστο πλήθος σε όλους τους βρόγχους ίσο με 229 όπου είναι ο αριθμός των γραμμών στους πίνακες για το συγκεκριμένο δείγμα δεδομένων. Έτσι προέκυψε ένα latency το οποίο ορίζεται ως σημείο αναφοράς.

Η μεγαλύτερη επιβάρυνση στο Latency παρατηρείται στους εξωτερικούς βρόγχους «LOOP1» και «LOOP4». Το συγκεκριμένο directive είναι το μόνο το οποίο δεν αφαιρείται καθ' όλη τη διάρκεια της εξέτασης των υπόλοιπων directives για να υπάρχει μορφή αποτελεσμάτων η οποία να μπορεί να είναι συγκρίσιμη με τα επόμενα.

DEPENDENCE: Το directive αυτό εφαρμόστηκε στους πίνακες leftrow[], rightColumn[], rowpoint[], matrix[] της συνάρτησης Transpose(), καθώς και στον πίνακα transpose[] της συνάρτησης Transpose() χωρίς να μεταβάλλει την απόδοση του αλγορίθμου. Συμπεραίνουμε πως δεν υπάρχουν εξαρτήσεις μεταξύ των πινάκων στις ίδιες ή σε διαφορετικές επαναλήψεις των βρόγχων.

EXPRESSION_BALANCE: Το directive αυτό εφαρμόστηκε στους βρόγχους «LOOP1», «LOOP2», «LOOP3», «LOOP4», «LOOP5» καθώς και στις συναρτήσεις Solver() και Transpose(), χωρίς να δώσει κάποια μεταβολή στην απόδοση του αλγορίθμου επειδή δεν μπόρεσε να δημιουργηθεί κάποια πιο αποδοτική ανακατανομή της αλληλουχίας των πράξεων.

INLINE: Το directive αυτό εφαρμόστηκε στις συναρτήσεις Solver() και Transpose() χωρίς κάποιο αποτέλεσμα καθώς δεν υπάρχει ιεραρχία επιπέδων, συνεπώς δεν μπορεί να βελτιώσει την απόδοση του αλγορίθμου.

Ακολουθούν τα directives με την εφαρμογή των οποίων με όπιο δυνατό συνδυασμό επιλογών έγινε, προέκυψε είτε θετική είτε αρνητική μεταβολή της απόδοσης του αλγορίθμου.

DATAFLOW: Το directive αυτό εφαρμόστηκε στο «LOOP1» αυξάνοντας το latency, προφανώς λόγω κάποια διένεξης μεταξύ των δεδομένων που παράγονται στους εμφωλευμένους βρόγχους. Στην συνέχεια εφαρμόστηκε στο «LOOP2» αυξάνοντας το latency, λόγω κάποιας διένεξης μεταξύ των αναγνώσεων, εγγραφών που γίνονται στον βρόγχο και των δεδομένων του εμφωλευμένου βρόγχου. Έπειτα εφαρμόστηκε στο «LOOP3» παρουσιάζοντας μήνυμα λάθους διότι υπήρχε κάποια εξάρτηση στον βρόγχο με την κλιμακωτή αύξηση της μεταβλητής «cell» η οποία υπάρχει μέσα στον βρόγχο την οποία δεν υποστήριζε το directive.Στην συνέχεια εφαρμόστηκε στην συνάρτηση Solver() αυξάνοντας το latency λόγω πιθανών κάποιας διένεξης δεδομένων, ενώ η εφαρμογή στα «LOOP4», «LOOP5» έδωσε μήνυμα λάθους ότι υπάρχουν προσπελάσεις διαβάσματος δεδομένων σ' αυτόν τον πίνακα. Τέλος εφαρμόστηκε στην συνάρτηση Transpose() αυξάνοντας το latency κατά έναν κύκλο λόγω κάποιας διένεξης δεδομένων και πάλι.

ARRAY MAP: Το directive αυτό εφαρμόστηκε στους παρακάτω συνδυασμούς πινάκων των οποίων παρατίθενται τα αποτελέσματα. Στον συνδυασμό lgrid[] & result[] με οριζόντιο προσανατολισμό , lgrid[] & rgrid[] & result[] & transposed[] με οριζόντιο συνδυασμό lgrid[] & rgrid[] & result[] & transposed[] με κατακόρυφο συνδυασμό, lgrid[] & rgrid[] με κατακόρυφο προσανατολισμό δεν υπήρξε καμία μεταβολή στο latency.

Στην συνέχεια στους συνδυασμούς , lgrid[] & rgrid[] & result[] με οριζόντιο προσανατολισμό, rgrid[] & result[] με οριζόντιο προσανατολισμό, lgrid[] & rgrid[] & result[] με οριζόντιο προσανατολισμό, καθώς και lgrid[] & rgrid[] & result[] με κατακόρυφο προσανατολισμό υπήρξε αύξηση του latency και συνεπώς το συγκεκριμένο directive οδηγεί είτε στην μη μεταβολή του latency είτε στην αύξησή του.

Η μεταβολές στο latency πιθανόν να προκύπτουν από αλλαγές στην διευθυνσιοδότηση των πινάκων για την προσπέλαση των στοιχείων τους.

ARRAY PARTITION: Το directive αυτό εφαρμόστηκε στους πίνακες lgrid[], rgrid[], result[] και transpose[], σε όλους τους τύπους, πάντα με τον παράγοντα «4» και στις δύο διαστάσεις εξάγοντας τα παρακάτω αποτελέσματα.

Εφαρμόζοντας στον lgrid[] σε τύπους complete και cyclic στην πρώτη διάσταση και στην δεύτερη διάσταση τα αποτελέσματα έμειναν αμετάβλητα. Εφαρμόζοντας στον τύπο block στην πρώτη όσο και στην δεύτερη διάσταση παρατηρήθηκε αύξηση του.

Συνεχίζοντας την ίδια διαδικασία στον πίνακα rgrid[] παρήχθησαν τα παρακάτω αποτελέσματα. Η εφαρμογή τύπου complete στην πρώτη διάσταση, καθώς και στην δεύτερη διάσταση παρήγαγε αμετάβλητα αποτελέσματα. . Η εφαρμογή τύπου block και cyclic στην πρώτη διάσταση καθώς και στην δεύτερη διάσταση προκάλεσε αύξηση του latency.

Ακολούθησε η ίδια διαδικασία στον πίνακα result[] παράγοντας τα παρακάτω αποτελέσματα. Η εφαρμογή τύπου cyclic στην πρώτη και στην δεύτερη διάσταση δεν προκάλεσε καμία μεταβολή στα αποτελέσματα. Η εφαρμογή τύπου block και cyclic στην πρώτη διάσταση καθώς και στην δεύτερη διάσταση προκάλεσε αύξηση του latency.

Στην συνέχεια έγινε η ίδια διαδικασία στον πίνακα transpose[] όπου το latency αυξήθηκε σε όλες τις περιπτώσεις, δηλαδή εφαρμόζοντας τύπο block, cyclic και complete τόσο στην πρώτη όσο και στην δεύτερη διάσταση.

Η αύξηση του latency πιθανόν να προκλήθηκε λόγω αύξησης των πινάκων και κατ' επέκταση της αλλαγής διευθυνσιοδότησης για την προσπέλαση των στοιχείων τους.

ARRAY RESHAPE: Το directive αυτό εφαρμόστηκε στους πίνακες lgrid[], rgrid[], result[] και transpose[], σε όλους τους τύπους, και πάλι πάντα με τον παράγοντα «4» και στις δύο διαστάσεις εξάγοντας τα αποτελέσματα που φαίνονται στην συνέχεια. Αναφέρονται οι περιπτώσεις όπου δεν υπήρξε μήνυμα λάθους το οποίο προήλθε από την αλληλουχία προσπελάσεων στους πίνακες και ερχόταν σε σύγκρουση με την διάσπαση των πινάκων.

Στον πίνακα lgrid[] εφαρμόζοντας τύπο cyclic στην πρώτη και στην δεύτερη διάσταση το latency παρέμεινε σταθερό. Η εφαρμογή τύπου block και cyclic στην πρώτη διάσταση καθώς και στην δεύτερη διάσταση προκάλεσε αύξηση του latency.

Στην συνέχεια έγινε η ίδια διαδικασία στον πίνακα rgrid[] παράγοντας τα παρακάτω αποτελέσματα. Εφαρμόζοντας τύπο cyclic στην πρώτη διάσταση το latency αυξήθηκε. Η εφαρμογή τύπου block και complete στην πρώτη διάσταση καθώς και στην δεύτερη διάσταση προκάλεσε αύξηση του latency.

Στην συνέχεια έγινε η ίδια διαδικασία στον πίνακα result[] όπου το latency αυξήθηκε σε όλες τις περιπτώσεις, δηλαδή εφαρμόζοντας τύπο block, cyclic και complete τόσο στην πρώτη όσο και στην δεύτερη διάσταση.

Στην συνέχεια έγινε η ίδια διαδικασία στον πίνακα rtransposed[] όπου το latency αυξήθηκε σε όλες τις περιπτώσεις, δηλαδή εφαρμόζοντας τύπο block, cyclic και complete τόσο στην πρώτη όσο και στην δεύτερη διάσταση.

Οι πίνακες matrix[] και transpose[] της συνάρτησης Transpose() είναι οι πίνακες rgrid[] και rtransposed[] της συνάρτησης Solver() αντίστοιχα , συνεπώς δίνουν τα ίδια αποτελέσματα με αυτούς στην εφαρμογή του directive.

Η αύξηση του latency πιθανόν να προκλήθηκε λόγω αύξησης των πινάκων και κατ' επέκταση της αλλαγής διευθυνσιοδότησης για την προσπέλαση των στοιχείων τους και σ' αυτήν την περίπτωση, καθώς το συγκεκριμένο directive χρησιμοποιεί και «array_partition»

Αφού σε όλες τις περιπτώσεις εφαρμογής το συγκεκριμένο directive άφησε αμετάβλητο ή αύξησε το latency , δεν μπόρεσε να γίνει κάποια βελτιστοποίηση με αυτό.

UNROLL: Το συγκεκριμένο directive εφαρμόστηκε με παράγοντα ίσο με δύο(2) σε κάθε έναν από τους πέντε βρόγχους «LOOP1», «LOOP2», «LOOP3», «LOOP4», «LOOP5» του

αλγορίθμου δίνοντας τα παρακάτω αποτελέσματα. Εφαρμόζοντάς το στο «LOOP1» υπήρξε αύξηση του latency όπως και εφαρμόζοντάς το στο «LOOP2». Εφαρμόζοντάς το στο «LOOP3» υπήρξε μείωση του latency, ενώ εφαρμόζοντάς το στο «LOOP4» υπήρξε αύξηση του latency. Τέλος εφαρμόζοντάς το στο «LOOP5» υπήρξε αύξηση του latency και πάλι.

Προκύπτει τελικά πως αύξηση της απόδοσης του αλγορίθμου υπάρχει μόνο όταν εφαρμοστεί το directive στον πιο εσωτερικό βρόγχο σε μία σειρά από εμφωλευμένους βρόγχους. Σε διαφορετική περίπτωση όπως και στον προηγούμενο αλγόριθμο δημιουργείται ένα αντίγραφο του πιο εσωτερικού βρόγχου κατά την εφαρμογή του directive, γεγονός το οποίο αυξάνει το latency. Επίσης μπορεί να μειωθεί η απόδοση όταν υπάρχουν κάποιων τύπων εξαρτήσεις μεταξύ προσπελάσεων πινάκων, όπως συμβαίνει στο «LOOP5».

PIPELINE: Το directive αυτό εφαρμόστηκε στους πέντε βρόγχους του αλγορίθμου «LOOP1», «LOOP2», «LOOP3», «LOOP4», «LOOP5» καθώς και στις δύο συναρτήσεις function() και Transpose() εξάγοντας τα παρακάτω αποτελέσματα.

Εφαρμόζοντας στο «LOOP1» δεν παρατηρήθηκε καμία μεταβολή καθώς οι κύκλοι του latency παρέμειναν οι ίδιοι, εφαρμόζοντάς στο «LOOP2» παρατηρήθηκε αύξηση του latency, ενώ εφαρμόζοντάς στο «LOOP3» παρατηρήθηκε μείωση του latency. Εφαρμόζοντάς στο «LOOP4» δεν παρατηρήθηκε μεταβολή του latency και τέλος εφαρμόζοντάς στο «LOOP5» παρατηρήθηκε μείωση του latency. Στην συνέχεια εφαρμόζοντάς το στην συνάρτηση Solver() δεν παρατηρήθηκε μεταβολή του latency, ενώ εφαρμόζοντάς το στην συνάρτηση Transpose() παρατηρήθηκε αύξηση του latency κατά έναν κύκλο.

Όπως προκύπτει από τα αποτελέσματα το latency μειώνεται μόνο όταν εφαρμοστεί το directive στον πιο εσωτερικό βρόγχο των εμφωλευμένων, σε διαφορετική περίπτωση μπορεί να δημιουργηθούν stalls τα οποία δεν μπορούν να επιλυθούν.

CLOCK: Η αρχική περίοδος του ρολογιού είναι TS1 = 10ns, συνεπώς για την εκτέλεση του αλγορίθμου απαιτούνται ts1=144.318.779*10ns= 1.443.187.790ns. Αρχικά μειώνοντας την περίοδο μόνο στην συνάρτηση Solver() σε TS2=8ns προέκυψε αύξηση του latency σε κύκλους όμως μείωση του χρόνου εκτέλεσης. Έπειτα μειώνοντας την περίοδο περισσότερο στην συνάρτηση Solver() σε TS3=6ns προέκυψε και πάλι προέκυψε αύξηση του latency σε κύκλους όμως μείωση του χρόνου εκτέλεσης.

Στην συνέχεια αφαιρέθηκε το directive από την συνάρτηση Solver() της οποίας η περίοδος ρολογιού επέστρεψε και πάλι στην προκαθορισμένη και το directive εφαρμόστηκε μόνο στην

συνάρτηση Solver() με TR2=8ns όπου προέκυψε αύξηση του latency σε κύκλους όμως μείωση του χρόνου εκτέλεσης.

Στην συνέχεια εφαρμόστηκε το directive στην συνάρτηση Solver() με TS3 = 6ns και ταυτόχρονα στην συνάρτηση Transpose() με TR3=8ns και εξήγαγε αριθμό κύκλων latency πάλι μειωμένο, ίσο όμως με τα αποτελέσματα των TR2 και TS2.

Τέλος εφαρμόστηκε το directive στην συνάρτηση Solver() με TS4=6ns και ταυτόχρονα στην συνάρτηση Transpose() με TR4=8ns και εξήγαγε μειωμένο αριθμό κύκλων latency ίσο με τα αποτελέμσατα των TR3,TR2 και TS2.

Συνεπώς όταν υπάρχουν περισσότερα από ένα ρολόγια εφαρμοσμένα σε διαφορετικές συναρτήσεις, τότε το vivado υπολογίζει ως συνολικό ρολόι αυτό με την μεγαλύτερη περίοδο.

STREAM: Το directive αυτό εφαρμόστηκε και για τις δύο διαστάσεις των πινάκων και πάντα με βάθος (depth) ίσο με δύο (2).

Εφαρμόστηκε για τον πίνακα lgrid[] στην πρώτη διάσταση όπου παρουσίασε μήνυμα λάθους διότι βρέθηκαν προσβάσεις στον πίνακα σε μορφή στοίβας, οι οποίες δεν ήταν συμβατές με το συγκεκριμένο directive. Στην συνέχεια εφαρμόστηκε στην δεύτερη διάστασης όπου δεν παρουσίασε καμία μεταβολή του latency.

Η εφαρμογή στον πίνακα rgrid[] στην πρώτη διάσταση παρουσίασε μήνυμα λάθους για τον ίδιο λόγο με πριν, ενώ η εφαρμογή στην δεύτερη διάσταση προκάλεσε μείωση του latency.

Η εφαρμογή στον πίνακα result[] δίνει τα ίδια ακριβώς αποτελέσματα με την εφαρμογή στον πίνακα rgrid[], δηλαδή εμφάνιση λάθους στην πρώτη διάσταση και μείωση του latency στην δεύτερη διάσταση.

Η εφαρμογή στον πίνακα rtransposed[] παρουσίασε μηνύματα λάθους και στις δύο διαστάσεις, λόγω της αρχικοποίησης του μέσα στην συνάρτηση Solver().

Οι πίνακες matrix[] και transpose[] της συνάρτησης Transpose() παρουσιάζουν την ίδια συμπεριφορά με τους πίνακες rgrid[] και rtransposed[] της συνάρτησης Solver() αφού είναι οι ίδιοι πίνακες που εισάγονται από την μία συνάρτηση στην άλλη.

Συνεπώς απορρίφθηκαν όλα τα directives της επανεξέτασης και έτσι αυτά που απέμειναν για την τελική φάση είναι τα stream», «clock», «loop tripcount», «unroll» και «pipelining».

Το «loop_tripcount» χρησιμοποιείται σε όλες τις περιπτώσεις, ώστε να απαλειφθούν τα ερωτηματικά από τους χρόνους και να υπάρχει ένα σημείο αναφοράς

Συμπεράσματα

Τα directives με τα οποία αρχίζει η διαδικασία αξιολόγησης για τον κάθε αλγόριθμο είναι τα ίδια. Τα directives τα οποία παρείχαν θετικά αποτελέσματα για τον πρώτο αλγόριθμο είναι τα «loop_tripcount», «clock», «pipline», «stream» και «unroll», ενώ αυτά τα οποία εξήγαγαν θετικά αποτελέσματα για τον δεύτερο αλγόριθμο είναι τα «stream», «clock», «loop tripcount», «unroll», «pipeline». Παρατηρείται πως είναι τα ίδια, όπως παρόμοια είναι και τα directives τα οποία δεν χρειάστηκε καν να μελετηθούν λόγω της λειτουργίας τους. Είναι φανερό πως για παρόμοιους αλγόριθμους με κοινό γνώρισμα, όπως την χρήση πινάκων και εμφωλευμένων βρόγχων σ΄αυτήν την περίπτωση, τα directives τα οποία προσδίσουν τελικά βελτιωμένη απόδοση είναι τα ίδια και άρα μπορεί να προβλεφθεί με μεγάλη πιθανότητα το ποια directives θα έχουν επίδραση στην απόδοσή παρόμοιων μελλοντικών αλγορίθμων. Η επίδραση του συνδιασμού τους στο τελικό αποτέλεσμα μελετάται σε επόμενο κεφάλαιο.

Κεφάλαιο 5: Απόδοση

i. Απόδοση για το Sparse Matrix Multiplication

1. Πρώτος Συνδυασμό

Αρχικά εφαρμόστηκε «loop_tripcount» στο «INNER_LOOP» και στο «OUTER_LOOP» με μέγιστο αριθμό επαναλήψεων ίσο με 362 για λόγους που έχουν αναλυθεί παραπάνω. Αυτό εξάγει latency ίσο με 1.966.747 κύκλους.

Στην συνέχεια το εφαρμόζεται στο «pipeline» «INNER_LOOP» για την διοχέτευση δεδομένων στην επόμενη επανάληψη πριν ολοκληρωθεί η εκάστοτε, το οποίο πραγματοποιείται με επιτυχία και εξάγει latency ίσο με 659.927 το οποίο είναι καλύτερο , οπότε διατηρείται. Έπειτα ενεργοποιείται η επιλογή του directive «loop_rewind» με σκοπό να μην γίνεται παύση μεταξύ των επαναλήψεων του βρόγχου, το οποίο επιτεύχθηκε και προκύπτει latency ίσο 659.926 κύκλους, το οποίο είναι καλύτερο και συνεπώς διατηρείται. Έπειτα ενεργοποιείται η επιλογή «flush» του directive το για να επιλυθούν τυχόν stalls που μπορεί να δημιουργούνται, το οποίο δεν μεταβάλλει την απόδοση οπότε δεν υπήρξανε stalls επιλύσιμα.

Έπειτα εφαρμόζεται «stream» στους παρακάτω πίνακες, ώστε να ανακαλυφτεί ποιος από τους πίνακες υλοποιημένος ως FIFO μπορεί να βελτιώσει καλύτερα την απόδοση. Έτσι εξάγει με εφαρμογή στον val[] 659.926 κύκλους, με εφαρμογή στον col_ind[] 659.564 κύκλους, στον row_ptr[] 659.926 κύκλους, στον vector[] 659.564 κύκλους και στον result[] 569.926 κύκλους.

Είναι φανερό πως η καλύτερη απόδοση επιτεύχθηκε με το directive στους πίνακες col_ind[] και vector[] στους οποίους οι προσπελάσεις γίνονται πιο ακολουθιακά, ενώ ακολουθούν οι πίνακες row_ptr[],result[] και val[]. Δοκιμάζοντας την επιλογή «depth» έγινε προσπάθεια να βρεθεί αν υπάρχει κάποιος συγκεκριμένος ρυθμός προσπέλασης στους πίνακες με την καλύτερη απόδοση (col_ind[] και vector[]) χωρίς να επιφέρει κάποια μεταβολή στο αποτέλεσμα και άρα αφαιρέθηκε. Έπειτα γίνεται δοκιμή συνδυασμού πινάκων με αυτό το directive για να ερευνηθεί αν μπορεί να βελτιωθεί η απόδοση, με την εφαρμογή του directive

ταυτόχρονα σε πίνακες χωρίς να επηρεαστούν μεταξύ τους αρνητικά από τη μορφή FIFO. Αρχικά, ξεκινούν οι συνδυασμοί που μόνοι τους δώσανε τα καλύτερα αποτελέσματα και στην συνέχεια ακολουθούν οι συνδυασμοί με πίνακες που δώσανε λιγότερο καλά αποτελέσματα. Κάθε φορά που βρίσκεται ένας συνδυασμός πινάκων με καλύτερη απόδοση από πριν, δοκιμάζονται συνδυασμοί με τους εναπομείναντες πίνακες πάλι με την ίδια λογική. Έτσι παρακάτω δίνεται το latency για τους αντίστοιχους συνδυασμούς πινάκων.

Συνδυασμός Col_ind[] και vector[] με latency ίσο με 659.564 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Συνδυασμός Col_ind[] και vector[] και val[] με latency ίσο με 659.202 κύκλους, το οποίο είναι το καλύτερο που έχει επιτευχθεί εως τώρα άρα το διατηρείται ως επιλογή και ως μέτρο σύγκρισης.

Συνδυασμός Col_ind[] και vector[] και row_ptr[] με latency ίσο με 659.564 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Συνδυασμός Col_ind[] και vector[] και result[] με latency ίσο με 659.564 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Συνδυασμός Col_ind[] και vector[] και val[] και row_ptr[] με latency ίσο με 659.202 κύκλους, το οποίο είναι το καλύτερο που έχει επιτευχθεί εως τώρα άρα το διατηρείται ως επιλογή και ως μέτρο σύγκρισης.

Συνδυασμός Col_ind[] και vector[] και val[] και result[] με latency ίσο με 659.202 κύκλους, το οποίο είναι το καλύτερο που έχει επιτευχθεί εως τώρα άρα διατηρείται ως επιλογή και ως μέτρο σύγκρισης.

Προς το παρόν διατηρείται ο συνδυασμός των Col_ind[] μαζί με vector[] και val[], που ήταν ο πρώτος συνδυασμός που μας εξήγαγε τα καλύτερα αποτελέσματα γι’ αυτό το directive και η διαδικασία συνεχίζεται.

Στην συνέχεια εφαρμόζεται «clock» στην συνάρτηση Solver(). Σύμφωνα με την τωρινή περίοδο ρολογιού που είναι ίση με 10ns, η διάρκεια εκτέλεσης του αλγορίθμου είναι 659.202* 10ns = 6.592.020 ns. Σκοπός είναι να μειωθεί η περίοδος τόσο ώστε εκτελούνται οι απαραίτητοι υπολογισμοί, αλλά να μην υπάρχει περιττό διάστημα στον παλμό. Αλλάζοντάς την περίοδο προκύπτουν οι παρακάτω τα αντίστοιχα αποτελέσματα:

Για T1=7ns προκύπτει latency ίσο με 660.288 κύκλους , t1 = 660.288 * 7ns = 6.592.020ns

Για T2=6ns προκύπτει latency ίσο με 661.374, t2 = 661.374 * 6ns = 3.968.244ns.

Για T3=5ns προκύπτει latency ίσο με 1.054.868, t3 = 1.054.868 * 5ns = 5.274.340ns.

Παρατηρείται πως ενώ μειώθηκε η περίοδος ο χρόνος υπολογισμού αυξήθηκε, καθώς κάποιες λειτουργίες οι οποίες δεν μπορούσαν να πραγματοποιηθούν σε μία περίοδο του ρολογιού χωρίστηκαν σε δύο αυξάνοντας τον χρόνο αδράνειας μέσα στον παλμό. Παρατηρείται πως η καλύτερη απόδοση υπάρχει για T2=6ns οπότε και διατηρείται όπου επιτεύχθηκε ο σκοπός της εφαρμογής του directive.

Στην συνέχεια εφαρμόζεται «unroll» στον «INNER_LOOP» με «factor» ίσο με 2 ώστε να δημιουργηθεί ένα αντίγραφό του βρόγχου και να μειωθούν οι επαναλήψεις στο μισό, γεγονός το οποίο καθιστά το Estimate του ρολογιού μεγαλύτερο από το target. Στη συνέχεια αυξάνεται η περίοδος του ρολογιού σε T=11ns για να δοκιμαστεί αν θα αντιμετωπιστεί το πρόβλημα όμως και πάλι το estimated του ρολογιού ήταν μεγαλύτερο από το target. Στην συνέχεια αυξάνεται ξανά η περίοδος του ρολογιού σε T=20 ns αλλά και πάλι εξάγονται ίδιου είδους αποτελέσματα. Παρατηρείται πως το estimated είναι πάντα πάνω από το target του ρολογιού επειδή δημιουργώντας αντίγραφο των υπολογισμών του βρόγχου με το directive πάντα χρειαζόταν παραπάνω από μια περίοδος για να ολοκληρωθούν οι υπολογισμοί ανεξάρτητα από τον χρόνο της περιόδου, καθώς υπάρχουν λειτουργίες που χρειάζονται έναν παλμό. Έτσι αφαιρείται το «unroll» και επαναφέρεται η περίοδος T=6ns.

Στην συνέχεια αλλάζουμε τον συνδυασμό πινάκων που εξήγαγαν τα καλύτερα αποτελέσματα στο directive «stream». Ισοδύναμος με τον συνδυασμό col_ind[] μαζί με vector[] και val[], ήταν και ο col_ind[] μαζί με vector[] , val[] και row_ptr[] καθώς και col_ind[] μαζί με vector[] , val[] και result[] τους οποίους και δοκιμάζουμε εξάγοντας τα ίδια αποτελέσματα οπότε διατηρείται ο αρχικό συνδυασμό όπου εφαρμόζεται το directive σε 3 και όχι σε 4 πίνακες.

Τελικά ο χρόνος εκτέλεσης του αλγορίθμου είναι t = 661.374*6ns=3.968.244ns.

Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.1: Αποτελέσματα πρώτου συνδυασμού

LOOP_TRIPCOUNT	Latency = 1.966.747 κύκλοι Χρόνος εκτέλεσης = 1.966.747 * 10ns = 19.667.470 ns
+PIPELINE	Latency = 659.926 κύκλοι Χρόνος εκτέλεσης = 659.926 * 10ns = 6.599.260 ns
+STREAM	Latency = 659.202 κύκλοι Χρόνος εκτέλεσης = 659.202 * 10ns = 6.592.020 ns

+CLOCK	Latency = 661.374 κύκλοι Χρόνος εκτέλεσης = 661.374 * 6ns = 3.968.244 ns

2. Δεύτερος Συνδυασμός

Εφαρμόζεται «loop_tripcount» στο «INNER_LOOP» και στο «OUTER_LOOP» με μέγιστο αριθμό επαναλήψεων ίσο με 362 για λόγους που έχουν αναλυθεί παραπάνω. Αυτό εξάγει latency ίσο με 1.966.747 κύκλους.

Στην συνέχεια εφαρμόζεται «stream» με την ίδια προοπτική όπως και στον πρώτο συνδυασμό στον πίνακα val[] με latency ίσο με 2.098.153 κύκλους, στον col_ind[] με latency ίσο με 1.967.109 κύκλους, στον row_ptr[] με latency ίσο με 1.966.747 κύκλους, στον vector[] με latency ίσο με 1.967.109 κύκλους και στον result[] με latency ίσο με 1.966.747 κύκλους.

Είναι φανερό πως η καλύτερη απόδοση επιτεύχθηκε με το directive στους πίνακες col_ind[] και vector[], ενώ ακολουθούν οι πίνακες row_ptr[],result[] και τελευταίος έρχεται ο πίνακας val[]. Δοκιμάζοντας «depth» ίσο με 3 στους πίνακες με την καλύτερη απόδοση (col_ind[] και vector[]) χωρίς να υπάρξει κάποια μεταβολή στο αποτέλεσμα και άρα αφαιρείται. Έπειτα γίνεται δοκιμή συνδυασμού πινάκων με αυτό το directive για να ερευνηθεί αν μπορεί να βελτιωθεί η απόδοση, με την εφαρμογή του directive ταυτόχρονα σε πίνακες. Αρχικά, ξεκινούν οι συνδυασμοί που μόνοι τους δώσανε τα καλύτερα αποτελέσματα και στην συνέχεια ακολουθούν οι συνδυασμοί με πίνακες που δώσανε λιγότερο καλά αποτελέσματα. Κάθε φορά που βρίσκεται ένας συνδυασμός πινάκων υλοποιημένων ως FIFO με καλύτερη απόδοση από πριν, δοκιμάζονται συνδυασμοί με τους εναπομείναντες πίνακες πάλι με την ίδια λογική. Έτσι προκύπτει το latency για τους αντίστοιχους συνδυασμούς πινάκων.

Ο συνδυασμός Col_ind[] και vector[] δεν εξάγει καμία μεταβολή στο latency(1.967.109 κύκλοι) άρα αφαιρείται.

Ο συνδυασμός Col_ind[] και row_ptr[] εξάγει latency ίσο με 1.835.341 όπως φαίνεται παρακάτω, οπότε διατηρείται αυτός ο συνδυασμός.

Ο συνδυασμός Col_ind[] και result[] εξάγει latency ίσο με 1.967.110 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Ο συνδυασμός vector [] και row_ptr[] εξάγει latency ίσο με 1.835.341 κύκλους, ίσο με το καλύτερο άρα λαμβάνεται υπ’ όψιν ως πιθανός συνδυασμός.

Ο συνδυασμός row_ptr [] και result [] εξάγει latency ίσο με 1.966.747 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Ο συνδυασμός Col_ind [] και row_ptr[] και val[] εξάγει latency ίσο με 1.967.109 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Ο συνδυασμός vector [] και row_ptr[] και val[] εξάγει latency ίσο με 1.836.065 κύκλους, μεγαλύτερο από το καλύτερο ως τώρα άρα αφαιρείται.

Έτσι τελικά προκύπτουν ως καλύτεροι συνδυασμός οι Col_ind [] μαζί με row_ptr[], καθώς και vector[] μαζί με row_ptr[]. Συνεχίζουμε προς το παρόν κρατώντας τον πρώτο συνδυασμό.

Στην συνέχεια εφαρμόζεται «pipeline» στο «INNER_LOOP» ώστε να υπάρχει διοχέτευση δεδομένων στην επόμενη επανάληψη του βρόγχου πριν ολοκληρωθεί η εκάστοτε, το οποίο πραγματοποιείται με επιτυχία και εξάγει latency ίσο με 659.565 το οποίο είναι καλύτερο οπότε διατηρείται. Στην συνέχεια ενεργοποιείται η επιλογή του directive «loop_rewind» για την απομάκρυνση της παύσης μεταξύ των επαναλήψεων, το οποίο πραγματοποιείται με επιτυχία και το latency γίνεται ίσο με 659.564 κύκλους, το οποίο είναι καλύτερο και διατηρείται. Έπειτα ενεργοποιείται η επιλογή «flush» του directive η οποία δεν μεταβάλλει την απόδοση οπότε και απενεργοποιείται πάλι.

Στην συνέχεια εφαρμόζεται «unroll» στον «INNER_LOOP» με «factor» ίσο με 2 για να εξακριβωθεί ότι δεν προήλθε από κάποια διένεξη στα directives το πρόβλημα που προέκυψε στον πρώτο συνδυασμό, όμως και πάλι το Estimate του ρολογιού είναι μεγαλύτερο από το target.

Αφαιρείται το «pipeline» έχοντας αφήσει το unroll, οπότε τελικά το latency είναι ίσο με 1.774.887 κύκλους.

Στην συνέχεια εφαρμόζεται «clock» στην συνάρτηση Solver() για τους ίδιους σκοπούς με τον πρώτο συνδυασμό, δηλαδή την πιο αποδοτική περίοδο ρολογιού. Σύμφωνα με την παρούσα περίοδο ρολογιού που είναι ίση με 10ns, η διάρκεια εκτέλεσης του αλγορίθμου είναι 1.774.887 * 10ns = 17.748.870 ns. Αλλάζοντάς την ώστε να επιτευχθούν καλύτερα αποτελέσματα μέσω του directive, προκύπτουν οι παρακάτω δοκιμές με τα αντίστοιχα αποτελέσματα:

Για T1 = 6ns με latency ίσο με 2.563.685 κύκλους, t1 = 2.563.685 * 6ns = 15.382.110ns

Για T2 = 5ns με latency ίσο με 2.957.903 κύκλους, t2 = 2.957.903 * 5ns = 14.789.515ns

Για T3 = 4ns με latency ίσο με 3.352.121 κύκλους , t3 = 3.352.121 * 4ns = 13.408.484ns

Για T4 = 3ns το estimate του ρολογιού είναι μεγαλύτερο από το target προφανώς επειδή η περίοδος αυτή δεν ήταν αρκετά μεγάλη ώστε να ολοκληρωθούν όλες οι λειτουργίες που γίνονται μέσα σε αυτήν. Έτσι διατηρείται το T3=4ns.

Στην συνέχεια πραγματοποιείται «unroll» με «factor» ίσο με 3, ώστε να δημιουργηθούν 2 αντίγραφα των λειτουργιών του βρόγχου , ώστε να πραγματοποιούνται λιγότερες επαναλήψεις, οπότε και εξάγεται latency ίσο με 3.348.501 κύκλους, το οποίο και επιτεύχθηκε και διατηρείται, αφού είναι καλύτερο από πριν, με το αντίτιμο την παραγωγή περισσότερου υλικού. Δοκιμάζοντας το ίδιο directive για «factor» ίσο με 4 εξάγεται latency χειρότερο από πριν πιθανών λόγω κάποιας εξάρτησης στα δεδομένα η οποία προκαλεί καθυστέρηση άρα επαναφέρεται «factor» ίσο με 3.

Στην συνέχεια δοκιμάζεται ο ισοδύναμος συνδυασμός πινάκων που εξήγαγε τα καλύτερα αποτελέσματα στο directive «stream». Ισοδύναμος με τον συνδυασμό col_ind[] και row_ptr[] ήταν και ο vector[] μαζί με row_ptr[] ο οποίος και δοκιμάζεται εξάγοντας latency ίσο με 3.216.371 κύκλους, το οποίο είναι καλύτερο από πριν προφανώς λόγω καλύτερου συνδυασμού των directive, και άρα διατηρείται. Έτσι τελικά ο χρόνος εκτέλεσης του αλγορίθμου είναι t = 3.216.371*4ns=12.865.484 ns.

Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.2: Αποτελέσματα δεύτερου συνδυασμού

LOOP_TRIPCOUNT	Latency = 1.966.747 κύκλοι Χρόνος εκτέλεσης = 1.966.747 * 10ns = 19.667.470 ns
+STREAM	Latency = 1.835.887 κύκλοι Χρόνος εκτέλεσης = 1.835.887 * 10ns = 18.358.870 ns
+UNROLL	Latency = 1.774.887 κύκλοι Χρόνος εκτέλεσης = 1.774.887 * 10ns = 17.748.870 ns
+CLOCK	Latency = 3.216.371 κύκλοι Χρόνος εκτέλεσης = 3.216.371 * 4ns = 12.865.484 ns

3. Τρίτος Συνδυασμός

Αρχικά εφαρμόζεται «loop_tripcount» στο «INNER_LOOP» και στο «OUTER_LOOP» με μέγιστο αριθμό επαναλήψεων ίσο με 362 για λόγους που έχουν αναλυθεί παραπάνω. Αυτό εξάγει latency ίσο με 1.966.747 κύκλους.

Στην συνέχεια εφαρμόζεται «pipeline» για τους ίδιους σκοπούς με τις προηγούμενες υλοποιήσεις, στο «INNER_LOOP» το οποίο δίνει latency ίσο με 659.927 κύκλους, το οποίο είναι καλύτερο, οπότε διατηρείται. Στην συνέχεια ενεργοποιείται η επιλογή του directive «loop_rewind», και εξάγεται latency ίσο με 659.926 κύκλους το οποίο είναι καλύτερο και διατηρείται. Έπειτα ενεργοποιείται η επιλογή «flush» του directive η οποία δεν μεταβάλλει την απόδοση οπότε και απενεργοποιείται πάλι.

Στην συνέχεια εφαρμόζεται «unroll» για τους ίδιους σκοπούς με τις προηγούμενες υλοποιήσεις, στον «INNER_LOOP» με «factor» ίσο με 2 το οποίο καθιστά το Estimate του ρολογιού μεγαλύτερο από το target , οπότε και αφαιρείται το directive.

Εν συνεχεία εφαρμόζεται «clock» για τους ίδιους σκοπούς με τις προηγούμενες υλοποιήσεις, στην συνάρτηση Solver(). Σύμφωνα με την παρούσα περίοδο ρολογιού που είναι ίση με 10ns, η διάρκεια εκτέλεσης του αλγορίθμου είναι 659.927 * 10ns = 6.599.270 ns. Αλλάζοντάς την ώστε να επιτευχθούν καλύτερα αποτελέσματα μέσω του directive, προκύπτουν οι παρακάτω περιπτώσεις με τα αντίστοιχα αποτελέσματα:

Για T1 = 6ns προκύπτει latency ίσο με 662.460 κύκλους, t1= 662.460 * 6ns = 3.974.760ns.

Για T2=5ns εξάγεται latency ίσο με 1.059.954 κύκλους, t2 = 1.059.954 * 5ns = 5.299.770ns
Άρα τελικά διατηρείται T1=6ns και ο χρόνος εκτέλεσης του αλγορίθμου είναι

t =662.460*6ns=3.974.760ns.

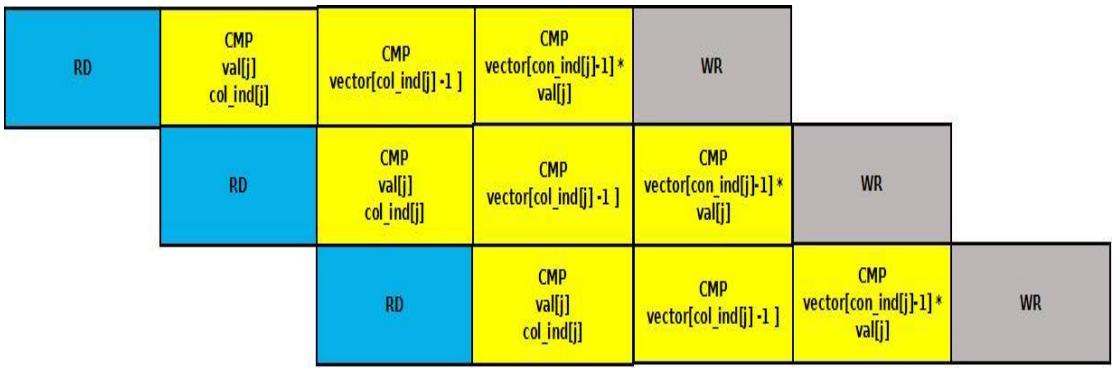
Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.3: Αποτελέσματα τρίτου συνδυασμού

LOOP_TRIPCOUNT	Latency = 1.966.747 κύκλοι Χρόνος εκτέλεσης = 1.966.747 * 10ns = 19.667.470 ns
+PIPELINE	Latency = 659.926 κύκλοι Χρόνος εκτέλεσης = 659.926 * 10ns = 6.599.260 ns
+CLOCK	Latency = 662.460 κύκλοι Χρόνος εκτέλεσης = 662.460 * 6ns = 3.974.760 ns

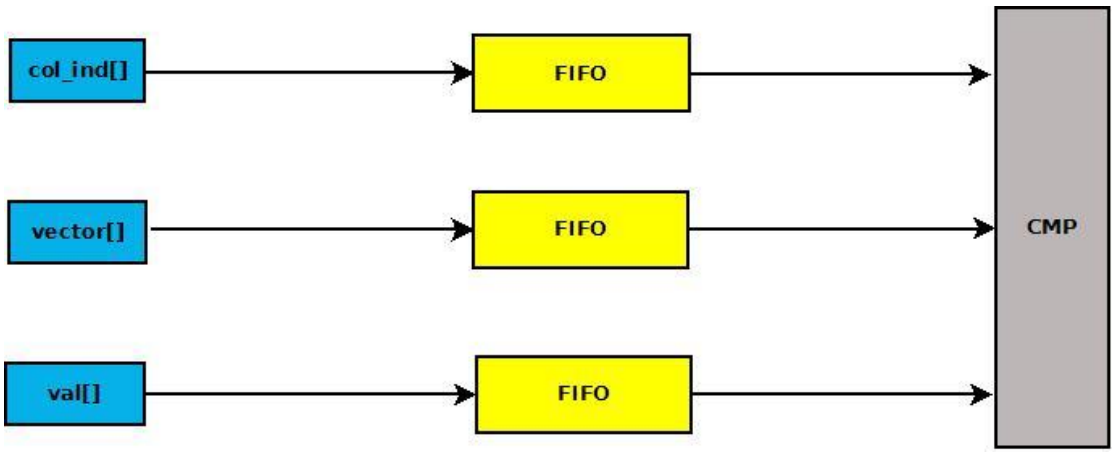
Από τις δοκιμές προκύπτει πως ο πρώτος συνδυασμός ήταν ο καλύτερος.

Παρατίθεται η αρχιτεκτονική του directive «pipeline» για τον καλύτερο συνδυασμό, όσο αναφορά τη ροή των δεδομένων.



Εικόνα 5.1: Δομή Pipeline directive

Επίσης παρατίθεται η αρχιτεκτονική του directive «stream» για τον καλύτερο συνδυασμό, όσο αναφορά τη ροή των δεδομένων.



Εικόνα 5.2: Δομή Stream directive

Τέλος εφαρμόζεται το directive “resource” στους πίνακες όπου έχουν υλοποιηθεί μέσω του “stream” ως FIFO ώστε να πάρουν αυτήν την δομή στην FPGA οι πίνακες.

ii. Απόδοση για το Dense- Matrix-Vector Multiplication

Στο παρόν κεφάλαιο επιλέχθηκαν τα directives τα οποία είχαν μεμονωμένα θετική επίδραση στον αλγόριθμο με σκοπό να βρεθεί ο καλύτερος συνδυασμός τους ,ως προς τον χρόνο εκτέλεσης του αλγορίθμου .Τα directives αυτά είναι τα «stream», «clock», «loop tripcount», «unroll», «pipelining».

Για να μπορεί να υπάρχει μέτρο σύγκρισης μεταξύ των διαφορετικών συνδυασμών διατηρήθηκε το directive «loop_tripcount» σε όλες τις περιπτώσεις λόγω του

μεταβλητού μεγέθους τον βρόγχων, το οποίο χωρίς το συγκεκριμένο directive δεν επιτρέπει την εξαγωγή αποτελεσμάτων. Το «loop tripcount» είναι το πρώτο directive που εφαρμόζεται σε όλες τις περιπτώσεις δίνοντας latency ίσο με 144.318.779 και σε περίοδο ρολογιού ίση με 10ns.

1. Πρώτος Συνδυασμός

Όπως προαναφέρθηκε αρχικά γίνεται εφαρμογή του «loop tripcount» εξάγοντας τα αποτελέσματα που αναφέρθηκαν παραπάνω.

Στην συνέχεια έγινε εφαρμογή του directive «stream» στον πίνακα rgrid[] ώστε να υλοποιηθεί ως FIFO απ' όπου προέκυψε ελαφρά μείωση του latency στους 144.318.550 κύκλους ρολογιού γεγονός το οποίο συντέλεσε στην διατήρηση της εφαρμογής του directive.

Το επόμενο βήμα ήταν η εφαρμογή του directive «pipeline» με σκοπό την διοχέτευση δεδομένων στην επόμενη επανάληψη του βρόγχου πριν ολοκληρωθεί η εκάστοτε. Η εφαρμογή του έγινε αρχικά στον βρόγχο «LOOP3» χωρίς κάποια επιπρόσθετη επιλογή καθώς και με τις επιλογές «flushing» και «loop rewind». Η εφαρμογή χωρίς κάποια επιπρόσθετη επιλογή, καθώς και με την επιλογή «flushing» για να επιλυθούν τυχόν stalls, εξήγαγε μειωμένο latency στους 60.674.699 κύκλους επιτυγχάνοντας τον στόχο, ενώ η τρίτη δεν μπόρεσε να ολοκληρωθεί αφού προέκυψε μήνυμα λάθους λόγω πολλαπλών βρόγχων μέσα στην συνάρτηση. Στην συνέχεια έγινε με τον ίδιο ακριβώς τρόπο εφαρμογή στον βρόγχο «LOOP5» όπου με την απλή εφαρμογή δεν παρατηρήθηκε κάποια μεταβολή των αποτελεσμάτων προφανώς επειδή δεν μπόρεσε να γίνει κάποια διοχέτευση δεδομένων. Στην συνέχεια εφαρμόστηκε η επιλογή «loop_rewind» ώστε να μην πραγματοποιείται παύση μεταξύ των επαναλήψεων με αποτέλεσμα την μείωση των κύκλων σε 144.318.550.

Στην συνέχεια επιχειρήθηκε ταυτόχρονη εφαρμογή του directive στα «LOOP3», «LOOP5» με τη μορφή που εξήγαγε θετικά αποτελέσματα στην μεμονωμένη εφαρμογή του σ' αυτά, με σκοπό την ταυτόχρονη εκμετάλλευση της βελτιστοποίησης σε δύο περιοχές του αλγορίθμου ταυτόχρονα. Η ταυτόχρονη εφαρμογή λοιπόν στο πρώτο στην απλή μορφή, και στο δεύτερο με επιλογή «loop rewind» εξήγαγε latency μειωμένη κατά έναν κύκλο, λόγω του «loop_rewind» με το οποίο δεν γίνεται η παύση στην τελευταία επανάληψη του «loop5» πριν επανέλθει το πρόγραμμα στην Solver().

Στην συνέχεια έγινε εφαρμογή του directive «unroll» στο «LOOP3» με παράγοντα ίσο με δύο, ώστε να δημιουργηθεί αντίγραφο των υπολογισμών του βρόγχου με σκοπό να μειωθεί το

latency με κόστος την παραγωγή περισσότερου υλικού. Στα αποτελέσματα φαίνεται πως τον αναμενόμενο(estimated) του ρολογιού είναι μεγαλύτερο από τον στόχο(target) γεγονός που είναι απαγορευτικό για την εφαρμογή του directive. Δοκιμάζοντας να αυξηθεί η περίοδος του ρολογιού παρατηρείται πως το πρόβλημα δεν λύνεται, οπότε πιθανόν να εκτελούνται λειτουργίες στον βρόγχο οι οποίες να απαιτούν έναν κύκλο ανεξάρτητα από την διάρκειά του και έτσι δημιουργώντας το αντίγραφο βγαίνει εκτός στόχου ο χρόνος.

Στην συνέχεια δοκιμάστηκε η επιλογή «region» με σκοπό να αντιληφθεί το πρόγραμμα τον βρόγχο ως μια ολόκληρη περιοχή, διατηρώντας τον παράγοντα ανάπτυξης σταθερό καθώς και αυξάνοντας τον παράγοντα ανάπτυξης σε τρία(3) χωρίς να υπάρχει μεταβολή των κύκλων. Συνεπώς το directive αυτό απορρίφθηκε.

Τέλος εφαρμόστηκε το directive «clock» με σκοπό να βρεθεί μια διάρκεια περιόδου η οποία να είναι αρκετά μεγάλη ώστε να ολοκληρώνονται οι απαραίτητες λειτουργίες εντός της, όμως όσο μικρή επιτρέπεται να είναι ώστε να μην υπάρχει αδρανής χρόνος εντός της. Ο χρόνος εκτέλεσης του αλγορίθμου με περίοδο ρολογιού T1 = 10ns είναι t1 = 60.674.698 * 10ns = 606.746.980ns . Κάνοντας χρήση του directive η περίοδος μειώθηκε σε T2 = 8ns , το οποίο μετέβαλλε το latency σε 60.779.583 κύκλους και άρα τον χρόνο εκτέλεσης σε t2 = 60.779.583 * 8ns = 486.236.664ns.

Έπειτα μειώθηκε περισσότερο η περίοδος του ρολογιού σε T3 = 6ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 60.832.253 και του χρόνου εκτέλεσης σε t3 = 60.832.253 * 6ns = 364.993.518ns.

Στην συνέχεια επιχειρήθηκε να μειωθεί και άλλο η περίοδος σε T4 = 5ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 96.964.332 και του χρόνου εκτέλεσης σε t3 = 96.964.332 * 5ns = 484.821.660ns.

Είναι φανερό πως ο χρόνος εκτέλεσης άρχισε να αυξάνετε επειδή κάποιες λειτουργίες λόγω της μικρής διάρκειας του παλμού διαχωρίστηκαν σε δύο παλμούς, γεγονός που αύξησε τον αδρανή χρόνο στον παλμό. Συνεπώς η περίοδος ρολογιού με την καλύτερη χρονική απόδοση ήταν η T3 = 6ns.

Τελικά, η καλύτερη απόδοση του συνδυασμού ήταν οι 60.832.253 κύκλοι με χρόνο περιόδου T3 = 6ns και συνολικό χρόνο εκτέλεσης t3 = 364.993.518ns .

Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.4: Αποτελέσματα πρώτου συνδυασμού

LOOP_TRIPCOUNT	Latency = 144.318.779 κύκλοι Χρόνος εκτέλεσης = 144.318.779 * 10ns = 1.443.187.790 ns
+STREAM	Latency = 144.318.550 κύκλοι Χρόνος εκτέλεσης = 144.318.550 * 10ns = 1.443.185.500 ns
+PIPELINE	Latency = 60.674.698 κύκλοι Χρόνος εκτέλεσης = 60.674.698 * 10ns = 606.746.980 ns
+CLOCK	Latency = 60.832.253 κύκλοι Χρόνος εκτέλεσης = 60.832.853 * 6ns = 364.993.518 ns

2. Δεύτερος Συνδυασμός

Αρχικά γίνεται εφαρμογή του «loop tripcount» εξάγοντας latency ίσο με 144.318.779 και σε περίοδο ρολογιού ίση με 10ns.

Ακολούθησε εφαρμογή του directive «pipeline» για τους ίδιους λόγους με την προηγούμενη υλοποίηση. Η εφαρμογή του έγινε αρχικά στον βρόγχο «LOOP3» χωρίς κάποια επιπρόσθετη επιλογή καθώς και με τις επιλογές «flushing» και «loop rewind». Η εφαρμογή πραγματοποιήθηκε με επιτυχία καθώς χωρίς κάποια επιπρόσθετη επιλογή εξήγαγε μειωμένο latency στους 60.674.928 κύκλους.

Η επιλογή «flushing» δεν μετέβαλλε τον αριθμό των κύκλων γεγονός που σημαίνει πως δεν δημιουργήθηκαν stalls λόγω του pipeline, ενώ η επιλογή «loop_rewind» δεν μπόρεσε να ολοκληρωθεί αφού προέκυψε μήνυμα λάθους λόγω πολλαπλών βρόγχων μέσα στην συνάρτηση. Στην συνέχεια έγινε με τον ίδιο ακριβώς τρόπο εφαρμογή στον βρόγχο «loop5» όπου η απλή εφαρμογή εξήγαγε μειωμένο latency στους 144.266.567 κύκλους, ενώ με την επιλογή «flushing» και «loop_rewind» δεν παρατηρήθηκε κάποια μεταβολή του αριθμού των κύκλων. Συνεπώς δεν δημιουργήθηκαν stalls και δεν υπήρχανε παύσεις μεταξύ των βρόγχων ή υπήρχανε αλλά δεν μπόρεσαν να αφαιρεθούν.

Στην συνέχεια επιχειρήθηκε ταυτόχρονη εφαρμογή του directive στα «LOOP3», «LOOP5» με τη μορφή που εξήγαγε θετικά αποτελέσματα στην μεμονωμένη εφαρμογή του σ' αυτά, με σκοπό την ταυτόχρονη εκμετάλλευση της βελτιστοποίησης σε δύο περιοχές του αλγορίθμου ταυτόχρονα. Η ταυτόχρονη εφαρμογή λοιπόν στο πρώτο και στο δεύτερο σημείο στην απλή μορφή πέτυχε μειώνοντας το latency στους 60.622.716 κύκλους. Συνεπώς η εφαρμογή του συγκεκριμένου directive διατηρείται όπως στην τελευταία δοκιμή.

Στην συνέχεια έγινε εφαρμογή του directive «stream» στην δεύτερη διάσταση του πίνακα rgrid[] για τους ίδιους λόγους με την προηγούμενη υλοποίηση, γεγονός το οποίο αύξησε τον αριθμό των κύκλων σε 60.674.628 και το directive απορρίφθηκε. Προφανώς η εφαρμογή του pipeline κατέστρεψε τον προηγούμενο τρόπο προσπέλασης του πίνακα και δεν μπόρεσε να υλοποιηθεί αποδοτικά ως FIFO.

Στην συνέχεια έγινε εφαρμογή του directive «unroll» στο «LOOP3» με παράγοντα ίσο με δύο για τους ίδιους λόγους με την προηγούμενη υλοποίηση. Στα αποτελέσματα φαίνεται πως τον αναμενόμενο(estimated) του ρολογιού είναι μεγαλύτερο από τον στόχο(target) γεγονός που είναι απαγορευτικό για την εφαρμογή του directive. Δοκιμάζοντας να αυξηθεί η περίοδος του ρολογιού παρατηρείται πως το πρόβλημα δεν λύνεται, οπότε πιθανόν να εκτελούνται λειτουργίες στον βρόγχο οι οποίες να απαιτούν έναν κύκλο ανεξάρτητα από την διάρκειά του και έτσι δημιουργώντας το αντίγραφο βγαίνει εκτός στόχου ο χρόνος.

Στην συνέχεια δοκιμάστηκε η επιλογή «region» για τους ίδιους λόγους με την προηγούμενη υλοποίηση, διατηρώντας τον παράγοντα ανάπτυξης σταθερό καθώς και αυξάνοντας τον παράγοντα ανάπτυξης σε τρία(3) χωρίς να υπάρχει μεταβολή των κύκλων. Συνεπώς το directive αυτό απορρίφθηκε.

Τέλος εφαρμόστηκε το directive «clock» για τους ίδιους λόγους με την προηγούμενη υλοποίηση. Ο χρόνος εκτέλεσης του αλγορίθμου με περίοδο ρολογιού T1 = 10ns είναι t1 = 60.622.716 * 10ns = 606.227.160ns . Κάνοντας χρήση του directive η περίοδος μειώθηκε σε T2 = 8ns , το οποίο μετέβαλλε το latency σε 60.727.830 κύκλους και άρα τον χρόνο εκτέλεσης σε t2 = 60.727.830 * 8ns = 485.822.640ns.

Έπειτα μειώθηκε περισσότερο η περίοδος του ρολογιού σε T3 = 6ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 60.780.500 και του χρόνου εκτέλεσης σε t3 = 60.780.500 * 6ns = 364.683.000ns.

Στην συνέχεια επιχειρήθηκε να μειωθεί και άλλο η περίοδος σε T4 = 5ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 96.912.579 και του χρόνου εκτέλεσης σε t3 = 96.912.579 * 5ns = 484.562.895ns.

Είναι φανερό πως ο χρόνος εκτέλεσης άρχισε να αυξάνετε για τους ίδιους που αναφέρθηκαν στην προηγούμενη υλοποίηση. Συνεπώς η περίοδος ρολογιού με την καλύτερη χρονική απόδοση ήταν η T3 = 6ns.

Τελικά, η καλύτερη απόδοση του συνδυασμού ήταν οι 60.780.500 κύκλοι με χρόνο περιόδου T3 = 6ns και συνολικό χρόνο εκτέλεσης t3 = 364.683.000ns .

Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.5: Αποτελέσματα δεύτερου συνδυασμού

LOOP_TRIPCOUNT	Latency = 144.318.779 κύκλοι Χρόνος εκτέλεσης = 144.318.779 * 10ns = 1.443.187.790ns
+PIPELINE	Latency = 60.622.716 κύκλοι Χρόνος εκτέλεσης = 60.622.716 * 10ns = 606.227.160ns
+CLOCK	Latency = 60.780.500 κύκλοι Χρόνος εκτέλεσης = 60.180.500 * 6ns = 364.683.000 ns

3. Τρίτος Συνδυασμός

Αρχικά γίνεται εφαρμογή του «loop tripcount» εξάγοντας latency ίσο με 144.318.779 και σε περίοδο ρολογιού ίση με 10ns.

Στην συνέχεια έγινε εφαρμογή του directive «stream» στην δεύτερη διάσταση του πίνακα rgrid[] για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις, γεγονός το οποίο μείωσε τον αριθμό των κύκλων σε 144.318.550. Συνεπώς η εφαρμογή του directive διατηρήθηκε.

Στην συνέχεια έγινε εφαρμογή του directive «unroll» για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις με παράγοντα ανάπτυξης ίσο με δύο(2) πετυχαίνοντας αυτήν την φορά μείωση των κύκλων σε 139.546.419, με κόστος το υλικό.

Στην συνέχεια αυξήθηκε ο παράγοντας ανάπτυξης σε τρία(3) γεγονός που προκάλεσε μεγαλύτερη μείωση του αριθμού των κύκλων σε 138.707.363. Στην συνέχεια αυξήθηκε ο

παράγοντας ανάπτυξης σε πέντε(5) γεγονός που προκάλεσε μεγαλύτερη μείωση του αριθμού των κύκλων σε 137.658.543, ενώ μία περεταίρω αύξηση του συντελεστή σε έξι(6) προκάλεσε αύξηση του αριθμού των κύκλων σε 140.175.711 πιθανόν λόγω κάποια συσχέτισης των προσπελάσεων στους πίνακες

Στην συνέχεια έχοντας συντελεστή ανάπτυξης ίσο με πέντε(5) χρησιμοποιήθηκε η επιλογή «region» για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις, χωρίς να προκαλέσει μεταβολή στα αποτελέσματα.

Συνεπώς διατηρήθηκε η εφαρμογή του directive με συντελεστή ανάπτυξης ίσο με πέντε(5) και χωρίς «region».

Τέλος εφαρμόστηκε το directive «clock» για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις. Ο χρόνος εκτέλεσης του αλγορίθμου με περίοδο ρολογιού T1 = 10ns είναι t1 = 137.658.543 * 10ns = 1.376.585.430ns . Κάνοντας χρήση του directive η περίοδος μειώθηκε σε T2 = 8ns , το οποίο μετέβαλλε το latency σε 156.327.539 κύκλους και άρα τον χρόνο εκτέλεσης σε t2 = 156.327.539 * 8ns =1.250.620.312ns.

Έπειτα μειώθηκε περισσότερο η περίοδος του ρολογιού σε T3 = 6ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 168.388.969 και του χρόνου εκτέλεσης σε t3 = 168.388.969 * 6ns = 1.010.333.814ns.

Στην συνέχεια επιχειρήθηκε να μειωθεί και άλλο η περίοδος σε T4 = 5ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 216.477.366 και του χρόνου εκτέλεσης σε t4 = 216.477.366* 5ns = 1.082.386.830ns.

Είναι φανερό πως ο χρόνος εκτέλεσης άρχισε να αυξάνετε για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις σ ’αυτό το βήμα. Συνεπώς η περίοδος ρολογιού με την καλύτερη χρονική απόδοση ήταν η T3 = 6ns.

Τελικά, η καλύτερη απόδοση του συνδυασμού ήταν οι 168.388.969 κύκλοι με χρόνο περιόδου T3 = 6ns και συνολικό χρόνο εκτέλεσης t3 = 1.010.333.814ns .

Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.6: Αποτελέσματα τρίτου συνδυασμού

LOOP_TRIPCOUNT	Latency = 144.318.779 κύκλοι Χρόνος εκτέλεσης = 144.318.779 * 10ns = 1.443.187.790 ns
+STREAM	Latency = 144.318.550 κύκλοι Χρόνος εκτέλεσης = 144.318.550 * 10ns = 1.443.185.500 ns
+UNROLL	Latency = 137.658.543 κύκλοι Χρόνος εκτέλεσης = 137.658.543 * 10ns = 1.376.585.430 ns
+CLOCK	Latency = 168.388.969 κύκλοι Χρόνος εκτέλεσης = 168.388.969 * 6ns = 1.683.889.690 ns

4. Τέταρτος Συνδυασμός

Αρχικά γίνεται εφαρμογή του «loop tripcount» εξάγοντας latency ίσο με 144.318.779 και σε περίοδο ρολογιού ίση με 10ns.

Ακολούθησε εφαρμογή του directive «pipeline» για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις. Η εφαρμογή του έγινε αρχικά στον βρόγχο «LOOP3» χωρίς κάποια επιπρόσθετη επιλογή καθώς και με τις επιλογές «flushing» και «loop rewind». Η εφαρμογή χωρίς κάποια επιπρόσθετη επιλογή έγινε επιτυχημένα και εξήγαγε μειωμένο latency στους 60.674.779 κύκλους. Η επιλογή «flushing» δεν μετέβαλλε τον αριθμό των κύκλων γεγονός που σημαίνει πως δεν δημιουργήθηκαν επιλύσιμα stalls. Στην συνέχεια έγινε με τον ίδιο ακριβώς τρόπο εφαρμογή στον βρόγχο «LOOP5» όπου η απλή εφαρμογή εξήγαγε μειωμένο latency στους 144.266.567 κύκλους, ενώ με την επιλογή «flushing» και «loop_rewind» δεν παρατηρήθηκε κάποια μεταβολή του αριθμού των κύκλων. Συνεπώς δεν δημιουργήθηκαν stalls και δεν υπήρχανε παύσεις μεταξύ των βρόγχων ή υπήρχανε αλλά δεν μπόρεσαν να αφαιρεθούν.

Έτσι στην συνέχεια επιχειρήθηκε ταυτόχρονη εφαρμογή του directive στα «LOOP3», «LOOP5» με τη μορφή που εξήγαγε θετικά αποτελέσματα στην μεμονωμένη εφαρμογή του σ'αυτά, με σκοπό την ταυτόχρονη εκμετάλλευση της βελτιστοποίησης σε δύο περιοχές του αλγορίθμου ταυτόχρονα. Η ταυτόχρονη εφαρμογή λοιπόν στο πρώτο και στο δεύτερο στην

απλή μορφή πέτυχε, εξάγοντας latency μειωμένη στους 60.622.716 κύκλους. Συνεπώς η εφαρμογή του συγκεκριμένου directive διατηρείται όπως στην τελευταία δοκιμή.

Στην συνέχεια έγινε εφαρμογή του directive «unroll» στο «LOOP3» με παράγοντα ίσο με δύο, για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις . Στα αποτελέσματα φαίνεται πως τον αναμενόμενο(estimated) του ρολογιού είναι μεγαλύτερο από τον στόχο(target) γεγονός που είναι απαγορευτικό για την εφαρμογή του directive. Δοκιμάζοντας να αυξηθεί η περίοδος του ρολογιού παρατηρείται πως το πρόβλημα δεν λύνεται, οπότε πιθανόν να εκτελούνται λειτουργίες στον βρόγχο οι οποίες να απαιτούν έναν κύκλο ανεξάρτητα από την διάρκειά του και έτσι δημιουργώντας το αντίγραφο βγαίνει εκτός στόχου ο χρόνος.

Στην συνέχεια δοκιμάστηκε η επιλογή «region» διατηρώντας τον παράγοντα ανάπτυξης σταθερό καθώς και αυξάνοντας τον παράγοντα ανάπτυξης σε τρία(3) χωρίς να υπάρχει μεταβολή των κύκλων. Συνεπώς το directive αυτό απορρίφθηκε.

Τέλος εφαρμόστηκε το directive «clock» για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις. Ο χρόνος εκτέλεσης του αλγορίθμου με περίοδο ρολογιού T1 = 10ns είναι t1 = 60.622.716 * 10ns = 606.227.160ns . Κάνοντας χρήση του directive η περίοδος μειώθηκε σε T2 = 8ns , το οποίο μετέβαλλε το latency σε 60.727.830 κύκλους και άρα τον χρόνο εκτέλεσης σε t2 = 60.727.830 * 8ns = 485.822.640ns.

Έπειτα μειώθηκε περισσότερο η περίοδος του ρολογιού σε T3 = 6ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 60.780.500 και του χρόνου εκτέλεσης σε t3 = 60.780.500 * 6ns = 364.683.000ns.

Στην συνέχεια επιχειρήθηκε να μειωθεί και άλλο η περίοδος σε T4 = 5ns προκαλώντας μεταβολή του αριθμού των κύκλων σε 96.912.579 και του χρόνου εκτέλεσης σε t3 = 96.912.579 * 5ns = 484.562.895ns.

Είναι φανερό πως ο χρόνος εκτέλεσης άρχισε να αυξάνετε για τους ίδιους λόγους με τις προηγούμενες υλοποιήσεις σ’ αυτό το βήμα. Συνεπώς η περίοδος ρολογιού με την καλύτερη χρονική απόδοση ήταν η T3 = 6ns.

Τελικά, η καλύτερη απόδοση του συνδυασμού ήταν οι 60.780.500 κύκλοι με χρόνο περιόδου T3 = 6ns και συνολικό χρόνο εκτέλεσης t3 = 364.683.000ns .

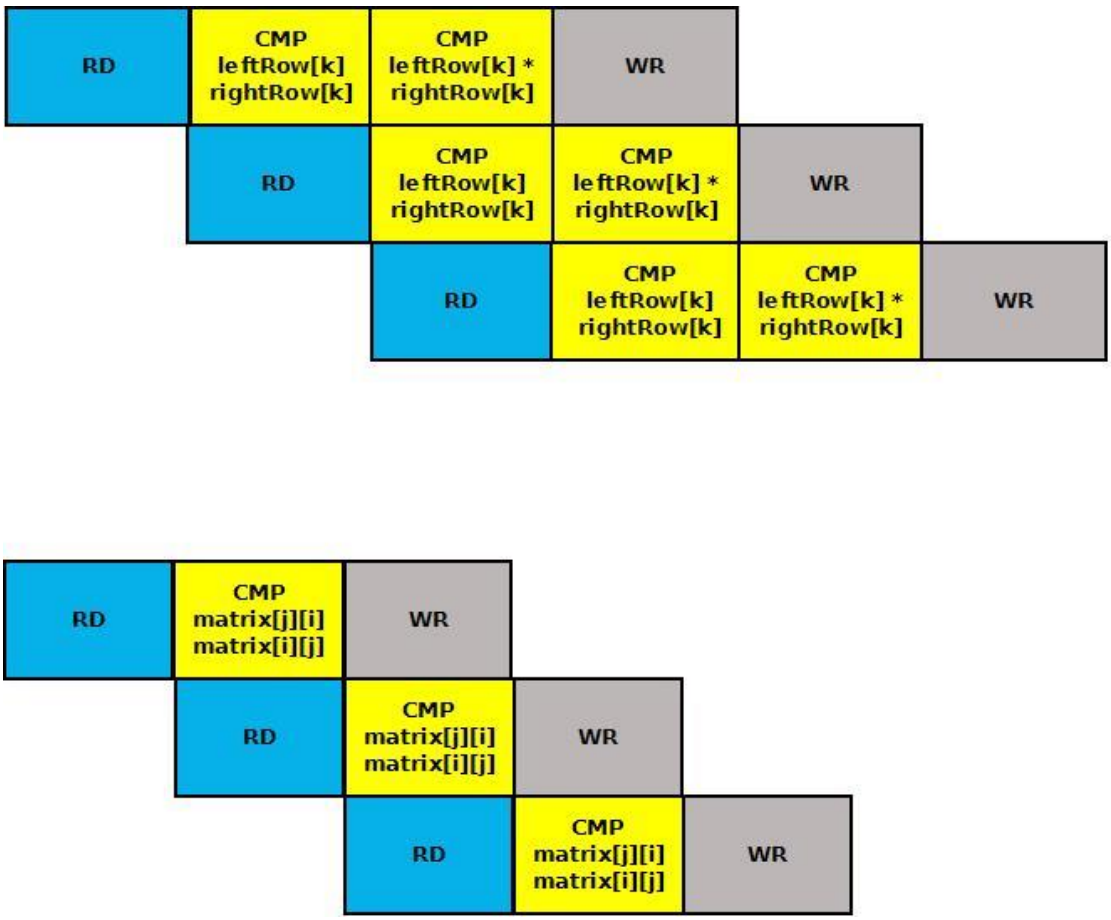
Ακολουθεί πίνακας στον οποίο φαίνεται συγκεντρωτικά η επίδραση κάθε directive που προστέθηκε σε αυτόν τον συνδυασμό, στο latency και στον χρόνο εκτέλεσης.

Πίνακας 5.7: Αποτελέσματα τέταρτου συνδυασμού

LOOP_TRIPCOUNT	Latency = 144.318.779 κύκλοι Χρόνος εκτέλεσης = 144.318.779 * 10ns = 1.443.187.790 ns
+PIPELINE	Latency = 60.622.716 κύκλοι Χρόνος εκτέλεσης = 60.622.716 * 10ns = 606.227.160 ns
+CLOCK	Latency = 60.780.500 κύκλοι Χρόνος εκτέλεσης = 60.780.500 * 6ns = 364.683.000 ns

Εν κατακλείδι, ο δεύτερος και ο τέταρτος συνδυασμός είχαν τον μικρότερο χρόνο εκτέλεσης , ωστόσο ο τέταρτος έχει ένα directive λιγότερο, συνεπώς με την λογική ότι δεν πρέπει να διατηρούνται directives τα οποία δεν έχουν επίδραση στον αλγόριθμό, **ο τέταρτος συνδυασμός θεωρείται ο καλύτερος.**

Παρατίθεται η αρχιτεκτονική του directive «pipeline» για τον καλύτερο συνδυασμό όσο αναφορά τη ροή των δεδομένων. Στον συγκεκριμένο συνδυασμό έχει εφαρμοστεί το directive σε δύο διαφορετικούς βρόγχου οι οποίοι φαίνονται στο σχήμα.



Εικόνα 5.3: Δομή Pipeline directive για τους δύο βρόγχους στην ίδια απεικόνιση

5. Απόδοση στην πλακέτα σε σχέση με το «Synthesis»

Στη συνέχεια γίνεται μία προσπάθεια να δημιουργηθεί μια σχέση μεταξύ των αποτελεσμάτων της διαδικασίας «Synthesis» και «Co-Simulation». Έτσι γίνεται μία σύγκριση για τον τέταρτο συνδυασμό ο οποίος ήταν ο βέλτιστος. Τα αποτελέσματα παραθέτονται για κάθε προσθήκη directive στον συνδυασμό.

Αρχικά μόνο με την προσθήκη των «loop tripcounts» το latency γίνεται ίσο με 144.318.779 κύκλους το οποίο είναι μειωμένο σε σχέση με τους κύκλους της διαδικασίας «Synthesis», οι οποίοι είναι ίσοι με 144.371.449.

Στην συνέχεια προστέθηκε το «pipeline» directive το οποίο εξήγαγε στη διαδικασία «Co-Simulation» latency ίσο με 60.622.716 κύκλους. το οποίο είναι μειωμένο σε σχέση με τους κύκλους του «Synthesis» οι οποίοι είναι ίσοι με 60.649.051.

Τέλος προστέθηκε το «clock» directive το οποίο εξήγαγε στο «Co-Simulation» latency ίσο με 60.780.500 κύκλους. το οποίο είναι μειωμένο σε σχέση με τους κύκλους του «Synthesis» οι οποίοι είναι ίσοι με 60.794.163.

Τα αποτελέσματα του «Co-simulation» παρατίθενται στον πίνακα που ακολουθεί.

Πίνακας 5.8: Αποτελέσματα απόδοσης στο «Synthesis»

LOOP_TRIPCOUNT	Latency = 144.371.449 κύκλοι Χρόνος εκτέλεσης = 144.371.449 * 10ns = 1.443.714.490ns
+PIPELINE	Latency = 60.649.051 κύκλοι Χρόνος εκτέλεσης = 60.649.051 * 10ns = 606.490.510 ns
+CLOCK	Latency = 60.794.163 κύκλοι Χρόνος εκτέλεσης = 60.794.163 * 6ns = 607.941.630 ns

Συμπεράσματα

Παρατηρείται πως πλέον ο καλύτερος συνδιασμός διαφοροποιείται για τους δύο αλγορίθμους. Αυτό που παρουσίαζει μεγάλο ενδιαφέρον ωστόσο είναι η σχέση εκτίμισης απόδοσης μεταξύ της διαδικασίας Synthesis και της διαδικασίας Co-Simulation. Όπως φαίνεται στο παράδειγμα του πίνακα που ακολουθεί, όπου υπάρχουν τα αποτελέσματα για τον καλύτερο συνδυασμό του «Dense-Mutrix Multiplication» και για τις δύο διαδικασίες συγκεντρωτικά, σε κάθε περίπτωση οι κύκλοι μετά την προσθήκη του κάθε directive είναι λιγότεροι στην διαδικασία «Co-Simulation» σε σχέση με την διαδικασία «Synthesis», επιτρέποντας θεωρητικά να οριστεί ένα άνω όριο.

DIRECTIVE	SYNTHESIS (latency)	CO-SIMULATION (latency)
LOOP_TRIPCOUNT	144.371.449 κύκλοι	144.318.779 κύκλοι
+PIPELINE	60.649.051 κύκλοι	60.622.716 κύκλοι
+CLOCK	60.794.163 κύκλοι	60.780.500 κύκλοι

Κεφάλαιο 6: Συμπεράσματα και μελλοντική εργασία

Από την διαδικασία προέκυψαν οι αλγόριθμοι των νάνων βελτιστοποιημένοι ως προς τον χρόνο εκτέλεσής τους. Για την ακρίβεια στον αλγόριθμο «Sparse-Matrix Multiplication» από τον αρχικό χρόνο εκτέλεσης και τον τελικό χρόνο εκτέλεσης μετά την εφαρμογή των κατάλληλων directives, προκύπτει μία χρονική επιτάχυνση σε βαθμό σχεδόν πενταπλάσιο στην ταχύτητας εκτέλεσης. Στον δεύτερο αλγόριθμο «Dense-Matrix Muliplication» από τον αρχικό χρόνο εκτέλεσης και τον τελικό χρόνο εκτέλεσης μετά την εφαρμογή των κατάλληλων directives, προκύπτει μία χρονική επιτάχυνση βαθμού σχεδόν τετραπλάσιου στην ταχύτητας εκτέλεσης.

Είναι φανερό πως σε τέτοιου είδους αλγορίθμους των οποίων το κύριο χαρακτηριστικό είναι οι εμφωλευμένοι βρόγχοι μεταβλητού μεγέθους, καθώς και οι πράξεις μεταξύ πινάκων υπάρχουν συγκεκριμένα directives στα οποία είναι πολύ πιθανόν να προκύψουν θετικές αλλαγές στον χρόνο εκτέλεσής τους. Για αρχή, σε κάθε αλγόριθμο βρόγχων μεταβλητού μεγέθους είναι απαραίτητη η εφαρμογή του «loop_tripcount» ώστε να οριστεί κάποιο (ακόμα και αυθαίρετο) πλήθος επαναλήψεων, ώστε να μπορεί το vivado να παρέχει αποτελέσματα τα οποία μπορεί να αποτελούν και το σημείο αναφοράς για την εφαρμογή των υπόλοιπων directives. Επιπροσθέτως σε κάθε υλοποίηση είναι απαραίτητη η ρύθμιση της περιόδου του ρολογιού με εφαρμογή του directive «CLOCK», καθώς είναι λογικό αρχικά η σχεδίαση να στηρίζεται σε μία προκαθορισμένη τιμή περιόδου, η οποία μπορεί να είναι μεγαλύτερη από την απαραίτητη και να διατηρεί το κύκλωμα σε αδράνεια σε κάποιο διάστημά της. Μεγάλη αύξηση της απόδοσης υπάρχει επίσης με την εφαρμογή του directive «PIPELINE» στους βρόγχους και για την ακρίβεια σε περίπτωση εμφωλευμένων βρόγχων στον πιο εσωτερικό βρόγχο. Εφαρμογή του directive στους εξωτερικούς βρόγχους πιθανόν να μην αυξήσει την απόδοση λόγω μεγάλου αριθμού stalls. Ακόμη, αύξηση της απόδοσης μπορεί να προκαλέσει και η εφαρμογή του directive «UNROLL» στους βρόγχους και σε περίπτωση εμφωλευμένων βρόγχων στο πιο εσωτερικό βρόγχο. Εφαρμογή σε εξωτερικό βρόγχο προκαλεί παραγωγή αντιγράφου του εσωτερικού βρόγχου το οποίο με τη σειρά του

προκαλεί σχεδόν πάντα πρόβλημα με τα όρια του παλμού του ρολογιού. Επίσης το συγκεκριμένο directive συνήθως προκαλεί προβλήματα στα όρια του παλμού του ρολογιού όταν εφαρμόζεται σε συνδυασμό με τον directive «PIPELINE». Τέλος πολύ αποτελεσματικό στην αύξηση της απόδοσης είναι η εφαρμογή του directive «STREAM» σε πίνακες όπου τα δεδομένα τους παράγονται ή διαβάζονται ακολουθιακά, ώστε να υλοποιούνται αποτελεσματικά ως μία FIFO δομή.

Τέλος παρατήρηση που πρέπει να σημειωθεί είναι η σχέση μεταξύ των αποτελεσμάτων της διαδικασίας «Synthesis» και της διαδικασίας «Co-Simulation». Η δεύτερη είναι μια διαδικασία η οποία μπορεί να απαιτεί πολύ χρόνο για την ολοκλήρωσή της. Έτσι κατά την διάρκεια μιας πειραματικής διαδικασίας στην οποία γίνονται δοκιμές και ο χρόνος είναι περιοσρισμένος, η εκτέλεση της διαδικασίας «Co-Simulation» θα ήταν εξαιρετικά χρονοβόρα για το σύνολο της πειραματικής διαδικασίας. Ωστόσο τα αποτελέσματα της διαδικασίας αυτής είναι πάντα πιο αποδοτικά όσο αναφορά το latency σε σχέση με τα αποτελέσματα στην διαδικασία «Synthesis», όπως σημειώθηκε σε προηγούμενο κεφάλαιο. Συνεπώς τα αποτελέσματα της διαδικασίας «Synthesis» μπορούν να λειτουργούν ως ένα άνω όριο σε μία σχεδίαση για να είναι πιο μικρή η χρονική της περίοδος.

Τα αποτελέσματα της παρούσας εργασία μπορούν να χρησιμοποιηθούν μελλοντικά για οποιοδήποτε νάνο ή άλλο αλγόριθμο με παρόμοια δομή, ώστε να πραγματοποιηθεί πιο γρήγορα η πειραματική διαδικασία επιτάχυνσης του. Επίσης μελλοντική εργασία μπορεί να αποτελέσει η επιτάχυνση και των υπόλοιπων νάνων, έχοντας ως στόχο την επιτάχυνση των εφαρμογών οι οποίες χρησιμοποιούν τους νάνους. Κατά συνέπεια θα έπρεπε να γίνει σχετική εργασία και για την επιτάχυνση των εφαρμογών οι οποίες χρησιμοποιούν τους νάνους, τόσο σε επιστημονικό όσο και σε εμπορικό επίπεδο. Οι εφαρμογές αυτές πιθανόν να χρησιμοποιούν παραπάνω από έναν νάνο, συνεπώς θα πρέπει να ερευνηθεί ακόμη ποιος είναι ο ποιο αποτελεσματικός τρόπος επικοινωνία μεταξύ των νάνων στα πλαίσια της εκάστοτε εφαρμογής. Χαρακτηριστικό παράδειγμα για τους νάνους της συγκεκριμένης εργασίας είναι η χρήση τους κατά την παραγωγή γραφικών, το οποίο θα μπορούσε να αποτελεί μία ξεχωριστή ερευνητική εργασία.

Βιβλιογραφία

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley, 2006.

[2] The stuff of Berkeley Design technology. An Independent Evaluation of: High-Level Synthesis Tools for Xilnx FPGAs, 2010.

[3] Anshuman Verma, Ahmed E. Helal, Konstantinos Krommydas, Wu-Chun Feng. Acceleration Workloads on FPGAs via OpenCL: A Case Study with OpenDwarfs, 2016.

[4] Konstantinos Krommydas, Wu-chun Feng, Muhsen Owaida, Christos D. Antonopoulos, Nikolaos Bellas. On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms. University Of California at Berkeley: Department of Electrical Engineering and Computer Sciences, 2015.

[5] Vivado Design Suite User Guide. High-Level Synthesis, November 2015.

[6] Vivado Design Suite Tutorial. High-Level Synthesis, November 2015.

[7] Vivado HLS Improving Performance, November 2013.

[8] W. Feng, H. Lin, T. Scogland, J. Zhang. OpenCL and the 13 Dwarfs: A Work in Progress, Blacksburg USA, 2012.

[9] Konstantinos Krommydas, Wu-chun Feng, Christos D. Antonopoulos, Nikolaos Bellas. OpenDwarfs: Characterization Of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures, New York 2015.

[10] Muhsen Owaida, Nikolaos Bellas, Konstantis Daloukas and Christos D. Antonopoulos. Synthesis of Platform Architectures from OpenCL Programs. In Proccedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, 2011.

[11] Grant Martin, Gary Smith. High-Level Synthesis: Past, Present and Future. IEEE Design and test of Computers, July/August 2009.

[12] S. Windh, X. Ma, R. Halstead, P. Budhkar, Z. Luna, O. Hussaini, W. Najjar. High-Level Language Tools for Reconfigurable Computing. Proceedings of the IEEE, March 2015.

[13] Tomasz S. Czajkowski, David Neto, Michael Kinsner, Utku Aydonat, Jason Wong, Dmitry Denisenko, Peter Yiannacouras, John Freeman, Deshanand P. Singh and Stephen D. Brown. OpenCL for FPGAs: Prototyping a Compiler, 2012.

[14] S. O. Settle, “High-Performance Dynamic Programming on FPGAs with OpenCL, 2013.