Ατομική Διπλωματική Εργασία

ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ XMT

Άντρια Αναστασίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Προσομοίωση και Πειραματική Αξιολόγηση Παράλληλων Τυχαιοποιημένων Αλγορίθμων Εύρεσης Ανεξάρτητου Συνόλου στην Πλατφόρμα XMT

Άντρια Αναστασίου

Επιβλέπων Καθηγητής
Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2015

Ευχαριστίες

Θέλω να ευχαριστήσω τον επιβλέποντα καθηγητή της παρούσας διπλωματικής εργασίας κ.Χρύση Γεωργίου, κυρίως για την εμπιστοσύνη που μου έδειξε αλλά και για την πολύτιμη βοήθεια του.

Θέλω επίσης να ευχαριστήσω τα αγαπημένα μου πρόσωπα, την μητέρα μου, την αδελφή μου, τον νονό και την νονά μου που αποδέχθηκαν όλες τις επιλογές μου και μου παρείχαν στήριξη όλο αυτό το διάστημα, χωρίς την οποία τίποτα από όσα έχω καταφέρει μέχρι σήμερα δε θα ήταν πραγματικότητα.

Τέλος, το πιο μεγάλο ευχαριστώ ανήκει στον μικρό Θοδωρή στον οποίο και αφιερώνω την παρούσα διπλωματική εργασία.

Περίληψη

Το κύριο θέμα με το οποίο ασχολείται η παρούσα διπλωματική εργασία είναι ο παραλληλισμός, δηλαδή η ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από ένα επεξεργαστές. Γίνεται υλοποίηση παράλληλων αλγορίθμων στην πλατφόρμα XMT με την γλώσσα προγραμματισμού XMTC. Οι αλγόριθμοι που έχουν υλοποιηθεί είναι στο μοντέλο παραλληλισμού PRAM.

Πιο συγκεκριμένα υλοποιήθηκαν δύο τυχαιοποιημένοι αλγόριθμοι για την εύρεση ανεξάρτητου συνόλου. Πάρθηκαν μετρήσεις για την πειραματική αξιολόγηση των αλγορίθμων, για την κατασκευή γραφικών παραστάσεων και την καταγραφή παρατηρήσεων, για την εξαγωγή συμπερασμάτων και για την σύγκριση της θεωρητικής και της πραγματικής απόδοσης των αλγορίθμων.

Κατά την υλοποίηση των αλγορίθμων και κατά την συγγραφή της εργασίας αυτής απέκτησα αρκετές εμπειρίες και χρήσιμες γνώσεις για την χρήση του προσομοιωτή XMT αλλά και γενικότερα για την παράλληλη επεξεργασία.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχος Διπλωματικής Εργασίας

Στόχος της παρούσας Ατομικής Διπλωματικής Εργασίας είναι η υλοποίηση και η προσομοίωση παράλληλων αλγορίθμων με σκοπό την εξαγωγή παρατηρήσεων και συμπερασμάτων. Η υλοποίηση των αλγορίθμων αυτών έγινε στην γλώσσα προγραμματισμού XMTC και η προσομοίωση τους έγινε με την χρήση του προσομοιωτή XMT.

Βασικό κομμάτι της παρούσας Ατομικής Διπλωματικής Εργασίας είναι η πειραματική αξιολόγηση με σκοπό την θεωρητική και πρακτική σύγκριση των αλγορίθμων. Αποτέλεσμα της πειραματικής αξιολόγησης είναι η διαπίστωση αν πράγματι τα αποτελέσματα της πρακτικής αξιολόγησης συνάδουν με τα αποτελέσματα της θεωρητικής ανάλυσης.

Στόχος των αλγορίθμων της παρούσας Ατομικής Διπλωματικής Εργασίας είναι η εύρεση ανεξάρτητου συνόλου σε κατευθυνόμενους κύκλους και σε επίπεδους γράφους.

Ανεξάρτητο Σύνολο: Δεδομένου ενός γραφήματος, ένα σύνολο κόμβων είναι ανεξάρτητο αν δεν υπάρχουν κόμβοι που να ενώνονται ανά δύο με μία άκρη. Το ανεξάρτητο σύνολο πρέπει να είναι όσο το δυνατό πιο μεγάλο [17].

Το πρόβλημα του ανεξάρτητου συνόλου έχει χρήση σε αρκετές εφαρμογές. Ένα μικρό παράδειγμα των εφαρμογών που χρησιμοποιείται είναι [18]:

1) Χρωματισμός
2) Υπολογιστική Όραση
3) Χρονοπρογραμματισμό
4) Μοριακή Βιολογία
5) Θεωρία Κωδικοποίησης
6) Ανάθεση Πόρων
7) Πρόβλημα k-βασιλισσών
8) Τηλεπικοινωνίες
9) Εκτέλεση Παράλληλων Εργασιών

1.2 Μεθοδολογία Διπλωματικής Εργασίας

Η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της παρούσας Ατομικής Διπλωματικής Εργασίας είναι η ακόλουθη. Αρχικά, έγινε μελέτη του προσομοιωτή XMT και της γλώσσας XMTC με στόχο να θυμηθώ τους κανόνες συγγραφής της λόγω του ότι ήδη γνώριζα την γλώσσα προγραμματισμού από το μάθημα ‘‘ΕΠΛ 431: ΣΥΝΘΕΣΗ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ’’ το οποίο παρακολούθησα κατά την διάρκεια των σπουδών μου. Γενικά η περαιτέρω και πιο αναλυτική εκμάθηση της γλώσσας XMTC ήταν αρκετά εύκολη λόγω του ότι είναι παρόμοια με την γλώσσα προγραμματισμού C την οποία έμαθα αρκετά καλά κατά την διάρκεια των σπουδών μου. Λόγω του ότι οι δύο αυτές γλώσσες είναι παρόμοιες και όχι όμοιες για καλύτερη κατανόηση της γλώσσας έγινε υλοποίηση απλών αλγορίθμων. Έπειτα έγινε μελέτη και κατανόηση δύο (2) παράλληλων τυχαιοποιημένων αλγορίθμων εύρεσης ανεξάρτητου συνόλου. Αφού κατανοήθηκαν πλήρως οι αλγόριθμοι έγινε η υλοποίηση τους. Τέλος, έγινε η πειραματική αξιολόγηση τους και η εξαγωγή συμπερασμάτων με βάση της αξιολόγησης αυτής.

1.3 Δομή Κειμένου

Στο δεύτερο κεφάλαιο γίνεται εισαγωγή στον παραλληλισμό και στο μοντέλο PRAM στο οποίο είναι οι αλγόριθμοι που έχουν αναπτυχθεί. Στο τρίτο κεφάλαιο γίνεται μια περιγραφή της πλατφόρμας XMT, παρουσιάζονται τα βασικά χαρακτηριστικά της γλώσσας προγραμματισμού XMTC και έπειτα περιγράφεται το εργαλείο μνήμης με το οποίο έχει γίνεται η εισαγωγή των δεδομένων. Στο τέλος του κεφαλαίου γίνεται μια συνοπτική καταγραφή των περιορισμών της γλώσσας. Στο τέταρτο και πέμπτο κεφάλαιο περιγράφεται

το πρόβλημα που έχουμε να αντιμετωπίσουμε και έπειτα παρουσιάζεται ο ψευδοκώδικας του παράλληλου αλγόριθμου. Επίσης, γίνεται πειραματική αξιολόγηση των αλγορίθμων για εξαγωγή συμπερασμάτων. Στο έκτο κεφάλαιο αναφέρονται τα τελικά συμπεράσματα που εξάγονται από την υλοποίηση των αλγορίθμων, τα προβλήματα υλοποίησης των αλγορίθμων, τα οφέλη που μου προσφέρθηκαν από την διπλωματική εργασία και στο τέλος του κεφαλαίου γίνεται πρόταση για μελλοντική εργασία.

Κεφάλαιο 2

Παράλληλος Υπολογισμός

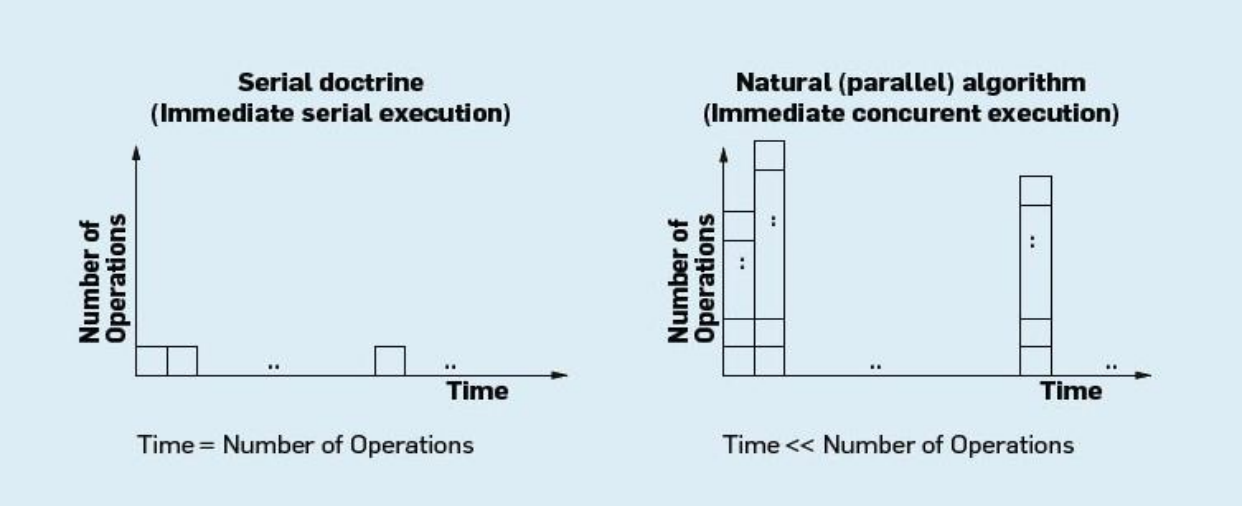
2.1 Παραλληλισμός

Αν και οι περισσότεροι από εμάς δεν το γνωρίζουμε, ο παραλληλισμός είναι μέρος της καθημερινότητας μας. Παρόλο που ο κάθε άνθρωπος ενεργεί σειριακά, ωστόσο αποτελεί τμήμα κάποιου οργανισμού, ο οποίος περιέχει πολλούς ανθρώπους που όλοι λειτουργούν παράλληλα.

Ας δώσουμε ένα παράδειγμα με το οποίο θα εμπεδώσουμε καλύτερα την σημαντικότητα του παραλληλισμού στην ανθρώπινη ζωή. Σκεφτείτε λοιπόν ένα γεωργό που έχει στο περιβόλι του πολλά δένδρα με φρούτα τα οποία είναι ώριμα και πρέπει να τα μαζέψει το συντομότερο δυνατόν. Πρέπει να έχουμε υπόψη μας πως το μάζεμα πρέπει να γίνει όσο το δυνατό πιο γρήγορα γιατί διαφορετικά τα φρούτα θα σαπίσουν. Υποθέτουμε λοιπόν πως κάθε δένδρο χρειάζεται μία μέρα για να ολοκληρωθεί το μάζεμα των καρπών του και πως το περιβόλι έχει 100 δένδρα. Αν ο γεωργός μαζεύει μόνος του τα φρούτα για να ολοκληρώσει το μάζεμα τους θα χρειαστεί 100 μέρες. Αν όμως έχουμε περισσότερους ανθρώπους να δουλεύουν ταυτόχρονα τότε θα χρειαστούν λιγότερες μέρες. Αν έχουμε 2 εργάτες τότε θα χρειαστούν συνολικά 50 ημέρες, αν έχουμε 50 εργάτες τότε θα χρειαστούν συνολικά 2 ημέρες ενώ στην καλύτερη περίπτωση αν έχουμε 100 εργάτες θα χρειαστεί συνολικά 1 ημέρα.

Αυτό το παράδειγμα από την καθημερινότητα μπορεί να μεταφραστεί και στην γλώσσα της πληροφορικής. Οι άνθρωποι-εργάτες οι οποίοι έχουν ως κοινό χαρακτηριστικό το ότι γνωρίζουν την δουλεία που πρέπει να εκτελέσουν αντιστοιχούν με τους επεξεργαστές και τα δέντρα αντιστοιχούν με τον όγκο πληροφοριών που έχουμε να επεξεργαστούμε. Δηλαδή, αν έχουμε ένα μεγάλο όγκο πληροφοριών με μόνο ένα επεξεργαστή θα πρέπει ο επεξεργαστής αυτός να αναλύσει τα δεδομένα μας σειριακά, κάτι το οποίο είναι πολύ χρονοβόρο. Αν έχουμε ένα μεγάλο όγκο πληροφοριών που πρέπει να επεξεργαστούμε με περισσότερους από ένα επεξεργαστές, η επεξεργασία θα γίνεται παράλληλα από όλους τους επεξεργαστές και έτσι θα χρειαστούμε λιγότερο χρόνο από ότι στην σειριακή επεξεργασία.

Στο πιο κάτω σχήμα 2.1 παρουσιάζεται ο τρόπος με τον οποίο γίνεται τόσο η σειριακή αλλά και η παράλληλη επεξεργασία. Από το σχήμα βλέπουμε πως στην σειριακή επεξεργασία ο χρόνος εκτέλεσης ισούται με τον αριθμό των διεργασιών που πρέπει να γίνουν ενώ στην παράλληλη επεξεργασία ο χρόνος εκτέλεσης είναι λιγότερος από των αριθμό των διεργασιών που πρέπει να γίνουν λόγω του ότι μπορούν να εκτελούνται ταυτόχρονα από τους διάφορους επεξεργαστές.



Σχήμα 2.1 [19]

2.2 Παράλληλοι Αλγόριθμοι

Πριν ορίσουμε το τι είναι ένας παράλληλος αλγόριθμος πρέπει να ορίσουμε το τι είναι ένας αλγόριθμος.

Αλγόριθμος είναι μία πεπερασμένη ακολουθία εντολών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο που οδηγούν στην επιτυχή επίλυση ενός προβλήματος

[2]. Ο όρος αλγόριθμος αναφέρεται συνήθως σε εντολές οι οποίες εκτελούνται σε σειριακά συστήματα.

Παράλληλος Αλγόριθμος είναι ένας αλγόριθμος ειδικά σχεδιασμένος για παράλληλους υπολογιστές [2]. Είναι δηλαδή, μία πεπερασμένη ακολουθία εντολών των οποίων η εκτέλεση πάνω σε παράλληλο υπολογιστικό σύστημα οδηγεί στην επιτυχή επίλυση ενός προβλήματος.

Ένας παράλληλος αλγόριθμος σχεδιάζεται για παράλληλους υπολογιστές. Ακολουθεί ο ορισμός ενός παράλληλου υπολογιστή για να μπορέσουμε να καταλάβουμε καλύτερα τον ορισμό.

Παράλληλος Υπολογιστής είναι μία συλλογή από επεξεργαστές (συνήθως του ιδίου τύπου) διασυνδεδεμένους με συγκεκριμένο τρόπο ώστε να επιτρέπουν τον συντονισμό των δραστηριοτήτων και την ανταλλαγή δεδομένων [2]. Οι επεξεργαστές βρίσκονται σε κοντινή απόσταση ο ένας με τον άλλο και συνεργάζονται για την επίλυση των διάφορων προβλημάτων. Οι παράλληλοι υπολογιστές χωρίζονται σε δύο κατηγορίες (1) στα συστήματα πολλών υπολογιστών που είναι κατασκευασμένα από ένα αριθμό ανεξάρτητων υπολογιστών που επικοινωνούν μεταξύ τους μέσω της ανταλλαγής μηνυμάτων και (2) στους συμμετρικούς πολυεπεξεργαστές που είναι ολοκληρωμένα συστήματα πολλών επεξεργαστών που μοιράζονται την ίδια κοινή μνήμη.

2.3 Κριτήρια Αξιολόγησης Παράλληλων Αλγορίθμων

Για ανάλυση της πολυπλοκότητας ενός αλγορίθμου πρέπει να αναλύσουμε το πλήθος τον επεξεργαστών, τον παράλληλο χρόνο εκτέλεσης και το κόστος του αλγορίθμου. Οι μετρικές αυτές περιγράφονται πιο κάτω [2]:

- *Παράλληλος Χρόνος Εκτέλεσης: T(n)*
  Είναι ο χρόνος που χρειάζονται οι επεξεργαστές δουλεύοντας παράλληλα, να επιλύσουν ένα πρόβλημα μεγέθους *n*.

- *Πλήθος Επεξεργαστών P(n) = p*
  Ο αριθμός των επεξεργαστών που χρησιμοποιήθηκαν παράλληλα για να επιλύσουν ένα πρόβλημα μεγέθους *n*.

- *Κόστος: C(n) = Tp(n) * P(n)*

  Είναι το γινόμενο του παράλληλου χρόνου εκτέλεσης επί τον αριθμό των επεξεργαστών που χρησιμοποιήθηκαν για να επιλυθεί παράλληλα ένα πρόβλημα μεγέθους *n*.

Ας μελετήσουμε ένα απλό παράδειγμα για να δούμε και στην πράξη πως εφαρμόζονται οι μετρικές για την ανάλυση της πολυπλοκότητας ενός αλγορίθμου.

Πρόβλημα: Έχουμε ένα πίνακα *n* θέσεων και πρέπει σε κάθε μία από τις *n* θέσεις του πίνακα να γράψουμε την τιμή δύο. Για την ολοκλήρωση οποιασδήποτε εγγραφής τιμής απαιτείται μία (1) μονάδα χρόνου.

Λύση:

Πρώτα παρουσιάζουμε την σειριακή λύση του αλγορίθμου:

Σειριακός Ψευδοκώδικας:

```
i := 1
Αν i <= n τότε:
        Γράψε στην i θέση την τιμή 2
        i := i + 1
```

Πλήθος Επεξεργαστών: $P(n) = 1$

Έχουμε σειριακή εκτέλεση, άρα έχουμε μόνο 1 επεξεργαστή.

Χρόνος Εκτέλεσης: $T(n) = \Theta(n)$

Αφού έχουμε 1 επεξεργαστή και n εγγραφές να γίνουν τότε χρειαζόμαστε Θ(n) χρόνο.

Κόστος Εκτέλεσης: $C(n) = T(n) * P(n)$

$= \Theta(n) * 1$

$= \Theta(n)$

Αφού έχουμε 1 επεξεργαστή και χρειαζόμαστε $\Theta(n)$ χρόνο για να εκτελέσει τις εγγραφές τότε το κόστος του αλγορίθμου μας είναι $\Theta(n)$.

Αφού έχουμε μελετήσει και κατανοήσει την σειριακή ανάλυση του αλγορίθμου μας προχωράμε στην παράλληλη ανάλυση του:

Παράλληλος Ψευδοκώδικας:

```
Κάνε παράλληλα pi = 1, ..., n
        Γράψε την τιμή 2 στην θέση [i] του πίνακα
```

Πλήθος Επεξεργαστών: $P(n) = p$

Έχουμε p πανομοιότυπους επεξεργαστές να δουλεύουν παράλληλα

Χρόνος Εκτέλεσης: $T(n) = \Theta(1)$

Κάθε εγγραφή χρειάζεται Θ(1) χρόνο

Κόστος Εκτέλεσης: $C(n) = T(n) * P(n)$

$= \Theta(1) * p$

$= p$

Αφού κάθε εγγραφή χρειάζεται Θ(1) χρόνο και έχουμε p πανομοιότυπους επεξεργαστές να δουλεύουν παράλληλα τότε το κόστος ισούται με p.

2.4 Μοντέλα Παράλληλου Υπολογισμού

Ο παράλληλος υπολογισμός αποτελεί την συνεταιριστική και ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από ένα επεξεργαστές που αποσκοπεί στη γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων

Υπάρχουν διάφορα μοντέλα υπολογισμού και διάφορες ταξινομήσεις. Η πιο διαδεδομένη ταξινόμηση στα μοντέλα υπολογισμού είναι η ταξινόμηση Flynn's και χρησιμοποιείται από το 1966. Η ταξινόμηση Flynn's διακρίνει τους υπολογιστές σύμφωνα με το πως μπορούν να ταξινομηθούν με βάση την ροή εντολών (instruction stream) και την ροή δεδομένων (data stream). Κάθε μία από τις δύο αυτές διαστάσεις έχει άλλες δύο πιθανές καταστάσεις, την απλή (single) και την πολλαπλή (multiple). Κατά συνέπεια έχουμε τέσσερις διαφορετικές κατηγορίες οι οποίες φαίνονται στο πιο κάτω Πίνακα 2.1.

	Single Instruction	Multiple Instruction
Single Date	SISD	MISD
Multiple Data	SIMD	MIMD

Πίνακας 2.1

Πρέπει όμως να ορίσουμε το τι είναι ένα ρεύμα εντολών και το τι είναι ένα ρεύμα δεδομένων για να μπορέσουμε να κατανοήσουμε την κατηγοριοποίηση.

Ένα *ρεύμα εντολών* λέει στον υπολογιστή τι να κάνει σε κάθε βήμα, είναι δηλαδή το σύνολο των εντολών που απαρτίζουν τον αλγόριθμο μας.

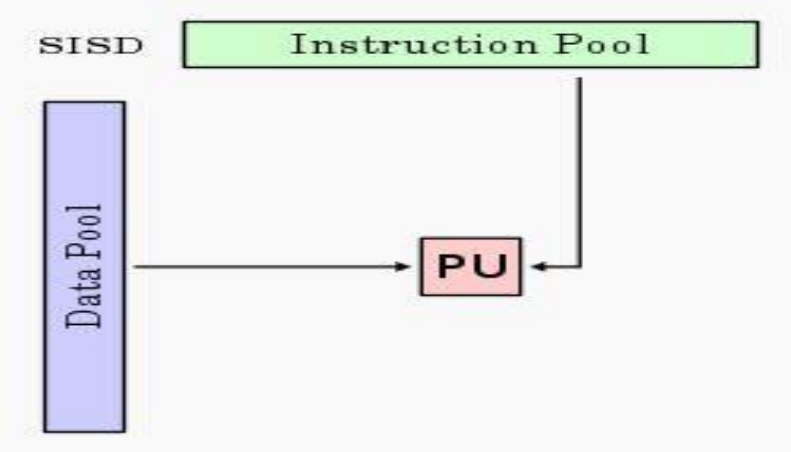
Ένα *ρεύμα δεδομένων* είναι το σύνολο των δεδομένων που επεξεργάζεται ο αλγόριθμός μας και επηρεάζονται από το ρεύμα των εντολών και διαφοροποιούνται ανάλογα.

Η ταξινόμηση αυτή βασίζεται στην λειτουργία των υπολογιστών (σειριακών και παράλληλων), που δεν είναι άλλη από την εκτέλεση εντολών πάνω σε δεδομένα.

Οι τέσσερις κατηγορίες της ταξινόμησης Flynn's είναι οι ακόλουθες [3]:

- SISD: Single Instruction stream, Single Data stream [4]

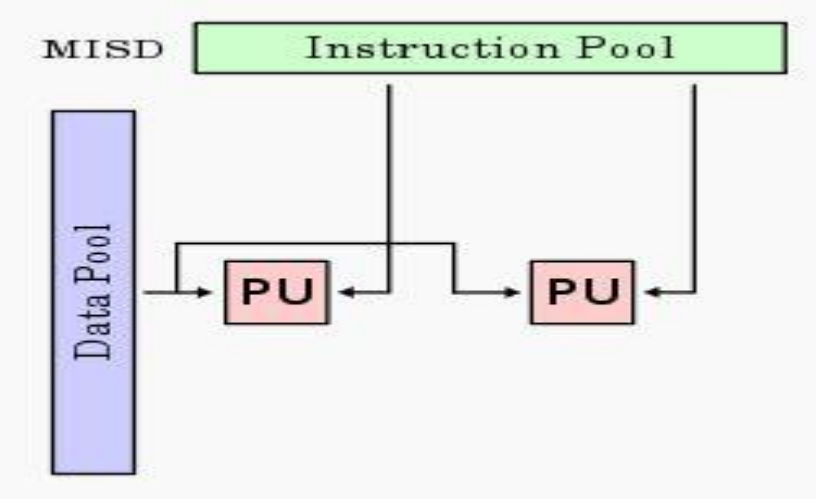
  Σε αυτό το μοντέλο υπολογισμού υπάρχει μόνο ένας επεξεργαστής άρα το μοντέλο αυτό είναι για σειριακούς υπολογισμούς. Ο ένας και μοναδικός επεξεργαστής που υπάρχει στο σύστημα φορτώνει μόνο ένα ρεύμα εντολών κάθε φορά από την μνήμη και ακολούθως εκτελεί βάσει του συγκεκριμένου ρεύματος πράξεις σε ένα μόνο ρεύμα δεδομένων. Το μοντέλο αυτό απεικονίζεται στο Σχήμα 2.3.



` Σχήμα 2.3 [4]

- MISD: Multiple Instruction stream, Single Data stream [6]

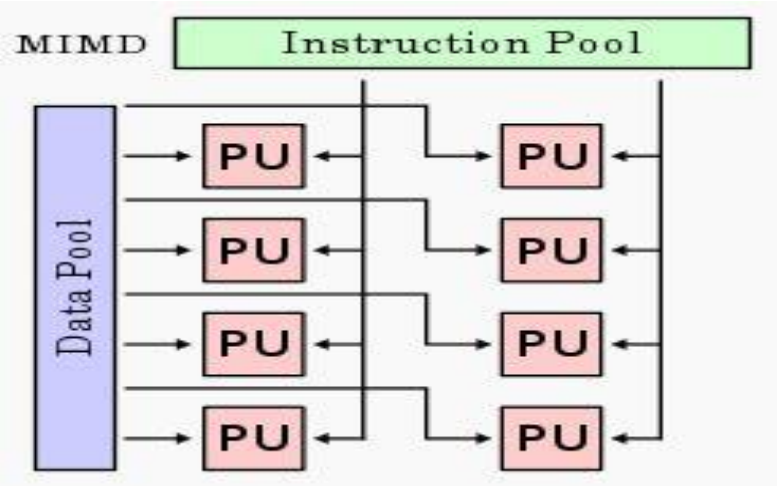
  Σε αυτό το μοντέλο έχουμε N επεξεργαστές όπου ο κάθε ένας έχει την δική του μονάδα ελέγχου και μοιράζονται μία κοινή μνήμη στην οποία βρίσκονται τα δεδομένα. Λόγω του ότι κάθε επεξεργαστής έχει δική του μονάδα ελέγχου σε κάθε βήμα οι επεξεργαστές μπορούν να εκτελούν διαφορετική εντολή πάνω στο ίδιο όμως δεδομένο. Δεν έχει κατασκευαστεί αρχιτεκτονική βάσει αυτού του μοντέλου γιατί υπάρχει το μοντέλο MIMD (Multiple Instruction stream, Multiple Data stream) το οποίο υπερκαλύπτει αυτό το μοντέλο. Το μοντέλο αυτό απεικονίζεται στο σχήμα 2.4.



Σχήμα 2.4 [6]

- MIMD: Multiple Instruction stream, Multiple Data stream [7]

  Σε αυτό το μοντέλο έχουμε Ν επεξεργαστές, Ν ρεύματα εντολών και Ν ρεύματα δεδομένων. Κάθε επεξεργαστής δουλεύει σε διαφορετικό ρεύμα εντολών και σε διαφορετικό ρεύμα δεδομένων. Άρα οι επεξεργαστές μπορούν να εκτελέσουν διαφορετικά προγράμματα πάνω σε διαφορετικά δεδομένα ώστε να επιλύσουν διαφορετικά υπο-προβλήματα του ιδίου προβλήματος ή και διαφορετικά προβλήματα. Σε σύγκριση με τα άλλα μοντέλα το μοντέλο αυτό είναι το πιο πολύπλοκο. Το μοντέλο αυτό απεικονίζεται στο Σχήμα 2.5.

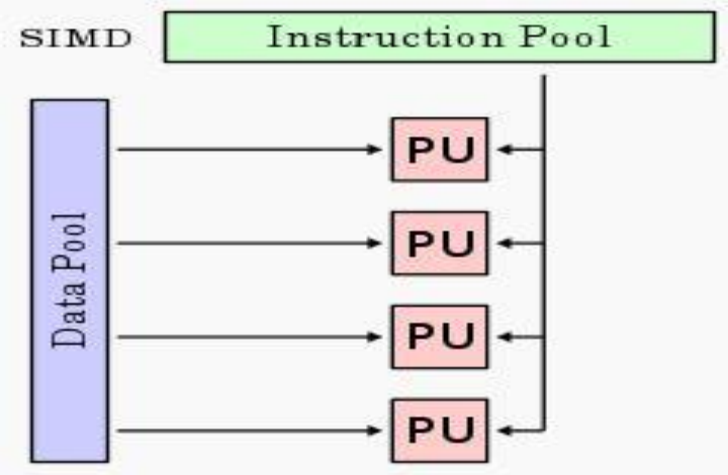


Σχήμα 2.5 [7]

- SIMD: Single Instruction stream, Multiple Data stream [5]

  Σε αυτό το μοντέλο υπολογισμού έχουμε Ν πανομοιότυπους επεξεργαστές όπου ο κάθε ένας έχει τη δική του τοπική μνήμη και διαφορετικό ρεύμα δεδομένων. Στο μοντέλο αυτό οι επεξεργαστές λειτουργούν συγχρονισμένα δηλαδή είτε σε κάθε βήμα όλοι οι επεξεργαστές εκτελούν την ίδια εντολή, πάνω σε διαφορετικό δεδομένο είτε

όταν μόνο ένα υποσύνολο των επεξεργαστών είναι αναγκαίο να εκτελέσει μία εντολή οι υπόλοιποι εκτελούν την εντολή no-op(no operation). Για την επικοινωνία των επεξεργαστών υπάρχουν δύο τρόποι είτε μέσω κοινόχρηστης μνήμης είτε μέσω δικτύου αλληλοσύνδεσης. Στην παρούσα εργασία θα ασχοληθούμε μόνο με την επικοινωνία μέσω κοινόχρηστης μνήμης. Το μοντέλο αυτό απεικονίζεται στο Σχήμα 2.6.



Σχήμα 2.6 [5]

2.5 Μοντέλο PRAM

Το μοντέλο PRAM (Parallel Random Access Machine) [8] είναι μηχανή τυχαίας προσπέλασης που χρησιμοποιείται στον παράλληλο υπολογισμό και αποτελεί κατηγορία του μοντέλου παράλληλου υπολογισμού SIMD(Single Instruction Multiple Data) με κοινόχρηστη μνήμη.

Στο μοντέλο PRAM έχουμε *p* πανομοιότυπους και συγχρονισμένους επεξεργαστές όπου ο κάθε επεξεργαστής διαθέτει τοπική/προσωπική μνήμη με χρόνο πρόσβασης Θ(1). Ως μέσο επικοινωνίας οι επεξεργαστές χρησιμοποιούν την κοινόχρηστη μνήμη. Ένας επεξεργαστής γράφει το μήνυμα σε μία συγκεκριμένη κυψελίδα της κοινόχρηστης μνήμης και στην συνέχεια άλλος επεξεργαστής διαβάζει το μήνυμα από αυτή την κυψελίδα. Τα δεδομένα εισόδου, τα ενδιάμεσα αποτελέσματα και τα δεδομένα εξόδου αποθηκεύονται στη κοινόχρηστη μνήμη μέσω κοινόχρηστων μεταβλητών/δομών.

Το βασικό μοντέλο PRAM επιτρέπει στους επεξεργαστές ταυτόχρονη πρόσβαση στην κοινόχρηστη μνήμη, φτάνει οι κυψελίδες μνήμης που προσπαθούν να διαβάσουν ή να γράψουν να είναι διαφορετικές. Με βάση τους περιορισμούς ταυτόχρονης πρόσβασης στην μνήμη έχουμε τέσσερις διαφορετικούς τύπους PRAM οι οποίοι είναι [2] [8]:

- EREW: Exclusive Read, Exclusive Write

  Είναι το πιο περιοριστικό μοντέλο. Στο μοντέλο αυτό δεν μπορούν περισσότεροι από έναν επεξεργαστή να διαβάσουν ή να γράψουν την ίδια χρονική στιγμή στην ίδια κυψελίδα μνήμης. Στην περίπτωση ταυτόχρονης πρόσβασης στην μνήμη το πρόγραμμα τερματίζει.

- CREW: Concurrent Read, Exclusive Write

  Όλοι οι επεξεργαστές μπορούν να διαβάσουν από την ίδια κυψελίδα ταυτόχρονα αλλά μόνο ένας επεξεργαστής μπορεί να γράψει στην ίδια κυψελίδα.

- ERCW: Exclusive Read, Concurrent Write

  Όλοι οι επεξεργαστές έχουν την δυνατότητα να γράψουν στην ίδια κυψελίδα ταυτόχρονα, αλλά μόνο ένας έχει την δυνατότητα να διαβάσει από την ίδια κυψελίδα μνήμης. Το μοντέλο αυτό δεν παρουσιάζει κανένα θεωρητικό ή πρακτικό ενδιαφέρον.

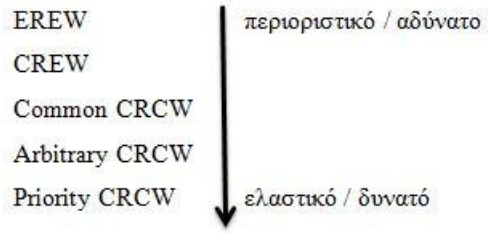
- CRCW: Concurrent Read, Concurrent Write

  Είναι ισχυρή μηχανή στην οποία δεν υπάρχουν περιορισμοί στην ταυτόχρονη πρόσβαση στην μνήμη. Δηλαδή, όλοι οι επεξεργαστές έχουν την δυνατότητα να γράψουν στην ίδια κυψελίδα μνήμης και όλοι οι επεξεργαστές έχουν την δυνατότητα να διαβάσουν από την ίδια κυψελίδα μνήμης.

Από τα πιο πάνω είναι φανερό πως το ταυτόχρονο διάβασμα από την μνήμη δεν δημιουργεί προβλήματα γιατί κάθε επεξεργαστής αντιγράφει το περιεχόμενο της κυψελίδας στην προσωπική του μνήμη. Πρόβλημα προκύπτει όταν έχουμε ταυτόχρονο γράψιμο στην μνήμη και γι' αυτό στην περίπτωση του CRCW PRAM έχουμε τρεις (3) κατηγορίες:

- Common (κοινό) CRCW PRAM: Όλοι οι επεξεργαστές πρέπει να γράψουν την ίδια τιμή.
- Arbitrary (αυθαίρετο) CRCW PRAM: Ένας από τους επεξεργαστές αυθαίρετα επιλεγμένος καταφέρνει να γράψει την δική του τιμή.
- Priority (προτεραιότητα) CRCW PRAM: Ο επεξεργαστής με την μεγαλύτερη προτεραιότητα γράφει την δική του τιμή.

Τα διαφορετικά υπομοντέλα του PRAM μπορούν να ιεραρχηθούν με βάση την δυναμικότητα τους. Η ιεραρχία παρουσιάζεται πιο κάτω στο Σχήμα 2.7.



Σχήμα 2.7

Ένας αλγόριθμος που έχει σχεδιαστεί για ένα αδύνατο μοντέλο PRAM μπορεί να εκτελεστεί με την ίδια πολυπλοκότητα χρόνου και κόστους σε ένα πιο δυνατό μοντέλο. Το αντίθετο, δηλαδή η εκτέλεση ενός αλγορίθμου που έχει σχεδιαστεί για ένα δυνατό μοντέλο μπορεί να εκτελεστεί σε ένα πιο αδύνατο μοντέλο είτε με ασυπτωτικά περισσότερους επεξεργαστές ή με ασυπτωτικά μεγαλύτερο χρόνο εκτέλεσης. Κατά συνέπεια όταν ένας προγραμματιστής σχεδιάζει ένα αλγόριθμο πρέπει να έχει υπόψη του σε ποιο μοντέλο θα εκτελεστεί ο αλγόριθμος καθώς και τους περιορισμούς του μοντέλου.

Κεφάλαιο 3

Πλατφόρμα XMT

3.1 Πλατφόρμα XMT

Η ιδέα της πλατφόρμας εφευρέθηκε το 1998 από τον καθηγητή Uzi Vishkin και υλοποιήθηκε από τον ίδιο και από την ομάδα του το 2007. Κατάφεραν να υλοποιήσουν ένα PRAM-On-Chip 64-Core 75MHz FPGA. Πιο συγκεκριμένα, η πλατφόρμα XMT [11] υλοποιεί το μοντέλο παράλληλου υπολογισμού Arbitary CRCW SIMD με την εφαρμογή της μεθοδολογίας Work-Depth κατά την δημιουργία του αλγόριθμου–κώδικα.

Το όραμα ήταν η δημιουργία PRAM-On-Chip που επιδίωκε την δημιουργία ενός εύκολου και εύχρηστου μοντέλου που θα μπορούσε να προγραμματίζει παράλληλα ενώ θα είχαμε στην διάθεση μας ένα τσιπ που θα αποτελείτο από χιλιάδες επεξεργαστές. Η δυσκολία ήταν στο να βρεθεί ο τρόπος όπου θα γινόταν η μετατροπή ενός PRAM αλγορίθμου σε έμπρακτο αλγόριθμο XMT. Οι αλγόριθμοι που είναι προγραμματισμένοι για PRAM ακολουθούν την δομή Work-Depth όπως έχει αναφερθεί και προηγουμένως. Παρόλο που κάθε επεξεργαστής έχει την δική του ταχύτητα, η δομή αυτή καθορίζει ότι οι επεξεργαστές θα λειτουργήσουν συγχρονισμένα και με τον ίδιο ρυθμό ανά γύρο.

Η πλατφόρμα XMT είναι μία μηχανή με 64 επεξεργαστές που βασίζεται στην τεχνολογία FPGA. Είναι υλοποιημένοι με τρία FPGA τσιπς και μία κάρτα MTCU, παράλληλα κι

πολλαπλά TCU, ένα διαχειριστή μνήμης, μια prefix unit και καθολικούς καταχωρητές. Η αναβάθμιση της πλατφόρμας αποτελείται από 128 επεξεργαστές, από τρία τσιπς FPGA, δύο Virtex-4 LX200 και ένα Virtex-4 FX100. Τα υπόλοιπα στοιχεία παραμένουν τα ίδια. Η επέκταση αυτή είναι πολύ σημαντική αφού δίνει τα επιθυμητά αποτελέσματα σε λιγότερο χρόνο. Η περαιτέρω ανάπτυξη της πλατφόρμα είναι επιθυμητή ώστε να γίνει πιο ευέλικτη και όσο το δυνατό πιο γρήγορη. Η ανάπτυξη όμως θέλουμε να έχει όσο το δυνατό λιγότερο κόστος.

Η ροή εκτέλεσης στο XMT μοντέλο εναλλάσσεται μεταξύ σειριακής και παράλληλης εκτέλεσης. Η αρχή της παράλληλης εκτέλεσης ορίζεται από το spawn και τελειώνει με το join. Η εντολή αυτή αναλύεται αναλυτικά στο Υποκεφάλαιο 3.2.

Η γλώσσα προγραμματισμού της πλατφόρμας είναι η XMTC, η οποία είναι προέκταση της γλώσσας προγραμματισμού C. Η γλώσσα προγραμματισμού XMTC αναλύεται αναλυτικά στο Υποκεφάλαιο 3.2.

3.2 Γλώσσα Προγραμματισμού XMTC

Ο προγραμματισμός των παράλληλων αλγορίθμων γίνεται στην γλώσσα προγραμματισμού XMTC. Μπορεί η XMTC να είναι σχετικά καινούργια γλώσσα, αλλά έχει ως βάση της γλώσσα προγραμματισμού C, αφού είναι και η προέκταση της C. Έτσι η εκμάθηση της είναι σχετικά εύκολη αφού το μόνο καινούργιο είναι οι προεκτάσεις και οι περιορισμοί που έχουν προστεθεί. Αξίζει να σημειωθεί πως λόγω της ομοιότητας των δύο γλωσσών ένα σειριακό πρόγραμμα γραμμένο στην C μπορεί να τρέξει με τον προσομοιωτή της γλώσσας XMTC, αλλά το αντίθετο δεν είναι δυνατό λόγω των επιπλέον περιορισμών και προεκτάσεων. Ένα πρόγραμμα σε XMTC κώδικα χωρίζεται σε δύο κύρια κομμάτια, το σειριακό και το παράλληλο. Το σειριακό κομμάτι δεν διαφέρει σε τίποτα από ένα γνωστό σειριακό πρόγραμμα γραμμένο στην C. Το παράλληλο κομμάτι εισάγεται με την εντολή spawn (Η εντολή αυτή αναλύεται σε επόμενο υποκεφάλαιο). Γενικά, στην αρχή γίνεται η ενσωμάτωση των βιβλιοθηκών και ακολουθεί η συνάρτηση main(). Οι δηλώσεις των μεταβλητών όπως και στα σειριακά προγράμματα μπορούν να γίνουν σε οποιοδήποτε σημείο του κώδικα. Υπάρχουν αρκετά προβλήματα και περιορισμοί τα οποία γίνεται προσπάθεια από την ερευνητική ομάδα που ασχολείται με το θέμα αυτό να τα εξαλείψει.

3.2.1 Main()

Όπως όλα τα σειριακά προγράμματα έτσι και τα παράλληλα προγράμματα απαιτούν την κλίση της συνάρτησης main(). Μέσα στην συνάρτηση βρίσκεται ολόκληρο το πρόγραμμα μας και η κλίση της συνάρτησης αυτής είναι απαραίτητη για να ορίζεται το πρόγραμμα μας. Οι εντολές καθώς και η δήλωση μεταβλητών ή συναρτήσεων πρέπει να είναι στα επιτρεπτά όρια της γλώσσας προγραμματισμού XMTC. Έξω από την συνάρτηση μπορούμε να ορίσουμε μόνο συναρτήσεις και μεταβλητές.

3.2.2 Spawn()

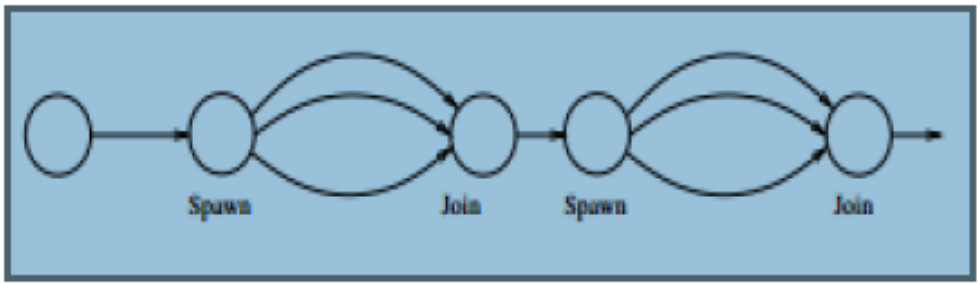
Μέσα στην εντολή αυτή περιέχεται το παράλληλο κομμάτι του αλγορίθμου μας. Η εντολή υποδηλώνει ότι σταματά η σειριακή εκτέλεση του αλγορίθμου και ξεκινά η παράλληλη λειτουργία του. Πολλοί επεξεργαστές ξεκινούν να τρέχουν ταυτόχρονα για να λύσουν το συγκεκριμένο υπο-πρόβλημα που βρίσκεται μέσα στο μπλοκ της εντολής. Το τέλος της παράλληλης εκτέλεσης υποδηλώνεται με το τέλος της εντολής(join).

Οι επεξεργαστές παρόλο που δουλεύουν παράλληλα δεν λειτουργούν συγχρονισμένα. Κάθε επεξεργαστής εκτελεί τις εντολές που έχει να εκτελέσει με την δική του ταχύτητα αλλά όλοι οι επεξεργαστές έχουν να εκτελέσουν τις ίδιες εντολές σε κάθε παράλληλο κομμάτι του κώδικα. Λόγω του ότι κάθε επεξεργαστής έχει τους δικούς του ρυθμούς όταν τελειώσουν όλοι την εκτέλεση των εντολών τους πρέπει να γίνει συγχρονισμός για να μπορέσουν να προχωρήσουν στο επόμενο βήμα. Ο συγχρονισμός αυτός επιτυγχάνεται με την εντολή join. Η εντολή ορίζεται ως

```
spawn(start thread, end thread)
{
        parallel section;
}
```

Το start thread και το end thread είναι ακέραιες τιμές(αφού οι επεξεργαστές ορίζονται μόνο με ακέραιες τιμές) που ορίζουν ποιοι επεξεργαστές θα εργαστούν παράλληλα για το μπλοκ που ακολουθεί. Ο πρώτος επεξεργαστής που θα δουλέψει παράλληλα με τους υπόλοιπους είναι ο επεξεργαστής με ταυτότητα την τιμή του start thread ενώ ο τελευταίος επεξεργαστής που θα δουλέψει παράλληλα με τους υπόλοιπους είναι ο επεξεργαστής με ταυτότητα την τιμή του end thread.

Το πιο κάτω σχεδιάγραμμα μας παρουσιάζει το πως δουλεύει η εντολή spawn σε ένα παράλληλο κώδικα



3.2.3 Βιβλιοθήκες

Η κύρια βιβλιοθήκη είναι η βιβλιοθήκη <xmtc.h>, όπου χωρίς την χρήση της ο κώδικας μας δεν μπορεί να μεταγλωττιστεί. Με την εισαγωγή της βιβλιοθήκης ο χρήστης εισάγει ταυτόχρονα όλες τις εντολές και πληροφορίες που χρειάζονται για την παράλληλη μεταγλώττιση. Η βιβλιοθήκη <xmtio.h> εισάγεται αυτόματα και επιτρέπει ο χρήστης να τυπώνει στην κονσόλα. Επίσης, επιτρέπεται η εισαγωγή βιβλιοθηκών από τον χρήστη τις οποίες έχει δημιουργήσει ο ίδιος και έχει αποθηκεύσει μεταβλητές. Η εισαγωγή αυτών των βιβλιοθηκών μπορεί να γίνει με την εντολή #include απευθείας από το πρόγραμμα ή με την εντολή –include από την κονσόλα.

3.2.4 Μεταβλητές

Οι μεταβλητές σε ένα παράλληλο κώδικα χωρίζονται στις καθολικές και στις τοπικές όπως και σε ένα σειριακό κώδικα.

Καθολικές Μεταβλητές: Είναι μεταβλητές στις οποίες έχουν πρόσβαση όλοι οι επεξεργαστές από οποιοδήποτε σημείο του κωδικά μας. Οι μεταβλητές αυτές δηλώνονται σε οποιοδήποτε σημείο του κωδικά μας ή εισάγονται στο πρόγραμμα μας μέσω του εργαλείου μνήμης.

Τοπικές Μεταβλητές: Οι τοπικές μεταβλητές χωρίζονται σε δύο υπο-κατηγορίες. Στην μία υπο-κατηγορία ανήκουν οι μεταβλητές των οποίων το εύρος τους είναι μόνο μέσα στο μπλοκ που ορίζονται. Στην άλλη κατηγορία ανήκουν οι μεταβλητές που ορίζονται στα παράλληλα κομμάτια του κώδικα και πιο συγκεκριμένα ορίζονται μέσα σε ένα spawn block. Οι μεταβλητές αυτές είναι ιδιωτικές για κάθε επεξεργαστή, αποθηκεύονται στην προσωπική του μνήμη και δεν είναι προσβάσιμες από άλλο επεξεργαστή.

Στην περίπτωση που ορίσουμε μια τοπική και μια καθολική μεταβλητή με το ίδιο όνομα θα υπάρξει σύγχυση και γι’αυτό πρέπει να είμαστε προσεκτικοί κατά τον ορισμό των μεταβλητών.

Μέχρι στιγμής, οι μόνοι τύποι μεταβλητών που επιτρέπονται να οριστούν είναι int και long μεγέθους 32 bit. Δηλαδή οι μεταβλητές μας μπορούν να είναι μόνο ακέραιοι αριθμοί. Επίσης, μπορούμε να ορίσουμε μονοδιάστατους και δισδιάστατους πίνακες.

3.2.5 Σύμβολο $

Κάθε επεξεργαστής έχει ένα μοναδικό αναγνωριστικό το οποίο είναι ακέραιος αριθμός. Το αναγνωριστικό αυτό του κάθε επεξεργαστή ορίζεται από το σύμβολο αυτό και δίνει στον χρήστη την δυνατότητα να έχει πρόσβαση στον επεξεργαστή και να εκτελεί εντολές με βάση το αναγνωριστικό του. Το σύμβολο χρησιμοποιείται για το κάλεσμα τον επεξεργαστών. Επίσης, η τιμή του συμβόλου δεν μπορεί να αλλάξει και ούτε η τιμή του ανατίθεται από τον χρήστη.

3.2.6 Συναρτήσεις κλίσεις συστήματος

Η γλώσσα XMTC δεν υποστηρίζει την κλίση συνάρτησης μέσα σε spawn block. Αν γίνει κλίση συνάρτησης μέσα σε παράλληλο τμήμα κώδικα θα παρουσιαστεί πρόβλημα κατά την εκτέλεση του προγράμματος. Επιτρέπεται η συγγραφή συναρτήσεων με βάση τους κανόνες συγγραφής στην C αλλά το κάλεσμα το συναρτήσεων δεν επιτρέπεται στα παράλληλα κομμάτια του κώδικα. Επίσης, λόγω του ότι δεν υποστηρίζονται αρκετές βιβλιοθήκες είναι περιορισμένες και οι συναρτήσεις που μπορούμε να καλέσουμε. Για παράδειγμα, λόγω του ότι δεν υποστηρίζεται η βιβλιοθήκη <math.h> δεν μπορούμε να καλέσουμε καμία από τις συναρτήσεις που υποστηρίζει η βιβλιοθήκη αυτή. Η συνάρτηση printf() μπορεί μεν να υποστηρίζεται στο σειριακό τμήμα κώδικα αλλά δεν υποστηρίζεται στο παράλληλο τμήμα κώδικα. Για την χρήση της συνάρτησης δεν απαιτείται η ενσωμάτωση καμίας βιβλιοθήκης από τον προγραμματιστή γιατί γίνεται αυτόματη ενσωμάτωση της απαραίτητης βιβλιοθήκης.

3.2.7 XMT Serializer

Η πλατφόρμα XMT έχει αρκετά λάθη και αδυναμίες λόγω του ότι δεν είναι στο τελικό της στάδιο. Δημιουργούνται προβλήματα τα οποία δεν είναι ανιχνεύσιμα ούτε από τον μεταγλωττιστή ούτε από τον προσομοιωτή αλλά ούτε και από τον ίδιο τον προγραμματιστή. Είναι αναγκαίο λοιπόν να γίνεται έλεγχος για τα προβλήματα αυτά. Ο έλεγχος γίνεται με το εργαλείο σειριοποίησης (XMT Serializer). Το εργαλείο αυτό δημιουργεί ένα αντίγραφο του κώδικα μας και έτσι ο αρχικός παράλληλος μας κώδικας παραμένει αναλλοίωτος. Γίνεται μετατροπή του παράλληλου κώδικα σε σειριακό και τρέχει σαν κανονικό πρόγραμμα σε γλώσσα προγραμματισμού C. Για την διαδικασία αυτή ο χρήστης πρέπει απλά να δώσει δύο εντολές στην κονσόλα:

```
xmtcser mycode.c
```

gcc mycode.serialized.c –o mycode
./mycode

3.2.8 Μεταγλώττιση και Προσομοίωση Προγράμματος σε XMTC

Για να τρέξει ένα πρόγραμμα πρώτα πρέπει να μεταγλωττιστεί (compile) και μετά να προσομοιωθεί (simulate) ο κώδικας του.

- **Μεταγλώττιση κώδικα**

  Η μεταγλώττιση γίνεται όταν ο χρήστης καλεί τον μεταγλωττιστή μέσω της εντολής xmtcc. Η εντολή δίνεται από τον χρήστη από την κονσόλα.
  Εντολή: xmtcc my_program.c (όπου my_program.c το όνομα του προγράμματος)

  Στην περίπτωση όπου θα εισάγουμε δεδομένα μέσω του εργαλείου μνήμης πρέπει να δώσουμε και το όνομα του αρχείου μας στην κονσόλα. Η εντολή μας θα διαφοροποιηθεί.
  Εντολή: xmtcc my_program.c my_data.xbo (όπου my_program.c το όνομα του προγράμματος και όπου my_data.xbo το όνομα του εισαγόμενου αρχείου)

- **Προσομοίωση κώδικα**

  Η προσομοίωση ενός κώδικα μπορεί να γίνει με δύο τρόπους όπου και στις δύο περιπτώσεις η εντολή δίνεται από την κονσόλα. Οι εντολές είναι οι ίδιες και στην περίπτωση που γίνεται εισαγωγή δεδομένων και στην περίπτωση που δεν γίνεται εισαγωγή δεδομένων.

  Ο πρώτος τρόπος είναι με assembly simulation όπου εκτελείται σειριακή προσομοίωση.
  Εντολή: xmtsim –binload a.b a.sim

  Ο δεύτερος τρόπος είναι με cycle-accurate simulation όπου είναι παράλληλη εκτέλεση και όχι σειριακή όπως τον προηγούμενο τρόπο.
  Εντολή: xmtsim –cycle –binload a.b a.sim

3.3 Εργαλείο Μνήμης για Δεδομένα Εισόδου

Μέχρι στιγμής δεν υπάρχει η δυνατότητα ο προσομοιωτής XMT να χειριστεί κλίσεις στο λειτουργικό σύστημα. Έτσι δεν υπάρχει η δυνατότητα να γίνεται εισαγωγή αρχείων από

κλίσεις του συστήματος. Η εισαγωγή δεδομένων σε ένα XMTC πρόγραμμα γίνεται μόνο μέσω του εργαλείου μνήμης memMapCreate.

Το εργαλείο αυτό παρέχει την δυνατότητα στατικής δέσμευσης και αρχικοποίησης μεταβλητών και την δημιουργία αρχείων που περιέχουν πληροφορίες που αφορούν τα δεδομένα εισόδου.

Γι κάθε μεταβλητή που χρειάζεται ο προγραμματιστής πρέπει να δώσει όνομα στην μεταβλητή, να ορίσει το μέγεθος της και να επιλέξει πως θα αρχικοποιήσει την μεταβλητή με ένα από τους πιο κάτω τρόπους:

(1) 0 (για μεταβλητές / για όλα τα στοιχεία του πίνακα)
(2) Σταθερή τιμή (για μεταβλητές)
(3) Με τυχαίες τιμές από 0 μέχρι 1 για πίνακες κινητής υποδιαστολής (float table) και με τιμές από 0 μέχρι έναν αριθμό που επιλέγει ο χρήστης για ακέραιους πίνακες (integer table
(4) Με αρχείο που περιέχει τιμή για κάθε θέση του πίνακα (για όλους τους τύπους πίνακα)

Με την προσωρινή μέθοδο που έχει δημιουργηθεί οι χρήστες μπορούν να δουλεύουν με εξωτερικά δεδομένα και τους επιτρέπει να δεσμεύσουν μνήμη στατικά. Κατά την διαδικασία αυτή δημιουργούνται τρία διαφορετικά αρχεία τα οποία περιγράφονται πιο κάτω:

(1) Header file (.h): Το αρχείο αυτό περιέχει τις δηλώσεις των κοινόχρηστων μεταβλητών που έχουν δεσμευτεί στην μνήμη. Το αρχείο αυτό λειτουργεί ως βιβλιοθήκη και γι'αυτό μπορεί να ενσωματωθεί στον κώδικα με τον ίδιο τρόπο που ενσωματώνονται οι βιβλιοθήκες (#include). Επίσης, μπορεί να ενσωματωθεί στον κώδικα μέσω της κονσόλας με την εντολή –include.

(2) Binary file (.xbo): Το αρχείο αυτό περιέχει τις αρχικοποιήσεις των μεταβλητών που έχουν οριστεί στο header file. Ο τρόπος αρχικοποίησης των μεταβλητών και των πινάκων περιγράφεται στην προηγούμενη παράγραφο. Το αρχείο αυτό πρέπει να δοθεί ως είσοδο στον μεταγλωττιστή κατά την διάρκεια μεταγλώττισης του προγράμματος.

(3) Text file (.txt): Το αρχείο αυτό περιέχει τα περιεχόμενα του binary file σε δεκαδική μορφή. . Δεν χρησιμοποιείται από τον προσομοιωτή ή από την μηχανή FPGA. Χρησιμοποιείται μόνο για σκοπούς ελέγχου από τον προγραμματιστή.

3.4 Περιορισμοί

Η γλώσσα παρουσιάζει αρκετά προβλήματα και περιορισμούς τα οποία μελετώνται και γίνονται προσπάθειες βελτίωσης τους. Αρκετά από τα προβλήματα αναφέρονται σε διάφορα σημεία στην παρούσα Διπλωματική Εργασία. Πιο κάτω αναφέρονται τα προβλήματα που εντόπισα:

- Κάθε spawn block μπορεί να έχει ορισμένες μέχρι και 100 εντολές
- Σε κάθε παράλληλο κομμάτι είναι περιορισμένος ο αριθμός τοπικών μεταβλητών που μπορούν να οριστούν λόγω του περιορισμού μνήμης.
- Δεν επιτρέπεται η χρήση της συνάρτησης random() για την παραγωγή τυχαίων αριθμών.
- Δεν επιτρέπεται το κάλεσμα συναρτήσεων μέσα σε spawn block.
- Δεν επιτρέπονται τα φωλιασμένα spawn.
- Οι παράμετροι της συνάρτησης spawn πρέπει να είναι ακέραιοι αριθμοί.
- Δεν επιτρέπεται η χρήση αρκετών βιβλιοθηκών.
- Δεν επιτρέπεται η δυναμική δέσμευση πίνακα.
- Οι μεταβλητές επιτρέπεται να είναι μόνο ακέραιοι.
- Δεν επιτρέπεται το τύπωμα μέσα στο παράλληλο τμήμα κώδικα

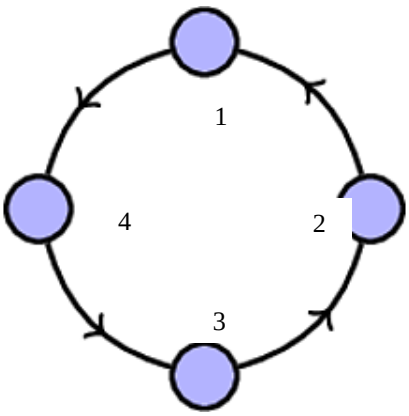
Κεφάλαιο 4

Κατευθυνόμενοι Δακτύλιοι

4.1 Πρόβλημα

Ως εισαγόμενα ο αλγόριθμος έχει ένα κατευθυνόμενο δακτύλιο $G = (V,E)$ του οποίου οι ακμές αναπαριστώνται με ένα πίνακα S μήκους $|V| = n$, όπου $S(i) = j$ εάν και μόνο αν $<i,j>$ είναι ακμή στο Ε. Σκοπός του αλγορίθμου είναι ο υπολογισμός ενός ανεξάρτητου συνόλου από κορυφές $X=\{ v \in V \mid label(v) = 1\}$, το οποίο αποτελεί ένα φραγμένο ανεξάρτητο σύνολο με μεγάλη πιθανότητα. Το μέγεθος του X είναι σταθερά φραγμένο από το $|V|$.

Παράδειγμα:
Στο σχήμα 4.1 παρουσιάζεται ένας κατευθυνόμενος δακτύλιος με 4 κόμβους.



Σχήμα 4.1

Τα πιθανά ανεξάρτητα σύνολα από τον δακτύλιο είναι μεγέθους 1 ή 2 και είναι:

1) $X = \{1\}$
2) $X = \{2\}$
3) $X = \{3\}$
4) $X = \{4\}$
5) $X = \{1,3\}$
6) $X = \{2,4\}$
7) $X = \{3,1\}$
8) $X = \{4,2\}$

4.2 Παράλληλος Αλγόριθμος

Ο παράλληλος αλγόριθμος για την εύρεση ανεξάρτητου συνόλου σε κατευθυνόμενο κύκλο είναι ο ακόλουθος [1]:

begin

for all $v \in V$ **pardo**

1. Assign label(v) = 1 or 0 randomly with equal probability

2.if (label(v) = 1 and label(S(v)) = 1) then set label(v) := 0

end

Ο πιο πάνω αλγόριθμος (Randomized Symmetry Breaking) αποτελείται από δύο βήματα. Όλοι οι επεξεργαστές δουλεύουν παράλληλα και στα δύο βήματα. Στο πρώτο βήμα γίνεται τυχαία ανάθεση τιμής 1 ή 0 σε κάθε κόμβο. Στην πράξη λόγω των περιορισμών της πλατφόρμα XMT η υλοποίηση του βήματος αυτού γίνεται στην C και εισάγονται τα δεδομένα μέσω του εργαλείου μνήμης στο παράλληλο πρόγραμμα μας. Στο δεύτερο βήμα δεν υπάρχει κάποιος περιορισμός. Γίνεται έλεγχος αν η τιμή του κόμβου και η τιμή του απόγονου του συγκεκριμένου κόμβου ισούται με ένα, τότε μηδενίζεται η τιμή του κόμβου μας. Μετά το τέλος της εκτέλεσης του αλγορίθμου όσοι από τους κόμβους εξακολουθούν να έχουν τιμή 1 αποτελούν και το ανεξάρτητο σύνολο μας.

Η βασική τεχνική που χρησιμοποιείται είναι η τεχνική Randomized Symmetry Breaking η οποία αποτελείται από (1) ανάθεση της τιμής 0 ή 1 με ίση πιθανότητα σε κάθε κορυφή $v \in V$ (2) ξανά αναθέτει τιμή σε κάθε κορυφή v της οποίας η τιμή και η τιμή του απογόνου της S(v) είναι και οι δύο ίσες με 1.

Μοντέλο PRAM:

Ο αλγόριθμος μπορεί να υλοποιηθεί στο μοντέλο EREW PRAM λόγω του ότι το βήμα 1 δεν απαιτεί ταυτόχρονη πρόσβαση στην μνήμη και το βήμα 2 μπορεί να εκτελεστεί αφού δημιουργήσουμε ένα αντίγραφο του πίνακα μας με τις τυχαίες ετικέτες (που δημιουργήσαμε στο βήμα 1) και τότε μπορούμε να έχουμε πρόσβαση στο αντίγραφο όταν χρειαζόμαστε την τιμή του προγόνου ενώ έχουμε πρόσβαση στον αρχικό πίνακα όταν θέλουμε την τιμή για τον κόμβο.

4.3 Πειραματική Αξιολόγηση

Ο αλγόριθμος Randomized Symmetry Breaking αναπτύχθηκε στην γλώσσα προγραμματισμού XMTC και έχε προσομοιωθεί με την χρήση του XMT για να πάρουμε τα αποτελέσματα του παράλληλου υπολογισμού. Ακολουθούν οι μετρήσεις που πάρθηκαν από την προσομοίωση του αλγορίθμου, γραφικές παραστάσεις που παρουσιάζουν τα αποτελέσματα των μετρήσεων και τέλος ακολουθούν τα συμπεράσματα στα οποία έχω καταλήξει.

Στην εκτέλεση μας για να δούμε τα αποτελέσματα του αλγορίθμου έχουμε 6 διαφορετικές εκτελέσεις για κάθε αριθμό επεξεργαστών. Δηλαδή, για 2 επεξεργαστές έχω τρέξει 6 διαφορετικά αρχεία όπου κάθε αρχείο έχει διαφορετικό πίνακα τυχαίων αριθμών, για 4 επεξεργαστές έχω τρέξει 6 διαφορετικά αρχεία όπου το κάθε αρχείο έχει διαφορετικό πίνακα τυχαίων αριθμών, κ.ο.κ. Οι εκτελέσεις έχουν γίνει για 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 και 4096 επεξεργαστές. Σε κάθε εκτέλεση λόγω του ότι έχω διαφορετικούς τυχαίους αριθμούς παίρνω διαφορετικό ανεξάρτητο σύνολο και ο χρόνος εκτέλεσης κάθε περίπτωσης είναι διαφορετικός.

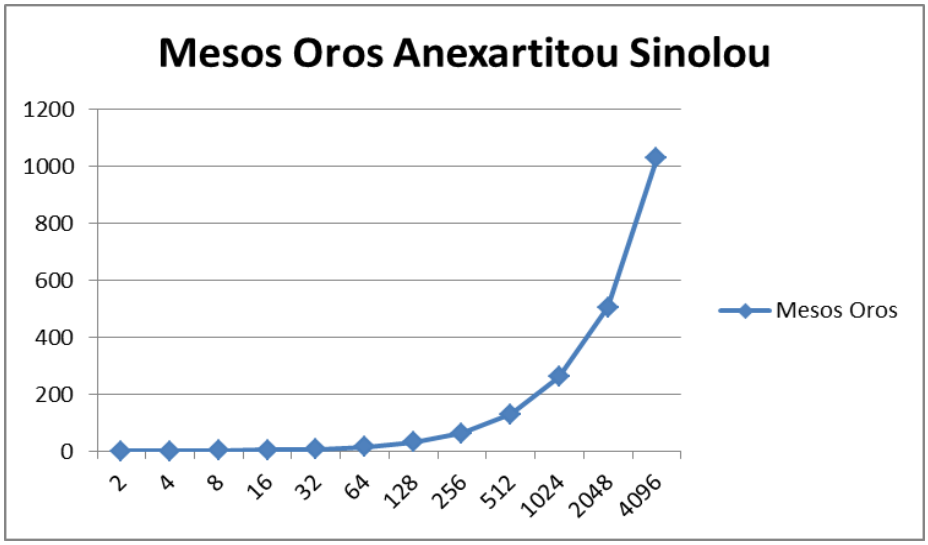
Τα δεδομένα του προβλήματος δημιουργήθηκαν στην γλώσσα προγραμματισμού C και γράφτηκαν σε αρχεία και εισάχθηκαν στο πρόγραμμα μας με την βοήθεια του εργαλείου μνήμης memMapCreate. Οι τιμές των δεδομένων εισόδου καθορίζονται τυχαία και είναι ακέραιοι αριθμοί από το 0 μέχρι και το 1.

Στο Πίνακα 4.1 παρουσιάζονται τα αποτελέσματα του ανεξάρτητου συνόλου από κάθε εκτέλεση καθώς και ο μέσος όρος του αριθμού του ανεξάρτητου συνόλου που παίρνω από κάθε εκτέλεση με διαφορετικό αριθμό επεξεργαστών.

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
Ανεξάρτητο Σύνολο	0	2	2	5	5	17	33	64	127	259	521	1038
	0	2	2	4	8	16	35	69	124	259	494	1045
	1	1	2	5	10	17	28	67	134	257	502	1040
	1	0	2	4	7	14	33	58	139	269	520	1004
	0	1	2	4	9	16	33	60	132	264	509	1013
	1	2	1	5	7	15	34	64	127	261	490	1033
Μέσος Όρος	0,5	1,33	1,83	4,5	7,66	15,83	32,66	63,66	130,5	261,5	506	1028,83
Τυπική Απόκλιση	0,5	0,81	0,40	0,5	1,75	1,16	2,42	4,13	5,54	4,37	13,00	16,46

Πίνακας 4.1

Στην Γραφική παράσταση 4.1 παρουσιάζεται ο μέσος όρος του ανεξάρτητου συνόλου που παίρνουμε από τις 6 εκτελέσεις για κάθε αριθμό επεξεργαστή.



Γραφική 4.1

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και ο μέσος όρος του ανεξάρτητου συνόλου. Η αύξηση είναι λογαριθμική.

Στην Γραφική παράσταση 4.2 παρουσιάζεται η τυπική απόκλιση του ανεξάρτητου συνόλου που παίρνουμε από τις 6 εκτελέσεις για κάθε αριθμό επεξεργαστή.

Γραφική 4.2

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και ο αριθμός της τυπικής απόκλισης. Όσο πιο μικρός είναι ο αριθμός των κόμβων τόσο πιο μικρή είναι και η τυπική απόκλιση. Η τυπική απόκλιση δεν έχει σταθερό ρυθμό αύξησης.

Στο Πίνακα 4.2 παρουσιάζονται τα αποτελέσματα του χρόνου εύρεσης του ανεξάρτητου συνόλου από κάθε εκτέλεση καθώς και ο μέσος όρος του χρόνου εύρεσης του ανεξάρτητου συνόλου που παίρνω από κάθε εκτέλεση με διαφορετικό αριθμό επεξεργαστών. Ο χρόνος δίνεται από τον προσομοιωτή μετά το τέλος της εκτέλεσης του αλγορίθμου και είναι σε clock cycles.

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
Χρόνος	634	622	671	688	742	854	1063	1424	2200	3775	6141	10881
	600	623	660	692	742	854	1063	1424	2195	3774	6159	10873
	617	637	676	690	742	854	1063	1424	2195	3790	6149	10886
	618	607	676	692	742	854	1063	1424	2192	3775	6147	10891
	600	638	670	686	742	854	1063	1424	2197	3775	6153	10905
	618	623	672	683	742	854	1063	1424	2227	3780	6145	10869
Μέσος Όρος	614.5	625	670.83	688.50	742	854	1063	1424	2201	3778.16	6149.00	10884.17

Πίνακας 4.2

Στην Γραφική παράσταση 4.3 παρουσιάζεται ο μέσος όρος του χρόνου εύρεσης του ανεξάρτητου συνόλου που παίρνουμε από τις 6 εκτελέσεις για κάθε αριθμό επεξεργαστή.

Γραφική 4.3

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και ο χρόνος που απαιτείται από τον αλγόριθμό μας για την εύρεση του ανεξάρτητου συνόλου. Ο θεωρητικός χρόνος εκτέλεσης είναι O(1), δηλαδή ο πρακτικός χρόνος εκτέλεσης έπρεπε να είναι σταθερός για οποιαδήποτε τιμή του αριθμού των κόμβων (n). Από την γραφική παρατηρούμε πως ο πρακτικός χρόνος εκτέλεσης δεν είναι σταθερός. Αυτό συμβαίνει λόγω του κόστους υλοποίησης που έχουμε για τον συγχρονισμό των επεξεργαστών μετά από κάθε παράλληλο κομμάτι του κώδικα.

Σε αντίθεση σε σχέση με τον χρόνο προσομοίωσης του αλγορίθμου, το κόστος του δεν παρέχεται από τον προσομοιωτή, αλλά υπολογίζεται βάσει του γινομένου του χρόνου εκτέλεσης του αλγορίθμου επί τον αριθμό των επεξεργαστών που χρησιμοποιούνται σε κάθε περίπτωση. Στους Πίνακες 4.3 και 4.4 παρουσιάζονται τα αποτελέσματα του κόστους του αλγορίθμου.

	Αριθμός Επεξεργαστών						
	2	**4**	**8**	**16**	**32**	**64**	**128**
Κόστος	1268	2488	5368	11008	23744	54656	136064
	1200	2492	5280	11072	23744	54656	136064
	1234	2548	5408	11040	23744	54656	136064
	1236	2428	5408	11072	23744	54656	136064
	1200	2552	5360	10976	23744	54656	136064
	1236	2492	5376	10928	23744	54656	136064
Μέσος Όρος	1229	2500	5366,667	11016	23744	54656	136064

Πίνακας 4.3

	Αριθμός Επεξεργαστών				
	256	**512**	**1024**	**2048**	**4096**
Κόστος	364544	1126400	3865600	12576768	44568576
	364544	1123840	3864576	12613632	44535808
	364544	1123840	3880960	12593152	44589056
	364544	1122304	3865600	12589056	44609536
	364544	1124864	3865600	12601344	44666880
	364544	1140224	3870720	12584960	44519424
Μέσος Όρος	364544	1126912	3868843	12593152	44581547

Πίνακας 4.4

Στην Γραφική παράσταση 4.4 παρουσιάζεται ο μέσος όρος του κόστους εκτέλεσης του αλγορίθμου.

Γραφική 4.4

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και το κόστος του αλγορίθμου μας για την εύρεση του ανεξάρτητου συνόλου. Η αύξηση είναι λογαριθμική.

Κεφάλαιο 5

Επίπεδοι Γράφοι

5.1 Πρόβλημα

Σε αυτό το κεφάλαιο θα μελετήσουμε μια πιο γενική περίπτωση από το προηγούμενο κεφάλαιο. Σε αυτό το κεφάλαιο το ενσωματωμένο γράφημα $G=(V,E)$ είναι ένας αυθαίρετος επίπεδος γράφος. Υποθέτουμε ότι το ενσωματωμένο γράφημα G αναπαρίσταται με την λίστα των ακμών του. Για κάθε κορυφή $v \in V$, η λίστα των ακμών του v περιέχει της ακμές του κόμβου αριστερόστροφα με την σειρά που εμφανίζονται γύρω από τον κόμβο.

Μικρού – Βαθμού Κόμβου (low degree vertex): Είναι οι κόμβοι όπου ο βαθμός τους είναι μικρότερος ή ίσος με έξι, δηλαδή είναι οι κόμβοι που έχουν 6 ή λιγότερους από 6 γείτονες.

Για την εύρεση φραγμένου ανεξάρτητου συνόλου από το το G, ακολουθούμε την ίδια στρατηγική την οποία ακολουθούμε στους κατευθυνόμενους κύκλους. Αναθέτουμε τυχαία τιμή 0 ή 1 με ίση πιθανότητα στους μικρού βαθμού κόμβους (low degree vertex). Το σύνολο των κόμβων όπου η ετικέτα (τυχαία τιμή) που τους ανατέθηκε είναι ίση με 1 δεν αποτελούν απαραίτητα το ανεξάρτητο σύνολο.

Για κάθε ακμή $(u,v) \in E$, ξανά αναθέτουμε τιμή στις κορυφές u και v όπου το u και το v έχουν και τα δύο τιμή 1. Οι υπόλοιπες κορυφές με ετικέτα 1 αποτελούν το ανεξάρτητο σύνολο του οποίου το μέγεθος του είναι φραγμένο σταθερά από το $|V|$ με μεγάλη

πιθανότητα. Δηλαδή, το εξαγόμενο του αλγορίθμου είναι ένα σύνολο από μικρού βαθμού κόμβους, όπου οι κόμβοι αυτοί σχηματίζουν ένα ανεξάρτητο σύνολο με μεγάλη πιθανότητα.

5.2 Παράλληλος Αλγόριθμος

Ο παράλληλος αλγόριθμος για την εύρεση ανεξάρτητου συνόλου σε επίπεδο γράφο είναι ο ακόλουθος [1]:

begin

1. **for** each vertex $v \in V$ **pardo**
   if(deg(v) <= 6**) then** set lowdeg := 1
   else set lowdeg := 0
2. **for** each vertex v ∈ V **pardo**
   if(lowdeg(v) = 1) **then** Randomly assign label(v) := 0 or 1 with equal probability.
   else set label(v) := 0
3. **for** each vertex v ∈ V **pardo**
   if (label(v) = 1) **then**
   if (label(u) = 1 for some u on the list of v) **then** set label(v) := 0

end

Ο πιο πάνω αλγόριθμος (Fractional Independent Set) αποτελείται από τρία βήματα. Όλοι οι επεξεργαστές δουλεύουν παράλληλα σε όλα τα βήματα του αλγορίθμου. Στο πρώτο βήμα γίνεται ανάθεση τιμής 1 σε κάθε κόμβο που έχει 6 ή και λιγότερους γείτονες ή γίνεται ανάθεση της τιμής 0 σε κάθε κόμβο που έχει περισσότερους από 6 γείτονες. Στο δεύτερο βήμα γίνεται τυχαία ανάθεση των τιμών 1 ή 0 σε όσους κόμβους πήραν τιμή 1 από το προηγούμενο βήμα. Στην πράξη λόγω των περιορισμών της πλατφόρμα XMT η υλοποίηση του βήματος αυτού γίνεται στην C και εισάγονται τα δεδομένα μέσω του εργαλείου μνήμης στο παράλληλο πρόγραμμα μας. Στο τρίτο βήμα αν κάποιος κόμβος έχει τιμή 1 και υπάρχει έστω και ένας γείτονας του συγκεκριμένου κόμβου με τιμή 1 τότε η τιμή του συγκεκριμένου κόμβου γίνεται 0. Μετά το τέλος της εκτέλεσης του αλγορίθμου όσοι από τους κόμβους εξακολουθούν να έχουν τιμή 1 αποτελούν το ανεξάρτητο σύνολο μας.

Μοντέλο PRAM: Τα βήματα 1 και 2 του αλγορίθμου δεν απαιτούν ταυτόχρονη πρόσβαση στην μνήμη. Πολλές ακμές μπορεί να μοιράζονται τον ίδιο κόμβο, οπόταν το βήμα 3 του αλγορίθμου χρειάζεται ταυτόχρονη ανάγνωση. Παρόλα αυτά δεν χρειάζεται οποιοδήποτε ταυτόχρονο γράψιμο, οπόταν ο αλγόριθμος μπορεί να τρέχει στο μοντέλο CREW PRAM.

5.3 Πειραματική Αξιολόγηση

Ο αλγόριθμος Fractional Independent Set αναπτύχθηκε στην γλώσσα προγραμματισμού XMTC και έχει προσομοιωθεί με την χρήση του XMT για να πάρουμε τα αποτελέσματα του παράλληλου υπολογισμού. Ακολουθούν οι μετρήσεις που πάρθηκαν από την προσομοίωση του αλγορίθμου, γραφικές παραστάσεις που παρουσιάζουν τα αποτελέσματα των μετρήσεων και τέλος ακολουθούν τα συμπεράσματα στα οποία έχω καταλήξει.

Έχουμε δύο διαφορετικές εκτελέσεις για να δούμε τα αποτελέσματα του αλγορίθμου. Στην πρώτη καθώς και στην δεύτερη εκτέλεση έχουμε 4 διαφορετικά αρχεία με διαφορετικούς τυχαίους αριθμούς για κάθε αριθμό επεξεργαστών. Η διαφορά ανάμεσα στις δύο εκτελέσεις είναι ότι στην δεύτερη εκτέλεση έχω ένα γράφο στον οποίο προσθέτω ακμές και κόμβους και τον προσαρμόζω ανάλογα κάθε φορά. Δηλαδή, για 2 επεξεργαστές έχω τρέξει 4 διαφορετικά αρχεία όπου κάθε αρχείο έχει διαφορετικό πίνακα τυχαίων αριθμών, για 4 επεξεργαστές έχω τρέξει 4 διαφορετικά αρχεία όπου το κάθε αρχείο έχει διαφορετικό πίνακα τυχαίων αριθμών , κ.ο.κ. Οι εκτελέσεις έχουν γίνει για 2, 4, 8, 12, 16, 18, 20, 25, 30, 32, 40 και 60 επεξεργαστές. Σε κάθε εκτέλεση λόγω του ότι έχω διαφορετικούς τυχαίους αριθμούς παίρνω διαφορετικό ανεξάρτητο σύνολο και ο χρόνος εκτέλεσης κάθε περίπτωσης είναι διαφορετικός.

Τα δεδομένα του προβλήματος εισάχθηκαν στο πρόγραμμα μέσω αρχείων που δημιουργήθηκαν στην γλώσσα προγραμματισμού C και εισάχθηκαν στο πρόγραμμα μας με την βοήθεια του εργαλείου μνήμης memMapCreate. Οι τιμές των δεδομένων εισόδου καθορίζονται τυχαία και είναι ακέραιοι αριθμοί από το 0 μέχρι και το 1.

Στους Πίνακες 5.1 και 5.2 παρουσιάζονται τα αποτελέσματα του ανεξάρτητου συνόλου από κάθε παράδειγμα της πρώτης και δεύτερης εκτέλεσης αντίστοιχα καθώς και ο μέσος όρος του αριθμού του ανεξάρτητου συνόλου.

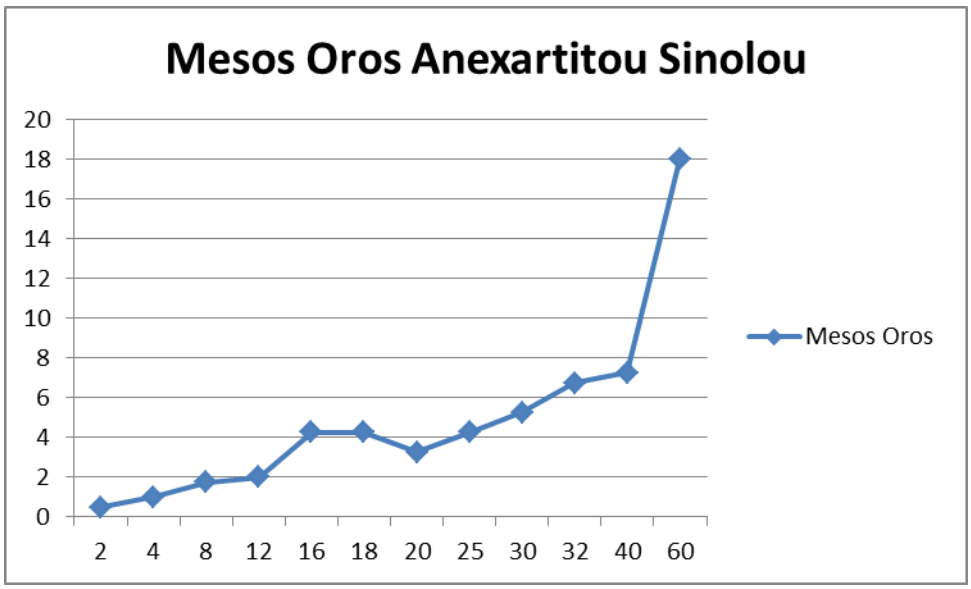
	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Ανεξάρτητο Σύνολο	1	2	4	2	1	3	3	3	6	7	9	17
	0	1	1	2	7	6	3	4	6	7	8	18
	0	0	1	1	5	4	1	5	4	7	5	17
	1	1	1	3	4	4	6	5	5	6	7	20
Μέσος Όρος	0.5	1	1.75	2	4.25	4.25	3.25	4.25	5.25	6.75	7.25	18

Πίνακας 5.1

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Ανεξάρτητο Σύνολο	1	2	1	3	2	2	3	3	6	7	5	29
	0	2	3	4	3	1	3	3	7	5	7	14
	1	0	2	2	4	1	5	5	2	0	8	19
	1	1	3	6	3	5	3	3	2	5	7	15
Μέσος Όρος	0.75	1.25	2.25	3.75	3	2.25	3.5	3.5	4.25	4.25	6.75	19.25

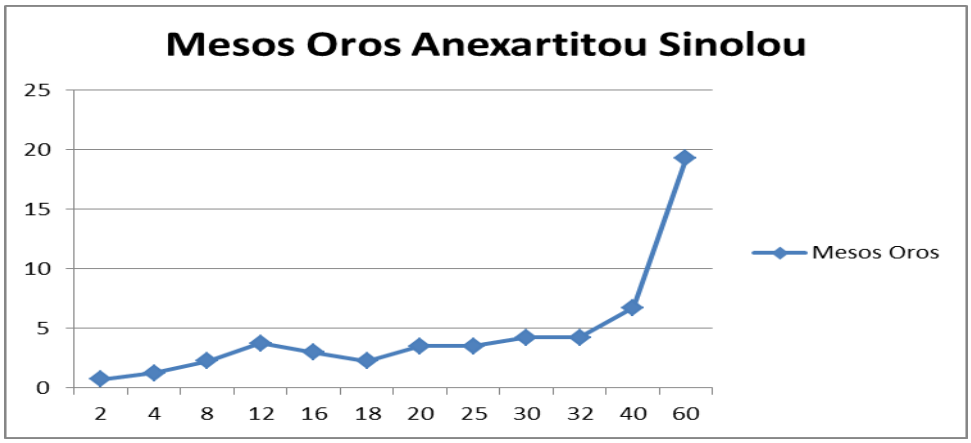
Πίνακας 5.2

Στην Γραφική παράσταση 5.1 παρουσιάζεται ο μέσος όρος του ανεξάρτητου συνόλου που παίρνουμε από τα παραδείγματα για κάθε αριθμό επεξεργαστή της πρώτης εκτέλεσης.



Γραφική 5.1

Στην Γραφική παράσταση 5.2 παρουσιάζεται ο μέσος όρος του ανεξάρτητου συνόλου που παίρνουμε από τα παραδείγματα για κάθε αριθμό επεξεργαστή της δεύτερης εκτέλεσης.



Γραφική 5.2

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και ο μέσος όρος του ανεξάρτητου συνόλου. Στην πρώτη εκτέλεση έχουμε μία μικρή πτώση τους 20 επεξεργαστές αλλά μετά ο μέσος όρος του ανεξάρτητου συνόλου συνεχίζει να αυξάνεται. Το ίδιο συμβαίνει και στην δεύτερη εκτέλεση με την διαφορά ότι έχουμε μία μικρή πτώση τους 16 και18 επεξεργαστές αλλά μετά ο μέσος όρος του ανεξάρτητου συνόλου συνεχίζει να αυξάνεται.

Στους Πίνακες 5.3 και 5.4 παρουσιάζονται τα αποτελέσματα του χρόνου εύρεσης του ανεξάρτητου συνόλου από τα παραδείγματα της πρώτης και δεύτερης εκτέλεσης αντίστοιχα καθώς και ο μέσος όρος του χρόνου εύρεσης του ανεξάρτητου συνόλου. Ο χρόνος δίνεται από τον προσομοιωτή μετά το τέλος της εκτέλεσης του αλγορίθμου και είναι σε clock cycles.

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Χρόνος	1561	1700	2298	2901	3512	3770	4043	4723	5386	5652	6887	9597
	1185	1659	2296	2901	3454	3712	4134	4774	5292	5707	6887	9699
	1185	1703	2197	2899	3507	3700	4133	4777	5335	5653	6836	9592
	1361	1705	2243	2901	3507	3712	4041	4769	5390	5650	6802	9687
Μέσος Όρος	1323	1691,7	2258,5	2900,5	3495	3723,5	4087,7	4760,7	5350,7	5665,5	6853	9643,7

Πίνακας 5.3

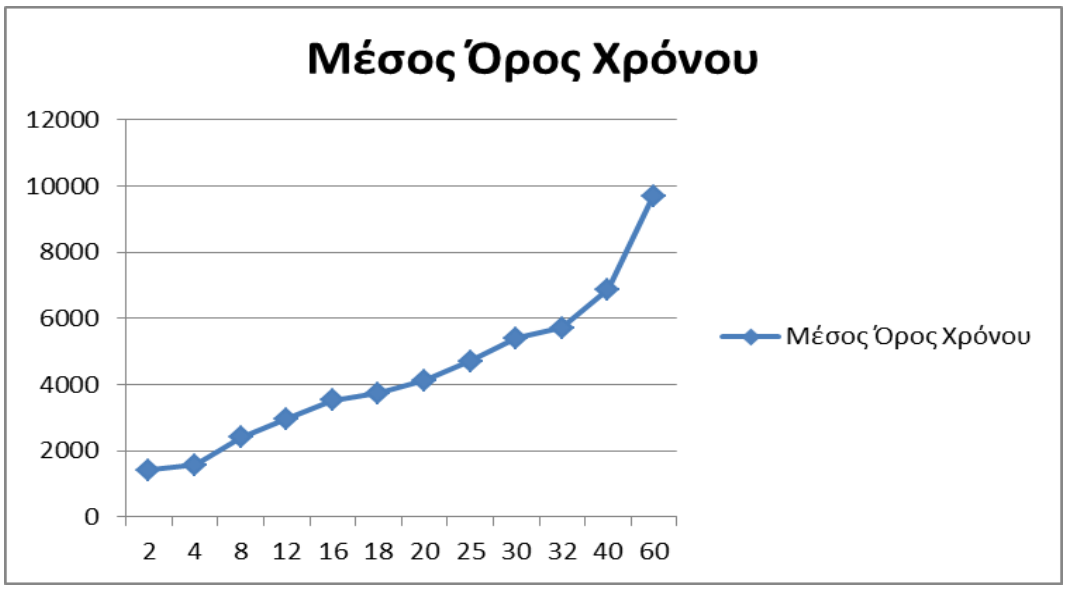
	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Χρόνος	1559	1660	2400	2993	3651	3741	4136	4694	5440	5662	6853	9677
	1185	1660	2412	2991	3468	3740	3983	4696	5393	5712	6898	9693
	1561	1312	2410	2923	3496	3739	4197	4691	5393	5717	6854	9676
	1361	1610	2409	2934	3472	3738	4142	4694	5394	5709	6843	9694
Μέσος Όρος	1416,5	1560,5	2407,7	2960,2	3521,7	3739,5	4114,5	4693,7	5405	5700	6862	9885

Πίνακας 5.4

Στην πιο κάτω γραφική παράσταση 5.3 παρουσιάζεται ο μέσος όρος του χρόνου εύρεσης του ανεξάρτητου συνόλου που παίρνουμε από τα 4 παραδείγματα για κάθε αριθμό επεξεργαστή από την πρώτη εκτέλεση.

Γραφική 5.3

Στην Γραφική παράσταση 5.4 παρουσιάζεται ο μέσος όρος του χρόνου εύρεσης του ανεξάρτητου συνόλου που παίρνουμε από τα 4 παραδείγματα για κάθε αριθμό επεξεργαστή από την δεύτερη εκτέλεση.



Γραφική 5.4

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και ο χρόνος που απαιτείται από τον αλγόριθμό μας για την εύρεση του ανεξάρτητου συνόλου. Και στις δύο εκτελέσεις του αλγορίθμου οι τιμές που παίρνουμε είναι παρόμοιες. Ο θεωρητικός χρόνος εκτέλεσης είναι O(1), δηλαδή ο πρακτικός χρόνος εκτέλεσης έπρεπε να είναι σταθερός για οποιαδήποτε τιμή του αριθμού των κόμβων (n). Από

την γραφική παρατηρούμε πως ο πρακτικός χρόνος εκτέλεσης δεν είναι σταθερός. Αυτό συμβαίνει λόγω του κόστους υλοποίησης που έχουμε για τον συγχρονισμό των επεξεργαστών μετά από κάθε παράλληλο κομμάτι του κώδικα.

Σε αντίθεση σε σχέση με τον χρόνο προσομοίωσης του αλγορίθμου, το κόστος του δεν παρέχεται από τον προσομοιωτή, αλλά υπολογίζεται βάσει του γινομένου του χρόνου εκτέλεσης του αλγορίθμου επί τον αριθμό των επεξεργαστών που χρησιμοποιούνται σε κάθε περίπτωση. Στους Πίνακες 5.5 και 5.6 παρουσιάζονται τα αποτελέσματα του κόστους του αλγορίθμου της πρώτης και της δεύτερης εκτέλεσης αντίστοιχα.

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Κόστος	3122	6800	18384	34812	56192	67860	80860	118075	161580	180864	275480	575820
	2370	6636	18368	34812	55264	66816	82680	119350	158760	182624	275480	571940
	2370	6812	17576	34788	56112	66600	82660	119425	160050	180896	273440	575520
	2722	6820	17944	34812	56112	66816	80820	119225	161700	180800	272080	571220
Μέσος Όρος	2426	6767	18068	34806	55920	67023	81755	119018,8	160522,5	181926	274120	578625

Πίνακας 5.5

	Αριθμός Επεξεργαστών											
	2	**4**	**8**	**12**	**16**	**18**	**20**	**25**	**30**	**32**	**40**	**60**
Κόστος	3118	6640	19200	35916	58416	67338	82720	117350	163200	181184	274120	580620
	2370	6640	19296	35892	55488	67320	79660	117400	161790	182784	275920	581580
	3122	5248	19280	35076	55936	67302	83940	117275	161790	182944	274160	580560
	2722	6440	19272	35208	55552	67284	82840	117350	161820	182688	273720	581640
Μέσος Όρος	2833	6242	19262	35523	56348	67311	82290	117343,8	162150	182400	274480	581100

Πίνακας 5.6

Στην πιο κάτω γραφική παράσταση 5.5 παρουσιάζεται ο μέσος όρος του κόστους εκτέλεσης του αλγορίθμου από την πρώτη εκτέλεση.

Γραφική 5.5

Στην Γραφική παράσταση 5.6 παρουσιάζεται ο μέσος όρος του κόστου εκτέλεσης του αλγορίθμου από την δεύτερη εκτέλεση.

Γραφική 5.6

Παρατηρήσεις

Από την πιο πάνω γραφική παράσταση παρατηρούμε πως όσο αυξάνεται το μέγεθος του πίνακα (το μέγεθος του πίνακα σε κάθε εκτέλεση ισούται με τον αριθμό των επεξεργαστών της εκτέλεσης) αυξάνεται και το κόστος του αλγορίθμου μας για την εύρεση του ανεξάρτητου συνόλου. Η αύξηση γίνεται λογαριθμικά. Αυτό το παρατηρούμε με μεγάλη ευκολία από τις γραφικές μας, αφού έχουμε αύξηση του κόστους που χρειαζόμαστε με το πέρας των δεδομένων. Και στις δύο εκτελέσεις του αλγορίθμου οι τιμές που παίρνουμε είναι

παρόμοιες. Άρα όσα περισσότερα στοιχεία έχουμε, τόσο περισσότερο κόστος χρειάζεται να λυθεί το πρόβλημα μας.

Κεφάλαιο 6

Συμπεράσματα

6.1 Τελικά Συμπεράσματα

Η παρούσα εργασία έχει ως βάση της την υλοποίηση, την προσομοίωση και την πειραματική αξιολόγηση παράλληλων αλγορίθμων. Από την πειραματική αξιολόγηση πάρθηκαν μετρήσεις οι οποίες καταγράφονται με την μορφή πινάκων και παρουσιάζονται με γραφικές παραστάσεις.

Από τις μετρήσεις παρατηρήθηκε ότι σε όλους τους αλγορίθμους ο χρόνος εκτέλεσης τους εξαρτάται από το πλήθος των δεδομένων εισόδου. Πιο συγκεκριμένα, όσο αυξάνεται το πλήθος των δεδομένων εισόδου αυξάνεται και ο χρόνος εκτέλεσης. Ο θεωρητικός χρόνος εκτέλεσης είναι O(1), δηλαδή ο πρακτικός χρόνος εκτέλεσης έπρεπε να είναι σταθερός για οποιαδήποτε τιμή του αριθμού των κόμβων (n). Αυτό συμβαίνει λόγω του κόστους υλοποίησης που έχουμε για τον συγχρονισμό των επεξεργαστών μετά από κάθε παράλληλο κομμάτι του κώδικα. Αυτό συμβαίνει και στους δύο αλγορίθμους που έχουν υλοποιηθεί. Δεν μπορούμε να πούμε πως η πλατφόρμα μας δεν είναι αξιόπιστη λόγω του ότι έχουν μελετηθεί προηγούμενες μελέτες που τα αποτελέσματα της πρακτικής εκτέλεσης επαλήθευαν τα αποτελέσματα της θεωρητικής εκτέλεσης.

Επίσης, και στους δύο αλγορίθμους όσο αυξάνεται ο αριθμός των κόμβων αυξάνεται και ο αριθμός του ανεξάρτητου συνόλου κάτι το οποίο ήταν και το αναμενόμενο.

6.2 Οφέλη Ατομικής Διπλωματικής Εργασίας

Τα οφέλη που αποκόμισα κατά την διάρκεια υλοποίησης της παρούσας Διπλωματικής Εργασίας είναι αρκετά. Πρώτα από όλα κατανόησα τον τρόπο ανάπτυξης ενός παράλληλου αλγορίθμου. Επίσης, έμαθα και κατανόησα καλύτερα τους περιορισμούς του παραλληλισμού και της γλώσσας προγραμματισμού XMTC. Λόγω των περιορισμών της γλώσσας έπρεπε να βρω εναλλακτική λύση για την εισαγωγή των τυχαίων αριθμών και αυτό με βοήθησε να αναπτύξω την σκέψη μου. Ακόμη, η πειραματική αξιολόγηση των αλγορίθμων με βοήθησε στην ανάπτυξη της κριτικής σκέψης και εξαγωγής συμπερασμάτων.

Δεν θα μπορούσα να μην αναφέρω το γεγονός ότι ο παραλληλισμός αποτελεί το μέλλον στην επιστήμη της πληροφορικής. Πιστεύω πως ο παράλληλος τρόπος σκέψης που ανέπτυξα σίγουρα θα με βοηθήσει στην μετέπειτα πορεία μου.

6.3 Μελλοντική Εργασία

Στις μέρες μας η παράλληλη επεξεργασία είναι πάρα πολύ σημαντική γιατί μας παρέχει ευκαιρίες που δεν έχουμε στην σειριακή επεξεργασία όπως την πιο γρήγορη και αποδοτική επίλυση των προβλημάτων.

Ως μελλοντική εργασία θα μπορούσαμε να υλοποιήσουμε τους ίδιους αλγορίθμους σε άλλη πλατφόρμα παραλληλισμού και να συγκρίνουμε τα αποτελέσματα με την παρούσα εργασία. Έτσι θα δούμε ποια πλατφόρμα είναι πιο αποδοτική.

Επίσης, ως μελλοντική εργασία μπορούμε να υλοποιήσουμε στην πλατφόρμα XMT και στην γλώσσα προγραμματισμού XMTC πιο πολύπλοκους και πιο απαιτητικούς αλγορίθμους.

Προς το παρόν η πλατφόρμα XMT και η γλώσσα προγραμματισμού XMTC έχουν αρκετά προβλήματα και πάρα πολλούς περιορισμούς έτσι θα μπορούσαμε να υλοποιήσουμε τους ίδιους αλγορίθμους στην πλατφόρμα XMT και στην γλώσσα προγραμματισμού XMTC αφού αναβαθμιστούν.

Βιβλιογραφία

[1] Joseph Jaja, An Introduction to Parallel Algorithms, Addision-Wesly Publishing Company, Inc,1992

[2] Χρύσης Γεωργίου, Σημειώσεις Μαθήματος ΕΠΛ431-Σύνθεση Παράλληλων Αλγορίθμων, Τμήμα Πληροφορικής Κύπρου http://www.cs.ucy.ac.cy/~chryssis/EPL431/

[3] http://en.wikipedia.org/wiki/Flynn%27s_taxonomy.

[4] http://en.wikipedia.org/wiki/SISD

[5] http://en.wikipedia.org/wiki/SIMD

[6] http://en.wikipedia.org/wiki/MISD

[7] http://en.wikipedia.org/wiki/MIMD

[8] http://en.wikipedia.org/wiki/Parallel_random-access_machine

[9] http://en.wikipedia.org/wiki/Parallel_algorithm

[10] http://www.umiacs.umd.edu/~vishkin/XMT/index.shtml

[11] http://www.umiacs.umd.edu/users/vishkin/XMT/spaa07paper.pdf

[12] http://en.wikipedia.org/wiki/XMTC

[13] http://drum.lib.umd.edu/bitstream/1903/146/1/dissertation.pdf

[14] https://computing.llnl.gov/tutorials/parallel_comp/#Flynn

[15] Audin O.Balkan, Uzi Vishkin, George C. Caragea, Alexandros Tzannes, "XMT Toolchain Manual for XMTC Language, XMTC Compiler, XMT Simulator and Paraleap XMT FPGA Computer" 18th March 2014
http://www.umiacs.umd.edu/users/vishkin/XMT/manual4xmtc1out-of2.pdf

[16] Audin O.Balkan, Uzi Vishkin, George C. Caragea, Alexandros Tzannes, "Programmer's Manual for XMTC Language, XMTC Compiler and Paraleap XMT FPGA Computer" 27th October 2010
http://www.umiacs.umd.edu/users/vishkin/XMT/tutorial4xmtc2out-of2.pdf

[17] Jon Kleinberg, Ena Tardos, Σχεδιασμός Αλγορίθμων, Εκδόσεις Κλειδάριθμος, 2008

[18] http://eclass.uoa.gr/modules/document/file.php/DI277/SLIDES/L3-IntegerProgramming.pdf

[19] http://www.umiacs.umd.edu/users/vishkin/XMT/IPDPS08PanelPresentation.pdf

Παράρτημα Α

Στο σημείο αυτό θα εισάγω τους κώδικες που υλοποιήθηκαν στην παράλληλη γλώσσα προγραμματισμού XMTC.

A1. Random Numbers Generator

```
/* Mathima:      Ptixiaki - Xeimerino 2014
 *
 * Programma:    Random Numbers Generator
 * Skopos: Gemisma enos Pinaka me tixaious arithmous apo 0-1

 *
 * Sigrafeas:    Antria Anastasiou
 *
 * Imerominia:  29/10/2014
 *
 */

/* Compile Instructions
 * - gcc rad_function.c -o rand_function
 * - ./rand_function
*/

/*
 * How to change N
 * change the value of N in line 32(variable declaration)

 *
 */

//include libraries
#include <stdio.h>
#include <stdlib.h>

//main program
int main(){

//value of random nums that we need
int N=4096;

FILE *fp;
fp=fopen("test.txt", "w");

//help variable
int i;
int rand_array[N];

//use time to take different values each time
time_t t;
srand((unsigned) time(&t));
```

```
for(i=0;i<N;i++)
{
t=rand() % 2;
printf("%d\n", t);
fprintf(fp, "%d\n",t);
}

}//end of main
```

A2. Randomized Symmetry Breaking

```
/* Mathima:      Ptixiaki - Xeimerino 2014
 *
 * Programma:    Algorithm 9.1 - Randomized Symmetry Breaking
 * Skopos: Ena sinolo apo korifes to opoio apotelei ena aneksartito
sinolo me megali pithanotita
 *
 * Sigrafeas:    Antria Anastasiou
 *
 * Imerominia:  22/10/2014
 *
 */

/* Compile Instructions
 * - xmtcc algorithm_9_1.c rand.xbo
 * - xmtsim -cycle -binload a.b a.sim
*/

/*
 * How to change n
 * change the value of n in line 31(variable declaration)
 *
 */

//include libraries
#include <xmtc.h>
#include "rand_8.h"

//main program
int main(){

//variable to store the number of the proccessors
int n=8;

//array to represent the directed cycle
int S[n];

//array to label each vertex with 0 or 1
int label[n];

//variable to store apogono of a node
int apogonos;

//help variable
int i;
```

```
//copy rand_array(get this array from the input file) in label array
spawn(0,n-1)
{
      label[$]=rand_array[$];
}

//use this to fill the array S[n]
//S(i)=j if and only if <i,j> is an arc in E
spawn(0,n-1)
{
      if($!=n-1)
      {
            S[$]=$+1;
      }

      //to be a direct cycle the last must be connected with the
first
      if($==n-1)
      {
            S[$]=0;
      }
}

//step 1
//assign label(v) = 1 OR 0 randomly with equal probability
//DONE in memMapCreate

//step 2
//if ( label(v)=1 AND label(S(v))=1 )
//then set label(v)=0
spawn(0,n-1)
{
      apogonos=S[$];
      if ((label[$] == 1) && (rand_array[apogonos] == 1))
      {
            label[$]=0;
      }//end if
}//end spawn

//for loop to print the final array
/*************************************************************/
/*     Put this in comments to take the parallel time        */
/*                                                           */
/*************************************************************/
/*
printf("\nAneksartito Sinolo\n");
for(i=0;i<=n-1;i++)
{
      if (label[i]==1)
      printf("Thesi %d : Number %d\n",i+1,label[i]);
}*/
}//end main
```

A3. Random Numbers Generator for Fractional Independed Set Algorithm

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(){

//num of processors
int n=4;

//table to store the lowdeg
//have 1 when the node has less or equal to 6 neighbors
//have 0 when the node has more than 6 neighbors
int lowdeg[n];

//table to store the number of the neighbors of each node
int deg[n];

//table to store the planar graph
int gitniasi[n][n];

//help variables
int counter;
int i,j;

//table to store the label for the nodes
int temp_label[n];

//open file to store the label table
FILE *fp;
fp=fopen("label_file_para_2.txt", "w");

//EDO VAZO TON PINAKA
/*POU EXO SE KATHE EKTELESI
*
*
*
*/

//Step 1
for(i=0;i<n;i++)
{
      counter=0;
      for(j=0;j<n;j++)
      {
            if (gitniasi[i][j] == 1)
                  counter++;
      }
      deg[i]=counter;
}

for(i=0;i<n;i++)
{
      if(deg[i] <= 6)
            lowdeg[i]=1;
      else
            lowdeg[i]=0;
}

//use time to take different values each time
```

```
time_t t;
srand((unsigned) time(&t));

//Step 2
for(i=0;i<n;i++)
{
      if(lowdeg[i]==1)
      {
            temp_label[i]=rand()%2;
      }
      else
      {
            temp_label[i]=0;
      }
}

}//end of main
```

A4. Fractional Independed Set

```
/* Mathima:      Ptixiaki - Earino 2014
 *
 * Programma:    Algorithm 9.2 - Fractional Independent Set
 * Skopos: A labeled set of low-degree vertices forming a large
independent set with high probability
 *
 * Sigrafeas:    Antria Anastasiou
 *
 * Imerominia:  04/02/2015
 *
 */

/* Compile Instructions
 * - xmtcc algorithm_9_2.c rand.xbo
 * - xmtsim -cycle -binload a.b a.sim
*/

/*
 * How to change n
 * change the value of n in line 31(variable declaration)
 *
 */

//include libraries
#include <xmtc.h>
#include "paradeigma1_4.h"

//main program
int main(){

//variable to store the number of the proccessors
int n=2;
int sinolo=0;
int gitniasi[2][2];
```

```
//parageigma 1
gitniasi[0][1]=1;

gitniasi[1][0]=1;

//if deg(v) <= 6
      //set value = 1
//if deg(v) > 6
      //set value = 0
int lowdeg[n];

//array to label each vertex with 0 or 1
int label_[n];

//store the neighbors of a node
int deg[n];

//help variables
//int counter[n];
int j;
int temp_num[n];

//step 1
spawn(0,n-1)
{
      while(temp_num[$] < n)
      {
            if (gitniasi[$][temp_num[$]] == 1)
            {
                  deg[$]=deg[$]+1;
            }
            temp_num[$]=temp_num[$]+1;
      }
}//end spawn

spawn(0,n-1)
{
      if (deg[$] <= 6)
            lowdeg[$] = 1;
      if (deg[$] > 6)
            lowdeg[$] = 0;
}//end of spawn

//step 2
//DONE in C program
//I get this table from C program
//copy temp_label(get this array from the input file) in label array
spawn(0,n-1)
{
      label_[$]=label[$];
}

//step3
spawn(0,n-1)
{
      if (label[$] == 1)
      {
            while(temp_num[$] >0)
            {
```

```
                if(gitniasi[$][temp_num[$]] == 1)
                {
                      if(label[temp_num[$]]==1)
                            label_[$]=0;
                }
           temp_num[$]=temp_num[$]-1;
           }//end while
      }//end if
}//end spawn

//for loop to print the final array
/*************************************************************/
/*     Put this in comments to take the parallel time        */
/*                                                           */
/*************************************************************/
/*
printf("\nAneksartito Sinolo\n");
for(j=0;j<n;j++)
{
      printf("label %d is %d\n",j,label_[j]);
      if(label_[j]==1)
            sinolo++;
}
printf("sinolo %d\n",sinolo);*/
}//end main
```

Παράρτημα Β

Στο σημείο αυτό θα εισάγω τον τρόπο με τον οποίο γίνεται η εκτέλεση του προγράμματος.

Β1. Εκτέλεση Πρώτου Αλγορίθμου

1. **<u>Παραγωγή αρχείου:</u>** Γίνεται παραγωγή ενός .txt αρχείου με N τυχαίους αριθμούς από με 0 ή 1.

   Εκτέλεση:
   - gcc rand_function.c –o rand_function
   - ./rand_function

     - Για αλλαγή του ονόματος του παραγόμενου αρχείου αλλάζω το όνομα στην γραμμή 35.
     - Για αλλαγή του αριθμού των επεξεργαστών (N) αλλάζω την τιμή στην γραμμή 32(variable declaration).

2. **<u>Εισαγωγή αριθμών:</u>** Γίνεται εισαγωγή των N τυχαίων αριθμών μέσω του εργαλείου μνήμης memMapCreate

   Εκτέλεση:
   - Τρέχω memMapCreate στο terminal
   - Επιλέγω 1(Set File Name)
   - Επιλέγω 4(ώστε και τα τρία αρχεία .xbo, .h, .txt να έχουν το ίδιο όνομα)
   - Πληκτρολογώ το όνομα που θέλω να δώσω στα αρχεία μου
   - Πληκτρολογώ < για επιστροφή στο κυρίως μενού
   - Επιλέγω 2(Read/Write Header to Memory Map)
   - Επιλέγω 2(Add Integer array variable)
     - Name of array variable: *γράφω το όνομα του πίνακα*
     - Number of dimensions(1-2):1
     - Size of dimension 1: *δίνω το μέγεθος του πίνακα*
     - Source: *όνομα_αρχείου*.txt (το αρχείο που παράχθηκε στην C)
     - Is this correct(y/n)? y

- Επιλέγω L(List Current Variables)
- Επιλέγω H(Create Header File)
- Επιλέγω B(Create text and Binary Files from sources)
- Επιλέγω q(quit)

3. Εκτέλεση Παράλληλου Κώδικα
   - Ανοίγω το αρχείο με τον κώδικα
     - Line 25: Κάνω #include το ανάλογο .h αρχείο
     - Line 31: Αλλάζω το N στον αριθμό των επεξεργαστών
   - xmtcc algorithm_9_1.c rand.xbo (όπου rand.xbo το όνομα του αρχείου)
   - xmtsim –cycle –binload a.b a.sim

Β2. Εκτέλεση Δεύτερου Αλγορίθμου

1. Εκτέλεση Παράλληλου Κώδικα
   - Ανοίγω το αρχείο με τον κώδικα
     - Line 25: Κάνω #include το ανάλογο .h αρχείο
     - Line 31: Αλλάζω το N στον αριθμό των επεξεργαστών
   - xmtcc algorithm_9_1.c rand.xbo (όπου rand.xbo το όνομα του αρχείου)
   - xmtsim –cycle –binload a.b a.sim

1. **<u>Παραγωγή αρχείου:</u>** Γίνεται παραγωγή ενός .txt αρχείου με N τυχαίους αριθμούς από με 0 ή 1.

   Εκτέλεση:
   - gcc rand_function.c –o rand_function
   - ./rand_function

     - Για αλλαγή του ονόματος του παραγόμενου αρχείου αλλάζω το όνομα στην γραμμή 30.
     - Για αλλαγή του αριθμού των επεξεργαστών (N) αλλάζω την τιμή στην γραμμή 8(variable declaration).

2. Εισαγωγή αριθμών: Γίνεται εισαγωγή N τυχαίων αριθμών μέσω του εργαλείου μνήμης memMapCreate

Εκτέλεση:

- Τρέχω memMapCreate στο terminal
- Επιλέγω 1(Set File Name)
- Επιλέγω 4(ώστε και τα τρία αρχεία .xbo, .h, .txt να έχουν το ίδιο όνομα)
- Πληκτρολογώ το όνομα που θέλω να δώσω στα αρχεία μου
- Πληκτρολογώ < για επιστροφή στο κυρίως μενού
- Επιλέγω 2(Read/Write Header to Memory Map)
- Επιλέγω 2(Add Integer array variable)
    - Name of array variable: *γράφω το όνομα του πίνακα*
    - Number of dimensions(1-2):1
    - Size of dimension 1: *δίνω το μέγεθος του πίνακα*
    - Source: *όνομα_αρχείου*.txt (το αρχείο που παράχθηκε στην C)
    - Is this correct(y/n)? y
- Επιλέγω L(List Current Variables)
- Επιλέγω H(Create Header File)
- Επιλέγω B(Create text and Binary Files from sources)
- Επιλέγω q(quit)

4. Εκτέλεση Παράλληλου Κώδικα
    - Ανοίγω το αρχείο με τον κώδικα
        - Line 25: Κάνω #include το ανάλογο .h αρχείο που δημιουργήθηκε από το εργαλείο μνήμης
        - Line 31: Αλλάζω το N στον αριθμό των επεξεργαστών το
        - Line 33: Αλλάζω το μέγεθος του πίνακα να ισούται με τον αριθμό των επεξεργαστών
    - xmtcc algorithm_9_2.c rand.xbo (όπου rand.xbo το όνομα του αρχείου)
    - xmtsim –cycle –binload a.b a.sim

Παράρτημα Γ

Κατά την διάρκεια υλοποίησης των αλγορίθμων της παρούσας Ατομικής Διπλωματικής Εργασίας είχα αρκετά προβλήματα. Η αιτία των περισσότερων προβλημάτων είναι το ότι ο μεταγλωττιστής XMT καθώς και η γλώσσα προγραμματισμού XMTC δεν είναι πλήρως ανεπτυγμένα και έχουν πολλούς περιορισμούς και ελλείψεις.

Ένα από τα πρώτα προβλήματα από τα οποία συνάντησα είναι ο περιορισμός του ότι δεν μπορούμε να έχουμε εκτύπωση (printf) μέσα σε κώδικα που εμπερικλείεται σε spawn - join. Λόγω του περιορισμού αυτού δεν είχα την δυνατότητα να τυπώνω τα αποτελέσματα μέσα στο spawn – join block και έτσι αναγκαστικά έπρεπε να τυπώνονται έξω από το block εντολών κάτι το οποίο ήταν πιο δύσκολο στον έλεγχο των αποτελεσμάτων.

Άλλο πρόβλημα το οποίο συνάντησα κατά την υλοποίηση των αλγορίθμων είναι το ότι δεν μπορούμε να ορίσουμε δυναμικά πίνακα. Το μέγεθος των πινάκων κατά τον ορισμό τους πρέπει να είναι σταθερά.

Επίσης, άλλο πρόβλημα το οποίο συνάντησα είναι το ότι δεν υποστηρίζονται αρκετές βιβλιοθήκες. Για παράδειγμα λόγω του ότι δεν υποστηρίζονται οι βιβλιοθήκες <time.h> και <stdlib.h> δεν είχα την δυνατότητα να δημιουργήσω τον πίνακα με τις τυχαίες τιμές απευθείας στην πλατφόρμα XMT. Για τον τυχαίο πίνακα με τιμές 0 ή 1 έπρεπε να γράψω πρόγραμμα στην C όπου να δημιουργώ τον πίνακα και να περνώ τις τιμές μέσω του εργαλείου μνήμης memMapCreate