ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

«Σχεδίαση μικροεπεξεργαστή ειδικού σκοπού για εφαρμογές Βιοπληροφορικής»

Γκογκίδης Ανάργυρος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

Κακαρούντας Αθανάσιος

Λαμία, Σεπτέμβριος 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

«Σχεδίαση μικροεπεξεργαστή ειδικού σκοπού για εφαρμογές Βιοπληροφορικής»

Γκογκίδης Ανάργυρος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

Κακαρούντας Αθανάσιος

Λαμία, Σεπτέμβριος 2019

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 25/09/2019

Ο δηλών

Γκογκίδης Ανάργυρος

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

«Σχεδίαση μικροεπεξεργαστή ειδικού σκοπού για εφαρμογές Βιοπληροφορικής»

Γκογκίδης Ανάργυρος

Τριμελής Επιτροπή:

Αθανάσιος Κακαρούντας, Επίκουρος Καθηγητής (επιβλέπων)

Ιωάννης Αναγνωστόπουλος, Αναπληρωτής Καθηγητής

Αθανάσιος Λουκόπουλος, Επίκουρος Καθηγητής

Περίληψη

Την τελευταία δεκαετία, οι ανάγκες για ταχύτερη ανάλυση δεδομένων με υψηλή απόδοση αυξήθηκαν δραματικά. Ειδικά στον τομέα της Βιοπληροφορικής και της ανάλυσης Βιολογικών δεδομένων. Ένας συνηθισμένος τρόπος για να ξεπεραστεί αυτό το ζήτημα είναι η ανάπτυξη ενός Ειδικού Επεξεργαστή Οδηγιών Εφαρμογής (Application Specific Instruction Processor - ASIP) ώστε να επιταχύνει τις απαιτητικές εργασίες επεξεργασίας του αλγορίθμου.

Στην παρούσα εργασία παρουσιάζεται η διαδικασία για την επίτευξη αυτού του στόχου για τον αλγόριθμο Kalign, έναν αλγόριθμο που χρησιμοποιείται στο πεδίο της Βιοπληροφορικής, χρησιμοποιώντας τον επαναπρογραμματιζόμενο μικροεπεξεργαστή Xtensa. Τα αποτελέσματα της εργασίας, παρουσιάζουν αύξηση των επιδόσεων κατά 30%, μειώνοντας τους κύκλους εκτέλεσης και τον απαιτούμενο χρόνο, υλοποιώντας ένα νέο σύνολο εντολών ως μια νέα οδηγία για τον μικροεπεξεργαστή.

Λέξεις κλειδιά: Επαναπρογραμματιζόμενος μικροεπεξεργαστής, αρχιτεκτονική συνόλου εντολών, επεκτάσιμο σύνολο εντολών, επιτάχυνση αλγορίθμου

Abstract

In the last decade, the needs for faster and high-performance data analysis has increased dramatically. Especially in the field of Bioinformatics and biological data analysis. A common way to overcome this issue is to develop an Application Specific Instruction Processor (ASIP) to speed up the demanding algorithm processing tasks.

This work describes the process of achieving this goal for Kalign, an algorithm from the field of Bioinformatics, using the re-configurable Xtensa processor. The results show a 30% performance increase, reducing execution cycles and time required by implementing a new set of instructions as a new assembly instruction.

Key words: Re-configurable processor, instruction set architecture, extended instruction set, algorithm acceleration

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με στήριξαν όλα αυτά τα χρόνια. Αρχικά ευχαριστώ πολύ τον καθηγητή μου κ. Αθανάσιο Κακαρούντα για την εμπιστοσύνη, την καθοδήγηση και τη βοήθειά του ώστε να ολοκληρωθεί αυτή η εργασία. Ένα μεγάλο ευχαριστώ θέλω να πω στους φίλους μου και υποψήφιους Διδάκτορες, Λένα Μπούμπα και Βασίλη Τσούκα για την πολύτιμη στήριξη και βοήθειά τους για την διεκπεραίωση της παρούσας εργασίας.

Επιπλέον θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξή τους όλα αυτά τα χρόνια και για τα εφόδια που μου έδωσαν, ώστε να καταφέρω να ολοκληρώσω αυτό τον κύκλο σπουδών μου και να γίνω ένας σωστός Άνθρωπος.

Τέλος, θα ήθελα να αφιερώσω την παρούσα εργασία στον Ηλία και τον Κωνσταντίνο που πιστεύουν σε εμένα και μου δίνουν συνεχώς δύναμη για να κυνηγήσω τα όνειρά μου.

Research is formalized curiosity.
It is poking and prying with a purpose.

Zora Neale Hurston

Περιεχόμενα

Ευρετήριο Εικόνων

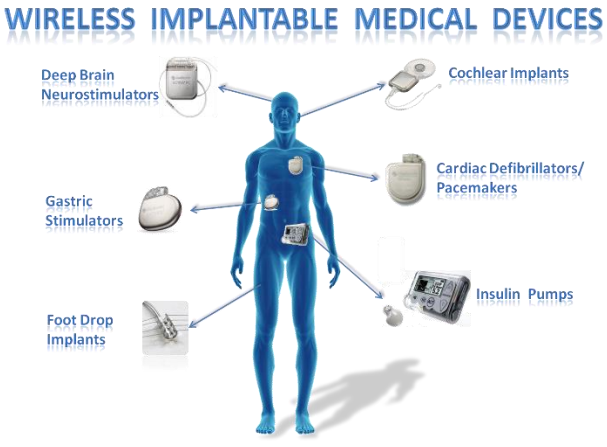
1. Εισαγωγή

1.1 Ενσωματωμένο σύστημα

Ενσωματωμένο σύστημα θεωρείται ένα σύστημα που αποτελεί συνδυασμό υλικού και λογισμικού, περιλαμβάνει μηχανικά μέρη και εκτελεί μια ειδική λειτουργία μέσα σε ένα μεγαλύτερο μηχανικό ή ηλεκτρικό σύστημα. Η εμφάνιση τους έγινε τη δεκαετία του 60. Ο πυρήνας ενός ενσωματωμένου συστήματος αποτελείται από τουλάχιστον έναν μικροεπεξεργαστή ο οποίος είναι προγραμματισμένος να τρέχει μια συγκεκριμένη εφαρμογή λογισμικού. Σήμερα μπορούμε να συναντήσουμε τα ενσωματωμένα συστήματα σε αυτοκίνητα, κινητά τηλέφωνα, MP3, κάμερες, ρομπότ, αεροπλάνα, ψηφιακή τηλεόραση και στην πλειοψηφία των οικιακών συσκευών. Αποτελούν μία ταχύτατα αναπτυσσόμενη βιομηχανία και μάλιστα τη βάση των IoT συσκευών.

Οι πρώτοι ηλεκτρονικοί υπολογιστές ήταν συστήματα τα οποία καταλάμβαναν το χώρο ενός δωματίου και απαιτούσαν μεγάλη κατανάλωση ενέργειας χωρίς να θεωρούνται συστατικά στοιχεία μεγαλύτερων ή άλλων συστημάτων. Η ραγδαία ανάπτυξη της τεχνολογίας όμως, και πιο συγκεκριμένα των ολοκληρωμένων κυκλωμάτων, προσέφερε τη δυνατότητα της ενσωμάτωσης μικρότερων υπολογιστών ως συστατικά ενός κυκλώματος, τα οποία υλοποιούν μέρος του κυκλώματος.

Οι ενσωματωμένοι υπολογιστές δεν έχουν την ίδια μορφή με τους υπολογιστές γενικού σκοπού. Ένας ενσωματωμένος υπολογιστής αποτελείται από τον πυρήνα του επεξεργαστή, τη μνήμη για την αποθήκευση των δεδομένων και τους απαραίτητους διαύλους για τη μεταφορά δεδομένων μεταξύ του πυρήνα του επεξεργαστή και του υπολοίπου συστήματος. Κάποια από τα κύρια χαρακτηριστικά τους είναι το χαμηλό κόστος παραγωγής τους, η γενικότερα συνεχής εκτέλεσή τους καθώς δεν τερματίζεται ποτέ, οι περιορισμοί οι οποίοι μπορούν να επηρεάσουν το σχεδιασμό τους, και τέλος η μη ελεγχόμενη αλληλεπίδραση τους με το περιβάλλον.



Εικόνα 1: Ενδεικτικές εφαρμογές ενσωματωμένων συστημάτων στο χώρο της υγείας.

1.2 System on Chip

Ένα System on Chip (SoC), είναι το ενσωματωμένο σύστημα που περιέχει όλα τα απαραίτητα μέρη ενός υπολογιστή ή άλλου ηλεκτρονικού συστήματος. Τα μέρη αυτά, όπως για παράδειγμα ο επεξεργαστής και η μνήμη, είναι συγκολλημένα στην μητρική πλακέτα και δεν μπορούν να αφαιρεθούν - αναβαθμιστούν όπως σε ένα υπολογιστικό σύστημα κοινού σκοπού.

Τα SoCs έρχονται με διάφορα εξαρτήματα σύμφωνα με τον προορισμό τους. Μερικά από τα συστατικά τους είναι:

Η Κεντρική Μονάδα επεξεργασίας (Core Processing Unit - CPU): Είναι ο εγκέφαλος του SoC, ο οποίος έχει την εντολή να κάνει τους υπολογισμούς και γενικά να «δένει» κάθε άλλο στοιχείο μαζί του. Οι λειτουργίες του είναι η Λήψη, η Αποκωδικοποίηση και η Εκτέλεση των εντολών. Μια CPU μπορεί να είναι ενός, δύο, τεσσάρων, ακόμα και οκτώ πυρήνων αναλόγως της επεξεργαστικής ισχύς που απαιτείται για κάθε περίπτωση χρήσης.

Η μονάδα γραφικής επεξεργασίας (Graphics Processing Unit - GPU): Είναι η κάρτα γραφικών. Σε αντίθεση με την CPU που κάνει τους υπολογισμούς, η κάρτα γραφικών είναι υπεύθυνη για την επεξεργασία των οπτικών μεταβάσεων στο περιβάλλον του χρήστη, όπως τα κινούμενα σχέδια και τα παιχνίδια 3D.

Η μνήμη τυχαίας προσπέλασης (Random Access Memory - RAM): Είναι η μνήμη του υπολογιστή, πάνω στην οποία έχουν φορτωθεί οι διαδικασίες του προγράμματος/εφαρμογής πριν μπορέσουν να χρησιμοποιηθούν. Όσο περισσότερη μνήμη RAM έχει το σύστημα, τόσες περισσότερες εφαρμογές μπορούν να εκτελούνται ταυτόχρονα χωρίς κολλήματα και καθυστερήσεις.

Η μνήμη μόνο για ανάγνωση (Read Only Memory - ROM): Είναι το μη προσβάσιμο μέρος ενός συστήματος. Στη ROM βρίσκεται το ζωτικής σημασίας σύστημα firmware και το λειτουργικό σύστημα.

Κάποια από τα δημοφιλέστερα SoCs είναι:

- Qualcomm Snapdragon
- Samsung Exynos
- Nvidia Tegra

Στο κεφάλαιο 2 παρουσιάζεται η γενική έννοια της αρχιτεκτονικής συνόλου εντολών, η διαφορά μεταξύ αρχιτεκτονικής RISC και CISC, τι είναι το επεκτάσιμο σύνολο εντολών καθώς και κάποιες από τις υπάρχουσες πλατφόρμες για την ανάπτυξη συστημάτων ειδικού σκοπού. Στο κεφάλαιο 3 γίνεται παρουσίαση και ανάλυση αλγορίθμων από τον τομέα της Βιοπληροφορικής. Στο κεφάλαιο 4 παρουσιάζεται το προτεινόμενο σύστημα και γίνεται ανάλυση της πλατφόρμας που επιλέχθηκε. Στο κεφάλαιο 5 αναλύεται η μεθοδολογία η οποία ακολουθήθηκε ώστε να γίνει επιτάχυνση

του αλγορίθμου Kalign και παρουσιάζονται κάποιες από τις βασικές εντολές της πλατφόρμας. Στο κεφάλαιο 6 παρουσιάζεται η υλοποίηση του συστήματος και αναλύονται τα αποτελέσματα που προέκυψαν καθώς και κάποιες παρατηρήσεις. Στο κεφάλαιο 7 γίνεται αναφορά στα συμπεράσματα της εκπόνησης της πτυχιακής εργασίας και παρουσιάζονται πιθανές μελλοντικές προεκτάσεις. Τέλος στο παράρτημα Α παρατίθενται όλοι οι κώδικες που αναπτύχθηκαν ή έγινε επεξεργασία.

2. Αρχιτεκτονική Συνόλου Εντολών (Instruction Set Architecture - ISA)

2.1 Εισαγωγή

Το τμήμα της αρχιτεκτονικής υπολογιστών που συνδέεται με τον προγραμματισμό και έχει ως στοιχεία του τις εντολές, τους τύπους δεδομένων, τους καταχωρητές, την αρχιτεκτονική μνήμης, τον χειρισμό διακοπών και εξαιρέσεων και την εξωτερική είσοδο και έξοδο, ονομάζεται Σύνολο Εντολών. Η αρχιτεκτονική του συνόλου εντολών, περιλαμβάνει τις εντολές που υλοποιούνται από τον ίδιο επεξεργαστή, αλλά και τις εντολές της γλώσσας μηχανής. Οι εντολές αυτές προσομοιώνονται στο λογισμικό από έναν διερμηνέα .

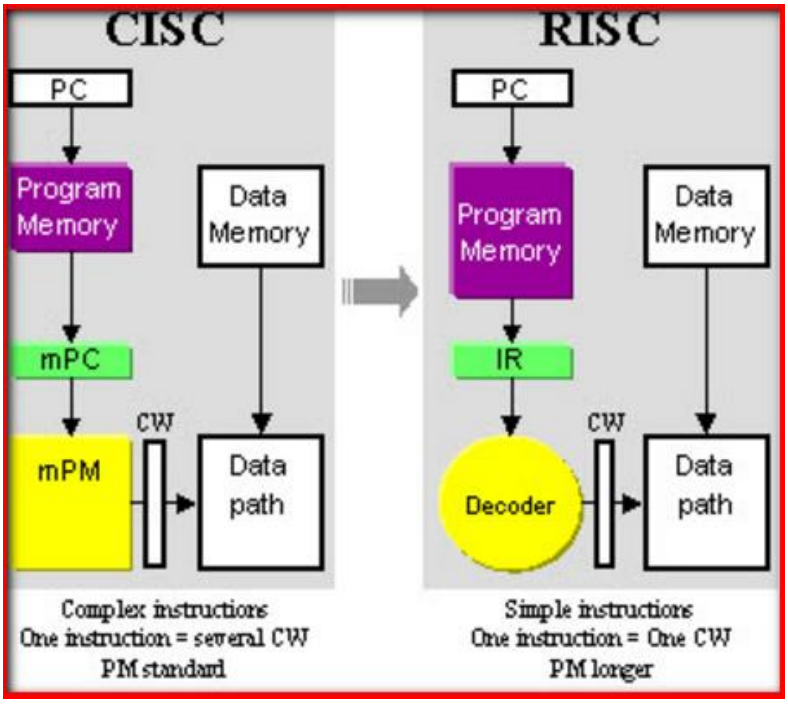
Οι βασικές αρχιτεκτονικές για τον σχεδιασμό ενός επεξεργαστή είναι δύο. Υπάρχει η αρχιτεκτονική επεξεργαστή Σύνθετου Συνόλου Εντολών (Complex Instruction Set Computing - CISC) και επεξεργαστή Μειωμένου Συνόλου Εντολών (Reduced Instruction Set Computing - RISC) [1].

2.1.1 Αρχιτεκτονική επεξεργαστή Σύνθετου Συνόλου Εντολών (Complex Instruction Set Computing – CISC)

Ένας επεξεργαστής αρχιτεκτονικής σύνθετου συνόλου εντολών (CISC), είναι ένας υπολογιστής στον οποίο μεμονωμένες εντολές μπορούν να εκτελούν πολλές λειτουργίες χαμηλού επιπέδου ή είναι ικανές να λειτουργούν σε πολλαπλά στάδια. Ορισμένες λειτουργίες είναι η ανάγνωση δεδομένων από τη μνήμη, μια αριθμητική πράξη, η αποθήκευση δεδομένων στη μνήμη κ.ά. Ο όρος δημιουργήθηκε αναδρομικά, σε αντίθεση με την αρχιτεκτονική RISC, και ως εκ τούτου έχει γίνει ένας γενικός όρος για όλα όσα δεν κάνει η RISC, από μεγάλους και σύνθετους κεντρικούς υπολογιστές, έως απλοϊκούς μικροελεγκτές, όπου η προσπέλαση στη μνήμη και οι εργασίες αποθήκευσης δεν διαχωρίζονται από τις αριθμητικές εντολές.

2.1.2 Αρχιτεκτονική επεξεργαστή Μειωμένου Συνόλου Εντολών (Reduced Instruction Set Computing – RISC)

Η αρχιτεκτονική μειωμένου συνόλου εντολών (RISC), αναπτύχθηκε στα τέλη του 1960 από την IBM, για την ανάπτυξη μικροελεγκτών οι οποίοι χρησιμοποιήθηκαν σε ηλεκτρονικές συσκευές όπως οι αριθμομηχανές, συστήματα ήχου και βιντεοπαιχνίδια της εποχής. Βασικό χαρακτηριστικό της αρχιτεκτονικής αυτής είναι ότι το σύνολο των εντολών που υποστηρίζει, καθώς είναι αρκετά μικρό και έτσι επιτρέπεται η εκτέλεση μιας εντολής σε λίγους κύκλους εκτέλεσης (Cycles Per Instruction - CPI).



Εικόνα 2: Αρχιτεκτονική CISC vs Αρχιτεκτονική RISC

2.2 Σύνολο Εντολών

Το σύνολο εντολών (Instruction Set) παρέχει στον επεξεργαστή οδηγίες σε επίπεδο μηχανής για να μπορέσουν να εκτελεστούν τα προγράμματα. Αυτό το σύνολο αποτελείται από οδηγίες για την διευθυνσιοδότηση, τους καταχωρητές, τη διαχείριση λαθών και εξαιρέσεων, καθώς και τη διαχείριση των δεδομένων προς ανάγνωση ή εγγραφή. Ορισμένα παραδείγματα εντολών είναι:

- Χειρισμός δεδομένων και λειτουργίες μνήμης
  Set → Ορίζει έναν καταχωρητή από την προσωρινή μνήμη του επεξεργαστή σε μια σταθερή τιμή.
  Read → Ανάγνωση δεδομένων.
  Write → Εγγραφή δεδομένων
  Move → Μετακίνηση δεδομένων από τη θέση μνήμης στον καταχωρητή ή αντίστροφα.

- Αριθμητική και Λογική
  Add → Πρόσθεση τιμών δύο καταχωρητών.
  Subtract →Αφαίρεση τιμών δύο καταχωρητών.
  Multiply → Πολλαπλασιασμός τιμών δύο καταχωρητών.
  Divide → Διαίρεση τιμών δύο καταχωρητών.

Compare → Σύγκριση τιμών δύο καταχωρητών.
Bitwise operations → Σύζευξη και διάζευξη μεταξύ bit σε καταχωρητές.

- Ροή ελέγχου
  Branch → Συνέχιση της εκτέλεσης των εντολών από μία θέση σε μία άλλη.
  Conditionally Branch → Συνέχιση της εκτέλεσης των εντολών από μία θέση σε μία άλλη όταν ισχύει κάποια συνθήκη.
  Indirectly branch → Συνέχιση της εκτέλεσης των εντολών από μία θέση σε μία άλλη, αποθηκεύοντας την θέση της εντολής για να αποτελέσει ένα σημείο επιστροφής .

Σύνθετες εντολές

Οι εντολές αυτές απαιτούν παραπάνω από ένα βήματα και πραγματοποιούν ελέγχους σε πολλαπλές λειτουργικές μονάδες. Παραδείγματα τέτοιων εντολών είναι:

- η ταυτόχρονη αποθήκευση πολλών καταχωρητών στη στοίβα
- η μετακίνηση μεγάλων τμημάτων μνήμης
- η εκτέλεση της εντολής test-and-set

INSTRUCTION	INSTRUCTION FORMAT				
	15 – 12	11 - 8	7 - 4	3 - 0	SEMANTICS
ADD Rt, Rs1,Rs2	0	R target	R source1	R source2	Rt ← Rs1 + Rs2 ; Inz ; Icv
SUB Rt, Rs1,Rs2	1	R target	R source1	R source2	Rt ← Rs1 - Rs2 ; Inz ; Icv
AND Rt, Rs1,Rs2	2	R target	R source1	R source2	Rt ← Rs1 *and* Rs2 ; Inz
OR Rt, Rs1,Rs2	3	R target	R source1	R source2	Rt ← Rs1 *or* Rs2 ; Inz
XOR Rt, Rs1,Rs2	4	R target	R source1	R source2	Rt ← Rs1 *xor* Rs2 ; Inz
ADDI Rt, *cte8*	5	R target	Constant		Rt ← Rt + ("00000000" & constant) ; Inz ; Icv
SUBI Rt, *cte8*	6	R target	Constant		Rt ← Rt - ("00000000" & constant) ; Inz ; Icv
LDL Rt, *cte8*	7	R target	Constant		Rt ← RtH & constant
LDH Rt, *cte8*	8	R target	Constant		Rt ← constant & RtL
LD Rt, Rs1,Rs2	9	R target	R source1	R source2	Rt ← MEMP (Rs1+Rs2)

Εικόνα 3: Υπόδειγμα σετ εντολών

2.3 Επέκταση Συνόλου Εντολών

Τα τελευταία χρόνια, με τους γρήγορους ρυθμούς ανάπτυξης της τεχνολογίας και της έρευνας, έχει δημιουργηθεί η ανάγκη για τη σχεδίαση και την υλοποίηση καλύτερων υπολογιστικών συστημάτων. Αυτό έχει ωθήσει αρκετές ομάδες και εταιρείες, στη εύρεση νέων τεχνικών, ώστε να μειωθεί κι άλλο ο χρόνος απόκρισης των συστημάτων, η κατανάλωση, το μέγεθος αλλά και το κόστος τους.
Μια τέτοια τεχνική είναι η επέκταση του συνόλου εντολών ενός επεξεργαστή. Με αυτό τον τρόπο, είναι εφικτό να σχεδιαστεί ένας επεξεργαστής ειδικού σκοπού για να καλύψει τις ανάγκες μιας συγκεκριμένης εφαρμογής, με αποτέλεσμα την μείωση του χρόνου απόκρισης και της επεξεργασίας των δεδομένων, με συνέπεια την μείωση της

κατανάλωσης ενέργειας. Προφέρεται η δυνατότητα να δημιουργηθούν καινούριες εντολές για τον επεξεργαστή, και στη συνέχεια να μεταβληθεί ελάχιστα ο κώδικας που επιθυμούμε να βελτιστοποιηθεί, ώστε να χρησιμοποιήσει τις νέες εντολές. Για παράδειγμα, αν έχουμε μια πρόσθεση 3 ακεραίων αριθμών, για κάθε αριθμό χρειάζονται Ν κύκλοι εκτέλεσης. Με την επέκταση μπορούμε να μειώσουμε τους κύκλους εκτέλεσης γιατί πλέον η μετάφραση σε γλώσσα assembly θα χρησιμοποιεί την νέα εντολή που πραγματοποιεί την πρόσθεση των 3 ακεραίων αριθμών με το ελάχιστο δυνατό κόστος.

2.4 Πλατφόρμες και Εφαρμογές

Το Νοέμβριο του 1971 η Intel εμφάνισε τον πρώτο εμπορικό επεξεργαστή (Intel 4004). Από τότε και μέχρι σήμερα, σχεδόν όλοι οι επεξεργαστές που υπάρχουν έχουν σταθερή αρχιτεκτονική συνόλου εντολών. Αυτό βοηθάει στην εμπορικότητα και τη γενική συμβατότητα των εφαρμογών, όμως δυσκολεύει την ανάπτυξη εφαρμογών που έχουν απαιτήσεις από πολύ συγκεκριμένες λειτουργίες του επεξεργαστή.

Έτσι αναπτύχθηκαν διάφορες λύσεις για να αντιμετωπιστεί αυτό το πρόβλημα και οι σχεδιαστές υλικού πλέον έχουν τη δυνατότητα να αναπτύξουν επεξεργαστές, βελτιστοποιημένους για τις ανάγκες της εκάστοτε εφαρμογής, με απώτερο σκοπό την μείωση της κατανάλωσης ενέργειας και την αύξηση της απόδοσης.

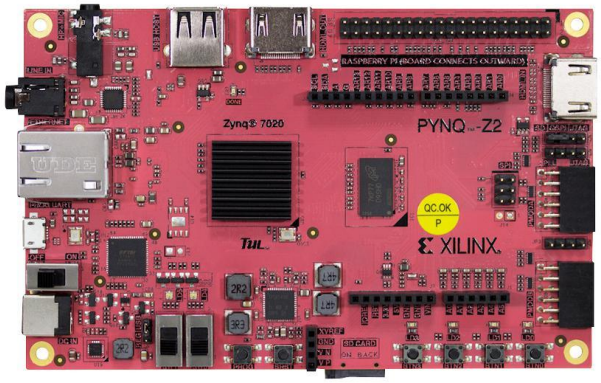
2.4.1 Field Programmable Gate Array (FPGA) Designs

Η πλατφόρμα FPGA είναι μια συσκευή η οποία έχει ευρεία χρήση στο χώρο των ενσωματωμένων συστημάτων και σε εφαρμογές που απαιτούν εξειδικευμένο υπολογιστικό σύστημα. Βασικό χαρακτηριστικό του FPGA είναι ότι διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης κ.ά. Δεν διαθέτει έτοιμο επεξεργαστής για να εκτελεστεί το λογισμικό, αλλά ο χρήστης είναι υπεύθυνος να σχεδιάσει το κύκλωμα. Θα μπορούσε να χρησιμοποιηθεί σαν μια απλή πύλη AND, ή ακόμα και σαν έναν πολυπύρηνο επεξεργαστή.

Επιπλέον, ένα πολύ ιδιαίτερο χαρακτηριστικό της συγκεκριμένης πλατφόρμας είναι ο επαναπρογραμματισμός λόγω λάθους ή αναβάθμισης. Επίσης, η ανάπτυξη και η σχεδίαση του κυκλώματος, μπορεί να γίνει πολύ εύκολα από κάποια γλώσσα περιγραφής υλικού όπως η VHDL ή η VERILOG [2] [3].

Ενδεικτικές εφαρμογές που έχουν χρησιμοποιηθεί τα FPGA συστήματα είναι στο χώρο της αεροναυπηγικής και άμυνας, στην αυτοκινητοβιομηχανία με τα συστήματα αυτόματου πιλότου, σε Data Center και σε υπολογιστές υψηλής απόδοσης (High Performance Computing – HPC). Κατά καιρούς έχουν υλοποιηθεί εφαρμογές στο χώρο

της Ιατρικής-Βιοϊατρικής με εφαρμογές οι οποίες πετυχαίνουν επιτάχυνση στην ανάλυση των δεδομένων [4].



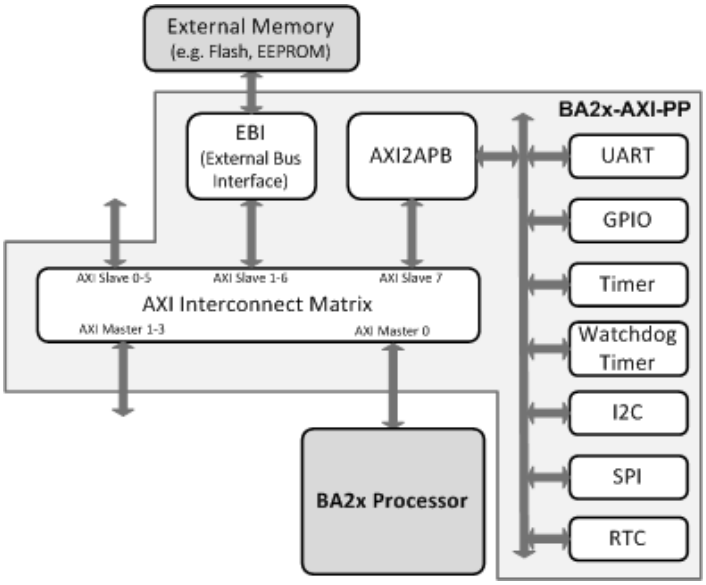
Εικόνα 4: XILINX FPGA PYNQ-Z2

2.4.2 CAST BA2x IP Core

Μια άλλη λύση για την σχεδίαση και την ανάπτυξη επεξεργαστή ειδικού σκοπού είναι η οικογένεια IP Core BA2x από την CAST [5] [6]. Ιδιαίτερο χαρακτηριστικό της είναι η αρχιτεκτονική PipelineZero, η οποία της επιτρέπει να μην υπάρχει καθυστέρηση μεταξύ των τμημάτων κώδικα προς εκτέλεση. Έτσι η συγκεκριμένη οικογένεια καταφέρνει να:

- Μειώσει τους Δομικούς Κινδύνους (Structural Hazards), τους Κινδύνους Ελέγχου (Control Hazards) και τους Κινδύνους Δεδομένων (Data Hazards) με αποτέλεσμα την υψηλή απόδοση,
- Έχει λιγότερους καταχωρητές pipeline (pipeline registers) και χρειάζεται ένα μικρό πλήθος από flip-flops με αποτέλεσμα να υπάρχει ένας επεξεργαστής με μικρή επιφάνεια,
- Έχει μικρό αποτύπωμα κατανάλωσης ενέργειας.

Επιπλέον, οι σχεδιαστές μπορούν να αυξήσουν τις δυνατότητες του επεξεργαστή χρησιμοποιώντας πολλαπλασιαστές και διαιρέτες σε επίπεδο υλικού, πολλαπλασιασμό πολλών μπλοκ μαζί και μπορούν να χρησιμοποιήσουν το πρότυπο IEEE-754 για μονάδες κινητής υποδιαστολής.



Εικόνα 5: Σχεδιάγραμμα της οικογένειας επεξεργαστών BA2x, που απεικονίζει όλους τους διαθέσιμους διαύλους επικοινωνίας και όλες τις περιφερειακές μονάδες επεξεργασίας.

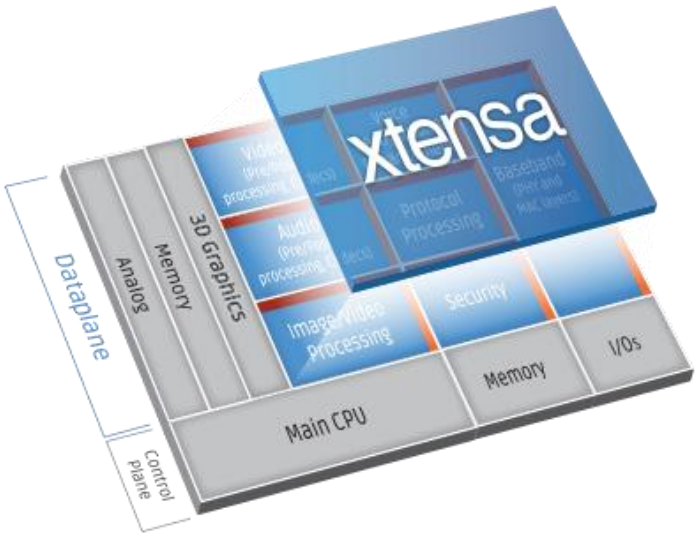
2.4.3 Tensilica - Cadence

Η Tensilica ιδρύθηκε το 1997 και δραστηριοποιήθηκε στο χώρο των επαναπρογραμματιζόμενων μονάδων (κυκλώματα ή λογικές μονάδες, στα οποία μπορούμε να μεταβάλουμε τη λειτουργία τους). Στον χώρο της ηλεκτρονικής σχεδίασης αυτό είναι γνωστό ως IP Core. Έτσι, αναπτύχθηκαν παραμετροποιήσιμοι μικροεπεξεργαστές, με την ιδιαιτερότητα ότι είναι και επεκτάσιμοι [7]. Με αυτό τον τρόπο, μπορούμε να προσαρμόσουμε τον επεξεργαστή στις ανάγκες της εκάστοτε εφαρμογής και ταυτόχρονα μπορούμε να παραμετροποιήσουμε τα στοιχεία του ή να επιλέξουμε από έτοιμα σχέδια. Παράλληλα, μπορούμε να επεκτείνουμε το σύνολο εντολών του, με τη χρήση της γλώσσας TIE (Tensilica Instruction Extension) [8].

Αρκετές από τις υπόλοιπες λύσεις για την σχεδίαση και την κατασκευή επεξεργαστή ειδικού σκοπού, απαιτούν τη χρήση γλώσσας περιγραφής υλικού με κώδικα RTL (Register Transfer Level), όπως VHDL ή Verilog. Από την άλλη πλευρά, πολύ σημαντική διαφορά της λύσης που προτείνει η Tensilica, είναι το χαμηλό κόστος σχεδίασης και ανάπτυξης του επεξεργαστή, καθώς είναι προσανατολισμένη στο σχεδιασμό μέσω του επεξεργαστή, και όχι μέσω RTL κώδικα. Αυτό σημαίνει ότι κάθε φορά σχεδιάζεται ο επεξεργαστής για τις ανάγκες τις εκάστοτε εφαρμογής, με αποτέλεσμα η σχεδίαση να γίνεται πιο γρήγορα και με περισσότερη λεπτομέρεια στις εξειδικευμένες λειτουργίες που απαιτούνται.

Αυτή η διαδικασία οδηγεί τον χρήστη που αναπτύσσει αυτό το σύστημα, να υλοποιήσει έναν μικροεπεξεργαστή με μειωμένη επιφάνεια, καθώς πλέον χρειάζονται λιγότερες

πύλες για να υλοποιηθεί μια εντολή, έχοντας ως αποτέλεσμα την μείωση της κατανάλωσης ρεύματος που απαιτείται.



Εικόνα 6: Μοντέλο επεκτάσιμου επεξεργαστή Xtensa.

Ο λόγος ύπαρξης και ανάπτυξης τέτοιου είδους τεχνολογιών, όπως προαναφέρθηκε, είναι για τη σχεδίαση και τη δημιουργία βέλτιστων συστημάτων ή συστημάτων τα οποία θα αποδώσουν το μέγιστο δυνατό αποτέλεσμα, με το ελάχιστο δυνατό κόστος, είτε αναφερόμαστε στο κόστος κατανάλωσης, είτε στο κόστος ανάπτυξης.

3. Αλγόριθμοι Βιοπληροφορικής - Βιοϊατρικής

3.1 Εισαγωγή

Στη σημερινή εποχή, οι ανάγκες για παρακολούθηση της υγείας ή η πρόβλεψη ασθενειών, έχουν ωθήσει στην ανάπτυξη συστημάτων για εξατομικευμένη χρήση. Τέτοιου είδους συστήματα μπορεί να είναι είτε φορετά, είτε όχι. Όλα αυτά όμως απαιτούν κάτι σημαντικό, την σύμπραξη της Βιολογίας με την Πληροφορική. Έτσι μπορούμε να συζητάμε για την έννοια της Βιοπληροφορικής, όπου είναι ο κλάδος που συνδυάζει τη Βιολογία, τα εφαρμοσμένα Μαθηματικά και τεχνικές από τον χώρο της επιστήμης των υπολογιστών (Πληροφορικής) [9] [10].

Μία από τις θεμελιώδεις μεθόδους που χρησιμοποιείται στην Βιοπληροφορική είναι η Πολλαπλή Στοίχιση Ακολουθιών (Multiple Sequence Alignment - MSA). Η MSA είναι ένας τρόπος οργάνωσης των οι αλληλουχιών του DNA ή των πρωτεϊνών, ώστε να εντοπιστούν οι περιοχές ομοιότητας που μπορεί να είναι συνέπεια λειτουργικών, δομικών ή εξελικτικών σχέσεων μεταξύ των ακολουθιών.

3.2 Αλγόριθμοι

Ο σκοπός ενός αλγορίθμου MSA είναι να συγκεντρώσει τις ευθυγραμμίσεις που αντικατοπτρίζουν τη βιολογική σχέση μεταξύ αρκετών ακολουθιών. Ο υπολογισμός των MSAs με μεγάλη ακρίβεια είναι σχεδόν αδύνατος υπολογιστικά και στην πράξη χρησιμοποιούνται προσεγγιστικοί αλγόριθμοι (heuristics) για την ευθυγράμμιση των αλληλουχιών, μεγιστοποιώντας την ομοιότητά τους. Η βιολογική συνάφεια αυτών των MSAs συνήθως αξιολογείται με συστηματική σύγκριση με τις καθιερωμένες συλλογές MSA με βάση τη δομή τους. Δεδομένου ότι μόνο μερικές ακολουθίες έχουν γνωστές δομές, η ακρίβεια που μετράται στις αναφορές είναι απλώς μια εκτίμηση για το πόσο καλά μπορεί να χρησιμοποιηθεί ένα πακέτο σε τυποποιημένα σύνολα δεδομένων. Έτσι, δημιουργήθηκαν κάποια πρότυπα (Gold Standards) τα οποία είχαν σημαντική επίδραση στην εξέλιξη των αλγορίθμων MSAs, επαναπροσδιορίζοντας την όλη μεθοδολογική εξέλιξη προς την ανάπτυξη ορθών ευθυγραμμίσεων [11].

Αυτός ο άπληστος αλγόριθμος ευρετικής συναρμολόγησης περιλαμβάνει την εκτίμηση ενός δυαδικού δέντρου οδηγού (root binary tree) από μη ευθυγραμμισμένες ακολουθίες και στη συνέχεια την ενσωμάτωση των ακολουθιών στο MSA με έναν αλγόριθμο ευθυγράμμισης κατά ζεύγη ακολουθώντας την τοπολογία του δέντρου. Ο προοδευτικός αλγόριθμος είναι συχνά ενσωματωμένος σε έναν επαναληπτικό βρόγχο όπου το δέντρο οδηγός και το MSA επαναξιολογούνται μέχρι τη σύγκλιση.

Παρακάτω παρουσιάζονται κάποιοι γνωστοί αλγόριθμοι που χρησιμοποιούνται για MSAs.

3.2.1 Αλγόριθμος Clustal

Η οικογένεια αλγορίθμων Clustal [12], είναι μια σειρά από ευρέως χρησιμοποιούμενα προγράμματα που χρησιμοποιούνται στη Βιοπληροφορική για την ευθυγράμμιση πολλών ακολουθιών (MSA). Οι τελευταίες δύο εκδόσεις του αλγορίθμου είναι οι ακόλουθες:

1) **Clustal Omega (ClustalΩ)**

   Είναι η τελευταία σταθερή έκδοση του αλγορίθμου Clustal. Είναι ένας αλγόριθμος γραμμένος σε γλώσσα προγραμματισμού C και C++, που χρησιμοποιείται για MSA. Βασικό χαρακτηριστικό του είναι η χαμηλότερη πολυπλοκότητα που έχει O(NlogN), σε αντίθεση με προηγούμενες εκδόσεις $O(N^2)$, όπου N είναι ο αριθμός των ακολουθιών προς στοίχιση. Αυτό έγινε εφικτό καθώς υλοποιήθηκε μια καινούργια μέθοδος (mBed) η οποία επιτρέπει τη δημιουργία εκατοντάδων χιλιάδων καθοδηγούμενων δένδρων (guide trees) από τις ακολουθίες, καταφέρνοντας να περιορίσει τον χρόνο υπολογισμού των ακολουθιών σε O(NlogN) πολυπλοκότητα. Επίσης, έχει γίνει και η χρήση μιας βιβλιοθήκης η οποία υλοποιεί Hidden Markov Model (HMM) για την διαδικασία της στοίχισης, έχοντας ως αποτέλεσμα την αύξηση της απόδοσης και της ακρίβειας του αλγόριθμου [13].

2) **ClustalW / ClustalX**

   Σε αυτή την έκδοση υπάρχουν δυο υλοποιήσεις, όπου η διαφορά τους είναι ότι ο αλγόριθμος ClustalX έχει γραφικό περιβάλλον (GUI), σε αντίθεση με τον αλγόριθμο ClustalW που υποστηρίζει τη γραμμή εντολών (command line) [14]. Και οι δύο αλγόριθμοι χρησιμοποιούν μεθόδους προοδευτικής ευθυγράμμισης, οι οποίες ευθυγραμμίζουν πρώτα τις ακολουθίες που μοιάζουν περισσότερο μεταξύ τους, και στη συνέχεια αυτές με τις λιγότερο παρόμοιες ακολουθίες, έως ότου δημιουργηθεί μια καθολική ευθυγράμμιση. Η συγκεκριμένη έκδοση έχει χρονική πολυπλοκότητα $O(N^2)$, λόγω της μεθόδου «γειτονικής σύζευξης» όπου δημιουργείται ένα δένδρο οδηγός.

3.2.2 Αλγόριθμος FAMSA

Ο αλγόριθμος FAMSA είναι ένας σύγχρονος αλγόριθμος, ο οποίος καταφέρνει να κάνει χρήση πολυπύρηνων συστημάτων και έχει τη δυνατότητα να χρησιμοποιήσει και τους πυρήνες που παρέχουν οι κάρτες γραφικών (Cuda cores). Έχει τη δυνατότητα να επεξεργαστεί χιλιάδες ακολουθίες πρωτεϊνών και να επιτύχει υψηλή ακρίβεια. Στα χαρακτηριστικά του συμπεριλαμβάνονται η χρήση μιας μεθόδου για τη σύγκριση όμοιων ζευγών από την μακρύτερη κοινή ακολουθία, μια καινούργια μέθοδο για τον υπολογισμό των κενών, καθώς και ένα καινούργιο σχήμα επαναληπτικής μεθόδου. Η

υλοποίηση του συγκεκριμένου αλγορίθμου έχει γίνει με τέτοιο τρόπο ώστε να αξιοποιήσει στο έπακρο τις σύγχρονες πλατφόρμες και την παράλληλη επεξεργασία δεδομένων, επιτρέποντάς του να κάνει ανάλυση και επεξεργασία εκατοντάδων χιλιάδων ακολουθιών σε μικρό χρονικό διάστημα και να επιτυγχάνει υψηλή απόδοση και ακρίβεια [15].

3.2.3 Αλγόριθμος Kalign

Ο αλγόριθμος Kalign [16] χρησιμοποιεί μια μέθοδο για την αντιστοίχιση αλφαριθμητικών ακολουθιών αξιοποιώντας την τεχνική του αλγορίθμου Wu-Manber [17]. Με αυτό τον τρόπο, επιτυγχάνει μεγάλη ακρίβεια αλλά και ταχύτητα στην διαδικασία της πολλαπλής στοίχισης ακολουθιών. Ένα ιδιαίτερο χαρακτηριστικό του είναι ότι για μικρές ακολουθίες επιτυγχάνει υψηλή ακρίβεια, όμως για αρκετά μεγάλες ακολουθίες, με χαμηλό ποσοστό ομοιότητας, επιτυγχάνει ακόμα υψηλότερη ακρίβεια, σε αντίθεση με άλλους αλγορίθμους. Η χρήση του προτείνεται για μεγάλο πλήθος, μεγάλων ακολουθιών.

Ο συγκεκριμένος αλγόριθμος ακολουθεί μια στρατηγική αντίστοιχη με την τυπική προοδευτική μέθοδο ευθυγράμμισης αλληλουχιών. Υπολογίζει τα ζεύγη αποστάσεων και κατασκευάζει ένα δέντρο οδηγό, όπου οι ακολουθίες είναι στοιχισμένες με την σειρά που θα δώσει το δέντρο. Όπως προαναφέρθηκε, σε αντίθεση με άλλες μεθόδους, χρησιμοποιεί τον αλγόριθμο Wu-Manber για τον υπολογισμό της απόστασης. Στην τελευταία έκδοση του συγκεκριμένου αλγόριθμου, υποστηρίζεται η στοίχιση των νουκλεοτιδίων και επιτρέπεται η σήμανση των ακολουθιών.

Στην παρούσα πτυχιακή εργασία, επιλέχθηκε ο αλγόριθμος Kalign 1.4 [18], λόγω της άμεσης συμβατότητας με το λειτουργικό Windows 10 και τον compiler GCC [19], τον οποίο υποστηρίζει και ο Xtensa Xplorer που θα παρουσιαστεί στη συνέχεια.

4. Προτεινόμενο Σύστημα

4.1 Εισαγωγή

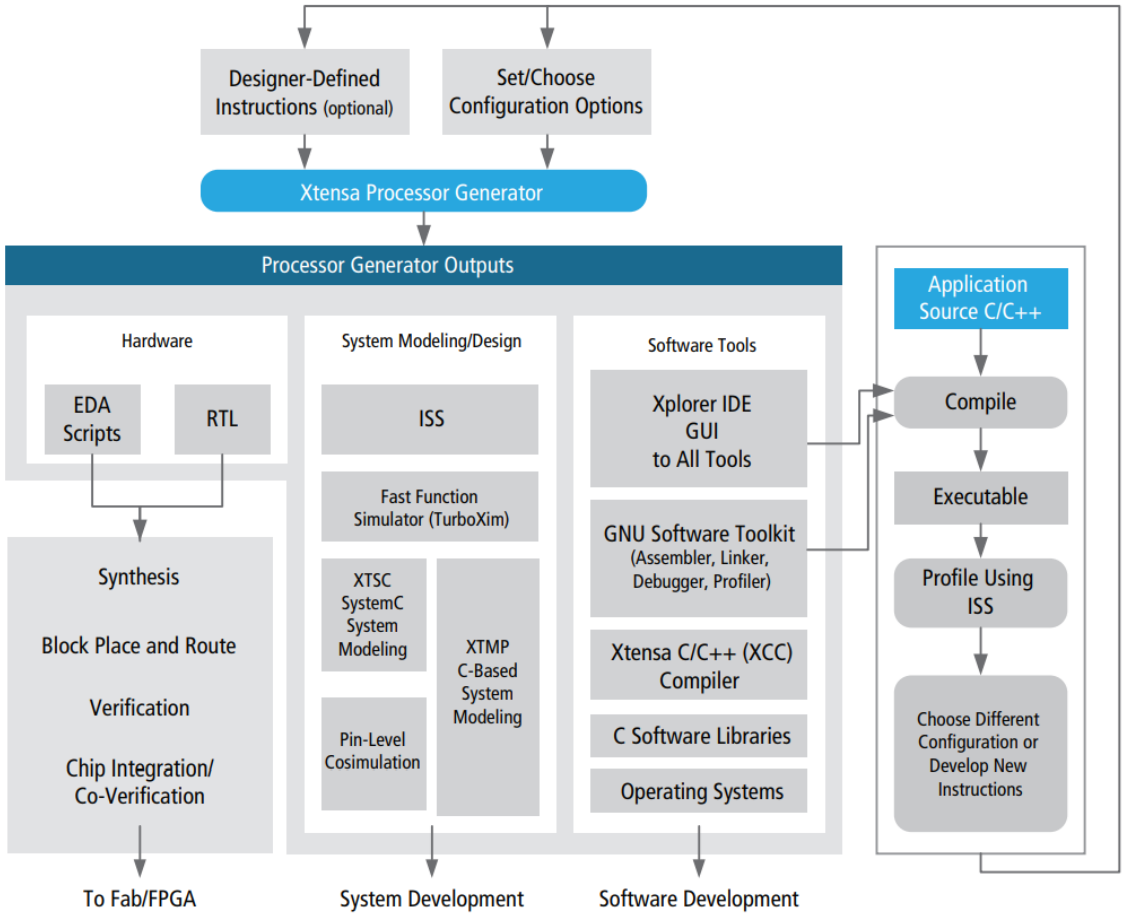
Η σημερινή εποχή έχει ανάγκη από επεξεργασία και ανάλυση μεγάλου όγκου δεδομένων. Ειδικά στο χώρο της Βιοϊατρικής οι αλγόριθμοι, οι οποίοι αναλύθηκαν στο προηγούμενο κεφάλαιο, καλούνται να επεξεργαστούν εκατοντάδες χιλιάδες ακολουθίες, με μεγάλη ανάγκη στην παραγωγή των αποτελεσμάτων όσο πιο άμεσα γίνεται. Έτσι, όπως παρουσιάστηκε και στο Κεφάλαιο 2, αναπτύχθηκαν διάφορα συστήματα για την βελτιστοποίηση και την επιτάχυνση της εκτέλεσης τέτοιων αλγορίθμων.

Στην παρούσα πτυχιακή εργασία επιλέχθηκε να χρησιμοποιηθεί το πρόγραμμα Xtensa Xplorer από την εταιρία Cadence [20]. Επιθυμητός στόχος είναι η ανάπτυξη ενός επεκτάσιμου συνόλου εντολών, ώστε να δημιουργηθεί ένας μικροεπεξεργαστής για εφαρμογές της Βιοπληροφορικής που πραγματοποιεί ταχύτερη ανάλυση δεδομένων και άμεση παραγωγή αποτελεσμάτων.

4.2 Το πρόγραμμα Xtensa Xplorer

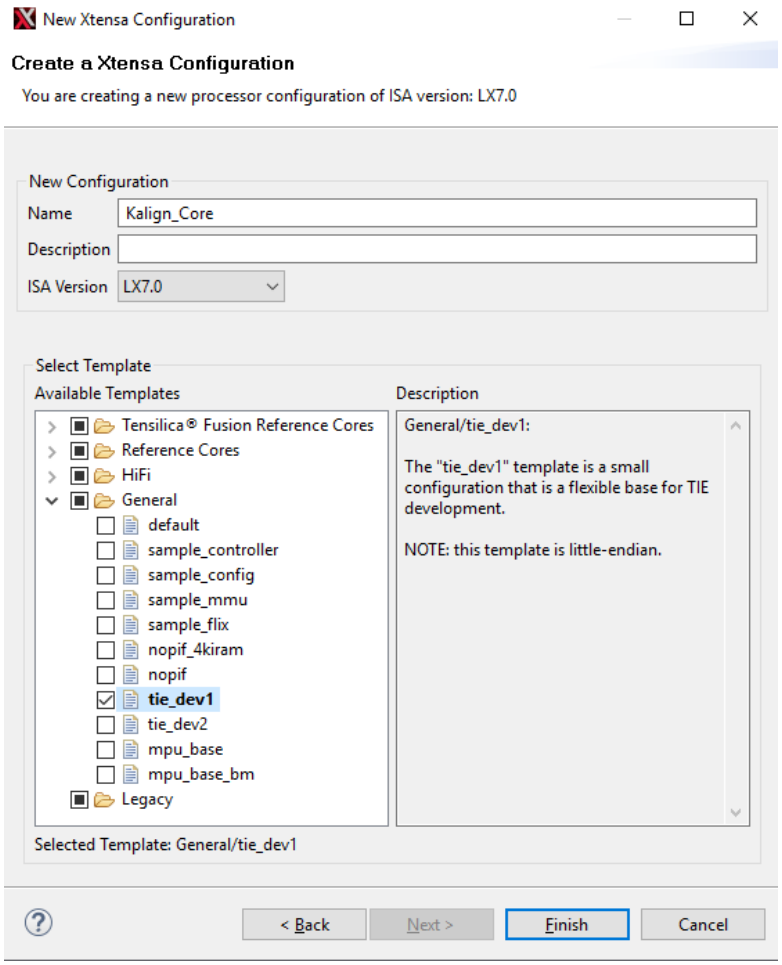
Η εταιρεία Tensilica, με την χρήση της γλώσσας TIE, κατάφερε να βελτιστοποιήσει την ταχύτητα αλλά και την απόδοση ενός επεξεργαστή Xtensa. Αυτή η γλώσσα, η οποία μοιάζει αρκετά με την γλώσσα Verilog, επιτρέπει στον χρήστη να προσθέσει καινούργιες λειτουργίες στον επεξεργαστή, με αρκετά απλό κώδικα. Με λίγες μόνο εντολές, μπορούν να αυξηθούν αρκετά οι επιδόσεις του επεξεργαστή για μια συγκεκριμένη εφαρμογή. Αυτό έχει ως αποτέλεσμα, η τελική υλοποίηση να είναι γρηγορότερη από έναν συμβατικό (οικιακό) επεξεργαστή, να καταλαμβάνει λιγότερο εμβαδόν, και κατά συνέπεια με την επίτευξη της μείωσης του χρόνου εκτέλεσης - λειτουργίας του επεξεργαστή, μειώνεται και το κόστος κατανάλωσης.

Για να καταστεί αυτό εφικτό, η Tensilica δημιουργήθηκε ένα IDE (Integrated Development Environment) βασισμένο στο πρόγραμμα Eclipse [21], που ονομάζεται Xtensa Xplorer. Μέσα από αυτό το περιβάλλον, δίνεται η δυνατότητα να αναπτυχθεί εφαρμογή σε γλώσσα προγραμματισμού C και C++. Επίσης, με τον compiler της γλώσσας TIE μπορούν υλοποιηθούν οι κατάλληλες επεκτάσεις στο σύνολο εντολών (instruction set) που χρειάζονται, ώστε το πρόγραμμα να εκτελείτε όσο το δυνατόν καλύτερα.



Εικόνα 7: Διαδικασία ανάπτυξης μικροεπεξεργαστή με τη χρήση του Xtensa Xplorer

Για να ξεκινήσει η διαδικασία, με τη χρήση του Xtensa Xplorer, πρέπει να δημιουργηθεί η αρχική έκδοση του επεξεργαστή. Μέσα από το εργαλείο αυτό, υπάρχει η δυνατότητα επιλογής ρυθμίσεων από έτοιμα IP Cores που υπάρχουν ήδη υλοποιημένα από άλλες ερευνητικές ομάδες ή εταιρείες καθώς η όλη πλατφόρμα είναι βασισμένη στην τεχνολογία του Νέφους (Cloud). Όπως απεικονίζεται και στην παρακάτω εικόνα, επιλογή ενός τυπικού TIE Core πραγματοποιείται ώστε να σχεδιαστεί και να αναπτυχθεί από την αρχή το σύστημα για τις ανάγκες της εφαρμογής.



Εικόνα 8: Επιλογή ενός τυπικού core, για τη σχεδίαση και την ανάπτυξη εξειδικευμένης λύσης

Κάποιες από τις έτοιμες επιλογές IP Core που υπάρχουν, αφορούν κατά κύριο λόγο εφαρμογές που επεξεργάζονται σήματα όπως εικόνες και βίντεο. Στη συνέχεια, θα πρέπει να δημιουργηθεί το profile του αλγορίθμου, διαδικασία η οποία αναλύεται στην Ενότητα 5.1, ώστε να εντοπιστούν τα τμήματα του κώδικα τα οποία θα μπορούσαν να υλοποιηθούν σε επίπεδο υλικού με τη χρήση της γλώσσας TIE.

Ένα απλό παράδειγμα είναι ο ακόλουθος κώδικας σε γλώσσα προγραμματισμού C.

```
int main( int argc, char **argv )
{
      int x,y;
      x=5;
      y=3;
      x = x + y;
}
```

Ο παραπάνω κώδικας, έχει δύο μεταβλητές (x και y) και εκτελεί μια απλή πρόσθεση. Με τη διαδικασία του profiling, θα παραχθεί το ακόλουθο τμήμα γλώσσας προγραμματισμού Assembly.

Count	Address	Instruction
		main
		int main(int argc, char **argv)
3	60000ad8	entry a1, 64
1	60000adb	s32i.n a2, a1, 16
1	60000add	s32i.n a3, a1, 20
		int x,y;
		x=5;
1	60000adf	movi.n a8, 5
1	60000ae1	s32i.n a8, a1, 0
		y=3;
1	60000ae3	movi.n a7, 3
1	60000ae5	s32i.n a7, a1, 4
		x = x + y;
1	60000ae7	l32i.n a6, a1, 4
1	60000ae9	l32i.n a5, a1, 0
2	60000aeb	add.n a5, a5, a6
1	60000aed	s32i.n a5, a1, 0
		int x,y;
		x=5;
		y=3;
		x = x + y;
		}
1	60000aef	retw.n

Εικόνα 9: Κώδικας Assembly από το profiling μέσω του Xtensa Xplorer.

Όπως θα παρουσιαστεί στη συνέχεια, αφού πλέον ο χρήστης γνωρίζει και τον κώδικα σε γλώσσα προγραμματισμού Assembly, μπορεί να αναπτύξει τις δικιές του εντολές. Για παράδειγμα, θα μπορούσε να δημιουργηθεί η ακόλουθη εντολή, σε γλώσσα TIE.

```
operation ADDACC {inout AR a, in AR b} {}
{
    assign a = a + b;
}
```

Στη συνέχεια, μπορεί να μετασχηματιστεί ο κώδικας σε γλώσσα προγραμματισμού C, και να πραγματοποιηθεί η χρήση αυτής της νέας εντολής. Για να γίνει αυτό, θα πρέπει αρχικά να γίνει η εισαγωγή της βιβλιοθήκης που δημιουργήθηκε στον Xtensa Xplorer και στη συνέχεια να μεταβληθεί ο κώδικας.

```
#include <xtensa/tie/TestTie.h>
int main( int argc, char **argv )
{
	int x,y;
	x=5;
	y=3;
	ADDACC(x,y);

}
```

Πλέον, η πράξη της πρόσθεσης γίνεται με τη χρήση της νέας εντολής `ADDACC()`. Πραγματοποιώντας ξανά την διαδικασία του profiling, ο νέος κώδικας σε γλώσσα Assembly που παράγεται είναι ο ακόλουθος:

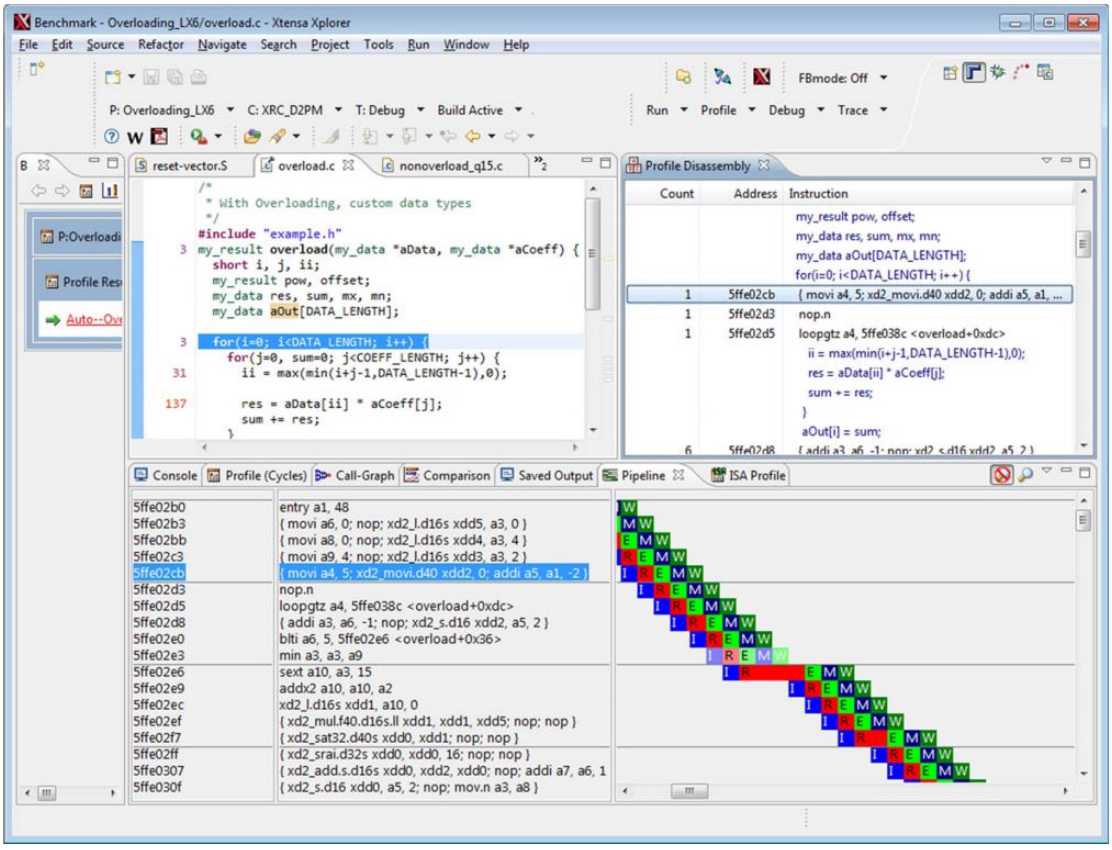
```
                        int main( int argc, char **argv )
3       60000ad8          entry a1, 64
1       60000adb          s32i.n a2, a1, 16
1       60000add          s32i.n a3, a1, 20
                           int x,y;
                           x=5;
1       60000adf          movi.n a8, 5
1       60000ae1          s32i.n a8, a1, 0
                           y=3;
1       60000ae3          movi.n a7, 3
1       60000ae5          s32i.n a7, a1, 4
                           ADDACC(x,y);
1       60000ae7          l32i.n a5, a1, 0
1       60000ae9          l32i.n a6, a1, 4
2       60000aeb          addacc a5, a6
1       60000aee          s32i.n a5, a1, 0
                           int x,y;
                           x=5;
                           y=3;
                           ADDACC(x,y);
                        }
1       60000af0          retw.n
```

Εικόνα 10: Κώδικας Assembly, με τη χρήση νέας εντολής ADDACC()

5. Μεθοδολογία

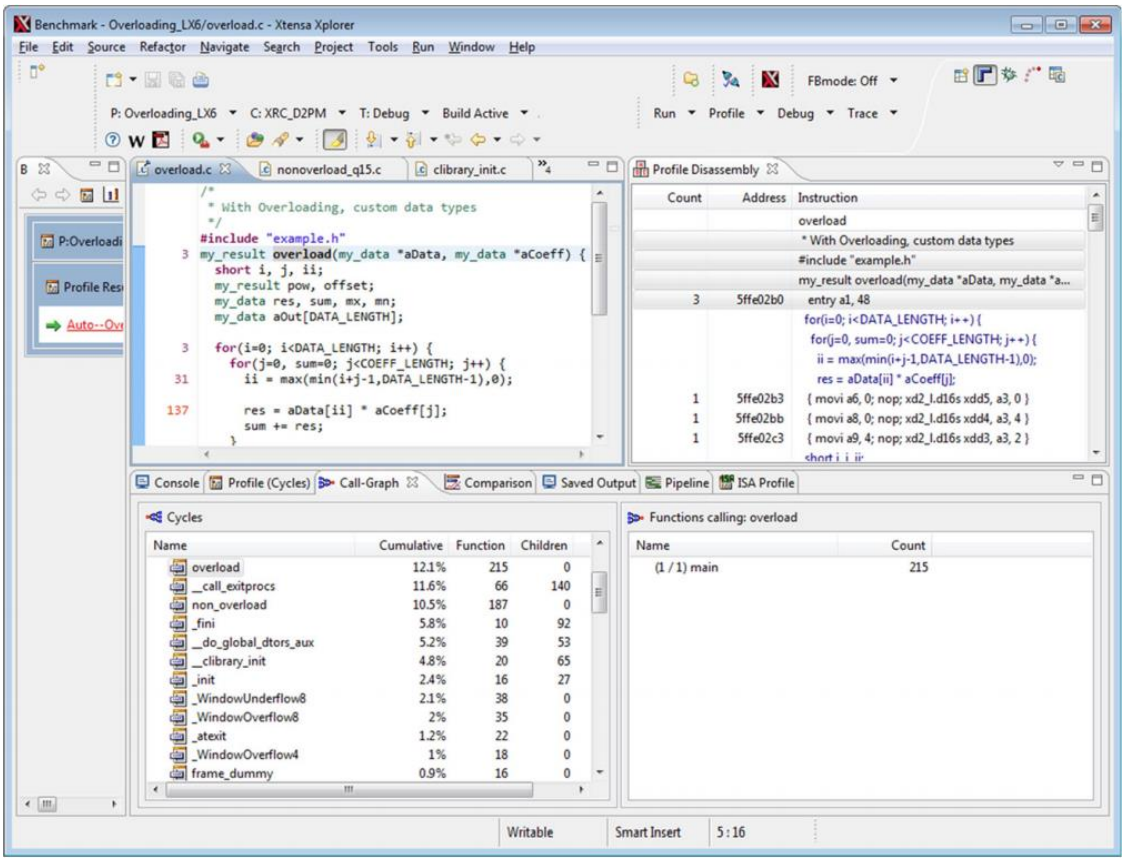
5.1 Χαρακτηρισμός Αλγορίθμου (Profiling)

Ο χαρακτηρισμός ενός αλγορίθμου (profiling) είναι ένα εξαιρετικά χρήσιμο εργαλείο, για την βελτιστοποίηση της απόδοσης της εφαρμογής. Με το πρόγραμμα Xtensa Xplorer μπορούμε να χαρακτηρίσουμε τον αλγόριθμο μέσω του προσομοιωτή συνόλου εντολών ISS (Instruction Set Simulator), ο οποίος πραγματοποιεί ακριβή προσομοίωση της διασωλήνωσης (pipeline-accurate). Με τη χρήση του profiler παράγονται δεδομένα όπως το σύνολο των κύκλων εκτέλεσης μια εντολής, οι εκτελέσεις των υπο-ρουτινών, οι κύκλοι εκτέλεσης της κάθε υπο-ρουτίνας, η απόδοση της μνήμης ταχείας προσπέλασης - cache κ.ά. Παράλληλα, μπορούμε να δούμε τον κώδικα σε γλώσσα assembly που παράγεται κατά τη μετάφρασή του, καθώς και τα instruction counts που απαιτούνται για την εκτέλεση της κάθε εντολής.



Εικόνα 11: Αποτέλεσμα του profiler, με ένδειξη στα επίπεδα διασωλήνωσης

Με το profile του αλγορίθμου είναι δυνατό να εντοπιστούν τα τμήματα κώδικα τα οποία είναι "ακριβά", με απώτερο σκοπό τα τμήματα αυτά να υλοποιηθούν σε επίπεδο υλικού (hardware), ώστε να επιτευχθεί η βέλτιστη εκτέλεσή τους. Αυτή η διαδικασία πραγματοποιείται από το πρόγραμμα Xtensa Xplorer και το αποτέλεσμα που παράγεται είναι το ακόλουθο:



Εικόνα 12: Αποτέλεσμα του profiler. Κάτω αριστερά παρουσιάζεται το Call-Graph και πάνω δεξιά ο κώδικας Assembly μαζί με τα instruction counts για κάθε γραμμή.

Με αυτό τον τρόπο ο χρήστης γνωρίζει επακριβώς τι γίνεται σε κάθε γραμμή του κώδικά του. Για κάθε συνάρτηση, με τη βοήθεια του παραθύρου του Call-Graph, ο χρήστης μπορεί να δει τι ποσοστό καταναλώνει σε επεξεργασία η κάθε συνάρτηση. Παράλληλα, όπως προαναφέρθηκε, είναι εμφανής και ο κώδικας σε γλώσσα Assembly που παράγεται για κάθε γραμμή του κώδικα, μαζί με την ένδειξη instruction count, που δείχνει πόσους κύκλους εκτέλεσης χρειάζεται η κάθε εντολή για να εκτελεστεί.

Εκτός από το αποτέλεσμα της παραπάνω εικόνας, υπάρχει και η δυνατότητα που δίνεται από τον προσομοιωτή συνόλου εντολών (Instruction Set Simulator – ISS), ο χρήστης να δει λεπτομέρειες για τη συνολική εκτέλεση του προγράμματος.

```
Xtensa Core: "test22"  ISS Version: 13.0.7

Time for Simulation = 0.30 seconds

Current PC = 0x60004647

Events                            Number  Number
                                          per 100
                                          instrs

Committed instructions          206269 ( 100.00 )
Taken branches                   14180 (   6.87 )
Exceptions                         285 (   0.14 )
   WindowOverflow                  143 (   0.07 )
   WindowUnderflow                 142 (   0.07 )
Loads                            85470 (  41.44 )
Stores                           32860 (  15.93 )

Cycles: total = 288622
                                                Summed |            Summed
                                     CPI         CPI   |% Cycle   % Cycle
Committed instructions          206269 ( 1.0000   1.0000 |  71.47    71.47 )
Taken branches                   28476 ( 0.1381   1.1381 |   9.87    81.33 )
Pipeline interlocks              51585 ( 0.2501   1.3881 |  17.87    99.21 )
Exceptions                        1425 ( 0.0069   1.3950 |   0.49    99.70 )
Sync replays                       577 ( 0.0028   1.3978 |   0.20    99.90 )
Special instructions               249 ( 0.0012   1.3991 |   0.09    99.99 )
Loop overhead                       36 ( 0.0002   1.3992 |   0.01   100.00 )
Reset                                5 ( 0.0000   1.3993 |   0.00   100.00 )
```

Εικόνα 13: Τελικό αποτέλεσμα μετά από μια προσομοίωση

5.2 Επέκταση Συνόλου Εντολών (TIE)

Η βελτιστοποίηση της απόδοσης ενός επεξεργαστή Xtensa, επιτυγχάνεται με τη χρήση του επεκτάσιμου συνόλου εντολών με τη χρήση της γλώσσας TIE. Παρακάτω παρουσιάζονται ορισμένες από τις βασικές εντολές, καθώς και το συντακτικό αυτής της γλώσσας.

5.2.1 Η εντολή Assign

Με την εντολή assign μπορούν να αρχικοποιηθούν ή να ανατεθούν τιμές στους καταχωρητές μέσα από τη γλώσσα TIE. Ένα τυπικό παράδειγμα είναι το ακόλουθο:

assign <όνομα_ register> = <έκφραση>;

```
assign register_i = 5;
```

5.2.2 Η εντολή Operation

Η συγκεκριμένη εντολή είναι το δομικό στοιχείο της γλώσσας TIE. Με αυτή την εντολή ορίζονται οι νέες λειτουργίες, τις οποίες όπως θα δούμε, μπορεί να αξιοποιήσει ο κώδικας που είναι γραμμένος σε γλώσσα C. Έστω το ακόλουθο τμήμα κώδικα σε γλώσσα προγραμματισμού C, το οποίο κάνει 32-bit endian [22] μετατροπή:

```
unsigned byteswap(unsigned swap_in) {
    unsigned swap_out = (swap_in<<24) |
                        ((swap_in<<8)&0xff0000) |
                        ((swap_in>>8)&0xff00) |
                        (swap_in>>24);
    return swap_out;
}
```

Η παραπάνω συνάρτηση θα χρειαστεί αρκετούς κύκλους εκτέλεσης για να υπολογιστεί σε επίπεδο υλικού (software). Όμως, μπορεί να υπολογιστεί σε έναν κύκλο εκτέλεσης με τη χρήση μίας εντολής TIE operation. Μία χρήσιμη τεχνική για την επιτάχυνση των "hot spots" είναι ο συνδυασμός πολλών εντολών σε μία operation. Αυτό επιτρέπει στον επεξεργαστή Xtensa να εκτελέσει περισσότερους υπολογισμούς σε έναν κύκλο εκτέλεσης. Έτσι, για το παραπάνω παράδειγμα, με την εντολή operation θα χρειαστεί μόνο ένας κύκλος εκτέλεσης. Στη συνέχει παρουσιάζεται πως γίνεται ο αλγόριθμος να αξιοποιήσει τις καινούργιες εντολές operations για να επιτευχθεί η επιτάχυνση.

```
operation byteswap {out AR swap_out, in AR swap_in} {}
{
assign swap_out =
{swap_in[7:0],swap_in[15:8],swap_in[23:16],swap_in[31:24]
};
}
```

Το συντακτικό είναι:

```
operation <name> {<argument list>} {<state-interface
list>}
{
    <operation description>
}
```

Στο τμήμα `{<argument list>}` προσδιορίζεται για κάθε καταχωρητή αν θα είναι εισόδου (in), εξόδου (out) ή θα έχει και τις δύο λειτουργίες (inout).

5.2.3 Η εντολή Wire

Επειδή η γλώσσα TIE μπορεί να περιγράψει τη λειτουργικότητα με πολλούς διαφορετικούς και σύνθετους τελεστές, η ανάθεση του αποτελέσματος με μια εντολή TIE γίνεται αρκετά πολύπλοκη. Οι σύνθετες εκφράσεις TIE, μεταφράζονται σε λειτουργίες TIE που πολύ συχνά είναι δύσκολο να διαβαστούν και να διορθωθούν. Με τη χρήση της εντολής TIE Wire, οι σύνθετες εντολές εγγράφονται σε απλούστερες εντολές TIE.

Η δήλωση γίνεται ως εξής:

```
wire[high_bit:low_bit] <wire_name>;
```

Η αρχικοποίηση γίνεται ως:

```
assign <wire_name> = <variable>[high_bit:low_bit];
```

Το παράδειγμα που ακολουθεί είναι από μια δομή επανάληψης σε γλώσσα προγραμματισμού C και στη συνέχεια η αντίστοιχη εντολή σε γλώσσα TIE.

```
for (i=0;i<N;i++){
     acc += (Sample[i]*Coeff[i])>>8;
}
```

Ο παραπάνω κώδικας κάνει μια αρκετά σύνθετη πράξη η οποία περιλαμβάνει έναν πολλαπλασιασμό μεταξύ δύο τιμών από δύο πίνακες (Sample και Coeff), μετάθεση των bit δεξιά κατά 8 θέσεις, και τέλος πρόσθεση στην μεταβλητή acc. Αν προσπαθήσουμε να δημιουργήσουμε απλές εντολές TIE (TIE operation), το αποτέλεσμα θα είναι κάτι πολύ δύσκολο στη διαχείρισή του, ειδικά για τη διόρθωση τυχόν σφαλμάτων. Έτσι με τη χρήση της εντολής TIE wire μπορεί να παραχθεί ο παρακάτω κώδικας:

```
operation scaled_mac {inout AR oper0, in AR oper1, in AR
oper2}{}
{
     //declare wires
     wire [15:0] temp1, temp2;
     wire [31:0] product, scaled;
     //assignment of intermediate computations
     assign temp1 = oper1[15:0];
     assign temp2 = oper2[15:0];
     assign product = temp1 * temp2;
     assign scaled = product >> 8;
     assign oper0 = oper0 + scaled;
}
```

Επομένως μπορεί να μετασχηματιστεί ο κώδικας σε γλώσσα C στον ακόλουθο, ώστε να χρησιμοποιήσει την εντολή που δημιουργήθηκε σε γλώσσα TIE.

```
for (i=0;i<N;i++){
     scaled_mac(acc, Sample[i], Coeff[i]);
}
```

5.2.4 Η εντολή State

Η εντολή TIE state, είναι καταχωρητής ειδικού σκοπού και χρησιμοποιείται αποκλειστικά από εντολές TIE (TIE operations). Η χρήση της συνιστάται σε περιπτώσεις όπου χρειάζεται να διατηρηθούν δεδομένα τα οποία χρησιμοποιούνται συχνά από τον επεξεργαστή. Με αυτόν τον τρόπο αυξάνεται η απόδοση καθώς ο επεξεργαστής έχει άμεση πρόσβαση σε αυτά τα δεδομένα.

Η δήλωση γίνεται της συγκεκριμένης εντολής γίνεται ως εξής:

```
state <name> <width_in_bits> <add_read_write>;
```

π.χ.

```
state acc0 32 add_read_write
```

Με την παραπάνω δήλωση **add_read_write**, ο compiler της γλώσσας TIE θα δημιουργήσει δύο εντολές (μία για ανάγνωση και μία για εγγραφή), ώστε να μπορεί το πρόγραμμα σε γλώσσα C να έχει πρόσβαση. Έτσι, έχουμε τις συναρτήσεις WUR (Write User Register) και RUR (Read User Register), όπου το τελικό συντακτικό είναι το ακόλουθο:

```
WUR_acc0(value_to_assign);
RUR_acc0();
```

5.2.5 Η εντολή Functions

Με την εντολή TIE function δημιουργείται ένα μπλοκ κώδικα από συνεχόμενες αναθέσεις, οι οποίες έχουν προκαθορισμένο το μέγεθος εισόδου και εξόδου. Αυτή η τεχνική χρησιμοποιείται για να γίνει ο κώδικας TIE πιο ευέλικτος. Θα μπορούσε για παράδειγμα να δημιουργηθεί μια TIE function, η οποία θα περιλαμβάνει μια πρόσθεση και έναν πολλαπλασιασμό. Το συντακτικό της συγκεκριμένης εντολής είναι:

```
function [<width>] (input argument list)
     {function description}
```

Η συνάρτηση αυτή δέχεται ως όρισμα ένα πλήθος από N μεταβλητές και παράγει ως αποτέλεσμα μια τιμή.

5.2.6 Η εντολή Register File

Με την γλώσσα TIE μπορεί να δημιουργηθεί ένα αρχείο από καινούριους καταχωρητές (Register File), το οποίο θα ενσωματωθεί στον επεξεργαστή Xtensa. Ένα Register File δηλώνεται ως εξής:

```
regfile <name> <width> <depth> <short_name>,
```

όπου `<depth>` είναι ο αριθμός των καταχωρήσεων που μπορούν να πραγματοποιηθούν, `<short_name>` είναι το όνομα το οποίο θα εμφανίζεται στις εντολές assembly, και `<name>` είναι το όνομα του καταχωρητή. Αυτό έχει ως αποτέλεσμα όποια μεταβλητή, στον κώδικα της γλώσσας C δηλωθεί με τύπο το όνομα του καταχωρητή, όλες οι εντολές TIE να έχουν άμεση πρόσβαση.

5.2.7 Η εντολή TIE Modules

Η TIE modules είναι παρόμοια με τις συναρτήσεις TIE (TIE functions), σε ότι αφορά την ενσωμάτωση των υπολογισμών και τη διευκόλυνση της ανάπτυξης του κώδικα. Ωστόσο, η TIE modules είναι πιο ευέλικτη. Μπορεί να αντικαταστήσει πλήρως τις συναρτήσεις TIE, ενώ ταυτόχρονα υποστηρίζει πολλαπλούς κύκλους υπολογισμών και πολλαπλών εξόδων.

5.2.7.1 Η εντολή TIEadd

Η συγκεκριμένη εντολή, προσθέτει δύο αριθμούς μαζί με το κρατούμενο τους, δηλαδή πρέπει να έχει ακριβώς τρεις παραμέτρους ως είσοδο.

```
TIEadd(a, b, cin),
```

όπου `a` και `b` οι δύο αριθμοί και `cin` το κρατούμενο.

5.2.7.2 Η εντολή TIEaddn

Η εντολή TIEaddn προσθέτει n αριθμούς με ελάχιστο αριθμό εισόδου τις τρεις μεταβλητές. Το πλεονέκτημα αυτής της εντολής είναι ότι είναι πιο γρήγορη από τον κλασικό τελεστή πρόσθεσης (+).

```
TIEaddn(A0, A1, An-1)
```

5.2.7.3 Η εντολή TIEcsa

Η εντολή TIEcsa προσθέτει ακριβώς τρείς αριθμούς και αποθηκεύει το κρατούμενο που προκύπτει. Η εντολή αυτή χρησιμοποιείται κατά κύριο λόγο όταν πρέπει να γίνει πρόσθεση πολλών αριθμών μαζί, χωρίς να χαθεί το κρατούμενό τους.

```
TIEcsa(a, b, c)
```

5.2.7.4 Η εντολή TIEmul

Η συγκεκριμένη εντολή πολλαπλασιάζει δύο αριθμούς. Αν η τιμή της τρίτης εισόδου είναι 1 τότε το αποτέλεσμα είναι προσημασμένο, διαφορετικά αν η τιμή είναι 0 τότε το αποτέλεσμα είναι μη προσημασμένο.

```
TIEmul(a, b, sign)
```

5.2.7.5 Η εντολή TIEcmp

Με την εντολή TIEcmp γίνεται σύγκριση (προσημασμένη ή μη) μεταξύ δύο αριθμών. Έχει ως είσοδο τρεις παραμέτρους και ως έξοδο τέσσερις παραμέτρους.

```
{lt, le, eq, ge, gt} = TIEcmp(a, b, sign)
```

Ανάλογα με το ποια συνθήκη ισχύει γίνεται 1 η αντίστοιχη έξοδος και 0 οι υπόλοιπες. Με “lt” να σημαίνει (a < b), “le” (a <= b), “eq” (a == b), “ge” (a >= b) και “gt” (a > b).

5.2.7.6 Η εντολή TIEmux

Η εντολή TIEmux είναι ένας πολυπλέκτης n εισόδων, ο οποίος επιστρέφει ως έξοδο την αντίστοιχη είσοδο με βάση το σήμα επιλογής “s”. Βασικό χαρακτηριστικό είναι πως ο αριθμός εισόδων θα πρέπει να είναι δύναμη του 2, και το σήμα επιλογής θα πρέπει να έχει τιμή μέχρι $\log_2 n$.

```
TIEmux(s, D0, D1, …, Dn-1)
```

5.2.7.7 Η εντολή TIEmac

Αυτή η εντολή είναι ίσως από τις πιο σύνθετες εντολές της γλώσσας TIE. Έχει πέντε παραμέτρους ως είσοδο.

```
TIEmac(a, b, c, sign, negate)
```

Γίνεται προσημασμένος (ή μη προσημασμένος) πολλαπλασιασμός μεταξύ των μεταβλητών a και b αν η τέταρτη μεταβλητή εισόδου (sign) έχει τιμή 1 ή 0 αντίστοιχα,

και γίνεται αφαίρεση ή πρόσθεση της τρίτης μεταβλητής c αν η πέμπτη μεταβλητή εισόδου negate έχει τιμή 1 ή 0 αντίστοιχα.

5.2.8 Η εντολή TIEprint

Η εντολή TIEprint μπορεί να χρησιμοποιηθεί για την αποσφαλμάτωση του κώδικα TIE, καθώς επιτρέπει στον χρήστη να εμφανίσει μηνύματα και τιμές τελεστών. Το συντακτικό της είναι παρόμοιο με της εντολής printf() στην γλώσσα C.

```
TIEprint("\t VALUES-> %d, %d,\t %d \n",loopj,acc1,add);
```

Επιπλέον, στην παρακάτω εικόνα απεικονίζονται όλοι οι διαθέσιμοι τελεστές της γλώσσας TIE.

Operator Type	Operator Symbol
Sub-range	a[n:m]
Arithmetic	+, -, *
Logical	, &&, ||
Relational	>, <, >=, <=, ==, =
Bit-wise	~, &, |, ^,^~,~^
Reduction	&, |, ^,^~,~^
Shift	>>, <<
Concatenation	{ }
Replication	{n{}}
Conditional	?:

Εικόνα 14: Διαθέσιμοι τελεστές της γλώσσας TIE

5.3 Η χρήση της γλώσσας TIE

Αξιοποιώντας την τεχνική του χαρακτηρισμού του αλγορίθμου, είναι εφικτό να εντοπιστούν τα τμήματα κώδικα τα οποία “καθυστερούν” την εκτέλεση του αλγορίθμου. Έτσι, με όλα όσα παρουσιάστηκαν στο κεφάλαιο 5.2, είναι δυνατό να αναπτυχθούν νέες οδηγίες-εντολές στον επεξεργαστή με τη χρήση της γλώσσας TIE,

και με σχετική τροποποίηση του αρχικού κώδικα στη γλώσσα C μπορεί να επιτευχθεί επιτάχυνση. Όπως προαναφέρθηκε, με τον όρο επιτάχυνση εννοείται η μείωση του χρόνου εκτέλεσης αλλά και η μείωση στον αριθμό των απαιτούμενων κύκλων εκτέλεσης.

Έτσι, σε συνέχεια του παραδείγματος της Ενότητας 4.2, μπορεί να παραχθεί μια νέα οδηγία προς τον επεξεργαστή η οποία θα αξιοποιήσει την εντολή TIEaddn, ώστε να μετασχηματιστεί το τμήμα που κάνει πρόσθεση σε παραπάνω από 2 μεταβλητές. Θα πρέπει να δημιουργηθεί ένα αρχείο TIE, στο οποίο θα δηλωθεί η νέα αυτή εντολή:

```
operation addN {inout AR var1, in AR var2, in AR var3, in AR var4} {}
{ assign var1 = TIEaddn(var1,var2,var3,var4); }
```

Στη συνέχεια, πρέπει να πραγματοποιηθεί η εισαγωγή της βιβλιοθήκης που θα παράξει ο compiler της γλώσσας TIE, `#include <xtensa/tie/file_name.h>` και να μετασχηματιστεί ο κώδικας στη γλώσσα C για να κάνει χρήση της νέας εντολής `addN`. Επομένως, το τμήμα του κώδικα που πραγματοποιεί την πρόσθεση τεσσάρων μεταβλητών (`c2 += a+b+i;`), πλέον μπορεί να γραφεί ως `addN(c2,a,b,i);`.

Να σημειωθεί ότι επειδή ο καταχωρητής `var1` έχει δηλωθεί ως **`inout`** δεν υπάρχει λόγος να γίνει ξανά ανάθεση στην μεταβλητή `c2`. Η αλλαγή αυτή θα μειώσει τον αριθμό των απαιτούμενων κύκλων εκτέλεσης, καθώς θα μειωθούν αρκετά τα instruction fetches. Όμως, επειδή η μεταβλητή `i` είναι μια σταθερά η οποία στο συγκεκριμένο παράδειγμα χρησιμοποιείται ως μετρητής, θα μπορούσε να δημιουργηθεί ένας καταχωρητής ειδικού σκοπού (TIE state) με αποτέλεσμα να αυξηθεί κι άλλο η απόδοση του συστήματος.

6. Υλοποίηση και Σύγκριση

Χρησιμοποιώντας το πρόγραμμα Xtensa Xplorer, μελετήθηκε ο αλγόριθμος KAlign 1.4 [18]. Οι αρχικές ρυθμίσεις για όλα τα πειράματα που ακολούθησαν είναι οι ακόλουθες:

- Δέσμευση 24 GB Ram από το IDE για κάθε προσομοίωση.
- Επιλογή ως αρχείο εισόδου από το dataset extHomfam [23].

Όπως αναλύθηκε και στο κεφάλαιο 4, το πρώτο βήμα είναι η διερεύνηση και ο χαρακτηρισμός του αλγορίθμου. Με τη χρήση του profiler, διαπιστώθηκε πως υπάρχει μια συνάρτηση η οποία χρησιμοποιείται κατά 60% κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Αυτή είναι η συνάρτηση `void update_gaps(int old_len,int *gis, int new_len,int *newgaps)`, η οποία έχει είσοδο δύο δυναμικούς πίνακες. Αυτό πρακτικά σημαίνει πως η εκτέλεση της εξαρτάται πάντα από την είσοδο. Επίσης, η συγκεκριμένη συνάρτηση είναι υπεύθυνη για την συμπλήρωση των κενών στις ακολουθίες ώστε να επιτευχθεί η στοίχιση. Αυτό επιτυγχάνεται με μια δομή εμφωλευμένης επανάληψης, όπου ο αριθμός επαναλήψεων της εμφωλευμένης δομής είναι δυναμικός καθώς εξαρτάται από την εκάστοτε τιμή της θέσης `i` για έναν από τους δυο δυναμικούς πίνακες.

```
for (i = 0; i <= old_len;i++){
     add = 0;
     for (j = rel_pos;j <= rel_pos + gis[i];j++){
          if (newgaps[j] != 0){
               add += newgaps[j];
          }
     }
     rel_pos += gis[i]+1;
     gis[i] += add;
}
```

Η πρώτη προσπάθεια ήταν να μειωθεί ο αριθμός του κύκλου εκτέλεσης εντολών για το τμήμα της συνάρτησης, το οποίο είναι υπεύθυνο για την σύνθεση του τελικού αποτελέσματος. Έτσι, αναπτύχθηκε ο παρακάτω κώδικας σε γλώσσα TIE και η συνάρτηση μετασχηματίστηκε για να χρησιμοποιήσει τις δύο καινούριες εντολές.

```
operation add0 {inout AR var1, in AR var2} {}
{
     assign var1 = var1 + var2;
}

operation add1 {inout AR var1, in AR var2} {}
{
     assign var1 = TIEaddn(var1,var2,1);
}
```

Στην πρώτη συνάρτηση, `add0` πραγματοποιείται η πρόσθεση των δύο καταχωρητών. Από την άλλη πλευρά, η συνάρτηση `add1` κάνει χρήση της εντολής **`TIEaddn`** ώστε να γίνει μια πρόσθεση πολλών όρων ταυτόχρονα. Επίσης, όπως φαίνεται και στις εικόνες παρακάτω, έχει μόνο δύο όρους καθώς ο τρίτος είναι μια σταθερά που έχει τιμή 1.

```
        void update_gaps(int old_len,int*gis,int new_len,int *newgaps)
 167K   {
            unsigned int i,j;
            int add = 0;
            int rel_pos = 0;
6733K       for (i = 0; i <= old_len;i++){
2939K           add = 0;
 26M            for (j = rel_pos;j <= rel_pos + gis[i];j++){
9792K               if (newgaps[j] != 0){
 24K                    add += newgaps[j];
                    }
                }
7348K           rel_pos += gis[i]+1;
9553K           gis[i] += add;
//              add1(rel_pos,gis[i]);
//              add0(gis[i],add);
            }
 95K    }
```

Εικόνα 15: Αρχική έκδοση της συνάρτησης update_gaps.

```
        void update_gaps(int old_len,int*gis,int new_len,int *newgaps)
 167K   {
            unsigned int i,j;
            int add = 0;
 47K        int rel_pos = 0;
6733K       for (i = 0; i <= old_len;i++){
2939K           add = 0;
 26M            for (j = rel_pos;j <= rel_pos + gis[i];j++){
9792K               if (newgaps[j] != 0){
 24K                    add += newgaps[j];
                    }
                }
//              rel_pos += gis[i]+1;
//              gis[i] += add;
6613K           add1(rel_pos,gis[i]);
9553K           add0(gis[i],add);
            }
 95K    }
```

Εικόνα 16: Η συνάρτηση update_gaps με χρήση των 2 νέων εντολών TIE.

Όπως φαίνεται από τις παραπάνω δύο εικόνες η ανάπτυξη των δύο εντολών βοήθησε, καθώς μειώθηκε ο αριθμός των απαιτούμενων κύκλων εκτέλεσης, αλλά αυτό συνέβη μόνο στο τμήμα όπου χρησιμοποιήθηκε η εντολή `add0`. Αυτό γίνεται διότι η πρόσθεση

δύο μεταβλητών είτε είναι σε επίπεδο αλγορίθμου, είτε είναι σε επίπεδο επεξεργαστή, όπως είναι η εντολή `add0`, η οποία έχει σταθερό κόστος σε κύκλους εκτέλεσης.

Όμως, αυτό δεν ήταν αρκετό καθώς το αποτέλεσμα ήταν αρκετά μικρότερο από 1% στο σύνολο της προσομοίωσης.

Έτσι, σε συνδυασμό με αρκετές από τις εντολές που παρουσιάστηκαν στο κεφάλαιο 4, αναπτύχθηκε ο τελικός κώδικας TIE, και η συνάρτηση μετασχηματίστηκε εντελώς για να γίνει αποκλειστική χρήση εντολών TIE, οι οποίες αναπτύχθηκαν για την επιτάχυνση του συγκεκριμένου τμήματος του αλγορίθμου.

Αρχικά, για όλες τις τοπικές μεταβλητές της συνάρτησης αυτής, δημιουργήθηκαν καταχωρητές ειδικού σκοπού με αποτέλεσμα την μείωση του χρόνου, αλλά και των κύκλων εκτέλεσης που χρειαζόταν για τη διαχείριση αυτού το τμήματος.

```
state add 32 add_read_write
state rel_pos 32 add_read_write
state loopi 32 add_read_write
state loopj 32 add_read_write
state loopiInit 32 add_read_write
```

Στη συνέχεια, αναπτύχθηκαν οι ακόλουθες συναρτήσεις, όπου η κάθε μια αναλαμβάνει να εκτελέσει μια λειτουργία και στις περισσότερες περιπτώσεις γίνεται και χρήση των καταχωρητών ειδικού σκοπού.

```
operation core1 {in AR var1} {inout add}
{
     assign add = var1!=0? add + var1: add;
}

operation outerForLoop {out AR result} {inout loopi, in
loopiInit,out add,out loopj,in rel_pos}
{
     assign result = loopi <= loopiInit;
     assign loopi = loopi+1;
     assign add=0;
     assign loopj=rel_pos;
}

operation innerForLoop {out AR result, in AR *var1} {out
VAddr, in MemDataIn32,inout loopj, in loopi,inout acc1,in
rel_pos}
{
     assign VAddr = (var1+((loopi-1)*4));
     assign acc1 = MemDataIn32;
     assign result = loopj <= rel_pos +acc1 ;
     assign loopj = loopj + 1;
}
```

```
operation updateStates {inout AR var1} {inout rel_pos, in
add}
{
      assign rel_pos = rel_pos +var1 +1;
      assign var1 = var1+add;
}
```

Με τις νέες αυτές λειτουργίες, η συνάρτηση του αλγορίθμου Kalign, μετασχηματίστηκε ώστε να κάνει χρήση αυτών, και το αποτέλεσμα είναι το ακόλουθο:

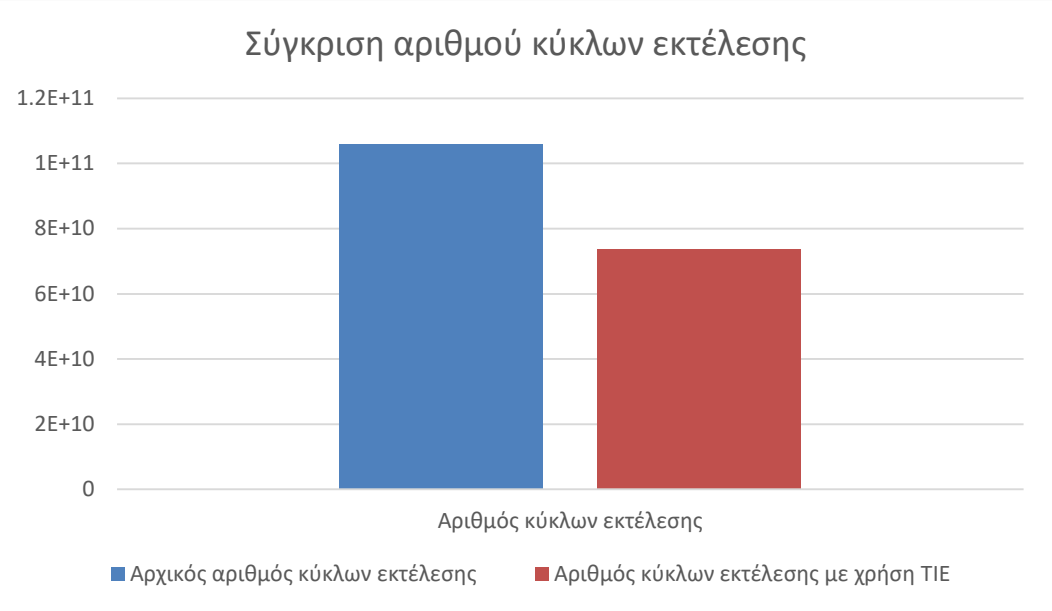
```
void update_gaps(int old_len,int*gis,int new_len,int
*newgaps)
{
      WUR_rel_pos(0);
      WUR_loopi(0);
      WUR_loopj(0);
      WUR_loopiInit(old_len);
      while(outerForLoop()){

            // CORE 1
            while(innerForLoop(gis)){
                  core1(newgaps[RUR_loopj()-1]);
            }
            updateStates(gis[RUR_loopi()-1]);
      }
}
```

Στο πρώτο μέρος γίνεται η αρχικοποίηση των καταχωρητών ειδικού σκοπού, ακριβώς όπως γινόταν και στις τοπικές μεταβλητές που είχε προηγουμένως η συνάρτηση. Στη συνέχεια, γίνεται χρήση των νέων εντολών TIE που αναπτύχθηκαν και οι δύο δομές επανάληψης, καθώς και το τμήμα που έκανε επεξεργασία για να βρεθούν τα κενά σε κάθε ακολουθία. Αξίζει να σημειωθεί, πως η εντολή `innerForLoop`, δέχεται ως παράμετρο την διεύθυνση μνήμης του δυναμικού πίνακα `gis`. Σε αυτό το στάδιο θα αναρωτηθεί κανείς πως γίνεται να γνωρίζει η εντολή αυτή, ποια θέση του πίνακα χρειάζεται για να επεξεργαστεί τα δεδομένα. Η απάντηση είναι στους καταχωρητές ειδικού σκοπού και στη δυνατότητα που παρέχει το πρόγραμμα Xtensa Xplorer, με τη χρήση της γλώσσας TIE, όπου ο χρήστης μπορεί να έχει πρόσβαση σε μια διεύθυνση μνήμης με τη χρήση ενός άλλου καταχωρητή, **`out VAddr, in MemDataIn32`**. Στην πραγματικότητα είναι πάλι ένας καταχωρητής ειδικού σκοπού όπου με τη δήλωση **`out VAddr`**, ο χρήστης μπορεί να έχει πρόσβαση στη θέση μνήμης και με την δήλωση **`in MemDataIn32`**, μπορεί να γίνει προσπέλαση των δεδομένων που υπάρχουν εκεί.

Πλέον διαπιστώθηκε πως για αρχεία μεγαλύτερα από 100KB, η επιτάχυνση είναι εμφανής και κατά τη διάρκεια των πειραμάτων, όπου χρησιμοποιήθηκε το αρχείο hemopexin (293KB), και το τελικό αποτέλεσμα της επιτάχυνσης έφτασε το 30.29%. Ως επιτάχυνση αναφέρεται η μείωση του αριθμού των απαιτούμενων κύκλων

εκτέλεσης αλλά και ο συνολικός χρόνος εκτέλεσης του αλγορίθμου, όπου με τις συνολικές αλλαγές η επιτάχυνση στο χρόνο έφτασε στο 43.49%.



Εικόνα 17: Σύγκριση των απαιτούμενων instruction count, πριν και μετά τη χρήση TIE για επιτάχυνση της συνάρτησης update_gaps

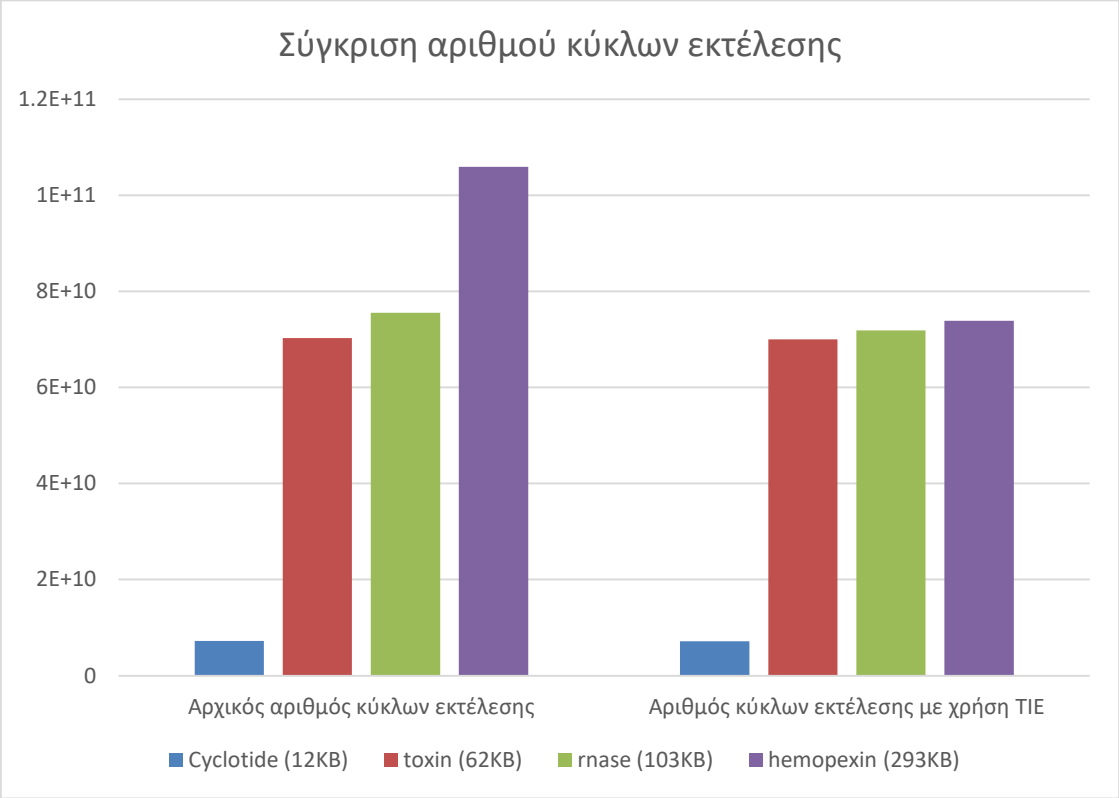
Χωρίς τη χρήση των νέων εντολών που δημιουργήθηκαν ο αρχικός αριθμός κύκλων εκτέλεσης ήταν 105940161745, ενώ με τη χρήση του κώδικα που αναπτύχθηκε οι κύκλοι εκτέλεσης είναι 73845557474.

Εικόνα 18: Σύγκριση του απαιτούμενου χρόνου εκτέλεσης, πριν και μετά τη χρήση TIE για επιτάχυνση της συνάρτησης update_gaps

Χωρίς τη χρήση των νέων εντολών που δημιουργήθηκαν ο χρόνος που απαιτούνταν για την εκτέλεση του αλγορίθμου ήταν 123861 δευτερόλεπτα, ενώ με τη χρήση των εντολών TIE που αναπτύχθηκαν ο νέος χρόνος εκτέλεσης είναι 69992 δευτερόλεπτα.

Με βάση όλα τα παραπάνω, συμπεραίνουμε πως η επιτάχυνση που επιτυγχάνεται δεν είναι σταθερή, αλλά εξαρτάται κάθε φορά από το μέγεθος του αρχείου εισόδου. Αυτό γίνεται διότι, όπως παρουσιάστηκε παραπάνω, η συνάρτηση `update_gaps`, δέχεται ως ορίσματα πίνακες ακέραιων αριθμών, οι οποίοι έχουν δυναμικό μέγεθος.

Σύμφωνα με τα πειράματα που διεξήχθησαν, παρατηρήθηκε πως όσο μεγαλύτερο είναι το αρχείο εισόδου, τόσο μεγαλύτερη είναι και η επιτάχυνση που επιτυγχάνεται. Επίσης, κατά τη διάρκεια των πειραμάτων παρατηρήθηκε πως, όταν ο αλγόριθμος άγγιζε στο 80-82% της επεξεργασίας των ακολουθιών, υπήρχε απότομη πτώση της ταχύτητας του ακόμα και στην περίπτωση της χρήσης των εντολών TIE.



Εικόνα 19: Σύγκριση του συστήματος με διαφορετική είσοδο.

7. Συμπεράσματα

Με την εκπόνηση της παρούσας πτυχιακής εργασίας, αναπτύχθηκε ένα IP Core, με στόχο την επιτάχυνση του αλγορίθμου Kalign 1.4 [18]. Με τη χρήση της γλώσσας TIE και του προγράμματος Xtensa Xplorer, παρουσιάστηκε ένας μικροεπεξεργαστής όπου κάνοντας χρήση των νέων εντολών, μπορεί να επιτευχθεί επιτάχυνση στον αλγόριθμο Kalign. Η συγκεκριμένη εργασία αποτελεί τη βάση για την ανάπτυξη ενός μικροεπεξεργαστή ειδικού σκοπού, για εφαρμογές στον τομέα της Βιοπληροφορικής. Προτείνεται η ανάπτυξη ενός νέου συνόλου εντολών το οποίο θα μπορεί να αξιοποιηθεί από ένα ευρύ φάσμα αλγορίθμων Βιοπληροφορικής. Τέλος συμπεραίνουμε ότι με τη μείωση του χρόνου εκτέλεσης, μειώνεται και η κατανάλωση ρεύματος.

Λόγω της ακαδημαϊκής άδειας που υπήρχε για την χρήση του Xtensa Xplorer, δεν υπήρχε πρόσβαση στα εργαλεία που είναι υπεύθυνα για να δείξουν τον χώρο που θα καταλαμβάνει τελικά αυτός ο μικροεπεξεργαστής καθώς και την πιθανή κατανάλωση ενέργειας, η οποία προκύπτει μέσα από τις προσομοιώσεις.

7.1 Μελλοντικές προεκτάσεις της εργασίας

Στην παρούσα πτυχιακή εργασία, ερευνήθηκε ο τρόπος επιτάχυνσης ενός αλγορίθμου με τη χρήση των εργαλείων που έχει αναπτύξει η εταιρεία Tensilica. Καθώς ο στόχος είναι η ανάπτυξη ενός μικροεπεξεργαστή ειδικού σκοπού, για όσο το δυνατόν περισσότερους αλγορίθμους Βιοπληροφορικής γίνεται, η πτυχιακή εργασία αποτελεί το κίνητρο για μια σειρά μελλοντικών ερευνητικών δράσεων οι οποίες μπορούν να πραγματοποιηθούν είτε από τον συγγραφέα, είτε από οποιονδήποτε επιθυμεί να εξελίξει το IP Core που αναπτύχθηκε.

Σε ερευνητικό επίπεδο, η ανάλυση των αλγορίθμων θα συνεχιστεί ώστε το IP Core, να μπορεί να χρησιμοποιηθεί σε διάφορους αλγορίθμους, είτε κάνουν πολλαπλή στοίχιση ακολουθιών, είτε κάποιο άλλο είδος ανάλυσης το οποίο απαιτεί γρήγορη επεξεργασία των δεδομένων και άμεση παραγωγή των αποτελεσμάτων. Επίσης, θα διερευνηθεί η χρήση σύνθετων εντολών TIE με απώτερο σκοπό την αύξηση της επιτάχυνσης. Παράλληλα θα εξεταστεί και η χρήση επεκτάσιμων GPU, όπως είναι η κάρτες ALVEO από τη XILINX [24], ώστε αλγόριθμοι όπως ο FAMSA, να μπορούν να αξιοποιήσουν στο έπακρο τις νέες τεχνολογίες.

Σε επίπεδο ανάπτυξης, το προτεινόμενο σύστημα μπορεί να υλοποιηθεί σε ολοκληρωμένη εμπορική λύση, αξιοποιώντας την πλατφόρμα FPGA, ώστε να αναπτυχθεί αρχικά ένας μικροεπεξεργαστής και στη συνέχεια αυτή η λύση μπορεί να τυπωθεί ως ανεξάρτητη μονάδα επεξεργασίας, για παράδειγμα ένα USB stick όπως είναι το Coral από την Google για μηχανική μάθηση [25].

Μια άλλη επέκταση που θα μπορούσε να διερευνηθεί είναι η υλοποίηση σε πλατφόρμα που υποστηρίζει RISC-V [26]. Είναι μια αρχιτεκτονική συνόλου εντολών (ISA), η οποία είναι ανοικτού κώδικα. Ξεκίνησε το 2010 στο Berkeley, με στόχο να

δημιουργηθεί ένα ISA, το οποίο θα είναι ανοικτό και θα μπορεί να χρησιμοποιηθεί ακαδημαϊκά, αλλά και σε όποιο άλλο ανοικτό project [27] [28].

7.2 Ερευνητικά Αποτελέσματα

Με την ολοκλήρωση της παρούσας εργασίας θα υποστηριχτεί μια δημοσίευση με τίτλο «Exploration Study on Configurable Instruction Setfor Bioinformatics' Applications». Σε αυτή γίνεται ανάλυση της διαδικασίας που μπορεί να ακολουθήσει κάποιος, ώστε να ξεκινήσει για πρώτη φορά με το πρόγραμμα Xtensa Xplorer και να υλοποιήσει ένα σύνολο εντολών, με αποτέλεσμα να δημιουργηθεί ένα νέο ISA.

ΠΑΡΑΡΤΗΜΑ Α

Κώδικας 1 (Kalign – Αρχική έκδοση)

```
void update_gaps(int old_len,int*gis,int new_len,int
*newgaps)
{
	unsigned int i,j;
	int add = 0;
	int rel_pos = 0;
	for (i = 0; i <= old_len;i++){
		add = 0;

		for (j = rel_pos;j <= rel_pos + gis[i];j++){
			if (newgaps[j] != 0){
				add += newgaps[j];
			}
		}
		rel_pos += gis[i]+1;
		gis[i] += add;
	}
}
```

Κώδικας 2 (Kalign – Με χρήση TIE)

```
void update_gaps(int old_len,int*gis,int new_len,int
*newgaps)
{
	WUR_rel_pos(0);
	WUR_loopi(0);
	WUR_loopj(0);
	WUR_loopiInit(old_len);
	while(outerForLoop()){

		// CORE 1
		while(innerForLoop(gis)){
			core1(newgaps[RUR_loopj()-1]);
		}
		updateStates(gis[RUR_loopi()-1]);
	}
}
```

Κώδικας 3 (TIE)

```
state add 32 add_read_write
state rel_pos 32 add_read_write
state loopi 32 add_read_write
state loopj 32 add_read_write
state loopiInit 32 add_read_write

operation core1 {in AR var1} {inout add}
{
     assign add = var1!=0? add + var1: add;
}

operation outerForLoop {out AR result} {inout loopi, in
loopiInit,out add,out loopj,in rel_pos}
{
     assign result = loopi <= loopiInit;
     assign loopi = loopi+1;
     assign add=0;
     assign loopj=rel_pos;
}

operation innerForLoop {out AR result, in AR *var1} {out
VAddr, in MemDataIn32,inout loopj, in loopi,inout acc1,in
rel_pos}
{
     assign VAddr = (var1+((loopi-1)*4));
     assign acc1 = MemDataIn32;
     assign result = loopj <= rel_pos +acc1 ;
     assign loopj = loopj + 1;
}

operation updateStates {inout AR var1} {inout rel_pos, in
add}
{
     assign rel_pos = rel_pos +var1 +1;
     assign var1 = var1+add;
}
```

Βιβλιογραφία

[1] Stanford, «RISC vs. CISC,» Stanford, [Ηλεκτρονικό]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/. [Πρόσβαση 8 2019].

[2] R. Prew, «Battle Over the FPGA: VHDL vs Verilog! Who is the True Champ?,» DIGILENT, [Ηλεκτρονικό]. Available: https://blog.digilentinc.com/battle-over-the-fpga-vhdl-vs-verilog-who-is-the-true-champ/. [Πρόσβαση 9 2019].

[3] F. 4. STUDENT, «Verilog vs VHDL: Explain by Examples,» FPGA 4 STUDENT, [Ηλεκτρονικό]. Available: https://www.fpga4student.com/2017/08/verilog-vs-vhdl-explain-by-example.html. [Πρόσβαση 9 2019].

[4] Xilinx, «What is FPGA,» Xilinx, [Ηλεκτρονικό]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Πρόσβαση 8 2019].

[5] CAST, «BA2X 32-bit Processor IP,» CAST, [Ηλεκτρονικό]. Available: http://www.cast-inc.com/ip-cores/processors32bit/index.html. [Πρόσβαση 8 2019].

[6] CAST, «BA2x-AXI-PP,» CAST, [Ηλεκτρονικό]. Available: http://www.cast-inc.com/ip-cores/processors32bit/ba2x-axi-pp/index.html. [Πρόσβαση 9 2019].

[7] CADENCE, «Tensilica Customizable Processors,» CADENCE, [Ηλεκτρονικό]. Available: https://ip.cadence.com/ipportfolio/tensilica-ip/xtensa-customizable. [Πρόσβαση 6 2019].

[8] CADENCE, «TIE Language - The Fast Path to High-Performance,» CADENCE, [Ηλεκτρονικό]. Available: https://ip.cadence.com/uploads/980/TIP_WP_TIE_FINAL-pdf. [Πρόσβαση 6 2019].

[9] Π. Μπάγκος, «Βιοπληροφορική,» σε *Εισαγωγή στη Βιοπληροφορική*, Εκδόσεις Κάλλιπος, 2015.

[10] N. M. a. G. D. a. G. M. Luscombe, «What is bioinformatics? A proposed definition and overview of the field,» *Methods of information in medicine,* τόμ. 40, pp. 346-358, 2001.

[11] C. Notredame, «Recent Evolutions of Multiple Sequence Alignment Algorithms,» *PLoS computational biology,* τόμ. 3, 2007.

[12] S. &. U. DUBLIN, «Clustal,» [Ηλεκτρονικό]. Available: http://www.clustal.org/. [Πρόσβαση 8 2019].

[13] F. a. H. D. G. Sievers, «Clustal Omega for making accurate alignments of many protein sequences,» *Protein Science,* τόμ. 27, 2018.

[14] M. A. a. B. G. a. B. N. a. C. R. a. M. P. A. a. M. H. a. V. F. a. W. I. M. a. W. A. a. L. R. a. o. Larkin, «Clustal W and Clustal X version 2.0,» *Bioinformatics,* τόμ. 23, 2007.

[15] A. D.-G. &. A. G. Sebastian Deorowicz, «FAMSA: Fast and accurate multiple sequence alignment of huge protein families,» *Scientific Reports,* αρ. 6, 2016.

[16] T. L. &. E. L. Sonnhammer, «Kalign – an accurate and fast multiple sequence alignment algorithm,» *BMC Bioinformatics,* τόμ. 6, αρ. 1, 2005.

[17] V. a. G. V. Bhardwaj, «Efficient Wu Manber string matching algorithm for large number of patterns,» *International Journal of Computer Applications,* τόμ. 132, 2015.

[18] S. B. Centers, «Kalign,» SBC, [Ηλεκτρονικό]. Available: http://msa.sbc.su.se/cgi-bin/msa.cgi. [Πρόσβαση 8 2019].

[19] GCC, «GCC,» GCC, [Ηλεκτρονικό]. Available: https://gcc.gnu.org/install/binaries.html. [Πρόσβαση 2019].

[20] Cadence, «Xtensa Xplorer,» Cadence, [Ηλεκτρονικό]. Available: https://ip.cadence.com/uploads/103/SWdev-pdf. [Πρόσβαση 8 2019].

[21] Eclipse, «Eclipse,» Eclipse Foundation, [Ηλεκτρονικό]. Available: https://www.eclipse.org. [Πρόσβαση 8 2019].

[22] H. Adiga, «Writing endian-independent code in C,» IBM, [Ηλεκτρονικό]. Available: https://developer.ibm.com/articles/au-endianc/. [Πρόσβαση 9 2019].

[23] G. Adam, «extHomFam benchmark,» Harvard Dataverse, [Ηλεκτρονικό]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/BO2SVW. [Πρόσβαση 8 2019].

[24] «Xilinx,» Xilinx, 2019. [Ηλεκτρονικό]. Available: https://www.xilinx.com/products/boards-and-kits/alveo.html. [Πρόσβαση 4 2019].

[25] Google, «Coral Beta,» Google, [Ηλεκτρονικό]. Available: https://coral.withgoogle.com/products/accelerator/. [Πρόσβαση 2019].

[26] RISC-V, «RISC-V,» RISC-V, [Ηλεκτρονικό]. Available: https://riscv.org. [Πρόσβαση 8 2019].

[27] K. A΄. I. Andrew Waterman, «RISC-V Specifications,» [Ηλεκτρονικό]. Available: https://riscv.org/specifications/. [Πρόσβαση 8 2019].

[28] D. A. P. Krste Asanović, «Instruction sets should be free: The case for risc-v,» *EECS Department, University of California, Berkeley,* 2014.

[29] MIT, «WIRELESS IMPLANTABLE MEDICAL DEVICES,» MIT, [Ηλεκτρονικό]. Available: https://groups.csail.mit.edu/netmit/IMDShield/. [Πρόσβαση 9 2019].

[30] E. Pull, «Electronic Pull,» [Ηλεκτρονικό]. Available: https://electronicpull.blogspot.com/2016/05/risc-and-cisc-architectures.html. [Πρόσβαση 8 2019].

Πηγές Εικόνων