ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Μικροεπεξεργαστών και υλικού

# *Μελέτη, σχεδίαση και υλοποίηση αλγορίθμου κατασκευής φυλογενετικών δέντρων σε πλατφόρμα βασισμένη σε αναδιατασσόμενη λογική*

Διπλωματική Εργασία

Αντρέας Κουκκουλλής

**Χανιά 2013**

Επιβλέπων : Καθηγητής Απόστολος Δόλλας

Εξεταστική επιτροπή:
Καθηγητής Απόστολος Δόλλας
Καθηγητής Διονύσιος Πνευματικάτος
Αναπληρωτής καθηγητής Ιωάννης Παπαευσταθίου

# Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Απόστολο Δόλλα που ήταν και ο επιβλέπων της παρούσας διπλωματικής, για την εποικοδομητική συνεργασία που είχαμε σε επίπεδο διπλωματικής και για τον χρόνο που αφιέρωσε για να με κατατοπίσει και να με βοηθήσει.

Ευχαριστώ τον καθηγητή κύριο Διονύσιο Πνευματικάτο και τον αναπληρωτή καθηγητή κύριο Ιωάννη Παπαευσταθίου που δέχτηκαν να αξιολογήσουν την διπλωματική μου διατριβή.

Ευχαριστώ τον Δρ. Ευριπίδη Σωτηριάδη για την υποστήριξή του κατά την διάρκεια της εκπόνησης καθώς και όλα τα μέλη του εργαστηρίου του MHL.

Ευχαριστώ τον κύριο Brian Durwood για την άδεια που μου χορήγησε για να χρησιμοποιήσω το εργαλείο CoDeveloper Impulse C, τον κύριο Scott Thibault που μου έδωσε το Convey Platform Support Package και την υποστήριξη του στην χρήση του εργαλείου CoDeveloper, καθώς και τον κύριο Edward Trexel που με βοήθησε πολλές φορές.

Ευχαριστώ τον Νίκο Αλαχιώτη, για της σημαντικές πληροφορίες που μου έχει δώσει για να λύσω τις απορίες μου.

Πιο πολύ θα ήθελα να ευχαριστήσω τον μεταπτυχιακό φοιτητή και φίλο μου Χρήστο Ρουσόπουλο για την βοήθεια και για τον χρόνο που αφιέρωσε.

Οπωσδήποτε ευχαριστώ τους φίλους και συμφοιτητές μου, Ορέστη, Νίκο, Μαρία, Χρήστο, Μπάμπη, Λιάνα, Κωνσταντίνο, Θεόδωρο.

Τέλος ευχαριστώ τους γονείς μου για την στήριξη σε όλη την διάρκεια των σπουδών μου.

# Περίληψη

Από τις αρχές της δεκαετίας του 1960 άρχισε να επικρατεί η υπόθεση ότι συγκεκριμένες περιοχές του γενετικού υλικού μπορεί να περιέχουν σημαντικές πληροφορίες για την εξέλιξη των ειδών. Σήμερα χρησιμοποιούνται αλγόριθμοι για φυλογενετικές αναλύσεις οι οποίοι είναι ιδιαίτερα ακριβοί υπολογιστικά και απαιτούν πολύ χρόνο να εκτελεστούν.

Στην παρούσα διπλωματική έχει μελετηθεί το πρόγραμμα RAxML, το οποίο χρησιμοποιείται για φυλογενετικές αναλύσεις χρησιμοποιώντας την μέθοδο της Μέγιστης Πιθανοφάνειας(Maximum Likelihood), μιας μεθόδου με μεγάλη ακρίβεια στα αποτελέσματα για αναλύσεις του συγκεκριμένου τύπου. Το μεγαλύτερο μέρος του χρόνου εκτέλεσης του προγράμματος καταναλώνεται στη συνάρτηση υπολογισμού του βαθμού Μέγιστης Πιθανοφάνειας του φυλογενετικού δέντρου που σε μερικές περιπτώσεις ξεπερνά το 90% του συνολικού χρόνου εκτέλεσης.

Με σκοπό την επιτάχυνση του προγράμματος, η εργασία αυτή παρουσιάζει την υλοποίηση της συνάρτησης υπολογισμού Μεγίστης πιθανοφάνειας σε πλατφόρμα αναδιατασσόμενης λογικής, με την βοήθεια του εργαλείου παραγωγής HDL κώδικα από κώδικα γραμμένο σε γλώσσα προγραμματισμού C, CoDeveloper Impulse C.

Το σύστημα σχεδιάστηκε για επιτάχυνση του προγράμματος RAxML για όλους τους αλγορίθμους εκτέλεσής του που χρησιμοποιούν την συνάρτηση υπολογισμού μέγιστης πιθανοφάνειας.

# Περιεχόμενα

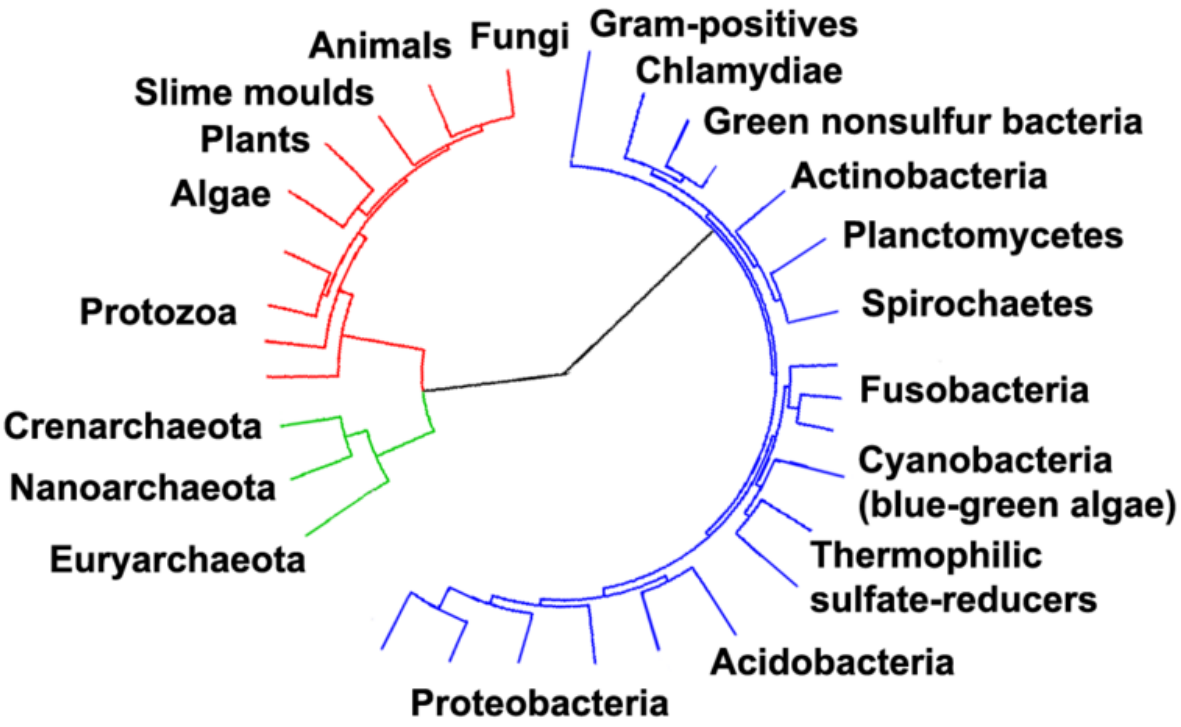
# Λίστα Εικόνων - Πινάκων

# Κεφάλαιο 1. Εισαγωγή

Στο πρώτο κεφάλαιο της διπλωματικής, γίνεται μια εισαγωγή σχετικά με διαδικασία της εξέλιξης. Γίνεται αναφορά στην επιστημονική συνεισφορά της δουλειάς μας και μια περιγραφή της δομής του κειμένου.

## 1.1 Εξελικτική Βιολογία

Η εξελικτική ιστορία της ζωής στη γη δείχνει την διαδικασία με την οποία οι έμβιοι και άβιοι οργανισμοί έχουν εξελιχθεί από την πρώτη στιγμή που εμφανίστηκε ζωή στον πλανήτη. Η γη δημιουργήθηκε περίπου 4.5 δισεκατομμύρια χρόνια πριν και η ζωή στον πλανήτη εμφανίστηκε μετά από ένα δισεκατομμύριο χρόνια. Οι ομοιότητες μεταξύ των ειδών του παρόντος φανερώνουν την ύπαρξη ενός κοινού προγόνου από τον οποίο όλα τα γνωστά είδη που πέρασαν από την Γη έχουν αποκλίνει μέσα από την διαδικασία της εξέλιξης. Όλα τα γνωστά είδη αποτελούν το Δέντρο της ζωής, δηλαδή ένα εξελικτικό δέντρο που παρουσιάζει την σχέση των οργανισμών μεταξύ τους. Μια γενική μορφή του δέντρου της ζωής φαίνεται πιο κάτω με τους οργανισμούς να χωρίζονται σε τρείς βασικές κατηγορίες: τα Βακτήρια (μπλε), τα Αρχαιοβακτήρια (πράσινο) και τα Ευκαρυωτικά (κόκκινο).[33]




Εικ. 1-1 Εξελικτικό δέντρο που δείχνει την εξέλιξη των ειδών από τους κοινούς τους προγόνους

(Πηγη http://en.wikipedia.org/wiki/File:CollapsedtreeLabels-simplified.svg)

## 1.2 Επιστημονική Συνεισφορά και Κίνητρο

Από τις αρχές της δεκαετίας του 1960 επικρατούσε η υπόθεση ότι το γενετικό υλικό περιέχει σημαντικές πληροφορίες για την εξέλιξη των ειδών. Αυτό οδήγησε στην αύξηση των δεδομένων προς επεξεργασία και έγινε απαραίτητη η χρήση αλγορίθμων και εργαλείων για αυτό. Σήμερα υπάρχουν πολλά προγράμματα για φυλογενετικές αναλύσεις χρησιμοποιώντας διάφορες μεθόδους. Η μέθοδος της μέγιστης πιθανοφάνειας έχει αποδειχθεί ότι είναι ένα επαρκές μοντέλο για τον υπολογισμό φυλογενετικών δέντρων. Καθώς αυξάνεται το πλήθος των οργανισμών που θέλουμε να μελετήσουμε το πρόβλημα γίνεται ακόμα πιο δύσκολο. Ο υπολογισμός του βαθμού πιθανοφάνειας (Likelihood score) κάθε πιθανής τοπολογίας είναι υπερβολικά απαιτητικός υπολογιστικά.

Έχοντας σαν κίνητρο τη μείωση εκτελέσεων των αλγορίθμων για την κατασκευή φυλογενετικών δέντρων αποφασίσαμε να υλοποιήσουμε την συνάρτηση

φυλογενετικής πιθανοφάνειας σε αναδιατασσόμενη λογική (Field Programmable Gate Array – FPGA).

Η συνεισφορά της παρούσας εργασίας είναι η εξής:

- Μελέτη του αλγορίθμου του προγράμματος RAxML και αλγοριθμική χρονική ανάλυση της software υλοποίησης.
- Μελέτη και δοκιμή του εργαλείου CoDeveloper Impulse C, το οποίο μετατρέπει κώδικα C σε VHDL.
- Υλοποίηση μέσω του εργαλείου CoDeveloper Impulse C σε VHDL της μεθόδου Μέγιστης πιθανοφάνειας(Maximum Likelihood) του προγράμματος RAxML, που σε μερικές περιπτώσεις ξεπερνά το 90% του συνολικού χρόνου εκτέλεσης του αλγορίθμου.
- Σύνδεση και εκτέλεση της υλοποίησης στην πλατφόρμα Convey.
- Μέτρηση αποτελεσμάτων και σύγκρισή τους με προηγούμενη εργασία στην οποία έχει χρησιμοποιηθεί διαφορετική προσέγγιση.

## 1.3 Σχετική Εργασία

Σχετική εργασία έχει γίνει από τον παλιό φοιτητή του τμήματος και κάτοχο διδακτορικού διπλώματος Δρ. Νίκο Αλαχιώτη στα πλαίσια της δικής του διπλωματικής[3]. Συγκεκριμένα υλοποίησε την συνάρτηση υπολογισμού μέγιστης πιθανοφάνειας σε πλατφόρμα αναδιατασσόμενης λογικής αλλά η εφαρμογή του ακολουθεί διαφορετική προσέγγιση από την δική μας. Επίσης ο ίδιος έχει ασχοληθεί με παρόμοιο θέμα και στο διδακτορικό του[7].

Αυτό που διακρίνει την δουλειά που θα παρουσιαστεί στα επόμενα κεφάλαια από την ήδη υπάρχουσα εργασία, είναι η διαφορετική προσέγγιση της ροής δεδομένων, η χρήση μεταγενέστερης έκδοσης του RAxML και η χρήση πλατφόρμας με περισσότερους πόρους. Επίσης, η σχεδίαση που παρουσιάζεται μπορεί να εκτελεστεί για όλους τους τρόπους λειτουργίας του RAxML καθώς και μπορεί να επεξεργαστεί μεγαλύτερα αρχεία εισόδου σε αντίθεση με την σχεδίαση του Δρ. Αλαχιώτη, που εκτελούσε συγκεκριμένο αλγόριθμο και μόνο για συγκεκριμένα αρχεία εισόδου.

Ο κ. Σταματάκης, δημιουργός του RAxML έχει βελτιώσει τις επιδόσεις του κώδικα με την υλοποίηση του κώδικα σε άλλες αρχιτεκτονικές (SSE3, AVX , multithread version).[4][5][6][8] Οι συγκεκριμένες αρχιτεκτονικές χρησιμοποιούν μεθόδους

για παράλληλο υπολογισμό αποτελεσμάτων για να πετύχουν καλύτερες επιδώσεις.

Τέλος, σχετική εργασία έχει γίνει και στα εργαστήρια του τμήματος από τον κ. Δόλλα και τον κ. Σωτηριάδη, σε συνεργασία με τους κ. Σταματάκη και Νίκο Αλαχιώτη[1][2].

## 1.4 Δομή της Διπλωματικής

Στο πρώτο κεφάλαιο παρουσιάζεται η συνεισφορά της συγκεκριμένης εργασίας. Στο δεύτερο κεφάλαιο γίνεται αναφορά στην υπολογιστική φυλογενετική και τις φυλογενετικές σχέσεις. Στην συνέχεια, στο τρίτο κεφάλαιο παρουσιάζεται ο τρόπος λειτουργίας του προγράμματος RAxML και η μέθοδος της μέγιστης πιθανοφάνειας και τα αποτελέσματα των χρόνων εκτέλεσης διαφόρων λειτουργιών του προγράμματος. Στα κεφάλαια τέσσερα και πέντε παρουσιάζεται η περιγραφή της πλατφόρμας Convey και περιγραφή του εργαλείου CoDeveloper Impulse C που χρησιμοποιήθηκε για την σχεδίαση. Στο έκτο και έβδομο κεφάλαιο γίνεται αναλυτική περιγραφή της υλοποίησης, της χρήσης του εργαλείου Impulse C για την υλοποίηση της συνάρτησης μέγιστης πιθανοφάνειας καθώς και η σύνδεση της σχεδίασης με το Convey.

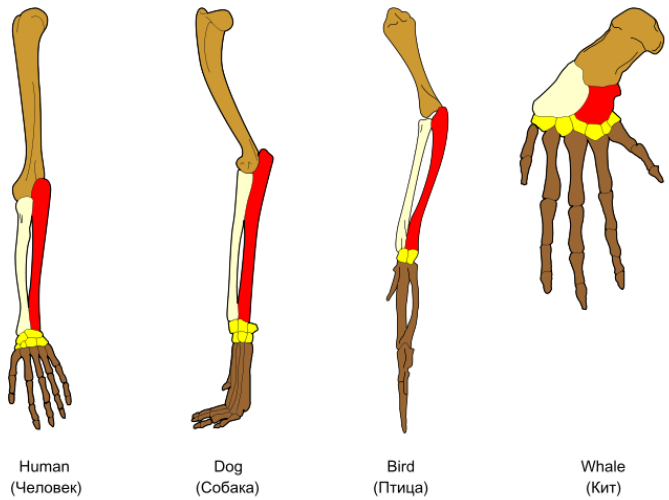
# Κεφάλαιο 2. Φυλογενετικές Σχέσεις

Το κεφάλαιο αυτό περιλαμβάνει το απαραίτητο βιβλιογραφικό υπόβαθρο. Εξηγούνται βασικές έννοιες σχετικά με τα φυλογενετικά δέντρα και ορολογία που θα χρησιμοποιηθεί στα επόμενα κεφάλαια.

## 2.1 Υπολογιστική Φυλογενετική

Υπολογιστική φυλογενετική[34] (computational phylogenetics) είναι η εφαρμογή υπολογιστικών αλγορίθμων και προγραμμάτων για φυλογενετική ανάλυση, με στόχο την σύνθεση ενός φυλογενετικού δέντρου που παρουσιάζει μια υπόθεση για την εξελικτική σχέση μεταξύ ενός σετ από γονίδια, οργανισμών ή άλλων ειδών (taxa). Η παραγωγή ενός φυλογενετικού δέντρου απαιτεί την σύγκριση της ομολογίας(1) ανάμεσα στα χαρακτηριστικά που μοιράζονται τα είδη που συγκρίνονται. Στης μορφολογικές μελέτες τα είδη κατηγοριοποιούνται με βάση τα φυσικά χαρακτηριστικά. Στις βιολογικές μελέτες χρησιμοποιούνται πολλαπλές ακολουθίες γονιδίων ή αμινοξέων, και με διάφορες μεθόδους υπολογίζεται η γενετική απόσταση μεταξύ των ειδών.

(1) Ομολογία (homology) ορίζετε ως «το ίδιο όργανο σε διαφορετικά ζώα σε κάθε μορφή και για κάθε χρήση. [25]



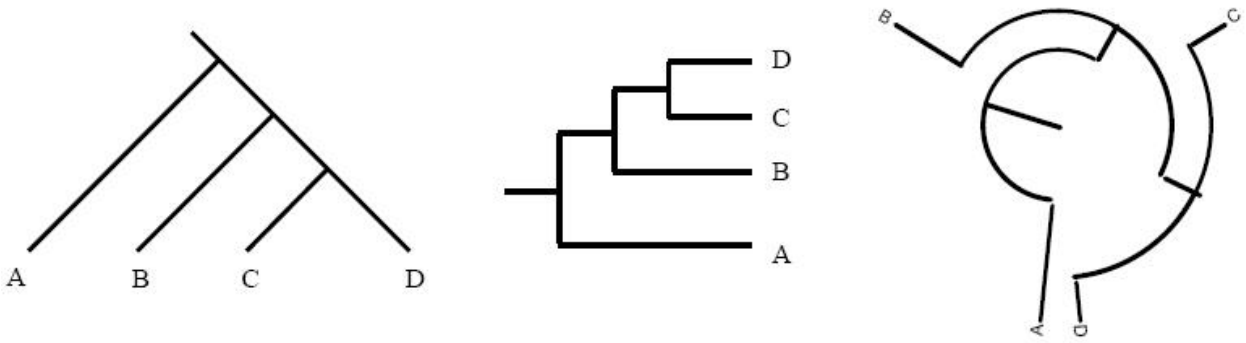

2-1 Η αρχή της ομολογίας. Πηγή /wiki/Homology_(biology).

Η αρχή της ομολογίας: Η βιολογική εξελικτική σχέση (εμφανίζεται με διαφορετικά χρώματα) των διαφόρων οστών των πρόσθιων τεσσάρων

σπονδυλωτών, είναι γνωστή ως ομολογία, και ήταν ένα από τα επιχειρήματα του Δαρβίνου υπέρ της εξέλιξης.

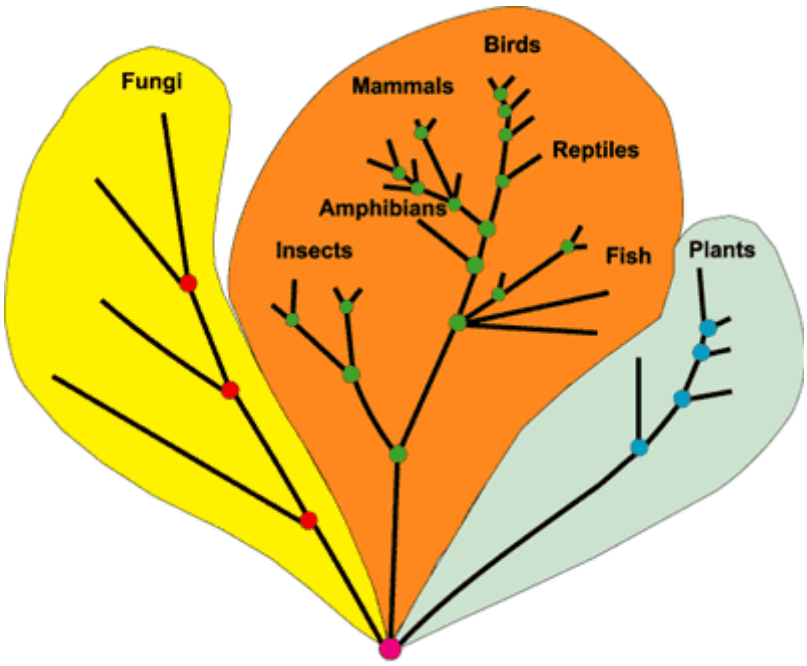
## 2.2 Τι είναι τα φυλογενετικά δέντρα

Ένα φυλογενετικό δέντρο ή αλλιώς φυλογένεια, είναι ένα διάγραμμα που απεικονίζει τις γραμμές τις εξέλιξης των ειδών, οργανισμών ή γονιδίων από ένα κοινό πρόγονο. Τα φυλογενετικά δέντρα είναι χρήσιμα στην οργάνωση της βιολογικής διαφορετικότητας των ειδών και για κατηγοριοποίηση της μορφολογίας τους. Στα φύλλα του δέντρου έχουμε οργανισμούς που είναι νεότεροι από τους άλλους και συνήθως είναι οι οργανισμοί που ακόμα υπάρχουν στον πλανήτη, που δεν έχουν εξαφανιστεί. Οι κόμβοι του δέντρου αναπαριστούν κοινούς προγόνους των ειδών που ίσως να αποτελούν είδη που έχουν εξαφανιστεί. Όσο προχωρούμε προς τα μέσα, τα είδη που συναντούμε είναι όλο και πιο «παλιά», φτάνοντας στο τέλος σε μικροοργανισμούς και μικρόβια που αποτελούν την αρχή της εξελικτικής αλυσίδας.




Εικ 2-2 Το ίδιο δέντρο με διαφορετικούς τρόπους αναπαράστασης.

(Πηγή http://learn/concepts/whatisphylogeny.html)



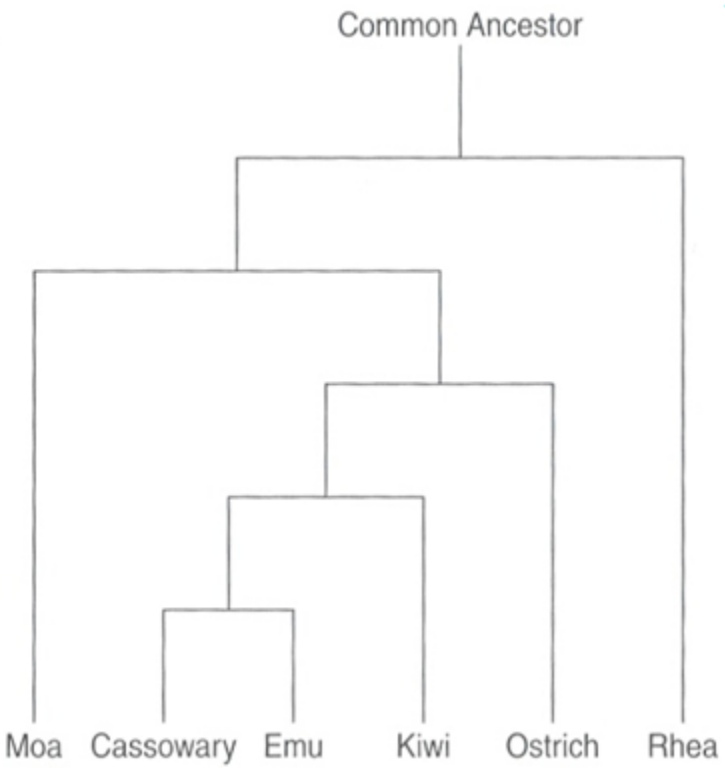

Εικ 2-3 κάθε φύλλο αναπαριστά ένα συγκεκριμένο είδος.

(Πηγή http://toldweb.org/tree/learn/concepts/whatisphylogeny.html)

### 2.2.1 Τύποι Φυλογενετικών Δέντρων

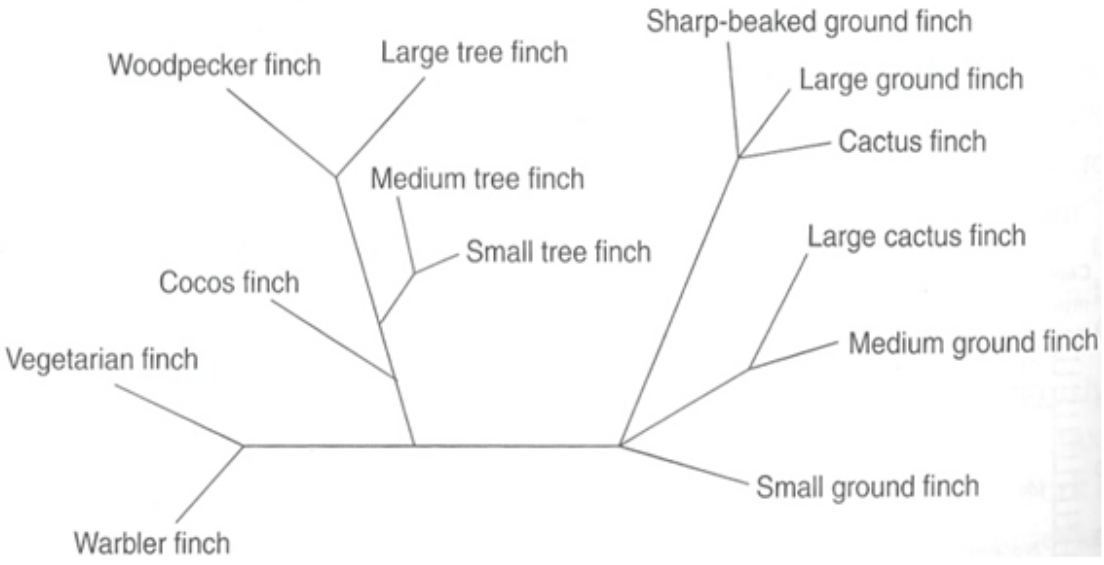
Τα φυλογενετικά δέντρα που παράγονται με μεθόδους υπολογιστικής φυλογενετικής είναι τα δέντρα με ρίζα (rooted) και τα δέντρα χωρίς ρίζα (unrooted), ανάλογα με τον αλγόριθμο που χρησιμοποιείται και από τα δεδομένα εισόδου.

Ένα δέντρο με ρίζα είναι ένας κατευθυνόμενος γράφος που δείχνει τον πρόσφατο κοινό πρόγονο (most recent common ancestor MRCA) και την εξελικτική πορεία. Η γενετική απόσταση των ειδών μπορεί να χρησιμοποιηθεί για να σχεδιαστεί ένα δέντρο με τις ακολουθίες εισόδου σαν φύλλα και η απόσταση τους από την ρίζα χαρακτηρίζει την γενετική απόσταση από τον κοινό τους πρόγονο. Για να γίνει αυτό χρειάζεται εκτός από τις ακολουθίες εισόδου, να δοθεί σαν είσοδο ακόμα μια (τουλάχιστον) ακολουθία που είναι γνωστό ότι είναι πρόγονος των υπόλοιπων ειδών.




Εικ. 2-4 Παράδειγμα δέντρου με ρίζα

Αντίθετα, ένα δέντρο χωρίς ρίζα δείχνει τις αποστάσεις και τις φυλογενετικές σχέσεις μεταξύ των ακολουθιών εισόδου χωρίς να γίνονται υποθέσεις για τον κοινό τους πρόγονο, αλλά υπολογίζεται μόνο η σχέση που έχουν τα είδη μεταξύ τους.




Εικ. 2-5 Παράδειγμα δέντρου χωρίς ρίζα

Ένα δέντρο χωρίς ρίζα μπορεί να παραχθεί από ένα δέντρο με ρίζα, αλλά ένα δέντρο με ρίζα δεν μπορεί να ενσωματωθεί σε ένα δέντρο χωρίς ρίζα χωρίς να χρειαστεί να προσθέσουμε περεταίρω πληροφορία. [26]

Ο αριθμός των δέντρων που μπορεί να προκύψουν μπορεί να υπολογιστεί με τους ακόλουθους τύπους.

Για δέντρα με ρίζα

$$\Delta_{\rho} = \frac{(2n-3)!}{2^{n-2}(n-2)!} \quad \text{για } 2 \leq \text{n}$$

Ενώ για δέντρα χωρίς ρίζα

$$\Delta_{x\rho} = \frac{(2n-5)!}{2^{n-3}(n-3)!} \quad \text{για } 3 \leq \text{n}$$

με n να είναι ο αριθμός των ακολουθιών εισόδου. Στον ακόλουθο πίνακα φαίνεται ο τρόπος που αυξάνεται ο αριθμός των δυνατών δέντρων ανάλογα με την αύξηση των ειδών.

| Αριθμός ειδών | Δέντρα με ρίζα | Δέντρα χωρίς ρίζα |
|---|---|---|
| 2 | 1 | 1 |
| 3 | 3 | 1 |
| 4 | 15 | 3 |
| 5 | 105 | 15 |
| 10 | 34,459,425 | 2,027,025 |
| 15 | 213,458,046,767,875 | 7,905,853,580,625 |
| 20 | 8,200,794,532,637,891,559,375 | 221,643,095,476,699,771,875 |
| | | |

Πίνακας 1 Ρυθμός αύξησης πιθανών δέντρων ανάλογα με το αρχείο εισόδου

Αλλά μόνο ένα δέντρο παρουσιάζει την πραγματική εξελικτική πορεία.

### 2.2.2 Μέθοδοι Κατασκευής Φυλογενετικών δέντρων

Οι μέθοδοι κατασκευής φυλογενετικών δέντρων κατατάσσονται σε δύο βασικές κατηγορίες, την κατηγορία μητρών απόστασης (Distance-matrix methods) και την κατηγορία που βασίζεται στην παρουσία ή απουσία πληροφοριακών χαρακτηριστικών (character – based methods). Στην κατηγορία μητρών απόστασης ανήκουν οι μέθοδοι UPGMA, Fitch-Margoliash και neighbor joining. Στη δεύτερη κατηγορία ανήκουν η μέθοδος της μέγιστης φειδωλότητας (maximum parsimony), η μέθοδος της μπεϊσιανής ανάλυσης (Bayesian analysis) και η μέθοδος της μέγιστης πιθανοφάνειας, η οποία και μας ενδιαφέρει.

# Κεφάλαιο 3. Ανάλυση RAxML

Σε αυτό το κεφάλαιο θα παρουσιαστεί η μέθοδος Maximum Likelihood, ο αλγόριθμος RAxML, ο τρόπος λειτουργίας του προγράμματος, οι παράμετροι που παίρνει, το profiling του αλγορίθμου καθώς και μια περιγραφή των επιμέρους αλγορίθμων που μπορεί να εκτελέσει το πρόγραμμα.

## 3.1 Maximum Likelihood

Maximum Likelihood (ML) είναι μια στατιστική μέθοδος για να ταιριάξουμε ένα μαθηματικό μοντέλο στα δεδομένα που έχουμε. Σύμφωνα με την αρχή της πιθανοφάνειας (likelihood principle), το προτιμώμενο δέντρο είναι αυτό που μεγιστοποιεί την πιθανότητα παρατηρώντας τα δεδομένα, ( πιο δέντρο είναι το πιο πιθανό να προκύψει από τα δεδομένα), αφού τα πιθανά δέντρα που μπορεί να προκύπτουν είναι πάρα πολλά. Η πρώτη εφαρμογή ML σε κατασκευή φυλογενετικών δέντρων χρονολογείται στο 1964.

## 3.2 Το πρόγραμμα RAxML και ο τρόπος λειτουργίας.

Το RAxML (Random Accelerated Maximum Likelihood) είναι ένας φυλογενετικός κώδικας που δημιουργεί ένα δέντρο στο οποίο φαίνεται ποιο είδος είναι συγγενικό με ποιο. Χρησιμοποιείται ευρέως για τν διεξαγωγή μεγάλης κλίμακας φυλογενετικών αναλύσεων χρησιμοποιώντας την μέθοδο της μέγιστης πιθανοφάνειας. Δέχεται σαν είσοδο ένα αρχείο με τα είδη που θέλουμε να ελέγξουμε και δημιουργεί ένα φειδωλό δέντρο χρησιμοποιώντας ένα πρόγραμμα του πακέτου PHYLIP.

Ξεκινά με ευθυγράμμιση πολλαπλών ακολουθιών (matrix of taxa versus characters )

| | |
|---|---|
| Human | AAGCTTCACCGGCGCAGTCATTCTCATAAT... |
| Chimpanzee | AAGCTTCACCGGCGCAATTATCCTCATAAT... |
| Gorilla | AAGCTTCACCGGCGCAGTTGTTCTTATAAT... |
| Orangutan | AAGCTTCACCGGCGCAACCACCCTCATGAT... |
| Gibbon | AAGCTTTACAGGTGCAACCGTCCTCATAAT... |

Ελέγχει το DNA κάθε είδους με όλα τα υπόλοιπα και βρίσκει πόσο διαφέρουν τα είδη μεταξύ τους. Στο παράδειγμα φάνεται ότι ο άνθρωπος από τον χιμπαντζή διαφέρουν σε 3 νουκλεοτίδια. ( οι ακολουθίες είναι αποκομμένες ).

Αφού γίνουν οι συγκρίσεις παράγεται ένα δέντρο που δείχνει την διαδοχή των απογόνων του κάθε είδους. Έπειτα ξεκινάει μια ακολουθία αναδιατάξεων στα υποδέντρα.

Ο τρόπος που κατασκευάζει το δέντρο είναι ο εξής:

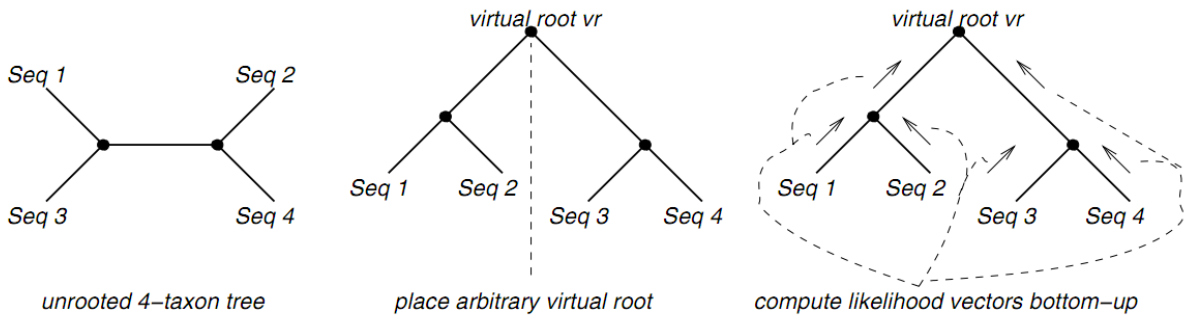
Αρχικά κατασκευάζει ένα δέντρο με τα είδη όπως προκύπτει από την ευθυγράμμιση πιο πριν.


Εικ. 3-1 Δέντρο που παράγεται με ευθυγράμμιση

Στη συνέχεια θέτει μια κορυφή και ακολούθως με διάφορες μεθόδους εναλλάσσει την σειρά που εμφανίζονται τα είδη στο δέντρο ώστε να πάρει την βέλτιστη τοπολογία. Για να υπολογίσει την πιθανότητα ένας κόμβος να αλλάξει θέσει με κάποιο άλλο χρησιμοποιεί ένα πιθανοτικό μοντέλο αλλαγής νουκλεοτιδίων, Ποια είναι δηλαδή η πιθανότητα ένα νουκλεοτίδιο να μεταλλαχθεί σε κάποιο άλλο.

Οι πιθανές αλλαγές μεταξύ των ακολουθιών μπορεί να οφείλονται σε

- Αλλαγή του νουκλεοτιδίου ( μετάλλαξη πχ A -> G)
- Εισαγωγή ή διαγραφή νουκλεοτιδίου
- Διαφοροποιήσεις στη δόμηση (πχ duplication inversion translocation )
- Ανασυνδυασμός και οριζόντια μεταφορά γονιδίων ( bacteria )



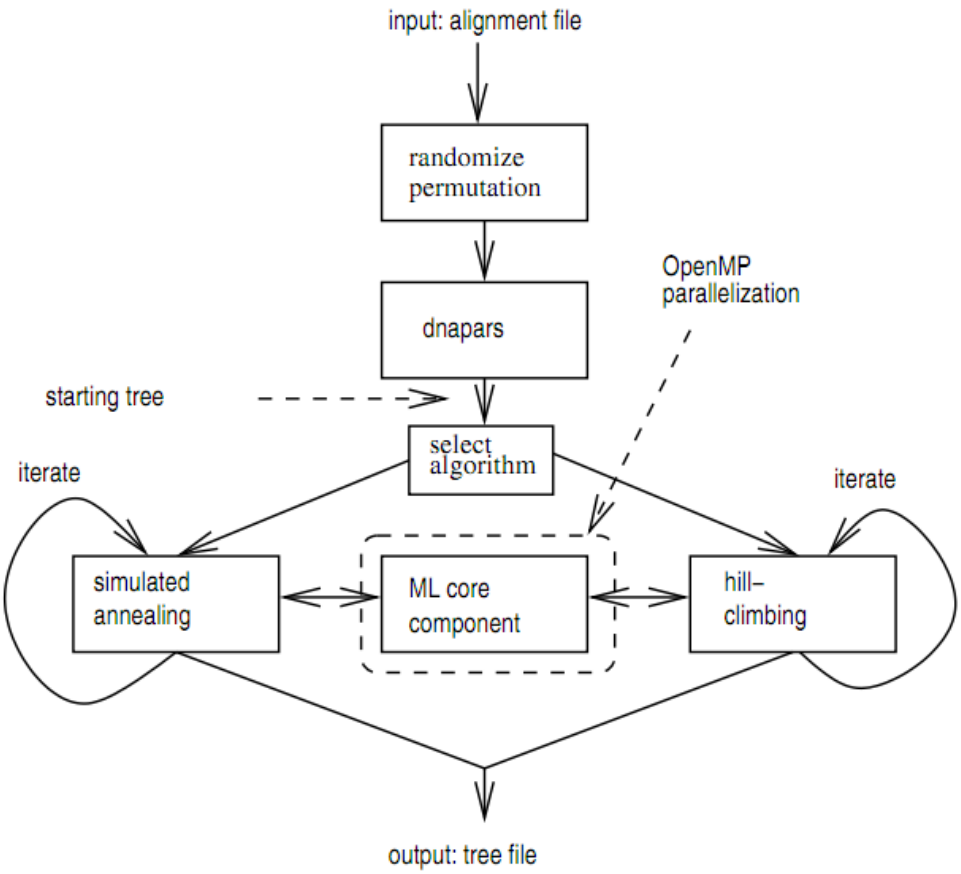

Εικ. 3-2 Διαδικασία υπολογισμού πιθανοφάνειας.

Αν κάποια από τις αναδιατάξεις δημιουργήσει δέντρο με μεγαλύτερο βαθμό πιθανοφάνειας τότε το δέντρο αυτό χρησιμοποιείται σαν δέντρο αναφοράς για τις επόμενες αναδιατάξεις, ξεκινώντας τις αναδιατάξεις από την αρχή.

Η διαδικασία σταματά μετά από πολλές επαναλήψεις όταν μετά από αναδιατάξεις και βελτιστοποιήσεις ικανοποιηθούν κάποια κριτήρια σύγκλισης. Σε κάθε αναδιάταξη κρατούμε μια λίστα με τα 20 καλύτερα δέντρα και στο τέλος της διαδικασίας συνδυάζονται για να προκύψει και προκύπτει το τελικό δέντρο με την μέγιστη πιθανοφάνεια.

Όπως και κάθε άλλο πρόγραμμα που χρησιμοποιείται σε φυλογενετικές αναλύσεις με την μέθοδο της μέγιστης πιθανοφάνειας, έτσι και στο RAxML, το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης καταναλώνεται στον υπολογισμό του βαθμού της μέγιστης πιθανοφάνειας.

## 3.3 Βασικά Μέρη του Αλγορίθμου.




Εικ. 3-3 Βασικά μέρη RAxML

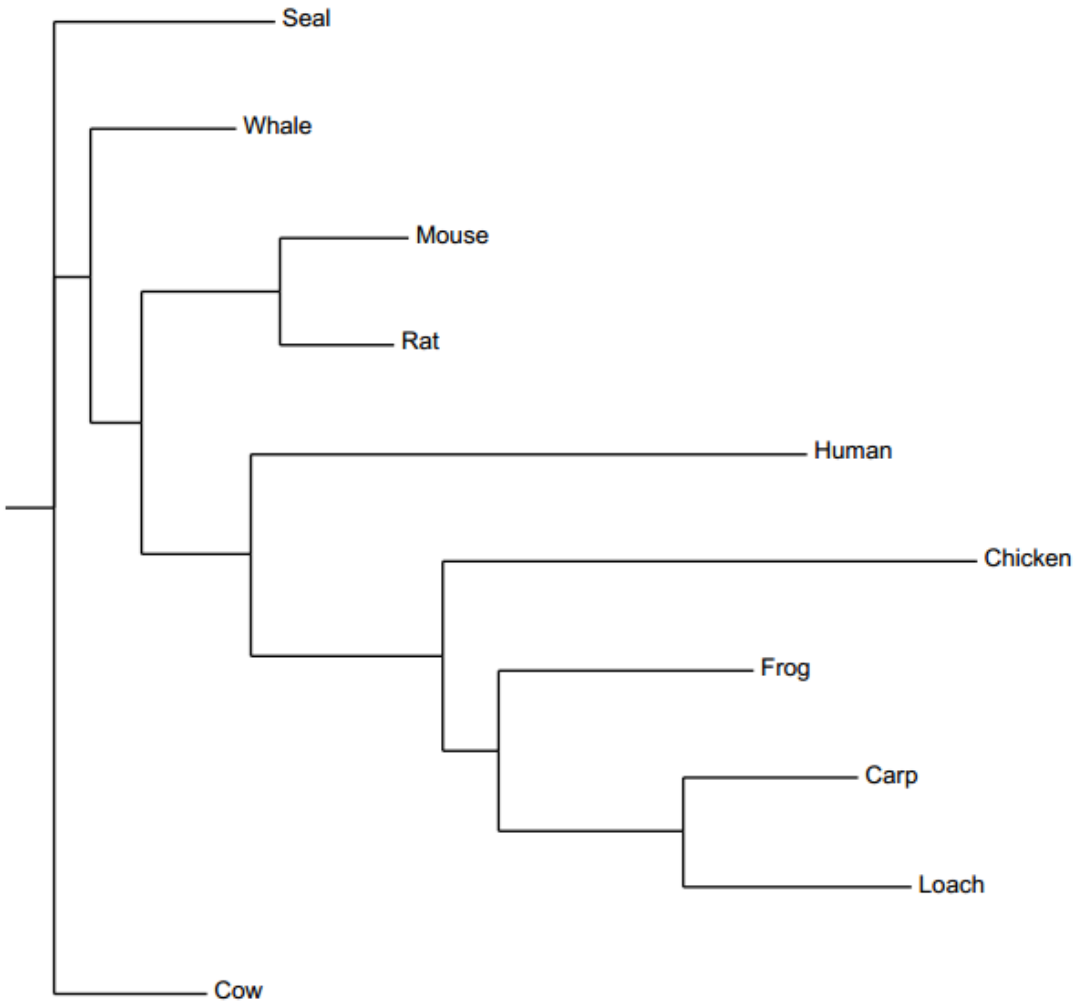
Ο τρόπος που υπολογίζεται η πιθανότητα εμφάνισης του κάθε νουκλεοτιδίου βασίζεται στην μέγιστη πιθανοφάνεια των συμβόλων. Μέγιστη πιθανοφάνεια είναι η από κοινού συνάρτηση πιθανότητας των δειγμάτων. Εφαρμόζεται σε ένα σύνολο δειγμάτων, στην δική μας περίπτωση στο σύνολο των δειγμάτων DNA του αρχείου εισόδου, και με βάση το σύνολο των εμφανίσεων τους υπολογίζεται προσεγγιστικά η πιθανότητα εμφάνισης του κάθε ενός, υποθέτοντας ότι η κατανομή ακολουθεί κάποιο συγκεκριμένο μοντέλο.[27]

Στην έξοδο έχουμε τα αρχεία με τα δέντρα, τα οποία μπορούν να σχεδιαστούν με την βοήθεια online εργαλείων σε γραφική μορφή, εύκολη προς ανάγνωση.

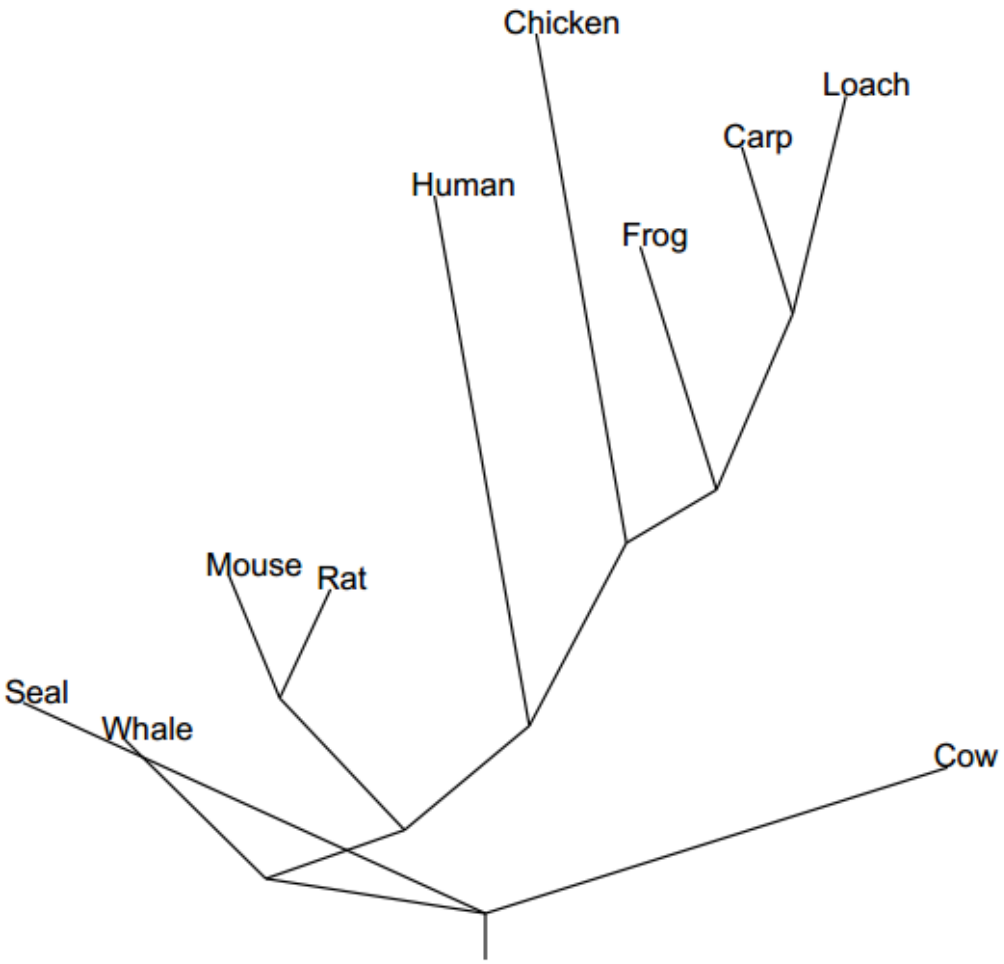
Παράδειγμα αρχείου εξόδου:

(Seal:0.22842223148364010354,(Whale:0.15103525370329479172,((Mouse:0.13099526206556216801,Rat:0.11690757515641610198):0.14315414331843051565,(Human:0.57305835921783387921,(Chicken:0.55194713951884799563,(Frog:0.26335809514180857915,(Carp:0.17803811221351056870,Loach:0.23478747346548728414):0.19077463013579104567):0.05920544600034371302):0.19702486683874867457):0.11212288735131313577):0.05418961973926209735):0.03682831562229804678,Cow:0.15729560645174342493):0.0;

Παράδειγμα του αντίστοιχου δέντρου σε μορφή φενογράμματος και κλαδογράμματος :




Εικ. 3-4 Δέντρο σε μορφή φενογράμματος




Εικ. 3-5 Δέντρο δε μορφή κλαδογράμματος

Και τις δύο εικόνες έχουμε πραγματικό δέντρο που έχει παραχθεί με τα δεδομένα που είχαμε.

## 3.4 Παράμετροι Προγράμματος RAxML

Το πρόγραμμα δέχεται σαν είσοδο ένα αρχείο σε μορφή PHYLIP καθώς και διάφορες επιλογές που διαφοροποιούν το αποτέλεσμα.[B]

Κάθε παράμετρος επηρεάζει την εκτέλεση του προγράμματος. Ακολουθώντας τα παραδείγματα της βιβλιογραφίας αποφασίσαμε να ακολουθήσουμε συγκεκριμένους τρόπους εκτέλεσης, οι οποίοι εκτελούν τις συνηθέστερες λειτουργίες του RAxML .

Ακολουθώντας τα παραδείγματα της βιβλιογραφίας, τις υποδείξεις του δρ. Αλαχιώτη αλλά και προτάσεις από διάφορα forums σχετικά με φυλογενετικούς αλγορίθμους επιλέξαμε συγκεκριμένους τρόπους εκτέλεσης του RAxML, που είναι και οι πιο διαδεδομένοι.

Στην μελέτη μας για την απόδοση του προγράμματος πειραματιστήκαμε μόνο την αλλαγή των κρατώντας όλες τις άλλες παραμέτρους σταθερές. (αρχεία εισόδου, αριθμό επαναλήψεων, μέγεθος αποτελεσμάτων ..)

## 3.5 Profiling Αλγορίθμων

Κατά την διαδικασία του profiling είδαμε πολλούς αλγορίθμους που εκτελούνται με το πρόγραμμα RAxML. Μετά από πολλά πειράματα καταλήξαμε σε συγκεκριμένο σετ δεδομένων για τις μετρήσεις. Οι αλγόριθμοι έχουν εκτελεστεί με την ίδια είσοδο, ούτος ώστε ο χρόνος εκτέλεσης να επηρεάζεται μόνο από την επιλογή του αλγορίθμου και όχι από το αρχείο εκτέλεσης. Για κάθε αλγόριθμο παρουσιάζεται ένας πίνακας με τα αποτελέσματα του profiling που φαίνεται ποιες συναρτήσεις καταναλώνουν τον περισσότερο χρόνο εκτέλεσης. Συνοπτική περιγραφή του κάθε ενός από τα πειράματα γίνεται στην συνέχεια, μαζί με τα αποτελέσματα της εκτέλεσης.

Η περιγραφή των όρων που χρησιμοποιούνται γίνεται στο εγχειρίδιο του RAxML[9].

Το profiling έχει γίνει σε υπολογιστή Pentium core 2 duo 2.0 Ghz με 4GB Ram. Τα πειράματα που διεξήχθηκαν ακολουθούν παρακάτω:

**Πείραμα 1:**

Ο πρώτος αλγόριθμος του προγράμματος RAxML δέχεται σαν είσοδο ένα αρχείο με τις ακολουθίες DNA που θέλουμε να επεξεργαστούμε και παράγει ένα αρχείο με 100 δέντρα. Στη συνέχεια, σε κάθε βήμα δημιουργείται ένα νέο δέντρο σε μια προσπάθεια να βελτιώσει το αποτέλεσμα, κρατώντας το δέντρο με το καλύτερο bootstrap likelihood. Τέλος κάνει διάσχιση του δέντρου χρησιμοποιώντας maximum likelihood search σε μια προσπάθεια να βελτιώσει και άλλο το δέντρο που έχει δημιουργηθεί στο πρώτο βήμα.

Στην έξοδο έχουμε ένα αρχείο με όλα τα δέντρα που έχουν δημιουργηθεί καθώς και 3 αρχεία με το Best- Scoring δέντρο με διάφορα έξτρα δεδομένα στο κάθε ένα.

**Πείραμα 2:**

Ο δεύτερος αλγόριθμος δέχεται σαν είσοδο το Best Scoring three από την προηγούμενη εκτέλεση και με βάση το αρχείο με τα δέντρα που έχουν δημιουργηθεί επίσης από το πρώτο βήμα δημιουργεί ένα αρχείο με το αρχικό δέντρο και κάποιες επιπρόσθετες πληροφορίες. Είναι το τελευταίο βήμα της προηγούμενης εκτέλεσης.

**Πείραμα 3:**

Ο τρίτος αλγόριθμος βελτιώνει τις παραμέτρους μιας δοσμένης τοπολογίας και το μήκος του δέντρου.

Δέχεται σαν είσοδο ένα δέντρο και σαν έξοδο παράγει ένα αρχείο με το βελτιωμένο δέντρο.

Αν η είσοδος είναι το best scoring tree από τα προηγούμενα βήματα δεν το βελτιώνει άλλο. Αν η είσοδος είναι ένα άλλο (συμβατό με την ακολουθία εισόδου) δέντρο, τότε το βελτιώνει. Βελτιώνει το score του δέντρου, δεν κάνει αναδιάταξη των κόμβων.

**Πείραμα 4:**

Ο τέταρτος αλγόριθμος δέχεται σαν είσοδο ένα αρχείο με δέντρα, και υπολογίζει το per-side log Likelihood του κάθε ενός. ( διαβάζει το δέντρο και για κάθε θέση των νουκλεοτιδίων κάθε δείγματος υπολογίζει το log Likelihood). Είναι η «απόσταση» που έχει το κάθε είδος από το άλλο[28].

**Πείραμα 5:**

Ο πέμπτος αλγόριθμος υπολογίζει ένα log likelihood test (SH Test)[29] μεταξύ ενός βέλτιστου δέντρου και ενός αρχείου με άλλα δέντρα.

Δεδομένου ότι το δέντρο εισόδου είναι το καλύτερο δέντρο που μπορούμε να κατασκευάσουμε, προσπαθούμε να υπολογίσουμε «πόσο καλά» είναι τα υπόλοιπα δέντρα.

**Πείραμα 6:**

Ο έκτος αλγόριθμος παράγει πολλά αρχεία με ακολουθίες DNA κάνοντας αλλαγές σε μια αρχική ακολουθία.

**Πείραμα 7:**

Ο έβδομος αλγόριθμος υπολογίζει το log likelihood των δέντρων σε ένα αρχείο εισόδου.

**Πείραμα 8:**

Ο όγδοος αλγόριθμος εκτελεί ένα προκαθορισμένο αριθμό από αναζητήσεις σε ένα αρχείο με δέντρα ξεκινώντας πάντα από ένα αρχικό δέντρο. Βελτιώνει τα score των δέντρων. Παράγει ένα αρχείο για κάθε βελτιωμένο δέντρο.

**Πείραμα 9:**

Ο ένατος αλγόριθμος εκτελεί ένα ELW- TEST [26] σε όλα τα δέντρα εισόδου. Βρίσκει το δέντρο με το καλύτερο score και ακολούθως ταξινομεί τα δέντρα του αρχείου εισόδου με βάση τις a posteriori πιθανότητες εμφάνισης των στοιχείων του κάθε δέντρου έχοντας ως δεδομένο ότι τα δέντρα ίσως να είναι κακώς προσδιορισμένα.

**Πείραμα 10:**

Ο δέκατος αλγόριθμος δέχεται σαν είσοδο ένα δέντρο και υπολογίζει την «απόσταση» (ML-based pair-wise distance) του κάθε είδους με όλα τα υπόλοιπα.

Στον πίνακα που ακολουθεί βλέπουμε τον συνολικό χρόνο εκτέλεσης του κάθε αλγορίθμου και τις συναρτήσεις που καταναλώνουν το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης.

| | | | **Υπολογιστικά χρονοβόρες συναρτήσεις** | | |
|---|---|---|---|---|---|
| **Αλγόριθμος** | **Dataset** | **Συνολικός χρόνος εκτέλεσης** | **#1** | **#2** | **#3** |
| a | 50 ακολουθίες DNA | 598 sec | newviewIterative | execCore | newviewGTRGAMMA |
| b | 1 δέντρο με 50 είδη | αμελητέος | | | |
| e | 1 δέντρο με 50 είδη | 7.4 sec | newviewGTRGAMMA | execCore | |
| g | 50 ακολουθίες DNA | 38 sec | newviewGTRGAMMA | execCore | makenewzIterative |
| h | 1 δέντρο με 50 είδη | 37 sec | newviewGTRGAMMA | execCore | makenewzIterative |
| j | 50 ακολουθίες DNA | αμελητέος | | | |
| n | 1 δέντρο με 50 είδη | 37.25 sec | newviewGTRGAMMA | execCore | makenewzIterative |
| t | 1 δέντρο με 50 είδη | 5868 sec | newviewGTRGAMMA | execCore | makenewzIterative |
| w | 1 δέντρο με 50 είδη | 2231 sec | newviewGTRGAMMA | execCore | makenewzIterative |
| x | 1 δέντρο με 50 είδη | 7 sec | newviewGTRGAMMA | execCore | makenewzIterative |

Πίνακας 9 Υπολογιστικά χρονοβόρες συναρτήσεις

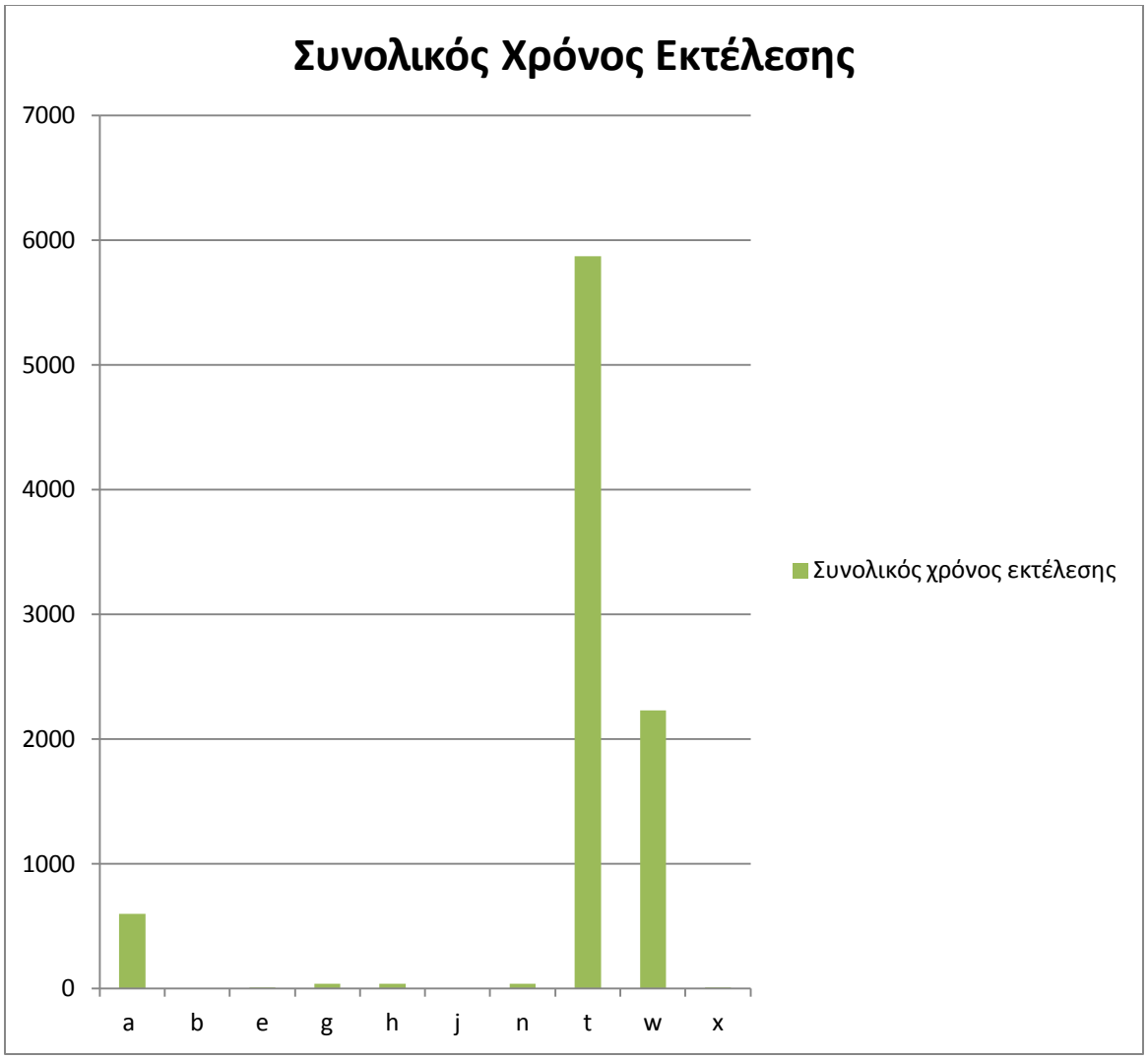
Στον επόμενο πίνακα φαίνεται το ποσοστό του χρόνου που καταναλώνεται στην κρίσιμη συνάρτηση για κάθε αλγόριθμο.

| **Αλγόριθμος** | **Συνολικός χρόνος εκτέλεσης** | **Κρίσιμη συνάρτηση** | **Συνολικός Χρόνος εκτέλεσης συνάρτησης** | **% χρόνου** | **Αριθμός κλήσεων** |
|---|---|---|---|---|---|
| a | 598 sec | newviewItera tive | 239 | 59.59 | 7798174 |
| b | αμελητέος | | | | |
| e | 7.4 sec | newviewGTR GAMMA | 6,59 | 92,3 | 61392 |
| g | 38 sec | newviewGTR GAMMA | 28 | 79.2 | 254127 |
| h | 37 sec | newviewGTR GAMMA | 29 | 79 | 258352 |
| j | αμελητέος | | | | |
| n | 37.25 sec | newviewGTR GAMMA | 29 | 79.58 | 258675 |
| t | 5868 sec | newviewGTR GAMMA | 4245 | 75.6 | 36864714 |
| w | 2231 sec | newviewGTR GAMMA | 1666.30 | 77.63 | 21349104 |
| x | 7 sec | newviewGTR GAMMA | 6.82 | 90.81 | 61776 |

Πίνακας 10 Κρίσιμη συνάρτηση

## 3.6 Παρατηρήσεις Αποτελεσμάτων Profiling

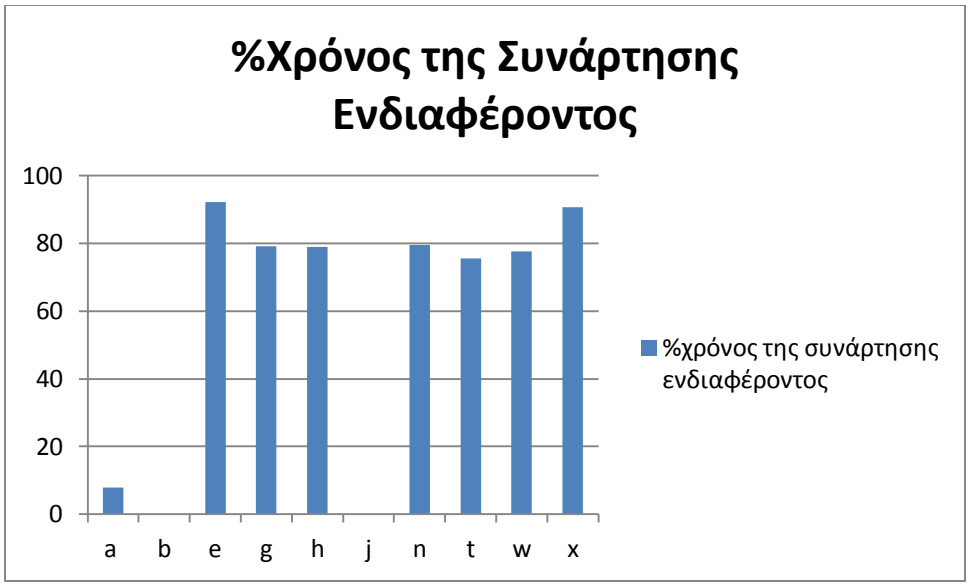
Όπως βλέπουμε στους πίνακες πιο πάνω, καθώς και στα διαγράμματα, παρατηρούμε ότι η συνάρτηση που εμφανίζεται με μεγαλύτερη βαρύτητα είναι η συνάρτηση **newviewGTRGAMMA().** Καλείτε τις περισσότερες φορές και το ποσοστό του χρόνου που καταναλώνει είναι πολύ μεγαλύτερο σε σχέση με την αμέσως επόμενη συνάρτηση.




Διάγραμμα 11 Συνολικός Χρόνος εκτελεσης

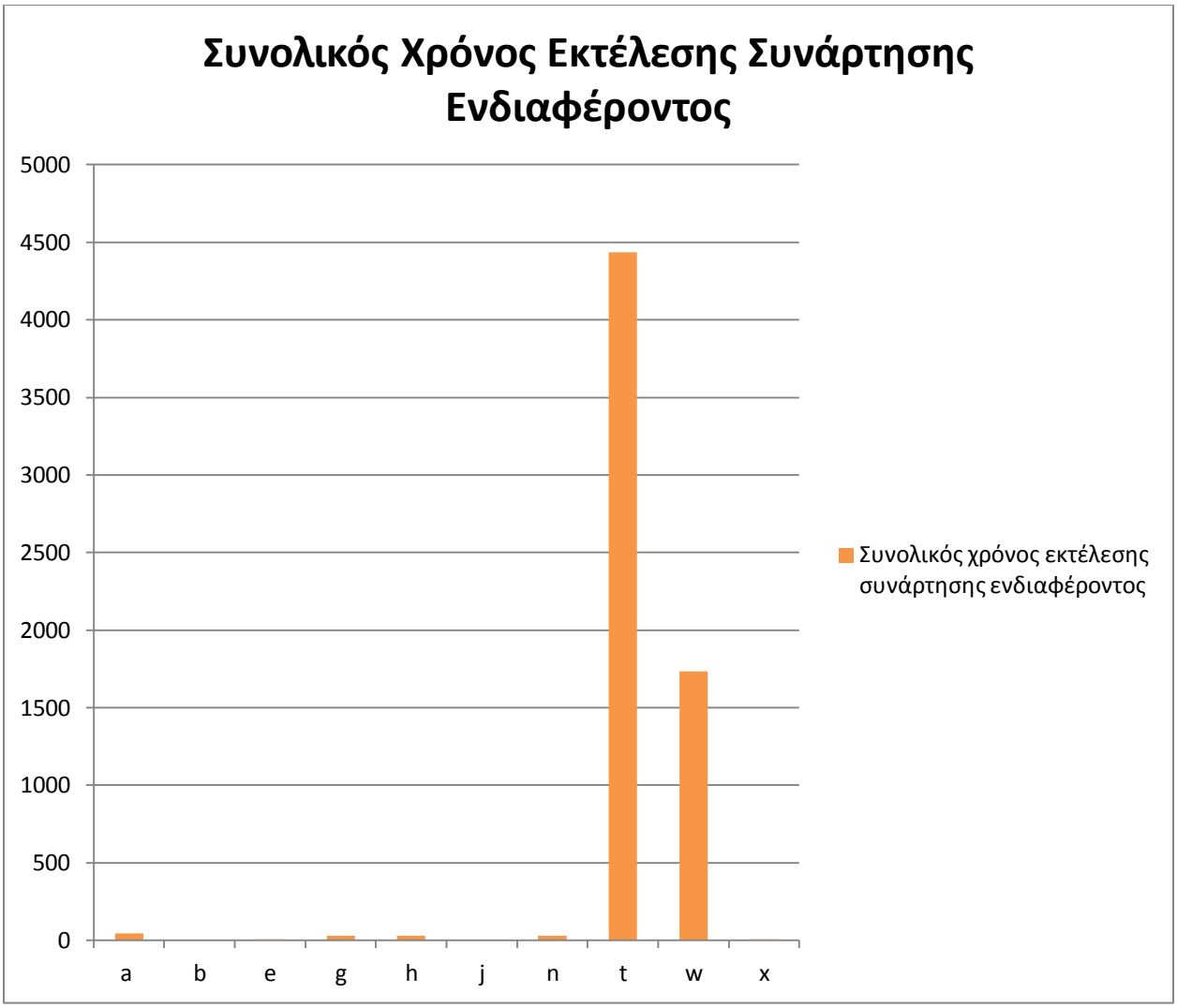
Στο διάγραμμα βλέπουμε ότι ο χρόνος εκτέλεσης είναι πολύ μεγάλος για ορισμένες εκτελέσεις του κώδικα. Ο οριζόντιος άξονας δείχνει τον αλγόριθμο εκτέλεσης και ο κάθετος άξονας τον χρόνο σε δευτερόλεπτα που απαιτείται για να εκτελεστεί.

Σε όλες ουσιαστικά της εκτελέσεις του προγράμματος, η συνάρτηση που καλείται τις πιο πολλές φορές είναι η **newviewGTRGAMMA** και στις περισσότερες περιπτώσεις καταναλώνει μεγάλο μέρος του συνολικού χρόνου εκτέλεσης, πάνω από 75% για εκτελέσεις που χρειάζονται πολλή ώρα να εκτελεστούν αλλά και πάνω από 90 % σε μερικές εκτελέσεις που ο συνολικός χρόνος εκτέλεσης όμως είναι μικρός.




Διάγραμμα 12 Ποσοστό χρόνου εκτέλεσης της συνάρτησης ενδιαφέροντος

Στο διάγραμμα φαίνεται το ποσοστό του χρόνου που καταναλώνει η κρίσιμη συνάρτηση **newviewGTRGAMMA** σε σχέση με τον συνολικό χρόνο εκτέλεσης του προγράμματος.



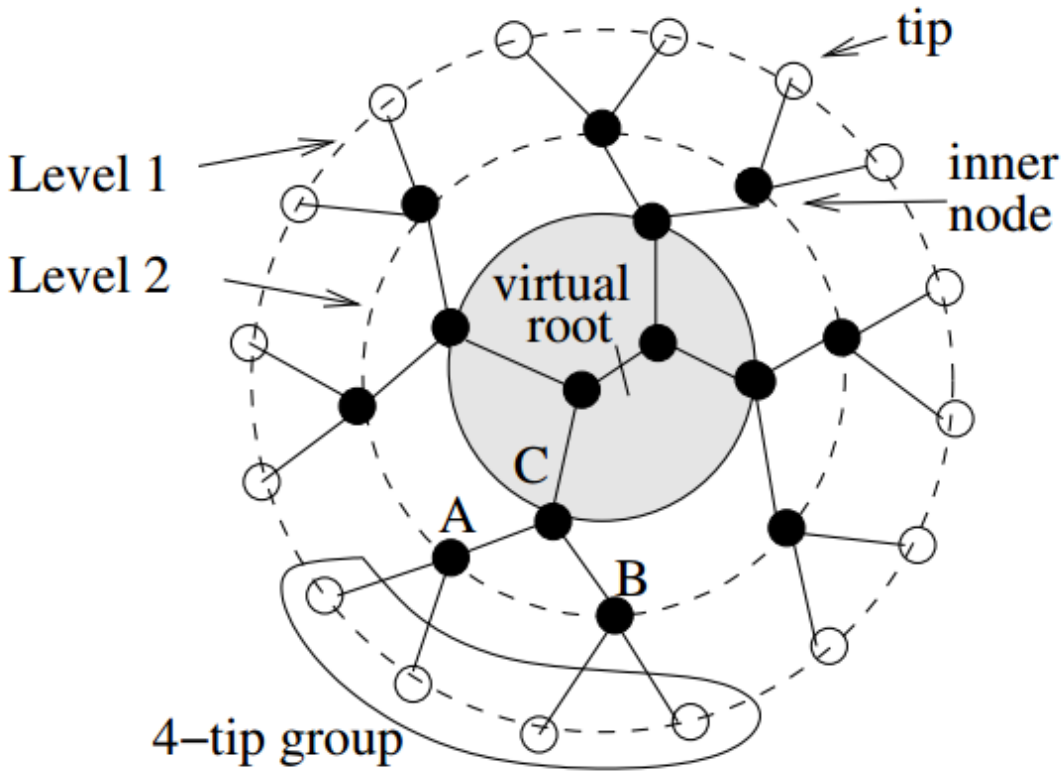

Διάγραμμα 13 Χρόνος που καταναλώνεται μέσα στην κρίσιμη συνάρτιση.

Σε αυτό το διάγραμμα φαίνεται η σχέση του συνολικού χρόνου εκτέλεσης με το ποσοστό του χρόνου που καταναλώνεται στην κρίσιμη συνάρτηση(συνολικός χρόνος * % χρόνος συνάρτησης). Μπορούμε να δούμε ότι για δύο εκτελέσεις του κώδικα ο χρόνος που καταναλώνεται στην εκτέλεση της κρίσιμης συνάρτησης είναι μεγάλος, επομένως η προσπάθεια για επιτάχυνση του προγράμματος έχει σημασία.

Παρατηρώντας τα παραπάνω, και λόγω του ότι η συνάρτηση **newviewGTRGAMMA** καταναλώνει το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης (70% - 90%) αποφασίσαμε να την μελετήσουμε και να την απεικονίσουμε σε hardware με σκοπό την επιτάχυνση του αλγορίθμου.

## 3.7 Περιγραφή Λειτουργίας Συνάρτησης newviewGTRGAMMA

Η συνάρτηση δέχεται σαν όρισμα το likelihood score του δέντρου και με πολλές επαναλήψεις υπολογίζει την πιθανότητα ένας κόμβος να αλλάξει θέση με κάποιο άλλο. Όπως φαίνετε στην εικόνα 3-1, τα δέντρα αποτελούνται από εσωτερικούς και εξωτερικούς κόμβους. Οι εξωτερικοί κόμβοι(tip) αντιπροσωπεύουν ένα είδος, ενώ οι εσωτερικοί (ιννερ) κόμβοι αναπαριστούν τον (πιθανό) κοινό πρόγονο των ειδών-παιδιών τους. Για να υπολογιστεί το βέλτιστο δέντρο συγκρίνει το likelihood score των κόμβων. Σε κάθε κλήση της συνάρτησης επιλέγεται το case σύγκρισης, ανάλογα με το είδος των κόμβων που συγκρίνουμε. Μπορούμε να συγκρίνουμε εξωτερικό κόμβο με εξωτερικό(TIP_TIP), εξωτερικό με εσωτερικό (TIP_INNER) και εσωτερικό με εσωτερικό (INNER_INNER). Τα tip nodes είναι δεδομένα που αφορούν τα είδη εισόδου και τα inner nodes αφορούν τους κοινούς απογόνους των ειδών όπως υπολογίζονται από την συνάρτηση πιθανοφάνειας.




Εικ. 3-6 Αναπαράσταση θέσης κόμβων. Level 1 – tip, level 2 – inner

Από τον κώδικα της συγκεκριμένης συνάρτησης, παρατηρήθηκε ότι ο χρόνος εκτέλεσης εξαρτάται από την είσοδο *width* της συνάρτησης, το οποίο είναι ο αριθμός των alignment pattern των ακολουθιών DNA του αρχείου εισόδου. Στο παράδειγμα της εικόνας παρακάτω έχουμε ένα σετ ακολουθιών DNA.

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

Εικ. 3-7 Ακολουθίες εισόδου

Σε κάθε γραμμή έχουμε τον γενετικό κώδικα ενός είδους. Για να υπολογιστεί το alignment pattern του συγκεκριμένου σετ δεδομένων πρέπει να βρούμε πόσα κομμάτια γενετικού κώδικα είναι όμοια σε όλα τα είδη. Στην επόμενη εικόνα βλέπουμε ότι η υποακολουθία ATGCAACT είναι κομμάτι του γενετικού υλικού όλων των ειδών, άρα η υποακολουθία μπορεί να θεωρηθεί σαν ένα κομμάτι.

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

Εικ. 3-8 Κοινή υποακολουθία σε όλες τις ακολουθίες

Το alignment pattern είναι το πλήθος των διαφορετικών υποακολουθιών που συναντούμε σε ένα σετ ακολουθιών εισόδου. Ο μέγιστος αριθμός που μπορεί να φτάσει είναι το μήκος της ακολουθίας, που είναι και η χειρότερη περίπτωση, αν δηλαδή οι ακολουθίες DNA δεν έχουν κοινές υποακολουθίες. Για παράδειγμα οι ακολουθίες:

ACGT
TGCA

έχουν alignment pattern = 4, όσο το μήκος τους αφού δεν έχουμε υποακολουθίες με μήκος μεγαλύτερο του 1 που να εμφανίζονται και στα δύο δείγματα DNA. Εξαρτάται από το μέγεθος του αρχείου εισόδου και το μήκος των ακολουθιών.

Μετά από επιβεβαίωση του Δρ. Νίκου Αλαχιώτη ότι τα αποτελέσματά μας είναι σωστά συνεχίσαμε με την σχεδίαση της συγκεκριμένης συνάρτησης σε Hardware.

# Κεφάλαιο 4. CoDeveloper Impulse C και Πλατφόρμα Convey Computer

Η γλώσσα προγραμματισμού Impulse C είναι ένα υποσύνολο της γλώσσας C συνδυασμένη με βιβλιοθήκες συμβατές με την γλώσσα C που υποστηρίζουν παράλληλο προγραμματισμό συγκεκριμένα για να προγραμματίζονται εφαρμογές που στοχεύουν συσκευές FPGA. Έχει αναπτυχθεί από την εταιρεία Impulse Accelerated Technologies που εδρεύει στην Washington.[32]

## 4.1 Γενικές Πληροφορίες

Το high level synthesis εργαλείο CoDeveloper περιλαμβάνει ένα compiler της Impulse C και τις σχετικές συναρτήσεις που είναι απαραίτητες για την ανάπτυξη εφαρμογών βασισμένες στην τεχνολογία που υποστηρίζουν οι FPGAs. Η Impulse C είναι συμβατή με το πρότυπο ANSI C, επιτρέποντας σε συνηθισμένα εργαλεία C να χρησιμοποιηθούν για την σχεδίαση και την αποσφαλμάτωση εφαρμογών που σχεδιάζονται για FPGAs. Ο compiler της Impulse C δέχεται ένα κομμάτι κώδικα C και παράγει ένα ισοδύναμο hardware για FPGA στη μορφή ενός αρχείου HDL (Hardware Description Language). Η Impulse C επιτρέπει στους προγραμματιστές ενσωματωμένων συστημάτων και προγραμματιστές software εφαρμογών να στοχεύσουν συσκευές FPGA για να επιταχύνουν εφαρμογές που είναι γραμμένες σε C.

Αυτό που διακρίνει την Impulse C από την standard C είναι το γεγονός ότι επιτρέπει τον παράλληλο προγραμματισμό για εφαρμογές που τρέχουν σε FPGA και επεξεργαστή. Για τον σκοπό αυτό, η Impulse C περιέχει extensions σε C στην μορφή συναρτήσεων και τύπων δεδομένων, επιτρέποντας την εφαρμογή που είναι γραμμένη σε C να ξανασχεδιαστεί με παράλληλη αρχιτεκτονική που μπορεί να περιλαμβάνει ένα επεξεργαστή σε συνδυασμό με προγραμματιζόμενο FPGA hardware.

Το εργαλείο CoDeveloper περιλαμβάνει hardware/software co-simulation καθώς και τεχνολογία χρονοπρογραμματισμού/βελτιστοποίισης για να κάνει map τα κομμάτια της εφαρμογής σε hardware με εργαλεία σύνθεσης FPGA.

## 4.2 Προγραμματιστικό Μοντέλο

Η Impulse C υποστηρίζει πολλές από τις διαδικασίες σειριακής επικοινωνίας (communicating sequential processes) του προγραμματιστικού μοντέλου,

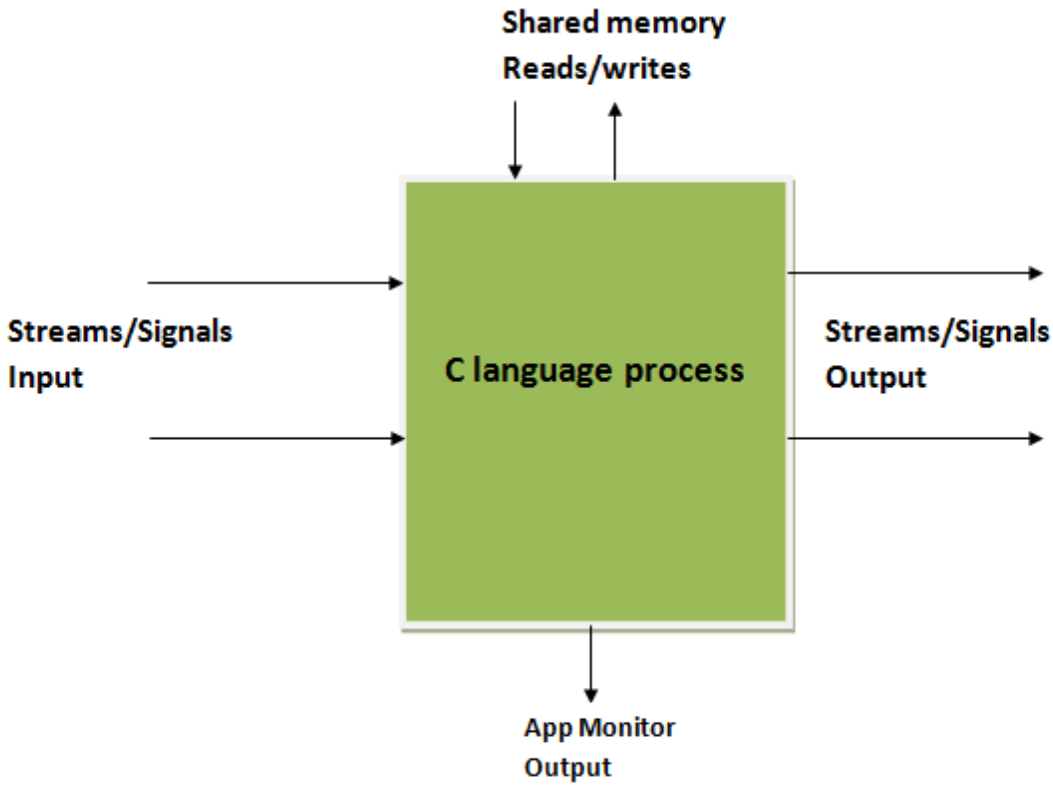
παραμένοντας συμβατή με τους κλασσικούς compilers και profilers της C. Έχει σχεδιαστεί για dataflow-oriented εφαρμογές, όπως επίσης και για να υποστηρίζει και εναλλακτικά προγραμματιστικά μοντέλα συμπεριλαμβανομένης και της χρήσης της κοινής μνήμης (Shared memory) σαν μέσο επικοινωνίας.

Σε μια εφαρμογή Impulse C, οι hardware διαδικασίες και οι software διαδικασίες επικοινωνούν μέσω buffered data streams που είναι υλοποιημένα απευθείας στο hardware. Το buffering των δεδομένων υλοποιείται με την χρήση dual-clock FIFOs που τις παράγει ο compiler, δίνοντας την δυνατότητα να γράψουμε παράλληλες εφαρμογές σε high level abstraction, χωρίς να είναι απαραίτητος ο συγχρονισμός κύκλο προς κύκλο.

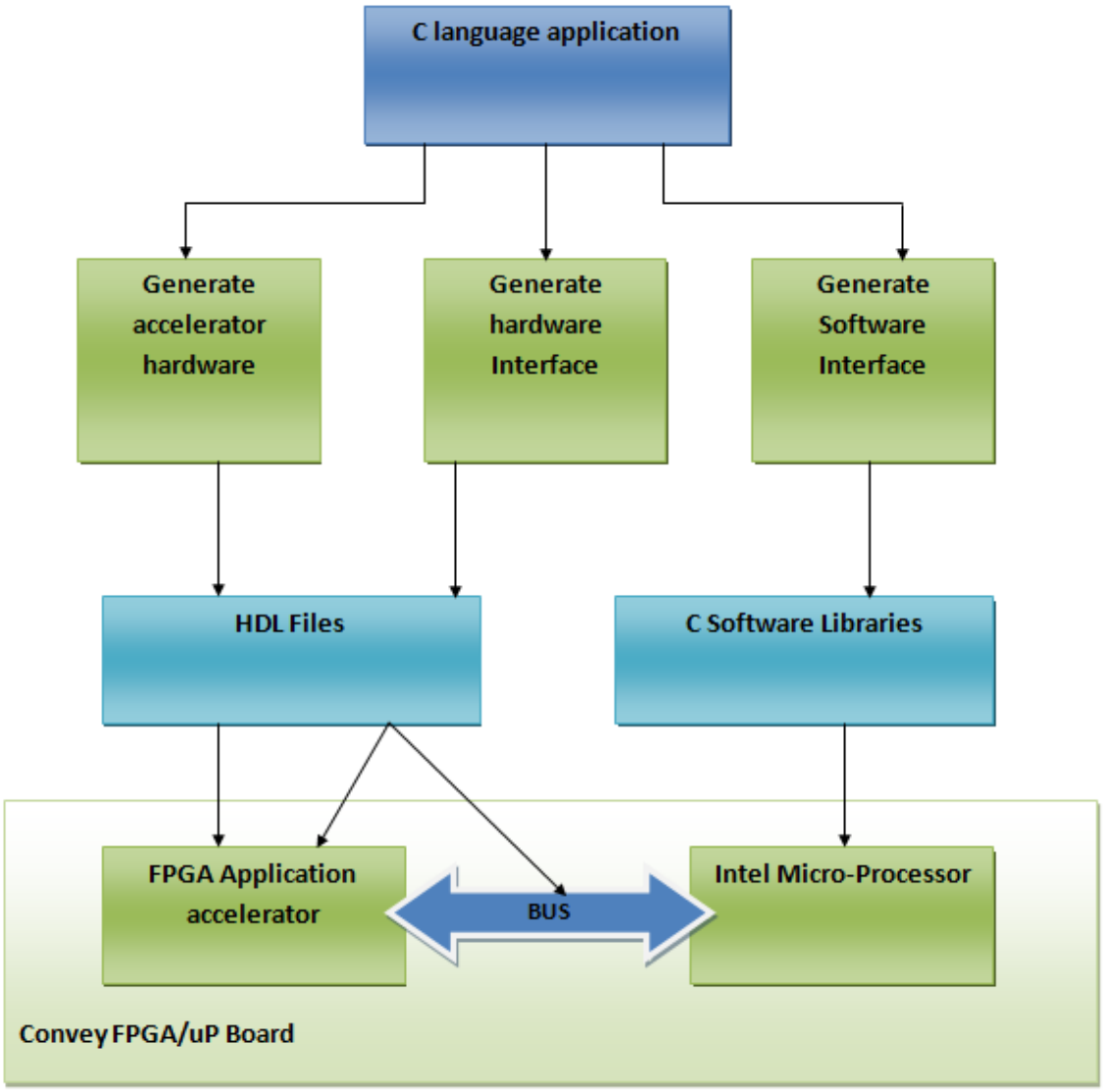
Με την χρήση της Impulse C μια εφαρμογή μπορεί να χωριστεί και να σχεδιαστεί μια υλοποίηση που είναι χωρισμένη σε software και hardware μέρη ή να υλοποιηθεί ολόκληρη πάνω σε μια FPGA.

Στο software μέρος της εφαρμογής, οι βιβλιοθήκες της Impulse C χρησιμοποιούνται για να ανοίξουν και να κλείσουν τα streams, να γράψουν και να διαβάσουν τα streams και να στείλουν μηνύματα ή signals ή να εμφανίσουν αποτελέσματα.

Στο Hardware μέρος της εφαρμογής, οι συναρτήσεις της Impulse C καθώς και της C μεταγλωττίζονται για να παραχθεί ισοδύναμη παράλληλη υλοποίηση σε μορφή HDL αρχείου ( VHDL, Verilog.. ). Αυτά τα αρχεία επεξεργάζονται από τα εργαλεία της FPGA ( Xilinx ) για να πάρουμε το hardware bitmap.




Εικ. 4-1 Γενική μορφή εφαρμογών Impulse C




Εικ. 4-1 Impulse C – Convey Design flow

## 4.3 Εφαρμογές

Η Impulse C χρησιμοποιείται σε εφαρμογές επεξεργασίας εικόνας, επεξεργασίας σήματος, σε ενσωματωμένα συστήματα καθώς και για την επιτάχυνση high-performance εφαρμογών.

## 4.4 Convey Computer

Το 2009 η εταιρεία Convey Computer έχει παρουσιάσει το HC-1, ένα υβρίδιο υπολογιστή με πλακέτες αναδιατασσόμενης λογικής.[31][16][17][18][19][20][19]

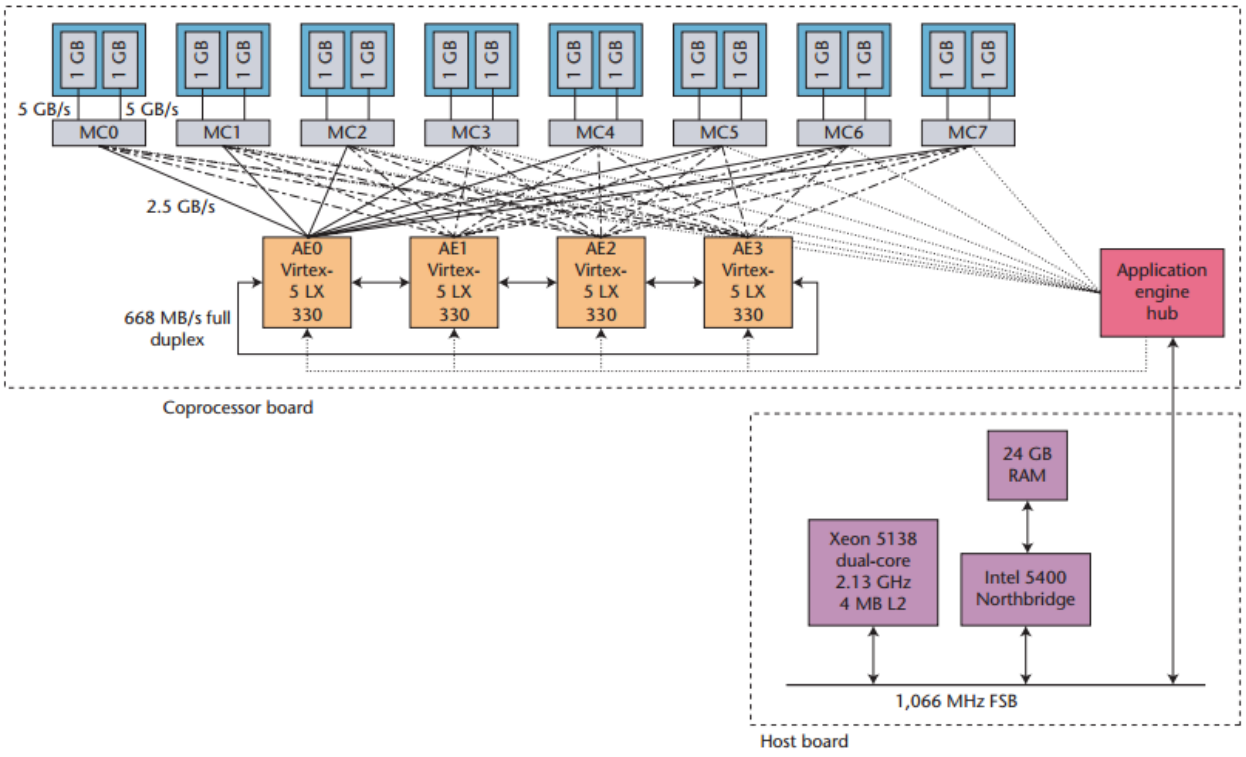
## 4.5 Χαρακτηριστικά Συστήματος

Το convey HC-1 είναι ένα υβριδικό σύστημα υπολογιστή με μια κοινή motherboard με δύο επεξεργαστές μαζί με ένα συνεπεξεργαστή βασισμένο σε FPGAs.

- dual-socket Intel Server motherboard
- Intel 5400 memory controller hub chipset
- 24 GBytes of RAM
- 1066 MHz FSB
- 2.13 GHz Xeon 5138 ,a dual-core , low-voltage processor

## 4.6 Top Level Design

Στην εικόνα φαίνεται το σχεδιάγραμμα του συνεπεξεργαστή. Υπάρχουν τέσσερεις προγραμματιζόμενες Virtex-5 LX 330s τις οποίες το Convey αποκαλεί application engines (AEs). Αναφέρεται σε μια συγκεκριμένη διαμόρφωση αυτόν των FPGAs με την ονομασία Personality. Κάθε μια από τις τέσσερεις AEs είναι ενωμένη με οκτώ memory controllers μέσω full crossbar. Οι FPGAs είναι ενωμένες και μεταξύ τους σε μια τοπολογία δακτυλίου.




**4-2 Το HC-1 coprocessor board. Τέσσερεις application engines ενώμένες με 8 memory controllers μέσω full crossbar. Κάθε memory controller είναι υλοποιημένος στην δική του FPGA πηγή www.convey.com**

## 4.7 Personalities

Το convey ανέπτυξε και διαθέτει τα δικά του σετ από personalities αλλά επιτρέπει στον χρήστη να αναπτύξει τις δικές του χρησιμοποιώντας τον Personality Development Kit (PDK). Οι personalities που διαθέτει το Convey έχουν σχεδιαστεί για συγκεκριμένες εφαρμογές. Μερικές από αυτές είναι οι:

- single-precision vector personality
- double-precision vector personality,
- financial analytics personality,
- Smith-Waterman personality.

Όλες μπορούν να χρησιμοποιηθούν σε πολλές εφαρμογές.

Για την σχεδίαση μιας δικής μας personality απαιτείται η χρήση του PDK που διαθέτει

- Ένα σετ από Makefiles που υποστηρίζουν simulation και synthesis design flow
- Ένα σετ από αρχεία Verilog για την επικοινωνία του επεξεργαστή με τις FPGAs
- Ένα σετ από μοντέλα προσομοίωσης για όλα τα μη προγραμματιζόμενα κομμάτια του συνεπεξεργαστή ( memory controllers , memory modules)
- Ένα interface γλώσσας προγραμματισμού για να μπορεί ο κώδικας να ελεγχθεί σε ένα εργαλείο Simulation όπως το Modelsim.

## 4.8 Convey Compilers

Το Convey διαθέτει compilers για C και FORTRAN που μπορούν να τρέξουν στον επεξεργαστή μαζί με τα personalities. Για την κλήση του κομματιού κώδικα που θα εκτελεστεί στις FPGAs γίνεται χρήση εντολών για την επικοινωνία με τον συνεπεξεργαστή.

# Κεφάλαιο 5. Υλοποίηση Συνάρτησης Μέγιστης Πιθανοφάνειας

Όπως έχουμε δει σε προηγούμενο κεφάλαιο, η συνάρτηση που καταναλώνει το μεγαλύτερο μέρος χρόνου είναι η **newviewGTRGAMMA().** Αρχικά έχουν γίνει κάποια Simulation σε Matlab καθώς και ένα simulation της ολοκληρωμένης σχεδίασης με τη βοήθεια των εργαλείων του CoDeveloper impulse C.

## 5.1 Αποτελέσματα Simulation

Για να μπορέσουμε να προχωρήσουμε με την σχεδίαση του Hardware έπρεπε να δούμε την ροή τον δεδομένων. Αρχικά έγινε μια προσομοίωση σε Matlab για μια κλήση της συνάρτησης. Για να γίνει αυτό, τα δεδομένα πριν την κλήση της συνάρτησης είχαν αποθηκευτεί σε αρχεία, και ακολούθως φορτώθηκαν στην Matlab, ούτως ώστε να έχουμε simulation με πραγματικά δεδομένα.

Αφού τα αποτελέσματα επιβεβαιώθηκαν, ξεκίνησε η σχεδίαση με το εργαλείο CoDeveloper Impulse C.

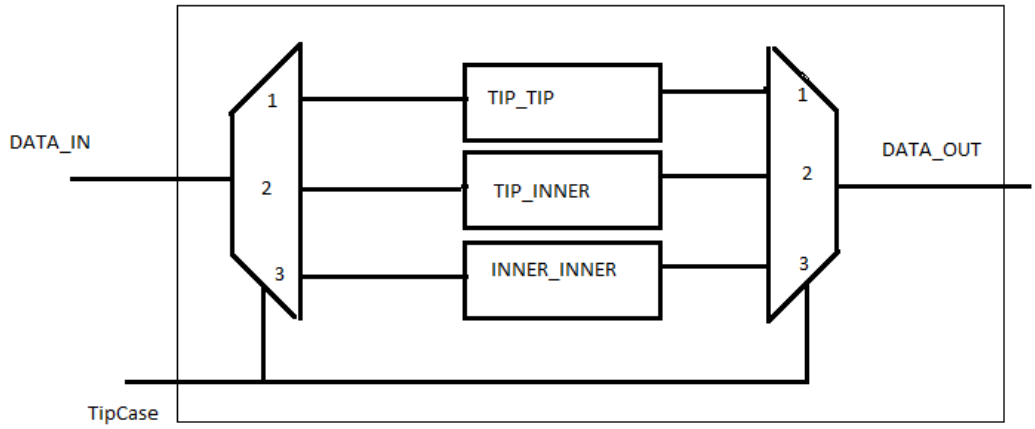
## 5.2 Υλοποίηση με την χρήση του Impulse C

Σε hardware έχει υλοποιηθεί μόνο μια συνάρτηση, η δήλωση της οποίας φαίνεται πιο κάτω.

```
static void newviewGTRGAMMA(int tipCase,
                            double *x1_start, double *x2_start, double *x3_start,
                            double *EV, double *tipVector,
                            int *ex3, unsigned char *tipX1, unsigned char *tipX2,
                            const int n, double *left, double *right, int *wgt, int
*scalerIncrement, const boolean useFastScaling
                            )
```

Κατά την κλήση της η συνάρτηση δέχεται πολλά ορίσματα υπό μορφή pointer. Το hardware είναι χωρισμένο σε 3 cases και σε κάθε κλήση εκτελείται μόνο το ένα, ανάλογα με την τιμή που έχει το *tipCase*. Για τον λόγο ότι όλα τα δεδομένα που χρησιμοποιούνται στο hardware διαβάζονται από την μνήμη, πριν από την κλήση της συνάρτησης επιλέγεται πιο μέρος του hardware θα εκτελεστεί και στέλνονται μόνο τα δεδομένα που είναι απαραίτητα.

Η γενική μορφή της αρχιτεκτονικής είναι η ακόλουθη.




Εικ. 5-1 Αφαιρετική μορφή Αρχιτεκτονικής

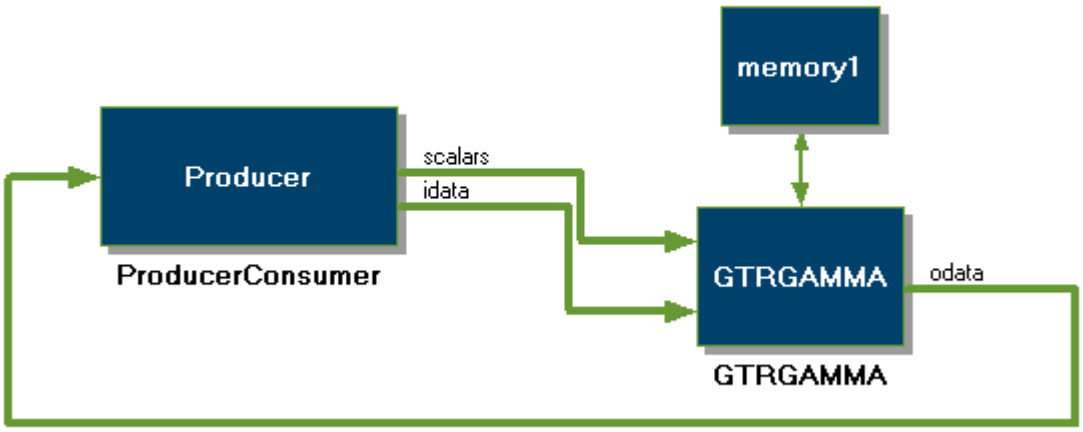
Όπως έχει αναφερθεί στο κεφάλαιο 3 η συνάρτηση αποτελείτε από 3 cases, ανάλογα με το είδος των κόμβων που συγκρίνονται κάθε φορά. Σε κάθε κλήση εκτελείται μόνο μια περίπτωση. Τα δεδομένα ανάλογα με το case που εκτελείται είναι διαφορετικά. Για κάθε case στέλνουμε μόνο τα απαραίτητα δεδομένα, τα οποία μεταφέρονται μέσω της κοινόχρηστης μνήμης. Τα αποτελέσματα της εκτέλεσης αποθηκεύονται στην μνήμη αντικαθιστώντας τα δεδομένα εισόδου.

Στην συνέχεια θα παρουσιαστεί αναλυτικά η ανάλυση της συνάρτησης υπολογισμού της μέγιστης πιθανοφάνειας και θα παρουσιαστεί βήμα – βήμα η συνολική υλοποίηση και ροή δεδομένων.

## 5.3 Ανάλυση των επιμέρους στοιχείων της Υλοποίησης

Αρχικά έχει ξαναγραφτεί το κρίσιμο κομμάτι κώδικα σε γλώσσα Impulse C. Η σχεδίασή μας ακολουθεί το template του one-process testbench σε συνδυασμό με το one memory template. [A]

Η μορφή του template όπως έχει σχεδιαστεί για τις δικές μας απαιτήσεις καταλήγει να έχει την ακόλουθη μορφή.




Εικ. 5-2 Template με το SW κομμάτι (ProducerConsumer), το HW κομμάτι (GTRGAMMA) και την μνήμη.

Στην υλοποίησή μας, χρησιμοποιούμε δύο διαδικασίες/συναρτήσεις. Η συνάρτηση ProducerConsumer, η οποία αποτελεί το SW κομμάτι της υλοποίησης, και η συνάρτηση GTRGAMMA η οποία είναι σχεδιασμένη με τέτοιο τρόπο ώστε να μεταφραστεί αργότερα σε HW. Η υλοποίησή της γίνετε για να αντικαταστήσει την συνάρτηση που καταναλώνει το μεγαλύτερο ποσοστό του χρόνου όπως έχει υπολογιστεί σε προηγούμενο κεφάλαιο. Τα δεδομένα όπως φαίνεται στα προηγούμενα διαγράμματα μεταφέρονται από το SW στο HW και αντίθετα μέσω της μνήμης. Το μπλοκ GTRGAMMA υλοποιεί τις τρεις περιπτώσεις που παρουσιάστηκαν πιο πριν.

Οι διαδικασίες αποτελούν το πιο βασικό κομμάτι της σχεδίασης μιας Impulse C εφαρμογής.

Στην συνάρτηση του software (*SW-Process – Producer/consumer*) αρχικά πρέπει να δηλωθούν τα streams και οι μνήμες που θα χρησιμοποιηθούν.

Τα streams και οι μνήμες αποτελούν τα πιο βασικά στοιχεία με τα οποία πρέπει να εξοικειωθεί ο προγραμματιστής.

Για την δική μας υλοποίηση απαιτούνται τρία streams και μια μνήμη. Τα streams είναι κανάλια επικοινωνίας που ενώνουν τις διαδικασίες. Οι μνήμες είναι εναλλακτικό των streams και χρησιμοποιούνται γιατί έχουμε μεγάλες ποσότητες δεδομένων που θα περάσουν από το SW στο HW και αντίθετα.

Αρχικά πρέπει να «οριστεί» η αρχιτεκτονική. Δημιουργούνται τα streams και οι μνήμες, και δημιουργεί την διεπαφή μεταξύ του Software και του Hardware.

Πρέπει να δηλωθούν τα streams και οι μνήμες, καθώς και τα ονόματα των συναρτήσεων( hardware και software).

Η αρχικοποίηση της αρχιτεκτονικής του σχήματος 5-5 γίνεται με εντολές γραμμένες σε Impulse C.

**Δημιουργία μνήμης [A.4.4]:**

Δημιουργείται η μνήμη με την κλήση της συνάρτησης co_memory_create().

Σε ένα .h αρχείο[A.5] έχουμε τις απαραίτητες τιμές(πλάτος και βάθος) για την δήλωση της μνήμης και μπορούν να μεταβληθούν αν πρέπει να αλλάξει το μέγεθος της δεσμευμένης μνήμης.

**Δημιουργία scalar stream [A.4.2]:**

Με παρόμοιο τρόπο δημιουργείται και το stream που θα μεταφέρει τα δεδομένα στο hardware. Το συγκεκριμένο stream μεταφέρει μια ακεραία ποσότητα, η οποία καθορίζει το Case που θα εκτελεστεί κάθε φορά.

**Δημιουργία control streams [A.4.2]:**

Ομοίως δημιουργούνται τα control streams. Έχουν την ιδιότητα κάποιου signal και χρησιμοποιούντα για την ενημέρωση του HW ότι μπορεί να ξεκινήσει την εκτέλεση, και για την ενημέρωση του SW ότι η εκτέλεση έχει τελειώσει και τα

δεδομένα είναι έτοιμα για να διαβαστούν. Ο λόγος που είναι απαραίτητα τα δύο αυτά streams είναι για να συγχρονιστεί η εγγραφή και η ανάγνωση της μνήμης.

**Δημιουργία host process [A.4.1]:**

Η host process είναι η software διαδικασία, στην οποία έχουμε το κυρίως πρόγραμμα. Δημιουργείται με την χρήση της συνάρτησης co_process_create() και δέχεται σαν ορίσματα την δήλωση της διαδικασίας καθώς και όλα τα streams και μνήμες που απαιτεί η σχεδίαση.

**Δημιουργία HW process [A.4.1]:**

Η διαδικασία GTRGAMMA_process είναι η HW διαδικασία[A.4.1], που στην συνέχεια θα μεταφραστεί σε VHDL. Δηλώνεται με τον ίδιο τρόπο. Παρατηρούμε ότι τα streams και οι μνήμες που δέχονται σαν είσοδο, είναι οι ίδιες μεταξύ SW και HW.

Η χρήση των εντολών όπως έχουν παρουσιαστεί πιο πάνω είναι απαραίτητη για την δημιουργία της σχεδίασης. Μέχρι αυτό το σημείο έχουμε δηλώσει πιο μέρος του κώδικα θα εκτελεστεί σε SW και πιο σε HW. Ακολούθως θα ξεκινήσουμε την σχεδίαση.

**Υλοποίηση SW συνάρτησης:**

Στην συνέχεια ξεκινούμε την υλοποίηση του project ξεκινώντας από την υλοποίηση της SW συνάρτησης.

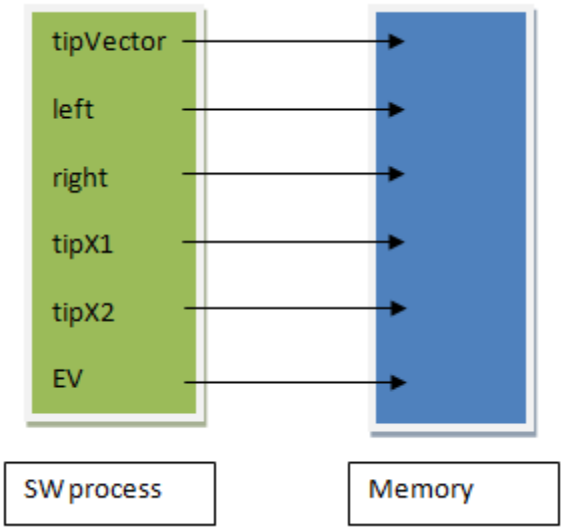
Το stream με την ονομασία scalars χρησιμοποιείται για να στείλει στο hardware process (*HW-Process–Filter*) την ακέραια τιμή που απαιτείται στο HW για την επιλογή του case που θα εκτελεστεί κάθε φορά.

Τα streams opassdata και ipassdata χρησιμοποιούνται σαν signals για να ξέρει τόσο η συνάρτηση HW όσο και η συνάρτηση SW ότι η μνήμη είναι έτοιμη για να διαβαστεί.

Επειδή όπως αναφέρθηκε και πιο πριν κάθε φορά εκτελείται μόνο ένα case, στη SW-process επιλέγουμε πια δεδομένα θα στείλουμε μέσω της μνήμης κάθε φορά. Η συγκεκριμένη συνάρτηση προσομοιώνει μόνο μια κλήση της συνάρτησης ενδιαφέροντος, και γίνετε για να μπορέσουμε να επιβεβαιώσουμε τα αποτελέσματα. Ουσιαστικά έχουμε προφορτωμένα τα δεδομένα που θα

μεταφερθούν από το SW στο HW. Τα δεδομένα είναι πραγματικά και τα έχουμε πάρει από την SW εκτέλεση της εφαρμογής, ακριβώς πριν από την κλήση της συνάρτησης ενδιαφέροντος και τα γράφουμε στην μνήμη.

Πρώτα, αρχικοποιούμε την μνήμη αναθέτοντας της ένα pointer, και στην συνέχεια την γεμίζουμε με τα δεδομένα που θέλουμε. Τα δεδομένα που πρέπει να εισάγουμε στο HW είναι πίνακες double.




Ανάλογα και με τα υπόλοιπα cases γίνεται το ίδιο. Αφού τελειώσουμε με την μνήμη ενημερώνουμε το HW ότι μπορεί να ξεκινήσει την εκτέλεσή του. Η μνήμες που δηλώνουμε στην σχεδίαση μεταφράζονται σαν Block-RAM πάνω στην FPGA.

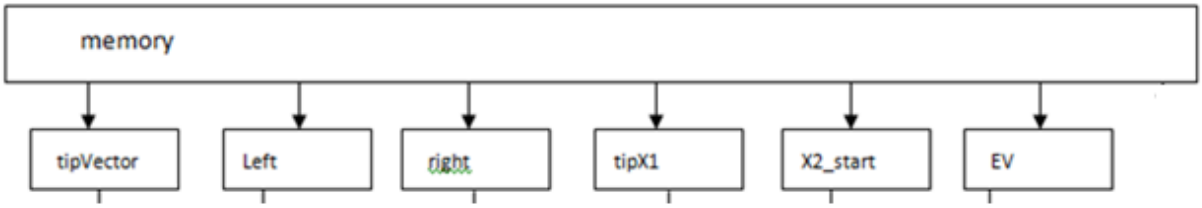
**Υλοποίηση HW process:**

Στην συνάρτηση HW πρέπει να δηλωθούν τα streams και μνήμες που επικοινωνούν με την συνάρτηση SW.

Αρχικά δεχόμαστε μέσω stream το δεδομένο tipCase για την επιλογή του case που θα εκτελεστεί.

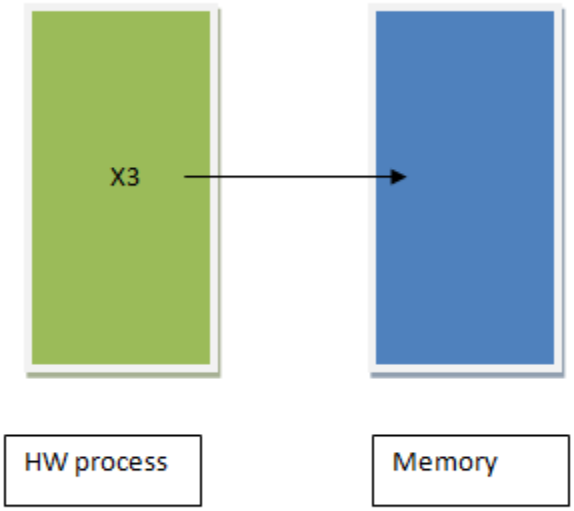
Ανάλογα με την τιμή του tipCase εκτελείται διαφορετικό κομμάτι κώδικα, διαβάζοντας την μνήμη. Σε κάθε περίπτωση περιμένουμε το signal από το SW και ακολούθως εκτελείται ο κώδικας.

Η συνάρτηση που θα εκτελεστεί σε HW έχει πρόσβαση στα δεδομένα της μνήμης.




Αφού διαβάσουμε όλα τα δεδομένα από την μνήμη, προχωρούμε στην εκτέλεση του κώδικα και στον υπολογισμό της συνάρτησης πιθανοφάνειας*, της οποίας ο υπολογισμός θα παρουσιαστεί αναλυτικά στην επόμενη ενότητα.*

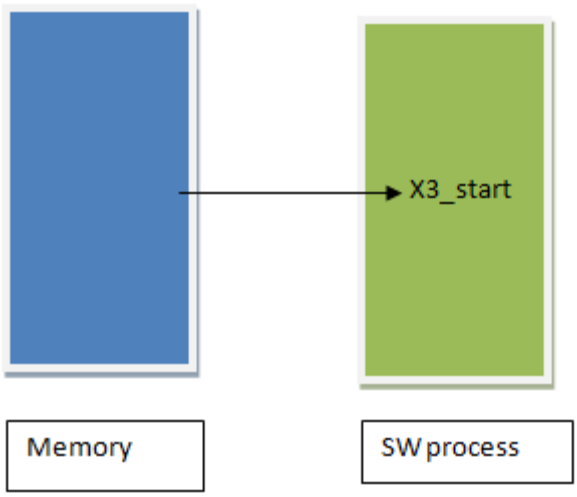
Αφού τα αποτελέσματα υπολογιστούν πρέπει να τα επιστρέψουμε στην SW-process , και να ενημερώσουμε ότι η εγγραφή στην μνήμη έχει τελειώσει.




**Επιστροφή αποτελεσμάτων στο HW**

Η διαδικασία SW δέχεται ενημέρωση ότι τα αποτελέσματα είναι έτοιμα, για να συνεχιστεί η εκτέλεση από το σημείο που έχει σταματήσει πριν την κλήση του συνεπεξεργαστή.

Ανάλογα με το  case εκτέλεσης αποθηκεύει τα δεδομένα στις μεταβλητές που πρέπει για να συνεχίσει η εκτέλεση του προγράμματος. Τα αποτελέσματα που επιστρέφονται, διαβάζονται σειριακά από την μνήμη και αντικαθιστούν τις τιμές του δείκτη.



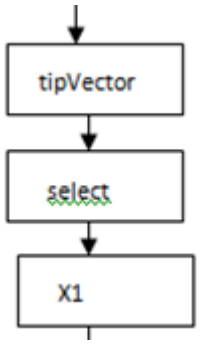

Η υλοποίηση της αρχιτεκτονικής έγινε σε πολλά βήματα μέχρι να καταλήξουμε στην κατάλληλη σχεδίαση. Σε κάθε βήμα γινόταν αποτίμηση της απόδοσης μέχρι να έχουμε την καλύτερη απόδοση σε σχέση με το μέγεθος της σχεδίασης αλλά και την εγκυρότητα των αποτελεσμάτων.

Στην επόμενη ενότητα παρουσιάζεται η πρώτη υλοποίηση και η παρουσίαση της ροής των δεδομένων μέσα στο HW.

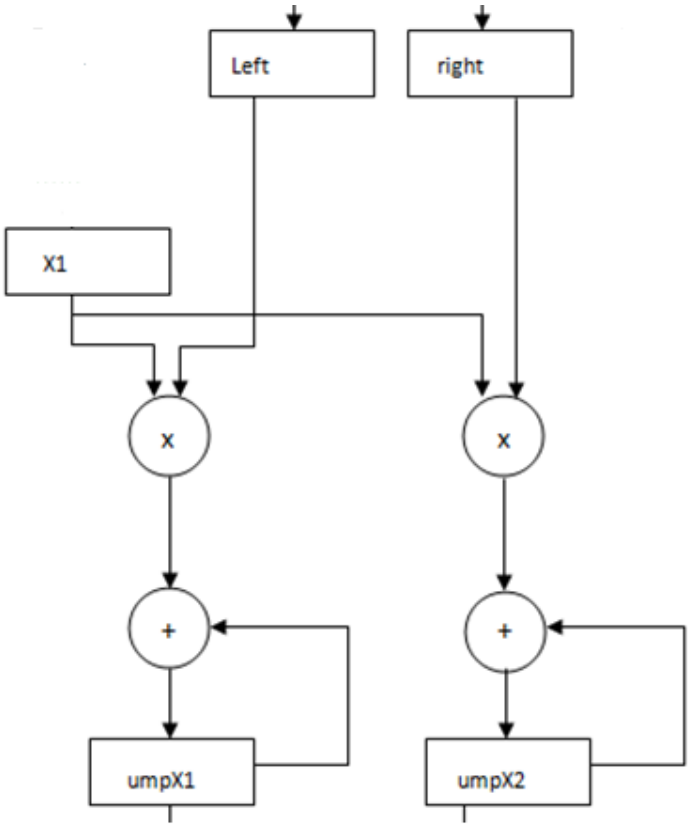
## 5.4 Αρχικός Σχεδιασμός

Αρχικά η σχεδίαση έγινε χωρίς βελτιστοποιήσεις και μπορούσε να υποστηρίξει μόνο ένα αρχείο εισόδου. Στο HW κομμάτι της σχεδίασης, όπως αναφέρθηκε και πιο πριν, τα δεδομένα αφού διαβαστούν από την μνήμη αποθηκεύονται σε προσωρινούς πίνακες όπως φαίνετε στο παράδειγμα πάνω. Στην συνέχεια οι υπολογισμοί γίνονται με τα δεδομένα που βρίσκονται σε αυτούς τους πίνακες.

Το μαθηματικό μοντέλο του υπολογισμού της μέγιστης πιθανοφάνειας χρησιμοποιεί μέρος των δεδομένων σε κάθε επανάληψη. Λόγω της αδυναμίας του εργαλείου να χειρίζεται δείκτες, γίνεται χρήση πινάκων.

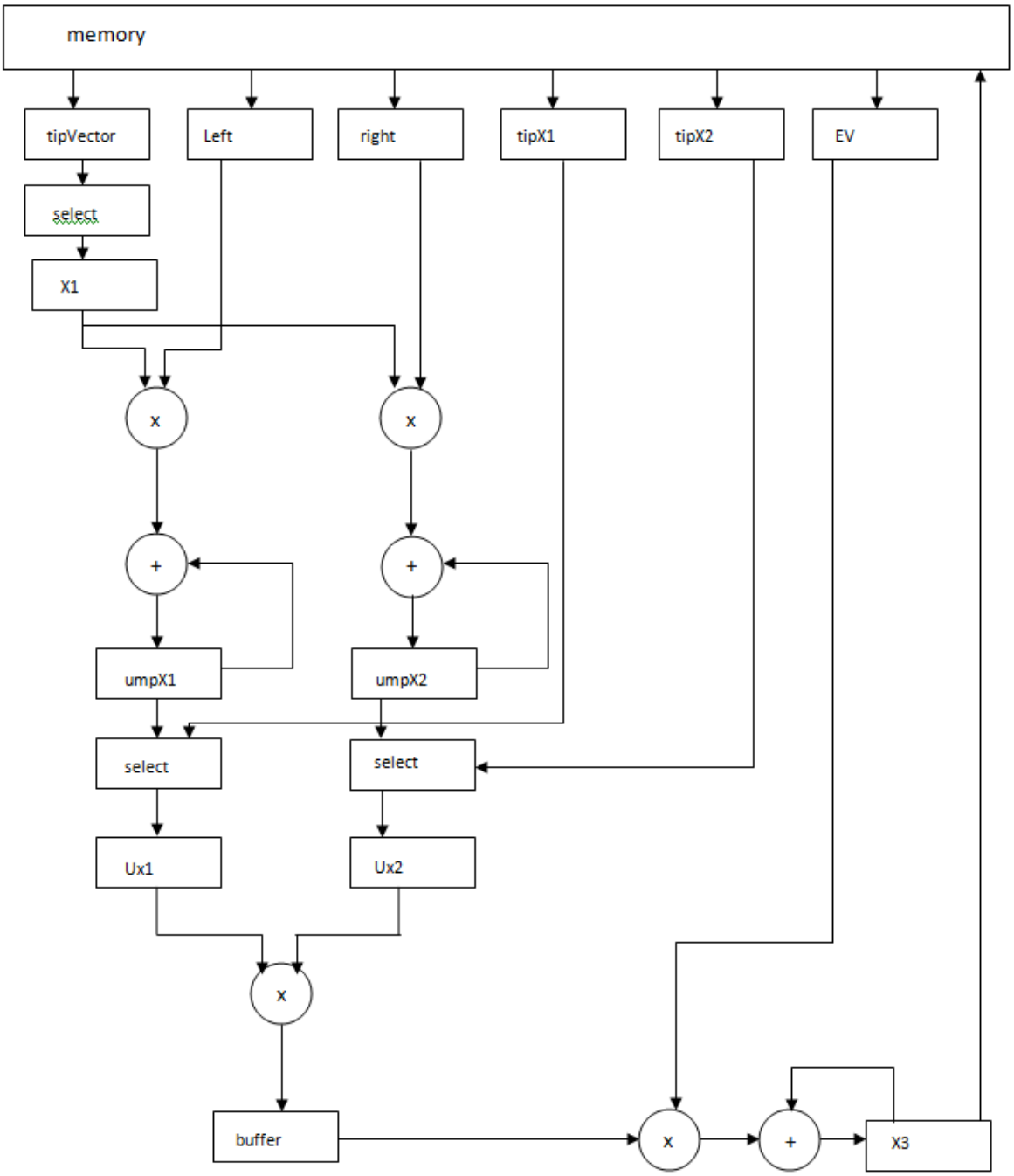


Υλοποιείται το μαθηματικό μοντέλο του RAxML για τον υπολογισμό των βαρών στα φύλλα του φυλογενετικού δέντρου για να μπορέσει να γίνει στο τέλος αναδιάταξη των κόμβων για να έχουμε βελτιωμένο δέντρο.



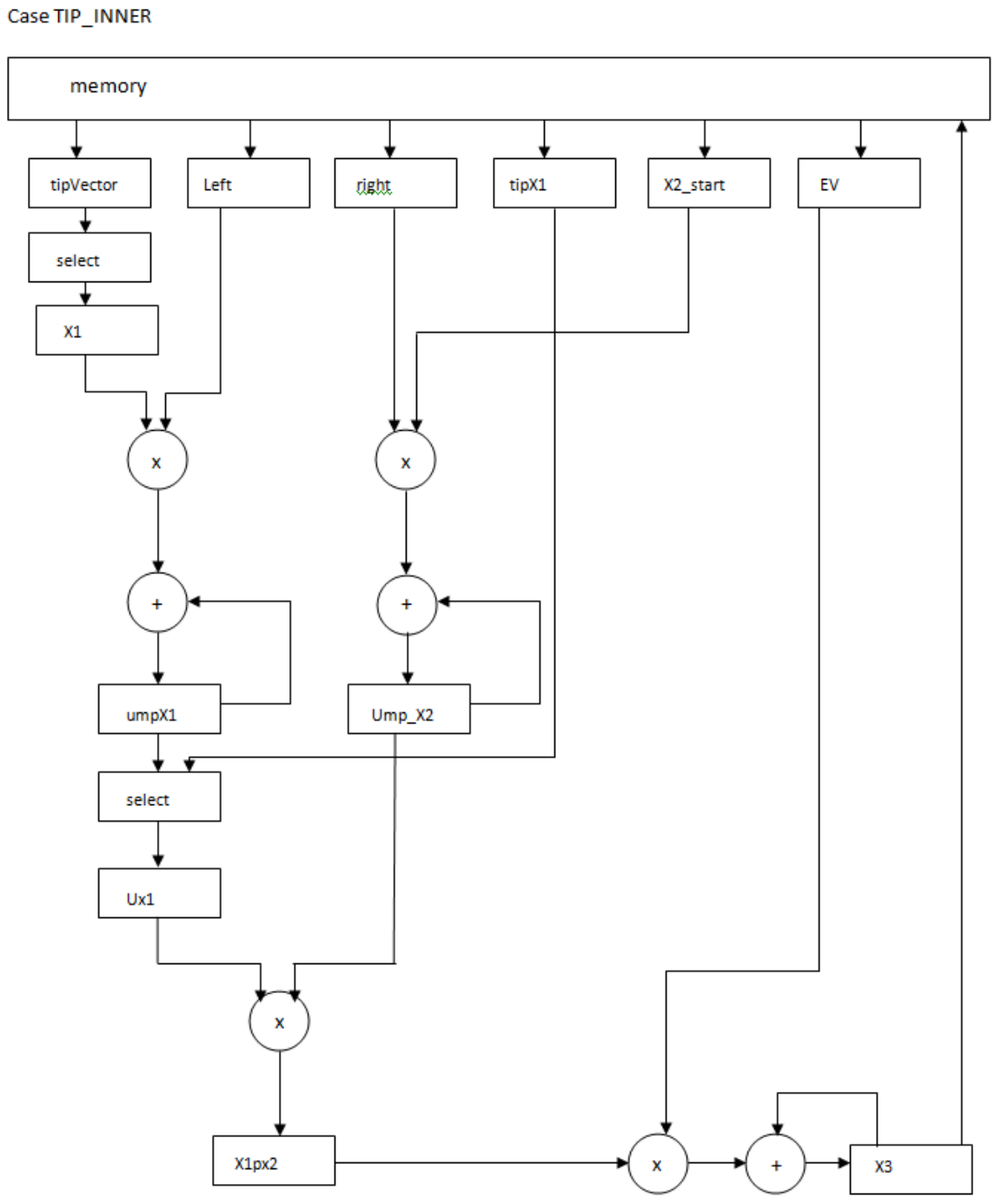

Με παρόμοιο τρόπο υλοποιείται το υπόλοιπο κομμάτι κώδικα. Τελικά, η ολοκληρωμένη μορφή της ροής των δεδομένων που προκύπτει από τα προηγούμενα καταλήγει να έχει την ακόλουθη μορφή.

Case TIP_TIP



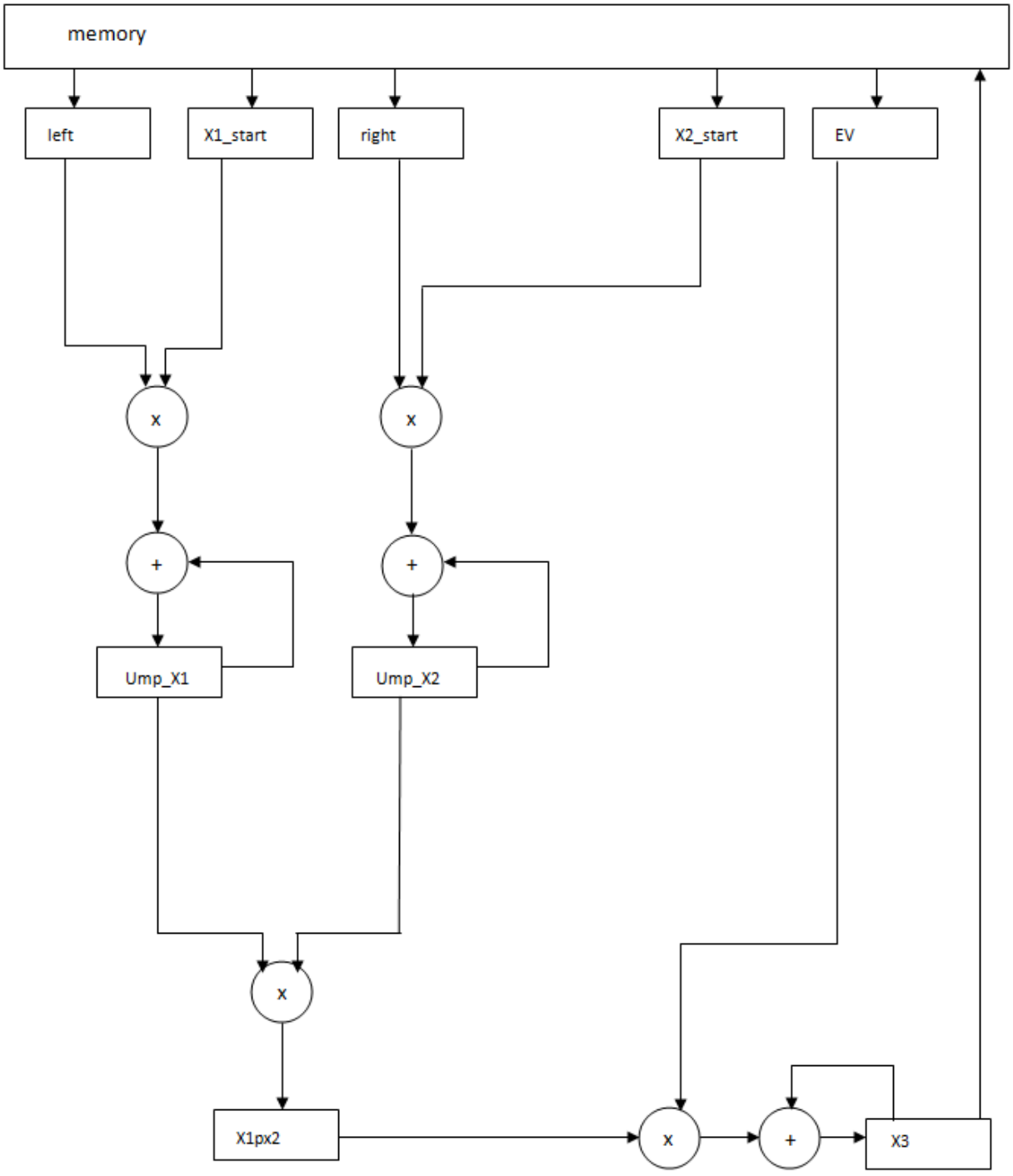

Εικ. 5-3 Case TIP_TIP

Όπως αναφέρθηκε πιο πριν, η συνάρτηση υπολογισμού της μέγιστης πιθανοφάνειας αποτελείται από τρεις περιπτώσεις που εκτελούνται ανάλογα με την τιμή του tipCase. Ακλουθούν τα σχήματα με τη ροή δεδομένων για τις άλλες δύο περιπτώσεις.




Εικ.5-4 Case TIP_INNER

Case INNER_INNER



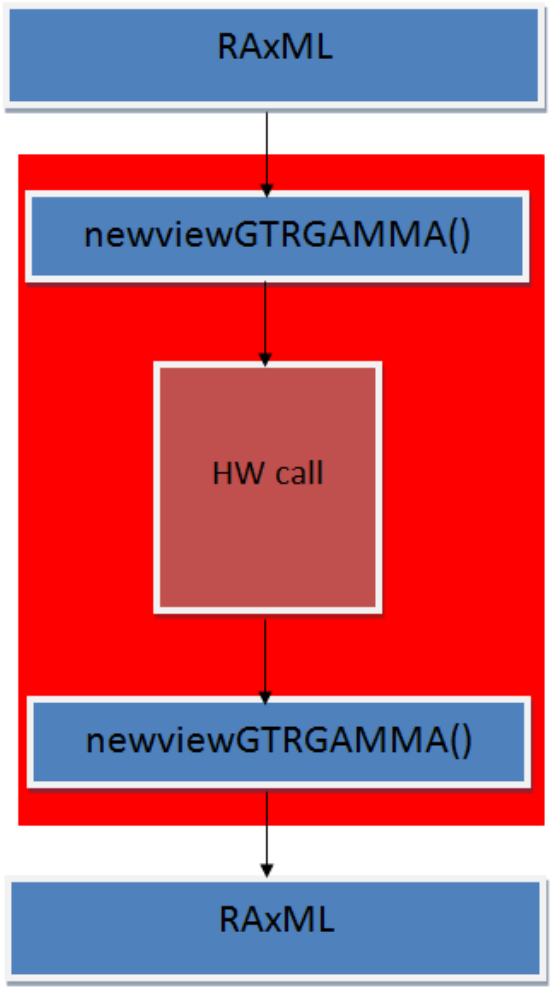

Εικ. 5-5 Case INNER_INNER

Με την ολοκλήρωση της υλοποίησης έγινε ταυτοποίηση των αποτελεσμάτων του Simulation και προχωρήσαμε στην παραγωγή του κώδικα. Έγινε μια πρόχειρη εκτίμηση των πόρων που απαιτούνται, και στην συνέχεια προχωρήσαμε στην παραγωγή του HW. [A.7]

### 5.4.1 Χρήση Convey και Σύνδεση Σχεδίασης με το Πρόγραμμα RAxML

Αφού επιβεβαιώσαμε τα αποτελέσματα του simulation με τα αποτελέσματα της εκτέλεσης του κώδικα, κατεβάσαμε την σχεδίαση μας στο Convey με την βοήθεια του μεταπτυχιακού φοιτητή Χρίστου Ρουσόπουλου. Το εργαλείο Impulse C έχει την δυνατότητα να σχεδιάσει την διασύνδεση μεταξύ του SW και της σχεδίασής μας. Το κομμάτι που θα εκτελεστεί σε SW το έχω σχεδιάσει με τις απαραίτητες παραμετροποιήσεις για να μπορεί να στέλνει και να δέχεται δεδομένα σωστά, και να μπορεί να συνεχιστεί η εκτέλεση του υπόλοιπου κώδικα χωρίς προβλήματα. Αρχικά έγινε η εκτέλεση για μια κλήση της συνάρτησης όπως και στο Simulation και ταυτοποίηση των αποτελεσμάτων του simulation με τα αποτελέσματα της εκτέλεσης στο πραγματικό σύστημα.

Στην συνέχεια έγινε προσπάθεια να συνδέσουμε την σχεδίαση με ολόκληρο το πρόγραμμα RAxML ούτως ώστε να μπορεί να εκτελέσει όλους τους αλγορίθμους που μπορεί να υποστηρίξει και τρέξαμε τον SW κώδικα μαζί με το HW κομμάτι με πραγματικά δεδομένα. Η ολοκληρωμένη σχεδίαση έχει την ακόλουθη μορφή:




Εικ. 6-5 Συνολική σχεδίαση SW και HW

Αρχικά ξεκινά να εκτελείτε ο επιλεγμένος αλγόριθμος του RAxML και όταν πρέπει να κληθεί η συνάρτηση ενδιαφέροντος, εκτελείται η σχεδίαση που έχουμε φτιάξει. Μετά την περάτωση του υπολογισμού, τα αποτελέσματα επιστρέφονται στο SW, αποθηκεύονται στις θέσεις μνήμης που πρέπει και το RAxML συνεχίζει την εκτέλεσή του.

Για να μπορέσουμε να ενώσουμε την σχεδίαση όπως έχει παραχθεί με το εργαλείο CoDeveloper Impulse C, έπρεπε να φορτώσουμε τις επιπλέων βιβλιοθήκες που χρησιμοποιεί το Impulse C πρόγραμμα που αφορούν κυρίως την δήλωση και τις συναρτήσεις εκτέλεσης των τύπων δεδομένων που χρησιμοποιεί. Προσθέσαμε στον κώδικα τα αρχεία C που παράγει αυτόματα το εργαλείο κατά την μετάφραση, που αρχικοποιούν την αρχιτεκτονική (αρχικοποίηση streams και memories), για να γίνουν compile μαζί με το υπόλοιπο πρόγραμμα. Για να μπορέσουμε να το κάνουμε αυτό έπρεπε να αλλάξουμε το Makefile του RAxML

για να μπορεί να κάνει link και τα επιπλέων αρχεία που προσθέσαμε και να αλλάξουμε και τον compiler γιατί ο gcc compiler που χρησιμοποιούσε το RAxML δεν ήταν συμβατό με τον cnycc compiler του Impulse C.

Με σχετική υπόδειξη του κ. Scott Thibault[35], προσθέσαμε στην main συνάρτηση του RAxML τις κλήσης που γίνονταν στην main συνάρτηση της δικής μας υλοποίησης για να δημιουργείται αμέσως η διασύνδεση με το HW.

Αρχικοποιείτε η σχεδίαση και δημιουργούνται τα streams και τις μνήμες. Κάθε stream και μνήμη χαρακτηρίζεται από ένα pointer. Για να μπορέσουμε να τα χρησιμοποιήσουμε σε ολόκληρο τον κώδικα, δηλώσαμε νέα streams και μνήμες σαν global.

Στην συνέχεια τους δώσαμε την ίδια διεύθυνση με τους pointers που παράγει το εργαλείο.

Με αυτή την διαδικασία, έχουμε διαθέσιμα τα streams και την μνήμη σε κάθε κομμάτι του SW κώδικα για να μπορεί να επικοινωνεί με την HW συνάρτηση.

Πλέων ακολουθείτε η ίδια διαδικασία εγγραφής και ανάγνωσης της μνήμης που παρουσιάστηκε στην αρχή του κεφαλαίου, αυτή τη φορά από τα δεδομένα των μεταβλητών του προγράμματος. Σε κάθε κλήση της συνάρτησης έχουμε διαφορετικές τιμές και εγγραφές στην μνήμη. Οι αλλαγές που έγιναν στο RAxML για να μπορεί να αντικαταστήσει την συνάρτηση ενδιαφέροντος με την σχεδίασή μας, δεν χρειάζονται κάποια παραμετροποίηση αν αλλάξει η σχεδίαση μελλοντικά.

Το αποτέλεσμα της εκτέλεσης ήταν το ίδιο με το αποτέλεσμα των πειραμάτων στην εκτέλεση του SW κώδικα για το συγκεκριμένο αρχείο εισόδου.

### 5.4.2 Αποτίμηση Υλοποίησης

Η πρώτη υλοποίηση έγινε με τέτοιο τρόπο ώστε να μπορεί να δέχεται λίγα δεδομένα για να έχουμε σχετικά μικρή ροή δεδομένων, να έχουμε αποτελέσματα που να μπορούν να ταυτοποιηθούν εύκολα και να υπάρχει η ευχέρεια να παραμετροποιηθεί εύκολα. Έγινε για να μπορέσουμε να επιβεβαιώσουμε την συμβατότητα του παραγόμενου HDL κώδικα με το Convey και όχι για εκτίμηση απόδοσης.

Ο χρόνος εκτέλεσης δεν έχει υπολογιστεί, γιατί το σετ δεδομένων εισόδου δεν είναι αντιπροσωπευτικά των αποτελεσμάτων του profiling, είναι όμως πραγματικά δεδομένα από την βιβλιογραφία.

Στις επόμενες ενότητες παρουσιάζεται η βελτίωση της αρχικής σχεδίασης ώστε να δέχεται ρεαλιστικά δεδομένα μαζί με μετρήσεις απόδοσης.

# Κεφάλαιο 6. Δεύτερη Σχεδίαση

Στην δεύτερη φάση της υλοποίησης έγινε παραμετροποίηση του HW κώδικα, ώστε να μπορεί να εκτελεστεί έχοντας σαν είσοδο αρχεία όλων των μεγεθών. Έγινε σχεδίαση νέας αρχιτεκτονικής, ίδιας με την προηγούμενη, αλλά έγινε προσπάθεια να χρησιμοποιηθεί *όλη η διαθέσιμη μνήμη* της FPGA, ούτως ώστε να μπορεί να δεχτεί μεγάλα σε μέγεθος δεδομένα. Έγινε επίσης προσπάθεια χρήσης των ειδικών χαρακτηριστικών του εργαλείου[A.5.1] για βελτιστοποίηση της αρχιτεκτονικής.

## 6.1 Χρήση Pipelining και Unrolling

Για να επιταχύνουμε την σχεδίαση, το εργαλείο CoDeveloper Impulse C, δίνει την δυνατότητα να δηλώσουμε ποια κομμάτια κώδικα θέλουμε να σχεδιαστούν χρησιμοποιώντας Pipeline και ποιες επαναληπτικές δομές μπορούν να αναπτυχθούν.

Με την εισαγωγή του pragma CO PIPELINE ο compiler θα μεταφράσει το block κώδικα που βρίσκεται μεταξύ των αγκίστρων σαν ένα κομμάτι pipeline. Ένα μειονέκτημα του τρόπου που γίνεται το pipelining, είναι ότι σε εμφωλευμένα loops, μπορούμε να έχουμε pipeline μόνο το εσωτερικό loop.

Με παρόμοιο τρόπο γίνεται και η χρήση του Pragma UNROLL, που κατευθύνει τον compiler να αναπτύξει το loop (αν μπορεί) στις επιμέρους εντολές.

Προφανώς αν υπάρχουν εξαρτήσεις μεταξύ των εντολών όπως στο παράδειγμα που ακολουθεί, η χρήση του UNROLL είναι μεν επιτρεπτή, αλλά δεν θα υλοποιηθεί.

### 6.1.1 Ταυτοποίηση Πρώτης Βελτίωσης.

Αφού τα αποτελέσματα έχουν και πάλι επιβεβαιωθεί με την χρήση SW-HW co simulation, έγινε παραγωγή του HDL κώδικα και ξανακατεβάσαμε την υλοποίηση στο Convey. Τα αποτελέσματα της εκτέλεσης στο πραγματικό σύστημα ήταν λανθασμένα. Ένας από τους λόγους που γίνεται αυτό είναι η χρήση του pipeline και unroll.

Η ιδέα του UNROLLING έχει εγκαταλειφτεί, γιατί με αυτό τον τρόπο η σχεδίαση χρησιμοποιούσε περισσότερα DSPs από τα διαθέσιμα.

Η χρήση του Pipeline έχει επίσης περιοριστεί για τον λόγο ότι η σχεδίαση γινόταν αρκετά μεγαλύτερη, ο χρόνος εκτέλεσης των επιμέρους κομματιών κώδικα δεν

βελτιωνόταν(παραγωγή περισσότερων κύκλων, χρήση πιο μεγάλου ρολογιού), και σε διάφορα σημεία τα αποτελέσματα δεν ήταν σωστά, λόγω της εξάρτησης των επόμενων υπολογισμών από τους προηγούμενους.

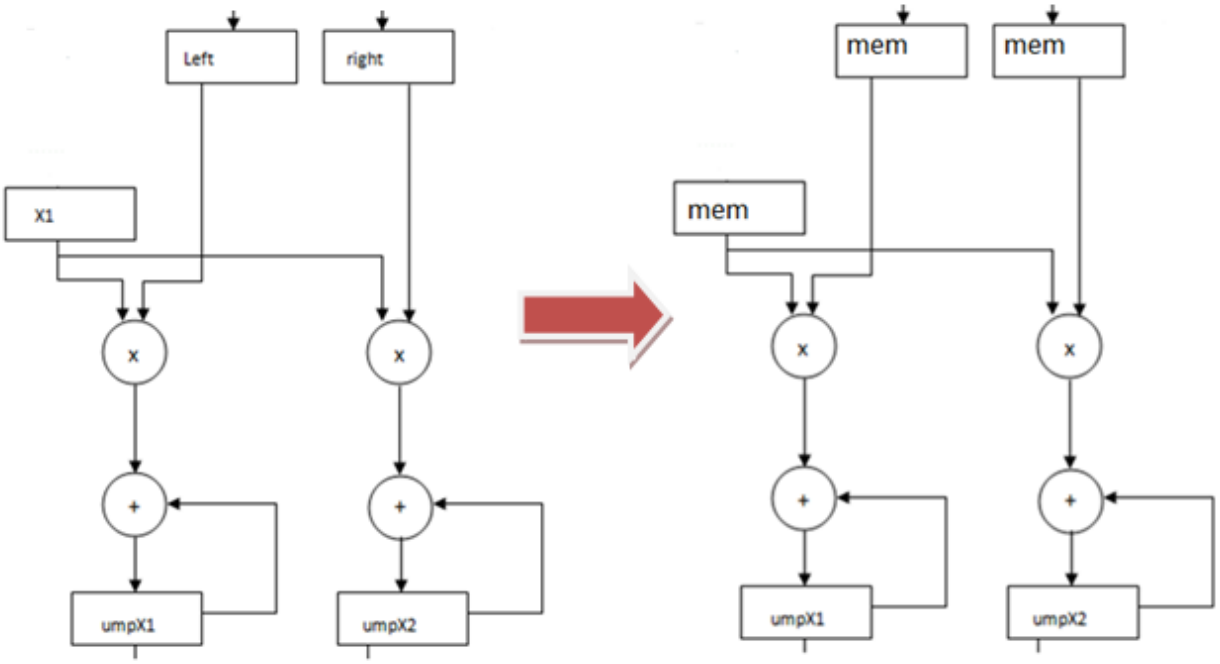
Με την βοήθεια του Χρήστου Ρουσόπουλου παρατηρήσαμε ότι ένας άλλος λόγος που η εκτέλεση γινόταν λάθος, ήταν το γεγονός ότι κάποιες μνήμες ήταν μακριά από την λογική της σχεδίασής μας και αυτό επηρέαζε το mapping της σχεδίασης.

Αυτό είχε ως αποτέλεσμα, να μην συγχρονίζονται οι υπολογισμοί και τα αποτελέσματα εξόδου να είναι λάθος.

## 6.2 Μείωση Απαιτούμενης Μνήμης

Η προηγούμενη προσέγγιση έχει ένα μεγάλο μειονέκτημα. Οι πίνακες που δηλώνονται στο Impulse C μεταφράζονται σαν Block-RAM. Όπως αναφέρθηκε και πιο πάνω, το ίδιο και οι μνήμες, άρα δεσμεύουμε την διπλάσια μνήμη από αυτή που είναι απαραίτητη. Αυτό έχει ως αρνητικό αποτέλεσμα να έχουμε μεγαλύτερη κατανάλωση πόρων, και περισσότερους κύκλους για την ολοκλήρωση της εκτέλεσης, αφού έχουμε μια εγγραφή και μια ανάγνωση για κάθε δεδομένο η οποία είναι αχρείαστη.

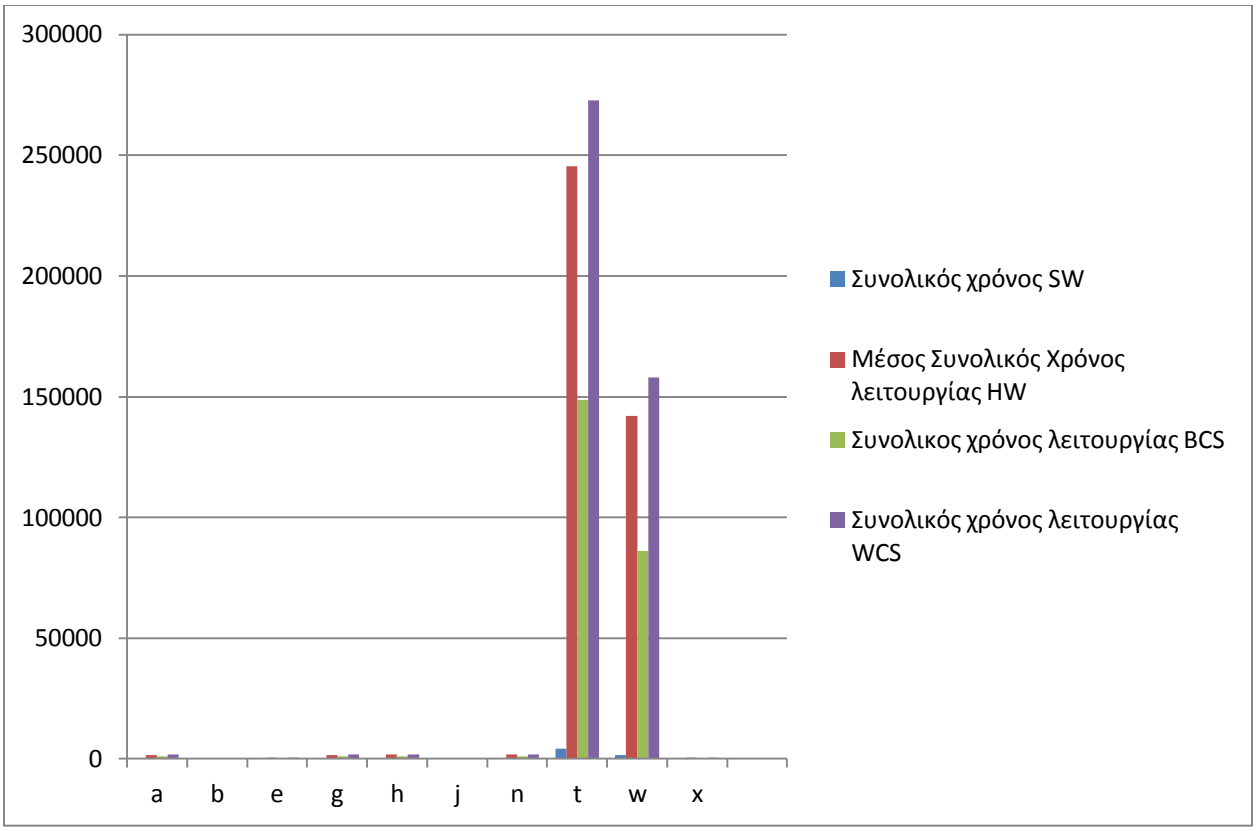
Για να λυθεί το πρόβλημα έγινε άλλη σχεδίαση, η οποία δεν χρησιμοποιούσε αχρείαστους πίνακες, και η ανάγνωση των δεδομένων γινόταν ακριβώς πριν να χρησιμοποιηθούν, και ο υπολογισμός γινόταν αμέσως.




Με αυτή την μέθοδο, η εκτέλεση γίνεται πιο γρήγορη αφού ο χρόνος για την ανάγνωση της μνήμης μειώνεται. Η μέθοδος αυτή χρησιμοποιήθηκε σε κάθε ανάγνωση από την μνήμη. Επίσης η δεσμευμένη μνήμη μειώθηκε, για να έχουμε την σχεδίαση μαζεμένη.

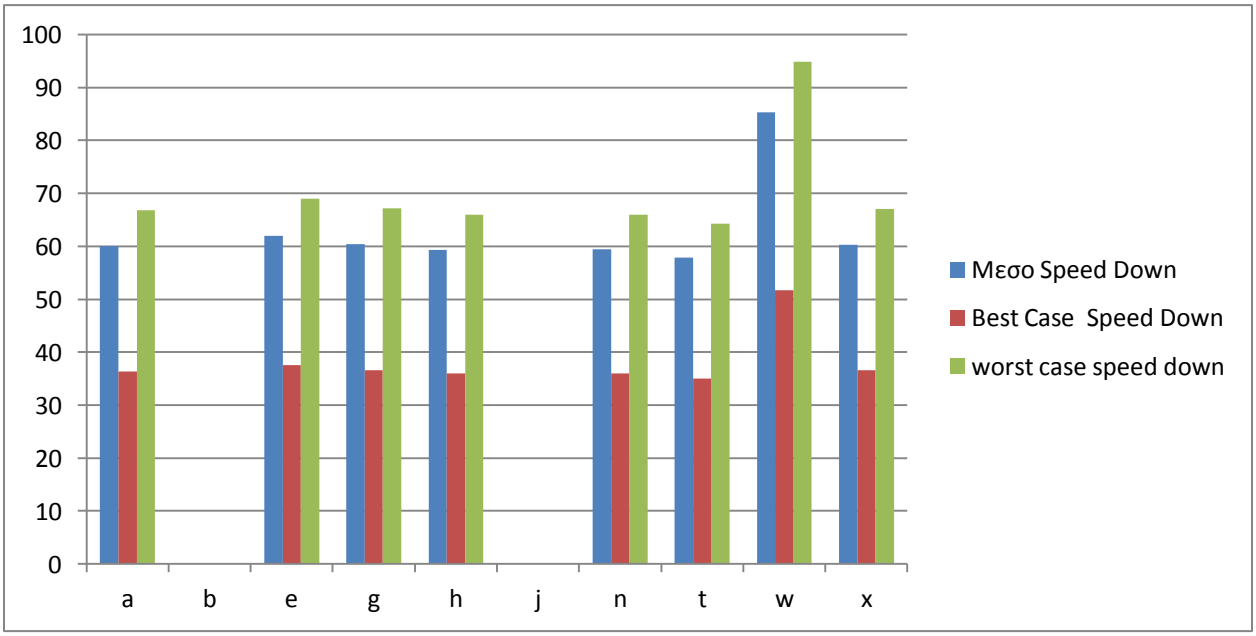
## 6.3 Απόδοση Αρχικής Σχεδίασης

Η απόδοση της σχεδίασης έχει υπολογιστεί για την διαφορά του συνολικού χρόνου εκτέλεσης της συνάρτησης ενδιαφέροντος σε HW και σε SW. Δεν μπορούμε να έχουμε συνολική εικόνα της διαφοράς για τον λόγο ότι το κομμάτι του SW έτρεξε σε διαφορετικά μηχανήματα κατά την εκτέλεση του profiling και της σχεδίασης με συνεπεξεργαστή. Ο υπολογισμός έγινε υπολογίζοντας τον χρόνο κλήσης της κάθε περίπτωσης (tipCase) και έγινε υπολογισμός του συνολικού χρόνου που καταναλώθηκε κατά την εκτέλεση του HW και του χρόνου που αναλώνεται για το I/O.




Διάγραμμα 14 Συνολικός χρόνος εκτέλεσης πρώτης υλοποίησης

Συγκρίνοντας τα αποτελέσματα με τον χρόνο εκτέλεσης του SW παρατηρούμε ότι η αρχιτεκτονική μας έχει χειρότερη απόδοση.




Διάγραμμα 15 Speed down πρώτης υλοποίησης

Όπως φαίνεται στον πίνακα έγινε υπολογισμός για την καλύτερη περίπτωση, για την χειρότερη περίπτωση και για την μέση περίπτωση. Πιο αντιπροσωπευτική είναι η μέση περίπτωση για τον λόγο ότι έχουμε ισοζυγισμένες κλήσεις των περιπτώσεων (tipCase). Η HW κλήση της συνάρτησης ενδιαφέροντος ήταν πιο αργή από την αντίστοιχη κλήση σε SW. Η μικρότερη επιβράδυνση που είχαμε είναι –57.81 x (0.017x), η μέση επιβράδυνση είναι -63.07 x (0.015x) και η μεγαλύτερη επιβράδυνση είναι -85.30 x (0.011x).

Στις επόμενες ενότητες θα γίνει προσπάθεια να βελτιωθεί η απόδοση της σχεδίασης με την χρήση των ειδικών εργαλείων της Impulse C καθώς και με περεταίρω ανάλυση και βελτίωση της σχεδίασης.

## 6.4 Βελτίωση Απόδοσης με την χρήση Registers

Με την βοήθεια των εργαλείων του Convey, είδαμε ότι κάποια σήματα κατανάλωναν μεγάλο χρόνο και η σχεδίαση γινόταν συνολικά πιο αργή. Για να βελτιώσουμε την απόδοση της σχεδίασης τοποθετήσαμε ενδιάμεσους καταχωρητές μεταξύ του προβληματικού σήματος και του προορισμού του, για να έχουμε περισσότερους κύκλους αλλά καλύτερο ρολόι. Αρχικά βλέπαμε ποιο σήμα ήταν προβληματικό. Ακολούθως βρίσκαμε στον HDL κώδικα το κομμάτι κώδικα που επηρεαζόταν. Η ιδιαιτερότητα του εργαλείου να παράγει FSM για την εκτέλεση της σχεδίασης μας επέτρεψε να δούμε σε ποια κατάσταση είχαμε το πρόβλημα. Με την χρήση του Stage Master Explorer, βλέπαμε ποιο κομμάτι Impulse C αντιστοιχούσε στην κατάσταση, και αλλάξαμε τον κώδικα ακολουθώντας το προγραμματιστικό μοντέλο του Impulse C, ούτως ώστε να προσθέσουμε ενδιάμεσους καταχωρητές και περισσότερες καταστάσεις – κύκλους. Επίσης, μεταβλητές που επαναχρησιμοποιούνται στον κώδικα (και μεταφράζονται σαν καταχωρητές), αντικαταστήθηκαν με νέες μεταβλητές ώστε η σχεδίαση να φέρει τους παραγόμενους καταχωρητές πιο κοντά στο κομμάτι της σχεδίασης που τους χρησιμοποιεί.

Όπου τοποθετήθηκε ένας καταχωρητής για να κρατήσει την τιμή που διαβάζετε από την μνήμη, προσθέσαμε ένα επιπλέων κύκλο, και ακολούθως γίνετε η ανάθεση της τιμής και η αύξηση των μετρητών του loop.

Η χρήση ενδιάμεσων καταχωρητών έγινε όπου ήταν απαραίτητη.

## 6.5 Εκτέλεση Πολλαπλών Κλήσεων

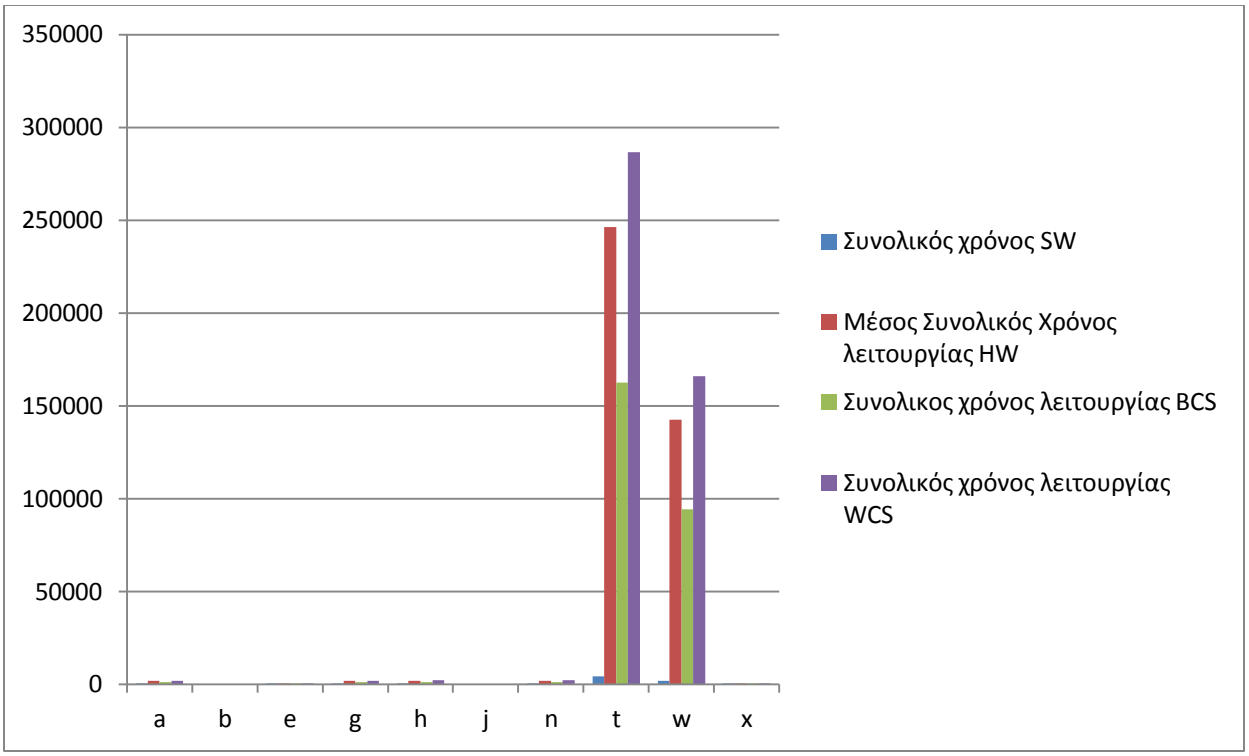
Παρατηρήθηκε ότι ο κώδικας υπολογισμού της μέγιστης πιθανοφάνειας μπορεί να σπάσει σε μικρότερα κομμάτια. Η κάθε κλήση δέχεται σαν όρισμα την ακέραια παράμετρο alignment pattern η οποία έχει επεξηγηθεί σε προηγούμενο κεφάλαιο. Αυτή η τιμή καθορίζει τον αριθμό των επαναλήψεων που απαιτούνται για να ολοκληρωθεί ο υπολογισμός της πιθανοφάνειας. Για κάθε επανάληψη απαιτείται μικρότερη ποσότητα δεδομένων. Συνεπώς μπορούμε να καλέσουμε την συνάρτηση υπολογισμού της Μέγιστης πιθανοφάνειας είτε μία φορά φορτώνοντας (τόσο στην SW εκτέλεση όσο και στην HW εκτέλεση) όλα τα δεδομένα, ή περισσότερες φορές στέλνοντας μόνο τα απαιτούμενα δεδομένα.

Στην δική μας υλοποίηση σχεδιάσαμε το HW με τέτοιο τρόπο ώστε να δεσμεύει ακόμη μικρότερο ποσοστό της μνήμης, και αλλάξαμε το SW ώστε να καλεί την συνάρτηση που υλοποιείται σε HW *δύο* φορές. Στην πρώτη κλήση δέχεται τα μισά δεδομένα και στην δεύτερη τα υπόλοιπα.

## 6.6 Απόδοση Τελικού Συστήματος

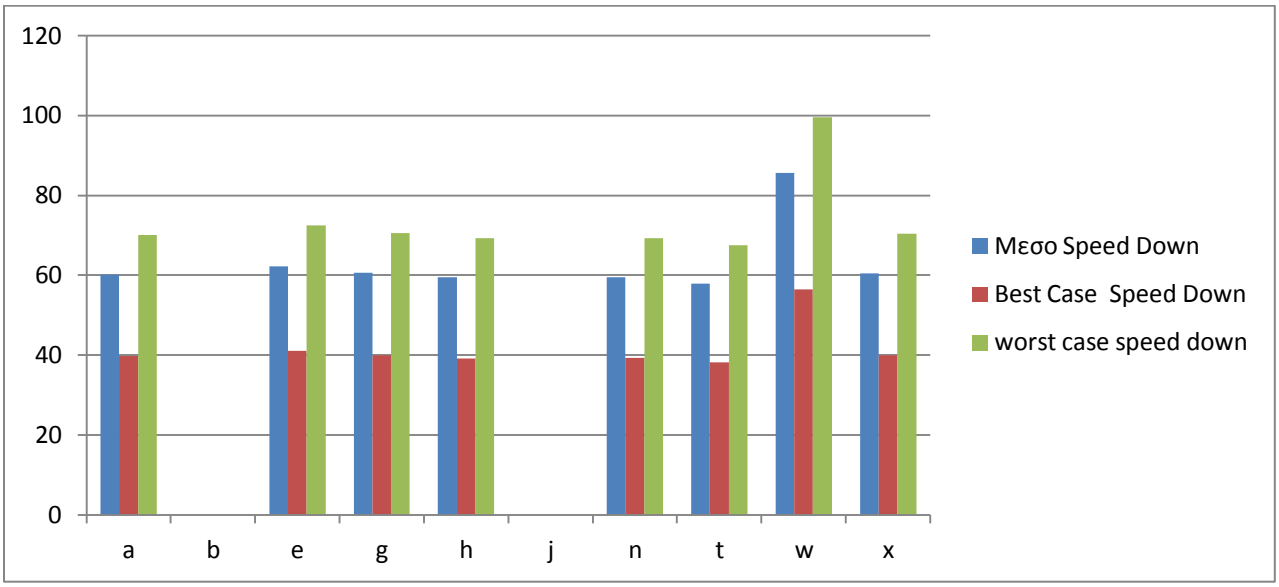
Το πρόγραμμα RAxML εκτελέστηκε σε φορητό υπολογιστή με Core 2 Duo στα 2 GHz με 4 GB RAM. Εκτελέστηκε με διάφορα αρχεία εισόδου και για πολλές περιπτώσεις εκτέλεσης των αλγορίθμων εκτέλεσης και έγινε επιλογή συγκεκριμένου αρχείου για τις μετρήσεις που πληρούσε επαρκώς τα κριτήρια ούτως ώστε τα αποτελέσματα να είναι ρεαλιστικά.

Όπως και πριν, η απόδοση έχει υπολογιστεί για την διαφορά του συνολικού χρόνου εκτέλεσης της συνάρτησης ενδιαφέροντος σε HW και SW. Ο υπολογισμός έγινε υπολογίζοντας τον χρόνο κλήσης της κάθε περίπτωσης (tipCase) και έγινε υπολογισμός του συνολικού χρόνου που καταναλώθηκε κατά την εκτέλεση του HW και του χρόνου που αναλώνεται για το I/O.




Διάγραμμα 16 Συνολικός χρόνος εκτέλεσης τελικής υλοποίησης

Και πάλι η απόδοση του συστήματος είναι χειρότερη από την SW εκτέλεση παρά της βελτιώσεις.



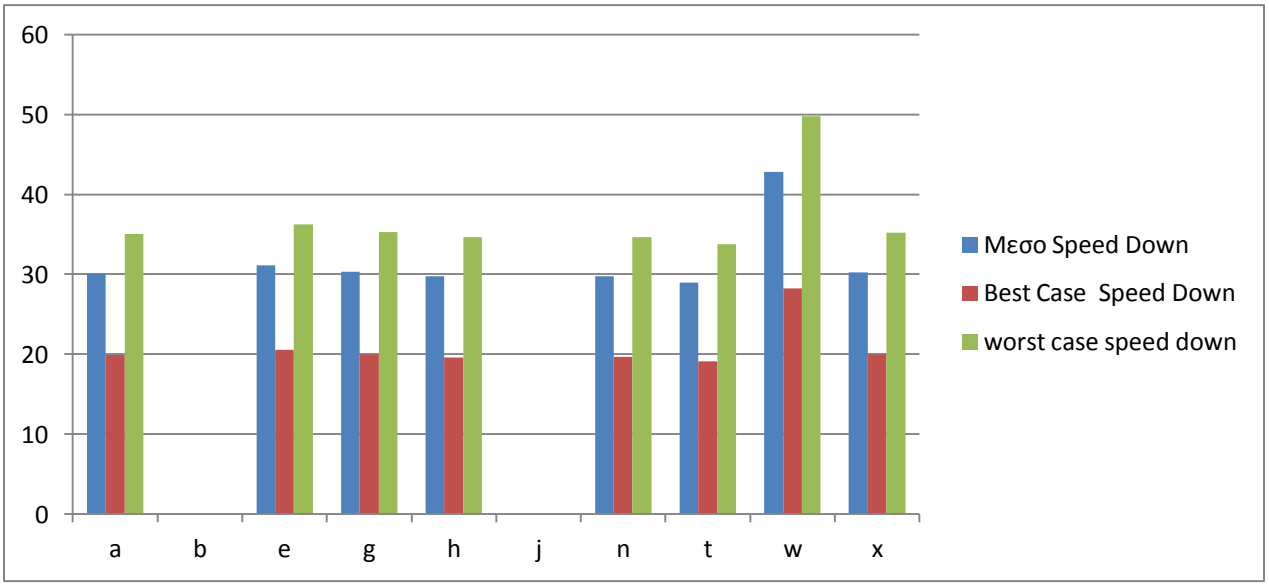

Διάγραμμα 17 speed down τελικής υλοποίησης

Και η τελική σχεδίαση, όπως και η αρχική έχει μεγάλο speed down, περίπου στα ίδια επίπεδα με την αρχική σχεδίαση. Και σε αυτή την περίπτωση πιο αντιπροσωπευτική είναι η μέση περίπτωση, λόγο των ισοζυγισμένων εκτελέσεων των περιπτώσεων (tipCase).

Η HW κλήση της συνάρτησης ενδιαφέροντος ήταν πιο αργή από την αντίστοιχη κλήση σε SW. Η μικρότερη επιβράδυνση που είχαμε είναι – 57.98 x (0.017x), η μέση επιβράδυνση είναι - 63.26 x (0.015x) και η μεγαλύτερη επιβράδυνση είναι – 85.56 x (0.011 x).

Στην συγκεκριμένη εκτέλεση πρέπει να επισημάνουμε ότι έγιναν δύο διαδοχικές κλήσεις της συνάρτησης ενδιαφέροντος. Πρακτικά μπορεί να γίνει σχεδίαση αξιοποιώντας δύο FPGA από τις τέσσερεις που διαθέτει το Convey. Έτσι ο χρόνος εκτέλεσης θα μειωνόταν στο μισό, αφού θα έχουμε παραλληλισμό των κλήσεων.

Σε αυτή την περίπτωση το speed down θα ήταν (θεωρητικά) το ακόλουθο:




Διάγραμμα 18 Θεωρητική βελτίωση speed down

Με μικρότερη επιβράδυνση –28.99 x (0.034 x), μέση επιβράδυνση – 31,63 x (0.032 x) και μεγαλύτερη επιβράδυνση – 42.78 x (0.023 x ).

Οι δυνατότητες του Convey μας επιτρέπουν να υπολογίσουμε θεωρητικά την παραπάνω απόδοση αφού διαθέτει τους απαραίτητους πόρους για να γίνει αυτό. Το εργαλείο Impulse C δίνει την δυνατότητα χρήσης και των τεσσάρων FPGA του Convey, αλλά λόγω χρόνου δεν μπορέσαμε να υλοποιήσουμε κάποια σχεδίαση που να εκμεταλλεύεται την δυνατότητα παράλληλης χρήσης των FPGA.

## 6.7 Χρήση Πόρων Συστήματος

Πραγματοποιήθηκε Synthesis με το εργαλείο Xilinx 12.4 και προέκυψαν τα ακόλουθα αποτελέσματα, όσο αφορά την συχνότητα του ρολογιού και τους πόρους που καταναλώνει η συγκεκριμένη αρχιτεκτονική.

Clock frequency 150 MHz

| Device Utilization Summary xc5vlx330-2ff1760 | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Registers | 10790 | 207360 | 5% |
| Number of Slice LUTs | 19944 | 207360 | 9% |
| Number of fully used LUT-FF pairs | 9174 | 24763 | 42% |
| Number of bonded IOBs | 597 | 1200 | 49% |
| Number of Block RAM/FIFO | 127 | 288 | 44% |
| Number of BUFG/BUFGCTRLs | 1 | 32 | 3% |
| Number of DSP483s | 184 | 192 | 95% |

Πίνακας 2 Device Utilization Summary xc5vlx330-2ff1760

Η ίδια εκτίμηση πόρων όπως την υπολόγισαν τα εργαλεία του Convey

| Device Utilization Summary Convey | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Registers | 43,757 | 207,360 | 21% |
| Number of Slice LUTs | 52,828 | 207,360 | 25% |
| Number of fully used LUT-FF pairs | 66,578 | | |
| Number of bonded IOBs | 861 | 1200 | 71% |
| Number of Block RAM/FIFO | 127 | 288 | 44% |
| Number of BUFG/BUFGCTRLs | 26 | 32 | 81% |
| Number of DSP483s | 184 | 192 | 95% |

Πίνακας 2 Devise Utilization Summary Convey

## 6.8 Σχόλια Περί των Αποτελεσμάτων

Όπως έχουμε δει, στην τελική υλοποίηση έχουμε μεγάλο speed down. Ο κυριότερος λόγος είναι ο χρόνος που καταναλώνεται σε I/O και έχει προσμετρηθεί στα αποτελέσματα που παρουσιάζονται.

Η συνάρτηση ενδιαφέροντος στην SW εκτέλεση, καταναλώνει το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης (75% - 90%) αλλά ο αριθμός των κλήσεων είναι πολύ μεγάλος.

Μια άλλη παράμετρος που επηρεάζει την απόδοση, είναι η κακή τοποθέτηση της σχεδίασης στην FPGA από το εργαλείο Impulse C, κάτι που μας μειώνει δραστικά το ρολόι που μπορούμε να έχουμε.

Ο τρόπος που χρησιμοποιεί το εργαλείο Impulse C για να επικοινωνεί το SW με το HW, (streams, memories …) απαιτούν προσβάσεις στον συνεπεξεργαστή για την δημιουργία και την αρχικοποίηση, χρόνος που έχει προσμετρηθεί στον συνολικό χρόνο εκτέλεσης του HW.

Τέλος, η ιδιαιτερότητα του κώδικα RAxML με τα μεγάλα (και πολλά) εμφωλευμένα loops, μαζί με την αδυναμία του εργαλείου να χειριστεί σωστά το pipelining και το unrolling, αυξάνουν τον χρόνο εκτέλεσης.

## 6.9 Παραγωγικότητα

Παρά τα απογοητευτικά αποτελέσματα της απόδοσης, πρέπει να επισημανθεί η μεγάλη παραγωγικότητα που είχαμε με το εργαλείο. Ο χρόνος που απαιτήθηκε για την ολοκλήρωση του βασικού σχεδιασμού ήταν πολύ μικρότερος από τον (θεωρητικά) χρόνο που θα απαιτούνταν για την σχεδίαση της αρχιτεκτονικής στο χέρι. Επίσης, γίνεται αυτόματα η παραγωγή όλων των απαραίτητων αρχείων που απαιτούνται για την σύνδεση της σχεδίασης με το Convey. Αυτό το πλεονέκτημα μας έδωσε την δυνατότητα να έχουμε διάφορες σχεδιάσεις, να πειραματιστούμε περισσότερο και να αφιερώσουμε περισσότερο χρόνο στην αποσφαλμάτωση και στην βελτίωση της σχεδίασης[36].

Το μέγεθος του κώδικα που γράφτηκε σε Impulse C, σε σχέση με τον κώδικα VHDL που έχει παραχθεί είναι κατά πολύ μικρότερο (αναλογία γραμμών ~ 1/20).

Ο τρόπος που γράφεται ο κώδικας Impulse C, δεν διαφέρει σε μεγάλο βαθμό με τον αντίστοιχο κώδικα C. Έτσι είχαμε ένα πρωτότυπο της σχεδίασης με τον κώδικα C, χωρίς πολλές αλλαγές, και σε μετέπειτα στάδια προχωρήσαμε σε ανασχεδιασμό του κώδικα για να έχουμε παραλληλισμό και καλύτερη διαχείριση μνήμης. Είχαμε την κάθε νέα έκδοση σε σύντομο χρονικό διάστημα, αφού οι αλλαγές γίνονταν στο Impulse C κομμάτι κώδικα, δεν χρειαζόταν να πειράξουμε την VHDL (νέα port map, έλεγχο για τα εσωτερικά σήματα κτλ), κάτι που θα ήταν πολύ χρονοβόρο αν η σχεδίαση γινόταν στο χέρι.

Με την χρήση των εργαλείων Simulation του Impulse C, μπορούμε να δούμε την ροή των δεδομένων πριν την παραγωγή του SW και την εκτέλεση της εφαρμογής για να μπορέσουμε να διορθώσουμε λάθη και να ελαττωθεί ο χρόνος της αποσφαλμάτωσης που θα αυξανόταν αισθητά αν έπρεπε να ακολουθήσουμε την διαδικασία των αντίστοιχων εργαλείων της Xilinx. Δίνεται η δυνατότητα να έχουμε εκτυπώσεις (printf()) μέσα στην συνάρτηση που θα μεταφραστεί σε VHDL, και να έχουμε εικόνα της ροής δεδομένων κατά την εκτέλεση.

Συνοψίζοντας, ο χρόνος που απαιτείται για να έχουμε μια ολοκληρωμένη σχεδίαση είναι αισθητά μικρότερος από τον αντίστοιχο χρόνο που θα σπαταλούσαμε για να γράψουμε την σχεδίαση στο χέρι. Η απόδοση ωστόσο ενός

συστήματος γραμμένο με το χέρι θα είναι καλύτερη από την απόδοση του ίδιου συστήματος που παράγεται με την βοήθεια εργαλείων όπως το Impulse C.

Σύμφωνα και με την βιβλιογραφία, η σχέση του χρόνου που απαιτείται για την σχεδίαση μιας εφαρμογής με την χρήση του Impulse C σε σχέση με τον χρόνο σχεδίασης της ίδιας εφαρμογής σε VHDL γραμμένη στο χέρι, είναι 1/4, και η σχέση του χρόνου παραμετροποίησης/επέκτασης της ίδιας εφαρμογής, είναι 1/7.

# Κεφάλαιο 7. Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στο τελευταίο κεφάλαιο αναφέρονται τα τελικά συμπεράσματα, αναφορά στα προτερήματα και τα ελαττώματα του εργαλείου Impulse C και μελλοντικές επεκτάσεις της σχεδίασης.

## 7.1 Πλεονεκτήματα Χρήσης CoDeveloper Impulse C

Το κύριο πλεονέκτημα της σχεδίασης με την χρήση εργαλείων αυτού του τύπου είναι κατά κανόνα η μείωση του απαιτούμενου χρόνου σχεδίασης. Επίσης η σχεδίαση είναι πιο εύκολο να γίνει κατανοητή αφού είναι γραμμένη σε γλώσσα προγραμματισμού C και υπάρχει η δυνατότητα παρακολούθησης της ροής δεδομένων κατά την διάρκεια του Simulation. Ο σχεδιαστής μπορεί εύκολα να κάνει αλλαγές στον κώδικα C, και η παραγωγή του νέου HW γίνεται αμέσως. Το συγκεκριμένο εργαλείο μπορεί να παράγει HDL κώδικα τόσο σε VHDL όσο και σε Verilog, και μπορεί να παραχθεί HW για διαφορετικές FPGA από τον ίδιο κώδικα C.

## 7.2 Μειονεκτήματα Χρήσης CoDeveloper Impulse C

Κατά κύριο λόγο, το μεγαλύτερο μειονέκτημα του εργαλείου είναι ότι η σχεδίαση είναι μεγάλη, κάτι που είναι αναμενόμενο, αφού το εργαλείο ακολουθεί μια συγκεκριμένη διαδικασία παραγωγής του HDL κώδικα. Το συγκεκριμένο θέμα επηρεάζει την δική μας σχεδίαση σε ένα κρίσιμο σημείο στον κώδικα.

Στον αλγόριθμο, το μεγαλύτερο μέρος του υπολογισμού γίνεται σε for loops τα οποία έχουν σαν όριο μια παράμετρο που εξαρτάται από το μέγεθος του αρχείου εισόδου, όπου n είναι το alignment pattern όπως έχει παρουσιαστεί σε προηγούμενο κεφάλαιο, το οποίο αλλάζει σε κάθε data set και μπορεί να φτάσει μέχρι τον αριθμό των sites στο αρχείο εισόδου. Αν προσπαθήσουμε να παράγουμε κώδικα HDL έχοντας το n σαν μεταβλητή εισόδου η σχεδίαση υπερδιπλασιάζετε, και απαιτητέ μεγαλύτερη μνήμη με αποτέλεσμα να μην χωράει στην FPGA. Αν η τιμή του n αλλάξει manual τότε η σχεδίαση δεν μεγαλώνει. Το πρόβλημα ωστόσο παρουσιάζεται μόνο αν στείλουμε την μεταβλητή μέσω stream. Για να το αποφύγουμε στέλνουμε την μεταβλητή μέσω της μνήμης. Επίσης, αν η τιμή του n είναι σταθερή, ανεξάρτητα από την τιμή, το παραγόμενο hardware είναι το ίδιο σε μέγεθος, αλλά χρειάζεται περισσότερους(ή λιγότερους) κύκλους για να ολοκληρωθεί. Αν το hardware είναι

παραμετροποιημένο (αν δηλαδή η εκτέλεσή του εξαρτάται από την είσοδο n) τότε η σχεδίαση είναι ελαφρώς μεγαλύτερη αλλά η εκτέλεσή του είναι πολύ πιο αργή.

Το alignment pattern περιορίζει το μέγιστο μέγεθος των ακολουθιών εισόδου. Στην συγκεκριμένη υλοποίηση μπορεί να δεχτεί σετ ακολουθιών με alignment pattern μέχρι 716. Οι ακολουθίες εισόδου στα δεδομένα ελέγχου είχαν μέγεθος πάνω από 1500 νουκλεοτίδια. Το ιδανικό μήκος ακολουθιών είναι το 716, αλλά εμπειρικά παρατηρήθηκε ότι οι ακολουθίες που πληρούν τα κριτήρια είναι ακολουθίες μήκους μεγαλύτερου των 1000 νουκλεοτιδίων, που είναι και η συνήθης περίπτωση σε πραγματικές φυλογενετικές αναλύσεις.

Μεγάλο μέρος του κώδικα είναι εμφωλευμένα loops,( που τα αποτελέσματα του ενός είναι απαραίτητα για την εκτέλεση του επόμενου), τα οποία για να μεταφραστούν σε HDL κώδικα καταναλώνουν μεγάλο κομμάτι λογικής και η σχεδίαση γίνεται με μεγάλο ρολόι. Δεν είναι εύκολο να χειριστεί κάποιος τον C κώδικα με τέτοιο τρόπο ώστε να μεταφραστεί σε pipeline πολλών επιπέδων, με αποτέλεσμα η σχεδίαση να μην είναι βέλτιστη.

Αυτό έχει σαν αποτέλεσμα να μειωθεί η καθυστέρηση του pipeline(περισσότερα αποτελέσματα σε λιγότερους κύκλους), αλλά έχουμε μείωση στην απόδοση του ρολογιού, που μπορεί να μειώσει γενικά την απόδοση όλου του προγράμματος. Στο πρόβλημα που μελετήθηκε στην διπλωματική είχαμε τετραπλά loops καθώς και loops τα οποία έπρεπε να υπολογίσουν κάποιες τιμές που θα χρησιμοποιούνταν σε κάποιο επόμενο loop,καθώς και μεγάλα loops που περιέχουν δύο ή τρία διπλά και τριπλά loops.Αν προσπαθήσουμε να εφαρμόσουμε το pragma UNROLL σε κάποιο από τα loop, τότε η σχεδίαση γίνεται πολύ μεγάλη και με αργό ρολόι.

Ένα άλλο μεγάλο μειονέκτημα της σχεδίασης με το εργαλείο Impulse C είναι το γεγονός ότι για μια εκτέλεση HW απαιτούνται πολλές κλήσεις του συνεπεξεργαστή. Αυτό έχει σαν αρνητικό μειονέκτημα την αύξηση του χρόνου που καταναλώνεται στο I/O. Πιο συγκεκριμένα, για κάθε κλήση συνάρτησης από τις βιβλιοθήκες του Impulse C (co_memory_ptr, co_stream_write, co_stream_read,co_stream_create), γίνεται κλήση του συνεπεξεργαστή.

Ένα σημείο που είναι αρκετά σημαντικό όταν σχεδιάζουμε με Impulse C είναι και το γεγονός ότι δεν μπορούμε να έχουμε fine grained control. Αν σε κάποια φάση της σχεδίασης πρέπει να προστεθούν registers για καθυστέρηση κύκλου, δεν

μεταφράζονται εύκολα από τον compiler. Σαν συνέπεια αυτού, είναι η δυσκολία να σχεδιαστεί αποδοτικά λογική με pipeline[36].

## 7.3 Συμπεράσματα

Η παρούσα διπλωματική παρουσιάζει την σχεδίαση συστήματος σε αναδιατασσόμενη λογική με την χρήση του εργαλείου Impulse C που υλοποιεί την συνάρτηση υπολογισμού μέγιστης πιθανοφάνειας φυλογενετικών δέντρων με τρόπο *λιγότερο* αποδοτικό από ένα συμβατικό υπολογιστή.

Έχουν υλοποιηθεί τρεις αρχιτεκτονικές στην προσπάθειά μας να έχουμε αποδοτική σχεδίαση, που να μπορεί να επεξεργαστεί αρχεία εισόδου με ικανοποιητικό μέγεθος. Στην πρώτη σχεδίαση, έγινε μια μικρή αρχιτεκτονική, που δέχεται μικρά αρχεία, ούτως ώστε να πειραματιστούμε με το εργαλείο Impulse C και την συμβατότητά του με το Convey. Στην πρώτη σχεδίαση δεν έγιναν μετρήσεις απόδοσης, αλλά μόνο πιστοποίηση της σωστής λειτουργίας, για τον λόγο ότι η σχεδίαση δεν μπορεί να δεχθεί ρεαλιστικά δεδομένα.

Στην δεύτερη υλοποίηση, επεκτείναμε την σχεδίαση για να μπορεί να δεχθεί datasets με περισσότερα δεδομένα και να μπορεί να εκτελέσει φυλογενετικές αναλύσεις που αναφέρονται τουλάχιστον στην βιβλιογραφία.

Στην τρίτη υλοποίηση βελτιώσαμε την αρχιτεκτονική, μειώσαμε τους πόρους και βελτιώσαμε την απόδοση. Επίσης παραμετροποιήθηκε με τέτοιο τρόπο ώστε η σχεδίαση να μην επηρεάζεται από το αρχείο εισόδου, αλλά με τις κατάλληλες αλλαγές στο SW να μπορεί να δέχεται dataset χωρίς περιορισμούς.

## 7.4 Παρατηρήσεις

Συνοπτικά μερικές παρατηρήσεις για τα αποτελέσματα και την χρήση του εργαλείου.

- Το τελικό αποτέλεσμα ήταν να έχουμε μεγάλο speed down στην σχεδίαση με την χρήση του εργαλείου Impulse C.
- Ο χρόνος υλοποίησης ήταν κατά πολύ μικρότερος από τον χρόνο που θα απαιτούνταν για την σχεδίαση της ίδιας αρχιτεκτονικής στο χέρι. Αυτό μας έδωσε την δυνατότητα να έχουμε πολλές διαφορετικές σχεδιάσεις.
- Η παραγόμενη σχεδίαση καταναλώνει μεγάλο μέρος από τους διαθέσιμους πόρους, κάτι που περιορίζει το μέγεθος της εφαρμογής.

- Αν ο κώδικας Impulse C είχε γραφτεί με διαφορετικό τρόπο ίσως να είχαμε καλύτερη απόδοση, αλλά όχι σε μεγάλο βαθμό λόγω της ιδιαιτερότητας του κώδικα RAxML.

## 7.5 Αξιολόγηση Εργαλείου CoDeveloper Impulse C

Γενικά ο κώδικας VHDL γραμμένος στο χέρι είναι πιο μικρός και πιο γρήγορος στην εκτέλεση, παρά την μετάφραση του από C. Ο γενικός κανόνας είναι πως όσο πιο χαμηλό είναι το επίπεδο της αφαιρετικότητας, τόσο πιο δύσκολο και χρονοβόρο είναι να σχεδιαστεί , αλλά και πιο ακριβό να συντηρηθεί. Ακόμα, όσο το μέγεθος των εφαρμογών μεγαλώνει, τόσο λιγότερο πρακτικό είναι να πετύχεις βελτιστοποίηση σχεδιάζοντας την εφαρμογή με το χέρι. Αυτοί οι παράγοντες οδήγησαν στο να σχεδιαστούν εργαλεία που να παράγουν HDL γλώσσα, ούτως ώστε να έχουμε αποτελεσματικά και αποδοτικά προγράμματα, αν υποθέσουμε ότι ο χρόνος που απαιτείται για βελτιστοποίηση με το χέρι είναι περιορισμένος. Το πλεονέκτημα του Impulse C, για μικρότερες σχεδιάσεις είναι ο μικρός χρόνος σχεδίασης ενός *πρωτοτύπου*, και όχι τόσο η αποδοτική σχεδίαση. Ωστόσο, το εργαλείο CoDeveloper παρέχει εκτεταμένη βιβλιογραφία και παραδείγματα για να βοηθήσει τον προγραμματιστή να καταλάβει πώς να γράψει κώδικα C με τέτοιο τρόπο ώστε να είναι αποδοτικός όταν αυτός μετατραπεί σε hardware.

## 7.6 Μελλοντικές Επεκτάσεις

Η αρχιτεκτονική έχει σχεδιαστεί με τέτοιο τρόπο ώστε να είναι δυνατόν να επεκταθεί και άλλο. Σε μελλοντικό στάδιο μπορεί να βελτιστοποιηθεί ο παραγόμενος κώδικας και να σχεδιαστούν κομμάτια του κώδικα από τον προγραμματιστή για να επιτύχει μικρότερο κύκλο ρολογιού.

### 7.6.1 Περαιτέρω Παραμετροποίηση Παραγόμενου HDL Κώδικα.

Όπως αναφέρθηκε προηγουμένως, η υλοποίηση είναι παραμετροποιημένη για να δέχεται μεγάλα αρχεία. Τα πειράματα που έγιναν καλύπτουν ένα μεγάλο εύρος αρχείων εισόδου και αρχεία εισόδου τουλάχιστον τόσο μεγάλα όσο αναφέρονται στην βιβλιογραφία για παρόμοιες υλοποιήσεις.

Εντούτοις μπορεί να γίνει προσπάθεια για να γίνει η υλοποίηση ακόμα πιο παραμετροποιημένη, ούτως ώστε να μπορεί να επεξεργαστεί αρχεία οποιουδήποτε μεγέθους.

Με αυτό τον τρόπο δεν χρειάζεται να αλλάξει το HW, οι αλλαγές γίνονται μόνο στο SW κομμάτι του κώδικα. Το αποτέλεσμα αυτής της παραμετροποίησης είναι να έχουμε περισσότερες κλήσης της HW συνάρτησης, άρα μεγαλύτερο συνολικό χρόνο εκτέλεσης.

Μελλοντικά μπορεί να ξαναγίνει υλοποίηση της συνολικής σχεδίασης ώστε να έχουμε περισσότερα αντίγραφα της σχεδίασης σε μια FPGA, για να μπορούμε να εκτελούμε περισσότερες από μια κλήσεις κάθε φορά.

### 7.6.2 Μελλοντική Δουλειά

Μελλοντική δουλειά μπορεί να γίνει τόσο σχετικά με τον αλγόριθμο RAxML, όσο και με το εργαλείο CoDeveloper Impulse C.

Σχετικά με το RAxML:

- η σχεδίαση σε hardware και άλλων συναρτήσεων του προγράμματος.
- Να γίνει καλύτερη εκτίμηση της ροής δεδομένων ούτως ώστε να χρησιμοποιηθούν περισσότεροι πόροι στο convey (περισσότερα αντίγραφα της σχεδίασης, και οι τέσσερεις FPGA).
- Να γίνει καλύτερη σχεδίαση ώστε να μπορεί να δεχτεί ακόμα μεγαλύτερα αρχεία εισόδου( ακολουθίες μεγαλύτερου μήκους).
- Χρήση της συνάρτησης υπολογισμού maximum likelihood και σε άλλα παρεμφερή προγράμματα με το RAxML.
- Σχεδιασμός / αναπροσαρμογή της συγκεκριμένης αρχιτεκτονικής ώστε να υποστηρίζει και τις άλλες εκδόσεις του RAxML (η συγκεκριμένη σχεδίαση αφορά την ακολουθιακή έκδοση του προγράμματος. Είναι διαθέσιμες και οι εκδόσεις για αρχιτεκτονική SSE3[24], AVX[23], threaded έκδοση , threaded SSE3, threaded AVX, coarse-grain MPI[22] , coarse-grain MPI SSE3, coarse-grain MPI AVX, Hybrid MPI/Pthreads version που η κάθε μια έχει υλοποίηση της συνάρτησης υπολογισμού maximum likelihood με ορισμένες διαφορές).

Σχετικά με το CoDeveloper Impulse C:

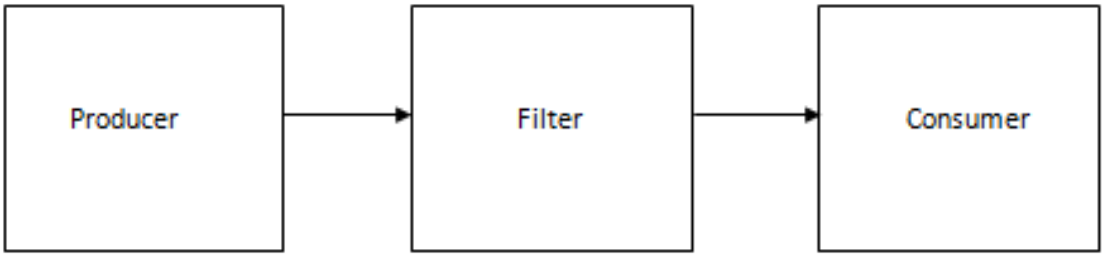
- Μελέτη περισσότερων δυνατοτήτων του εργαλείου (state machines, registers, pipeline processes, more components .. )

- Σχεδίαση εφαρμογών σε FPGAs άλλου τύπου.
- Σχεδίαση εφαρμογών σε FPGA με συνεπεξεργαστή.
- Σχεδίαση εφαρμογών που να χρησιμοποιούν περισσότερους πόρους του Convey (τέσσερεις FPGAs).
- Μελέτη του εργαλείου για εφαρμογές επεξεργασίας εικόνας και ήχου.
- Επανασχεδίαση εφαρμογών με Impulse C για σύγκριση απόδοσης.
- Σύγκριση του εργαλείου με παρεμφερή εργαλεία.

# Παράρτημα A. CoDeveloper και Impulse C

## A.1 General Application Templates

Το εργαλείο δίνει την δυνατότητα να ξεκινήσουν τα project ακολουθώντας μια συγκεκριμένη μορφή, η οποία βέβαια μπορεί να αλλάξει στην πορεία κάνοντας τις απαραίτητες αλλαγές στα κομμάτια του κώδικα που μας ενδιαφέρουν. Γενικά η δομή που ακολουθεί το εργαλείο είναι η δομή Producer – Filter Consumer.

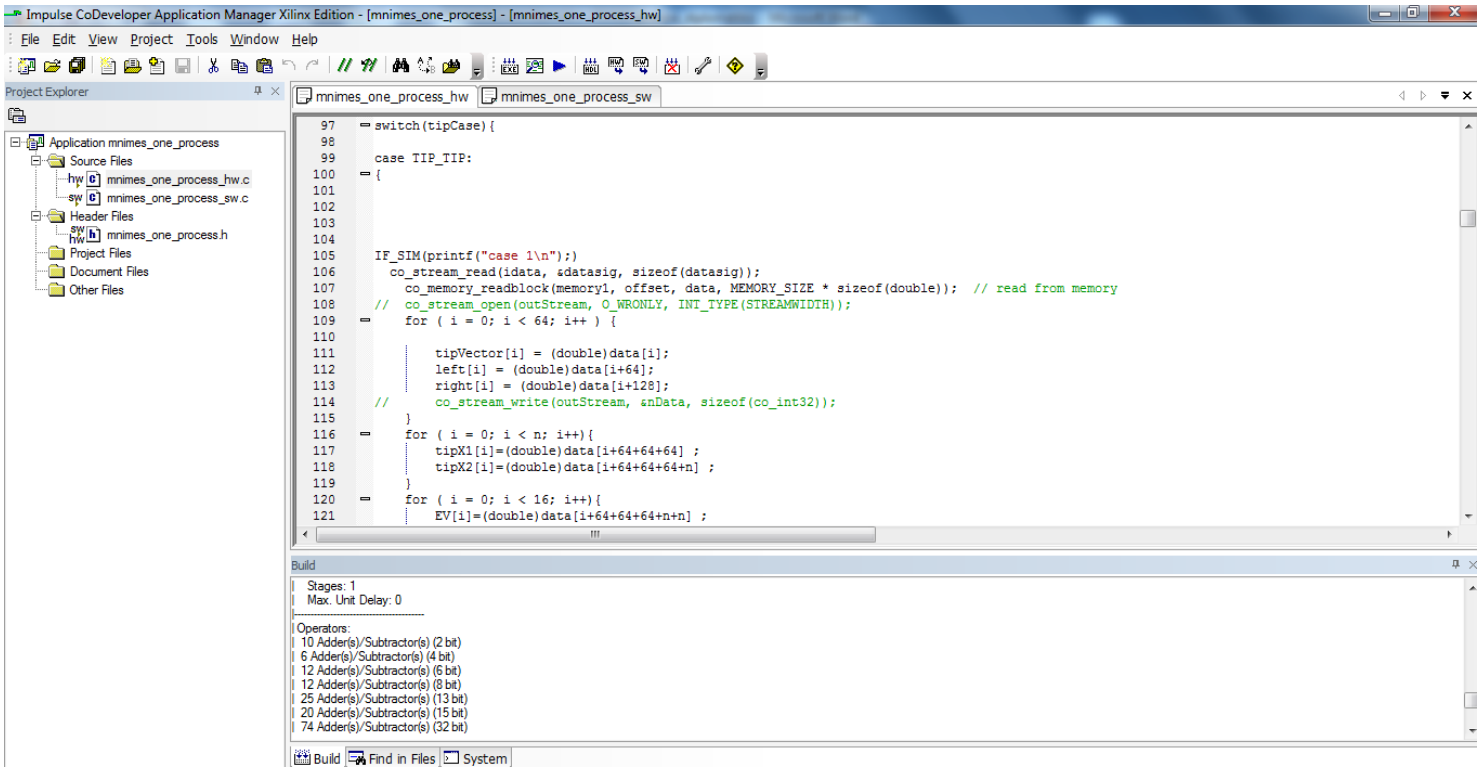



A-2 The producer-filter-consumer template

Κάθε κομμάτι είναι γραμμένο σε γλώσσα C – Impulse C. Το μέρος του producer και του Consumer είναι τα κομμάτια του software και το filter είναι το κομμάτι του hardware, δηλαδή το κομμάτι του hardware κώδικα που θέλουμε να επιταχύνουμε. Ο producer στέλνει τα δεδομένα που χρειάζονται στο filter μέσω stream, και στην συνέχεια το filter επιστρέφει τα δεδομένα στο Consumer για να συνεχίσει το software. (Τα κομμάτια producer και Consumer είναι μέρη του ιδίου κώδικα software, είναι δύο συναρτήσεις μέσα στο ίδιο πρόγραμμα).

```
Thesis_sw.c
void Producer(co_stream inputStream)
{
 Software process όπου έχουμε το κύριος πρόγραμμα
co_stream_open(inputStream,O_WRONLY,INT_TYPE(STREAMWIDTH));
co_stream_write(inputStream, &testValue, sizeof(co_int32));
co_stream_close(inputStream);
μέσω streams στέλνω στο hardware τα δεδομένα δίνοντας του την «έγκριση» να ξεκινήσει.
}
```

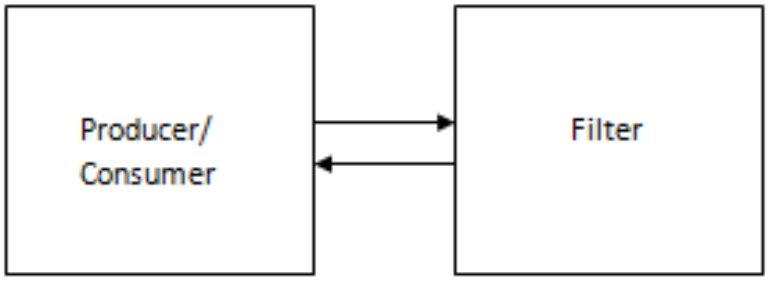
```
void Consumer(co_stream outputStream)
{
co_stream_open(outputStream, O_RDONLY, INT_TYPE(STREAMWIDTH));
co_stream_read(outputStream, &testValue, sizeof(co_int32));
co_stream_close(outputStream);
```
software process όπου το κυρίως πρόγραμμα δέχεται τα δεδομένα από το hardware.
```
}

Thesis_hw.c
void thesis(co_stream inputStream, co_stream outputStream)
{
co_stream_open(inputStream, O_RDONLY, INT_TYPE(STREAMWIDTH));
co_stream_open(outputStream, O_WRONLY, INT_TYPE(STREAMWIDTH));
co_stream_read(inputStream, &nSample, sizeof(co_int32));
```
Hardware process όπου τα δεδομένα επεξεργάζονται από το hardware και στην συνέχεια τα αποτελέσματα επιστρέφονται στο software.
```
co_stream_write(outputStream, &nSample, sizeof(co_int32));
co_stream_close(inputStream);
co_stream_close(outputStream);
}
```




A-3 Γραφικό περιβάλλον εργασίας CoDeveloper Impulse C

Μπορούμε να έχουμε μνήμες, παράλληλες διεργασίες, pipelined διεργασίες, πολλαπλά streams δεδομένων.

Στη παρούσα διπλωματική έχει χρησιμοποιηθεί το one-process testbench που ακολουθεί την ίδια λογική με τα προηγούμενα, αλλά τα κομμάτια Producer και Consumer αποτελούν μέρος τις ίδιας συνάρτησης.[35]




A-4 The one process testbench template

Επίσης έχει χρησιμοποιηθεί η μνήμη για την μεταφορά των δεδομένων, τα οποία στο hardware είναι αποθηκευμένα σε δείκτες, και μέσω streams στέλνονται οι

απαραίτητες ακέραιες ποσότητες για την εκτέλεση της συνάρτησης που εκτελείται στο hardware (μέγεθος loop κτλ).

Πρέπει να ληφθεί υπόψη μας ότι το κομμάτι του hardware τρέχει συνέχεια, άρα πρέπει με κάποιο τρόπο (πχ signal) να ενημερώσουμε hardware ότι τα δεδομένα που έφτασαν είναι αυτά που θέλουμε να επεξεργαστεί και να μας επιστρέψει τα αποτελέσματα.

Το παραγόμενο κομμάτι hardware είναι μια FSM χωρισμένη σε Stages. Υπολογίζεται η μέγιστη καθυστέρηση για όλα τα Stages. Κάθε Stage θέλει ένα κύκλο ρολογιού για να εκτελεστεί και ο Compiler θέτει το μέγιστο ρολόι που μπορούμε να έχουμε για να παίρνουμε τα σωστά αποτελέσματα. Μπορούμε να αλλάξουμε την μέγιστη καθυστέρηση σε κάτι πιο μικρό χρησιμοποιώντας κάποιο pragma (επεξήγηση πιο κάτω), και ο compiler θα ξαναδημιουργήσει το κομμάτι του hardware το οποίο θα τρέχει με πιο γρήγορο ρολόι, αλλά το κομμάτι που πριν είχε την πιο μεγάλη καθυστέρηση, θα ξανασχεδιαστεί με πιο πολλά και πιο μικρά stages.

## A.2 Ειδικά χαρακτηριστικά του προγράμματος CoDeveloper Impulse C

Το εργαλείο δίνει την δυνατότητα να ελέγξουμε την απόδοση της σχεδίασής μας με την χρήση κάποιων flags καθώς και με την βοήθεια κάποιων γραφικών εργαλείων.

### A.2.1 CoBuilder C Pragmas

Τα χαρακτηριστικά του pipelining και του scheduling μπορούμε να τα ελέγξουμε με την χρήση κάποιων προκαθορισμένων pragmas. Τα pragmas έχουν επίδραση σε επίπεδο block, μέσα σε άγκιστρα (“}”).

**#pragma CO PRIMITIVE** : Αν χρησιμοποιήσουμε αυτό το pragma σε ένα κομμάτι κώδικα, τότε αυτό παράγεται σαν ξεχωριστό κομμάτι hardware (Component) και μπορεί να καλεστεί από το hardware ανά πάσα στιγμή.

**#pragma CO FLATTEN** : με αυτό το pragma δίνουμε εντολή στον compiler να χειριστεί το κομμάτι του κώδικα σαν συνδυαστική λογική (combinational logic) αντί σαν μηχανή καταστάσεων (FSM). Με αυτό τον τρόπο το κομμάτι κώδικα που εμπλέκεται θα εκτελεστεί σε ένα κύκλο αντί σε όσους κύκλους είναι η αντίστοιχη FSM.

```
int abs(int a,int b){
  int abs(int a, int b) {
  #pragma CO PRIMITIVE
  return (a>=b) ? a : b;
  }
```

Η συνάρτηση αυτή θα υλοποιηθεί σαν μηχανή καταστάσεων που ελέγχεται από την συνθήκη a>=b και απαιτεί 2 κύκλους για να ολοκληρωθεί.

```
  int abs(int a, int b) {
  #pragma COPRIMITIVE
  #pragma CO FLATTEN
  return (a>=b) ? a : b;
  }
```

Η ίδια συνάρτηση με την χρήση του FLATTEN pragma υλοποιείται σε μόνο μια κατάσταση και θέλει μόνο ένα (μεγαλύτερο) κύκλο ρολογιού για να ολοκληρωθεί.

**#pragma CO PIPELINE** : Η διασωλήνωση των εντολών δεν γίνεται αυτόματα, αλλά απαιτητέ δήλωση του κομματιού κώδικα που θέλουμε να ακολουθήσει αυτή τη μέθοδο. Ο compiler έμμεσα θα χρησιμοποιήσει και το FLATTEN pragma μαζί με το PIPELINE pragma.

```
for (i=0; i<10; i++) {
#pragma CO PIPELINE
co_stream_read(stream_in1, &nSample1, sizeof(int32));
co_stream_read(stream_in2, &nSample2, sizeof(int32));
sum += (nSample2 + nSample2) >> 1;
}
```

Για να μπορέσουμε να χρησιμοποιήσουμε το PIPELINE pragma πρέπει να δηλωθεί στην αρχή του loop και να μην υπάρχουν εμφωλευμένα loops.

Κατά την υλοποίηση του pipeline μπορεί να παραχθούν πολλά stages και η σχεδίασή μας να μεγαλώσει πολύ σε βάθος. Χρησιμοποιώντας και την παράμετρο stageDelay μπορούμε να θέσουμε μεγαλύτερη επιτρεπτή καθυστέρηση στον χρόνο εκτέλεσης των stages -μεγαλώνοντας το ρολόι- για να περιορίσουμε το βάθος του pipeline.

**#pragma CO UNROLL** : Με το pragma UNROLL αντιγράφεται ο παραγόμενος HDL κώδικας όσες φορές χρειάζεται για να υλοποιηθεί η επανάληψη του loop. Για να είναι αποδοτικό σε σχέση με το χώρο πρέπει να προσέξουμε να το χρησιμοποιούμε σε σχετικά μικρά loops.

```
for (tap = 0; tap < TAPS; tap++) {

#pragma CO UNROLL

accum += firbuffer[tap] * coef[tap];

}
```

Οι επαναλήψεις δεν πρέπει να έχουν αλληλεξαρτήσεις μεταξύ τους ή αναθέσεις τιμών που θα εμποδίσουν το loop από το να υλοποιηθεί σαν παράλληλη διαδικασία. Το loop πρέπει να είναι for loop με ακέραιες τιμές και σταθερά όρια.

**#pragma CO SET** : με αυτό το pragma μπορούμε να θέσουμε το μέγιστο delay που θέλουμε να έχει το κάθε stage, δηλαδή με αυτό τον τρόπο κατευθύνουμε τον compiler να επεξεργαστεί διαφορετικά το pipeline για να έχουμε hardware με πιο πολλά και πιο γρήγορα stages ή πιο λίγα και πιο αργά stages.

#pragma CO SET defaultDelay 48 όπου το 48 είναι το gate Delay σε ένα stage.
#pragma CO SET stageDelay 48 όπου το 48 είναι το gate Delay σε ένα stage.

Το stageDelay εφαρμόζει την καθυστέρηση μόνο στο συγκεκριμένο stage που χρησιμοποιείται ενώ το defaultDelay εφαρμόζει καθυστέρηση σε όλα τα stages που ακολουθούν την δήλωσή του.

### A.2.2 co_par_break();

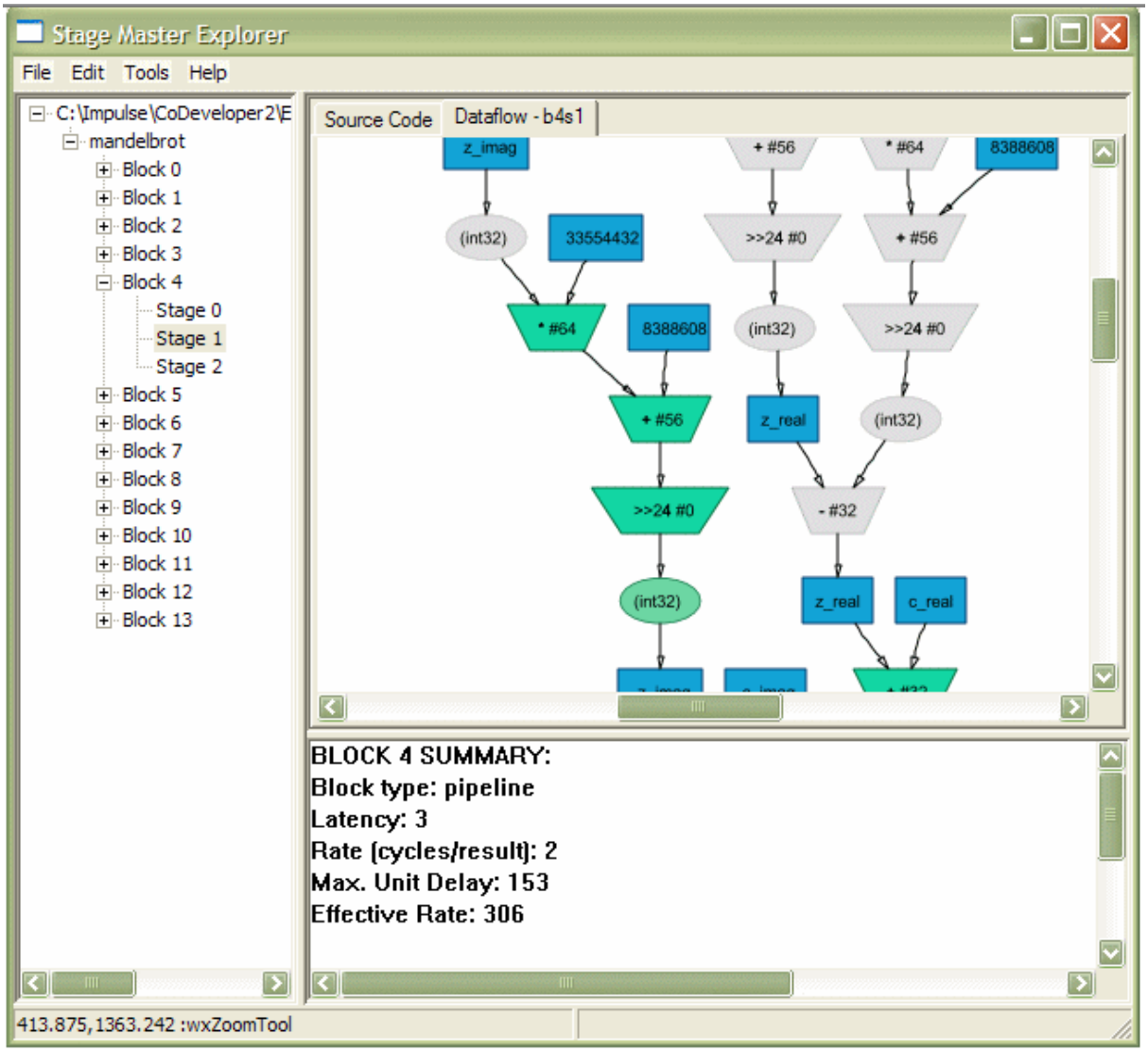
Όταν καλεστεί στο C πρόγραμμα, αυτή η συνάρτηση καθοδηγεί τον optimizer να θέσει ένα φράγμα ενός κύκλου και να καταστείλει την παραγωγή παράλληλης λογικής. Χρησιμοποιείται για να έχουμε μεγαλύτερη ακρίβεια στον έλεγχο και την παραγωγή του παραλληλισμού στον C κώδικα. Για παράδειγμα, αν θέλουμε τον optimizer να δημιουργήσει τέσσερεις κύκλους, απλά προσθέτουμε τρεις δηλώσεις co_par_break, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
Do{
    // Cycle 1 operations
   co_par_break();
    // Cycle 2 operations
   co_par_break();
    // Cycle 3 operations
   co_par_break();
    // Cycle 4 operations
} while(…);
```

Αν ο optimizer δημιουργήσει περισσότερους από τέσσερεις κύκλους για αυτό το κομμάτι κώδικα, σημαίνει ότι κάποια από τα operations δεν μπορούν να εκτελεστούν παράλληλα.

### A.2.3 Stage Master Explorer

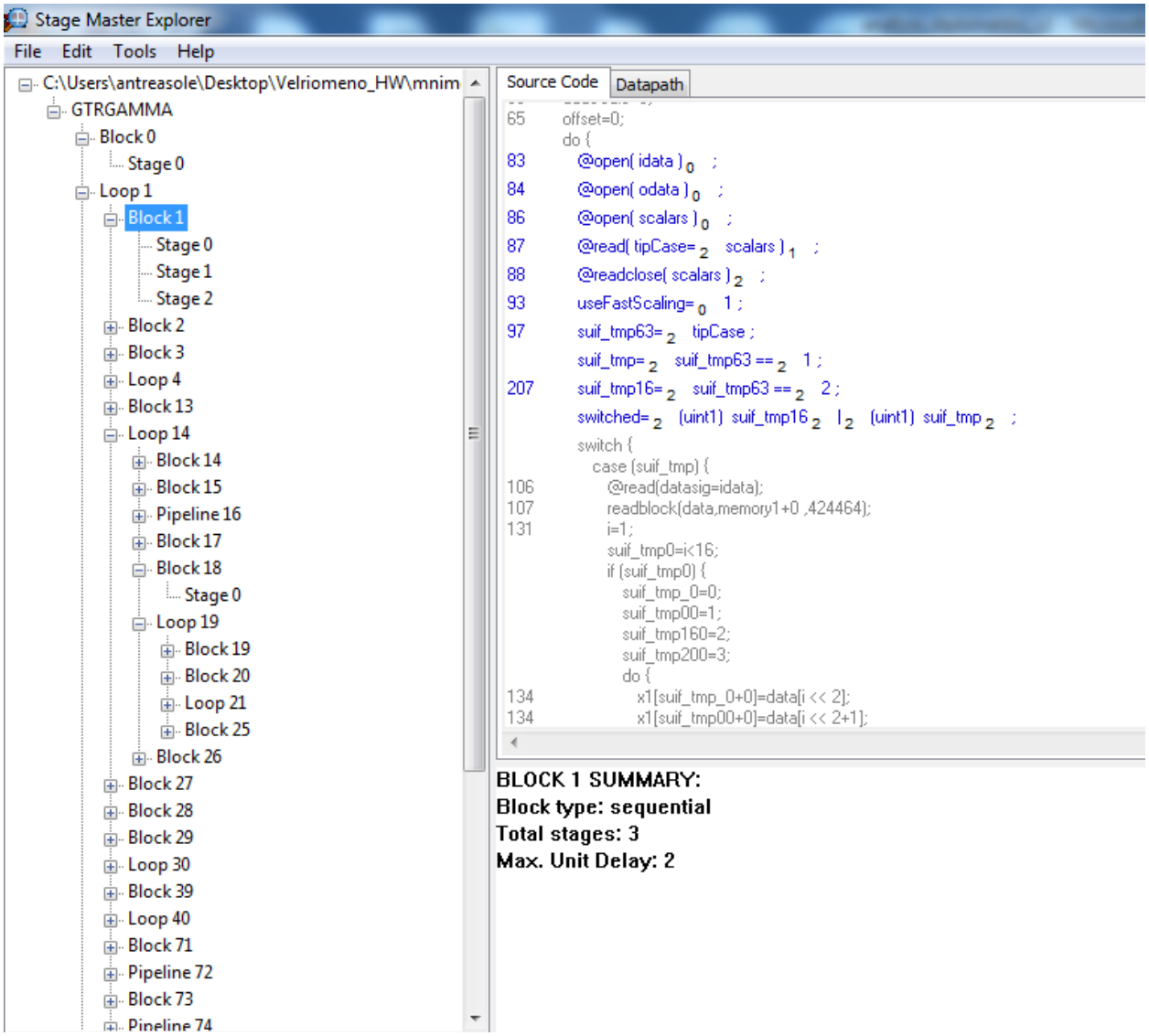
Το Stage master Explorer είναι ένα γραφικό περιβάλλον όπου μπορείς να δεις τα Stages που αποτελούν την σχεδίαση. Όταν επιλέξεις κάποιο Stage μπορείς να δεις το κομμάτι του κώδικα που αντιστοιχεί καθώς και τη ροή τον δεδομένων με




A-5 Stage Master Explorer - Dataflow

σχηματικό τρόπο.

Ένα πλεονέκτημα που σου δίνει το Stage Master Explorer είναι ότι μπορείς να δεις ποια Stages, και κατά κανόνα, ποια loop έχουν την μεγαλύτερη καθυστέρηση (αφού αυτά καταναλώνουν τους περισσότερους κύκλους).



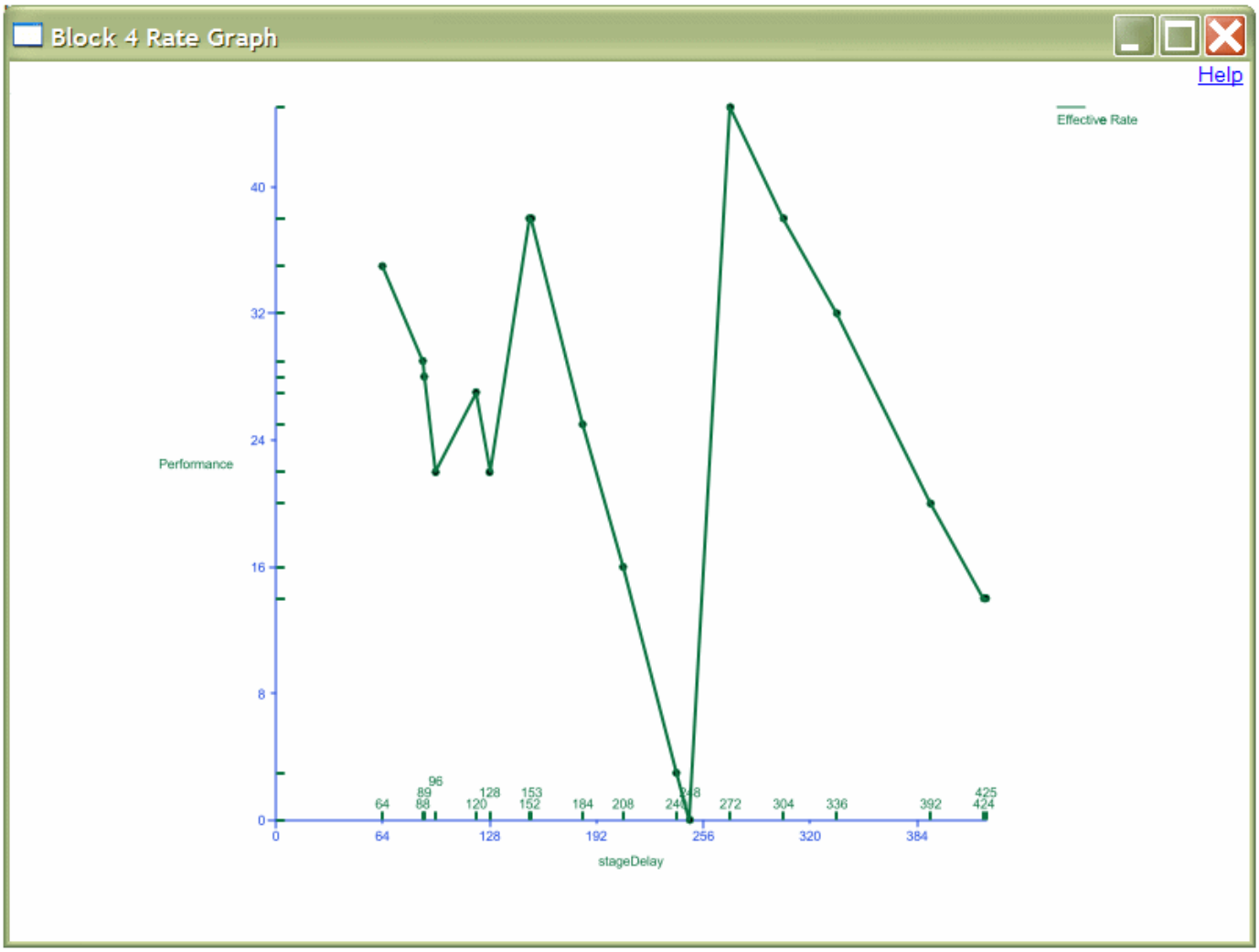

A-6 Stage Master explorer Source Code

Κάθε block (ή loop) αποτελείται από stages. Όταν επιλέξουμε κάποιο block βλέπουμε σε ποιο κομμάτι κώδικα αντιστοιχεί, και ποιες εντολές εκτελούνται παράλληλα. Σε κάθε γραμμή εντολών υπάρχουν κάποιοι δείκτες που υποδηλώνουν το stage το οποίο αποτελούν. Στο παράδειγμα της εικόνας, το block 1, αποτελείται από τρία stages, (0,1,2) και στο κομμάτι κώδικα που αντιστοιχεί το block φαίνεται η σειρά που θα εκτελεστούν οι εντολές από τους δείκτες στο κάτω δεξιά μέρος της εντολής.

Δίνεται η δυνατότητα να πειραματιστείς με αλλαγές όπως για παράδειγμα χρήση pipeline στα loop και συγκεκριμένης καθυστέρησης για να δεις πόσα νέα stages προκύπτουν και κατά πόσο η απόδοση στο συγκεκριμένο loop βελτιώνεται ή

χειροτερεύει, χωρίς να αλλάξεις προς το παρών κάτι στον κώδικα. Με την βοήθεια του γραφικού περιβάλλοντος Pipeline Graph μπορείς να δεις ένα γράφημα με την σχέση απόδοσης – καθυστέρησης, δηλαδή για ποια καθυστέρηση θα έχουμε την καλύτερη απόδοση στο συγκεκριμένο pipelined loop, και στην συνέχεια να εφαρμόσεις τις αλλαγές μόνιμα στο κώδικα με την χρήση pragmas.

Στο παράδειγμα των δύο εικόνων παρατηρούμε ότι έχουμε δύο picks στη γραφική παράσταση, για stage delay = 165 και για stage delay 272. Στην συγκεκριμένη σχεδίαση ο προγραμματιστής χρησιμοποιεί πιο αργό ρολόι για να έχει μεγαλύτερη απόδοση ( 1 αποτέλεσμα σε λιγότερους κύκλους ).

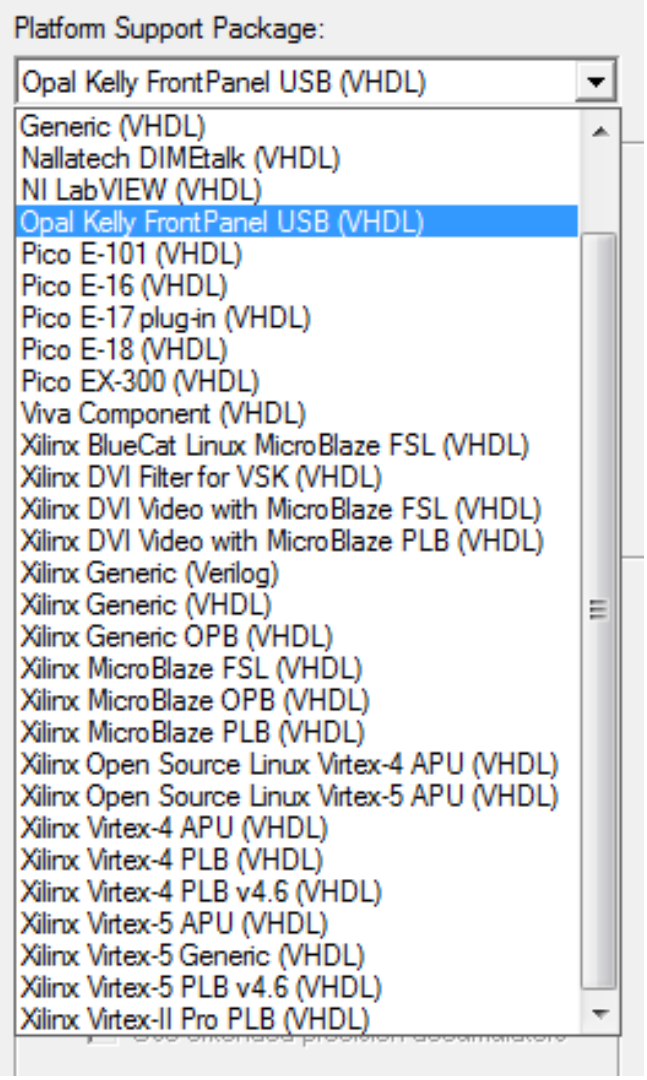



A-7 pipeline rate graph

Πρέπει να λάβουμε όμως υπόψη ότι το ρολόι της σχεδίασης θα τεθεί με βάση την σχεδίαση όλων των stages και όχι μόνο ενός.

## A.3 Target platforms

Altera, Xilinx Microblaze, PowerPC, Convey




**A-8** Platform Support Package

## A.4 Processes, Streams, Signals και Memory

Τα πιο σημαντικά στοιχεία που πρέπει να δαμάσει κάποιος ούτος ώστε να μπορεί να χρησιμοποιήσει την γλώσσα προγραμματισμού Impulse C για την σχεδίαση εφαρμογών είναι οι διαδικασίες , τα streams, τα σήματα και οι μνήμες, που αποτελούν τα βασικά χαρακτηριστικά της δομής ενός προγράμματος Impulse C.

## Α.4.1 Καταλαβαίνοντας τα Processes

Οι διαδικασίες (processes) είναι τα θεμελιώδη στοιχεία για υπολογισμούς σε μια Impulse C εφαρμογή. Μόλις κατασκευαστούν, αρχικοποιηθούν και ξεκινήσουν εκτελούνται σαν ανεξάρτητα και συγχρονισμένα κομμάτια κώδικα σε μια πλατφόρμα.

Προγραμματίζοντας με διαδικασίες στο Impulse C είναι εννοιολογικά παρόμοιο με τον προγραμματισμό με threats. Όπως και στον προγραμματισμό με threats, κάθε Impulse C process έχει τη δική της ροή ελέγχου και δεδομένων (control flow), είναι ανεξάρτητα συγχρονισμένη και έχει πρόσβαση στους δικούς της πόρους μνήμης (ανάλογα με την πλατφόρμα που τρέχει). Γι' αυτό το λόγο είναι σχετικά εύκολο να μετατρέψεις ένα πρόγραμμα που είναι γραμμένο με threaded C σε ένα πρόγραμμα Impulse C λαμβάνοντας υπόψη μερικές διαφορές.

- Στο νηματικό προγραμματισμό (threaded programming), οι global μεταβλητές και η heap memory, είναι κοινόχρηστη για όλα τα threats. Στο Impulse C η heap memory μπορεί να είναι ρητά κοινόχρηστη αλλά δεν υποστηρίζονται global μεταβλητές.
- Στο νηματικό προγραμματισμό τα threats εκτελούνται στον ίδιο πυρήνα (σε μονοπύρηνα μηχανήματα, και σε πολλούς πυρήνες σε πολυπύρηνα), ενώ στην Impulse C γίνεται η υπόθεση ότι κάθε διαδικασία εκτελείται σε ένα ανεξάρτητο και συγχρονισμένο πυρήνα ή σε ένα καθορισμένο block λογικής.
- Το προγραμματιστικό μοντέλο του Impulse C, είναι σχεδιασμένο να υποστηρίζει και software και hardware σχεδίαση με την επικοινωνία και τον συγχρονισμό των διαδικασιών να γίνεται κατά κύριο λόγο σε hardware buffers (FIFOs). Στο νηματικό προγραμματισμό τα threats σχεδόν πάντα επικοινωνούν χρησιμοποιώντας κοινόχρηστες δομές δεδομένων και semaphores.
- Στο Impulse C υποθέτουμε ότι οι διαδικασίες δηλώνονται κατά την έναρξη της εκτέλεσης του προγράμματος ενώ στον νηματικό προγραμματισμό οι διαδικασίες (threats) δημιουργούνται δυναμικά, εκτελούνται και καταστρέφονται.

Οι βιβλιοθήκες προσομοίωσης του Impulse C είναι βασισμένες σε νηματικό μοντέλο προγραμματισμού, επομένως οι global μεταβλητές και οι μνήμες είναι κοινόχρηστες (Shared). Κατά την εκτέλεση της προσομοίωσης οι Impulse C διαδικασίες εκτελούνται σε ένα στοιχείο, τον προσωπικό υπολογιστή μας, ακόμα και αν δεν είναι η πλατφόρμα που έχουμε επιλέξει. Είναι ευθύνη του

προγραμματιστή να αποφύγει την χρήση global μεταβλητών και μνήμης εκτός των βιβλιοθηκών του Impulse. (πχ, η χρήση pointers στο simulation δουλεύει σωστά αλλά δεν μπορεί να δουλέψει σε FPGA).

Η κατασκευή των διαδικασιών γίνεται με την εντολή co_process_create. Αρχικά πρέπει να δηλώσουμε την διαδικασία

```
co_process myProcess;
```

και ακολούθως να την αρχικοποιήσουμε

```
myProcess = co_process_create(“myProcess”,(co_funtion)myProcess,1,s1);
```

Η συνάρτηση co_process_create χρησιμοποιείται για να ορίσει και hardware και software διαδικασίες. Δέχεται τουλάχιστον τρία ορίσματα, όπου το πρώτο είναι ένας δείκτης σε string που είναι το όνομα της διαδικασίας. Δεν είναι απαραίτητο να είναι το ίδιο με το δηλωμένο όνομα που χρησιμοποιείται για την διαδικασία καθεαυτή, είναι περισσότερο ένα label για την παρακολούθηση της διαδικασίας εξωτερικά – για παράδειγμα στο application monitor.

Το δεύτερο όρισμα είναι ένας δείκτης τύπου co_function που δηλώνει την συγκεκριμένη διαδικασία που θα εκτελεστεί.

Το τρίτο όρισμα είναι ο αριθμός των εισόδων και εξόδων από και προς την διαδικασία (μνήμες, streams, signals.. ), και ακολουθείται από τα ορίσματα. Στο πάνω παράδειγμα, ο αριθμός των εισόδων/εξόδων είναι ένα, και ακολουθείται από μόνο ένα όρισμα, το s1.

### Α.4.2 Καταλαβαίνοντας τα Streams

Τα streams είναι κανάλια επικοινωνίας μονής κατεύθυνσης που ενώνουν πολλαπλές διεργασίες, hardware και software. Η συνάρτηση co_stream_create κατασκευάζει ένα stream ορίζει το πλάτος των δεδομένων και το μέγεθος του buffer, και κάνει το stream διαθέσιμο για χρήση στην κατασκευή της διαδικασίας όπως έχει περιγραφεί πιο πάνω. Ένα παράδειγμα δήλωσης και αρχικοποίησης stream είναι το ακόλουθο.

```
co_stream ipassdata;

ipassdata = co_stream_create("idata", INT_TYPE(64), STREAMDEPTH);
```

Έχουμε τρία ορίσματα στην συνάρτηση. Το πρώτο όρισμα είναι ένα προαιρετικό όνομα για το stream που χρησιμοποιείται περισσότερο για debugging και εξωτερική παρακολούθηση. Σε αυτή την περίπτωση το όνομα είναι υποχρεωτικό και μοναδικό.

Το δεύτερο όρισμα καθορίζει τον τύπο των δεδομένων που θα περάσουν μέσα από το stream.

Το τρίτο και τελευταίο όρισμα είναι το βάθος του buffer. Όσο μεγαλύτερο είναι το βάθος τόσο περισσότερο υλικό καταναλώνεται αλλά έχουμε ταχύτερη μεταφορά δεδομένων. Πρέπει να επιλεγεί τέτοιο βάθος ώστε να έχουμε την μέγιστη απόδοση.

### A.4.2.1 Stream I/O

Στην αρχή της διαδικασίας η συνάρτηση co_stream_open θέτει το stream διαθέσιμο για εγγραφή ή ανάγνωση.

```
co_stream_open(input_stream,O_RDONLY,INT_TYPE(32));
```

Η συνάρτηση δέχεται τρία ορίσματα. Το πρώτο είναι το stream που έχει δηλωθεί από πριν, το δεύτερο είναι για να προσδιοριστεί αν το stream είναι για ανάγνωση (O_RDONLY) ή εγγραφή (O_WRONLY) και το τρίτο είναι ο τύπος δεδομένων που θα περάσει μέσα από το stream.

Τα streams είναι μονής κατεύθυνσης άρα πρέπει να ανοίξουν σε ακριβός δύο διαδικασίες. Στην διαδικασία που στέλνει τα δεδομένα με την παράμετρο O_WRONLY και στην διαδικασία που λαμβάνει τα δεδομένα με την παράμετρο O_RDONLY.

Όταν το stream δεν χρειάζεται, μπορεί να κλείσει με την συνάρτηση

```
Co_stream_close(input_stream);
```

που δέχεται σαν όρισμα το stream.  Πρέπει όλα τα streams να κλείσουν για να ολοκληρωθεί το πρόγραμμα.

#### A.4.2.1.1 Χρήση Output Streams

Για να στείλουμε δεδομένα μέσω stream από μια διαδικασία σε μια άλλη, η διαδικασία που θα στείλει τα δεδομένα πρέπει να τα γράψει στο stream. Αυτό γίνεται με την χρήση της συνάρτησης

```
co_stream_write(outputStream,&data[i],sizeof(int32));
```

με επαναληπτική δομή αν χρειάζεται. Δέχεται σαν όρισμα το stream, τα δεδομένα που θέλουμε να στείλουμε και το μέγεθος του δεδομένου. Το stream πρέπει να είναι ήδη ανοικτό όπως έχει περιγραφεί πιο πριν.

#### A.4.2.1.2 Χρήση Input Stream

Για να λάβουμε δεδομένα από ένα stream χρησιμοποιούμε την εντολή

```
co_stream_write(inputStream,&data[i],sizeof(int32));
```

με επαναληπτική δομή αν χρειάζεται. Δέχεται σαν όρισμα το stream, την μεταβλητή που θα κρατήσει τα δεδομένα και το μέγεθος του δεδομένου. Κάθε φορά που καλείται η συνάρτηση θα διαβαστεί το επόμενο δεδομένο που βρίσκεται στο stream.

### A.4.3 Καταλαβαίνοντας τα Signals

Τα streams βοηθούν ώστε το Impulse C πρόγραμμα μας να είναι συγχρονισμένο. Αλλά μερικές φορές ο προγραμματιστής θέλει να έχει μεγαλύτερο έλεγχο του συγχρονισμού και αυτό γίνεται με την βοήθεια των signals. Η χρήση τους είναι απλή. Η συνάρτηση co_signal_wait μπλοκάρει τον κώδικα μέχρι να δεχθεί ένα signal από την συνάρτηση co_signal_post. Η βασικότερη χρήση τους είναι για την ανάγνωση και εγγραφή της μνήμης. Ο κώδικας μπορεί να ξεκινήσει να διαβάζει από την μνήμη πριν να τελειώσει η εγγραφή της, με αποτέλεσμα τα αποτελέσματα να είναι λανθασμένα. Με την χρήση signal, μόλις η εγγραφή τελειώσει, ειδοποιούμε την επόμενη διαδικασία ότι μπορεί να ξεκινήσει ανάγνωση.

Σημείωση: Για Convey δεν μπορούμε να χρησιμοποιήσουμε signals αφού δεν υποστηρίζονται από το support package , αντί για αυτό χρησιμοποιούμε streams.

## Α.4.4 Καταλαβαίνοντας τις μνήμες (Memories)

Το προγραμματιστικό μοντέλο του Impulse C δίνει την δυνατότητα να χρησιμοποιήσουμε κοινόχρηστη μνήμη(Shared memory) σαν εναλλακτικό των streams. Οι μνήμες είναι χρήσιμες για δεδομένα που χρησιμοποιούνται συχνά, για δεδομένα υπό μορφή πίνακα και κυρίως για διαχείριση μεγάλου όγκου δεδομένων.

Η χρήση τους είναι παρόμοια με των streams. Η δήλωση της μνήμης γίνεται με την εντολή

```
co_memory myMemory
```

και η αρχικοποίηση με την εντολή

```
myMemory=co_memory_create("myMemory","mem0", MAXLEN*sizeof(char));
```

Το πρώτο όρισμα είναι το όνομα της μνήμης.

Το δεύτερο όρισμα εξαρτάται από την πλατφόρμα που θέλουμε να χρησιμοποιήσουμε, γι αυτό και αν το ίδιο πρόγραμμα προσπαθήσουμε να το χρησιμοποιήσουμε για άλλη FPGA θα έχουμε πρόβλημα στην μετάφραση.

Το τρίτο όρισμα είναι το μέγεθος της μνήμης που είναι το πλήθος των δεδομένων πολλαπλασιασμένο από το μέγεθος του τύπου των δεδομένων. Προφανώς μπορούμε να χρησιμοποιήσουμε την ίδια μνήμη για διαφορετικούς τύπους δεδομένων.

Η κρίσιμη διαφορά των μνημών από τα streams είναι ο συγχρονισμός. Όταν μεταφέρουμε δεδομένα μέσω stream, η διαδικασία που δέχεται τα δεδομένα δεν θα συνεχίσει μέχρι να αδειάσει το stream. Αυτό δεν γίνεται με τις μνήμες γι αυτό πρέπει να χρησιμοποιήσουμε signals ( όπως αναφέρθηκε και πιο πάνω) για να ελέγξουμε πότε η εγγραφή/ανάγνωση από/προς την μνήμη έχει τελειώσει.

Για την εγγραφή της μνήμης χρησιμοποιείται η εντολή

```
co_memory_writeblock(memory1, offset, data, MEMORY_SIZE * sizeof(double));
```

όπου το πρώτο όρισμα είναι η μνήμη που θέλουμε να γράψουμε, το δέυτερο είναι το offset της μνήμης, το τρίτο είναι τα δεδομένα που θέλουμε να γράψουμε (μπορεί να είναι και πίνακας, δεν χρειάζεται επαναληπτική δομή όπως με τα streams) και τέλος είναι το μέγεθος της μνήμης.

Για ανάγνωση από την μνήμη χρησιμοποιείται η εντολή

co_memory_readblock(memory1,offset,data,MEMORY_SIZE*sizeof(double));

με τα ίδια ορίσματα όπως και πριν.

Η ανάγνωση και η εγγραφή της μνήμης γίνεται με μια μόνο εντολή, αντίθετα με τα streams που για κάθε ανάγνωση ή εγγραφή χρειάζεται μια εντολή, σύνολο τόσες εντολές όσα και τα δεδομένα που στέλνουμε. Αυτό αν μεταφραστεί σε hardware δεν θα πάρει ένα κύκλο, αλλά περισσότερους, και εδώ εμφανίζονται τα προβλήματα συγχρονισμού που έχουμε αναφέρει πιο πάνω, που αντιμετωπίζονται με την χρήση signals.

Ενδεικτικά παρουσιάζονται κάποιοι πίνακες με αποδόσεις της μνήμης σε διάφορες πλατφόρμες όπως αναφέρονται στην βιβλιογραφία.[21]

Memory test results for a Nios Processor in an Altera Stratix FPGA

| Memory test | Transfer Rate (KB) |
|---|---|
| Stream (one-way) | 1529 |
| Stream ( two-way) | 917 |
| RAM-CPU-Stream | 708 |
| Shared memory (4B) | 1167 |
| Shared memory (16B) | 1217 |
| Shared memory (1024B) | 1235 |

Πίνακας 3 Memory test results for a Nios Processor

Memory Test results for the PowerPC processor in a Xilinx Virtex II Pro FPGA

| Memory test | Transfer Rate (KB) |
|---|---|
| Stream (one-way) | 522 |
| Stream ( two-way) | 310 |
| RAM-CPU-Stream | 439 |
| Shared memory (4B) | 3732 |
| Shared memory (16B) | 5466 |
| Shared memory (1024B) | 6056 |

Πίνακας 3 Memory test results for the Power PC in a Virtex II

Memory Test results for the Microblaze processor in a Xilinx Virtex II FPGA

| Memory test | Transfer Rate (KB) |
|---|---|
| Stream (one-way) | 10706 |
| Stream ( two-way) | 4282 |
| RAM-CPU-Stream | 3536 |
| Shared memory (4B) | 3536 |
| Shared memory +Signal (4B) | 3528 |
| Shared memory (16B) | 4784 |
| Shared memory +Signal(16B) | 3881 |
| Shared memory (1024B) | 5484 |
| Shared memory +Signal (1024B) | 3892 |

Πίνακας 4 Memory test results Microblaze processor

## A.5 Header file

Σε κάθε project μπορούμε να έχουμε και αρχεία .h όπου να έχουμε δηλωμένες κάποιες σταθερές, που μπορούν να χρησιμοποιηθούν τόσο στο κομμάτι του SW όσο και στο κομμάτι του HW.

```
#define STREAMDEPTH 2        /* buffer size for FIFO in hardware */
#define STREAMWIDTH 32       /* buffer width for FIFO in hardware */
#define ABS(n) ((n) < 0 ? -(n) : (n))  /*returns the absolute value of n*/
```

Οι δήλωση συναρτήσεων στο .h αρχείο υλοποιείται σε HW χωρίς να χρειαστεί να επέμβει ο προγραμματιστής.

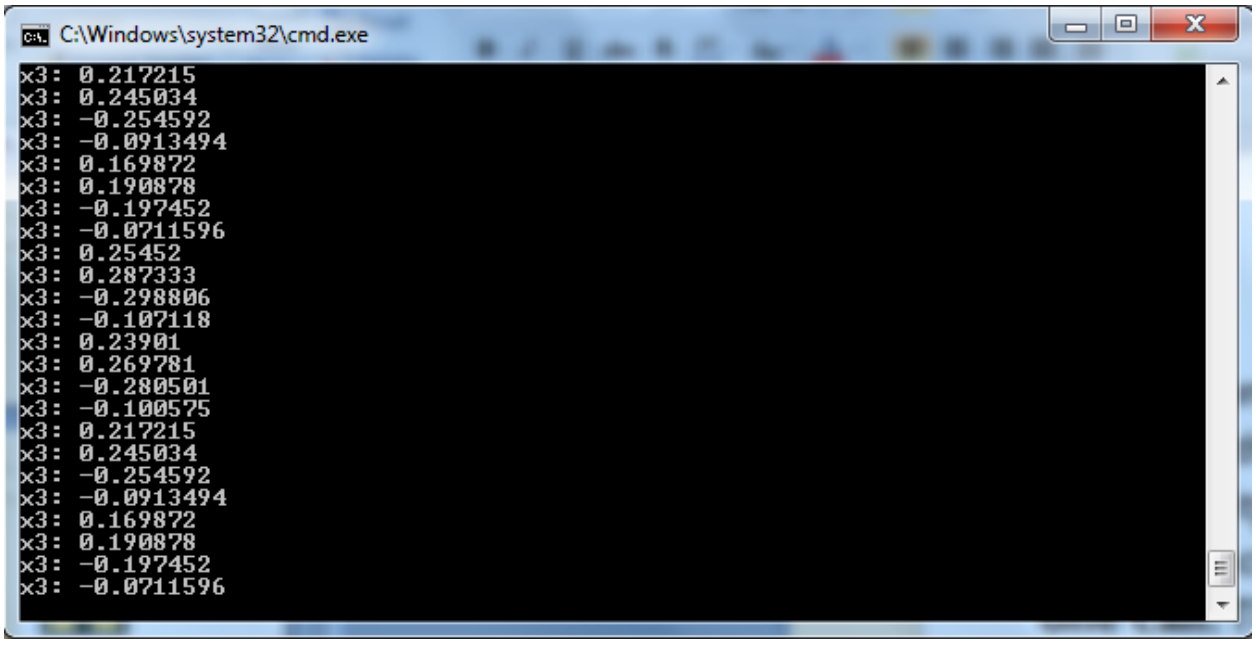
## A.6 CoDeveloper Impulse C και Convey

Το εργαλείο CoDeveloper υποστηρίζει την πλατφόρμα Convey HC-1 και HC-1ex, δίνοντας την δυνατότητα να γράψουμε το κρίσιμο κομμάτι του κώδικα μας σε Impulse C και στην συνέχεια να τον μετατρέψει σε κώδικα HDL. Αυτόματα παράγονται και όλα τα απαραίτητα αρχεία που χρειάζεται το μηχάνημα για να ενώσει το hardware με το software.

## A.7 Software – Hardware CoSimulation

Ένα Software Simulation Executable για windows παράγεται ως αποτέλεσμα από το compile και το linking της Impulse C εφαρμογής μας. Το παραγόμενο simulation executable μπορεί να τρέξει με οποιοδήποτε C debugger.

Με την βοήθεια του εργαλείου αυτού, μπορούμε να δούμε τα αποτελέσματα της εκτέλεσης. Αφού τα αποτελέσματα μεταφέρονται στο Software μέρος της εφαρμογής μπορούμε να τα δούμε στο terminal.




0-1 Αποτελέσματα Simulation

Αφού επιβεβαιώσουμε τα αποτελέσματα του simulation με τα πραγματικά αποτελέσματα, προχωρούμε στην παραγωγή του HDL κώδικα καθώς και της διεπαφής μεταξύ του SW και του HW.

Η παραγωγή του κώδικα γίνεται με το πάτημα ενός κουμπιού.


0-2 Panel για την εξαγωγή εκτελέσιμου αρχείου Simulation και για την παραγωγή της Αρχιτεκτονικής

- Παραγωγή του simulation executable
- Παρακολούθηση εκτέλεσης από το application monitor
- Εκτέλεση του simulation executable
- Παραγωγή του HDL κώδικα
- Εξαγωγή του HW κομματιού
- Εξαγωγή του SW κομματιού

- Clean

Κατά την παραγωγή του hardware μπορούμε να δούμε μια εκτίμηση των μονάδων της FPGA που θα χρησιμοποιηθούν καθώς και το σύνολο των stages που απαιτούνται για την σχεδίασή μας.

```
|----------------------------------------
| Operators:
|  10 Adder(s)/Subtractor(s) (2 bit)
|  6 Adder(s)/Subtractor(s) (4 bit)
|  12 Adder(s)/Subtractor(s) (6 bit)
|  12 Adder(s)/Subtractor(s) (8 bit)
|  25 Adder(s)/Subtractor(s) (13 bit)
|  46 Adder(s)/Subtractor(s) (15 bit)
|  74 Adder(s)/Subtractor(s) (32 bit)
|  86 Comparator(s) (32 bit)
|  3 Comparator(s) (64 bit)
|  11 Floating-point Adder(s)/Subtractor(s) (64 bit)
|  14 Floating-point Multiplier(s) (64 bit)
|----------------------------------------
| Total Stages: 334
| Max. Unit Delay: 75
| Estimated DSPs: 224
|----------------------------------------
```

Στον υπολογισμό των πόρων έχουμε πολλούς αθροιστές, που χρησιμοποιούνται για τον υπολογισμό των πράξεων στην συνάρτηση, αλλά και τον υπολογισμό των μετρητών στις επαναληπτικές δομές. Οι συγκρίσεις γίνονται κατά κύριο λόγο στα loops.

Οι υπολογισμοί παραπάνω είναι υπερεκτιμήσεις. Στο πραγματικό σύστημα έχουμε μικρότερη καθυστέρηση αλλά και λιγότερη κατανάλωση πόρων.

# Παράρτημα Β Παράμετροι χρήσης RAxML

Συνοπτική παρουσίαση των παραμέτρων εισαγωγής δεδομένων στο RAxML. Για αναλυτική περιγραφή ανατρέξτε στο manual του προγράμματος[9].

raxmlHPC[-MPI|-PTHREADS] -s sequenceFileName

-n outputFileName

-m substitutionModel

[-a weightFileName]

[-b bootstrapRandomNumberSeed]

[-c numberOfCategories]

[-d]

[-e likelihoodEpsilon]

[-E excludeFileName]

[-f a|b|c|d|e|g|h|i|j|m|n|o|p|s|t|w]

[-g groupingFileName]

[-h]

[-i initialRearrangementSetting]

[-j]

[-k]

[-l sequenceSimilarityThreshold]

6[-L sequenceSimilarityThreshold]

[-M]

[-o outGroupName1[,outGroupName2[,...]]]

[-p parsimonyRandomSeed]

[-P proteinModel]

[-q multipleModelFileName]

[-r binaryConstraintTree]

[-t userStartingTree]

[-T numberOfThreads]

[-u multiBootstrapSearches]

[-v]

[-w workingDirectory]

[-x rapidBootstrapRandomNumberSeed]

[-y]

[-z multipleTreesFile]

[-#|-N numberOfRuns]

# Βιβλιογραφία – Αναφορές


[1] N. Alachiotis, E. Sotiriades, A. Dollas, A. Stamatakis: "Exploring FPGAs for accelerating the Phylogenetic Likelihood Function". Proceedings of HICOMB 2009 (in conjunction with IPDPS 2009), accepted for publication, Rome, Italy, May 2009

[2] N. Alachiotis, E. Sotiriades, A. Dollas, A. Stamatakis: "A Reconfigurable Architecture for the Phylogenetic Likelihood Function". Proceedings of FPL 2009, accepted for publication as short paper, Prague, Czech Republic, September 2009

[3] N. Alachiotis: "Analysis, Design, and Implementation of the Phylogenetic Likelihood Function on Reconfigurable Logic", Technical University of Crete, Chania, Greece, September 2008

[4] P. Pavlidis, D. Zivkovic, A. Stamatakis, N. Alachiotis: "SweeD: Likelihood-based detection of selective sweeps in thousands of genomes'', in press,*Molecular Biology and Evolution*, 2013

[5] N. Alachiotis, A. Stamatakis: "A Vector-Like Reconfigurable Floating-Point Unit for the Logarithm". *International Journal of Reconfigurable Computing*, vol. 2011, Article ID 341510, 12 pages, 2011

[6] S.A. Berger, N. Alachiotis, A. Stamatakis: "An Optimized Reconfigurable System for Computing the Phylogenetic Likelihood on DNA Data", accepted for publication at IEEE RAW workshop

[7] N. Alachiotis: "Algorithms and Computer Architectures for Evolutionary Bioinformatics", Ph.D. thesis, Technische Universität München, Germany, November 2012

[8] J. Felsenstein. Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. Molecular Evolution, 17:368{376, 1981.

[9] Stamatakis, Alexandros. "The RAxML 7.0. 4 Manual." *Department of Computer Science. Ludwig-Maximilians-Universität München* (2008).

[10] http://www.impulseaccelerated.com/PR_Impulse_Convey_SC10.pdf visited 2013

[11] http://www.eetimes.com/electronics-products/electronic-product-reviews/fpga-pld-products/4215021/Impulse-and-Convey-enable-C-to-FPGA-for-hybrid-core-computing visited 2013

[12] http://www.impulseaccelerated.com/

[13] Dennis Pearl : “Maximum likelihood Tree Construction” Available at http://mbi.osu.edu/2005/tutorialmaterials/MBImle.pdf

[14] Συ-σχεδίαση υλικού και λογισμικού της υλοποίησης NCBI-BLAST σε υβριδική πλατφόρμα υψηλής απόδοσης (HPC) Ρουσόπουλος Κ. Χρήστος

[15] Hybrid Parallelization of the MrBayes & RAxML Phylogenetics Codes Wayne Pfeiffer (SDSC/UCSD) & Alexandros Stamatakis (TUM) February 25, 2010 available at http://sco.h-its.org/exelixis/Phylo100225.pdf

[16] http://www.conveycomputer.com/technology/partners/

[17] Jason D. Bakos, “High-Performance Heterogeneous Computing with the Convey HC-1”, 1-1-2010

[18] Convey, “Convey Personality Development Kit Reference Manual”, Version 5.2,April 2012

[19] Karl Savio Pimenta Pereira, “Characterization of FPGA-based High Performance Computers”

[20] Using Convey Hybrid Core Computers to solve Bioinformatics Problems http://www.youtube.com/watch?v=7kDaS6EQj2k

[21] Practical FPGA Programming in C , Book available at http://www.alibris.com/Practical-FPGA-Programming-in-C-David-Pellerin/book/8776226

[22] MPI Architecture http://en.wikipedia.org/wiki/Message_Passing_Interface

[23] AVX Architecture http://en.wikipedia.org/wiki/Advanced_Vector_Extensions

[24] SSE3 Architecture http://en.wikipedia.org/wiki/SSE3

[25] http://en.wikipedia.org/wiki/Homology_(biology)

[26] molecularclock hypothesis http://en.wikipedia.org/wiki/Molecular_clock

[27] http://sco.h-its.org/exelixis/Phylo100225.pdf

[28] http://www.is.titech.ac.jp/~shimo/prog/consel/

[29] http://bio.fsu.edu/~stevet/BSC5936/Wilgenbusch.2003.pdf

[30] http://www.uni-leipzig.de/~strimmer/lab/publications/journals/treesets2002.pdf

[31] http://scholarcommons.sc.edu/cgi/viewcontent.cgi?article=1078&context=csce_facpub&sei-redir=1&referer=http%3A%2F%2Fwww.google.gr%2Furl%3Fsa%3Dt%26rct%3Dj%26q%3Dconvey%2Bhc-1%2Barchitecture%26source%3Dweb%26cd%3D3%26ved%3D0CDcQFjAC%26url%3Dhttp%253A%252F%252Fscholarcommons.sc.edu%252Fcgi%252Fviewcontent.cgi%253Farticle%253D1078%2526context%253Dcsce_facpub%26ei%3DIV

_AUdOuMIToswbXroHwAQ%26usg%3DAFQjCNHyGudr6nJNJRW6yhri1A8RZFAifA#search=%22convey%20hc-1%20architecture%22

[32] http://en.wikipedia.org/wiki/Impulse_C

[33] http://en.wikipedia.org/wiki/Evolutionary_history_of_life

[34] http://en.wikipedia.org/wiki/Computational_phylogenetics

[35] Προσωπική επικοινωνία με κ. Scott Thibault , εκ τον ιδρυτών της εταιρείας.

[36] Betkaoui, Brahim, David B. Thomas, and Wayne Luk. "Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing."*Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010.