Ατομική Διπλωματική Εργασία

ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΥΡΩΣΤΩΝ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΟ ΜΟΝΤΕΛΟ SB-PRAM

Παναγιώτα Νικολάου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάρτιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΥΡΩΣΤΩΝ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΣΤΟ ΜΟΝΤΕΛΟ SB-PRAM

Παναγιώτα Νικολάου

Επιβλέπων Καθηγητής
Χρύσης Γεωργίου

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

Μάρτιος 2009

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον εποπτεύοντα καθηγητή μου, τον κύριο Χρύση Γεωργίου που μου έδωσε την ευκαιρία να εργαστώ πάνω σε αυτή την εργασία. Επίσης θέλω να τον ευχαριστήσω για την όντως πολύτιμη βοήθεια, καθοδήγηση και συμβουλές που μου πρόσφερε κατά την εκπόνηση αυτής της Ατομικής Διπλωματικής Εργασίας, αλλά και για την εμπιστοσύνη που μου έδειξε ώστε να φέρω σε πέρας αυτή την εργασία.

Ως επιβλέποντας καθηγητής μου έδωσε την κατάλληλη επιστημονική καθοδήγηση και με τις πάμπολλες εύστοχες παρατηρήσεις, υποδείξεις και συμβουλές του, κατάφερα να ολοκληρώσω την εργασία αυτή.

Περίληψη

Γνωρίζοντας ότι ο παράλληλος υπολογισμός μιας εφαρμογής προσφέρει πολλά πλεονεκτήματα και ότι η συνεταιριστική και ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από ένα επεξεργαστές αποσκοπεί στη γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων, προσπαθήσαμε σε αυτή την Ατομική Διπλωματική Εργασία να πειραματιστούμε και να εξακριβώσουμε τα πλεονεκτήματα της παράλληλης επεξεργασίας .
Σκοπός λοιπόν της εργασίας αυτής ήταν αρχικά η μελέτη και εξοικείωση με τον προσομοιωτή SB-PRAM και έπειτα η υλοποίηση παράλληλων αλγορίθμων με την βοήθεια της γλώσσας προγραμματισμού FORK.

Ο προσομοιωτής SB-PRAM μας επιτρέπει να εκτελέσουμε παράλληλους αλγορίθμους βασισμένους στο μοντέλο PRAM σε σειριακούς υπολογιστές. Παρέχει στον προγραμματιστή την ιδέα του PRAM αλλά και του A-PRAM, δηλαδή των συγχρονισμένων και ασυγχρόνιστων επεξεργαστών που χρησιμοποιούν την κοινόχρηστη μνήμη. Εδώ όμως οι επεξεργαστές προτού επικοινωνήσουν με την κοινόχρηστη μνήμη συνδέονται με ένα δύκτιο, το δύκτιο «πεταλούδας **.**

Στην συνέχεια ακολούθησε η περιγραφή και η υλοποίηση των διάφορων αλγορίθμων αλλά επίσης και η πειραματική αξιολόγηση τους . Αφού πάρθηκαν κάποιες μετρήσεις έγινε προσπάθεια ανάλυσης των αποτελεσμάτων και εξαγωγή συμπερασμάτων για το κατά πόσο έχουμε τα ίδια αποτελέσματα με την θεωρητική αξιολόγηση του αλγορίθμου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και κίνητρο της Διπλωματικής Εργασίας

Για να εκμεταλλευτούμε τις τεράστιες υπολογιστικές δυνατότητες που προσφέρονται από την ανάπτυξη συστημάτων παράλληλης επεξεργασίας, επινοήθηκαν παράλληλοι αλγόριθμοι που να μπορούν να διαχειρίζονται ένα μεγάλο αριθμό επεξεργαστών και θα δουλεύουν παράλληλα για την γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων. Για να το επιτύχουμε αυτό απαιτείται από μέρους των επεξεργαστών αποδοτική συνεργασία μεταξύ τους και προσεκτική διαχείριση των διαθέσιμων πόρων.

Επομένως και η διπλωματική εργασία που έχω αναλάβει έχει σαν σκοπό την προσομοίωση εύρωστων παράλληλων αλγορίθμων στο μοντέλο SB-PRAM με την χρήση της γλώσσας Fork. Ενώ τελικός της στόχος είναι η πειραματική αξιολόγηση των αλγορίθμων αυτών και η εξαγωγή κάποιων συμπερασμάτων όσον αφορά την αποτελεσματικότητα τους στο μοντέλο αυτό.

Τελειώνοντας λοιπόν αυτή την διπλωματική εργασία θα πρέπει να είμαι σε θέση να βγάλω κάποια συμπεράσματα στο κατά πόσο οι αλγόριθμοι που υλοποιήθηκαν και προσομοιώθηκαν διαφέρουν στην πρακτική αξιολόγηση από την θεωρητική, ή αν θεωρητική και πρακτική αξιολόγηση είναι η ίδια και τέλος αν οι βέλτιστοι αλγόριθμοι που επιλύουν ένα πρόβλημα είναι όντως βέλτιστοι και σε πρακτική αξιολόγηση συγκρίνοντας την πρακτική τους διαφορά με τους μη βέλτιστους.

1.2 Μεθοδολογία υλοποίησης μελέτης

Για την ολοκλήρωση της διπλωματικής μου εργασίας ακολουθήθηκε η εξής μεθοδολογία:
Αρχικά, έγινε κάποια μελέτη για κατανόηση των μοντέλων PRAM και SB-PRAM, ούτως ώστε να εξαχθούν οι ιδιότητες αλλά και να εντοπιστούν οι διαφορές των δύο αυτών μοντέλων. Για την μελέτη αυτή χρησιμοποιήθηκε το βιβλίο *Practical PRAM Programming* [1] αλλά και διάφορες ιστοσελίδες οι οποίες αναγράφονται στην βιβλιογραφία.
Έπειτα έγινε εκμάθηση της γλώσσας προγραμματισμού FORK με μελέτη επίσης του βιβλίου *Practical PRAM Programming* αλλά και προσομοίωση κάποιων μικρών προγραμμάτων σε μια μηχανή του Πανεπιστημίου Κύπρου. Ακολούθως έγινε μελέτη και κατανόηση συγκεκριμένων παράλληλων αλγορίθμων. Οι αλγόριθμοι αυτοί ήταν: αλγόριθμος για παράλληλη άθροιση, αλγόριθμος για παράλληλη μετάδοση, ο αλγόριθμος Write-All με δύο επεξεργαστές, ο αλγόριθμος X και ο αλγόριθμος W. Οι αλγόριθμοι αυτοί έπειτα προσομοιωθήκαν στο μοντέλο SB-PRAM με την βοήθεια τις γλώσσας FORK. Τέλος έγινε η πειραματική αξιολόγηση αυτών των αλγορίθμων και εξάχθηκαν διάφορα συμπεράσματα.

1.3 Δομή αυτής της διπλωματικής

Σ' αυτή την διπλωματική περιέχονται 9 κεφάλαια. Περιληπτικά στο **Κεφάλαιο 2** θα γίνει μια περιγραφή του τι είναι παραλληλισμός και ποια είναι η σημαντικότητα του. Επίσης γίνεται επεξήγηση του σημαντικότερου παράλληλου μοντέλου, του μοντέλου PRAM καθώς και του προσομοιωτή SB-PRAM. Έπειτα στο **Κεφάλαιο 3** γίνεται μια περιγραφή της γλώσσας προγραμματισμού, FORK καθώς και επεξήγηση των συστατικών στοιχείων της. Από το **Κεφάλαιο 4** ξεκινάνε οι περιγραφές των αλγορίθμων που έχουν υλοποιηθεί. Συγκεκριμένα στο κεφάλαιο αυτό περιγράφεται ο αλγόριθμος της παράλληλης άθροισης, παραθέτονται μετρήσεις που πάρθηκαν από την πρακτική υλοποίηση του και τέλος γίνονται αναλύσεις και εξάγονται συμπεράσματα πάνω σε αυτές τις μετρήσεις. Παρομοίως στο **Κεφάλαιο 5** περιγράφεται ο αλγόριθμος της παράλληλης μετάδοσης και πάλι παραθέτονται μετρήσεις και εξάγονται συμπεράσματα. Έπειτα εκτελούμε την ίδια δομή για το **Κεφάλαιο 6,7 και 8**

περιγράφοντας τους αλγόριθμους Write-All με δύο επεξεργαστές, τον αλγόριθμο X και τον αλγόριθμο W αντίστοιχα. Τέλος στο **Κεφάλαιο 9** κλείνουμε μιλώντας για μελλοντική δουλειά που μπορεί να γίνει και περιγράφουμε συνοπτικά τα τελικά συμπεράσματα αυτής της Ατομικής Διπλωματικής Εργασίας.

Κεφάλαιο 2

Υπόβαθρο μελέτης

2.1 Τι είναι Παραλληλισμός και ποια είναι η σημαντικότητα του

Για αρκετά χρόνια η επιστήμη των υπολογιστών ασχολείτο με τον σειριακό υπολογισμό όπου ένας επεξεργαστής ήταν υπεύθυνος να εκτελέσει ακολουθιακά τις εντολές κάποιου προγράμματος. Ωστόσο άρχισε βαθμιαία να ωριμάζει και να θέτει ως εξίσου σημαντική και θεμελιώδη και τον παραλληλισμό.

Ο παράλληλος υπολογισμός αφορά την συνεταιριστική και ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από έναν επεξεργαστές αποσκοπώντας στην γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων. Οι επεξεργαστές οι οποίοι συνήθως είναι του ίδιου τύπου, βρίσκονται σε κοντινή απόσταση και έχουν τεράστια υπολογιστική δύναμη, είναι υπεύθυνοι να εκτελούν κάποιους παράλληλους υπολογισμούς [5].

Η σημαντικότητα του παράλληλου υπολογισμού έγκειται στο ότι μπορούμε να διεκπεραιώσουμε πολύπλοκους και μεγάλους υπολογισμούς που ένας σειριακός υπολογιστής δεν θα μπορούσε να διεκπαιρεώσει. Επίσης, η χρήση του παράλληλου υπολογισμού είναι πλέον οικονομικά συμφέρουσα λόγω του χαμηλού κόστους των διάφορων παράλληλων συστημάτων [5].

Αυτό όμως το είδος αρχιτεκτονικής υπολογιστών με μεγάλο πλήθος επεξεργαστών εισάγει καινούργιες απαιτήσεις στο λογισμικό. Ένα πρόγραμμα για ένα κοινό σειριακό

σύστημα πρέπει να διαγράψει μια σειρά από λειτουργίες που ο επεξεργαστής έχει να ακολουθήσει. Αντιθέτως ένα πρόγραμμα για ένα παράλληλο σύστημα πρέπει να διαγράψει μια σειρά από λειτουργίες που ο κάθε επεξεργαστής έχει να ακολουθήσει παράλληλα, περιλαμβάνοντας και κάποιες λειτουργίες που συντονίζουν τους χωριστούς επεξεργαστές στην εκτέλεση μιας ενιαίας διεργασίας. Αυτή όμως η αναγκαιότητα της δημιουργίας και του συντονισμού πολλών παράλληλων διεργασιών προσθέτει μια καινούργια διάσταση στο χώρο του προγραμματισμού. Αλγόριθμοι για συγκεκριμένα προβλήματα πρέπει να σχεδιαστούν με τέτοιο τρόπο ώστε να παράγεται ένας μεγάλος αριθμός παράλληλων λειτουργιών που να εκτελούνται από διαφορετικούς επεξεργαστές. Έτσι παρόλο που τα παράλληλα συστήματα έχουν εξασφαλίσει τη δυνατότητα τεράστιας αύξησης της υπολογιστικής δύναμης με μειωμένο κόστος, αυτή η δυνατότητα μπορεί να πραγματοποιηθεί μόνο με την κατασκευή αποδοτικών παράλληλων αλγορίθμων.

Όμως η παράλληλη επεξεργασία δεν έρχεται σε σύγκρουση με την σειριακή με σκοπό να υπερισχύσει αλλά έρχεται να βοηθηθεί και να λειτουργήσει μαζί με την σειριακή στο ίδιο υπολογιστικό σύστημα ούτως ώστε να έχουμε ταχύτερα και πιο δυνατά παράλληλα συστήματα. Επομένως γι' αυτό και σε ορισμένες περιπτώσεις απλοί σειριακοί αλγόριθμοι μπορούν πολύ εύκολα να προσαρμοστούν και να χρησιμοποιηθούν σε παράλληλα προβλήματα [8].

Για να μπορέσουμε να αναλύσουμε θεωρητικά τους παράλληλους αλγορίθμους και τέλος να τους αξιολογήσουμε ως προς την αποδοτικότητα τους πρέπει να χρησιμοποιούμε τις παρακάτω έννοιες [5]:

Πλήθος Επεξεργαστών →*P(n):* Ο αριθμός των επεξεργαστών που χρησιμοποιήθηκαν για να επιλύσουν ένα πρόβλημα μεγέθους n.
Παράλληλος Χρόνος Εκτέλεσης →*Tp(n):*Είναι ο χρόνος που χρειάζονται οι p επεξεργαστές, δουλεύοντας παράλληλα, να επιλύσουν ένα πρόβλημα μεγέθους n.
Κόστος →*C(n):* Είναι το γινόμενο του παράλληλου χρόνου εκτέλεσης επί τον αριθμό των επεξεργαστών που χρησιμοποιήθηκαν για να επιλυθεί παράλληλα,
ένα πρόβλημα μεγέθους n. $Cp(n) = Tp(n) \bullet P(n)$.

Χρόνος καλύτερου σειριακού αλγορίθμου → *T *(n):* Είναι ο χρόνος που χρειάζεται ένας επεξεργαστής, να επιλύσει ένα πρόβλημα εκτελώντας τον καλύτερο σειριακό αλγόριθμο.
Επιτάχυνση → *Sp(n):* λόγος του χρόνου εκτέλεσης του καλύτερου σειριακού αλγορίθμου που επιλύει ένα πρόβλημα Α δια το χρόνο εκτέλεσης του παράλληλου αλγορίθμου που επιλύει το πρόβλημα Α μεγέθους n. *Sp(n) = T *(n) /Tp(n).*

Ο χρόνος και το κόστος του κάθε αλγορίθμου μπορούν να χρησιμοποιηθούν ως μέτρο σύγκρισης με άλλους αλγορίθμους. Έχοντας λοιπόν δύο παράλληλους αλγορίθμους έστω Α και Β μπορούμε να πούμε ότι ο Α είναι πιο αποδοτικός σε θέμα χρόνου από τον Β αν $T_A(n) = O(T_B(n))$ ενώ είναι πιο αποδοτικός σε θέμα κόστους από τον Β αν ισχύει ότι $C_A(n) = O(C_B(n))$. Το ποια προσέγγιση είναι καλύτερη εξαρτάται από την εκάστοτε εφαρμογή. Αν η ταχύτητα του αλγορίθμου δεν είναι το πρωτεύον ζήτημα, τότε το κόστος είναι προτιμότερο.

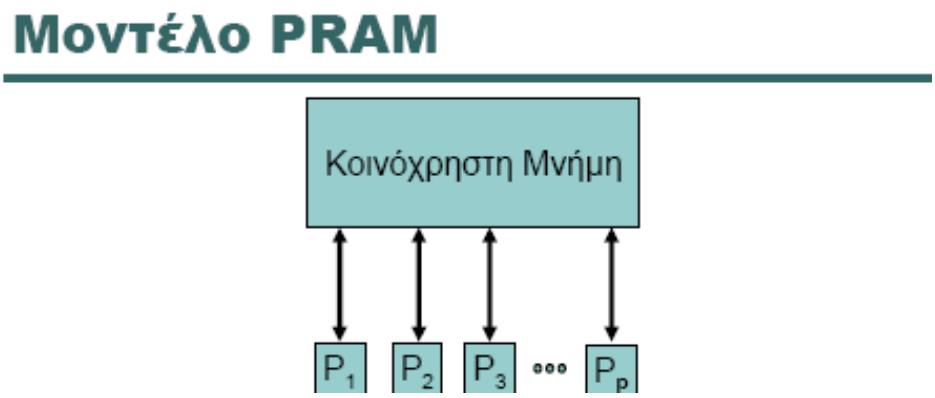
Τέλος προκειμένου να κρίνουμε αν ένας αλγόριθμος είναι βέλτιστος πρέπει να συγκρίνουμε το κόστος του με τον χρόνο του καλύτερου σειριακού αλγορίθμου που επιλύει το ίδιο πρόβλημα. Ένας λοιπόν παράλληλος αλγόριθμος θεωρείται βέλτιστος αν το κόστος εκτέλεσης του είναι μικρότερο από τον χρόνο εκτέλεσης του ταχύτερου σειριακού C(n) = O(T*(n)).

2.2 Το μοντέλο PRAM

Βασική προϋπόθεση για να θεωρηθεί ένα σύστημα παράλληλο είναι να υπάρχει πάνω από ένας επεξεργαστής στον ίδιο υπολογιστή. Η χρήση των πολλαπλών παράλληλων επεξεργαστών πάνω στο ίδιο υπολογιστικό σύστημα εισάγει μερικές επιπρόσθετες απαιτήσεις στον σχεδιασμό των αλγορίθμων. Μερικές από αυτές είναι για το πώς γίνεται η ανάθεση εργασιών στους επεξεργαστές, που θα αποθηκεύονται τα δεδομένα εισόδου, τα ενδιάμεσα αποτελέσματα και τα δεδομένα εξόδου, για το πώς οι επεξεργαστές θα έχουν πρόσβαση στο χώρο αποθήκευσης των δεδομένων και τέλος για το πώς θα επικοινωνούν μεταξύ τους αυτοί οι επεξεργαστές. Υπάρχουν, προς το παρόν,

δύο βασικές αρχιτεκτονικές προσεγγίσεις για την εκπλήρωση αυτών των απαιτήσεων: η κοινόχρηστη μνήμη και το δίκτυο αλληλοσύνδεσης. Στα συστήματα κοινόχρηστης μνήμης όλοι οι μεμονωμένοι επεξεργαστές έχουν να προσπελάσουν μια κοινή μνήμη, ενώ τους γίνεται επιτρεπτή η κοινή χρήση των τιμών δεδομένων και δομών δεδομένων που είναι αποθηκευμένες στην μνήμη αυτή. Στα συστήματα δικτύων αλληλοσύνδεσης, ο κάθε επεξεργαστής έχει την δική του τοπική μνήμη. Οι επεξεργαστές μοιράζονται τα δεδομένα μεταξύ τους με μετάβαση μηνυμάτων. Η εργασία αυτή θα επικεντρωθεί στον προγραμματισμό των συστημάτων κοινόχρηστης μνήμης και πιο συγκεκριμένα του PRAM.

Το PRAM (Parallel Random Access Machine) [4] είναι ένα μοντέλο παράλληλου υπολογισμού που ανήκει στην κατηγορία των μοντέλων **SIMD** (**S**ingle **I**nstruction **M**ultiple **D**ata). Η κατηγορία αυτή έχει την ιδιότητα ότι σε κάθε βήμα όλοι οι επεξεργαστές εκτελούν την ίδια εντολή πάνω σε διαφορετικά δεδομένα [4]



Σχήμα 2.1[5]

Στο μοντέλο PRAM λοιπόν έχουμε p πανομοιότυπους, συγχρονισμένους επεξεργαστές οι οποίοι χρησιμοποιούν την κοινόχρηστη μνήμη ως μέσο επικοινωνίας, όλοι όμως έχουν και τοπική/προσωπική μνήμη με χρόνο πρόσβασης Θ(1). Ο ένας επεξεργαστής γράφει σε μια κυψελίδα της κοινόχρηστης μνήμης και στην συνέχεια κάποιος άλλος διαβάζει το μήνυμα από αυτή την κυψελίδα. Τα δεδομένα εισόδου, τα ενδιάμεσα

αποτελέσματα και τα δεδομένα εξόδου αποθηκέυονται στην κοινόχρηστη μνήμη. Ο χρόνος πρόσβασης της κοινόχρηστης μνήμης από τον κάθε επεξεργαστή παίρνει σταθερό χρόνο Θ(1).

Το μοντέλο PRAM χωρίζεται σε 4 βασικούς τύπους ως προς τους περιορισμούς που έχουν για την πρόσβαση στην κοινόχρηστη μνήμη.

1. **EREW (E**xclusive **R**ead**, E**xclusive **W**rite**)** Σ' αυτό το μοντέλο μόνο ένας επεξεργαστής μπορεί να βρίσκεται ανά πάσα στιγμή σε μια κυψελίδα της κοινόχρηστης μνήμης. Δεν μπορούν περισσότεροι από ένας επεξεργαστές να διαβάσουν ή να γράψουν ταυτόχρονα στην ίδια κυψελίδα**.** . Είναι το πιο περιοριστικό (αδύνατο) μοντέλο.

2. **CREW (C**oncurrent **R**ead, **E**xclusive **W**rite**)** Όλοι οι επεξεργαστές έχουν την δυνατότητα να διαβάσουν από την ίδια κυψελίδα ταυτόχρονα αλλά μόνο ένας μπορεί να γράψει στην ίδια κυψελίδα.

3. **ERCW (E**xclusive **R**ead, **C**oncurrent **W**rite) Σ΄ αυτό το μοντέλο επιτρέπεται η ταυτόχρονη εγγραφή στην ίδια κυψελίδα, αλλά δεν επιτρέπεται η ταυτόχρονη ανάγνωση από περισσότερους από έναν επεξεργαστές κατά την ίδια χρονική στιγμή στην ίδια κυψελίδα της κοινόχρηστης μνήμης

4. **CRCW (C**oncurrent **R**ead, **C**oncurrent **W**rite) Εδώ δεν υπάρχει κανένας περιορισμός στην ταυτόχρονη πρόσβαση στην μνήμη. Έχουν την δυνατότητα ένας ή και περισσότεροι επεξεργαστές να βρίσκονται την ίδια χρονική στιγμή στην ίδια κυψελίδα της κοινόχρηστης μνήμης για να γράψουν ή και να διαβάσουν. Είναι το πιο ελαστικό (δυνατό) μοντέλο.

Στην περίπτωση που έχουμε ταυτόχρονο διάβασμα δεν προκύπτει κανένα πρόβλημα γιατί το μόνο που κάνουν οι επεξεργαστές είναι να διαβάζουν την τιμή που υπάρχει μέσα σε μια συγκεκριμένη κυψελίδα μνήμης την ίδια χρονική στιγμή. Η ενδιαφέρουσα περίπτωση προκύπτει όταν έχουμε ταυτόχρονο γράψιμο όπου εκεί μπορεί να έχουμε κάποια σύγχυση. Για τον λόγω αυτό το PRAM CRCW μοντέλο έχει χωριστεί σε κάποιους υποτύπους με βάση την ταυτόχρονη γραφή στην ίδια κυψελίδα:

1. **Common CRCW PRAM** όταν οι επεξεργαστές που λαμβάνουν μέρος στην ταυτόχρονη γραφή στην ίδια κυψελίδα της κοινόχρηστης μνήμης θέλουν να γράψουν την ίδια τιμή.

2. **Arbitrary CRCW PRAM** σε αυτό το μοντέλο επιλέγεται αυθαίρετα ένας επεξεργαστής για να υπερισχύσει η δική του τιμή και να γραφτεί στην κυψελίδα της κοινόχρηστης μνήμης.

3. **Priority CRCW PRAM** ο επεξεργαστής που γράφει την δική του τιμή στην κυψελίδα είναι αυτός με την μεγαλύτερη προτεραιότητα. Η προτεραιότητα συνήθως καθορίζεται από τον προσωπικό αριθμό του κάθε επεξεργαστή. Αν δηλαδή έχουμε n επεξεργαστές P1,P2,P3,…..,Pn αυτός με την μεγαλύτερη προτεραιότητα είναι ο επεξεργαστής Pn.

Η ιεραρχία δυναμικότητας φαίνεται στην πιο κάτω σειρά:
EREW → CREW → Common CRCW → Arbitrary CRCW → Priority CRCW
Ένας αλγόριθμος σχεδιασμένος για ένα πιο αδύνατο μοντέλο μπορεί να εκτελεστεί σε ένα πιο δυνατό με την ίδια πολυπλοκότητα κόστους και χρόνου.
Αντιθέτως ένας αλγόριθμος σχεδιασμένος για ένα πιο δυνατό μοντέλο μπορεί να προσομοιωθεί σε ένα πιο αδύνατο, είτε με ασυπτωτικά περισσότερους επεξεργαστές ή με ασυμπτωτικά μεγαλύτερο χρόνο εκτέλεσης.

Κλείνοντας, ένας σχεδιαστής παράλληλων αλγορίθμων πρέπει να έχει ένα συγκεκριμένο μοντέλο υπόψη του όταν σχεδιάζει ένα αλγόριθμο, ούτως ώστε να μπορεί να τον προσαρμόσει στις απαιτήσεις του μοντέλου αυτού.

Όπως έχει αναφερθεί, στο μοντέλο PRAM έχουμε p πανομοιότυπους, συγχρονισμένους επεξεργαστές να εργάζονται παράλληλα για την ολοκλήρωση μιας διεργασίας. Τι συμβαίνει όμως τώρα αν οι επεξεργαστές που χρησιμοποιούμε δεν είναι συγχρονισμένοι ή αν κάποιος από αυτούς τους επεξεργαστές καταρρεύσει; Εδώ εισάγονται δύο άλλα μοντέλα, το μοντέλο A-PRAM και το μοντέλο F-PRAM. Τα δύο αυτά μοντέλα κληρονομούν από το PRAM πολλά χαρακτηριστικά του, ενώ μπορούμε να τα διακρίνουμε με κάποιες μικρές διαφορές που έχουν [2].

Στο μοντέλο A-PRAM οι επεξεργαστές ενεργούν εντελώς ασυγχρόνιστα. Μπορούν να γράψουν και να διαβάσουν σε οποιαδήποτε κυψελίδα της κοινόχρηστης μνήμης, όμως κάθε φορά η πρόσβαση στην μνήμη μπορεί να πάρει διαφορετικό χρόνο. Βασικά σε αυτό το μοντέλο ο χρόνος δεν παίζει ουσιαστικό ρόλο στο σχεδιασμό αλγορίθμων, αλλά το κόστος θεωρείται πιο σημαντικό. Μια ακόμη διαφορά από το μοντέλο PRAM είναι ότι στο μοντέλο αυτό δεν έχουμε ταυτόχρονη πρόσβαση σε μια κυψελίδα της κοινόχρηστης μνήμης αλλά έχουμε την λεγόμενη ατομική πρόσβαση, όπου κάθε ακολουθία ασυγχρόνιστων προσβάσεων από τον ίδιο ή και διαφορετικούς επεξεργαστές σε μια συγκεκριμένη κυψελίδα μνήμης μπορεί να τοποθετηθεί σε μερική διάταξη. Με αυτό τον τρόπο η τιμή που θα επιστρέφεται κατά την ανάγνωση από μια κυψελίδα της κοινόχρηστης μνήμης θα είναι η τιμή της τελευταίας γραφής η οποία έχει ολοκληρωθεί ή η τιμή μιας γραφής η οποία συμβαίνει την ίδια χρονική στιγμή με την ανάγνωση και δεν έχει ολοκληρωθεί ακόμη. Παρομοίως η τιμή που θα επιστραφεί κατά την επόμενη ανάγνωση της κυψελίδας είτε θα είναι ίδια με την προηγούμενη ανάγνωση είτε θα έχει την τιμή μιας γραφής που εκτελέστηκε μετά από την προηγούμενη ανάγνωση.

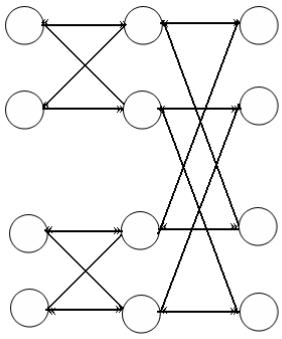
Το μοντέλο F-PRAM δεν έχει και πολλές διαφορές από το μοντέλο PRAM. Η μόνη σημαντική διαφορά είναι ότι οι επεξεργαστές υπόκεινται σε σφάλματα ενώ στο μοντέλο PRAM δεν έχουμε σφάλματα κατάρρευσης. Τα σφάλματα κατάρρευσης συμβαίνουν όταν κάποιος επεξεργαστής σταματήσει να λειτουργεί σε οποιοδήποτε σημείο του υπολογισμού και δεν επιστρέψει ξανά πίσω. Ένα πρόβλημα λέμε ότι έγκειται σε f σφάλματα κατάρρευσης αν από την αρχή του προβλήματος μέχρι και την επίλυση του καταρρεύσουν το πολύ f επεξεργαστές. Πρέπει όμως ένας οποιοσδήποτε επεξεργαστής να μην καταρρεύσει ποτέ έτσι ώστε να ολοκληρώσει την επίλυση του προβλήματος.
Ένας παράλληλος αλγόριθμος σχεδιασμένος για ένα συγκεκριμένο πρόβλημα ονομάζεται εύρωστος, αν επιλύει το πρόβλημα αποδοτικά και ταυτόχρονα ανέχεται σφάλματα επεξεργαστών.

2.3 Προσομοιωτής SB-PRAM

Για την ανάγκη λοιπόν της εξομοίωσης της συμπεριφοράς του PRAM δημιουργήθηκε η μηχανή SB-PRAM, η οποία είναι μια πρακτική πραγματοποίησης της προσέγγισης του Ranade. Η μηχανή αυτή λοιπόν αρχικά δημιουργήθηκε στο Universität des Saarlades, Saarbücken, στην Γερμανία, το 1990 με 64 φυσικούς επεξεργαστές. Αν και όμως εξομοιώνει την συμπεριφορά του PRAM έχει μια σημαντική ιδιαιτερότητα. Εδώ οι επεξεργαστές προτού επικοινωνήσουν με την κοινόχρηστη μνήμη συνδέονται με ένα δίκτυο σε σχήμα «πεταλούδας» (**βλ. Σχήμα 2.2**). Πιο συγκεκριμένα το SB-PRAM έχει τις ιδιότητες που έχει το μοντέλο priority CRCW PRAM. Η γενική ροή σχεδίασης ενός προς ανάπτυξης προγράμματος, στο SB-PRAM δεν διαφέρει και πολύ από την ανάπτυξη προγραμμάτων στην γλώσσα προγραμματισμού C. Τα προγράμματα όμως είναι καλύτερο να γραφτούν στην γλώσσα προγραμματισμού FORK η οποία είναι μια γλώσσα προγραμματισμού που εκμεταλλεύεται τις ιδιότητες του παραλληλισμού. Όμως η γλώσσα προγραμματισμού FORK έχει τις ιδιότητες του arbitrary CRCW PRAM μοντέλου. Το μοντέλο λοιπόν που υπερισχύει στην εκτέλεση των αλγορίθμων είναι το μοντέλο που υποστηρίζει το SB-PRAM δηλαδή το priority CRCW PRAM λόγω του ότι είναι πιο δυνατό από το arbitrary [7].

Λόγω του ότι λοιπόν αυτή η μηχανή υπάρχει μόνο στη Γερμανία δημιουργήθηκε ο προσομοιωτής **SB-PRAM, ο pramsim [1]** ο οποίος έχει την ιδιότητα να προσομοιώνει την συμπεριφορά του **SB-PRAM** στις δικές μας σειριακές μηχανές. Επιτρέπει δηλαδή την εκτέλεση PRAM παράλληλων αλγορίθμων σε σειριακούς υπολογιστές με την βοήθεια εικονικών επεξεργαστών. Δημιουργήθηκε από τους M.Bosch και S.Franziskus για να παρέχει στον προγραμματιστή την ιδέα του PRAM αλλά και του A-PRAM, δηλαδή των συγχρονισμένων και ασυγχρόνιστων εικονικών επεξεργαστών που χρησιμοποιούν την κοινόχρηστη μνήμη.

Σχήμα 2.1 (πρωτότυπο SB-PRAM)



Σχήμα 2.2 (δίκτυο πεταλούδας)

Κεφάλαιο 3

Περιγραφή της γλώσσας προγραμματισμού FORK

3.1 Η γλώσσα FORK

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση των αλγορίθμων είναι η γλώσσα FORK [3].

Η γλώσσα προγραμματισμού FORK είναι μια υψηλού επιπέδου γλώσσα με ένα απλό σύνολο αφαιρέσεων του παράλληλου προγραμματισμού, η οποία διαθέτει επαρκή δύναμη στην αναπαράσταση παράλληλων αλγορίθμων για τα συστήματα κοινόχρηστης μνήμης (PRAM). Πιο συγκεκριμένα, ακολουθεί τις ιδιότητες του μοντέλου Arbitary CRCW PRAM στο οποίο όταν έχουμε ταυτόχρονο γράψιμο στην ίδια κυψελίδα της κοινόχρηστης μνήμης επιλέγεται αυθαίρετα ένας επεξεργαστής για να καταγράψει την τιμή του. Ωστόσο, υπάρχει μια μικρή διαφορά. Στην γλώσσα FORK κάθε επεξεργαστής δεν διαθέτει προσωπική μνήμη αλλά η προσωπική του μνήμη είναι εντοιχισμένη μέσα στην κοινόχρηστη. Η FORK είναι βασισμένη στην γλώσσα ANSI C και γι' αυτό το λόγο βλέπουμε πολλά δεδομένα χαρακτηριστικά τους να είναι τα ίδια [1].

Έχει ιδιότητες που επιτρέπουν τη δυναμική δημιουργία παράλληλων διεργασιών που εκτελούνται σε φυσικούς επεξεργαστές. Η φιλοσοφία της γλώσσας δεν είναι να

ξεκινήσει μια σειριακή διεργασία και έπειτα να δημιουργήσει καινούργιες όπως συμβαίνει και στις περισσότερες MIMD (**M**ultiple **I**nstruction **M**ultiple **D**ata) γλώσσες προγραμματισμού όπου σε κάθε βήμα όλοι οι επεξεργαστές μπορούν να εκτελέσουν διαφορετική εντολή πάνω σε διαφορετικά δεδομένα, αλλά να πάρει όλους τους επεξεργαστές και να διανέμει σε αυτούς την διεργασία την οποία θα πρέπει να ολοκληρώσουν, ξεκινώντας όλοι την εκτέλεση τους από την κύρια συνάρτηση του προγράμματος, την συνάρτηση main(). Από την στιγμή που μια διεργασία ανατίθεται σε κάποιον επεξεργαστή τότε ξεκινάει και η υπολογιστική δραστηριότητα στο παράλληλο σύστημα. Μπορούμε ανεπίσημα βεβαίως, να θεωρήσουμε την διεργασία ως ένα τμήμα κώδικα του προγράμματος, το οποίο ανατίθεται σε κάποιον επεξεργαστή προς εκτέλεση.

Τέλος η FORK έχει σχεδιαστεί για το μοντέλο SB-PRAM. Αυτή η γλώσσα προγραμματισμού αναπτύχθηκε το 1994 από τους Christoph W. Kessler και Helmut Seidl , οι οποίοι ξεκίνησαν από μια αρχική έκδοση της γλώσσας ονομαζόμενη σαν Fork95 όπου έπειτα επεκτάθηκε αρκετές φορές κυρίως στο διάστημα από το 1995 μέχρι το 1999. Τελικά το 1999 η γλώσσα έφτασε στο τελικό της στάδιο καταλήγοντας με το όνομα FORK [6].

3.2 Μικρό παράδειγμα στη γλώσσα FORK.

<u>Αλγόριθμος μετάδοσης στοιχείου στο EREW PRAM</u>

Περιγραφή αλγορίθμου

Δεδομένου λοιπόν κάποιου μηνύματος που είναι αποθηκευμένο στην πρώτη κυψελίδα της κοινόχρηστης μνήμης μας ζητείται να το μεταδώσουμε σε n κυψελίδες της κοινόχρηστης μνήμης. Αρχικά το στοιχείο που πρέπει να μεταδοθεί βρίσκεται αποθηκευμένο στην πρώτη κυψελίδα της κοινόχρηστης μνήμης SM [1]. Κατά την εκτέλεση του βήματος 1 ο επεξεργαστής P1 μεταδίδει το στοιχείο που αντιστοιχεί στην κυψελίδα με τον προσωπικό του αριθμό στην θέση SM [2] της κοινόχρηστης μνήμης. Έπειτα στο βήμα 2 οι ενεργοί επεξεργαστές διπλασιάζονται διαβάζουν το στοιχείο που αντιστοιχεί στην θέση με τον προσωπικό τους αριθμό και το μεταδίδουν στις δύο συνεχόμενες θέσεις όπου δεν έχει ακόμη μεταδοθεί. Συνεχίζοντας με αυτόν

τον τρόπο, θα φτάσουμε στο βήμα log n όπου οι ενεργοί επεξεργαστές τώρα θα είναι ίσοι με n/2 λόγω του ότι τους διπλασιάζουμε συνεχώς σε κάθε βήμα, θα διαβάσουν το στοιχείο που αντιστοιχεί στην θέση με τον προσωπικό τους αριθμό και θα μεταδώσουν αυτό το στοιχείο σε n/2 συνεχόμενες θέσεις στις οποίες το στοιχείο δεν έχει ακόμη μεταδοθεί. Τότε θα έχουμε τελειώσει και το στοιχείο θα βρίσκεται αποθηκευμένο και στις n κυψελίδες της κοινόχρηστης μνήμης.

Ψευδοκώδικας αλγορίθμου

```
#include <fork.h>
#include <io.h>
#include <math.h>
#include <syscall.h>
sh int n;
sh int table [100000]; //pinakas ston opoio metadidoyme to stoixeio

async int main(void){
pr int i=1;
pr int a=0;
pr int b=0;
pr int counter=0;
pr int thesi;

 if($==0)   {   printf("Dose to plithos ton kipselidon pou theleis na metadothei to stoixeio:\n");
scanf("%d",&n);  }
  table [1]=1;//I proti thesi tou pinaka arxikopoiite me to stoixeio to opoio theloume na metadosoume
  barrier;
start{
 if($!=0)
   {   if(n/2>_STARTED_PROCS_)
 {while((2^b)<=n || table[n]!=table[1] && counter<n/_STARTED_PROCS_) {
     if($<=(2^b)){
   table[$+(2^b)]=table[$];}
   b=b+1;
counter=counter+1;   }
thesi=($*(n/_STARTED_PROCS_))-n/_STARTED_PROCS_+(2^b)+1;
  while(thesi<n/_STARTED_PROCS_)  {
  table[thesi]=table[$];
  thesi=thesi+1;} }
```

```
 if(n/2<=_STARTED_PROCS_){if($<=n/2)
{    while((2^b)<=n || table[n]!=table[1])
    {  if($<=(2^b))
     { table[$+(2^b)]=table[$];}
 b=b+1      }} }} }
barrier;
fp=fopen("input3.txt","w");
     if(fp==NULL)
     {   pprintf("ERROR The file does not exist\n");   exit(-1);
                }if($==0)
{for(i=1;i<=n;i++)
  {fprintf(fp,"Stin thesi %d exei metadothei to stoixeio %d\n",i,table[i]);
  } fclose(fp);
   barrier;
 return 0;}
```

Πιο κάτω επεξηγούνται τα συστατικά στοιχεία της γλώσσας FORK τα οποία βλέπουμε σκιασμένα και στον αλγόριθμο. Όλες οι πιο κάτω πληροφορίες έχουν αντληθεί από το [1].

3.3 Συστατικά στοιχεία της γλώσσας FORK

Αρχικά σε κάθε πρόγραμμα που είναι γραμμένο στην γλώσσα FORK χρειάζεται να ενσωματώσουμε ορισμένες απαραίτητες βιβλιοθήκες. Την βιβλιοθήκη <fork.h> και την βιβλιοθήκη <io.h> η οποία περιέχει συναρτήσεις εισόδου και εξόδου όπως και τις συναρτήσεις printf() και pprintf(). Οι printf() και pprintf() μας βοηθάνε να τυπώσουμε κάποιο μήνυμα με την διαφορά ότι στη πρώτη όλοι οι επεξεργαστές τυπώνουν ταυτόχρονα το μήνυμα με αποτέλεσμα να τυπωθεί το κάθε γράμμα του μηνύματος όσο είναι και το συνολικό πλήθος των επεξεργαστών. Αντιθέτως στην συνάρτηση pprintf() ο κάθε επεξεργαστής τυπώνει ολόκληρο το μήνυμα.

3.3.1 Δήλωση μεταβλητών στη γλώσσα FORK

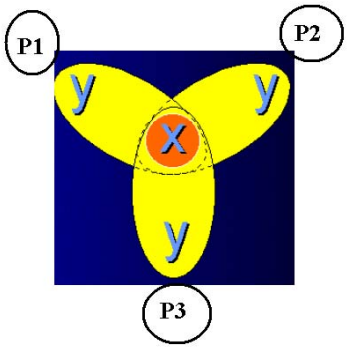
Δεδομένου ότι η γλώσσα έχει πολλές ομοιότητες με την γλώσσα προγραμματισμού C, οι ιδιότητες για τις δηλώσεις μεταβλητών είναι οι ίδιες. Επειδή όμως, η δημιουργία μεταβλητών είναι πολύ σημαντική στα παράλληλα προγράμματα, η FORK έχει μια

πρόσθετη ιδιότητα γι' αυτό το σκοπό. Πριν από τον τύπο της κάθε μεταβλητής πρέπει να δηλώνεται αν η μεταβλητή αυτή είναι κοινόχρηστη ή προσωπική. Στις κοινόχρηστες μεταβλητές ισχύει ότι η τιμή τους είναι κοινή για όλους τους επεξεργαστές. Δηλαδή όταν κάποιος επεξεργαστής αλλάξει την τιμή μιας κοινόχρηστης μεταβλητής τότε όλοι οι επεξεργαστές που θα διαβάσουν αυτή την μεταβλητή μετά από την αλλαγή, θα διαβάσουν την νέα τιμή. Οι μεταβλητές που δηλώνονται στην αρχή του κυρίως προγράμματος αλλά και στα κομμάτια κώδικα που είναι συγχρονισμένα, είναι απαραίτητα κοινόχρηστες. Αυτές τις μεταβλητές μπορούμε να της δηλώσουμε γράφοντας πριν από τον τύπο της μεταβλητής το sh από την αγγλική λέξη shared.

πχ *sh int x;*

Στις προσωπικές μεταβλητές ισχύει ότι η τιμή τους είναι διαφορετική για κάθε επεξεργαστή. Δηλαδή όταν κάποιος επεξεργαστής αλλάξει την τιμή μιας προσωπικής μεταβλητής και κάποιος άλλος επεξεργαστής θελήσει να διαβάσει αυτή την μεταβλητή μετά από την αλλαγή, τότε αυτός ο επεξεργαστής δεν θα διαβάσει την τιμή που έχει γραφτεί από τον άλλο επεξεργαστή αλλά θα διαβάσει την τιμή που έχει αποθηκευτεί στην μεταβλητή μετά από την αλλαγή που έχει κάνει ο ίδιος. Οι μεταβλητές που δηλώνονται σε κομμάτια κώδικα αλλά και συναρτήσεις που είναι ασυγχρόνιστες, είναι απαραίτητα προσωπικές. Αυτές τις μεταβλητές μπορούμε να της δηλώσουμε γράφοντας πριν από τον τύπο της μεταβλητής το pr από την αγγλική λέξη private.

πχ *pr int y;*



Σχήμα 3.1

3.3.2 Μεταβλητές του συστήματος

_STARTED_PROCS_ : Χρησιμοποιώντας αυτή τη μεταβλητή μπορούμε να έχουμε πρόσβαση στον αριθμό των συνολικών επεξεργαστών.
$: Αυτή η μεταβλητή μας δηλώνει την ταυτότητα του κάθε επεξεργαστή. Οι ταυτότητες των επεξεργαστών είναι αριθμημένες από το 0 μέχρι και το (_STARTED_PROCS_ -1).

3.3.3 Είδη εκτέλεσης και δηλώσεις συναρτήσεων

Η FORK προσφέρει δύο είδη εκτέλεσης, την συγχρονισμένη και την ασυγχρόνιστη. Από το παραπάνω πρόγραμμα παρατηρούμε ότι υπάρχει μια εντολή, η εντολή ***start{}*** η οποία περιέχει μέσα της ένα κομμάτι κώδικα. Με αυτή την εντολή όλοι οι επεξεργαστές συγχρονίζονται και εκτελούν το αντίστοιχο κομμάτι κώδικα. Η συγκεκριμένη εντολή μπορεί να χρησιμοποιηθεί μόνο σε ασυγχρόνιστες συναρτήσεις που ξεκινούν με την εντολή ***async*** που δηλώνεται πριν από τον τύπο της συνάρτησης. Η συνάρτηση ***main()*** είναι εξ ορισμού ασυγχρόνιστη ***async int main(void){…}.***
Το άλλο είδος εκτέλεσης που αν και δεν χρησιμοποιείται στο παραπάνω πρόγραμμα αλλά θεωρείται εξίσου σημαντικό και θα φανεί χρήσιμο για τα άλλα προγράμματα που έχουν υλοποιηθεί είναι το ασυγχρόνιστο είδος εκτέλεσης. Σ' αυτό το είδος εκτέλεσης τα κομμάτια κώδικα που θα εκτελεστούν ασυγχρόνιστα από όλους τους επεξεργαστές ξεκινούν με την εντολή ***farm{..}.*** Μια επιπρόσθετη εντολή για το συγκεκριμένο είδος εκτέλεσης είναι και η εντολή ***seq{..}*** στην οποία όμως το κομμάτι κώδικα που βρίσκεται μέσα σε αυτή την εντολή εκτελείται από έναν επεξεργαστή ο οποίος επιλέγεται αυθαίρετα. . Οι συγκεκριμένες εντολές μπορούν να χρησιμοποιηθούν μόνο σε συγχρονισμένες συναρτήσεις που ξεκινούν με την εντολή **sync** η οποία δηλώνεται πριν από τον τύπο της συνάρτησης.
sync int foo( ){…}.

Οι συγχρονισμένες συναρτήσεις μπορούν να καλεστούν μόνο σε συγχρονισμένες περιοχές δηλαδή σε κομμάτια κώδικα που ξεκινούν με το ***start*** αλλά και σε άλλες συγχρονισμένες συναρτήσεις. Αντιθέτως οι ασυγχρόνιστες συναρτήσεις μπορούν να καλεστούν μόνο σε ασυγχρόνιστες περιοχές, δηλαδή σε περιοχές που ξεκινούν με ***farm*** και ***seq*** αλλά και σε άλλες ασυγχρόνιστες συναρτήσεις.

3.3.4 Εντολή barrier

Αυτή η εντολή θεωρείται πολύ σημαντική στην υλοποίηση παράλληλων προγραμμάτων και αυτό φαίνεται και από το πιο πάνω πρόγραμμα αφού την χρησιμοποιούμε αρκετές φορές. Η εντολή barrier λοιπόν μπορεί να χρησιμοποιηθεί σε ασυγχρόνιστα κομμάτια κώδικα και η λειτουργία της είναι ότι όταν κάποιος επεξεργαστής την εκτελεί τον υποχρεώνει να περιμένει μέχρι όλοι οι επεξεργαστές του συνόλου που ανήκει να φτάσουν στο σημείο που βρίσκεται και να εκτελέσουν την εντολή barrier. Είναι σαν να συγχρονίζει όλους τους επεξεργαστές για να συνεχίσουν την εκτέλεση του κώδικα ανεξάρτητα αν αυτοί ακολουθήσουν διαφορετικά μονοπάτια ροής δεδομένων.
Για καλύτερη κατανόηση της χρήσης αυτής της εντολής παραθέτονται κάποιες εκτελέσεις ενός μικρού προγράμματος που δείχνει την χρήση αυτής της εντολής.

```
#include <fork.h>
#include <io.h>
int async main(void){
barrier;
if($==0){
printf("@@@@@@@@@@@@@@@\n");
pprintf("@ HELLO WORD @\n");
pprintf("@@@@@@@@@@@@@@@\n\n");}
pprintf("Processor %d say HELLO \n",$);
barrier;
return (0);}
```

```
cs05np1@athos:~/bin/pram/pramsim>pramsim -p 5 -v 1 $EXAMPL
       ***** PRAM-SIMULATOR (v2.0) *****
          >>>>>>>> DAMN FAST <<<<<<<<
       (c) by hirbli & stefran  07.10.94
You have 5 physical processors with 1 vP's each

Relocating file /home/students/cs/2005/cs05np1/../../../..
Loading file /var/tmp/aaaqSayRZ.cod...Doing realloc
 done
PRAM P0 = (p0, v0)> g
@@@@@@@@@@@@@@@
@ HELLO WORD @
@@@@@@@@@@@@@@@

#0001# Processor 1 say HELLO
#0002# Processor 2 say HELLO
#0003# Processor 3 say HELLO
#0004# Processor 4 say HELLO
EXIT: vp=#1, pc=$000001c6
EXIT: vp=#2, pc=$000001c6
EXIT: vp=#3, pc=$000001c6
EXIT: vp=#4, pc=$000001c6
Stop nach 19432 Runden, 3238.667 kIps
012a C0700005  BLT     12f
```

```
#include <fork.h>
#include <io.h>
int async main(void){
if($==0){
printf("@@@@@@@@@@@@@@@\n");
pprintf("@ HELLO WORD @\n");
pprintf("@@@@@@@@@@@@@@@\n\n");}
pprintf("Processor %d say HELLO \n",$);
return (0);
```

```
#0000# @@@@@@@@@@@@@@@

#0003# Processor 3 say HELLO
#0004# Processor 4 say HELLO
#0001# Processor 1 say HELLO
#0002# Processor 2 say HELLO
#0000# Processor 0 say HELLO
EXIT: vp=#0, pc=$000001c6
Stop nach 12963 Runden, 6481.500 kIps
01c6 18137FFF  POPNG   R6, ffffffff, R1
PRAM P0 = (p0, v0)>
```

3.4 Βήματα για να μπορέσει να εκτελεστεί ένα πρόγραμμα.

Αφού έχουμε ολοκληρώσει την συγγραφή του προγράμματος, απομένει τώρα να το μεταγλωττίσουμε και να το εκτελέσουμε. Είδη ο compiler ***fcc***, ο assembler ***prass*** και ο linker ***ld*** υπήρχαν εγκατεστημένοι σε μια μηχανή SOLARIS του Πανεπιστημίου Κύπρου, την μηχανή **athos,** για τις ανάγκες μιας προηγούμενης διπλωματικής εργασίας. Επομένως το μόνο που απομένει είναι οι εντολές που χρειάζονται για να τρέξουμε το πρόγραμμα μας.

1. Μεταγλωττίζουμε το πρόγραμμα χρησιμοποιώντας τον μεταγλωττιστή ***fcc*** με την εντολή:

fcc –m –I$FORKDIR/include parrallili_ -o parrallili

όπου $FORKDIR/include είναι το μονοπάτι στο οποίο βρίσκονται οι βιβλιοθήκες που χρησιμοποιούμε μέσα στο πρόγραμμα. Μετά την εκτέλεση αυτής της εντολής δημιουργείτε το εκτελέσιμο αρχείο parrallil.o.

2. Εκτελούμε το εκτελέσιμο αυτό αρχείο καλώντας τον pramsim:

pramsim –p 4 –v 1 parralli

Η εντολή αυτή θα εκτελέσει τον προσομοιωτή με 4 επεξεργαστές.

Κεφάλαιο 4

Αλγόριθμος παράλληλης άθροισης

4.1 Πρόβλημα παράλληλης άθροισης

Δεδομένου μιας λίστας n αριθμών a1,…..,an μας ζητείται να υπολογίσουμε το άθροισμα αυτών των αριθμών [3].

4.2 Σειριακή λύση

Εκτελώντας λοιπόν τον σειριακό αλγόριθμο στον οποίο λαμβάνει μέρος ένας επεξεργαστής όπου διαβάζει ένα, ένα τα στοιχεία από την λίστα και τα προσθέτει καταναλώνουμε χρόνο Θ(n) για να διαβάσουμε όλα τα στοιχεία από την λίστα και να τα προσθέσουμε μεταξύ τους.

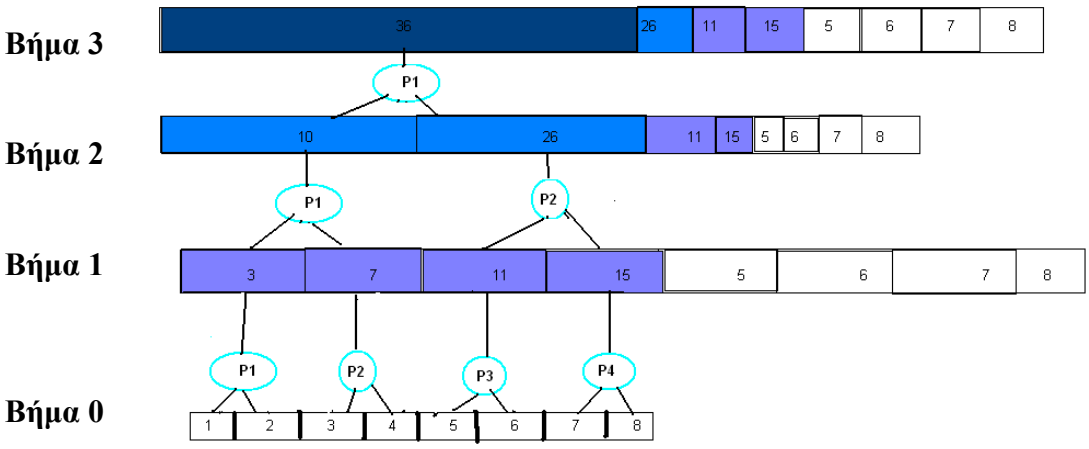
Αλγόριθμος Σειριακής Άθροισης

1. Set S = α1
2. For j = 2 to n do
3. S = S + αj
4. End do

Ο αλγόριθμος αυτός όντως αθροίζει και τα n στοιχεία δίνοντας μας το σωστό αποτέλεσμα και συνεπώς ο συνολικός χρόνος που χρειάζεται είναι Θ(n) λόγω του βήματος 2 όπου εκτελείται n-2 φορές και σε κάθε επανάληψη του κάνει πράξεις που παίρνουν σταθερό χρόνο. Ο χρόνος αυτός είναι αναπόφευκτος γιατί ο επεξεργαστής πρέπει να διαβάσει και τα n στοιχεία για να τα αθροίσει. Το κόστος του αλγορίθμου είναι και αυτό Θ(n) γιατί λαμβάνει μέρος στον υπολογισμό ένας μόνο επεξεργαστής.
Τι συμβαίνει τώρα αν εισάξουμε στο πρόβλημα μας περισσότερους επεξεργαστές οι οποίοι θα εργάζονται παράλληλά; Πόσους επεξεργαστές να εισάξουμε ούτως ώστε να μην έχουμε υπερβολική αύξηση του κόστους;

4.3 Περιγραφή παράλληλου αλγορίθμου

Οι επεξεργαστές που θα χρησιμοποιήσουμε δεν χρειάζεται να είναι περισσότεροι από n/2 αφού αυτό το πλήθος των επεξεργαστών είναι αρκετό για την επίλυση του προβλήματος. Προκειμένου λοιπόν να αθροίσουμε τα n στοιχεία με n/2 επεξεργαστές εργαζόμαστε ως εξής :



Σχήμα 4.1

Αρχικά τα στοιχεία του προβλήματος α1,α2,α3....,αn βρίσκονται αποθηκευμένα στην κοινόχρηστη μνήμη. Κατά την εκτέλεση του βήματος 0 ο κάθε επεξεργαστής Pi όπου

0< i ≤ n/2 παίρνει από 2 διαφορετικά στοιχεία (αi και ai+1) τα αθροίζει και τοποθετεί το άθροισμα τους στην κυψελίδα αi. Έπειτα στο βήμα 1 οι επεξεργαστές που εργάζονται μειώνονται στους μισούς των αρχικών όπου και πάλι ο κάθε επεξεργαστής Pi 0< i ≤n/4 παίρνει δυο στοιχεία αi και ai+1 τα αθροίζει και τοποθετεί το άθροισμα τους στην κυψελίδα αi. Οι επεξεργαστές οι οποίοι δεν είναι αναγκαίο να εργάζονται θεωρούμε ότι εκτελούν την εντολή no op .Η πιο πάνω διαδικασία επαναλαμβάνεται μέχρις στου να φτάσουμε στο log n -1 βήμα και ο επεξεργαστής P1 που έχει απομείνει να αποθηκεύσει το αποτέλεσμα στην πρώτη κυψελίδα της κοινόχρηστης μνήμης. Το κάθε βήμα εκτελείται παράλληλα από τους διαθέσιμους επεξεργαστές.

Αυτή είναι μια τεχνική ανάβασης ενός νοητού δυαδικού ισοσταθμισμένου δέντρου που με τον τερματισμό του αλγορίθμου το αποτέλεσμα θα βρίσκεται αποθηκευμένο στην ρίζα του. Για παράδειγμα όπως βλέπουμε και στο **Σχήμα4.1** όπου έχουμε 8 στοιχεία και 4 επεξεργαστές, ο επεξεργαστής P1 διαβάζει τα στοιχεία (1,2) εκτελεί την πράξη της πρόσθεσης και τοποθετεί το αποτέλεσμα τους στην κυψελίδα 1. Παράλληλα οι επεξεργαστές P2, P3 και P4 κάνουν την ίδια διαδικασία με τα στοιχεία (3,4), (5,6), (7,8)αντίστοιχα και τοποθετούν το αποτέλεσμα στις κυψελίδες 2,3,4 αντίστοιχα. Τέλος στο βήμα3 ο επεξεργαστής P1 αθροίζει τα στοιχεία των κυψελίδων 1 και 2 και τοποθετεί το άθροισμα τους, που στην συγκεκριμένη περίπτωση είναι το 36, στην κυψελίδα 1.

Αλγόριθμος Παράλληλης Άθροισης **[5]**

```
1. Processors i = 1 to n/2 do in parallel
2.   Set j=1
3.   while (i ≤ n/2^j)
4.      a = SM[2i-1]
5.      b = SM[2i]
6.      SM[i] = a + b
7.      j=j+1
8.    end while
9. end do
```

4.4 Θεωρητικός χρόνος και κόστος αλγορίθμου

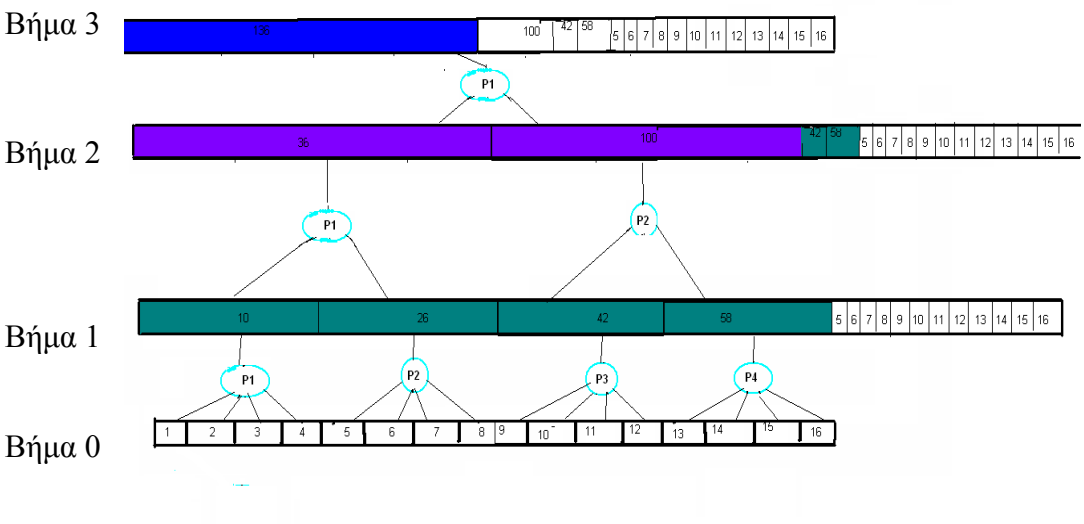
Τώρα ας αναλύσουμε τον χρόνο που χρειάζεται αυτός ο αλγόριθμος για να υπολογίσει το αποτέλεσμα και να τερματίσει. Κατ΄ αρχάς ο χρόνος που απαιτείται στο βήμα 3 είναι O(log n) αφού συνεχώς το n μειώνεται στο μισό άρα ο βρόγχος θα εκτελεστεί log n φορές. Στα βήματα 4, 5, 6 και 7 έχουμε χρόνο O(1) αφού ο κάθε επεξεργαστής διαβάζει, προσθέτει και γράφει το αποτέλεσμα όπου αυτές οι πράξεις παίρνουν σταθερό χρόνο. Άρα έχουμε O(log n) επαναλήψεις και η κάθε μια από αυτές καταναλώνει O(1) χρόνο. Συνεπώς ο συνολικός χρόνος που απαιτείται είναι O(log n). Επιπλέον το κόστος αυτού του αλγορίθμου θα είναι O(log n) * n/2 = O(n log n).

Επομένως ο αλγόριθμος παράλληλου αθροίσματος υπολογίζει σωστά το άθροισμα n αριθμών με n/2 επεξεργαστές σε χρόνο O(log n) και κόστος O(n log n).

4.5 Επεξήγηση βέλτιστου αλγορίθμου

Παρατηρούμε όμως πως ο αλγόριθμος αυτός δεν είναι αποδοτικός, το κόστος του δεν είναι της τάξεως του χρόνου του καλύτερου σειριακού αλγορίθμου. Ευτυχώς, αυτό μπορεί να επιλυθεί εύκολα με το να μειώσουμε τους επεξεργαστές μας σε σημείο που το κόστος να είναι ίσο με O(n).

Θεωρούμε λοιπόν ότι έχουμε n/log n επεξεργαστές και όχι n/2, έτσι το κόστος θα μειωθεί σε O(log n) *(n /log n) = O(n). Η μείωση των επεξεργαστών αν και καθιστά των αλγόριθμο μας κόστου-βέλτιστο επιφέρει και κάποιες αλλαγές στα βήματα του αλγορίθμου.



Σχήμα 4.2

Τα δεδομένα μας α1,a2,a3,….,an βρίσκονται και πάλι αποθηκευμένα στις πρώτες n θέσεις τις κοινόχρηστης μνήμης.

Τώρα όμως στο βήμα 0 ο κάθε επεξεργαστής αναλαμβάνει log n στοιχεία. Το άθροισμα των log n αυτών στοιχείων που αντιστοιχούν σε κάθε επεξεργαστή γίνεται με την εκτέλεση του σειριακού αλγορίθμου άθροισης. Ωστόσο, όλοι οι επεξεργαστές δουλεύουν παράλληλα σάυτό το βήμα. Έπειτα το άθροισμα των στοιχείων του κάθε επεξεργαστή αποθηκεύεται στις πρώτες n/log n θέσεις της κοινόχρηστης μνήμης. Ακολούθως στο βήμα 1 του αλγορίθμου οι επεξεργαστές μειώνονται στο μισό και εκτελούν τον αλγόριθμο της παράλληλης άθροισης με n/log n στοιχεία και n/2log n επεξεργαστές. Όπως φαίνεται και στο **Σχήμα 4.2**

Ας αναλύσουμε τώρα τον χρόνο που χρειάζεται ο αλγόριθμος για να τερματίσει και να μας δώσει το άθροισμα των n αριθμών. Καταρχάς για να εκτελεστεί στο βήμα 0 ο σειριακός αλγόριθμος άθροισης σε log n στοιχεία χρειαζόμαστε χρόνο Θ(log n). Έπειτα εκτελούμε τον αλγόριθμο παράλληλης άθροισης με n/log n στοιχεία και άρα ο βρόγχος στο βήμα 3 του αλγορίθμου θα εκτελεστεί O(log(n/log n))= O(log(p)) φορές όπου σε κάθε επανάληψη θα εκτελούνται πράξεις με σταθερό χρόνο. Προφανώς όμως βλέπουμε ότι υπερισχύει ο χρόνος που καταναλώνουμε κατά την σειριακή άθροιση άρα ο συνολικός χρόνος του αλγορίθμου θα είναι O(log n). Παρατηρούμε ότι αν και βέλτιστος αυτός ο αλγόριθμος δεν κατάφερε να μειώσει τον χρόνο εκτέλεσης του αλγορίθμου. Από την άλλη όμως το κόστος τώρα θα μειωθεί σε

O(log n) * n/log n = O(n) που καθιστά τον αλγόριθμο αυτό αποδοτικό.

Επομένως ο βέλτιστος αλγόριθμος παράλληλης άθροισης υπολογίζει σωστά το άθροισμα n αριθμών με n/log n επεξεργαστές σε ασυμπτωτικό χρόνο ίδιο με τον αλγόριθμο παράλληλης άθροισης (O(log n)) και κόστος της τάξεως του χρόνου του καλύτερου σειριακού αλγορίθμου (O(n)).

4.6 Πειραματική αξιολόγηση αλγορίθμων και συμπεράσματα

Άραγε όμως αυτό ισχύει και στην πράξη; Οι δύο αυτοί αλγόριθμοι έχουν τον ίδιο χρόνο εκτέλεσης και διαφορετικό κόστος; Και αν δεν έχουν τον ίδιο χρόνο εκτέλεσης η πρακτική τους διαφορά που προκύπτει είναι κάποια σταθερά; Ο βέλτιστος αλγόριθμος είναι όντως βέλτιστος και στην πράξη; Υλοποιώντας λοιπόν τους δύο πιο πάνω αλγορίθμους στο μοντέλο SB-PRAM με την βοήθεια της γλώσσας FORK παίρνουμε κάποιες μετρήσεις σε χρόνο και κόστος. Οι μετρήσεις που ακολουθούν πάρθηκαν εκτελώντας τους αλγόριθμους από 5 φορές τον κάθε ένα. Σε κάθε εκτέλεση πήραμε μια ξεχωριστή μέτρηση και από αυτές τις πέντε μετρήσεις που αντιστοιχούν στον κάθε αλγόριθμο αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει.

Αλγόριθμος παράλληλης άθροισης

N	*P*	*msec*	*Κόστος*	*log n*
8	**4**	**6.604**	**26.416**	**3**
16	**8**	**8.044**	**64.352**	**4**
32	**16**	**9.476**	**151.616**	**5**
64	**32**	**10.932**	**349.824**	**6**
128	**64**	**12.364**	**791.296**	**7**
256	**128**	**14.300**	**1830.400**	**8**

Πίνακας 4.1

Από τις παραπάνω μετρήσεις παρατηρούμε ότι πράγματι με την αύξηση των στοιχείων στο πρόβλημα μας αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος για να υπολογίσει το αποτέλεσμα και να τερματίσει. Προφανώς και το κόστος είναι ανάλογο

της αύξησης των στοιχείων αφού εξαρτάτε από τον χρόνο διεκπεραίωσης και από τον αριθμό των επεξεργαστών οι οποίοι και αυτοί κατ' ανάγκη θα αυξάνονται γιατί στο πρόβλημα μας απαιτείται να έχουμε αριθμό επεξεργαστών ίσο με το μισό των στοιχείων. Η παρατήρηση αυτή οδηγεί στο συμπέρασμα ότι ο χρόνος είναι ανάλογος του αριθμού των στοιχείων του προβλήματος. Επομένως επιβεβαιώνεται το ότι, στην ασυμπτωτική εξίσωση του χρόνου λάβαμε υπόψη μας το πλήθος (n) των στοιχείων.
Μένει τώρα να δούμε αν όντως ο χρόνος του προβλήματος είναι log n .
Από τον πιο πάνω πίνακα παρατηρούμε ότι ο χρόνος που παίρνουμε στην πράξη είναι περίπου διπλάσιος από τον χρόνο που υπολογίσαμε με την εξίσωση log n. Αυτό όμως συμβαίνει δικαιολογημένα γιατί ο χρόνος του προβλήματος πρέπει να είναι ασυμπτωτικά ίσος με τον θεωρητικό χρόνο. Έτσι λοιπόν στο χρόνο που υπολογίσαμε κατά την θεωρητική αξιολόγηση O(log n) δεν συμπεριλαμβάνει και κάποιες σταθερές που αυτές όμως συμβάλλουν στην αύξηση του χρόνου κατά την υλοποίηση.

Βέλτιστος αλγόριθμος παράλληλης άθροισης

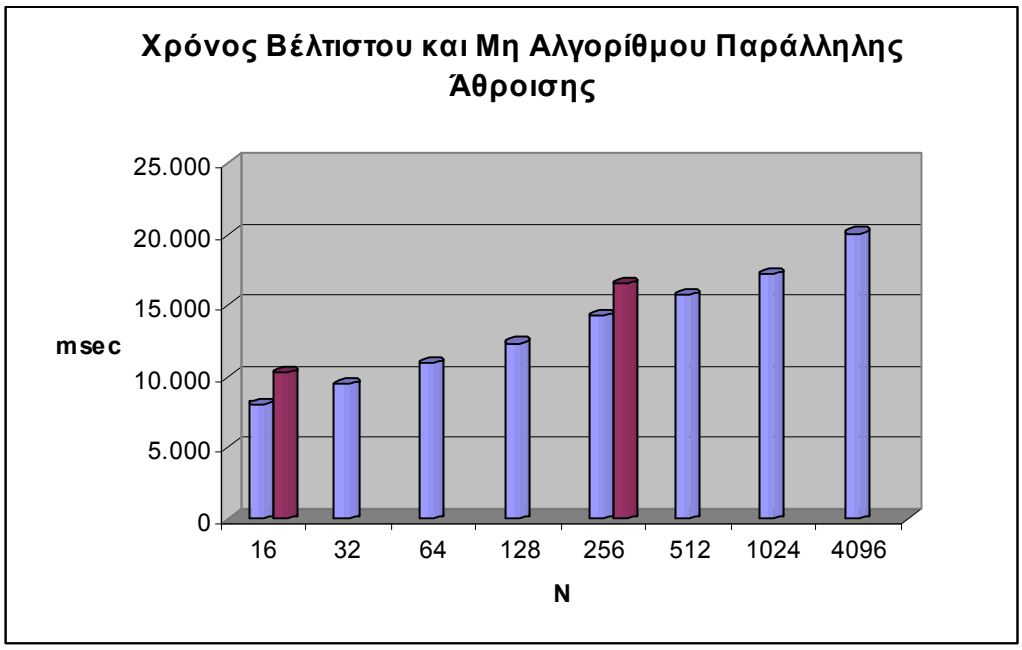
N	*P*	*CPU cycles*	*msec*	*Κόστος*
8	**2**	**2207**	**8.828**	**17.656**
16	**2**	**2695**	**10.780**	**21.560**
16	**4**	**2569**	**10.276**	**41.104**
32	**2**	**3671**	**14.684**	**29.368**
32	**4**	**3057**	**12.228**	**48.912**
256	**32**	**4135**	**16.540**	**529.280**

Πίνακας 4.2

Από τις μετρήσεις που πήραμε στο βέλτιστο αλγόριθμο παρατηρούμε πως πάλι έχουμε μια αύξηση στον χρόνο καθώς τα στοιχεία αυξάνονται. Επιπλέον όμως παρατηρούμε ότι με ίδιο πλήθος στοιχείων αλλά με διαφορετικό πλήθος επεξεργαστών, στα στοιχεία που αθροίζονται με τους λιγότερους επεξεργαστές καταναλώνεται περισσότερος χρόνος παρότι στα στοιχεία που έχουμε περισσότερους επεξεργαστές. Παραδείγματος χάρη στο παραπάνω πίνακα βλέπουμε ότι για n=16 χρησιμοποιώντας 4 επεξεργαστές καταναλώνουμε λιγότερο χρόνο από ότι καταναλώνουμε με 2 επεξεργαστές. Αν και αυτό δεν είναι λογικό να συμβαίνει γιατί ο χρόνος εξαρτάτε μόνο από το πλήθος (n)

των στοιχείων και όχι από το πλήθος των επεξεργαστών , έπεται από το ότι ο βέλτιστος αλγόριθμος εκτελείτε σε δύο βήματα το σειριακό αλλά και το παράλληλο. Έτσι στο σειριακό βήμα οι επεξεργαστές αναλαμβάνουν να αθροίσουν log n στοιχεία και επομένως ο χρόνος είναι O(log n). Στο παράλληλο όμως βήμα εκτελούμε τον αλγόριθμο μη βέλτιστης παράλληλης άθροισης σε O(log (p)) στοιχεία όπου p= n/log n. Επομένως υλοποιώντας τον αλγόριθμο ο χρόνος αυτός υπολογίζεται και άρα δικαιολογημένα ο χρόνος δεν εξαρτάται μόνο από το πλήθος των στοιχείων αλλά εξαρτάται και από το πλήθος των επεξεργαστών.

4.7 Παρουσίαση γραφικών και σύγκριση των δύο αλγορίθμων

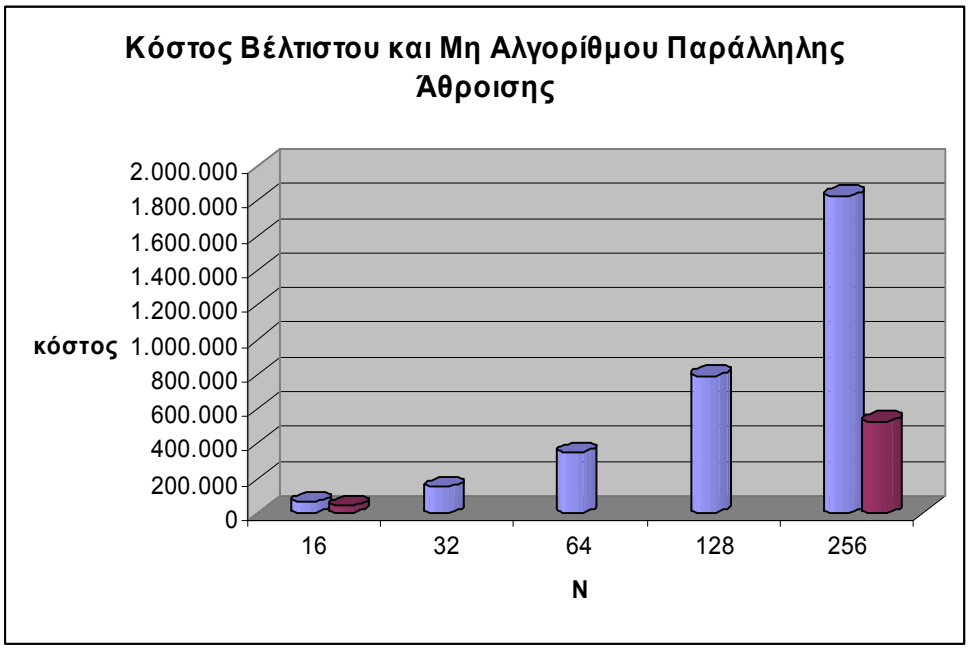


Διεκρύνηση: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με n/log n και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.

Γραφική 4.1

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος παράλληλης άθροισης είναι ελάχιστα γρηγορότερος από τον βέλτιστο. Αυτή η παρατήρηση μας

οδηγεί στο ακόλουθο συμπέρασμα: Ότι δεν ισχύει εκείνο που είδαμε στην θεωρητική αξιολόγηση, δηλαδή το ότι και οι δύο αλγόριθμοι έχουν χρόνο εκτέλεσης O(log n). Αυτό όμως μπορεί να εξηγηθεί εύκολα αφού όπως έχει αναφερθεί και πιο πάνω ο βέλτιστος αλγόριθμος χωρίζεται σε δύο βήματα και έτσι αν και ασυμπτωτικά καταναλώνει τον ίδιο χρόνο με τον μη βέλτιστο, στην πράξη αυτό δεν ισχύει γιατί πρέπει να υπολογίσουμε και τον χρόνο του παράλληλου βήματος τον οποίο δεν το υπολογίζουμε στον ασυμπτωτικό χρόνο κατά την θεωρητική αξιολογηση αφού υπερισχύει ο χρόνος του σειριακού βήματος. *Επομένως δικαιολογημένα στην πράξη ο αλγόριθμος παράλληλης άθροισης είναι γρηγορότερος από τον βέλτιστο αλγόριθμο.*



Διεκρύνηση: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με n/log n και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.

Γραφική 4.2

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος παράλληλης άθροισης έχει πολύ πιο μεγάλο κόστος από τον βέλτιστο αλγόριθμο.

Πράγματι αυτό ισχύει και στην θεωρητική αξιολόγηση του κόστους για τους δύο αυτούς αλγορίθμους. Καταφέραμε λοιπόν να μειώσουμε το κόστος του βέλτιστου αλγορίθμου αν και ο χρόνος του ήταν μεγαλύτερος. Αυτή η μείωση του κόστους λοιπόν οφείλεται στο ότι χρησιμοποιήσαμε λιγότερους επεξεργαστές. Πόση όμως είναι η διαφορά που προκύπτει από τα δύο αυτά κόστη;
Υπολογίζοντας τη διαφορά που προκύπτει από το κόστος του βέλτιστου αλγορίθμου παράλληλης άθροισης από το κόστος του μη βέλτιστου αλγορίθμου καταλήγουμε στον εξής πίνακα:

N	***Κόστος Μη Βέλτιστου***	***Κόστος Βέλτιστου***	***Διαφορά***
16	**64.352**	**41.104**	**23.248**
256	**1830.400**	**529.280**	**1301.12**

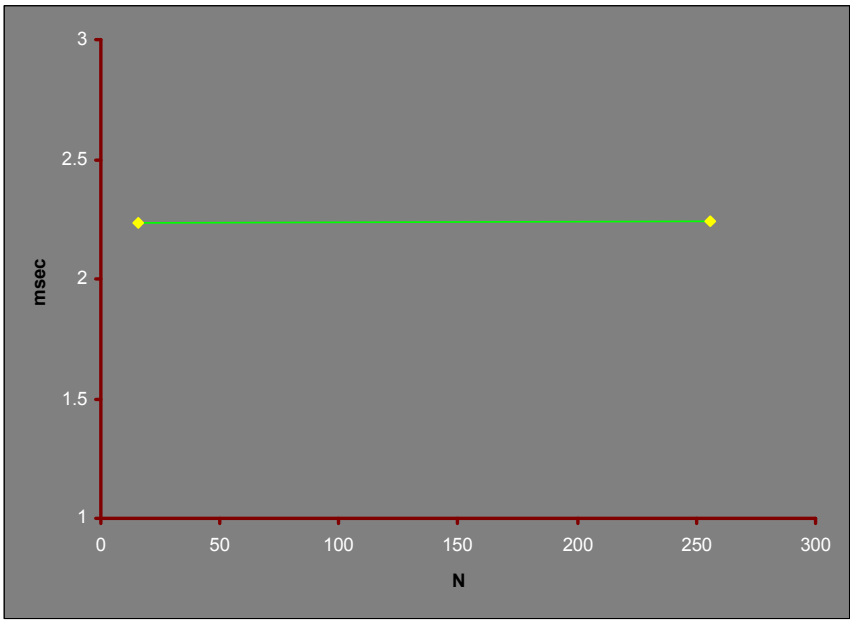
Πίνακας 4.3

Από αυτόν τον πίνακα παρατηρούμε αρχικά ότι έχουμε μεγάλη διαφορά κόστους. Κάτι όμως που δεν έχει αναφερθεί πιο πάνω είναι ότι όσο αυξάνεται το πλήθος των στοιχείων αυτή η διαφορά γίνεται όλο και πιο μεγάλη. Δεδομένου λοιπόν ότι όσο πιο πολλά στοιχεία εισάγουμε στο πρόβλημα μας τόσο πιο ρεαλιστικά σενάρια έχουμε, με την εισαγωγή περισσότερων στοιχείων θα ωφεληθούμε κατά πολύ ως προς το κόστος που θα καταναλώσουμε.

Συνοψίζοντας έχουμε δει ότι όντως ο βέλτιστος αλγόριθμος παράλληλης άθροιση έχει μικρότερο κόστος εκτέλεσης αλλά καταναλώνει πολύ περισσότερο χρόνο. Επομένως σίγουρα το ότι καταφέραμε και μειώσαμε το κόστος είναι πολύ σημαντικό. Τι γίνεται όμως αν καθώς αυξάνονται τα στοιχεία αυξάνεται κατά πολύ ο χρόνος του βέλτιστου και η διαφορά που προκύπτει με τον χρόνο του μη βέλτιστου είναι πολύ μεγάλη; Αυτό θα πρέπει να μας προβληματίσει γιατί καλό είναι να έχουμε μικρό κόστος αλλά και ο χρόνος παίζει ένα ουσιαστικό ρόλο. Αυτό λοιπόν μπορούμε να το δούμε υπολογίζοντας την πρακτική διαφορά που προκύπτει μεταξύ των δύο αλγορίθμων.

N	*Χρόνος Μη Βέλτιστου*	*Χρόνος Βέλτιστου*	*Διαφορά*
16	**8.044**	**10.276**	**2.232**
256	**14.300**	**16.540**	**2.24**

Πίνακας 4.3



Γραφική 4.3

Από αυτή την γραφική λοιπόν παρατηρούμε ότι η διαφορά στο χρόνο του μη βέλτιστου αλγορίθμου από τον βέλτιστο που προκύπτει παραμένει σταθερή όσο αυξάνεται το πλήθος των στοιχείων. Επομένως συμπεραίνουμε ότι αν και ο βέλτιστος αλγόριθμος καταναλώνει περισσότερο χρόνο στην εκτέλεση του από τον μη βέλτιστο η διαφορά του χρόνου είναι σταθερή. Επομένως ο βέλτιστος αλγόριθμος είναι όντως καλύτερος από τον μη βέλτιστο αλγόριθμο γιατί καταφέραμε να μειώσουμε και το κόστος αλλά και ο χρόνος που προκύπτει δεν έχει και πολύ μεγάλη διαφορά.

4.8 Γενικά συμπεράσματα

Συνοψίζοντας λοιπόν, καταλήγουμε στο συμπέρασμα ότι αν και στην θεωρητική αξιολόγηση ο χρόνος του βέλτιστου αλγορίθμου είναι ασυμπτωτικά ίδιος με τον χρόνο του μη βέλτιστου στην πράξη δεν ισχύει αυτό, αλλά ο χρόνος του βέλτιστου

αλγορίθμου είναι μεγαλύτερος. Όπως όμως έχουμε δει η διαφορά που προκύπτει στο χρόνο δεν είναι και πολύ μεγάλη αλλά και με την αύξηση των στοιχείων παραμένει σταθερή. Αντιθέτως το κόστος σε θεωρητική αλλά και σε πρακτική αξιολόγηση είναι μεγαλύτερο στον μη βέλτιστο αλγόριθμο παράλληλης άθροισης. Επομένως όντως ο βέλτιστος αλγόριθμος παράλληλης άθροισης είναι πολύ καλύτερος από τον μη βέλτιστο αλγόριθμο.

Κεφάλαιο 5

Αλγόριθμος παράλληλης μετάδοσης

5.1 Πρόβλημα παράλληλης μετάδοσης

Δεδομένου κάποιου μηνύματος που είναι αποθηκευμένο στην κυψελίδα 1 της κοινόχρηστης μνήμης SM[1] μας ζητείται να αντιγραφεί σε n κυψελίδες μνήμης [3].

5.2 Σειριακή λύση

Εκτελώντας λοιπόν τον σειριακό αλγόριθμο στον οποίο λαμβάνει μέρος ένας επεξεργαστής όπου διαβάζει το στοιχείο που είναι αποθηκευμένο στην κυψελίδα 1 της κοινόχρηστης μνήμης καταναλώνουμε χρόνο Θ(n) για να μεταδώσουμε (γράψουμε) σε όλες τις κυψελίδες αυτό το στοιχείο.

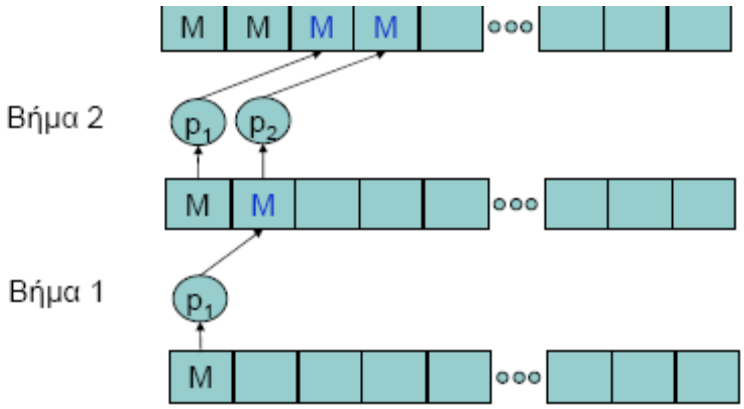
Αλγόριθμος Σειριακής Μετάδοσης

```
1. For j = 2 to n do
2.       S [j]=S[1];
3.  end do
```

Ο αλγόριθμος αυτός όντως μεταδίδει και στις n κυψελίδες της κοινόχρηστης μνήμης το στοιχείο που βρίσκεται στην πρώτη θέση και συνεπώς ο συνολικός χρόνος που χρειάζεται είναι Θ(n) λόγω του βήματος 1 όπου εκτελείται n-2 φορές και στο ότι σε κάθε επανάληψη του κάνει πράξεις που παίρνουν σταθερό χρόνο. Ο χρόνος αυτός είναι αναπόφευκτος γιατί ο επεξεργαστής πρέπει να περάσει και από τις n κυψελίδες της κοινόχρηστης μνήμης για να μεταδώσει το στοιχείο. Το κόστος του αλγορίθμου είναι και αυτό Θ(n) γιατί λαμβάνει μέρος στον υπολογισμό ένας μόνο επεξεργαστής.
Τι συμβαίνει τώρα αν εισάγουμε στο πρόβλημα μας περισσότερους επεξεργαστές οι οποίοι θα εργάζονται παράλληλά;

5.3 Περιγραφή παράλληλου αλγορίθμου

Στα μοντέλα CRCW και CREW PRAM εισάγοντας n επεξεργαστές ο κάθε επεξεργαστής διαβάζει ταυτόχρονα το στοιχείο που είναι αποθηκευμένο στην πρώτη κυψελίδα και το μεταδίδει στην θέση της κυψελίδας που αντιστοιχεί στον προσωπικό του αριθμό , ούτως ώστε το στοιχείο να μεταδοθεί και στις n θέσεις της κοινόχρηστης μνήμης. Ο χρόνος που απαιτείτε είναι σταθερός της τάξης του Θ(1) γιατί οι επεξεργαστές αναλαμβάνουν να εκτελέσουν παράλληλα κάποιες πράξεις που παίρνουν σταθερό χρόνο . Αντίστοιχα και το κόστος αυτού του αλγορίθμου είναι Θ(n) αφού χρησιμοποιούμε n επεξεργαστές οι οποίοι καταναλώνουν σταθερό χρόνο. Συνεπώς ο αλγόριθμος αυτός είναι κόστου βέλτιστος αφού είναι της τάξεως του καλύτερου σειριακού αλγορίθμου.
Η ενδιαφέρουσα περίπτωση όμως προκύπτει όταν προσπαθούμε να λύσουμε το πρόβλημα της μετάδοσης στο μοντέλο EREW PRAM όπου εδώ δεν μπορούμε να έχουμε ταυτόχρονη ανάγνωση.
Οι επεξεργαστές που θα χρησιμοποιήσουμε δεν χρειάζεται να είναι περισσότεροι από n/2 αφού αυτό το πλήθος των επεξεργαστών είναι αρκετό για την επίλυση του προβλήματος. Προκειμένου λοιπόν να μεταδώσουμε το στοιχείο από την πρώτη κυψελίδα της κοινόχρηστης μνήμης και στις n κυψελίδες με n/2 επεξεργαστές εργαζόμαστε ως εξής :



Σχήμα 5.1 [5]

Αρχικά το στοιχείο που πρέπει να μεταδοθεί βρίσκεται αποθηκευμένο στην πρώτη κυψελίδα της κοινόχρηστης μνήμης SM[1]. Κατά την εκτέλεση του βήματος 1 ο επεξεργαστής P1 μεταδίδει το στοιχείο που αντιστοιχεί στην κυψελίδα με τον προσωπικό του αριθμό, όπου στην συγκεκριμένη περίπτωση είναι το στοιχείο που βρίσκεται στην πρώτη θέση, στην διπλανή του θέση δηλαδή στην θέση SM[2] της κοινόχρηστης μνήμης. Έπειτα στο βήμα 2 οι ενεργοί επεξεργαστές διπλασιάζονται διαβάζουν το στοιχείο που αντιστοιχεί

στην θέση με τον προσωπικό τους αριθμό και το μεταδίδουν στις δύο συνεχόμενες θέσεις όπου δεν έχει ακόμη μεταδοθεί. Συνεχίζοντας με αυτόν τον **τρόπο, θα** φτάσουμε στο βήμα log n όπου οι ενεργοί επεξεργαστές τώρα θα είναι ίσοι με n/2 λόγω του ότι τους διπλασιάζουμε συνεχώς σε κάθε βήμα, θα διαβάσουν το στοιχείο που αντιστοιχεί στην θέση με τον προσωπικό τους αριθμό και θα μεταδώσουν αυτό το στοιχείο σε n/2 συνεχόμενες θέσεις στις οποίες το στοιχείο δεν έχει ακόμη μεταδοθεί. Τότε θα έχουμε τελειώσει και το στοιχείο θα βρίσκεται αποθηκευμένο και στις n κυψελίδες της κοινόχρηστης μνήμης. Η θέση στην οποία δεν έχει ακόμη μεταδοθεί το στοιχείο και είναι υπεύθυνος κάθε επεξεργαστής να το μεταδώσει μπορούμε να την βρούμε προσθέτοντας τον προσωπικό αριθμό κάθε επεξεργαστή με το πλήθος των ενεργών επεξεργαστών που έχουμε αυτήν την συγκεκριμένη στιγμή. Για παράδειγμα από το **Σχήμα 5.1** παρατηρούμε πως θέλουμε να μεταδώσουμε το γράμμα M σε 4 κυψελίδες μνήμης με την βοήθεια 2 επεξεργαστών. Αρχικά στο βήμα 1 ο ενεργός επεξεργαστής P1 διαβάζει το στοιχείο που βρίσκεται στην κυψελίδα με τον προσωπικό του αριθμό (δηλαδή στην κυψελίδα SM[1]) και το μεταδίδει όπως έχουμε αναφέρει στην θέση που προκύπτει από το άθροισμα του προσωπικού του αριθμού με το πλήθος των ενεργών

επεξεργαστών που στην συγκεκριμένη περίπτωση είναι ένα. Συνεπώς ο επεξεργαστής P1 θα μεταδώσει το M στην κυψελίδα με αριθμό 2. Ακολούθως οι ενεργοί επεξεργαστές διπλασιάζονται, επομένως τώρα έχουμε δύο επεξεργαστές τον P1 και P2. Παρομοίως ο επεξεργαστής P1 θα μεταδώσει το M που βρίσκεται στην κυψελίδα SM[1], στην κυψελίδα (#ενεργών επεξεργαστών + προσωπικός αριθμός = 2+1=3) SM[3] και ο επεξεργαστής P2 θα μεταδώσει το M που βρίσκεται στην κυψελίδα SM[2], στην κυψελίδα (2+2=4) SM[4]. Τέλος προκύπτει ότι έχουμε μεταδώσει και στις n κυψελίδες μνήμης το στοιχείο M.

Αλγόριθμος Μετάδοσης για EREW PRAM **[5]**

```
1. Processors i = 1 to n/2 do in parallel
2. Set j = 0
3.      while (2^j < n)
4.              if (i ≤ 2^j) then
5.                      SM[i + 2^j] = SM[i]
6.              end if
7.              j=j+1
8.      end while
9.      end do
10. end do
```

5.4 Θεωρητικός χρόνος και κόστος αλγορίθμου

Απομένει τώρα να αναλύσουμε τον χρόνο που χρειάζεται αυτός ο αλγόριθμος για να μεταδώσει το στοιχείο που βρίσκεται στην πρώτη θέση της κοινόχρηστης μνήμης σε n θέσεις και να τερματίσει. Κατ΄ αρχάς ο χρόνος που απαιτείται στο βήμα 3 είναι O(log n) αφού η συνθήκη του βρόγχου έχει κάποιον τελεστή που ξεκινάει από τον αριθμό 1 και διπλασιάζεται συνεχώς άρα ο βρόγχος θα εκτελεστεί log n φορές. Στα βήματα 4, 5, 6 και 7 έχουμε χρόνο O(1) αφού ο κάθε επεξεργαστής διαβάζει, προσθέτει και γράφει το αποτέλεσμα όπου αυτές οι πράξεις παίρνουν σταθερό χρόνο. Άρα έχουμε O(log n) επαναλήψεις και αφού η κάθε μια από αυτές καταναλώνει O(1) χρόνο συνεπώς ο

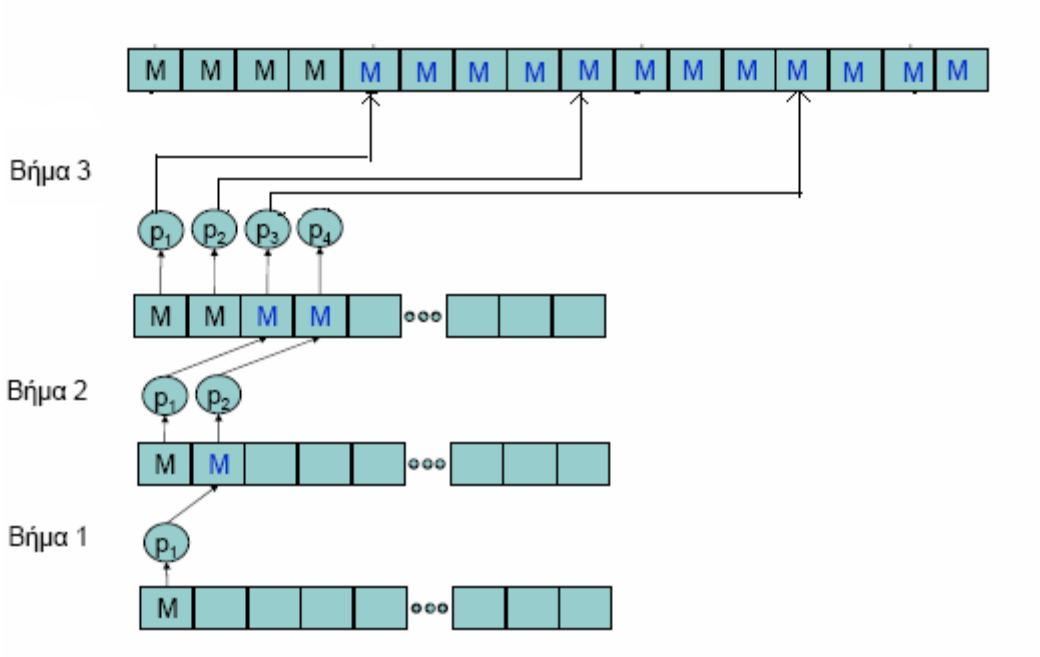
συνολικός χρόνος που απαιτείται είναι O(log n). Επιπλέον το κόστος αυτού του αλγορίθμου θα είναι O(log n) * n/2 = O(n log n).

Επομένως ο αλγόριθμος μετάδοσης για το EREW PRAM αντιγράφει σε n κυψελίδες το στοιχείο που βρίσκεται αποθηκευμένο στην κυψελίδα SM[1] με την βοήθεια n/2 επεξεργαστών σε χρόνο O(log n) και κόστος
O(n log n).

Αυτά που μόλις έχουμε ήδη αναφέρει δεν συμπεριλαμβάνουν και την λύση για το πρόβλημα στο οποίο θέλουν n επεξεργαστές να διαβάσουν το ίδιο στοιχείο. Ευτυχώς, όμως αυτό δεν απαιτεί πολλές αλλαγές αρκεί απλώς αντί για n/2 να εισάγουμε στο πρόβλημα μας n επεξεργαστές τους οποίους δεν χρειάζεται να τους χρησιμοποιήσουμε παρά μόνο όταν ολοκληρωθεί η αντιγραφή και στις n κυψελίδες όπου τότε ο κάθε επεξεργαστής θα διαβάσει το στοιχείο που βρίσκεται στην κυψελίδα με τον προσωπικό του αριθμό. Είναι προφανές ότι αυτό το βήμα μπορεί να εκτελεστεί σε σταθερό χρόνο επομένως δεν επιφέρει καμία αύξηση του ασυμπτωτικού μας χρόνου.

5.5 Επεξήγηση βέλτιστου αλγορίθμου

Παρατηρούμε όμως πως ο αλγόριθμος αυτός δεν είναι αποδοτικός, το κόστος του δεν είναι της τάξης του χρόνου του καλύτερου σειριακού αλγορίθμου. Ευτυχώς, αυτό μπορεί να επιλυθεί εύκολα με το να μειώσουμε τους επεξεργαστές μας σε σημείο που το κόστος να είναι ίσο με O(n).
Θεωρούμε λοιπόν ότι έχουμε n/log n επεξεργαστές και όχι n/2, έτσι το κόστος θα μειωθεί σε O(log n) *(n /log n) = O(n). Η μείωση των επεξεργαστών αν και καθιστά των αλγόριθμο μας κόστου-βέλτιστο επιφέρει και κάποιες αλλαγές στα βήματα του αλγορίθμου.



Σχήμα 5.2 [5]

Το στοιχείο που θέλουμε να μεταδώσουμε βρίσκεται και πάλι αποθηκευμένο στην πρώτη θέση της κοινόχρηστης μνήμης. Ακολουθούμε την ίδια διαδικασία όπως και πριν με τον μη βέλτιστο αλγόριθμο της παράλληλης μετάδοσης μέχρις ώτου οι επεξεργαστές οι οποίοι σε κάθε βήμα διπλασιάζονται συνεχώς να φτάσουν στο μισό των συνολικών επεξεργαστών. Έπειτα κάθε επεξεργαστής αναλαμβάνει να εκτελέσει τον αλγόριθμο της σειριακής μετάδοσης σε log n θέσεις δηλαδή να μεταδώσει σε log n συνεχόμενες κυψελίδες της κοινόχρηστης μνήμης το στοιχείο που βρίσκεται στην θέση με τον προσωπικό του αριθμό. Στην συγκεκριμένη περίπτωση δεν χρειάζεται ο επεξεργαστής με προσωπικό αριθμό ίσο με n/log n να μεταδώσει το στοιχείο που βρίσκεται στην κυψελίδα του, λόγω του ότι από τα προηγούμενα βήματα έχουμε ήδη μεταδώσει σε κάποιες κυψελίδες το στοιχείο και έτσι με το να χρησιμοποιήσουμε αυτόν τον επεξεργαστή θα κάνουμε κάποιες αχρείαστες αντιγραφές του στοιχείου μας σε περισσότερες από n θέσεις τις κοινόχρηστης μνήμης. Ένα άλλο σενάριο που αποφεύγει αυτή την ειδική περίπτωση είναι αντί κάθε επεξεργαστής να μεταδώσει το στοιχείο σε log n θέσεις να το μεταδώσει σε log n-1 θέσεις έτσι ώστε όλοι οι επεξεργαστές να λάβουν μέρος σ'αυτό το τελευταίο βήμα. Είναι προφανές όμως ότι τα δύο αυτά σενάρια είναι σχεδόν τα ίδια και έτσι η πρακτική τους υλοποίηση δεν πρόκειται να διαφέρει και πολύ ως προς τον χρόνο εκτέλεσης. Όπως βλέπουμε και από το **Σχήμα 5.2** θέλουμε να μεταδοθεί και πάλι το γράμμα Μ από 4 επεξεργαστές σε 16 κυψελίδες της κοινόχρηστης μνήμης. Μέχρι και το προτελευταίο βήμα (το βήμα log n) ακολουθούμε

τον αλγόριθμο παράλληλης μετάδοσης στον οποίο λαμβάνουν μέρος οι μισοί επεξεργαστές στην συγκεκριμένη περίπτωση οι 2 από τους 4 επεξεργαστές ενώ οι υπόλοιποι βρίσκονται σε κατάσταση no op. Έπειτα κατά την εκτέλεση του log n +1 βήματος (βήμα 3) ο κάθε επεξεργαστής εκτός από τον επεξεργαστή με προσωπικό αριθμό 4 διαβάζει το Μ που είναι αποθηκευμένο στην κυψελίδα με τον προσωπικό του αριθμό και το μεταδίδει σειριακά σε 4 συνεχόμενες κυψελίδες. Τη θέση στην οποία κάθε επεξεργαστής αναλαμβάνει να μεταδώσει το στοιχείο, μπορούμε να την βρούμε πολλαπλασιάζοντας τον προσωπικό αριθμό κάθε επεξεργαστή με το πλήθος των ενεργών επεξεργαστών που έχουμε την συγκεκριμένη αυτή στιγμή και προσθέτοντας στο αποτέλεσμα του πολλαπλασιασμού τον αριθμό 1. Έτσι κάθε επεξεργαστής ξεκινώντας από την κυψελίδα με τον αριθμό που έχουμε υπολογίσει αντιγράφει το στοιχείο στις επόμενες log n -1 συνεχόμενες θέσεις. Συνεχίζοντας με το παράδειγμα του σχήματος βλέπουμε ότι στο βήμα 3 ο επεξεργαστής με προσωπικό αριθμό P1 διαβάζει το στοιχείο που βρίσκεται στην κυψελίδα με τον προσωπικό του αριθμό και το μεταδίδει σε 4 κυψελίδες ξεκινώντας από την θέση (προσωπικός αριθμός επεξεργαστή * #αριθμός ενεργών επεξεργαστών +1 = 1*4+1= 5) SM[5] και φτάνοντας μέχρι και την θέση SM[8]. Παράλληλα οι επεξεργαστές P2 και P3 διαβάζουν το στοιχείο Μ που βρίσκεται στην κυψελίδα με τον προσωπικό τους αριθμό και το αντιγράφουν στις θέσεις SM[9]-SM[12] και SM[13]-SM[16] αντίστοιχα. Τέλος προκύπτει ότι έχουμε μεταδώσει και στις n κυψελίδες μνήμης το στοιχείο Μ με την βοήθεια n/log n επεξεργαστών.

Ας αναλύσουμε τώρα τον χρόνο που χρειάζεται ο αλγόριθμος για να τερματίσει αφότου έχει αντιγράψει και στις n κυψελίδες το στοιχείο. Για να αποδείξουμε το χρονικό φράγμα του αλγορίθμου, ξεκινάμε παρατηρώντας το πρώτο μέρος όπου οι επεξεργαστές εκτελούν τον αλγόριθμο παράλληλης μετάδοσης σε n/log n στοιχεία. Επομένως ο χρόνος που χρειαζόμαστε σε αυτό το βήμα είναι
log(n/log n) και αυτό προκύπτει από τον αλγόριθμο παράλληλης μετάδοσης και στο ότι τα στοιχεία που έχουμε μέχρι στιγμής μεταδώσει είναι n/log n. Στην συνέχεια εξετάζουμε το σειριακό κομμάτι του αλγορίθμου. Συνεπώς αφού τρέχουμε τον σειριακό αλγόριθμό μετάδοσης σε log n στοιχεία καταναλώνουμε χρόνο O(log n). Εξαιτίας λοιπόν του βήματος όπου εκτελείται ο σειριακός αλγόριθμος ο συνολικός χρόνος που απαιτείται για τον αλγόριθμο βέλτιστης παράλληλης μετάδοσης είναι O(log n).

Παρατηρούμε ότι αν και βέλτιστος αυτός ο αλγόριθμος δεν κατάφερε να μειώσει τον χρόνο εκτέλεσης του αλγορίθμου. Από την άλλη όμως το κόστος τώρα θα μειωθεί σε O(log n) * n/log n = O(n) που καθιστά τον αλγόριθμο αυτό αποδοτικό.

Επομένως ο βέλτιστος αλγόριθμος παράλληλης μετάδοσης είναι ορθός αφού αντιγράφει και στις n κυψελίδες της κοινόχρηστης μνήμης ένα οποιοδήποτε στοιχείο που βρίσκεται στη πρώτη κυψελίδα με την βοήθεια n/log n επεξεργαστών σε χρόνο ασυμπτωτικά ίδιο με τον μη βέλτιστο αλγόριθμο παράλληλης μετάδοσης (O(log n)) και κόστος της τάξεως του χρόνου του καλύτερου σειριακού αλγορίθμου (O(n)).

5.6 Πειραματική αξιολόγηση αλγορίθμων και συμπεράσματα

Άραγε όμως αυτό ισχύει και στην πράξη; Οι δύο αυτοί αλγόριθμοι έχουν τον ίδιο χρόνο εκτέλεσης και διαφορετικό κόστος; Και αν δεν έχουν τον ίδιο χρόνο εκτέλεσης η πρακτική τους διαφορά που προκύπτει είναι κάποια σταθερά; Ο βέλτιστος αλγόριθμος είναι όντως βέλτιστος και στην πράξη; Υλοποιώντας λοιπόν τους δύο πιο πάνω αλγορίθμους στο μοντέλο SB-PRAM με την βοήθεια της γλώσσας FORK παίρνουμε κάποιες μετρήσεις σε χρόνο και κόστος. Οι μετρήσεις που ακολουθούν πάρθηκαν εκτελώντας τους αλγόριθμους από 5 φορές τον κάθε ένα. Σε κάθε εκτέλεση πήραμε μια ξεχωριστή μέτρηση και από αυτές τις πέντε μετρήσεις που αντιστοιχούν στον κάθε αλγόριθμο αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει.

Αλγόριθμος παράλληλης μετάδοσης

N	*P*	*msec*	*Κόστος*
8	**4**	**5.172**	**20.688**
16	**8**	**8.884**	**71.072**
32	**16**	**16.308**	**260.928**
64	**32**	**31.156**	**996.992**
128	**64**	**60.852**	**3894.528**
256	**128**	**120.244**	**15391.232**

Πίνακας 5.1

Από τις παραπάνω μετρήσεις παρατηρούμε ότι πράγματι με την αύξηση των στοιχείων στο πρόβλημα μας αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος για να υπολογίσει το αποτέλεσμα και να τερματίσει. Προφανώς και το κόστος είναι ανάλογο της αύξησης των στοιχείων αφού εξαρτάτε από τον χρόνο διεκπεραίωσης και από τον αριθμό των επεξεργαστών οι οποίοι και αυτοί κατ' ανάγκη θα αυξάνονται γιατί στο πρόβλημα μας απαιτείται να έχουμε αριθμό επεξεργαστών ίσο με το μισό των στοιχείων. Η παρατήρηση αυτή οδηγεί στο συμπέρασμα ότι ο χρόνος είναι ανάλογος του αριθμού των στοιχείων του προβλήματος. Επομένως επιβεβαιώνεται το, ότι στην ασυμπτωτική εξίσωση του χρόνου λάβαμε υπόψη μας το πλήθος (n) των στοιχείων.

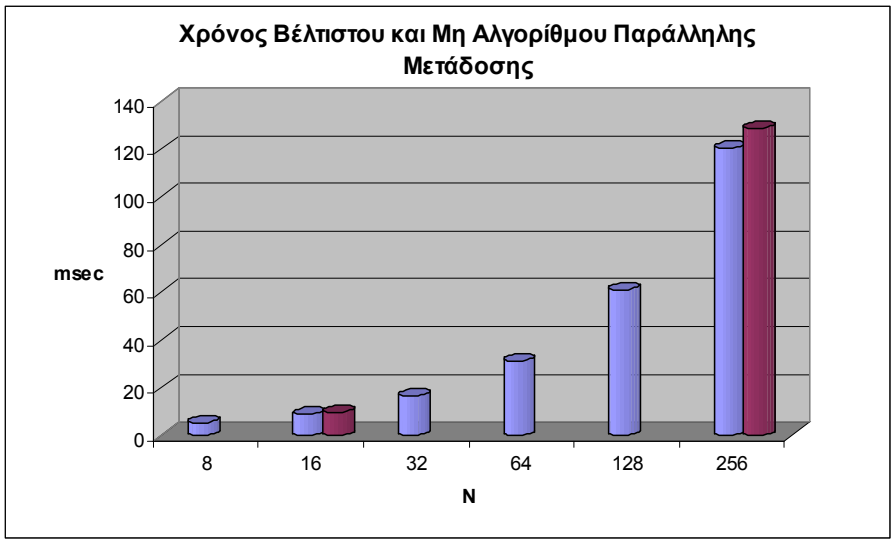
Βέλτιστος αλγόριθμος παράλληλης μετάδοσης

N	*P*	*msec*	*Κόστος*
8	**2**	**2.956**	**5.912**
16	**4**	**9.652**	**38.608**
32	**8**	**13.196**	**105.568**
64	**8**	**29.580**	**236.640**
128	**16**	**58.252**	**932.032**
128	**32**	**50.060**	**1601.920**
256	**32**	**128.692**	**4118.144**
1024	**8**	**521.100**	**4168.800**
1024	**16**	**517.004**	**8272.064**

Πίνακας 5.2

Από τις μετρήσεις που πήραμε στο βέλτιστο αλγόριθμο παρατηρούμε πως πάλι έχουμε μια αύξηση στον χρόνο καθώς τα στοιχεία αυξάνονται. Επιπλέον όμως παρατηρούμε ότι με ίδιο πλήθος στοιχείων αλλά με διαφορετικό πλήθος επεξεργαστών, στην περίπτωση όπου χρησιμοποιούμε λιγότερους επεξεργαστές καταναλώνεται περισσότερος χρόνος παρότι στην περίπτωση όπου έχουμε περισσότερους επεξεργαστές. Παραδείγματος χάριν στο παραπάνω πίνακα βλέπουμε ότι για n=128 χρησιμοποιώντας 32 επεξεργαστές καταναλώνουμε λιγότερο χρόνο απ' ότι καταναλώνουμε με 16 επεξεργαστές. Αν και αυτό δεν είναι λογικό να συμβαίνει γιατί ο ασυμπτωτικός χρόνος εξαρτάτε μόνο από το πλήθος (n) των στοιχείων και όχι από το πλήθος των επεξεργαστών , έπεται από το ότι ο βέλτιστος αλγόριθμος εκτελείται σε δύο βήματα το σειριακό αλλά και το παράλληλο. Έτσι στο σειριακό βήμα οι επεξεργαστές αναλαμβάνουν να αντιγράψουν σε log n συνεχόμενες κυψελίδες της κοινόχρηστης μνήμης το στοιχείο που διαβάζουν και επομένως ο χρόνος είναι O(log n). Στο παράλληλο όμως βήμα εκτελούμε τον αλγόριθμο μη βέλτιστης παράλληλης μετάδοσης μέχρι να έχει αντιγραφεί σε n/log p (p=n/log n) κυψελίδες της κοινόχρηστης μνήμης το στοιχείο. Επομένως κατά την υλοποίηση του αλγορίθμου το κάθε βήμα μετρά εις βάρος του χρόνου και άρα δικαιολογημένα ο χρόνος δεν εξαρτάται μόνο από το πλήθος των στοιχείων αλλά εξαρτάται και από το πλήθος των επεξεργαστών.

5.7 Παρουσίαση γραφικών και σύγκριση των δύο αλγορίθμων

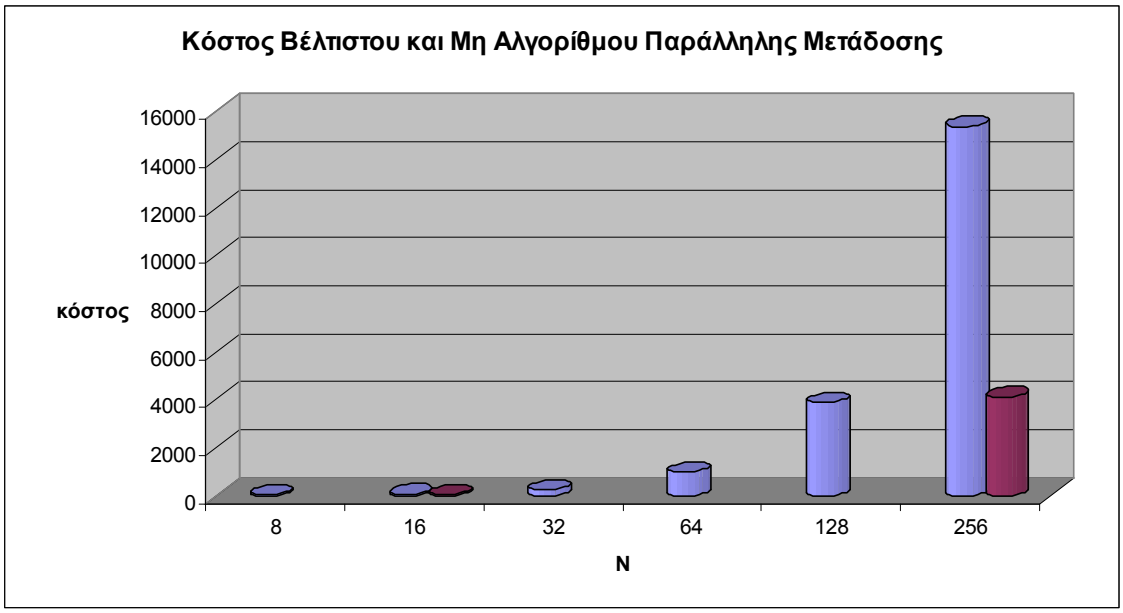


- **Αλγόριθμος παράλληλης μετάδοσης.**
- **Βέλτιστος αλγόριθμος παράλληλης μετάδοσης.**

Διεκρύνηση: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με n/log n και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.

Γραφική 5.1

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος παράλληλης μετάδοσης είναι γρηγορότερος από τον βέλτιστο. Αυτή η παρατήρηση μας οδηγεί στο ακόλουθο συμπέρασμα: Εκείνο που είδαμε στην θεωρητική αξιολόγηση, δηλαδή το ότι και οι δύο αλγόριθμοι έχουν τον ίδιο ασυμπτωτικό χρόνο εκτέλεσης (O(log n)), δεν ισχύει εδώ διότι γνωρίζουμε πως ο βέλτιστος αλγόριθμος χωρίζεται σε δύο βήματα και έτσι αν και ασυμπτωτικά καταναλώνει τον ίδιο χρόνο με τον μη βέλτιστο, στην πράξη αυτό δεν ισχύει γιατί εδώ χρειάζεται να υπολογίσουμε και τον χρόνο του παράλληλου βήματος τον οποίο δεν τον υπολογίζουμε στον ασυμπτωτικό χρόνο αφού υπερισχύει ο χρόνος του σειριακού βήματος. *Επομένως δικαιολογημένα στην πράξη ο αλγόριθμος παράλληλης άθροισης είναι γρηγορότερος από τον βέλτιστο αλγόριθμο.*



- **Αλγόριθμος παράλληλης μετάδοσης.**
- **Βέλτιστος αλγόριθμος παράλληλης μετάδοσης.**

Διεκρύνηση: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με n/log n και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.

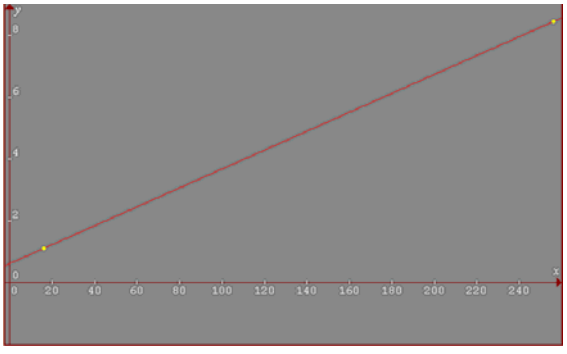
Γραφική 5.2

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος παράλληλης μετάδοσης έχει πολύ πιο μεγάλο κόστος από τον βέλτιστο αλγόριθμο.
Πράγματι αυτό ισχύει και στην θεωρητική αξιολόγηση του κόστους για τους δύο αυτούς αλγορίθμους. Καταφέραμε λοιπόν να μειώσουμε το κόστος του βέλτιστου αλγορίθμου αν και ο χρόνος του ήταν μεγαλύτερος. Αυτή η μείωση του κόστους λοιπόν οφείλεται στο ότι χρησιμοποιήσαμε λιγότερους επεξεργαστές.

Συνοψίζοντας έχουμε δει ότι όντως ο βέλτιστος αλγόριθμος παράλληλης άθροιση έχει μικρότερο κόστος εκτέλεσης αλλά καταναλώνει πολύ περισσότερο χρόνο. Επομένως σίγουρα το ότι καταφέραμε και μειώσαμε το κόστος είναι πολύ σημαντικό. Τι γίνεται όμως αν καθώς αυξάνονται τα στοιχεία αυξάνεται κατά πολύ ο χρόνος του βέλτιστου και η διαφορά που προκύπτει με τον χρόνο του μη βέλτιστου είναι πολύ μεγάλη; Αυτό θα πρέπει να μας προβληματίσει γιατί καλό είναι να έχουμε μικρό κόστος αλλά και ο χρόνος παίζει ένα ουσιαστικό ρόλο. Αυτό λοιπόν μπορούμε να το δούμε υπολογίζοντας την πρακτική διαφορά που προκύπτει μεταξύ των δύο αλγορίθμων.

N	*Χρόνος Μη Βέλτιστου*	*Χρόνος Βέλτιστου*	*Διαφορά*
16	**8.884**	**9.992**	**1.108**
256	**120.244**	**128.692**	**8.448**

Πίνακας 5.3



Γραφική 4.3

Από αυτή την γραφική λοιπόν παρατηρούμε ότι η διαφορά στο χρόνο του μη βέλτιστου αλγορίθμου από τον βέλτιστο που προκύπτει είναι γραμμική όσο αυξάνεται το πλήθος των στοιχείων. Τώρα λοιπόν καλό θα ήταν να υπολογίσουμε την κλίση αυτής της γραφικής για να δούμε κατά πόσο θα αυξάνεται ο χρόνος με την αύξηση των στοιχείων έτσι ώστε να δούμε αν ο βέλτιστος αλγόριθμος θα έχει μεγάλη αύξηση του χρόνου με την αύξηση του πλήθους των στοιχείων. Επομένως υπολογίζοντας την κλίση αυτού του ευθύγραμμου τμήματος προκύπτει ο αριθμός 0.0306. Όντως λοιπόν αυτός ο αριθμός είναι αρκετά μικρός . Καθώς λοιπόν αυξάνουμε τα στοιχεία θα έχουμε μια αύξηση στον χρόνο του βέλτιστου αλλά αυτή θα είναι πολύ μικρή. Καταλαβαίνουμε λοιπόν ότι ο βέλτιστος αλγόριθμος είναι αρκετά αποδοτικός αφού κατάφερε να μειώσει σημαντικά το κόστος του αλγορίθμου αλλά και ο χρόνος αν και είναι μεγαλύτερος από τον μη βέλτιστο, με την αύξηση των στοιχείων ο χρόνος αυξάνεται γραμμικά έχοντας μια πάρα πολύ μικρή κλίση.

4.9 Γενικά συμπεράσματα

Συνοψίζοντας λοιπόν, καταλήγουμε στο συμπέρασμα ότι αν και στην θεωρητική αξιολόγηση ο χρόνος του βέλτιστου αλγορίθμου είναι ασυμπτωτικά ίδιος με τον χρόνο του μη βέλτιστου στην πράξη δεν ισχύει αυτό, αλλά ο χρόνος του βέλτιστου αλγορίθμου είναι μεγαλύτερος. Όπως όμως έχουμε δει η διαφορά που προκύπτει στο χρόνο δεν είναι και πολύ μεγάλη αλλά και με την αύξηση των στοιχείων αυξάνεται γραμμικά με κλίση πολύ κοντά στο μηδέν . Αντιθέτως το κόστος σε θεωρητική αλλά και σε πρακτική αξιολόγηση είναι μεγαλύτερο στον μη βέλτιστο αλγόριθμο παράλληλης άθροισης. Επομένως όντως ο βέλτιστος αλγόριθμος παράλληλης μετάδοσης είναι πολύ καλύτερος από τον μη βέλτιστο αλγόριθμο.

Κεφάλαιο 6

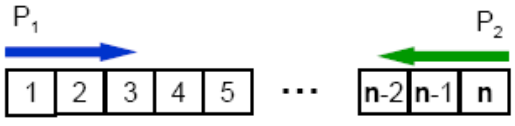
Αλγόριθμος Write-ALL στο A-PRAM με δύο επεξεργαστές

6.1 Πρόβλημα Write-ALL στο A-PRAM με δύο επεξεργαστές

Δεδομένου μιας λίστας n στοιχείων, όπου αρχικά τα στοιχεία έχουν την τιμή 0, ζητείται να μετατραπούν οι τιμές των στοιχείων σε 1 χρησιμοποιώντας 2 επεξεργαστές στο μοντέλο A-PRAM [3].

6.2 Περιγραφή παράλληλου αλγορίθμου

Αν και το πρόβλημα αυτό δεν φαίνεται και πολύ δύσκολο η πολυπλοκότητα του κρύβεται στο ότι το μοντέλο που θα χρησιμοποιήσουμε είναι το A-PRAM που όπως έχει αναφερθεί και παραπάνω η ιδιαιτερότητα αυτού του μοντέλου βρίσκεται στο ότι οι επεξεργαστές (στη συγκεκριμένη περίπτωση ο P1 και P2) λειτουργούν εντελώς ασυγχρόνιστα.



Σχήμα 6.1 [5]

Αρχικά ο κάθε επεξεργαστής αναλαμβάνει μια κυψελίδα της κοινόχρηστης μνήμης από την οποία και θα ξεκινήσει. Ο P1 ενδέχεται να αναλάβει την πρώτη κυψελίδα και ο P2

την τελευταία κυψελίδα δηλαδή την SM[1] και SM[n], αντίστοιχα. Ακολούθως ο κάθε επεξεργαστής διαβάζει την τιμή του στοιχείου που βρίσκεται στην κυψελίδα που έχει αναλάβει. Αν η τιμή είναι 0 την μετατρέπει σε 1 και προχωράει εκτελώντας την ίδια διαδικασία στις επόμενες γειτονικές του κυψελίδες. Ο P1 προχωράει προς τα δεξιά ενώ ο P2 προς τα αριστερά. Στην περίπτωση όμως που κάποιος επεξεργαστής συναντήσει μια κυψελίδα που έχει την τιμή 1 ή ο επεξεργαστής έχει φτάσει στο τέλος της λίστας, τότε τερματίζει επιστρέφοντας την θέση στην οποία έχει φτάσει. Ανεξάρτητα με το αν οι δύο επεξεργαστές ξεκινούν την εκτέλεση του αλγορίθμου ταυτόχρονα υπάρχει περίπτωση να τερματίσουν σε δύο πολύ διαφορετικές χρονικές στιγμές αφού μπορεί κάποιος από τους δύο να είναι πιο γρήγορος από τον άλλο και έτσι θα γράψει σε περισσότερες κυψελίδες την τιμή 1. Επομένως, προκύπτει ότι με την βοήθεια των επεξεργαστών P1 και P2 ο αλγόριθμος μας τερματίζει αφότου έχουμε γράψει και στις n κυψελίδες της κοινόχρηστης μνήμης την τιμή 1.

Αλγόριθμος Write-All στό A-PRAM με p=2 **[5]**

```
Processors P1, P2 do asynchronously
P1:     for j=1 to n
        if   SM[j]=0   then
                SM[j]=1
        else
                terminate
        end for
P2:     for j=n to 1
        if   SM[j]=0   then
                SM[j]=1
        else
                terminate
        end for
end do
```

6.3 Θεωρητικό κόστος αλγορίθμου

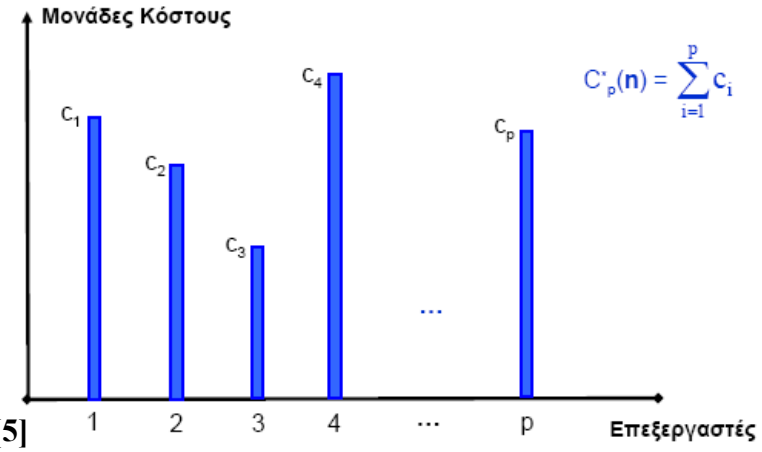
Για αυτόν τον αλγόριθμο δεν θα κάνουμε ανάλυση του χρόνου του γιατί σε αυτό το μοντέλο, το μοντέλο A-PRAM, ο χρόνος δεν μπορεί να υπολογιστεί θεωρητικά λόγω του ότι οι επεξεργαστές μας λειτουργούν εντελώς ασυγχρόνιστα, αυτό που παίζει λοιπόν τον πρωτεύον ρόλο είναι το κόστος του κάθε επεξεργαστή. Αντιθέτως πειραματικά ο χρόνος μπορεί να υπολογιστεί αφού έχουμε μια εκτέλεση η οποία κάποια στιγμή θα τερματίσει.

Προκειμένου λοιπόν να υπολογίσουμε το κόστος αυτού του αλγορίθμου θα ακολουθήσουμε μια άλλη τακτική και όχι αυτή που χρησιμοποιούσαμε μέχρι στιγμής, στην οποία πολλαπλασιάζαμε τον χρόνο του αλγορίθμου με το πλήθος των συνολικών επεξεργαστών. Η μέθοδος αυτή λειτουργάει ως εξής:

Κάθε διάβασε-υπολόγισε-γράψε πράξη, όπου κάθε επεξεργαστής διαβάζει μια κυψελίδα, κάνει κάποιους «μικρούς» τοπικούς υπολογισμούς και γράφει σε μια κυψελίδα, χρεώνεται με μια μονάδα κόστους. Αφότου έχουμε υπολογίσει για τον κάθε επεξεργαστή τις μονάδες κόστους που του αντιστοιχούν, ερχόμαστε και τις αθροίζουμε μεταξύ τους βγάζοντας το συνολικό κόστος του αλγορίθμου.

Για παράδειγμα αν c_i είναι ο συνολικός αριθμός κόστους που χρεώνεται ο επεξεργαστής i μέχρι να λυθεί το πρόβλημα Write-All μεγέθους n τότε το συνολικό κόστος θα είναι :

$$C^*_p(n) = \sum_{i=1}^{p} c_i$$



Σχήμα 6.2 [5]

Εξετάζοντας λοιπόν προσεκτικά τον παραπάνω αλγόριθμο παρατηρούμε ότι κάθε επεξεργαστής χρεώνεται τόσες μονάδες κόστους όσες και οι κυψελίδες μνήμης που διαβάζει. Προς τούτο, στη χειρότερη περίπτωση ο P1 θα καταναλώσει κόστος n/2+1 λόγω του ότι θα διαβάσει μέχρι το μεσαίο συν ένα στοιχείο της κοινόχρηστης μνήμης . Παρομοίως και ο P2 θα καταναλώσει το ίδιο κόστος. Επομένως το συνολικό κόστος και των δύο επεξεργαστών μαζί είναι:

C1=n/2+1

C2= n/2+1

$$\sum_{i=1}^{2} C_i = \mathbf{n+2 = O(n)}$$

Στην καλύτερη όμως περίπτωση ο ένας από τους δύο επεξεργαστές θα είναι πολύ γρηγορότερος και θα διαβάσει όλες τις κυψελίδες μνήμης, ενώ ο άλλος θα καταφέρει να διαβάσει μόνο μια κυψελίδα μνήμης. Επομένως ο επεξεργαστής P1 θα καταναλώσει κόστος n ενώ ο επεξεργαστής P2 θα καταναλώσει κόστος 1 ή και αντίστροφα . Συνεπώς το συνολικό κόστος που προκύπτει θα είναι ίσο με:

C1=n

C2=1

$$\sum_{i=1}^{2} C_i = \mathbf{n+1 = O(n)}$$

Άρα προκύπτει ότι το συνολικό κόστος θα είναι της τάξεως του Ω(n). Βεβαίως υπάρχουν και άλλες ενδιάμεσες περιπτώσεις αλλά αναλύοντας αυτές τις ακραίες γνωρίζουμε ότι ο αλγόριθμος μας δεν πρόκειται να υπερβεί αυτό το φράγμα κόστους. Ίσως όμως να μπορούμε να επιτύχουμε κάτι ακόμη καλύτερο; Δυστυχώς, αυτό το φράγμα είναι το καλύτερο που μπορούμε να κάνουμε γιατί για να λυθεί το πρόβλημα απαραίτητα πρέπει να διαβαστούν όλες οι κυψελίδες μνήμης. Δεν μπορούμε λοιπόν να ελπίσουμε σε κάποια στρατηγική που να εξασφαλίζει λιγότερο κόστος. Αυτό όμως δεν μας ανησυχεί γιατί ήδη ο αλγόριθμος μας είναι κόστου-βέλτιστος.

Επομένως ο αλγόριθμος Write- All επιλύεται στο μοντέλο A-PRAM με την βοήθεια 2 επεξεργαστών σε κόστος O(n).

6.4 Πειραματική αξιολόγηση αλγορίθμου και συμπεράσματα

Άραγε όμως το φράγμα κόστους επιβεβαιώνεται και στην πράξη; Υλοποιώντας τον πιο πάνω αλγόριθμο στο μοντέλο SB-PRAM, αλλά τώρα με τους δύο επεξεργαστές να εργάζονται εντελώς ασυγχρόνιστα και με την βοήθεια της γλώσσας FORK παίρνουμε τις παρακάτω μετρήσεις όσο αφορά το κόστος του αλγορίθμου. Όπως έχει αναφερθεί και παραπάνω, πειραματικά μπορούμε να υπολογίσουμε τον χρόνο γιατί τώρα μπορούμε να μιλάμε για κάποιο συγκεκριμένο σενάριο όπου ένας επεξεργαστής από τους δύο μπορεί να είναι πιο αργός. Επομένως πάρθηκαν και μετρήσεις για τον χρόνο που καταναλώνει ο αλγόριθμος μας οι οποίες επίσης παραθέτονται παρακάτω. Στον συγκεκριμένο αλγόριθμο και ειδικότερα στον κυρίως κώδικα του αλγορίθμου δεν χρησιμοποιήθηκε καθόλου η εντολή ***barrier*** και το κομμάτι του κώδικα βρισκόταν μέσα στη συνάρτηση ***async();*** χωρίς καμία εντολή που να συγχρονίζει τους επεξεργαστές. Με αυτό τον τρόπο οι επεξεργαστές μας λειτουργούσαν εντελώς ασυγχρόνιστα.

Οι μετρήσεις που ακολουθούν πάρθηκαν εκτελώντας τον αλγόριθμο 5 φορές. Σε κάθε εκτέλεση πήραμε μια ξεχωριστή μέτρηση και από αυτές τις πέντε μετρήσεις αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει.

N	*msec*	*Κόστος*
4	**1.296**	**6**
8	**2.192**	**10**
16	**3.984**	**18**
32	**7.552**	**34**
256	**57.064**	**258**
1024	**229.488**	**1027**
2048	**458.56**	**2050**
8192	**1833.072**	**8195**

Πίνακας 6.1

Από τις παραπάνω μετρήσεις παρατηρούμε ότι καθώς αυξάνονται τα στοιχεία αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος μας για να τερματίσει σωστά.

Παρομοίως αυξάνεται και το κόστος που καταναλώνεται για την ορθή ολοκλήρωση του αλγορίθμου. Συνεπώς έπεται ότι ο χρόνος αλλά και το κόστος είναι ανάλογα του αριθμού των στοιχείων του προβλήματος

Μόλις είδαμε τον χρόνο και το κόστος ολόκληρου του αλγορίθμου. Θα ήταν όμως ενδιαφέρων να βλέπαμε και το χρόνο και κόστος που χρειάζεται κάθε επεξεργαστής για να τερματίσει. Με αυτόν τον τρόπο θα μπορέσουμε να πούμε και ποιος από τους δύο επεξεργαστές είναι πιο γρήγορος σε συγκεκριμένο αριθμό στοιχείων.

Αποτελέσματα για Χρόνο και Κόστος για τον Επεξεργαστή P1

N	*msec*	*Κόστος*
4	**0.632**	**3**
8	**1.064**	**5**
16	**1.928**	**9**
32	**3.872**	**16**
256	**28.928**	**124**
1024	**114.680**	**532**
2048	**229.376**	**988**
8192	**916.472**	**3951**

Πίνακας 6.2

Αποτελέσματα για Χρόνο και Κόστος για τον Επεξεργαστή P2

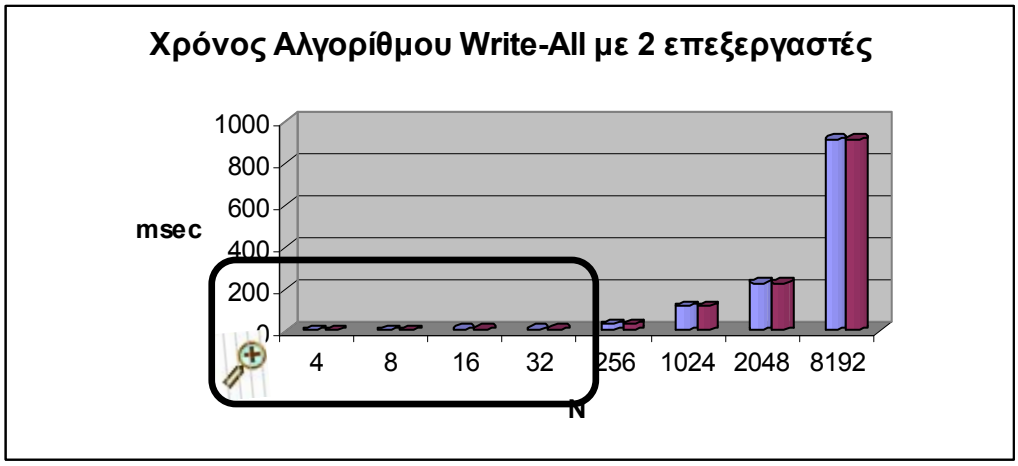
N	*msec*	*Κόστος*
4	**0.644**	**3**
8	**1.128**	**5**
16	**2.056**	**9**
32	**3.680**	**18**
256	**28.736**	**134**
1024	**114.808**	**495**
2048	**229.184**	**1062**
8192	**916.600**	**4244**

Πίνακας 6.3

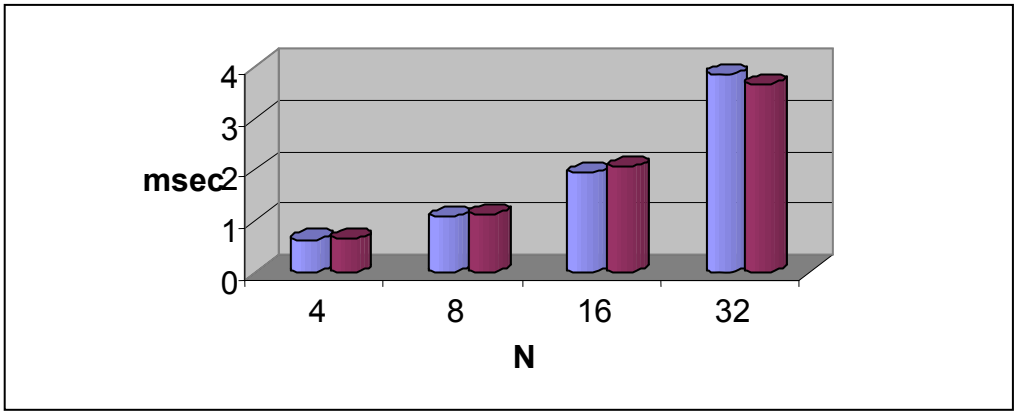
Από τις μετρήσεις που προκύπτουν για τον επεξεργαστή P1 αλλά και για τον επεξεργαστή P2 παρατηρούμε ότι πράγματι καθώς αυξάνονται τα στοιχεία αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος μας για να τερματίσει σωστά. Συνεπώς έπεται ότι ο χρόνος είναι ανάλογος του αριθμού των στοιχείων του προβλήματος. Επίσης αυξάνεται και το κόστος του κάθε επεξεργαστή, όπου και αυτό είναι

αναμενόμενο αφού το κόστος εξαρτάται από το πόσες κυψελίδες τις κοινόχρηστης μνήμης διαβάζει ο κάθε επεξεργαστής. Η απάντηση όμως στο ερώτημα για το πιος είναι πιο γρήγορος σε συγκεκριμένο πλήθος στοιχείων φαίνεται στην παρουσίαση της **Γραφικής 6.1 και 6.2**.

6.5 Παρουσίαση γραφικών και σύγκριση των δύο αλγορίθμων



Γραφική 6.1

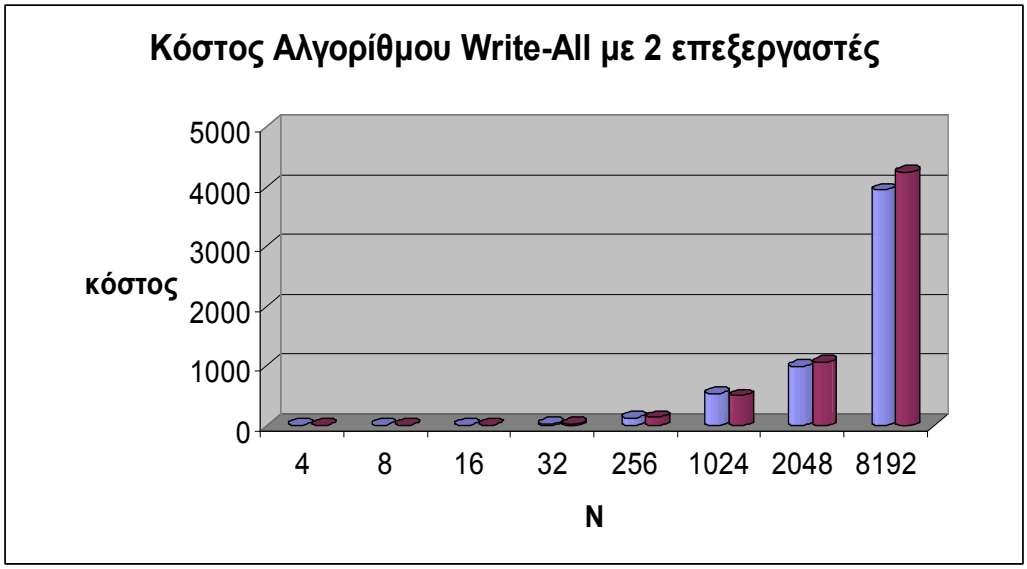


- **Χρόνος για τον επεξεργαστή P1**.
- **Χρόνος για τον επεξεργαστή P2.**

Γραφική 6.2

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο επεξεργαστής P2 για πλήθος στοιχείων ίσο με 4,8 και 16 καταναλώνει περισσότερο χρόνο από τον

επεξεργαστή P1. Για πλήθος όμως στοιχείων ίσο με 32 ο επεξεργαστής P2 είναι γρηγορότερος από τον επεξεργαστή P1. Παρατηρούμε λοιπόν ότι αν και το πλήθος των κυψελίδων που διαβάζουν οι δύο επεξεργαστές για αριθμό στοιχείων 4,8 και 16 είναι το ίδιο ο επεξεργαστής P1 είναι κατά ελάχιστα γρηγορότερος. Αυτή η περίπτωση όπως έχει αναφερθεί και παραπάνω είναι η χείριστη, όπου οι δύο επεξεργαστές διαβάζουν ακριβώς n/2+1 στοιχεία. Παρομοίως παρατηρούμε ότι αν και για πλήθος στοιχείων ίσο με 32, ο επεξεργαστής P2 είναι γρηγορότερος, διαβάζει πιο πολλές κυψελίδες μνήμης από τον επεξεργαστή P1. Από αυτό μπορούμε να συμπεράνουμε ότι ο επεξεργαστής P2 για το συγκεκριμένο πλήθος κυψελίδων είναι πιο γρήγορος από τον επεξεργαστή P1 και γι' αυτό διαβάζει περισσότερες κυψελίδες ούτως ώστε ο επεξεργαστής P1 που είναι πιο αργός να προσπαθήσει να γράψει σε μια κυψελίδα της κοινόχρηστης μνήμης στην οποία έχει ήδη γράψει ο επεξεργαστής P2.



- **Κόστος για τον επεξεργαστή P1**.
- **Κόστος για τον επεξεργαστή P2.**

Γραφική 6.3

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι στον αλγόριθμο Write-ALL και οι δύο επεξεργαστές για πλήθος στοιχείων 4, 8, 16 και 32 καταναλώνουν περίπου το ίδιο κόστος, εκτελούν δηλαδή την πράξη διάβασε-υπολόγισε-γράψε στον ίδιο αριθμό κυψελίδων. Επομένως οι ενδιαφέρουσες περιπτώσεις προκύπτουν για πλήθος στοιχείων μεγαλύτερο από 256. Για παράδειγμα, όταν έχουμε 256 στοιχεία ο επεξεργαστής P1

καταναλώνει λιγότερο κόστος από τον επεξεργαστή P2. Αυτό δεν ισχύει όμως και στην περίπτωση που έχουμε 1024 στοιχεία, αφού εδώ ο επεξεργαστής P2 καταναλώνει λιγότερο κόστος από τον επεξεργαστή P1 λόγω του ότι διαβάζει λιγότερες κυψελίδες της κοινόχρηστης μνήμης. Παρομοίως και για αριθμό στοιχείων 2048 και 8192 ο επεξεργαστής P2 διαβάζει περισσότερες κυψελίδες τις κοινόχρηστης μνήμης και δικαίως καταναλώνει και περισσότερο κόστος.

6.6 Γενικά συμπεράσματα

Συνοψίζοντας λοιπόν, καταλήγουμε στο συμπέρασμα ότι πράγματι ο χρόνος και το κόστος του αλγορίθμου αυξάνονται ανάλογα με την αύξηση του αριθμού των στοιχείων. Επίσης βλέπουμε ότι ο επεξεργαστής που διαβάζει περισσότερες κυψελίδες καταναλώνει και λιγότερο χρόνο για τον λόγο ότι είναι πιο γρήγορος. Αντιθέτως όμως το κόστος που καταναλώνει είναι πιο μεγάλο. Ακόμη παρατηρήθηκε ότι με την αύξηση των στοιχείων οι επεξεργαστές γίνονται όλο και πιο ασυγχρόνιστοι ενώ έχοντας μικρό πλήθος στοιχείων οι επεξεργαστές μας λειτουργούν σχεδόν εντελώς συγχρονισμένα.

Κεφάλαιο 7

Αλγόριθμοι Χ και Χ΄

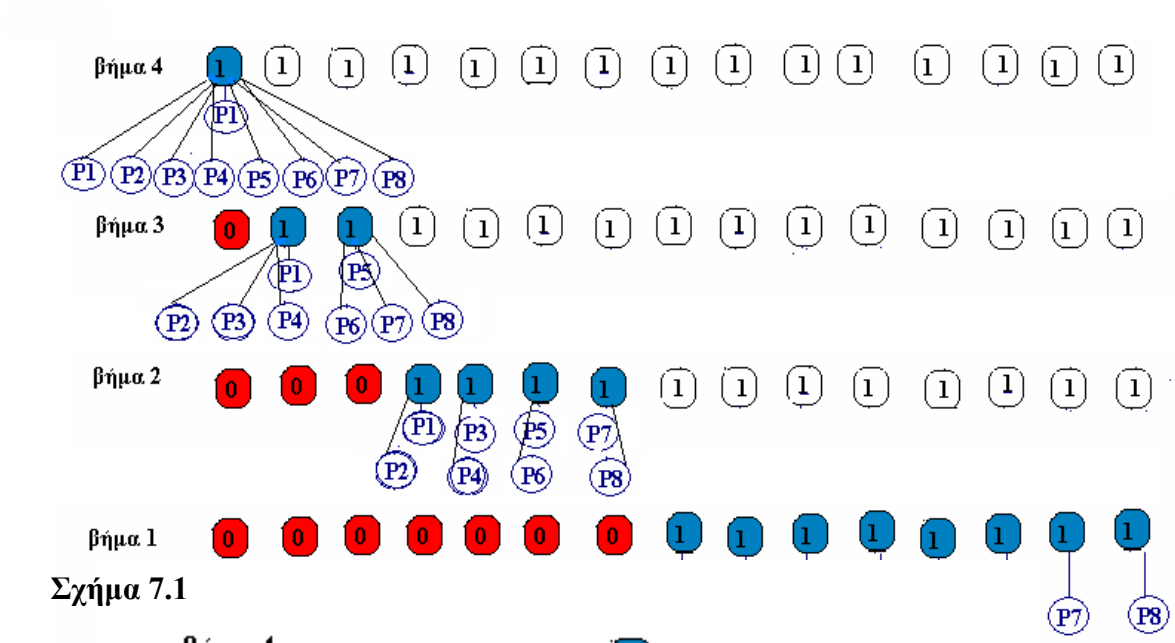
Προσπαθώντας να γενικεύσουμε το πιο πάνω πρόβλημα του Write-All σε $p \leq n$ επεξεργαστές αντί για 2, καταφεύγουμε στην τετριμμένη λύση όπου ο κάθε επεξεργαστής γράφει την τιμή 1 σε όλες τις κυψελίδες των στοιχείων. Επομένως ο κάθε επεξεργαστής χρεώνεται με n μονάδες κόστους και το συνολικό κόστος που προκύπτει είναι της τάξεως του $\Theta(p.n)$. Στην περίπτωση όμως που το $p=n$ τότε το κόστος του αλγορίθμου μας θα είναι ίσο με $\Theta(n^2)$.

Άραγε μπορούμε να έχουμε καλύτερο κόστος από αυτό; Η απάντηση είναι θετική. Μπορούμε να καταφέρουμε κόστος ελάχιστα πιο μικρό από $\Theta(n^2)$ χρησιμοποιώντας τον Αλγόριθμο Χ. Δηλαδή το νέο κόστος που θα πετύχουμε εκτελώντας αυτόν τον αλγόριθμο θα είναι της τάξεως του $O(n^{\lg 3}) = O(n^{1.6})$ στην χειρότερη περίπτωση.

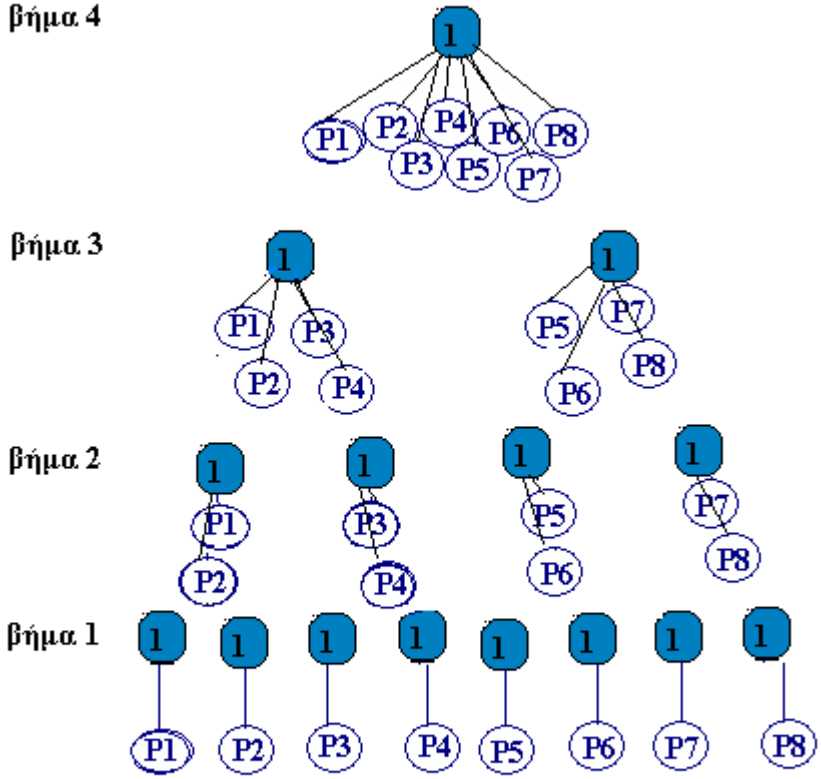
7.1 Πρόβλημα Write-ALL στο A-PRAM με n επεξεργαστές

Δεδομένου μιας λίστας n στοιχείων, όπου αρχικά τα στοιχεία έχουν την τιμή 0, ζητείται να μετατραπούν οι τιμές των στοιχείων σε 1 χρησιμοποιώντας n επεξεργαστές στο μοντέλο A-PRAM.

7.2 Αλγόριθμος X



Σχήμα 7.1



Σχήμα 7.2

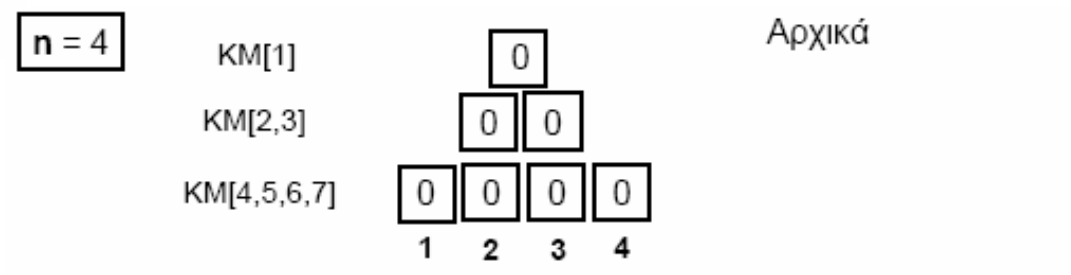
Όπως και στο πρόβλημα Write-All με δύο επεξεργαστές, οι επεξεργαστές μας θα εργάζονται εντελώς ασυγχρόνιστα. Αρχικά τα στοιχεία με την τιμή 0 βρίσκονται αποθηκευμένα στις πρώτες 2n-1 κυψελίδες της κοινόχρηστης μνήμης. Για την καλύτερη περιγραφή του αλγορίθμου θα προσπαθήσουμε να τον παρουσιάσουμε αναπαριστώντας το πρόβλημα μας ως ένα νοητό δυαδικό δένδρο, όπου τα στοιχεία του προβλήματος βρίσκονται αποθηκευμένα στα φύλλα του δέντρου δηλαδή στις κυψελίδες SM[n],…,SM[2n-1].

Αρχικά οι επεξεργαστές τοποθετούνται στα φύλλα του δέντρου με τον εξής τρόπο:
Ο επεξεργαστής Pi όπου $1 \leq i \leq n$ θα τοποθετηθεί στην θέση SM[i+#συνολικών επεξεργαστών-1]. Έπειτα κάθε επεξεργαστής ανάλογα με το πόσο γρήγορος είναι θα γράψει την τιμή 1 στο φύλλο που του αντιστοιχεί και θα ένα ανεβαίνει επίπεδο, καταλήγοντας στον πατέρα του φύλλου όπου βρισκόταν. Ακολούθως διαβάζει την τιμή που είναι αποθηκευμένη στον αντίστοιχο κόμβο. Αν η τιμή είναι το 1 τότε προχωράει και πάλι ένα βήμα προς τα πάνω και καταλήγει στον πατέρα του κόμβου που βρισκόταν. Αν όμως η τιμή που διαβάζει είναι ίση με 0 τότε ο επεξεργαστής κατεβαίνει ένα επίπεδο και ελέγχει τις τιμές που είναι αποθηκευμένες στο δεξιό και στον αριστερό θυγατρικό του κόμβου που βρισκόταν προηγουμένως πριν κατεβεί το επίπεδο. Αν κάποιος κόμβος από αυτούς έχει την τιμή 0 τότε ο επεξεργαστής μετατρέπει την τιμή της κυψελίδας σε 1 και επιστρέφει στον κόμβο που είναι ο πατέρας των δύο κόμβων αυτών, γράφει και εκεί την τιμή 1 και ανεβαίνει σε πιο ψηλό επίπεδο δηλαδή στον πατέρα του κόμβου που βρίσκεται. Αν όμως και οι δύο κόμβοι έχουν την τιμή 1 τότε ανεβαίνει πάλι επίπεδο, γράφει την τιμή 1 και ξανανεβαίνει επίπεδο. Υπάρχει όμως και η περίπτωση και τα δύο παιδιά να έχουν την τιμή 0. Στην περίπτωση αυτή λοιπόν οι επεξεργαστές που βρίσκονταν στον κόμβο που αναπαριστούσε τον πατέρα των δύο αυτών θυγατρικών χωρίζονται στα δύο υποδέντρα με τον εξής τρόπο: Υπολογίζεται ο δυαδικός αριθμός του κάθε επεξεργαστή με βάση τον προσωπικό του αριθμό και ανάλογα με το επίπεδο που βρισκόμαστε πάμε και ελέγχουμε το αντίστοιχο bit. Για παράδειγμα στην περίπτωση που ο επεξεργαστής P3 ο οποίος έχει δυαδικό αριθμό 011, βρίσκεται στο επίπεδο 1 τότε θα πρέπει να πάμε να ελέγξουμε το τελευταίο bit του δυαδικού του αριθμού, που στην συγκεκριμένη περίπτωση είναι το 1. Αν λοιπόν η τιμή του bit που υπολογίσαμε είναι ίση με 1 τότε ο συγκεκριμένος επεξεργαστής θα σταλεί στο δεξιό υποδέντρο, ενώ αν είναι ίση με 0 τότε θα σταλεί στο αριστερό υποδέντρο.

Αυτή η διαδικασία, όπου οι επεξεργαστές ανεβοκατεβαίνουν το νοητό δέντρο, γράφοντας την τιμή 1 στις κυψελίδες που διαβάζουν θα συνεχιστεί μέχρις ώτου έστω και ένας επεξεργαστής να διαβάσει ότι στην κυψελίδα SM[1] υπάρχει αποθηκευμένη η τιμή 1. Συνεπώς καταλήγουμε να έχουμε ένα ορθό αλγόριθμό που γράφει και στις 2n-1 κυψελίδες της κοινόχρηστης μνήμης την τιμή 1 χρησιμοποιώντας n ασυγχρόνιστους επεξεργαστές [2].

7.3 Παράδειγμα εκτέλεσης αλγορίθμου Χ

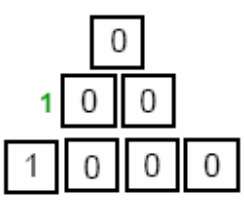
Επειδή ο αλγόριθμος είναι κάπως περίπλοκος ένα παράδειγμα θα ήταν απαραίτητο για την πλήρη κατανόηση του. Το παράδειγμα αυτό πάρθηκε από το [5].



Σχήμα 7.3

Δεδομένου ότι έχουμε σαν είσοδο 4 στοιχεία θα δεσμεύσουμε χώρο ίσο με 2n-1= 2*4-1=7 κυψελίδων στην κοινόχρηστη μνήμη. Αυτό επίσης συνεπάγεται ότι και οι επεξεργαστές μας θα είναι 4. Αρχικά και στις 7 θέσεις της κοινόχρηστης μνήμης γράφουμε την τιμή 0.

Βήμα 1

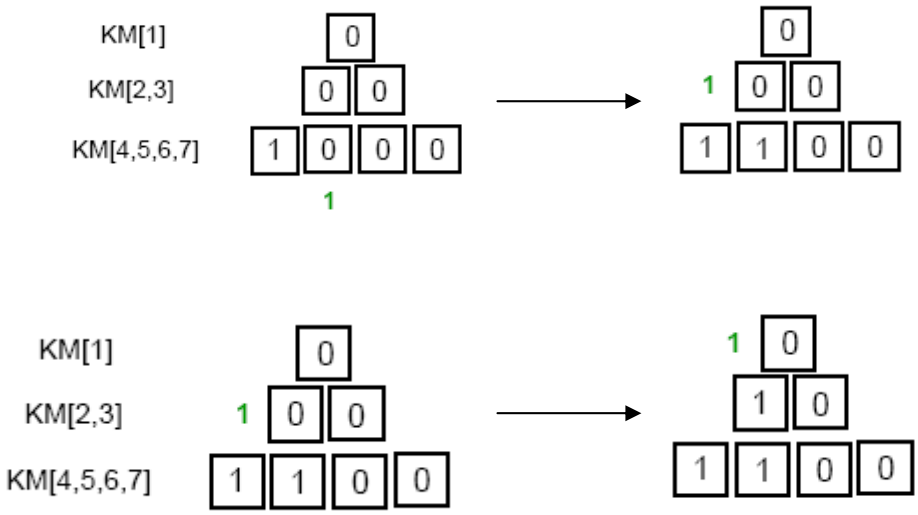


Σχήμα 7.4

Στο παράδειγμα αυτό υποθέτουμε ότι ο επεξεργαστής P1 είναι πολύ γρήγορος ενώ οι υπόλοιποι πολύ αργοί. Επομένως κατά την εκτέλεση του $1^{ου}$ βήματος ο επεξεργαστής P1 γράφει στο φύλλο με αριθμό ίσο με SM[1+4-1]= SM[4] την τιμή 1 και προχωράει

στον πατέρα αυτού του φύλλου δηλαδή στην θέση SM[⌊4/2⌋] = SM[2]. Έπειτα κατά την εκτέλεση του βήματος 2 ο επεξεργαστής ελέγχει αν στην κυψελίδα SM[2] είναι αποθηκευμένη η τιμή 1. Από το παράδειγμα φαίνεται ότι η τιμή που είναι αποθηκευμένη είναι το 0 επομένως ο επεξεργαστής θα πάει να ελέγξει τα παιδία της κυψελίδας αυτής δηλαδή την κυψελίδα SM[4] και την κυψελίδα SM[5]. Παρατηρούμε πως η κυψελίδα SM[5] έχει την τιμή 0. Έτσι ο επεξεργαστής P1 αλλάζει την τιμή της σε 1 και επιστρέφει πίσω στην κυψελίδα SM[2] για να γράψει και εκεί την τιμή 1.

Βήμα 2

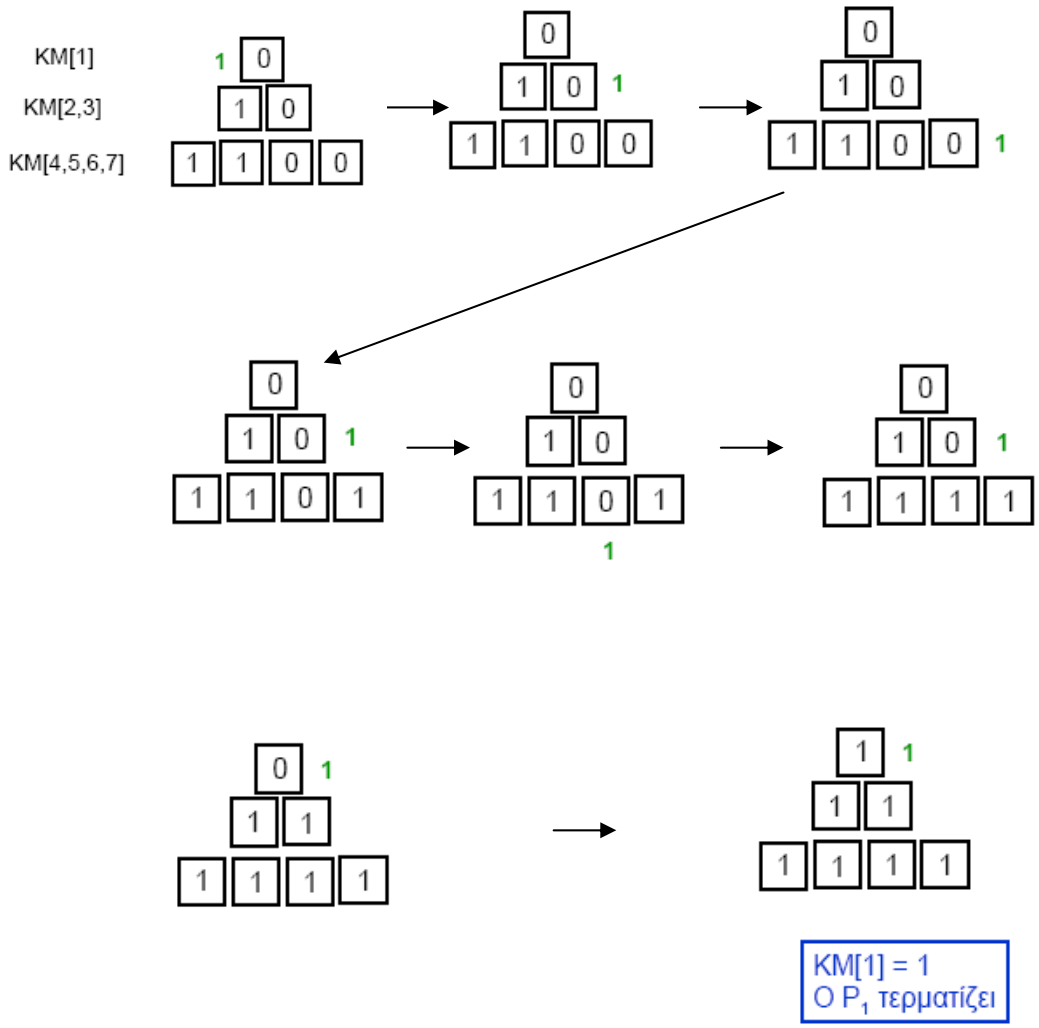


Σχήμα 7.5

Προχωρώντας λοιπόν στο επόμενο βήμα καταλήγουμε στην ρίζα του δέντρου. Αν και αυτό το βήμα είναι και το τελευταίο η δουλειά του επεξεργαστή P1 δεν τελειώνει εδώ γιατί απαραίτητη συνθήκη για να τερματίσει ο αλγόριθμος μας είναι η τιμή που θα είναι αποθηκευμένη στην ρίζα να είναι το 1 που αυτό δεν ισχύει σε αυτή την περίπτωση. Ακολούθως ο επεξεργαστής ελέγχει τις τιμές των κυψελίδων SM[2] και SM[3] και ανιχνεύει ότι στην κυψελίδα SM[3] υπάρχει αποθηκευμένη η τιμή 0. Επομένως ελέγχει τον αριστερό και τον δεξιός θυγατρικό της κυψελίδας SM[3]. Για τον λόγω ότι και τα δύο παιδιά έχουν την τιμή 0, υπολογίζεται ο δυαδικός αριθμος του επεξεργαστή P1, ο οποίος είναι το 0001. Επίσης το επίπεδο στο οποίο βρισκόμαστε είναι ίσο με 1. Συνεπώς θα ελεγχθεί το τελευταίο bit, το οποίο είναι ίσο με 1 και έτσι ο επεξεργαστής θα μετατρέψει τις τιμές των δύο φύλλων σε 1 ξεκινώντας πρώτα από τον δεξιό θυγατρικό. Έπειτα αλλάζει και το στοιχείο που είναι αποθηκευμένο στην SM[3]. Τέλος

καταλήγει στην ρίζα SM[1] και γράφει την τιμή 1, όπου πράγματι εδώ έχουμε τελειώσει.

Βήμα 3



Σχήμα 7.6

Αλγόριθμος X **[5]**

```
1.  Processors P1, ….. , Pn do asynchronously
2.          Pi: where = n + i - 1
3.              while SM[1] != 1 do
4.                  if (SM[where]!=1 && where > n-1) then %σε φύλλο
5.                          SM[where]=1
6.                          ⌊where=where/2⌋ %ανεβαίνει επίπεδο
7.                  else if  (SM[where] !=1 && where <= n-1) then %όχι σε φύλλο
8.                          if (SM[2*where]=1  && SM[2*where+1]=1) then
9.                                  SM[where]=1 % τελειωμένο υποδέντρο
10.                                 ⌊where= where/2⌋ %ανεβαίνει επίπεδο
11.                         if (SM[2*where]=1 && SM[2*where+1]=0) then
12.                                 where = 2*where+1 %κατεβαίνει επίπεδο
13.                         if (SM[2*where]=0 && SM[2*where]=1) then
14.                                 where =2*where %κατεβαίνει επίπεδο
15.                         if (SM[2*where]=0 && SM[2*where]=0) then
16.                                 if (i : even) then where = 1*where %κατεβαίνει
    επίπεδο
17.                                 else where = 2* where +1 %κατεβαίνει επίπεδο
18.                 end if
19.                 end while
20.  end do
```

7.4 Θεωρητικό κόστος αλγορίθμου

Ο προορισμός μας είναι να αναλύσουμε το κόστος αυτού του αλγορίθμου και να δούμε αν πράγματι καταφέραμε κάτι καλύτερο από n^2. Όπως έχουμε αναφέρει και στο πρόβλημα Write–All με δύο επεξεργαστές το κόστος για τους επεξεργαστές που λειτουργούν ασυγχρόνιστα μπορούμε να το βρούμε υπολογίζοντας την πράξη διάβασε-υπολόγισε-γράψε για κάθε επεξεργαστή και τέλος αθροίζοντας τα κόστη όλων των επεξεργαστών. Στην καλύτερη περίπτωση όπου όλοι οι επεξεργαστές λειτουργούν συγχρονισμένα έχουμε κόστος O(nlog n) που είναι και βέλτιστο. Όμως αυτό δεν συμβαίνει σχεδόν ποτέ. Επομένως είναι καλό να αναλύσουμε την χειρότερη περίπτωση. Στην περίπτωση αυτή έχουμε το σενάριο που οι μισοί επεξεργαστές γράφουν στο υποδέντρο που βρίσκεται αριστερά από την ρίζα. Έπειτα προχωράνε και στο δεξιό υποδέντρο για να ολοκληρώσουν τον αλγόριθμο όπου εκεί βρίσκουν και τους υπόλοιπους επεξεργαστές και λειτουργούν όλοι μαζί συγχρονισμένα. Σε αυτή την περίπτωση το κόστος που προκύπτει θα είναι :

$C_n(n) \leq C_{n/2}(n/2)+C_n(n/2) \leq C_{n/2}(n/2)+ 2C_{n/2}(n/2)=3\ C_{n/2}(n/2) \leq 3.3\ C_{n/4}(n/4) \leq 3.3.3\ C_{n/8}(n/8) \leq \ldots.. \leq 3^k\ C_{n/n}(n/n) \leq a.3^{\log_2 n} = a.n^{\log_2 3} = a.n^{1.6} \rightarrow C_n(n)=O(n^{1.6})$.

Όσον αφορά το χρόνο του αλγορίθμου δεν θα τον μελετήσουμε για τον ίδιο λόγω που έχουμε δει και στο πρόβλημα Write-All με δύο επεξεργαστές, αφού αυτό ισχύει σε κάθε αλγόριθμο που είναι υλοποιημένος με βάση το μοντέλο A-PRAM

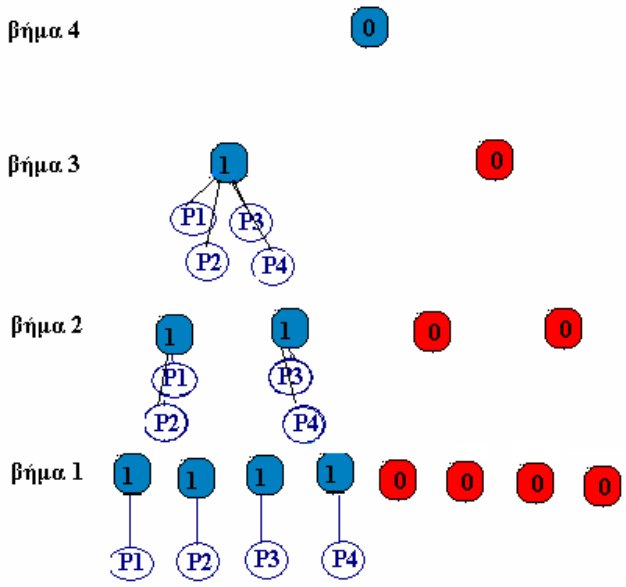
Επομένως ο αλγόριθμος X μετατρέπει την τιμή που βρίσκεται και στις n κυψελίδες από 0 σε 1 βοήθεια n επεξεργαστών σε κόστος O ($n^{1.6}$).

Παρατηρούμε όμως πως ο αλγόριθμος αυτός δεν είναι βέλτιστος, το κόστος του δεν είναι της τάξης του χρόνου του καλύτερου σειριακού αλγορίθμου.

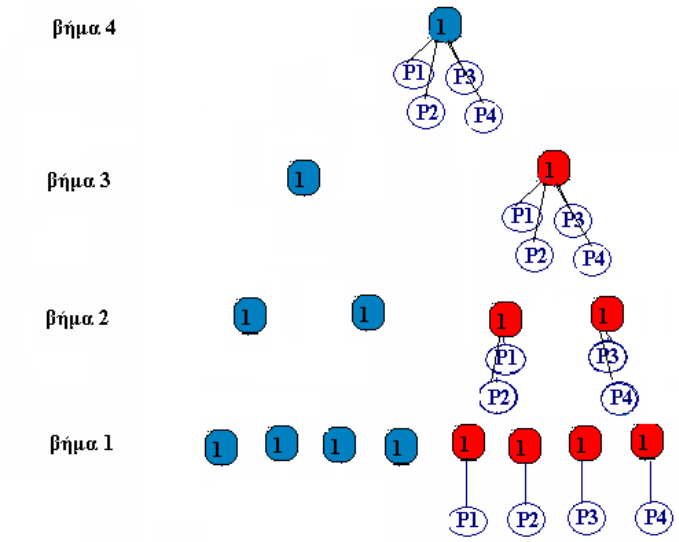
7.5 Αλγόριθμος X'

Παρατηρώντας τον πιο πάνω αλγόριθμο βλέπουμε ότι λειτουργεί ορθά για πλήθος επεξεργαστών ίσο με το πλήθος των στοιχείων (n). Τι γίνεται όμως στην περίπτωση που δεν μπορούμε να έχουμε στην διάθεση μας τόσο πλήθος επεξεργαστών;
Άραγε μπορούμε να καταφέρουμε πιο μικρό κόστος με το να μειώσουμε κάπως τους επεξεργαστές που χρησιμοποιούμε στο πρόβλημα μας;

Θεωρούμε λοιπόν ότι έχουμε p επεξεργαστές, όπου p<n. Αυτός ο αλγόριθμος που ονομάζεται και Αλγόριθμος X' είναι μία εκδοχή του Αλγορίθμου X.



Σχήμα 7.7



Σχήμα 7.8

Τα στοιχεία του προβλήματος Write-All βρίσκονται και πάλι αποθηκευμένα στα φύλλα του δέντρου (δηλαδή στις κυψελίδες SM[n],....,SM[2n-1]. Έπεται λοιπόν ότι κάθε επεξεργαστής τοποθετείται όπως και προηγουμένως στην κυψελίδα με αριθμό ίσο με το άθροισμα του προσωπικού του αριθμού και τον αριθμό n των στοιχείων. Η τελική θέση προκύπτει αφαιρώντας από το αποτέλεσμα του αθροίσματος τον αριθμό 1. Εκτελούμε τον αλγόριθμο X αλλά τώρα σε 2*p-1 κυψελίδες της κοινόχρηστης μνήμης. Έπειτα οι επεξεργαστές ανατίθενται στα φύλλα των επόμενων 2*p-1 στοιχείων με τον ίδιο τρόπο που τοποθετήθηκαν και προηγουμένως, με μόνη διαφορά ότι τώρα προσθέτουμε στην θέση που προκύπτει και τον αριθμό p. Εκτελείται και πάλι ο αλγόριθμος X. Αυτή η διαδικασία συνεχίζεται μέχρις ώτου να έχουμε γράψει σε όλα τα 2n-2 στοιχεία, απλά κάθε φορά αλλάζουμε την θέση που παίρνουν οι επεξεργαστές στα φύλλα. Παρατηρούμε ότι τα στοιχεία του προβλήματος Write-All διαμοιράζονται σε n/p ομάδες, όπου κάθε ομάδα έχει p διαφορετικά στοιχεία. Οπότε, οι p επεξεργαστές, καλούνται να επιλύσουν n/p προβλήματα Write-All μεγέθους p, το ένα μετά το άλλο εκτελώντας τον αλγόριθμο X για κάθε ομάδα. Τέλος όλοι οι επεξεργαστές μετατρέπουν την ρίζα του προβλήματος από 0 σε 1.

Απομένει τώρα να αναλύσουμε το κόστος που χρειάζεται ο αλγόριθμος για να τερματίσει γράφοντας στην κυψελίδα SM[1] (την ρίζα του δέντρου) την τιμή 1.

Όπως ήδη παρατηρήσαμε ο αλγόριθμος Χ' χωρίζεται σε n/p ομάδες όπου σε κάθε ομάδα εκτελούμε τον αλγόριθμο Χ πάνω σε p στοιχεία. Οδηγούμαστε λοιπόν στο ότι ο αλγόριθμος Χ θα εκτελεστεί n/p φορές. Επομένως γνωρίζοντας ότι έχουμε κόστος $O(p*p^{lg3})$ για κάθε εκτέλεση του αλγορίθμου Χ, το κόστος μας θα μειωθεί σε $O(n/p *p^{lg3}) \rightarrow O(n*p^{lg3/2})$.

Αυτό συνεπάγεται ότι το συνολικό κόστος εξαρτάτε και από το πλήθος των επεξεργαστών. Συνεπώς εισάγοντας λιγότερους επεξεργαστές θα έχουμε και μικρότερο κόστος.

Επομένως ο αλγόριθμος Χ' για επίλυση του προβλήματος Write-All με p επεξεργαστές είναι ορθός καταναλώνοντας κόστος ίσο με O $(n*p^{lg3/2})$.

7.6 Πειραματική αξιολόγηση αλγορίθμων και συμπεράσματα

Άραγε το άνω φράγμα κόστους επιβεβαιώνεται και στην πράξη; Υλοποιώντας τους πιο πάνω αλγόριθμους στο μοντέλο SB-PRAM, αλλά τώρα με τους p επεξεργαστές να εργάζονται εντελώς ασυγχρόνιστα και με την βοήθεια της γλώσσας FORK παίρνουμε τις παρακάτω μετρήσεις όσο αφορά το κόστος του αλγορίθμου. Όπως έχει αναφερθεί και παραπάνω, πειραματικά μπορούμε να υπολογίσουμε τον χρόνο γιατί τώρα μπορούμε να μιλάμε για κάποιο συγκεκριμένο σενάριο όπου κάποιοι από τους n επεξεργαστές μπορεί να είναι πιο αργοί, ενώ άλλοι πιο γρήγοροι. Επομένως πάρθηκαν και μετρήσεις για τον χρόνο που καταναλώνει ο αλγόριθμος μας οι οποίες επίσης παραθέτονται παρακάτω. Στους συγκεκριμένους αλγόριθμους και ειδικότερα στο κομμάτι του κώδικα που παρουσιάζονται οι αλγόριθμοι δεν χρησιμοποιήθηκε καθόλου η εντολή ***barrier*** και το κομμάτι του κώδικα βρισκόταν μέσα στη συνάρτηση ***async();*** χωρίς καμία εντολή που να συγχρονίζει τους επεξεργαστές. Με αυτό τον τρόπο οι επεξεργαστές μας λειτουργούσαν εντελώς ασυγχρόνιστα.
Για να πάρουμε τις μετρήσεις για τον αλγόριθμο Χ και Χ' δεν ακολουθήσαμε την ίδια μεθοδολογία με τους προηγούμενους αλγορίθμους όπου παίρναμε 5 μετρήσεις για κάθε αλγόριθμο αλλά τώρα παίρνουμε μόνο μια μέτρηση. Αυτό συμβαίνει γιατί σε κάθε εκτέλεση του βέλτιστου και μη αλγορίθμου οι επεξεργαστές που θα είναι πιο γρήγοροι

οι πιο αργοί δεν θα παραμένουν σταθεροί δηλαδή αν παραδείγματος χάριν εκτελώντας μια φορά τον αλγόριθμο ο επεξεργαστής P1 ήταν πιο γρήγορος από όλους τους άλλους στην δεύτερη μπορεί να ήταν ο πιο αργός ή να βρισκόταν κάπου στη μέση. Ουσιαστικά λοιπόν είναι σαν να μιλάμε για διαφορετικά σενάρια. Επομένως δεν θα ήταν ορθό αν ακολουθούσαμε την μέθοδο με τις 5 μετρήσεις.

Αλγόριθμος Χ

N	*P*	*Msec*	*Κόστος*
4	**4**	**2.516**	**12**
8	**8**	**3.356**	**32**
16	**16**	**4.076**	**80**
32	**32**	**4.892**	**192**
64	**64**	**5.708**	**448**
128	**128**	**8.556**	**1024**

Πίνακας 7.1

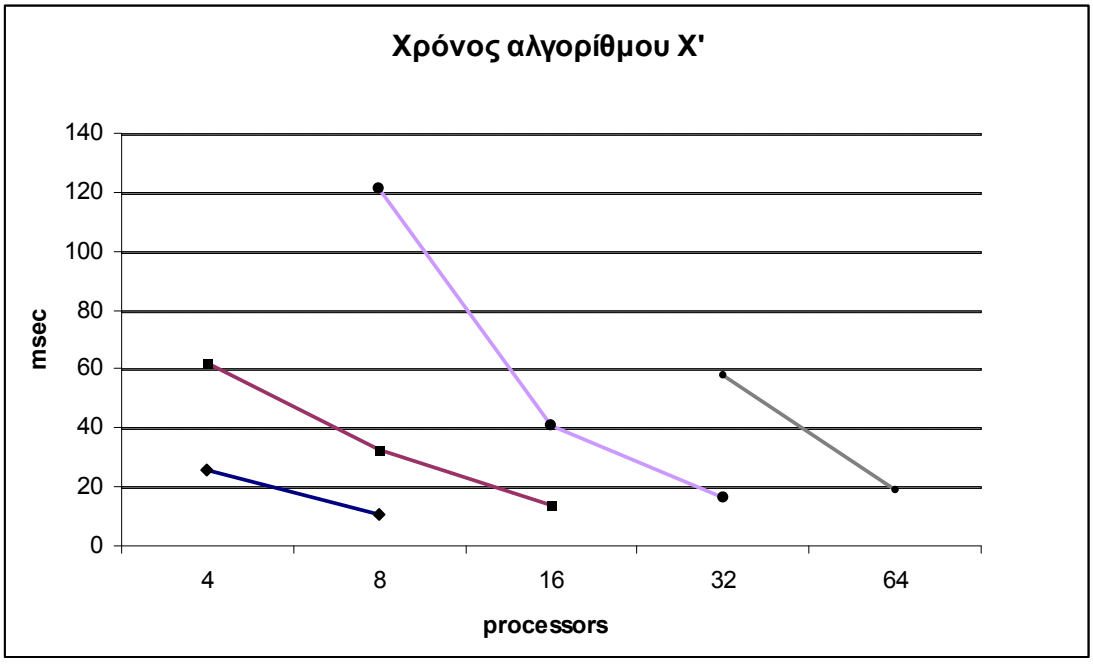
Από τις παραπάνω μετρήσεις παρατηρούμε ότι πράγματι με την αύξηση των στοιχείων στο πρόβλημα μας αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος για να επιλύσει το πρόβλημα και να τερματίσει. Παρομοίως και το κόστος είναι ανάλογο της αύξησης των στοιχείων. Η παρατήρηση αυτή μας οδηγεί στο να επιβεβαιώσουμε την εξίσωση για χρόνο και κόστος που είχαμε κατά την θεωρητική αξιολόγηση στην οποία συμπεριλάβαμε μέσα σ' αυτή και το πλήθος n των στοιχείων.

Αλγόριθμος Χ'

N	*P*	*msec*	*Κόστος*
8	**4**	**7.900**	**20**
16	**4**	**25.556**	**36**
16	**8**	**10.556**	**60**
32	**4**	**62.340**	**68**
32	**8**	**32.596**	**96**
32	**16**	**13.532**	**149**
64	**8**	**120.892**	**145**
64	**16**	**40.980**	**224**
64	**32**	**16.508**	**351**
128	**32**	**57.836**	**498**
128	**64**	**19.044**	**780**

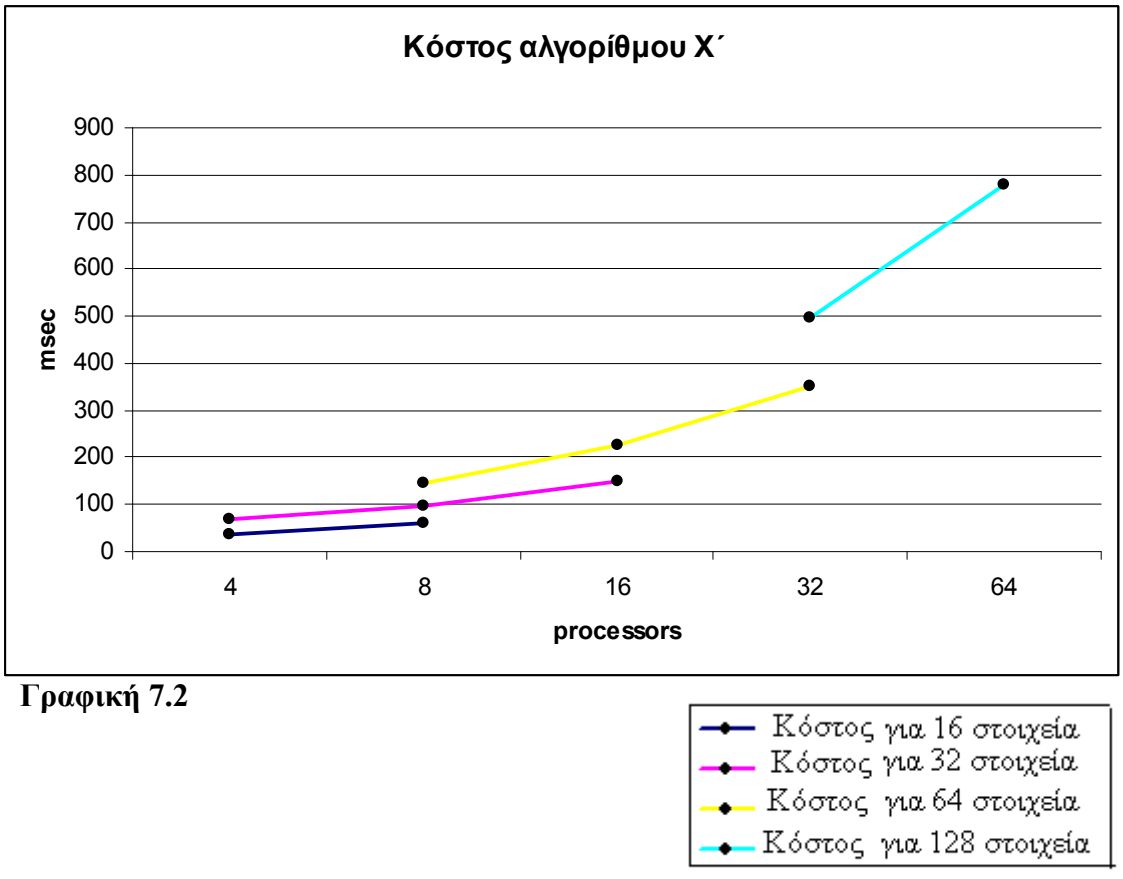
Πίνακας 7.2

Από τις μετρήσεις που πήραμε στο βέλτιστο αλγόριθμο παρατηρούμε πως πάλι έχουμε μια αύξηση στον χρόνο και στο κόστος καθώς τα στοιχεία αυξάνονται.



Γραφική 7.1

Αυτό μπορούμε να το δούμε και στην παραπάνω γραφική παράσταση όπου ο χρόνος που προκύπτει για ίδιο αριθμό επεξεργαστών αλλά διαφορετικό αριθμό στοιχείων είναι μεγαλύτερος στην περίπτωση που έχουμε περισσότερα στοιχεία. Επιπλέον, βλέπουμε ότι με ίδιο πλήθος στοιχείων αλλά με διαφορετικό πλήθος επεξεργαστών, στην περίπτωση που χρησιμοποιούμε λιγότερους επεξεργαστές έχουμε μεγαλύτερο χρόνο παρότι στην περίπτωση που έχουμε περισσότερους επεξεργαστές. Επομένως ο χρόνος εξαρτάτε και από τον αριθμό των επεξεργαστών. Έπεται λοιπόν ότι ο χρόνος είναι αντίστροφα ανάλογος του πλήθος των επεξεργαστών.



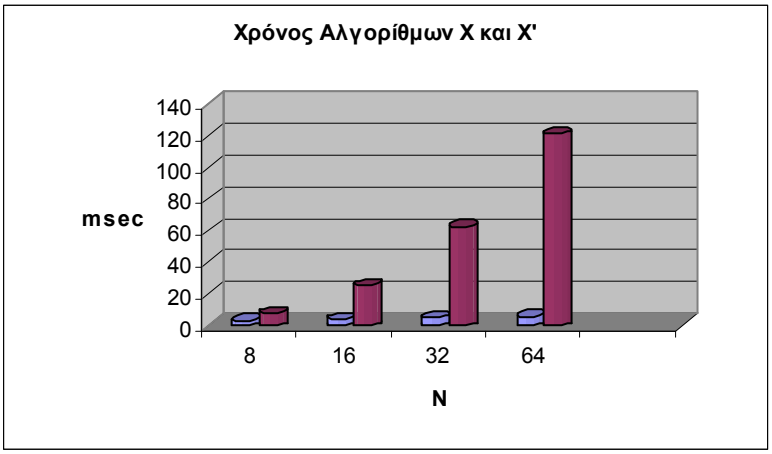
Γραφική 7.2

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι όσο αυξάνονται τα στοιχεία αυξάνεται και το κόστος του αλγορίθμου. Αρχικά για μικρό πλήθος στοιχείων δεν παρατηρούμε και μεγάλη διαφορά, όσο όμως τα στοιχεία γίνονται περισσότερα η διαφορά του κόστους είναι πιο εμφανής. Επίσης για ίδιο αριθμό στοιχείων αλλά για διαφορετικούς επεξεργαστές, το κόστος είναι πιο μεγάλο σε περίπτωση που έχουμε

περισσότερους επεξεργαστές. Αυτή η παρατήρηση επίσης είναι πιο εμφανής σε μεγαλύτερο αριθμό στοιχείων. Επομένως το κόστος εξαρτάται από το πλήθος των στοιχείων και το πλήθος των επεξεργαστών. Επιβεβαιώνεται λοιπόν η θεωρητική αξιολόγηση που κάναμε για το κόστος του αλγορίθμου.

Από τις δύο γραφικές καταλήξαμε στο συμπέρασμα ότι το κόστος και ο χρόνος για ένα συγκεκριμένο επεξεργαστή εξαρτάται από το πλήθος των στοιχείων. Όσο μεγαλώνει το πλήθος των στοιχείων μεγαλώνει και ο χρόνος και το κόστος αντίστοιχα. Όμως όσο αφορά το πλήθος των επεξεργαστών προκύπτει μια ειδική περίπτωση. Όσο αυξάνεται το πλήθος των επεξεργαστών έχουμε λιγότερο χρόνο αλλά περισσότερο κόστος. Αναλόγως με το τι θέλουμε να πετύχουμε εισάγουμε και το κατάλληλο πλήθος επεξεργαστών. Αν και σε αυτή την περίπτωση η ταχύτητα του αλγορίθμου δεν είναι το πρωτεύον ζήτημα αφού σκοπός του βέλτιστου ήταν να μειώσει το κόστος του αλγορίθμου. Συνεπώς όσο πιο λίγους επεξεργαστές εισάγουμε τόσο το καλύτερο.
Χρησιμοποιώντας αυτή την παρατήρηση στις συγκρίσεις των δύο αλγορίθμων που παρουσιάζονται πιο κάτω ο αλγόριθμος Χ' χρησιμοποιεί χρόνο και κόστος που πάρθηκαν από την εκτέλεση του αλγορίθμου με τους λιγότερους επεξεργαστές για ένα συγκεκριμένο στοιχείο.

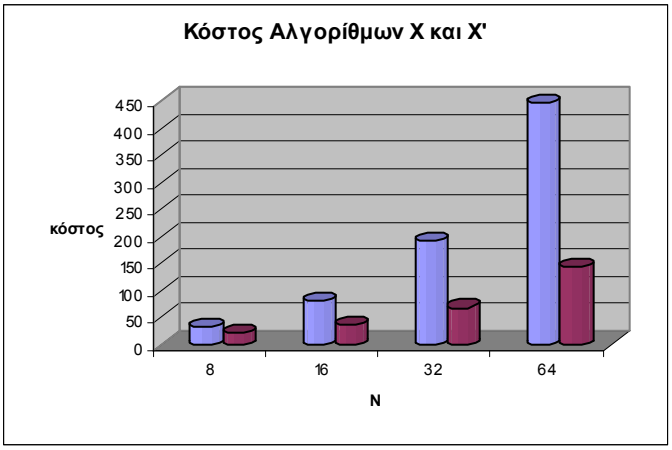
7.7 Παρουσίαση γραφικών και σύγκριση των δύο αλγορίθμων



Γραφική 7.3

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος Χ είναι κατά πολύ γρηγορότερος από τον Χ'. Αυτό δεν είναι λάθος, διότι ο αλγόριθμος Χ θα καλεστεί n/p φορές κατά την εκτέλεση του αλγορίθμου Χ' για να επιλύσει το πρόβλημα Write-All και έτσι δικαιολογημένα αυτός ο αλγόριθμος καταναλώνει περισσότερο χρόνο.

Επομένως δικαιολογημένα στην πράξη ο αλγόριθμος Χ είναι γρηγορότερος από τον αλγόριθμο Χ'.



- **Αλγόριθμος Χ.**
- **Αλγόριθμος Χ'.**

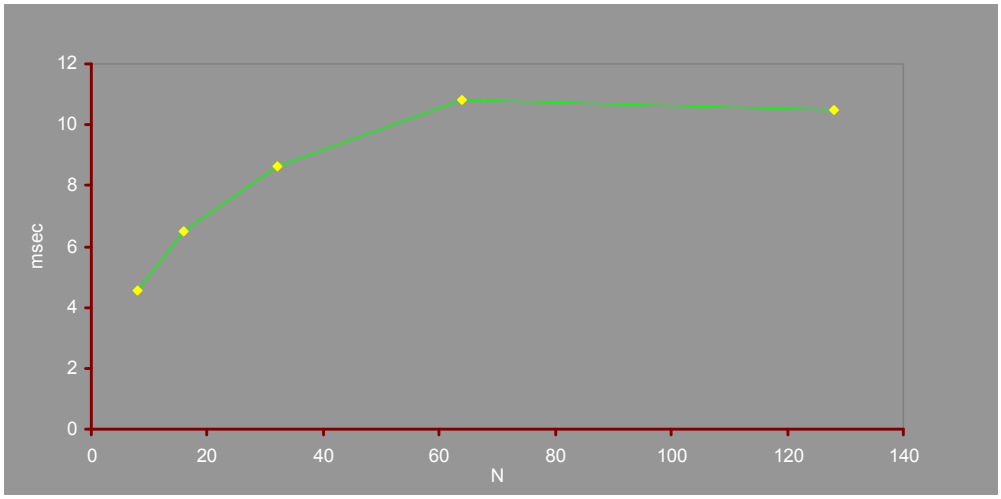
Γραφική 7.4

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος Χ' έχει πολύ πιο μικρό κόστος από τον αλγόριθμο Χ. Πράγματι αυτό ισχύει και στην θεωρητική αξιολόγηση του κόστους για τους δύο αυτούς αλγορίθμους. Καταφέραμε λοιπόν να μειώσουμε το κόστος του βέλτιστου αλγορίθμου αν και ο χρόνος του ήταν μεγαλύτερος. Αυτή η μείωση του κόστους λοιπόν οφείλεται στο ότι χρησιμοποιήσαμε λιγότερους επεξεργαστές από το πλήθος των στοιχείων.

Συνοψίζοντας έχουμε δει ότι ο αλγόριθμος Χ' έχει μικρότερο κόστος εκτέλεσης αλλά καταναλώνει πολύ περισσότερο χρόνο. Επομένως σίγουρα το ότι καταφέραμε και μειώσαμε το κόστος χρησιμοποιώντας λιγότερους επεξεργαστές είναι πολύ σημαντικό. Τι γίνεται όμως αν καθώς αυξάνονται τα στοιχεία αυξάνεται κατά πολύ ο χρόνος του αλγορίθμου Χ' και η διαφορά που προκύπτει με τον χρόνο αλγορίθμου Χ είναι πολύ μεγάλη; Αυτό θα πρέπει να μας προβληματίσει γιατί καλό είναι να έχουμε μικρό κόστος αλλά και ο χρόνος παίζει ένα ουσιαστικό ρόλο. Αυτό λοιπόν μπορούμε να το δούμε υπολογίζοντας την πρακτική διαφορά που προκύπτει μεταξύ των δύο αλγορίθμων.

N	*Χρόνος Αλγόριθμου Χ*	*Χρόνος Αλγόριθμου Χ'*	*Διαφορά*
8	**3.356**	**7.9**	**4.544**
16	**4.076**	**10.556**	**6.48**
32	**4.892**	**13.532**	**8.64**
64	**5.708**	**16.508**	**10.8**
128	**8.556**	**19.044**	**10.488**

Πίνακας 7.3



Γραφική 7.5

Από αυτή την γραφική λοιπόν παρατηρούμε ότι η διαφορά στο χρόνο του αλγορίθμου Χ από τον αλγόριθμο Χ' που προκύπτει μας δίνει μια λογαριθμική συνάρτηση. Αρχικά λοιπόν καθώς αυξάνονται τα στοιχεία αυξάνεται και η διαφορά του χρόνου. Ευτυχώς όμως από κάποιο σημείο και μετά αυτή η διαφορά αρχίζει να παραμένει σταθερή. Επομένως συμπεραίνουμε ότι αν και ο αλγόριθμος Χ' καταναλώνει περισσότερο χρόνο στην εκτέλεση του από τον αλγόριθμο Χ η διαφορά του χρόνου είναι λογαριθμική και έπεται από κάποιο σημείο και μετά σταθερή. Επομένως ο αλγόριθμος Χ' είναι πολύ καλύτερος από τον αλγόριθμο Χ αφού καταφέρνει να μειώσει κατά πολύ το κόστος αλλά και όσο αφορά τον χρόνο η διαφορά που προκύπτει μεταξύ των δύο αλγορίθμων για μεγάλο αριθμό στοιχείων παραμένει σταθερή.

7.8 Γενικά συμπεράσματα

Συνοψίζοντας λοιπόν, καταλήγουμε στο συμπέρασμα ότι κατά την πρακτική αξιολόγηση ο χρόνος του αλγορίθμου Χ' είναι μεγαλύτερος από το χρόνο του αλγορίθμου Χ. Όπως όμως έχουμε δει η διαφορά που προκύπτει στο χρόνο των δύο αυτών αλγορίθμων δεν είναι και πολύ μεγάλη αλλά καθώς αυξάνονται τα στοιχεία παίρνουμε μια λογαριθμική σχέση. Το κόστος όμως που προκύπτει είναι πολύ μεγαλύτερο στον αλγόριθμο Χ, το οποίο επιβεβαιώνει και την θεωρητική αξιολόγηση που κάναμε για τους δύο αυτούς αλγορίθμους όσο αφορά το κόστος τους. Επομένως όντως ο αλγόριθμος Χ' είναι πολύ καλύτερος από τον αλγόριθμο Χ.. Καταλήξαμε ακόμη στο συμπέρασμα ότι όσο πιο λίγοι είναι οι επεξεργαστές που εργάζονται για την επίλυση του προβλήματος για ένα συγκεκριμένο πλήθος στοιχείων κατά την εκτέλεση του αλγορίθμου Χ', τόσο πιο λίγο κόστος καταναλώνεται αντιθέτως με τον χρόνο που είναι πολύ πιο μεγάλος.

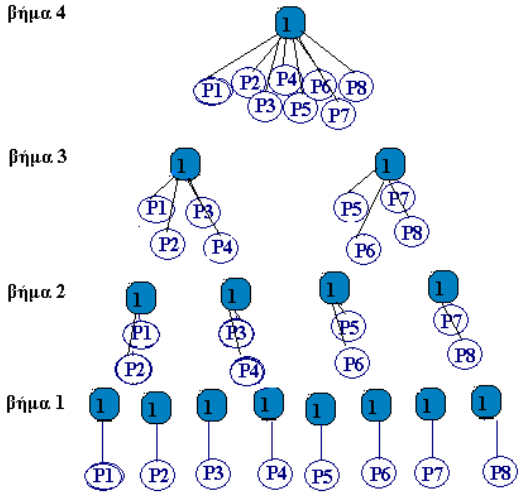
Κεφάλαιο 8

Αλγόριθμος W

8.2 Πρόβλημα Write-ALL στο F-PRAM με n επεξεργαστές

Θα επικεντρωθούμε και πάλι στη μελέτη και επίλυση του απλού προβλήματος, Write-All αλλά τώρα θα εργαστούμε στο μοντέλο F-PRAM που όπως έχουμε αναφέρει και παραπάνω είναι ένα μοντέλο που ανέχεται τα σφάλματα επεξεργαστών. Δεδομένου λοιπόν μιας λίστας n στοιχείων, όπου αρχικά τα στοιχεία έχουν την τιμή 0, ζητείται να μετατραπούν οι τιμές των στοιχείων σε 1 χρησιμοποιώντας p επεξεργαστές στο μοντέλο F-PRAM (δηλαδή μέχρι f<p επεξεργαστές μπορούν να καταρρεύσουν).

8.3 Αλγόριθμος W



Σχήμα 8.1

Αν και το πρόβλημα αυτό μοιάζει κατά πολύ με τον αλγόριθμο X, εδώ υπάρχει μια πολύ μεγάλη διαφορά, οι επεξεργαστές οι οποίοι καταρρέουν δεν επανέρχονται ποτέ πίσω. Επομένως για να λύσουμε αυτό το πρόβλημα πρέπει να εισαχθεί ένας καινούργιος αλγόριθμος στον οποίο θα μπορούμε να έχουμε ανοχή σφαλμάτων.
Αρχικά τα στοιχεία με την τιμή 0 βρίσκονται αποθηκευμένα στις πρώτες 2n-1 κυψελίδες της κοινόχρηστης μνήμης. Ο αλγόριθμος αυτός εκτελεί ένα βρόγχο τεσσάρων φάσεων έως ότου να λυθεί το πρόβλημα.

Στην πρώτη φάση W1 γίνεται καταμέτρηση των επεξεργαστών για εύρεση σφαλμάτων. Χρησιμοποιείτε λοιπόν ένα διάνυσμα Δ1 που είναι αναπαριστάμενο ως ένα νοητό πλήρες ισοσταθμισμένο δυαδικό δέντρο, για την καταμέτρηση των επεξεργαστών που δεν έχουν καταρρεύσει. Οι επεξεργαστές ξεκινώντας από τα φύλλα του δέντρου (τις κυψελίδες SM[1],…,SM[n]),στα οποία τοποθετούνται με βάση τον προσωπικό τους αριθμό συν το πλήθος των στοιχείων πλην τον αριθμό 1, γράφουν την τιμή 1 και ανεβαίνουν το δέντρο τρέχοντας μια εκδοχή του αλγορίθμου παράλληλης άθροισης στο PRAM . Τέλος στη ρίζα του δέντρου αποθηκεύεται ο υπολογισμένος αριθμός των επεξεργαστών που δεν έχουν καταρρεύσει. Αυτός ο αριθμός είναι μια υπέρ-εκτίμηση του πραγματικού αριθμού των επεξεργαστών που δεν έχουν καταρρεύσει ακόμη.

Η δεύτερη φάση W2, είναι η φάση στην οποία γίνεται ανοχή των σφαλμάτων μέσω ισοζυγισμένης ανάθεσης επεξεργαστών σε στοιχεία της λίστας. Χρησιμοποιούμε λοιπόν και πάλι ένα διάνυσμα Δ2, αναπαραστημένο ως ένα νοητό δυαδικό δέντρο, για να

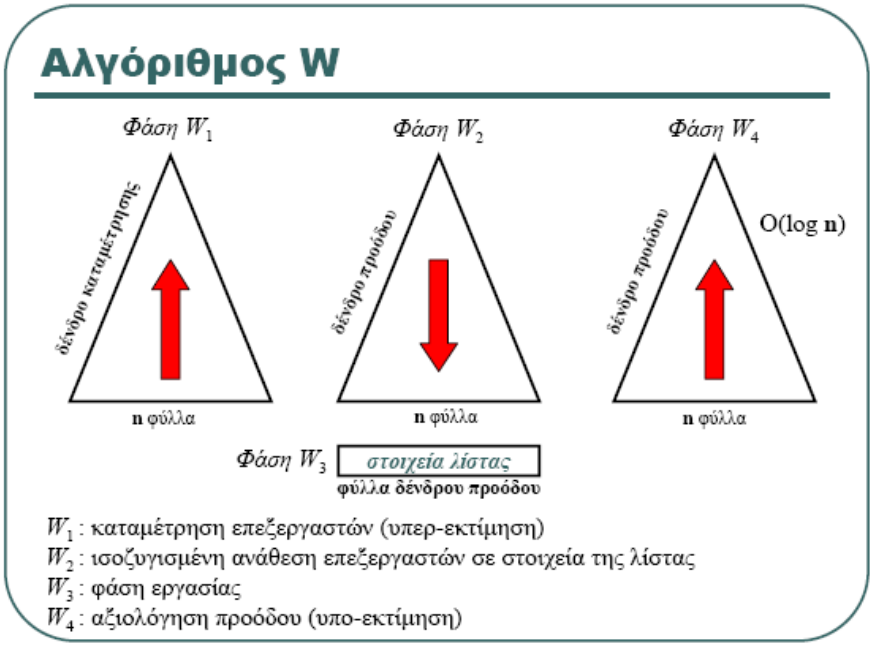
βοηθήσει στην ανάθεση επεξεργαστών σε στοιχεία της λίστας. Οι επεξεργαστές τώρα δεν ξεκινούν από τα φύλλα του δέντρου αλλά από την ρίζα (την κυψελίδα SM[1]) στην οποία υπάρχει αποθηκευμένη μια εκτίμηση του αριθμού των στοιχείων της λίστας που προκύπτει από την φάση W4. Έπειτα κατεβαίνουν προς τα κάτω και διαχωρίζονται στα φύλλα του δέντρου(που βρίσκονται τα στοιχεία της λίστας) χρησιμοποιώντας την εκτίμηση των ενεργών επεξεργαστών και τον αριθμό των υπολειπόμενων μηδενικών της λίστας. Αυτό μας λέει βασικά ότι σε κάθε επανάληψη του αλγορίθμου, με το τέλος της φάσης W2, οι επεξεργαστές είναι ανατιθέμενοι ισοζυγισμένα, σε στοιχεία της λίστας που δεν είναι σίγουρο ότι έχουν την τιμή 1.

Η τρίτη φάση W3 είναι και η φάση εργασίας, όπου τώρα οι ενεργοί επεξεργαστές βρίσκονται στα φύλλα του δέντρου προόδου και ο κάθε επεξεργαστής αλλάζει την τιμή του στοιχείου που αντιστοιχεί στο φύλλο που βρίσκεται από 0 σε 1.

Τέλος στη τελευταία φάση, τη φάση W4 έχουμε την αξιολόγηση προόδου όπου όλοι οι επεξεργαστές ξεκινούν από τα φύλλα του δέντρου προόδου που βρέθηκαν στην προηγούμενη φάση και ανεβαίνουν προς τα πάνω για να υπολογίσουν τον αριθμό των στοιχείων της λίστας που έχουν την τιμή 1. Αυτό υλοποιείται με μια εκδοχή του αλγορίθμου παράλληλης άθροισης του PRAM. Η ρίζα του δέντρου περιέχει το αποτέλεσμα του αθροίσματος. Το αποτέλεσμα αυτό είναι μια υπό -εκτίμηση του πραγματικού αριθμού των στοιχείων που έχουν τιμή 1.
Ο βρόγχος των τεσσάρων φάσεων τερματίζει, μόνο όταν η ρίζα του δένδρου προόδου έχει τιμή n [2].

Αλγόριθμος W **[5]**

```
1.  Processors i=1, …, n do in parallel
2.     PhaseW3
3.   Phase W4
4.   while δ2[1] != n do
5.                    Phase W1
6.                    Phase W2
7.                    Phase W3
8.                    Phase W4
9.                    end while
10. end do
```



Σχήμα 8.2 [5]

8.4 Θεωρητικός χρόνος και κόστος αλγορίθμου

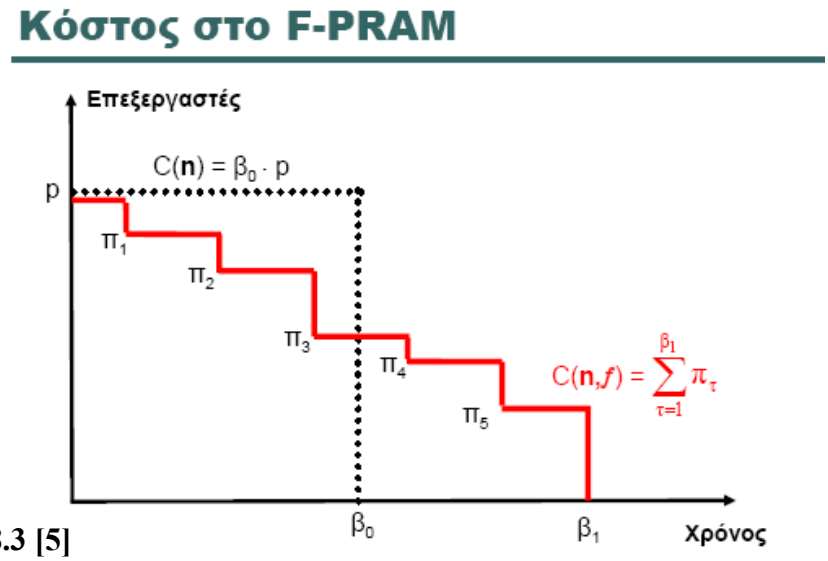
Ας αναλύσουμε τώρα τον χρόνο απόκρισης του αλγορίθμου μας. Ο βρόγχος επανάληψης της γραμμής 4 στην χειρότερη περίπτωση όπου ένας επεξεργαστής παραμένει ενεργός εκτελείται n φορές έως ότου η ρίζα του δέντρου προόδου να έχει μέσα της αποθηκευμένη την τιμή n. Επομένως τώρα απομένει να δούμε πόσος χρόνος καταναλώνεται σε κάθε επανάληψη του βρόγχου. Προκειμένου να υπολογίσουμε τον χρόνο που καταναλώνεται σε κάθε επανάληψη, πρέπει να δούμε πόσος χρόνος καταναλώνεται σε κάθε φάση του αλγορίθμου. Συγκεκριμένα, στις φάσεις W1, W2 και W4 καταναλώνουμε χρόνο ίσο με O(log n), λόγω του ότι ο κάθε επεξεργαστής επισκέπτεται τις log n κυψελίδες που βρίσκονται κατά μήκος του δέντρου. Επίσης στην φάση W3 καταναλώνουμε σταθερό χρόνο, λόγω του ότι οι επεξεργαστές εκτελούν σταθερό πλήθος πράξεων αφού το μόνο που κάνουν είναι να γράφουν στα φύλλα του δέντρου την τιμή 1. Συνεπώς ο συνολικός χρόνος που προκύπτει από τις γραμμές 4-9 στη χειρότερη περίπτωση είναι O(n log n).

Βασικά όμως σε αυτό το μοντέλο ο χρόνος δεν παίζει ουσιαστικό ρόλο στον σχεδιασμό των αλγορίθμων, αυτό που παίζει τον πρωτεύον ρόλο είναι το κόστος του κάθε επεξεργαστή. Προκειμένου λοιπόν να υπολογίσουμε το κόστος αυτού του αλγορίθμου

θα ακολουθήσουμε μια άλλη τακτική και όχι μια από τις δύο που έχουμε ήδη δει. Η τακτική αυτή λειτουργεί ως εξής:

Έστω ότι το πρόβλημα μας λύνεται σε **β** βήματα, δηλαδή σε β επαναλήψεις του βρόγχου. Σε κάθε επανάληψη υπολογίζουμε τους επεξεργαστές που δεν έχουν καταρρεύσει μέχρι το τέλος της συγκεκριμένης επανάληψης και τέλος αθροίζουμε τους υπολογισμούς που έχουμε κάνει και για τα **β** βήματα. Για παράδειγμα αν **πτ** είναι ο αριθμός των επεξεργαστών οι οποίοι δεν έχουν καταρρεύσει μέχρι το τέλος του βήματος **τ**, τότε το συνολικό κόστος θα είναι:

$$C_p(\mathbf{n}, f) = \sum_{\tau=1}^{\beta} \pi_\tau \quad \textbf{[5].}$$



Σχήμα 8.3 [5]

Άρα προκύπτει ότι το συνολικό κόστος που προκύπτει από την χειρότερη περίπτωση όπου σε κάθε επανάληψη καταρρέει έστω και ένας επεξεργαστής θα είναι της τάξεως του $O(n^2)$.

8.5 Επεξήγηση βέλτιστου αλγορίθμου

Παρατηρούμε όμως πως ο αλγόριθμος αυτός δεν είναι αποδοτικός, το κόστος του δεν είναι της τάξεως του χρόνου του καλύτερου σειριακού αλγορίθμου.

Άραγε μπορούμε να καταφέρουμε πιο μικρό κόστος με το να μειώσουμε κάπως τους επεξεργαστές που χρησιμοποιούμε στο πρόβλημα μας;

Ευτυχώς, αυτό μπορεί να επιτευχθεί εύκολα με το να κάνουμε τις απαραίτητες μετατροπές στην φάση W3 ούτως ώστε αυτή η φάση να καταναλώνει χρόνο ίδιο με τις άλλες φάσεις δηλαδή χρόνο ίσο με O(log n). Αυτό σημαίνει ότι θα χρησιμοποιήσουμε δέντρα με n / log n φύλλα, όπου κάθε φύλλο έχει log n στοιχεία. Επίσης θεωρούμε ότι έχουμε n loglog n/ $\log^2$ n επεξεργαστές και όχι n και έτσι το κόστος θα μειωθεί σε O(loglog n/ $\log^2$ n *n $\log^2$ n /loglog n) = O(n). Η μείωση των επεξεργαστών αν και καθιστά των αλγόριθμο μας κόστου -βέλτιστο επιφέρει και κάποιες αλλαγές στα βήματα του αλγορίθμου. Ενδέχεται λοιπόν οι επεξεργαστές μας να κάνουν και κάποιους σειριακούς υπολογισμούς στα log n στοιχεία που αντιστοιχούν στο φύλλο που ανατίθενται. Επομένως κατά την διάρκεια της φάσης W3 θα εργαστούν σειριακά για να γράψουν την τιμή 1 στα log n στοιχεία. Προφανώς όμως και στις άλλες φάσεις θα έχουμε κάποιες αλλαγές. Κατά τις φάσεις W1 και W4 θα ήταν καλύτερο να εκτελέσουμε τον βέλτιστο αλγόριθμο της παράλληλης άθροισης, όπου αρχικά εκτελεί ένα σειριακό βήμα.

Τελικά με τις τροποποιήσεις αυτές καταφέρνουμε να μειώσουμε το κόστος σε O(n) και να διατηρήσουμε τον χρόνο που είχαμε και στον μη βέλτιστο αλγόριθμο W(O(n log n)). Πιο συγκεκριμένα ο χρόνος διατηρείται λόγω του ότι πάλι θα έχουμε n επαναλήψεις του βρόγχου όπου στην κάθε επανάληψη καταναλώνουμε χρόνο ίσο με log n.

Επομένως ο βέλτιστος αλγόριθμος W για επίλυση του προβλήματος Write-All με n loglog n/ log2 n επεξεργαστές στο μοντέλο F-PRAM είναι ορθός καταναλώνοντας χρόνο ίδιο με τον μη βέλτιστο αλγόριθμο, (O(n log n)) και κόστος ίσο με O (n).

8.6 Πειραματική αξιολόγηση αλγορίθμων και συμπεράσματα

Αλγόριθμος W χωρίς σφάλματα κατάρρευσης

Άραγε όμως αυτό ισχύει και στην πράξη; Οι δύο αυτοί αλγόριθμοι έχουν τον ίδιο χρόνο εκτέλεσης και διαφορετικό κόστος; Και αν δεν έχουν τον ίδιο χρόνο εκτέλεσης η πρακτική τους διαφορά που προκύπτει είναι γραμμική, πολυωνική ή είναι κάποια σταθερά; Ο βέλτιστος αλγόριθμος είναι όντως βέλτιστος και στην πράξη; Υλοποιώντας

λοιπόν τους δύο πιο πάνω αλγορίθμους στο μοντέλο SB-PRAM με την βοήθεια της γλώσσας FORK παίρνουμε κάποιες μετρήσεις σε χρόνο και κόστος. Οι μετρήσεις που ακολουθούν πάρθηκαν εκτελώντας τους αλγόριθμους από 5 φορές τον κάθε ένα. Σε κάθε εκτέλεση πήραμε μια ξεχωριστή μέτρηση και από αυτές τις πέντε μετρήσεις που αντιστοιχούν στον κάθε αλγόριθμο αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει.

N	***P***	***Msec***	***Κόστος***
4	**4**	**3.588**	**4**
8	**8**	**4.868**	**8**
16	**16**	**6.148**	**16**
32	**32**	**7.428**	**32**
64	**64**	**8.708**	**64**
256	**256**	**10.543**	**256**

Πίνακας 8.1

Αυτές οι μετρήσεις πάρθηκαν για τον αλγόριθμο W στην περίπτωση όμως που κανένας επεξεργαστής δεν καταρρέει κατά την διάρκεια του αλγορίθμου. Αυτό σημαίνει ότι δεν εκτελείται ούτε μια φορά ο βρόγχος που περιέχει τις τέσσερις φάσεις. Επομένως εκτελούνται μόνο οι φάσεις W3 και W4 που βρίσκονται εκτός του βρόγχου.

Από τις παραπάνω μετρήσεις παρατηρούμε ότι πράγματι με την αύξηση των στοιχείων στο πρόβλημα μας αυξάνεται και ο χρόνος που χρειάζεται ο αλγόριθμος για να επιλύσει το πρόβλημα και να τερματίσει. Παρομοίως και το κόστος αυξάνεται με την αύξηση των στοιχείων, ενώ παρατηρούμε να παίρνει τον ίδιο αριθμό με το πλήθος των στοιχείων του προβλήματος. Αυτό πάλι μπορούμε να το δικαιολογήσουμε γιατί όπως έχουμε πει ο βρόγχος δεν εκτελείται καμία φορά. Η παρατήρηση αυτή μας οδηγεί στο να επιβεβαιώσουμε το ότι στην εξίσωση για χρόνο και κόστος που είχαμε κατά την θεωρητική αξιολόγηση συμπεριλάβαμε μέσα σ' αυτή και το πλήθος n των στοιχείων.

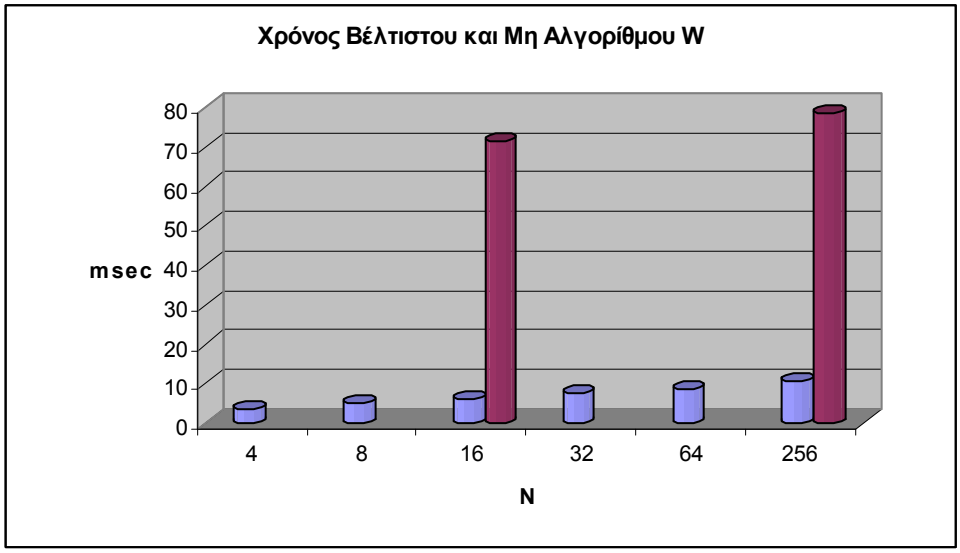
Αλγόριθμος W βέλτιστος

N	*P*	*msec*	*Κόστος*
16	**4**	**45.036**	**4**
16	**2**	**71.548**	**2**
256	**12**	**78.748**	**12**

Πίνακας 8.2

Από τις μετρήσεις που πήραμε στο βέλτιστο αλγόριθμο παρατηρούμε πως πάλι έχουμε μια αύξηση στον χρόνο και στο κόστος καθώς τα στοιχεία αυξάνονται.

8.7 Παρουσίαση γραφικών και σύγκριση των δύο αλγορίθμων

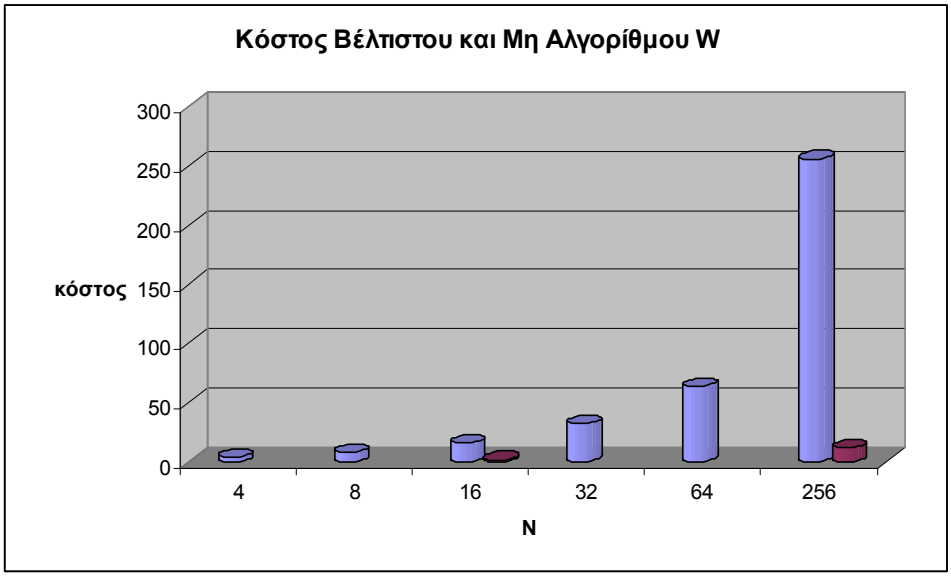


Διεκρύνηση: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με **n loglog n/ $\log^2$ n** *και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.*

Γραφική 8.1

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο αλγόριθμος W είναι πολύ γρηγορότερος από τον βέλτιστο. Αυτή η παρατήρηση μας οδηγεί στο ακόλουθο συμπέρασμα: Δεν ισχύει ότι και οι δύο αλγόριθμοι έχουν τον ίδιο ασυμπτωτικό χρόνο εκτέλεσης όπως έχουμε δει και στην θεωρητική αξιολόγηση των χρόνων των αλγορίθμων. Στην πραγματικότητα αυτό είναι δικαιολογημένο γιατί στον βέλτιστο

αλγόριθμο όπως έχουμε αναφέρει οι επεξεργαστές εκτελούν και κάποια σειριακά βήματα τα οποία δεν υπολογίζονται στην θεωρητική αξιολόγηση αφού υπερισχύει ο χρόνος O(n log n). Επίσης κάτι επιπλέον που συμβάλει στην αύξηση του χρόνου στον βέλτιστο αλγόριθμο είναι ότι χρησιμοποιούμε τον βέλτιστο αλγόριθμο παράλληλης άθροισης ο οποίος όπως έχουμε δει από τις μετρήσεις που πήραμε κατά την εκτέλεση του, καταναλώνει περισσότερο χρόνο από τον μη βέλτιστο αλγόριθμο παράλληλης άθροισης.



<u>Διεκρύνηση</u>: οι μετρήσεις για τον βέλτιστο αλγόριθμο που απεικονίζονται σε αυτή την γραφική παράσταση είναι μόνο για δύο αριθμούς στοιχείων για n=16 και n=256 και αυτό συμβαίνει γιατί αναγκαία συνθήκη για να είναι ο αλγόριθμος μας βέλτιστος είναι ότι οι επεξεργαστές πρέπει να είναι ίσοι με **n loglog n/ $\log^2$ n** *και μόνο με αυτό το πλήθος των στοιχείων μπορούμε να έχουμε ακέραιο αριθμό επεξεργαστών.*

Γραφική 8.2

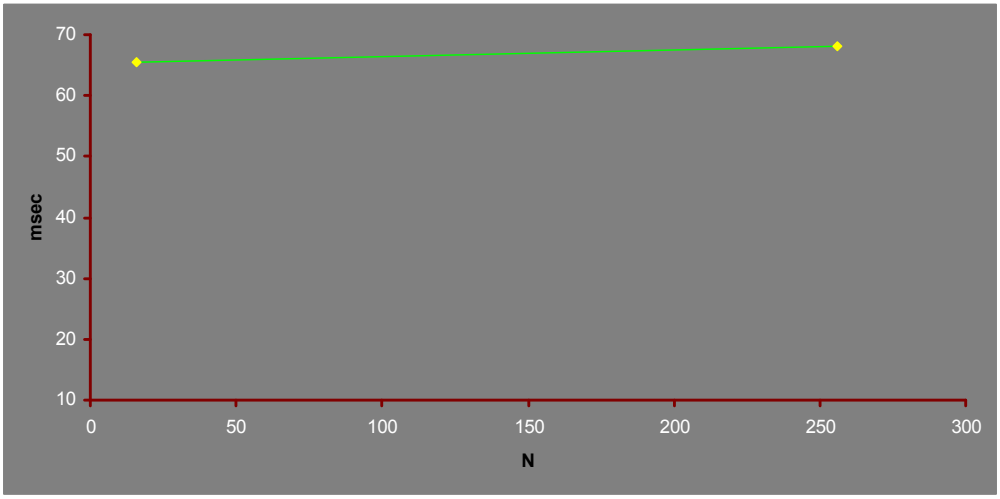
Από την παραπάνω γραφική παράσταση παρατηρούμε ότι ο βέλτιστος αλγόριθμος W έχει πολύ πιο μικρό κόστος από τον μη βέλτιστο αλγόριθμο. Πράγματι αυτό ισχύει και στην θεωρητική αξιολόγηση του κόστους για τους δύο αυτούς αλγορίθμους. Καταφέραμε λοιπόν να μειώσουμε το κόστος του βέλτιστου αλγορίθμου αν και ο χρόνος του ήταν μεγαλύτερος. Αυτή η μείωση του κόστους λοιπόν οφείλεται στο ότι χρησιμοποιήσαμε λιγότερους επεξεργαστές από το πλήθος των στοιχείων και πιο συγκεκριμένα **n loglog n/ $\log^2$ n** επεξεργαστές.

Συνοψίζοντας έχουμε δει ότι όντως ο βέλτιστος αλγόριθμος που επιλύει το πρόβλημα Write-All στο μοντέλο F-PRAM (αλγόριθμος W) έχει μικρότερο κόστος εκτέλεσης αλλά καταναλώνει πολύ περισσότερο χρόνο. Επομένως σίγουρα το ότι καταφέραμε και μειώσαμε το κόστος είναι πολύ σημαντικό. Τι γίνεται όμως αν καθώς αυξάνονται τα στοιχεία αυξάνεται κατά πολύ ο χρόνος του βέλτιστου και η διαφορά που προκύπτει με τον χρόνο του μη βέλτιστου είναι πολύ μεγάλη; Αυτό θα πρέπει να μας προβληματίσει γιατί καλό είναι να έχουμε μικρό κόστος αλλά και ο χρόνος παίζει ένα ουσιαστικό ρόλο. Αυτό λοιπόν μπορούμε να το δούμε υπολογίζοντας την πρακτική διαφορά που προκύπτει μεταξύ των δύο αλγορίθμων.

.

N	*Χρόνος Μη Βέλτιστου*	*Χρόνος Βέλτιστου*	*Διαφορά*
16	**6.148**	**71.548**	**65.4**
256	**10.543**	**78.748**	**68.205**

Πίνακας 8.3



Γραφική 8.3

Από αυτή την γραφική λοιπόν παρατηρούμε ότι η διαφορά στο χρόνο του μη βέλτιστου από τον βέλτιστο αλγόριθμο που επιλύουν το πρόβλημα Write-All χωρίς κανένα σφάλμα κατάρρευσης, που προκύπτει παραμένει σχεδόν σταθερή όσο αυξάνεται το πλήθος των στοιχείων. Επομένως συμπεραίνουμε ότι αν και ο βέλτιστος αλγόριθμος

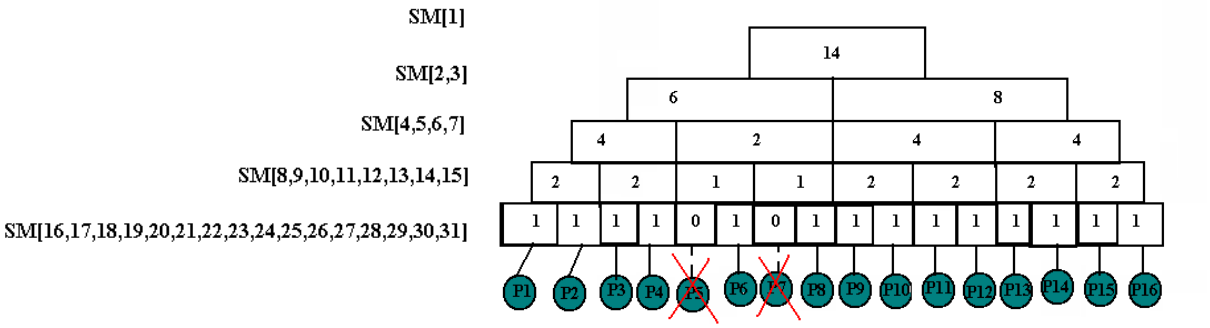
καταναλώνει περισσότερο χρόνο στην εκτέλεση του από τον μη βέλτιστο η διαφορά του χρόνου είναι σταθερή. Επομένως ο βέλτιστος αλγόριθμος είναι όντως καλύτερος από τον μη βέλτιστο αλγόριθμο γιατί καταφέραμε να μειώσουμε και το κόστος αλλά και ο χρόνος που προκύπτει θα έχει πάντα μια σταθερά για διαφορά με τον μη βέλτιστο καθώς αυξάνονται τα στοιχεία.

8.8 Στοχευμένα σενάρια εκτέλεσης του αλγορίθμου W

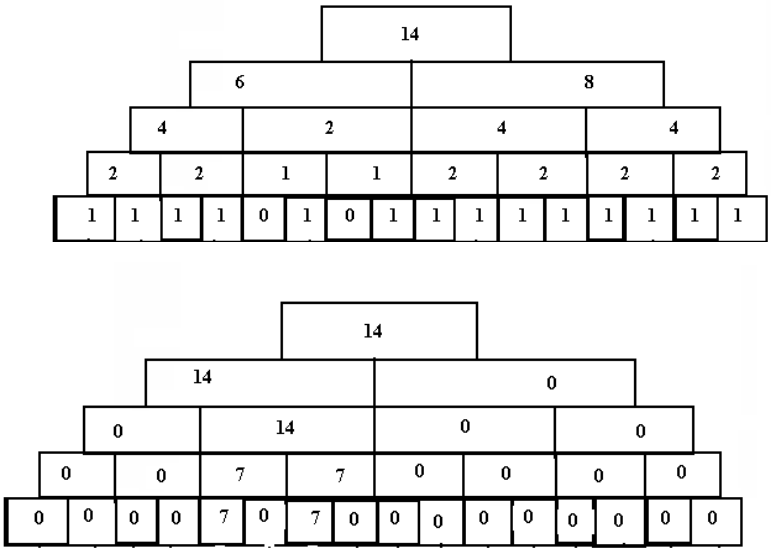
Παρότι δείξαμε ότι ο αλγόριθμος μας είναι ορθός σε περίπτωση που δεν έχουμε σφάλματα κατάρρευσης, θα ήταν καλό να τον δοκιμάσουμε και σε κάποια επιλεγμένα σενάρια στα οποία θα έχουμε κάποιους επεξεργαστές να καταρρέουν. Τα σενάρια που επιλέχθηκαν για αυτό τον σκοπό είναι τα εξής:

8.7.1 Σενάριο 1

Σε αυτό το σενάριο έχουμε μια πολύ απλή περίπτωση όπου προσπαθούμε να δούμε τι θα συμβεί αν κάποιοι συγκεκριμένοι επεξεργαστές καταρρεύσουν πριν γράψουν την τιμή 1 στα φύλλα του δέντρου. Φυσικά αυτό που αναμένετε είναι να μην καταναλώνεται και πολύ μεγάλος χρόνος αφού κατά την δεύτερη επανάληψη οι επεξεργαστές που παραμείνανε ενεργοί θα έρθουν να διορθώσουν το πρόβλημα. Τώρα όσον αφορά το κόστος αυτό εξαρτάτε από το πόσοι επεξεργαστές δεν έχουν καταρρεύσει την στιγμή πριν γράψουν την τιμή 1 στα φύλλα του δέντρου. Έχουμε λοιπόν ένα δέντρο προόδου με 16 φύλλα όπου είναι αποθηκευμένο στο διάνυσμα Δ2[1,…31] της κοινόχρηστης μνήμης. Θεωρούμε ότι οι επεξεργαστές 5 και 7 καθώς βρίσκονται στην φάση W3 καταρρέουν πριν γράψουν την τιμή 1 στα φύλλα που τους αντιστοιχούν (δηλαδή στις κυψελίδες με αριθμό 20 και 22).
Εκτελώντας λοιπόν αυτό το σενάριο 5 φορές παίρνουμε μια ξεχωριστή μέτρηση σε κάθε εκτέλεση. Από αυτές λοιπόν τις πέντε μετρήσεις αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει. Έτσι παίρνουμε τον εξής χρόνο και κόστος με είσοδο 16 στοιχείων και 16 επεξεργαστών.
Χρόνος: 30.715 msec
Κόστος: 28



Σχήμα 8.4 Αποτέλεσμα μετά από την φάση W3 και W4.



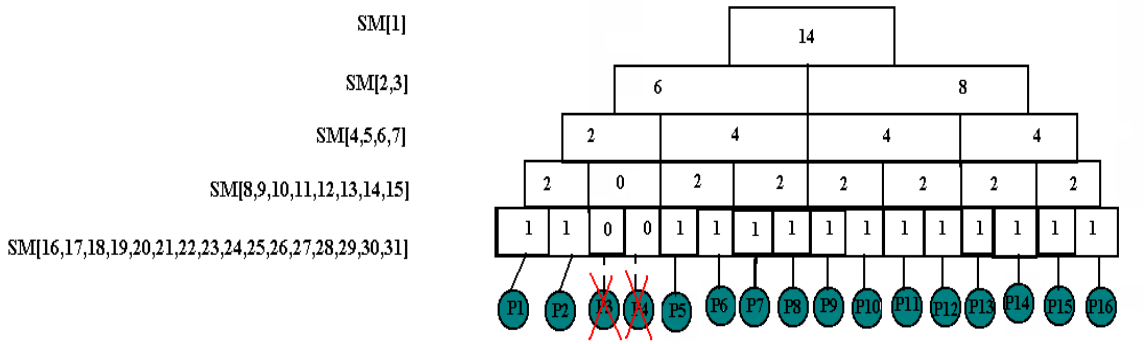
Σχήμα 8.5 Αποτέλεσμα μετά από την φάση W2 και ανάθεση επεξεργαστών στα φύλλα.

Από την περιγραφή του αλγορίθμου είδαμε ότι στον αλγόριθμο W εκτελείται ο αλγόριθμος παράλληλης άθροισης κατά τις φάσεις W1 και W4. Επομένως θα ήταν καλό να υπολογίσουμε το πόσες φορές καλείται ο αλγόριθμος της παράλληλης άθροισης και να συγκρίνουμε τον χρόνο που καταναλώνεται για την εκτέλεση του αλγορίθμου W αλλά και του αλγορίθμου της παράλληλης άθροισης. Φυσικά για το σενάριο στο οποίο δεν έχουμε σφάλματα ο αλγόριθμος παράλληλης άθροισης εκτελείται μόνο μια φορά κατά την φάση W4. Στο σενάριο 1 όμως εκτελείται 3 φορές αφού εκτελείται μια φορά στην φάση W4 πριν την εκτέλεση του βρόγχου, και έπειτα αφού το πρόβλημα λύνεται μόνο με μια εκτέλεση του βρόγχου, τότε ο αλγόριθμος της παράλληλης άθροισης εκτελείται ακόμη δύο φορές.

Ο χρόνος που έχουμε πάρει από τον αλγόριθμο της παράλληλης άθροισης για 16 στοιχεία είναι 8.044. Επομένως πολλαπλασιάζοντας τον με τον αριθμό 3 έχουμε αποτέλεσμα ίσο με 24.132. Όπως αναφέρθηκε και παραπάνω ο χρόνος που πήραμε εκτελώντας το σενάριο 1 ήταν 30.715 ένας αριθμός πολύ κοντά με το 24.132. Βεβαίως και είναι όμως μεγαλύτερος αφού στον αλγόριθμο εκτελούνται και άλλες επιπρόσθετες πράξεις.

8.7.2 Σενάριο 2

Σε αυτό το σενάριο έχουμε μια πιο πολύπλοκη περίπτωση όπου προσπαθούμε να δούμε τι θα συμβεί αν κάποιοι συγκεκριμένοι επεξεργαστές καταρρεύσουν πριν γράψουν την τιμή 1 στα φύλλα του δέντρου και κατά την δεύτερη επανάληψη όταν οι επεξεργαστές θα έρθουν να διορθώσουν το πρόβλημα να καταρρεύσουν όλοι εκτός από έναν επεξεργαστή ο οποίος θα αναλάβει να διορθώσει το πρόβλημα. Ανάλογα με το πλήθος των επεξεργαστών που έχουν καταρρεύσει και ανάλογα με το υποδέντρο στο οποίο βρίσκονται αυτοί που έχουν καταρρεύσει, ο επεξεργαστής που θα παραμείνει ενεργός θα εκτελέσει από μια μέχρι και κάποιο αριθμό επαναλήψεων. Επομένως ο χρόνος θα εξαρτηθεί από το πόσες επαναλήψεις θα γίνουν. Τώρα όσον αφορά το κόστος αυτό που γνωρίζουμε σίγουρα είναι ότι θα είναι μικρότερο από το προηγούμενο σενάριο.
Σε αυτό το σενάριο λοιπόν θεωρούμε ότι οι επεξεργαστές 3 και 4 καταρρέουν πριν γράψουν την τιμή 1 στα φύλλα του. Έπειτα κατά την πρώτη επανάληψη του βρόγχου καταρρέουν όλοι οι επεξεργαστές εκτός από τον επεξεργαστή με προσωπικό αριθμό ίσο με 1.
Εκτελώντας λοιπόν αυτό το σενάριο 5 φορές με 16 στοιχεία και 16 επεξεργαστές παίρνουμε μια ξεχωριστή μέτρηση σε κάθε εκτέλεση. Από αυτές λοιπόν τις πέντε μετρήσεις αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει. Έτσι παίρνουμε τον εξής χρόνο και κόστος.
Χρόνος: 39.068
Κόστος: 15



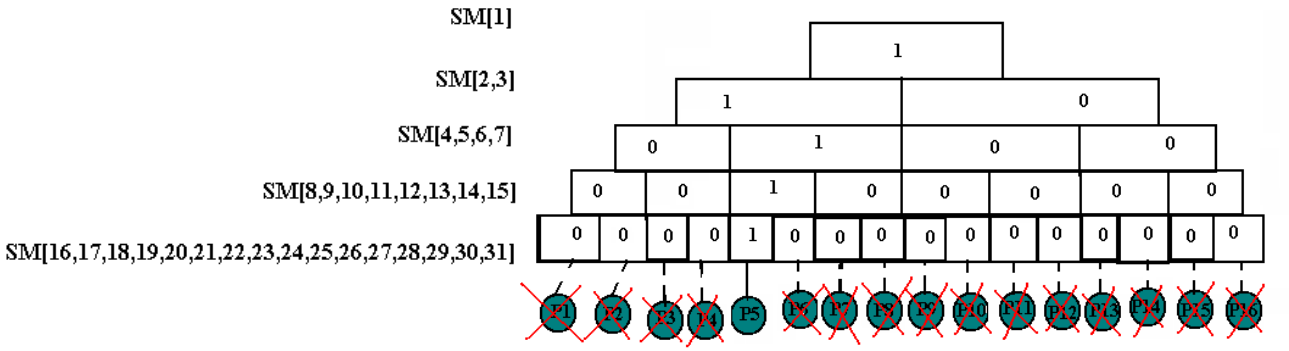
Σχήμα 8.6 Αποτέλεσμα μετά από την φάση W3 και W4.

Λόγω του ότι οι δύο επεξεργαστές που έχουν καταρρεύσει βρίσκονταν στις δύο γειτονικές κυψελίδες, ο επεξεργαστής P1 θα εκτελέσει μόνο μια επανάληψη και θα επιλύσει το πρόβλημα. Επομένως ο χρόνος δικαίως και είναι πολύ κοντά με αυτόν του προηγούμενου σεναρίου. Όσο αφορά το κόστος αναμέναμε να είναι πολύ μικρο.

8.7.3 Σενάριο 3

Σε αυτό το σενάριο εξετάζουμε μια ακραία περίπτωση όπου μόνο ένας επεξεργαστής είναι από την αρχή μέχρι το τέλος ενεργός. Αναμένουμε λοιπόν μεγάλο χρόνο γιατί σίγουρα θα έχουμε μεγάλο πλήθος επαναλήψεων. Όσον αφορά το κόστος όμως θα εξαρτηθεί μόνο από το πλήθος των επαναλήψεων.

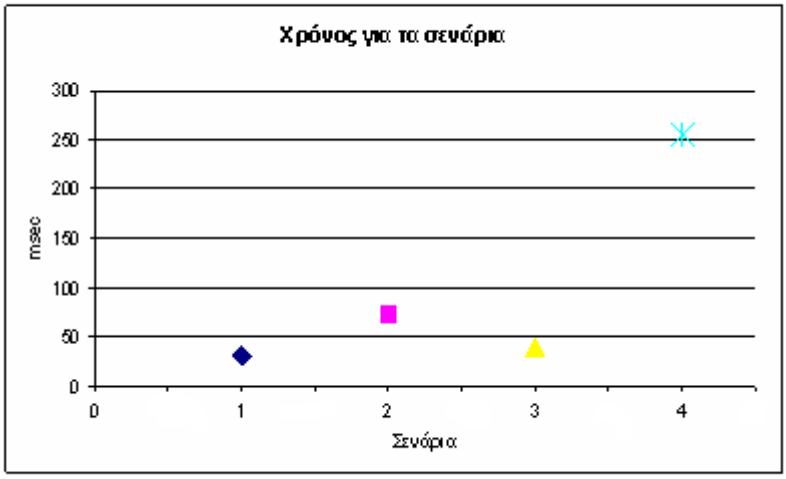
Σε αυτό το σενάριο θεωρούμε ότι όλοι οι επεξεργαστές εκτός από τον επεξεργαστή με προσωπικό αριθμό 5 καταρρέουν πριν γράψουν την τιμή 1 στο φύλλο που τους αντιστοιχεί. Εκτελώντας λοιπόν αυτό το σενάριο 5 φορές με 16 στοιχεία και 16 επεξεργαστές παίρνουμε μια ξεχωριστή μέτρηση σε κάθε εκτέλεση. Από αυτές λοιπόν τις πέντε μετρήσεις αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει. Έτσι παίρνουμε τον εξής χρόνο και κόστος.
Χρόνος: 255.32
Κόστος: 16



Σχήμα 8.7 Αποτέλεσμα μετά από την φάση W3 και W4.

Από τις παραπάνω μετρήσεις παρατηρήσαμε ότι ο χρόνος είναι πάρα πολύ μεγάλος όπου αυτό και αναμέναμε αφού εκτελούνται πάρα πολλές επαναλήψεις από έναν και μόνο επεξεργαστή. Για το κόστος τώρα παρατηρούμε ότι ο επεξεργαστής P1 εκτελεί 15 επαναλήψεις του βρόγχου και επομένως καταναλώνει συνολικό κόστος ίσο με 16.

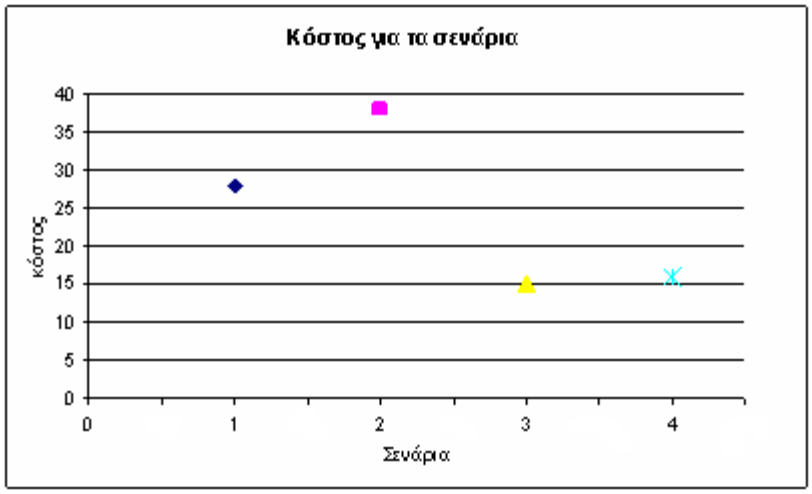
8.7.4 Σύγκριση των διαφόρων σεναρίων



Γραφική 8.3

Από την παραπάνω γραφική παράσταση παρατηρούμε ότι για τα σενάρια 1 και 3 καταναλώνουμε περίπου τον ίδιο χρόνο και αυτό συμβαίνει δικαιολογημένα λόγω του ότι καταρρέουν αρχικά δύο επεξεργαστές ενώ οι υπόλοιποι είναι υπεύθυνοι να ολοκληρώσουν τον αλγόριθμο. Στο σενάριο 3 αν και καταρρέουν κατά την πρώτη επανάληψη όλοι οι επεξεργαστές εκτός από ένα αυτός καταφέρνει να γράψει στις

κυψελίδες μνήμης που χρειάζεται εκτελώντας μόνο μια φορά τον βρόγχο. Δικαιολογημένα επίσης στο σενάριο 4 καταναλώνεται ο περισσότερος χρόνος αφού μόνο ένας επεξεργαστής είναι υπεύθυνος για να λύσει ολόκληρο το πρόβλημα.



Γραφική 8.4

Από την παραπάνω γραφική παράσταση του κόστους παρατηρούμε ότι στο σενάριο 2 καταναλώνουμε περισσότερο κόστος. Αυτό μπορούμε να το εξηγήσουμε από το ότι ο αλγόριθμος αυτός για να ολοκληρωθεί κάνει πολλές επαναλήψεις του βρόγχου όπου σε κάθε επανάληψη υπάρχουν πολλοί ενεργοί επεξεργαστές. Επίσης σε αυτή την γραφική βλέπουμε ότι τώρα τα σενάρια 1 και 3 δεν έχουν το ίδιο κόστος αν και όπως έχουμε δει και από την **Γραφική 8.3** είχαν τον ίδιο χρόνο εκτέλεσης. Αυτό συμβαίνει γιατί στο σενάριο 1 δεν καταρρέουν τόσοι πολλοί επεξεργαστές που καταρρέουν κατά την διάρκεια του σεναρίου 3 και επομένως υπολογίζεται και το κόστος όλων των ενεργών επεξεργαστών.

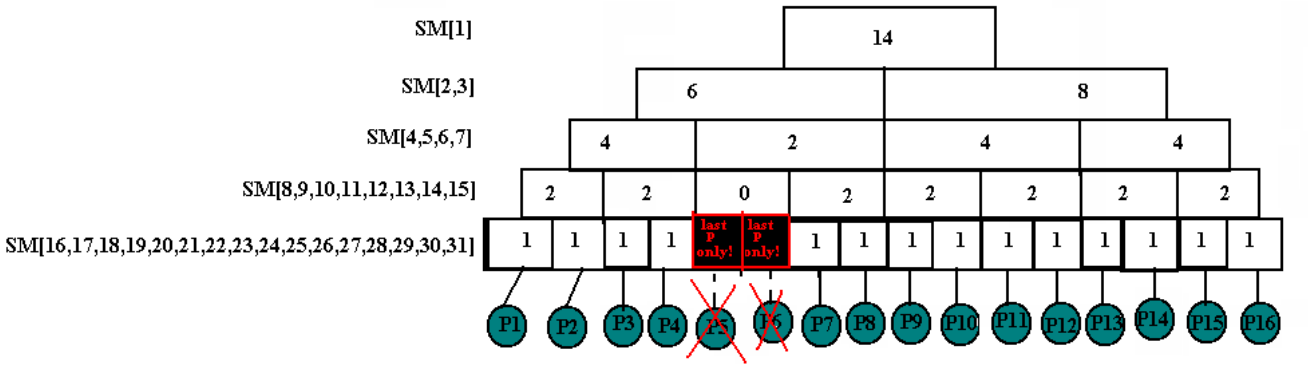
8.7.5 Πιο ρεαλιστικά σενάρια

Τα παραπάνω σενάρια που μελετήθηκαν είναι ενδιαφέροντα σενάρια απλά αφορούν την κατάρρευση συγκεκριμένων επεξεργαστών που αυτό είναι κάπως δύσκολο να εμφανιστεί σε πραγματικά δεδομένα. Μπορούμε εύκολα να θεωρήσουμε κάποια πιο γενικά σενάρια που να μην αφορούν συγκεκριμένα κάποιους επεξεργαστές έτσι ώστε

να βγάλουμε συμπεράσματα που θα βοηθήσουν στην επίλυση προβλημάτων παραλληλισμού.

8.7.5.1 Σενάριο 5

Σε αυτό το σενάριο θεωρούμε ότι στις κυψελίδες της SM[20] και SM[21] της κοινόχρηστης μνήμης δημιουργήθηκε κάποιο πρόβλημα με αποτέλεσμα όποιος επεξεργαστής καταλήξει στην κυψελίδα αυτή να καταρρέει. Για σκοπούς ολοκλήρωσης του αλγορίθμου ο τελευταίος επεξεργαστής που καταλήγει για σε αυτές τις κυψελίδες δεν καταρρέει. Αυτό είναι ένα συχνό φαινόμενο και στην πραγματικότητα.
Εκτελώντας λοιπόν αυτό το σενάριο 5 φορές με 16 στοιχεία και 16 επεξεργαστές παίρνουμε μια ξεχωριστή μέτρηση σε κάθε εκτέλεση. Από αυτές λοιπόν τις πέντε μετρήσεις αφαιρέθηκαν αυτές με την μεγαλύτερη και μικρότερη τιμή. Τέλος η τελική τιμή προέκυψε από τον μέσο όρο των τριών μετρήσεων που είχαν απομείνει. Έτσι παίρνουμε τον εξής χρόνο και κόστος:
Χρόνος: 49.894
Κόστος: 15



Σχήμα 8.8 Αποτέλεσμα μετά από την φάση W3 και W4.

Η εκτέλεση αυτού του σεναρίου είναι παρόμοια με το σενάριο 2 όπου αρχικά καταρρέουν οι επεξεργαστές P3 και P4 και στην συνέχεια μόνο ένας επεξεργαστής που παραμένει ενεργός πρέπει να επιλύσει το πρόβλημα. Επομένως και δικαιολογημένα ο χρόνος και το κόστος είναι παρόμοια με το σενάριο αυτό.

8.7.5.2 Σενάριο 6

Σε αυτό το σενάριο θεωρούμε ότι ο κάθε επεξεργαστής κάθε βήμα του αλγορίθμου έχει μια ξεχωριστή πιθανότητα η οποία επιλέγεται τυχαία. Επομένως έχοντας μια κοινή μεταβλητή η οποία εκφράζει και αυτή πιθανότητα, ελέγχουμε αν η πιθανότητα που έχει ο επεξεργαστής στο συγκεκριμένο βήμα είναι μεγαλύτερη από την μεταβλητή αυτή. Αν ναι τότε ο επεξεργαστής θα καταρρεύσει. Αν όχι θα συνεχίσει. Πάλι όμως για σκοπούς ολοκλήρωσης του αλγορίθμου ένας επεξεργαστής δεν καταρρέει ποτέ.
Φυσικά γνωρίζουμε ότι με διαφορετικές εκτελέσεις αυτού του αλγορίθμου θα έχουμε διάφορα αποτελέσματα
Σε μια από τις εκτελέσεις του αλγορίθμου αυτού λοιπόν με 16 στοιχεία και 16 επεξεργαστές έχουμε να καταρρέουν οι 10 επεξεργαστές πριν να γράψουν στα φύλλα του δέντρου και οι 5 κατά την πρώτη επανάληψη παραμένοντας ένας να ολοκληρώσει όλο τον αλγόριθμο. Σε αυτό το σενάριο υπάρχει όμως μια διαφορά στον αριθμό εκτέλεσης του από τα προηγούμενα σενάρια. Εδώ λοιπόν εκτελέσαμε το σενάριο μόνο μια φορά και όχι 5 φορές παίρνοντας τον μέσο όρο τον τριών μη ακραίων τιμών. Αυτό συμβαίνει γιατί σε κάθε εκτέλεση θα είχαμε διαφορετικούς επεξεργαστές να καταρρέουν και επομένως ο χρόνος που θα παίρναμε θα αφορούσε διαφορετικό σενάριο κάθε φορά. Επομένως με την εκτέλεση του αλγορίθμου παίρνουμε τον εξής χρόνο αλλά και κόστος.
Χρόνος: 216.001
Κόστος: 21

Παρατηρούμε ότι ο χρόνος ασφαλώς και είναι μεγάλος γιατί γίνονται πολλές επαναλήψεις του βρόγχου. Επίσης το κόστος που προκύπτει είναι αναλόγως μικρό γιατί καταρρέουν πάρα πολλοί επεξεργαστές.

8.8 Γενικά συμπεράσματα.

Συνοψίζοντας λοιπόν, καταλήγουμε στο συμπέρασμα ότι αν και στην θεωρητική αξιολόγηση ο χρόνος του βέλτιστου αλγορίθμου είναι ασυμπτωτικά ίδιος με τον χρόνο του μη βέλτιστου στην πράξη δεν ισχύει αυτό, αλλά ο χρόνος του βέλτιστου

αλγορίθμου είναι μεγαλύτερος. Όπως όμως έχουμε δει η διαφορά που προκύπτει στο χρόνο είναι μια σταθερά τιμή. Με την αύξηση λοιπόν των στοιχείων αυξάνεται και ο χρόνος αλλά πάντα με μια σταθερά. Αντιθέτως το κόστος σε θεωρητική αλλά και σε πρακτική αξιολόγηση είναι μεγαλύτερο στον μη βέλτιστο αλγόριθμο που επιλύει το πρόβλημα Write-All. Επίσης παρατηρήθηκε ότι όσο περισσότεροι επεξεργαστές καταρρέουν τόσο λιγότερο κόστος έχουμε ενώ με τον χρόνο συμβαίνει το αντίθετο. Επομένως όντως ο βέλτιστος αλγόριθμος W είναι πολύ καλύτερος από τον μη βέλτιστο αλγόριθμο.

Τώρα όσον αφορά τα σενάρια κατάρρευσης παρατηρήθηκε ότι αν οι επεξεργαστές καταρρέουν κατά την διάρκεια της πρώτης επανάληψης το κόστος είναι ποιο μεγάλο από το ότι όταν καταρρέουν κατά την διάρκεια της δεύτερης ή και τρίτης επανάληψης.

Κάτι ακόμα που παρατηρήθηκε είναι ότι όταν οι επεξεργαστές που καταρρέουν βρίσκονται σε διπλανά φύλλα όπου το φύλλο με τον άρτιο αριθμό θέσης έχει μεγαλύτερο αριθμό κυψελίδας από το φύλλο με τον περιττό αριθμό, αυτό καταναλώνει λιγότερο χρόνο και κόστος για να διορθωθεί παρά αν οι επεξεργαστές αυτοί βρίσκονταν σε μη διπλανά φύλλα.

Επίσης από το σενάριο 5 όπου καταρρέουν κάποιες κυψελίδες παρατηρήθηκε ότι ανάλογα με το πλήθος των κυψελίδων που καταρρέουν αυξάνεται και ο χρόνος αλλά το κόστος μειώνεται, λόγω του ότι θα έχουμε πιο λίγους επεξεργαστές οι οποίοι θα προσπαθήσουν να λύσουν ένα πιο μεγάλο πρόβλημα.

Κεφάλαιο 9

Συμπεράσματα

9.1 Τελικά συμπεράσματα

Σε αυτή τη διπλωματική εργασία έγινε κατορθωτή η υλοποίηση κάποιων παράλληλων αλγορίθμων αφού είχε μελετηθεί ο προσομοιωτής SB-PRAM. Έπειτα πάρθηκαν μετρήσει και παρουσιάστηκαν κάποιες γραφικές παραστάσεις για την πιο εύκολη εξαγωγή συμπερασμάτων.

Επομένως συνοψίζοντας τα συμπεράσματα που πάρθηκαν από αυτές τις μετρήσεις καταλήγουμε:

- Στην θεωρία αρχικά είδαμε ότι σε κάθε αλγόριθμο, εκτός από τους αλγορίθμους που επιλύουν το πρόβλημα Write-All στο A-PARM, σημαντικό ρόλο έπαιζε για την ασυμπτωτική ανάλυση του χρόνου το πλήθος των στοιχείων εισόδου. Το οποίο το επιβεβαιώσαμε και κατά την πρακτική υλοποίηση, αφού καθώς αυξάνονταν τα στοιχεία αυξάνονταν και ο χρόνος που χρειάζονταν ο αλγόριθμος για να ολοκληρωθεί επιτυχώς.

- Κατά την πρακτική υλοποίηση όλων των αλγορίθμων, εκτός από τους αλγορίθμους που επιλύουν το πρόβλημα Write-All στο A-PARM, παρατηρήθηκε πως η θεωρητική αξιολόγηση που είδαμε επιβεβαιωνόταν και στην πράξη. Αν και αυτό φαίνεται κάπως παράξενο όσο αφορά τον χρόνο γιατί ο βέλτιστος αλγόριθμος είχε μεγαλύτερο χρόνο από τον αντίστοιχο μη βέλτιστο του ενώ στην θεωρητική αξιολόγηση είχαν τον ίδιο ασυμπτωτικό χρόνο, υπολογίζοντας την πρακτική τους διαφορά παρατηρήσαμε ότι είτε ήταν κάποια

σταθερά που με την αύξηση των στοιχείων παρέμενε η ίδια είτε είχε μια γραμμική σχέση με πολύ μικρή κλίση. Στην περίπτωση λοιπόν του αλγορίθμου της παράλληλης άθροισης και του αλγορίθμου που επιλύει το πρόβλημα Write-All στο F-PRAM (αλγόριθμος W) ή διαφορά που προέκυπτε αφαιρώντας τον χρόνο που παίρναμε από τον μη βέλτιστο από τον αντίστοιχο βέλτιστο παρέμενε σταθερή με την αύξηση των στοιχείων. Στην περίπτωση όμως του αλγορίθμου της παράλληλης μετάδοσης η διαφορά που προέκυπτε ήταν γραμμική με κλίση πολύ κοντά στο μηδέν. Όσο αφορά όμως το κόστος σε θεωρητική και πρακτική αξιολόγηση ήταν πιο μεγάλο στον μη βέλτιστο αλγόριθμο.

- Κατά την πρακτική υλοποίηση όλων των αλγορίθμων που επιλύουν το πρόβλημα Write-All με p επεξεργαστές στο A-PARM αν και θεωρητικά δεν μπορούσαμε να υπολογίσουμε χρόνο, πρακτικά είδαμε ότι ο χρόνος του αλγόριθμου Χ' ήταν μεγαλύτερος από τον χρόνο του αλγόριθμου Χ. Αυτό καταφέραμε να το δικαιολογήσουμε γιατί ο αλγόριθμος Χ' εκτελούσε n/p φορές τον αλγόριθμο Χ. Το σημαντικό όμως ήταν ότι η πρακτική διαφορά του χρόνου που προέκυπτε ήταν μια λογαριθμική συνάρτηση όπου από κάποιο σημείο και μετά παρέμενε σταθερή. Αυτό λοιπόν είναι σημαντικό γιατί για μεγάλη είσοδο αριθμού στοιχείων μπορούμε να χρησιμοποιούμε τον αλγόριθμο Χ' αφού μας δίνει και λιγότερο κόστος.

- Παρατηρήθηκε επίσης ότι στον αλγόριθμο Χ' όσο αυξάνεται το πλήθος των επεξεργαστών για ένα συγκεκριμένο πλήθος στοιχείων, μειώνεται ο χρόνος που καταναλώνει ο αλγόριθμος αλλά αυξάνεται το κόστος του. Επομένως ανάλογα με τις απαιτήσεις που έχουμε θα χρησιμοποιήσουμε και το πλήθος των επεξεργαστών που θέλουμε.

- Ακόμη είδαμε ότι κατά την πρακτική υλοποίηση όλων των αλγορίθμων που επιλύουν το πρόβλημα Write-All στο A-PARM καθώς αυξάνονται τα στοιχεία οι επεξεργαστές γίνονται όλο και πιο ασυγχρόνιστοι ενώ έχοντας μικρό πλήθος στοιχείων οι επεξεργαστές μας λειτουργούν σχεδόν εντελώς συγχρονισμένα.

- Τέλος στον αλγόριθμο W παρατηρήθηκε ότι όσο πιο πολλοί επεξεργαστές καταρρέουν και όσο πιο λίγες επαναλήψεις γίνονται τόσο πιο μικρό είναι το κόστος που έχουμε.

Γενικό συμπέρασμα

Από τα παραπάνω συμπεράσματα αλλά και τις παρατηρήσεις που πήραμε σε κάθε πείραμα είδαμε ότι δεν διαφέρανε και πολύ από αυτά που αναμέναμε αλλά μελετήσαμε και κατά την θεωρία. Δεν υπήρχε κάτι το οποίο να μας οδηγούσε σε αμφιβολία. Συνοψίζοντας λοιπόν πήραμε μια αρκετά εικόνα για την πρακτική απόδοση των αλγορίθμων αυτών.

9.2 Μελλοντική Εργασία

Το τι συμπεράναμε από αυτή την διπλωματική εργασία είναι ότι κατά τη πρακτική αξιολόγηση των αλγορίθμων χρόνου και κόστους ισχύουν τα συμπεράσματα που πήραμε στην θεωρητική αξιολόγηση. Επίσης καταφέραμε να μελετήσουμε σενάρια όπως εκείνα με την κατάρρευση των επεξεργαστών αλλά και με τους επεξεργαστές να λειτουργούν ασυγχρόνιστα που είναι πιο πραγματικά και να εξάγουμε κάποια συμπεράσματα.

Επομένως μια μελλοντική δουλειά που μπορώ να προτείνω είναι να μπορούμε να συνδυάζουμε και άλλα προβλήματα με τα σενάρια αυτά. Παραδείγματος χάριν ο αλγόριθμος παράλληλης άθροισης να υλοποιηθεί στο μοντέλο A-PRAM αλλά και στο μοντέλο F-PRAM και να υπολογιστεί ο χρόνος και το κόστος που χρειάζεται.

Επίσης πρέπει να λάβουμε υπόψη μας ότι στα συμπεράσματα που πήραμε ήταν με βάση την συμπεριφορά των αλγορίθμων στο SB-PRAM.Τι γίνεται όμως με την προσομοίωση των ίδιων αλγορίθμων σε κάποιο άλλο μοντέλο; Τα αποτελέσματα θα είναι τα ίδια, καλύτερα, ή χειρότερα; Συνεπώς μπορούν να υλοποιηθούν οι ίδιοι αλγόριθμοι σε κάποιο άλλο μοντέλο και να συγκριθούν οι μετρήσεις που πήρα από αυτή την διπλωματική εργασία με αυτές που θα εξαχθούν από την εκτέλεση των αλγορίθμων στο άλλο μοντέλο. Έτσι ώστε να έχουμε καλύτερη και γενικότερη εικόνα για την πρακτική απόδοση των αλγορίθμων αυτών.

9.3 Κέρδος που αποκόμισα από την διπλωματική αυτή εργασία

Μέσα στα πλαίσια της ολοκλήρωσης της διπλωματικής μου εργασίας κατάφερα να αποκομίσω κάποια πολύ σημαντικά οφέλη τα οποία θα μου φανούν σίγουρα χρήσιμα στην μετέπειτα μου πορεία. Έμαθα λοιπόν την σωστή μεθοδολογία για την εκπόνηση μιας εργασίας ξεκινώντας από το ερευνητικό επίπεδο. Επίσης κάτι πολύ σημαντικό είναι ότι εμπλούτισα τις γνώσεις μου σχετικά με τον προσωμοιοτή SB-PRAM αλλά και το μοντέλο PRAM. Ακόμη βελτίωσα αρκετά τις προγραμματιστικές μου ικανότητες και έμαθα μια καινούργια για εμένα γλώσσα προγραμματισμού, την γλώσσα FORK όπου με την βοήθεια της κατάφερα να υλοποιήσω και να εξοικειωθώ με τους παράλληλους αλγορίθμους. Κλείνοντας λοιπόν κατάφερα να κατανοήσω αρκετά την σημαντικότητα του παράλληλου υπολογισμού στην ζωή μας, ο οποίος μας προσφέρει πολλά πλεονεκτήματα και θα μας απασχολήσει πολύ και κατά τα επόμενα χρόνια.
Το μέλλον λοιπόν είναι πλέον παράλληλο.

9.4 Προβλήματα και παραδοχές

Μέσα στα πλαίσια της ολοκλήρωσης αυτής της εργασίας και κατά την διάρκεια της πειραματικής αξιολόγησης των αλγορίθμων που υλοποιήθηκαν αντιμετώπισα ένα πρόβλημα σχετικά με τον αριθμό των επεξεργαστών. Δυστυχώς υπήρξε ο περιορισμός στο ότι το πλήθος των επεξεργαστών δεν θα έπρεπε να υπερβαίνει τους 128. Σίγουρα πιο ρεαλιστικά αποτελέσματα θα μπορούσαμε να πάρουμε εάν χρησιμοποιούσαμε πιο πολλούς επεξεργαστές. Αυτό το πρόβλημα ίσως να οφείλεται στο ότι οι επεξεργαστές που χρησιμοποιήθηκαν ήταν εικονικοί και όχι πραγματικοί λόγω του ότι χρησιμοποιήσαμε τον προσομοιωτή της μηχανής SB-PRAM και όχι την ίδια την μηχανή.

Βιβλιογραφία

[1] J.Keller, C.W. Kessler and J. L. Traff, Practical PRAM Programming, Wiley Interscience, New York, 2001.

[2] Paris C. Kanellakis and Alex A. Shvartsman, Fault-tolerant Parallel Computation, Kluwer Academics, 1997.

[3] Christoph W. Kessler, The PRAM Programming Language Fork http://www.ida.liu.se/~chrke/fork95.html, last updated 11 Jul 2005.

[4] Joseph Jaja, An Introduction to Parallel Algorithms, Addison-Wesley, 1992.

[5] Χρύσης Γεωργίου , Σημειώσεις Μαθήματος ΕΠΛ 431 – Σύνθεση Παράλληλων αλγορίθμων Τμήμα Πληροφορικής Πανεπιστήμιο Κύπρου, http://www2.cs.ucy.ac.cy/~chryssis/EPL431/.

[6] Pavel Tvrdik, PRAM models,http://pages.cs.wisc.edu/~tvrdik/2/html/Section2.html, 1999.

[7] College of Information Sciences and Technology at Penn State University, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.519, 2007.

[8] Τεχνικές Παράλληλου Προγραμματισμού http://www.it.uom.gr/project/parallel/.

Παράρτημα Α

Κώδικες από αλγορίθμους που έχουν υλοποιηθεί

Α.1 Αλγόριθμος παράλληλης άθροισης και βέλτιστος

```
/*
================================================================================
*/
/* parralliliAthrisi.c                                          */
/*                                                              */
/*Algorithmos    Parallilis Athroisis                                    */
/*kai           Beltistos Algorithmos Parallilis Athroisis              */
/*                                                              */
/*Algorihmos poy Dedomenou mias listas N arithmon a1,a2,a3,...an          */
/* ypologizei to athroisma ayton ton aritmon                            */
/*                                                              */
/*                                                              */
/* Panagiota Nicoloaou                                          */
/*                                                              */
/*
================================================================================
*/


        #include <fork.h>
        #include <io.h>
        #include <math.h>

        sh int n;// to plithos ton stoixeion pou tha athristoun
        sh int table[1000000];// pinakas pou apothikevontai mesa oi arithmoi
        sh int t[1000000];
        sh int starttime,stoptime;


// sinartisi pou einai ipefthini gia na ipologisei to athrisma ton n arithmon
        sync void psum()
        {
         pr int j=2;
         pr int a=0;
         pr int b=0;
         pr int p=0;
         pr int sum=0;
         pr int k=1;

// Otan ta stoixeia einai perrissotera apo toys epeksergastes
// tote o kathe epeksergastis tha parei n/p stoixeia
//Ayto to kommati kodika xrisimopoiitai gia ton veltisto algorithmo
//Seiriako Bima   ::
   if(n/2>((__STARTED_PROCS__-1)) && $!=0)
            {

     p=(n/(__STARTED_PROCS__-1));
              p=$*p;
           while(k<=(n/(__STARTED_PROCS__-1)))
           {
            sum=sum+table[p];
```

```
            t[$]=sum;
              p--;
              k++;
            }
              n=(__STARTED_PROCS__-1);


// Ypologismos toy athrismatos ston neo pinaka
// Parallilo Bima::
          if($<=n/2)
              {
               while($<=(n/(j-1)))
               {
       a=t[(2*$)-1];
                b=t[2*$];
                t[$]=a+b;
                j=2*j;
               }
               table[1]=t[1];
              }
            }

//Otan oi epeksergastes einai perissoteroi apo to miso ton stoixeion
//tote kapoii epeksergasts den tha xriastei na kanoyn kamia diergasia
//h Otan oi epeksergastes einai akrivos to miso ton stoixeion
   if( (n/2)<=(__STARTED_PROCS__-1) && $!=0)
            {
             if($<=n/2)
              {
               while($<=(n/(j-1)))
                {
         a=table[(2*$)-1];
                  b=table[2*$];
                  table[$]=a+b;
                  j=2*j;
                 }
               }
            }
 }

 async int main(void){
          FILE *fp;
          int i=1;
          int tempStoixeion=0;



//Xrisi arxeiou gia na diavastoun ta stoixeia sta opoia ha ginei h athrisi
          if($==0)
           {
            fp=fopen("input.txt","r");
            if(fp==NULL){
            pprintf("ERROR The file does not exist\n");
            exit(-1);
           }

//To proto stoixeio toy arxeiou einai to plithos ton arithmon
          fscanf(fp, "%d", &n);
```

```
            while(i<=n)
            {
   fscanf(fp, "%d", & table[i]);
    i++;
            }
            }
tempStoixeion=n;

// Kaleite h synartisi pou ypologizei to athrisma
// ypologismos xronou pou xreiazetai i sinartisi gia na diekpereothei
 start
            {
            initTracing(100000);
            startTracing();
            starttime=getct();

 psum();

 stoptime=getct();
            stopTracing();
            writeTraceFile("sum.trv","sum");
             // /

             }
             barrier;

  if($==0)
             {
             printf("\n\n");
             printf("_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_\n");
             printf("To athroisma ton %d stoixeion einai :: %d\n",tempStoixeion,table[1]);
  printf("_*_*_*_*_*_*_*__*_*_*_*_*__*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_\n\n");

  printf("CPU Cycles      :: %d\n",stoptime-starttime);
  printf("Xronos (se msec):: %d\n",(stoptime-starttime)*4);
  printf("Kostos          :: %d\n",((stoptime-starttime)*4)*(__STARTED_PROCS__-1));
             printf("============================================================\n");
              printf("\n\n");
  }
            barrier;
             return 0;
}
```

A.2 Αλγόριθμος παράλληλης μετάδοσης και βέλτιστος

```
/*
================================================================================
*/
/* broadcasting.c                                                 */
/*Algorithmos    Metadosis                                        */
/*kai            Beltistos Algorithmos Metadosis                  */
/*                                                                */
/* Algorihmos poy Dedomenou enos stoixeiou pou einai apothikemeno stin        */
/* kipselida 1 tis koinoxristi mnimis (pinakas) antigrafetai se               */
/* n kipselides tis koinoxristis mnimis                           */
/*                                                                */
/* Panagiota Nicoloaou                                            */
/*
*/
/*
================================================================================
*/


#include <fork.h>
#include <io.h>
#include <math.h>
#include <syscall.h>


sh int n; //to plithos ton kipselidon pou theloume na metadosoume to stoixeio
sh int table[100000]; //pinakas ston opoio metadidoyme to stoixeio
sh int starttime,stoptime;
FILE *fp;

async int main(void){
pr int i=1;
pr int a=0;
pr int b=0;
pr int counter=0;
pr int thesi;


 if($==0)
  {

pprintf(">>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>><<<<<<<<<<<<>>>>>>>
>>");
  pprintf("Dose to plithos ton kipselidon pou theleis na metadothei to stoixeio:\n");
  scanf("%d",&n);
  }

  table[1]=1;//I proti thesi tou pinaka arxikopoiite me to stoixeio to opoio theloume na metadosoume

  barrier;



// ypologismos xronou pou pou katanalonetai gia na oloklirothei i metadosei tou stoixeiou


 /*  if(n/2>(__STARTED_PROCS__-1))
 {
```

```
            while((2^b)<=n || table[n]!=table[1] && counter<(log10(n/2)/log10(2)))
   {
     if($<=(2^b))
      {
   table[$+(2^b)]=table[$];
       }
   b=b+1;
   counter=counter+1;
  }

  thesi=$*(log10(n)/log10(2))-log10(n)/log10(2)+(2^b)+1;
  while(thesi<(log10(n)/log10(2)))
  {
  table[thesi]=table[$];
  thesi=thesi+1;
  }
  }
 */
  // ypologismos xronou pou pou katanalonetai gia na oloklirothei i metadosei tou stoixeiou
 start{

  initTracing(100000);
  startTracing();
  starttime=getct();

  if($!=0)
   {
 // Otan ta stoixeia einai perrissotera apo toys epeksergastes
// tote o kathe epeksergastis tha metadidei kanonika to stoixeio
//mexri to teleftaio bima poy tha metadosei to stoixeio se n/p theseis pou apominan
//Ayto to kommati kodika xrisimopoiitai gia ton veltisto algorithmo
//Parallilo Bima   ::

  if(n/2>(__STARTED_PROCS__-1))
 {
            while((2^b)<=n || table[n]!=table[1] && counter<n/(__STARTED_PROCS__-1))
   {
     if($<=(2^b))
      {
   table[$+(2^b)]=table[$];
       }
   b=b+1;
   counter=counter+1;
  }


  //Ipologismos tis thesis gia na metadosei o kathe epeksergastis ta n/p stoixeia
  //Seiriako Bima::
  thesi=($*(n/(__STARTED_PROCS__-1)))-n/(__STARTED_PROCS__-1)+(2^b)+1;
  while(thesi<n/(__STARTED_PROCS__-1))
  {
  table[thesi]=table[$];
  thesi=thesi+1;
  }
  }


 //Otan oi epeksergastes einai perissoteroi i akrivos  to miso ton kipselidon
 //pou theloume na metadothei to stoixeio.
```

```
 //Stin periptosi pou inai perissoteroi apo to miso ton kipselidon
//tote kapoii epeksergasts den tha xriastei na kanoyn kamia ergasia
 if(n/2<=(__STARTED_PROCS__-1))
 {

 if($<=n/2)
{
  while((2^b)<=n || table[n]!=table[1])
    {
     if($<=(2^b))
     {
     table[$+(2^b)]=table[$];
     }
      b=b+1;
     }
  }
  }
}
stoptime=getct();
stopTracing();
writeTraceFile("sum.trv","sum");
  }

 barrier;
 if($==0)
              {
              //Xrisi arxeiou gia na apothikevsoume se pies kipselides exei metadothei to stoixeio
    fp=fopen("input6.txt","w");
     if(fp==NULL)
     {
               pprintf("ERROR The file does not exist\n");
               exit(-1);
               }



for(i=1;i<=n;i++)
  {
   fprintf(fp,"Stin thesi %d exei metadothei to stoixeio %d\n",i,table[i]);
  }

   fclose(fp);

   printf("\n\n");
              printf("CPU Cycles      :: %d\n",stoptime-starttime);
   printf("Xronos (se msec):: %d\n",(stoptime-starttime)*4);
   printf("Kostos          :: %d\n",((stoptime-starttime)*4)*(__STARTED_PROCS__-1));
              printf("============================================================\n");
              printf("\n\n");
  }

 barrier;
 return 0;

 }
```

A.3 Αλγόριθμος Write-All με δύο επεξεργαστές

```
/*
================================================================================
*/
/*   Write_all_2p.c                                                  */
/*                                                                   */
/*   Algorihmos poy Dedomenou mias listas N stoixeion pou exoyn tin timi 0,   */
/*   metatrepei tis times ton stoixeion se 1 xrisimopoiontas mono 2           */
/*   epeksergastes oi opoioi litoyrgoyn entelos asigxronista.                */
/*                                                                   */
/*     Panagiota Nicoloaou                                           */
/*
*/
/*
================================================================================
*/

#include <fork.h>
#include <io.h>

// metavlites oi opoies einai koines gia oloys toys epeksergastes
sh int N;
sh int table[100000];
sh int stoptime,starttime;
sh int stoptimeP2,starttimeP2;
sh int kostosP1=0;
sh int kostosP2=0;
//////////////////////////////////////////////////
int async main(void){
//prosopikes metavlites gia kathe epeksergasti
pr int i;
pr int k=0;
pr int j=0;

if($==0)
{
  pprintf("Dose to plithos ton stoixeion:\n");
  scanf("%d",&N);
  pprintf("\n%d\n",N);
  //while(n>=100 || n<=1){
 //pprintf("Dose to plithos ton stoixeionTo plithos stoixeion.\nTa stoixeia  prepei na einai apo 1 mexri 99
:\n");
  //scanf("%d",&n);
  //pprintf("\n%d\n",n);
    //}

for(i=1;i<=N;i++)
{
 table[i]=0;
  }
   }
    barrier;

///sinartiseis poy voithoun stin metrish toy xronou
start
{
initTracing(100000);
startTracing();
}
```

```
///simeio kodika poy einai ipeythinos o epeksergastis P1 na diekperaiosei
if ($==1)
{starttime=getct();

  for(j=1;j<=N;j++)
 {
    kostosP1++;
    if(table[j]==0)
      {table[j]=1;
      }
  else{pprintf("O epeksergastis 1 eftase sto stoixeio %d\n",j);
   stoptime=getct();
      break;
      }
    }
  }
///simeio kodika poy einai ipeythinos o epeksergastis P2 na diekperaiosei
 if ($==2){
   starttimeP2=getct();
     for(j=N;j>=1;j--){
      kostosP2++;
      if(table[j]==0)
       {table[j]=1;
      k++;
       }

  else {
  pprintf("O epeksergastis 2 eftase sto stoixeio %d\n",(k+1));
       stoptimeP2=getct();
      break;
     }
   }
}
start{
  stopTracing();
   writeTraceFile("sum.trv","sum");
    }
 barrier;
if ($==0){
printf("<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>\n");
printf("|¬P1¬|\n");
printf("cpu cycles       :: %d\n",stoptime-starttime);
printf("xronos (se msec) :: %d.%d\n",((stoptime-starttime)*4)/1000,((stoptime-starttime)*4)%1000);
printf("kostos           :: %d\n",kostosP1);
printf("<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>\n");
 printf("|¬P2¬|\n");
printf("cpu cycles       :: %d\n",stoptimeP2-starttimeP2);
printf("xronos (se msec) :: %d.%d\n",((stoptimeP2-starttimeP2)*4)/1000,((stoptimeP2-
starttimeP2)*4)%1000);
printf("kostos           :: %d\n",kostosP2);
printf("<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>\n");
printf("Sinolikos xronos :: %d.%d\n",((stoptime-starttime)+(stoptimeP2-starttimeP2)*4)/1000,((stoptime-
starttime)+(stoptimeP2-starttimeP2)*4)%1000);
printf("Sinoliko  kostos :: %d\n",kostosP1+kostosP2);
}
  barrier;
  return(0);

}
```

A.4Αλγόριθμος Χ και Χ’

```
/*
================================================================================
*/
/* X_newWithBit.c                                                  */
/*                                                                 */
/*Algorithmos X kai X'                                             */
/*                                                                 */
/*Xrisimopoiei ena dianisma megethous 2n-1                                  */
/*Oi epeksergastes anebokatevenoun asigxonista to dentro grafontas tin timh 1 */
/*stis kipselides pou diavazoun.                                   */
/*Kathe epeksergastis termatizei, otan diavasei oti i KM[1]=1               */
/*                                                                 */
/*                                                                 */
/* Panagiota Nicoloaou                                             */
/* 10/12/2008                                                      */
/*
================================================================================
*/



#include <fork.h>
#include <io.h>
# include<math.h>

sh int n=0;//to plithos ton kipselidon pou theloume na grapsoume ton arithmo 1


sh int logarithmos;
sh int tableLog[1000][1000];  //pinakas pou apothikevetai
                        //o diadikos arithmos kathe epeksergasti.
sh int table[1000000];//pinakas pou periexei mesa ta stoixeia
sh  int starttime,stoptime;
sh int athrismaKostous=0;
sh int countMetrisiCostous[128];
FILE *fp;
int async main(void){
  pr int j=0;
  pr int where;
  pr int prosorini;
  pr int prosoriniJ;
  pr int vathos;   //se poio epipedo tou diadikou dentrou briskomaste
  pr int flag=1;
  pr int i;
  pr int p;
  pr int apotelesmaThesis1=0;


  //Evresi ton megiston epeksergaston pou xrisimopoiountai sto programma


  logarithmos=1+(log10((__STARTED_PROCS__-1))/log10(2));

  if($==0)
    {
     pprintf("          ALGORITHMOS X KAI X'\n");
     pprintf("----------------------------------------\n\n");
```

```
    pprintf("Dose ta stoixeia pou theleis na exei o pinakas...\n");
    scanf("%d",&n);
    pprintf("%d\n",n);

    for(i=0;i<(2*n+1);i++)
          {
           table[i]=0;
          }


    //Arxikopoiisi pinaka
    for(i=1;i<=(__STARTED_PROCS__-1);i++)
          {
          countMetrisiCostous[i]=0;

            for(p=1;p<=logarithmos;p++)
              {
               tableLog[i][p]=0;
              }
             }



  }


  barrier;

  fp=fopen("input5.txt","w");
  if(fp==NULL)
  {
            pprintf("ERROR The file does not exist\n");
            exit(-1);
            }

  barrier;

 //Epipedo pou exoun ta filla toy dentrou
 vathos=(ln(n)/ln(2));

 barrier;

// ypologismos xronou pou katanalonetai gia na graftei kai stis 2n-1 kipselides i timi 1
 start initTracing(100000);
 start startTracing();
 starttime=getct();


 if($!=0 && $<=n)
   {
    where=n+$-1;


    while(apotelesmaThesis1!=1)
          {
           //Oloi oi epeksergastes grafoun sto fillo pou vriskontai tin timi 1
           if(table[where]!=1 && where>(n-1) )
             {
```

```
 //if($==0)
 ////pprintf(" ");
            // barrier;
  table[where]=1;


 pprintf("\n @ O epeksergastis %d egrapse stin %d thesi tou pinaka",$,where);
  countMetrisiCostous[$]++;
             where=where/2;
             vathos--;


 }

//An kai ta dyo paidia exoun tin timi 1
//tote oi epeksergastes grafoyn sto simeio pou briskontai tin timi 1
        else if(table[where]!=1  && where<=(n-1))
          {
           if(table[2*where]==1 && table[2*where+1]==1)
                 {

///pprintf(" ");
                  // barrier;
 table[where]=1;
 countMetrisiCostous[$]++;
                pprintf("\n @ O epeksergastis %d egrapse stin %d thesi tou pinaka ",$,where);
                  where=where/2;
                  vathos--;


                 }

            //Otan o aristeros thigatrikos tou komvou pou briskomaste exei tin timi 1
            //eno o deksios thigatrikos exei tin timi 0
            if(table[2*where]==1 && table[2*where+1]==0)
                 {
          //       countMetrisiCostous++;
                  where=2*where+1;
                  vathos++;
                  if(where>(2*n-1))
                    {
                     where=(where-1)/2;
                     vathos--;
                    }
                 }

            //Otan o deksios thigatrikos exei tin timi 1
            //eno o deksios thigatrikos exei tin timi 0
            if(table[2*where]==0 && table[2*where+1]==1)
                   {
                    //countMetrisiCostous++;
                    where=2*where;
                   vathos++;
                    if(where>(2*n-1))
                   {
                    where=where/2;
                    vathos--;
                   }
```

```
                }

          //Otan kai o deksios alla kai o aristeros thigatrikos exoun tin timi 0::
          //Tote oi epeksergastes pou einai diathesimoi anatithetai me basi ton diadiko
  //tous arithmo alla kai to epipedo pou briskontai se kapoia thesi tis mnimis.
          if(table[2*where]==0 && table[2*where+1]==0 && ((where*2)+1<2*n))
                {
                // countMetrisiCostous++;
                 if(flag==1)
                 {
                  //Tmima kodika pou Briskei ton
                  //diadiko arithmo kathe epeksergasti
                   prosorini=$;
                   prosoriniJ=logarithmos;

        while(prosorini!=1 && prosorini!=0)
                        {
                         if(prosorini%2==0)
                           {
                             tableLog[$][prosoriniJ]=0;
                             prosoriniJ--;
                           }

                          else
                            {
                             tableLog[$][prosoriniJ]=1;
                             prosoriniJ--;
                           }
                          prosorini=(prosorini/2);
                        }

                    if(prosorini==1 || prosorini==0)
                          {
                           tableLog[$][prosoriniJ]=1;
                          }
                    flag=2;
                  }

                ////////////

                ///Me basei se poio epipedo briskomaste briskei poio bit
                // apo ton diadiko arithmo toy epeksergasti tha eleksei.
                if((vathos)>logarithmos)
                  {
                   vathos=logarithmos;
                  }
                if(vathos<1)
                  {
                   vathos=1;
                  }
                else{vathos=vathos;}


                //

                //Afou exei vrethei to bit pou tha eleksei
//Eksetazoume an afto to bit einai 1 tote o epeksergastis paei aristera
//Allios an to bit einai 0tote o epeksergastis paei deksia
```

```
                    // -Paei aristera
    if(tableLog[$][vathos]==1)
                      {
                       where=where*2;
                       vathos++;
                       if(where>(2*n))
                              {
                                where=where/2;
                                vathos--;
                            }


                      }
                      //-Paei deksia
                     else if (tableLog[$][vathos]==0)
                      {
                       where=where*2+1;
                       vathos++;
                       if(where>(2*n))
                             {
                              where=(where-1)/2;
                              vathos--;
                               }
                      }
                     }
            }

    apotelesmaThesis1=table[1];
    }
  }


stoptime=getct();
start stopTracing();
start  writeTraceFile("sum.trv","sum");


barrier;

  if($==0)
{
  for(i=1;i<=(__STARTED_PROCS__-1);i++){
  athrismaKostous= countMetrisiCostous[i]+athrismaKostous;
  }
  }

  barrier;


  if($==0)
    {
   printf("\n\n");
            printf("CPU Cycles      :: %d\n",stoptime-starttime);
   printf("Xronos (se msec):: %d\n",(stoptime-starttime)*4);
   printf("Kostos          :: %d\n",athrismaKostous);
            printf("=======================================================\n");
            printf("\n\n");

    /*
```

```
   for(i=1;i<=(__STARTED_PROCS__-1);i++)
              {
               for(p=1;p<=logarithmos;p++)
                 {
                  fprintf("table[%d][%d]==%d",i,p,tableLog[i][p]);
                 }
               printf("\n");
               }*/
      }
  return (0);

  fclose(fp);
}
```

A.5 Αλγόριθμος W

```
/*
================================================================================
*/
/* W_Algorithm.c                                        */
/*                                                      */
/*Algorithmos W:                                        */
/*                                                      */
/*Xrisimopoiei ena dianisma megethous 2n-1                          */
/*O algorithmos ektelei ena brogxo 4on faseon                       */
/*eos otou lithei to provlima.                                      */
/*                                                      */
/*                                                      */
/*                                                      */
/* Panagiota Nicoloaou                                  */
/*                                                      */
/*
================================================================================
*/


#include <math.h>
#include <io.h>
#include <fork.h>



sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh  int starttime,stoptime;
sh int athrismaKostous=1;


//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
```

```
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;
pr int metritis=1;

  j=2;
m=1;
 pros=1;


if(flag==2)
    {
     j=2;

            if(tableW2[1]< (__STARTED_PROCS__-1))
              {
               while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[copyI][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

               if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                      {
                       id=copyI/2;

                       tableW2[copyI]=1;
                      }

               barrier;

               m=1;
               flag=2;
               while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[(copyI+1)][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

               if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                      {

                       id=copyI/2+1;
     tableW2[copyI+1]=1;

                      }
               barrier;
               n=id;
```

```
                while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                        {
                         tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                         n=n/2;

            }

                flag2=2;
                tableW2[1]=tableW2[2]+tableW2[3];

               }

  }barrier;



  while(tableW[1]==0 && flag2!=2)
    {

     if($!=0 )
{
if(pros==1 ){

             n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

              if(tableW[n/2]==0)
                {
                 n=n/2;
                 tableW[n]=tableW[n*2]+tableW[2*n+1];
                 pros=2;
               metritis++;

                }
}


 else if(pros==2 )
{
if(metritis>=3)
{

  fail=1;
              n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

  metritis++;
              if(tableW[n/2]==0)
                {
                 n=n/2;
                 tableW[n]=tableW[n*2]+tableW[2*n+1];
                 pros=2;
                }

}

else if(metritis<3)
{fail=1;
```

```
            n=($+(__STARTED_PROCS__-1)-1);
            n=n/(j/2);
            metritis++;
            if(tableW[n/2]==0)
              {
               n=n/2;
               tableW[n]=tableW[n*2]+tableW[2*n+1];
               pros=2;
              }
    }
}
            j=j*2;
           }
   barrier;
  }

  apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;
  //Kalesma algorithou parallilis Athroisis
  paralliliAthroisi(tableW4);

}




//phasi ergasias
sync void phaseW3()
{
  int i;
  int apotelesmaW3;




  if($!=0 )
  {
    tableW3[$+(__STARTED_PROCS__-1)-1]=1;
     }


 farm{ phaseW4(tableW3);}

  elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
  int i;
```

```
 phaseW4(tableW2); barrier;
 barrier;

//Ypologismos neou apotelesmatos meta apo tin ektelesi tou vrogxoy
 if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
   {
    pprintf("\nTo neo apotelesma einai..\n");
     pprintf("_____\n");
   }
 barrier;



 for(i=1;i<2*(__STARTED_PROCS__-1);i++)
   {
    if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
          {
           pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
          }
   }
 barrier;

 elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
 int tableEpeksergaston[1000];
 pr int p;
 int i;
 int j;
 int prosorini;
 int topicProcessor;


 prosorini=1;
topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
 if( flagPinaka==2 && $==0){
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
   tableW3[i]=tableW2[i];
   }
   }
 barrier;

 for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
    tableW2[i]=0;
    tableEpeksergaston[i]=0;
   }  tableW2[1]=tableW3[1];
```

```
 barrier;

  for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {
           for(j=1;j<128;j++)
             {
               tableEpeksergaston2[i][j]=0;
             }
     }
   barrier;


   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
    {
           if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
             {
               tableW2[prosorini*2]=0;
             }
           else
             {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
             }

           tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
           prosorini=prosorini+1;
     }

barrier;


   if($==0 && flagW2==0)
   {
           pprintf("\nTo apotelesma tou w2 einai...\n");
           pprintf("_____________________________\n");
           }
   barrier;
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {

           if($==0 && flagW2==0)
            {
              pprintf("|%d| stin thesi %d \n",tableW2[i],i);
              }
     }

   barrier;
   flagW2=1;

   p=1;

   //Anathesi epeksergaston oi opoioi den exoun
   //katareysei stis theseis
   //tou pinaka pou exoume tin timi 0
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
```

```
   start{
   while(topicProcessor!=1)
    {
          tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
          tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

          topicProcessor=topicProcessor/2;


      if ((tableEpeksergaston[p*2+1])!=0 )
           {

              p=(p*2)+1;
              }
else      if((tableEpeksergaston[p*2])!=0)
              {
              p=p*2;
              }
       }
      }

   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);


   //Arithis epeksergaston pou antistoixei stin kathe thesi
   if($==0 && flagE==0)
    {

  pprintf("\nOi epeksergastes anatithente os eksis\n");
          pprintf("_____________________________________\n");
    }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
          if($==0 && flagE==0)
           {
              pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

           }



                  if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
          {
           copyI=i;
             leftEpeksergastes=tableEpeksergaston[copyI];
             if(i>=(__STARTED_PROCS__-1)){

              break;

             }
     }
       }

   barrier;
   flagE=1;
```

```
//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
    {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
            {
             i++;
            }
          if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI][j]=i;
             j++;
             leftEpeksergastes--;
             i++;
             copy=i;
            }
    }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
    while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
      {
    i++;
      }
    if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI+1][j]=i;
             j++;
             rightEpeksergastes--;
             copy=i;
             i++;
            }
  }  barrier;
 if($==0)
  {
   pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
  }
 for(i=1;i<2*(__STARTED_PROCS__-1);i++)
  {
    for(j=1;j<128;j++)
      {
           if(tableEpeksergaston2[i][j]!=0)
            {
             if($==0)
               {
                     pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d \n",tableEpeksergaston2[i][j],i);
               }
            }
      }
```

```
   }

}


//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

int i;
flag=1;
flag2=1;


  //Arxikopoiisi pinakon
  if($==0)
{
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
          {
            tableW1[i]=0;
          }


    for(i=1;i<128;i++)
          {
            diathesimoiEpeksergastes[i]=0;
          }
      }

 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($!=0 )
   {
    tableW1[$+(__STARTED_PROCS__-1)-1]=1;
 }

   barrier;

//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
  paralliliAthroisi(tableW1);

  barrier;


apotelesmaW1=tableW1[1];

diathesimoiEpeksergastes[$]=1;

barrier;

flagW1=1;
flag=2;
```

```
}


async int main(void)
{
 int table[10000];
 int i;
 pr int flagW3=0;


 //arxikopoiisi pinaka pou periexei mesa ta stoixeia tou provlimatos
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
    tableW3[i]=0;
   } barrier;

 //Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
start{
 initTracing(100000);
 startTracing();
 starttime=getct();
 phaseW3();
 stoptime=getct();
 stopTracing();
 writeTraceFile("sum.trv","sum");
 }



//To apotelesma toy pinaka meta apo tin phasi W3 kai tin phasi W4
 if($==0 && flagW3==0)
   {
  pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai ::\n");
  pprintf("_____\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
          {
           pprintf("|%d| tin thesi %d\n",tableW3[i],i);
          }
    flagW3=1;
   }
  barrier;

 //An esto kai enas epeksergastis exei katarevsei tote ekteleitai
 //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
 //theoreite oti ekteloume ena dentriko bima
 while(elegxos!=(__STARTED_PROCS__-1))
   {
    phaseW1();  barrier;

    //ipologismos kostous se kathe dendriko bima
    athrismaKostous++;

    phaseW2();   barrier;
    flag=2;
    phase_W3(); barrier;
    flagPinaka=2;
}
```

```
barrier;

if($==0){

   printf("\n\n");
            printf("============================================================\n");
   printf("CPU Cycles      :: %d\n",stoptime-starttime);
   printf("Xronos (se msec):: %d\n",(stoptime-starttime)*4);
   printf("Kostos          :: %d\n",athrismaKostous*(__STARTED_PROCS__-1));
            printf("============================================================\n");
            printf("\n\n");

}

  return(0);
}
```

A.6 Αλγόριθμος W βέλτιστος

```
/*
================================================================================
*/
/* W_Algorithm.c                                                  */
/*                                                                */
/*Algorithmos W Veltistos:                                        */
/*                                                                */
/*Xrisimopoiei ena dianisma megethous 2n-1 alla toram me          */
/* epksergastes ligoterous apo ta stoixeia                        */
/*O algorithmos ektelei ena brogxo 4on faseon                     */
/*eos otou lithei to provlima.                                    */
/*                                                                */
/*                                                                */
/*                                                                */
/* Panagiota Nicoloaou                                            */
/* 10/12/2008                                                     */
/*
================================================================================
*/


#include <math.h>
#include <io.h>
#include <fork.h>


sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagW2=0;
sh int flagW3=0;
sh int n;
sh int veltistos=0;
sh int stoixeia;
sh int statiki=1;
sh  int starttime,stoptime;
sh int athrismaKostous=1;

//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
```

```
 pr int j;
 pr int l;
 pr int a;
 pr int b;
 pr int m;
 pr int p;
 pr int pros;
 pr int id;
 j=2;
 m=1;
pros=1;


 if(flag==2 )//&& $!=3)// $!=2 && $!=1)
   {
    j=2;
    if($!=0){

            if(tableW2[1]<n )
              {
               while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[copyI][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

               if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                      {

                       id=copyI/2;

                       tableW2[copyI]=1;
                      }

               barrier;

               m=1;
               flag=2;
               while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[(copyI+1)][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

               if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                      {

                      id=copyI/2+1;

                       tableW2[copyI+1]=1;

                      }
               barrier;
```

```
                l=id;


                 while((tableW2[l]!=tableW2[l*2]+tableW2[2*l+1]) && l>=1 && l<=n-
1)//((__STARTED_PROCS__-1)-1))
                        {
                          tableW2[l]=tableW2[l*2]+tableW2[2*l+1];

    l=l/2;
                        }

                 flag2=2;
                 tableW2[1]=tableW2[2]+tableW2[3];

               }
}

 }barrier;




 if(veltistos==0){

              while(tableW[1]==0 && flag2!=2){
 if($!=0 )//&& $!=3 && $!=4 )
 {
if(pros==1){
              l=($+(__STARTED_PROCS__-1)-1);
              l=l/(j/2);

              if(tableW[l/2]==0)
                {
                  l=l/2;
                  tableW[l]=tableW[l*2]+tableW[2*l+1];


      pros=2;


               }
}
 else
{

              l=($+(__STARTED_PROCS__-1)-1);
              l=l/(j/2);

              if(tableW[l/2]==0)
                {
                  l=l/2;
                  tableW[l]=tableW[l*2]+tableW[2*l+1];
                  pros=2;


               }
```

```
}
            j=j*2;
            }  barrier;

            }

 }

barrier;



 apotelesmaW4=tableW[1];

}


//
async void seiriako_athrisma(int tableW[])
{
 pr int siriaka;
 pr int thesi;
 pr int dummy;
 int i;
 pr int p=0;
 pr int m=0;
 pr int dummy1;



siriaka=stoixeia*$;
            thesi=n;
            dummy=0;


dummy1=1;
start{
while((__STARTED_PROCS__-1)<(thesi)){
  dummy=0;
  p=0;
  m=0;



                dummy=tableW[(siriaka+n-stoixeia+p)/dummy1]+tableW[((siriaka+n-
stoixeia+p)/dummy1)+1]+dummy;



 while(m<(n/(__STARTED_PROCS__-1)) ){

   if(((siriaka+n-stoixeia+m)/dummy1)<2*n){
            tableW[((siriaka+n-stoixeia+m)/dummy1)/2]=dummy;



 }
```

```
   m++;

  }

  dummy1=dummy1*2;

   thesi=thesi/2;
 }
}

barrier;



  barrier;
    paralliliAthroisi(tableW);
    barrier;

}



//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;
   //Kalesma algorithou parallilis Athroisis
  paralliliAthroisi(tableW4);

}


//phasi ergasias
sync void phaseW3()
{
  int i;
  int apotelesmaW3;

  pr int siriaka;
  pr int p=0;



  if(n>(__STARTED_PROCS__-1) && $!=0){// && $!=3){//&& $!=2 && $!=1){


     stoixeia=(log10(n)/log10(2))+1;


     if(n%(__STARTED_PROCS__-1)!=0){
           stoixeia=ceil(n/(__STARTED_PROCS__-1));}
     else{

           stoixeia=n/(__STARTED_PROCS__-1);
}
```

```
siriaka=$*stoixeia;
          while(p<stoixeia){

                  if((siriaka+n-stoixeia+p)<2*n){
tableW3[siriaka+n-stoixeia+p]=1;
                  }
 p++;

 }

              farm seiriako_athrisma(tableW3);

  }



 else{
   if($!=0 ){//&& $!=3 ){//$!=2 && $!=1){
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;
farm paralliliAthroisi(tableW3);
   }
     barrier;



 phaseW4(tableW3);

 }
 barrier;
  elegxos=apotelesmaW4;

}

//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
 int i;


 flag=2;
 phaseW4(tableW2); barrier;
 barrier;


 //Ypologismos neou apotelesmatos meta apo tin ektelesi tou vrogxoy
 if( flag2==2 && $==0 &&  apotelesmaW4==n)
   {
     printf("\nTo neo apotelesma einai..\n");
   pprintf("_____\n");
   }

 for(i=1;i<2*n;i++)
   {
    if( flag2==2 && $==0 &&  apotelesmaW4==n)
           {
            pprintf(" |%d| stin thesi  %d\n",tableW2[i],i);
           }
   }
 barrier;
```

```
  elegxos=apotelesmaW4;

}


//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
  int i;
  int j;
  int prosorini;
  int topicProcessor;

 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<n*2;i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
for(i=1;i<2*n;i++)
    {
      tableW2[i]=0;
      tableEpeksergaston[i]=0;
    }  tableW2[1]=tableW3[1];
 barrier;


    for(i=1;i<2*n;i++)
      {
            for(j=1;j<128;j++)
              {
                tableEpeksergaston2[i][j]=0;
              }
      }
    barrier;

    while(((prosorini*2)+1)<=((2*n)-1))
      {
            if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
              {
                tableW2[prosorini*2]=0;
              }
            else
              {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
              }

            tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
```

```
            prosorini=prosorini+1;
     }
   barrier;


   if($==0 && flagW2==0)
   {
            pprintf("\nTo apotelesma tou w2 einai...\n");
            pprintf("_____\n");
            }
   barrier;
   for(i=1;i<2*n;i++)
     {
            if($==0 && flagW2==0)
            {
               pprintf("|%d| stin thesi %d \n",tableW2[i],i);
               }


     }
   barrier;
   flagW2=1;
 barrier;



   p=1;


   //Anathesi epeksergaston oi opoioi den exoun
   //katareysei stis theseis
   //tou pinaka pou exoume tin timi 0
     tableEpeksergaston[1]=tableW1[1];
    topicProcessor=n;
   while(topicProcessor!=1)
     {
            tableEpeksergaston[p*2]=( tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
            tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];


            topicProcessor=topicProcessor/2;



 if((tableEpeksergaston[p*2])!=0 )
              {

p=p*2;
               }


else if((tableEpeksergaston[p*2+1])!=0 )
            {
   p=(p*2)+1;

            }

     }
```

```
   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);

  barrier;
   if($==0 && flagW3==0)
     {
          pprintf("\nOi epeksergastes anatithente os eksis...\n");
          pprintf("_____\n");
     }
start{

   for(i=1;i<2*n;i++)
     {

//Arithis epeksergaston pou antistoixei stin kathe thesi
          if($==0 && flagW3==0)
            {
              pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);
            }



                  if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1))
          {
           copyI=i;
             leftEpeksergastes=tableEpeksergaston[copyI];
             if(i>=n){

               break;

             }
          }
     }

flagW3=1;
     }
 if(copyI+1<2*n){

//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
   rightEpeksergastes=tableEpeksergaston[copyI+1];
    }
   else   rightEpeksergastes=0;
i=1;
   j=1;
   while(leftEpeksergastes!=0)
     {
          while (i<(__STARTED_PROCS__-1) &&  diathesimoiEpeksergastes[i]!=1)
            {
             i++;
            }
          if(diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI][j]=i;
             j++;
             leftEpeksergastes--;
             i++;
```

```
                copy=i;
              }

    }

 i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
    while (i<(__STARTED_PROCS__-1) &&  diathesimoiEpeksergastes[i]!=1)
      {
    i++;
      }

    if( diathesimoiEpeksergastes[i]==1)
              {
                tableEpeksergaston2[copyI+1][j]=i;
                j++;
                rightEpeksergastes--;
                copy=i;
                i++;
              }
  }  barrier;
  barrier;
if($==0)
  {
    pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
  }
 for(i=1;i<2*n;i++)
  {
    for(j=1;j<128;j++)
      {
            if(tableEpeksergaston2[i][j]!=0)
              {
                if($==0)
                  {
                        pprintf("\n-O epeksergastis (%d) tha paei stin thesi %d \n",tableEpeksergaston2[i][j],i);
                  }
              }
      }
  }

}


//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;
  flag=1;
  flag2=1;


//Arxikopoiisi pinakon
   if($==0)
{
     for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
```

```
            {

              tableW1[i]=0;
            }


     for(i=1;i<128;i++)
            {
              diathesimoiEpeksergastes[i]=0;
            }
        }
 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei
     if($!=0 )//&& $!=3)// $!=2 && $!=1){
    {  tableW1[$+(__STARTED_PROCS__-1)-1]=1;
    }
      barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
paralliliAthroisi(tableW1);


 barrier;
  apotelesmaW1=tableW1[1];


  //if($!=3)//$!=2 && $!=1)
    //     {
    diathesimoiEpeksergastes[$]=1;
    //  }

barrier;

}




async int main(void)
{
  int table[10000];
  int i;

//arxikopoiisi pinaka pou periexei mesa ta stoixeia tou provlimatos
 for(i=1;i<(2*n);i++)
    {
      tableW3[i]=0;
    }

   if($==0)
    {
```

```
      pprintf("          ALGORITHMOS W-BELTISTOS\n");
      pprintf("---------------------------------------\n\n");
      pprintf("Dose ta stoixeia pou theleis na exei o pinakas...\n");
      scanf("%d",&n);
      pprintf("%d\n",n);
    }

   barrier;
//Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
  start{
initTracing(100000);
startTracing();
  starttime=getct();
 phaseW3();
 stoptime=getct();
stopTracing();
writeTraceFile("sum.trv","sum");
  }

 //To apotelesma toy pinaka meta apo tin phasi W3 kai tin phasi W4
 barrier;
  if($==0)
    {
      pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai::\n");
      pprintf("_____________________________\n");
      for(i=1;i<2*n;i++)
            {
              pprintf("|%d| tin thesi %d\n",tableW3[i],i);
            }
    }

//An esto kai enas epeksergastis exei katarevsei tote ekteleitai
//o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
//theoreite oti ekteloume ena dentriko bima

while(elegxos!=n)
    {
      phaseW1();   barrier;
      phaseW2();   barrier;
      flag=2;
      phase_W3(); barrier;
      flagPinaka=2;
}

 barrier;
 if($==0){

   printf("\n\n");
            printf("============================================================\n");
   printf("CPU Cycles      :: %d\n",stoptime-starttime);
   printf("Xronos (se msec):: %d\n",(stoptime-starttime)*4);
   printf("Kostos          :: %d\n",athrismaKostous*(__STARTED_PROCS__-1));
            printf("============================================================\n");
            printf("\n\n");

}


  return(0);}
```

A.7 Αλγόριθμος W :: Σενάριο 1

```
/*
================================================================================
*/
/* W_AlgorithmSenario1.c                                        */
/*                                                              */
/*Algorithmos W:                                                */
/*                                                              */
/*Xrisimopoiei ena dianisma megethous 2n-1                      */
/*O algorithmos ektelei ena brogxo 4on faseon                   */
/*eos otou lithei to provlima.                                  */
/*Se ayto to Senario oi epeksergastes 5 kai 7 katarreoun        */
/*prin grapsoun tin timi 1 sta antistoixa filla                 */
/*                                                              */
/*                                                              */
/*                                                              */
/* Panagiota Nicoloaou                                          */
/* 10/12/2008                                                   */
/*
================================================================================
*/



#include <math.h>
#include <io.h>
#include <fork.h>

sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagW2=0;
sh int flagW3=0;
sh int starttime;
sh int stoptime;
sh int starttime2;
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
sh int sinolikoKostosEpeksergastes=0;

//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
```

```
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  j=2;
m=1;
 pros=1;

  if(flag==2)
    {
     j=2;
     if($!=0 && $!=5 && $!=7)
            {
             if(tableW2[1]< (__STARTED_PROCS__-1) )
               {
                while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                       {
                        if(tableEpeksergaston2[copyI][m]==0)
                          {
                           flag=3;
                          }
                        m++;
                       }

                if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                       {
                        $=copyI/2;
                        tableW2[copyI]=1;
                       }

                barrier;

                m=1;
                flag=2;
                while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                       {
                        if(tableEpeksergaston2[(copyI+1)][m]==0)
                          {
                           flag=3;
                          }
                        m++;
                       }

                if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                       {
                        $=copyI/2+1;
                        tableW2[copyI+1]=1;

                       }
                barrier;
                n=$;
```

```
                while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                        {
                         tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                         n=n/2;
                        }

                flag2=2;
                tableW2[1]=tableW2[2]+tableW2[3];

              }
            }
  }barrier;


  while(tableW[1]==0 && flag2!=2)
    {
       if($!=0 && $!=5 && $!=7)
            {
if(pros==1){
              n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

              if(tableW[n/2]==0)
                {
                  n=n/2;
                  tableW[n]=tableW[n*2]+tableW[2*n+1];
                  pros=2;


                }
            }
 else
{

              n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

              if(tableW[n/2]==0)
                {
                  n=n/2;
                  tableW[n]=tableW[n*2]+tableW[2*n+1];
                  pros=2;


                }



            }
             j=j*2;
              }  barrier;
  }

  apotelesmaW4=tableW[1];

}
```

```
//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;


 //Kalesma algorithou parallilis Athroisis
  paralliliAthroisi(tableW4);

}



//phasi ergasias
async void phaseW3()
{
  int i;
  int apotelesmaW3;



  if($!=0 && $!=5 && $!=7)
  {
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;
     }
     barrier;

  phaseW4(tableW3);

  elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
  int i;


  phaseW4(tableW2); barrier;
  barrier;



//Ypologismos neou apotelesmatos meta apo tin ektelesi tou vrogxoy

 if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
   {
    pprintf("\nTo neo apotelesma einai..\n");
   pprintf("_____________________________\n");
    }
 barrier;



  for(i=1;i<2*(__STARTED_PROCS__-1);i++)
    {
```

```
    if( flag2==2 && $==0 &&  apotelesmaW4==(__STARTED_PROCS__-1))
{
            pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
}
  }
  barrier;
  elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;

 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW2[i]=0;
      tableEpeksergaston[i]=0;
    }  tableW2[1]=tableW3[1];
 barrier;



    for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
      {
            for(j=1;j<128;j++)
              {
                tableEpeksergaston2[i][j]=0;
              }
      }
    barrier;

    while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
      {
            if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
              {
```

```
              tableW2[prosorini*2]=0;
             }
            else
             {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
             }

            tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
            prosorini=prosorini+1;
     }
   barrier;


  if($==0 && flagW2==0)
  {
            pprintf("\nTo apotelesma tou w2 einai...\n");
            pprintf("_____\n");
}
   barrier;
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {
            if($==0 && flagW2==0)
            {
              pprintf("|%d| stin thesi %d \n",tableW2[i],i);
              }


     }
  barrier;
   flagW2=1;
 barrier;

   p=1;

//Anathesi epeksergaston oi opoioi den exoun
//katareysei stis theseis
//tou pinaka pou exoume tin timi 0
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
   while(topicProcessor!=1)
     {
            tableEpeksergaston[p*2]=( tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
            tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

            topicProcessor=topicProcessor/2;

 if((tableEpeksergaston[p*2])!=0)
             {
              p=p*2;
             }


else         if((tableEpeksergaston[p*2+1])!=0 )
             {
```

```
            p=(p*2)+1;
           }


     }


   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);

//Arithis epeksergaston pou antistoixei stin kathe thesi
   if($==0 && flagW3==0)
   {
         pprintf("\nOi epeksergastes anatithente os eksis...\n");
         pprintf("_____________________________________\n");
    }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
         if($==0 && flagW3==0)
          {
            pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);
          }
         flagW3=1;


                  if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
         {
          copyI=i;
            leftEpeksergastes=tableEpeksergaston[copyI];
            if(i>=(__STARTED_PROCS__-1)){

              break;

            }


         }
     }



//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis

   rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
    {
         while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
          {
           i++;
          }
         if( diathesimoiEpeksergastes[i]==1)
          {
```

```
                tableEpeksergaston2[copyI][j]=i;
                j++;
                leftEpeksergastes--;
                i++;
                copy=i;
               }
     }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
    while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
      {
    i++;
      }
    if( diathesimoiEpeksergastes[i]==1)
              {
                tableEpeksergaston2[copyI+1][j]=i;
                j++;
                rightEpeksergastes--;
                copy=i;
                i++;
               }
  }  barrier;
 if($==0)
  {
   pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
 }
 for(i=1;i<2*(__STARTED_PROCS__-1);i++)
  {
   for(j=1;j<128;j++)
     {
            if(tableEpeksergaston2[i][j]!=0)
             {
               if($==0)
                {
                        pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d \n",tableEpeksergaston2[i][j],i);
                }
             }
     }
  }

}




//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;
  flag=1;
  flag2=1;
```

```
//Arxikopoiisi pinakon
  if($==0)
{
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
          {
           diathesimoiEpeksergastes[i]=0;
           tableW1[i]=0;
          }


    for(i=1;i<128;i++)
          {
           diathesimoiEpeksergastes[i]=0;
          }
      }
 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
  if($!=0 && $!=5 && $!=7 )
    {
     tableW1[$+(__STARTED_PROCS__-1)-1]=1;
   } barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
  paralliliAthroisi(tableW1);



  apotelesmaW1=tableW1[1];




  if($!=5 && $!=7 )
  {
     diathesimoiEpeksergastes[$]=1;
     }
}




async int main(void)
{
  int table[10000];
  int i;



 for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
```

```
    tableW3[i]=0;
   }

   for(i=1;i<100;i++)
   {
   dentriko_vima_Epeksergastes[i]=0;
    dentriko_vima_Xronos[i]=0;
   }


  barrier;



//Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
 start{
 initTracing(100000);
 startTracing();
 starttime=getct();
 phaseW3();
 stoptime=getct();
 stopTracing();
 writeTraceFile("sum.trv","sum");
 }
 barrier;

 //To apotelesma toy pinaka meta apo tin phasi W3 kai tin phasi W4
 if($==0)
   {
    pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai::\n");
    pprintf("_____\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
           {
            pprintf("|%d| tin thesi %d\n",tableW3[i],i);
           }
   }


 //An esto kai enas epeksergastis exei katarevsei tote ekteleitai
 //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
 //theoreite oti ekteloume ena dentriko bima

start{
 initTracing(100000);
 startTracing();}

i=1;
 while(elegxos!=(__STARTED_PROCS__-1))
   {
    starttime2=getct();
    phaseW1();   barrier;
    phaseW2();   barrier;
    flag=2;
    phase_W3(); barrier;
    flagPinaka=2;

    stoptime2=getct();

    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
```

```
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
  i++;
  }
   start{
    stopTracing();
    writeTraceFile("sum.trv","sum");
    }


barrier;



if($==0){



   for(i=1;i<100;i++){

   if(dentriko_vima_Epeksergastes[i]!=0){
   sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
  sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
   }
  else{break;}
   }

   printf("\n\n");
            printf("=========================================================\n");
  // printf("CPU Cycles               :: %d\n",stoptime-starttime+stoptime2-starttime2);
   printf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
   printf("Kostos apo Xrono           :: %d\n",(i-1)*(stoptime2-starttime2));
   printf("Kostos apo Epeksergastes   :: %d\n",sinolikoKostosEpeksergastes);
   pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
            printf("=========================================================\n");
            printf("\n\n");

}
  barrier;
  return(0);
}
```

A.8 Αλγόριθμος W :: Σενάριο 2

```
/*
================================================================================
*/
/* W_AlgorithmSenario3.c                                        */
/*                                                              */
/*Algorithmos W:                                                */
/*                                                              */
/*Xrisimopoiei ena dianisma megethous 2n-1                      */
/*O algorithmos ektelei ena brogxo 4on faseon                   */
/*eos otou lithei to provlima.                                  */
/*Se ayto to Senario oi epeksergastes 6 kai 16 katarreoun       */
/*prin grapsoun tin timi 1 sta antistoixa filla                 */
/*                                                              */
/*Epeita kata tin proti epanalipsi tou vrogxou katarreoun kai   */
/*oi epeksergastes 5,7,8,13,14,15                               */
/* Panagiota Nicoloaou                                          */
/*                                                              */
/*
================================================================================
*/



#include <math.h>
#include <io.h>
#include <fork.h>


sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh int starttime;
sh int stoptime;
sh int starttime2;
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
```

```
sh int sinolikoKostosEpeksergastes=0;


//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;


  j=2;
m=1;
 pros=1;

  if(flag==2)
    {
     j=2;
     if($!=0 && $!=6 && $!=16 && $!=5 && $!=7 && $!=8 && $!=13 && $!=14 && $!=15 )//&&
$!=1 && $!=2 && $!=3 && $!=4)
            {
             if(tableW2[1]< (__STARTED_PROCS__-1) )
               {
                while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[copyI][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

                if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                      {id=copyI/2;

                       tableW2[copyI]=1;
                      }

                barrier;

                m=1;
                flag=2;
                while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[(copyI+1)][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

                if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
```

```
                    { id=copyI/2+1;

                     tableW2[copyI+1]=1;

                    }
                barrier;
                n=id;


                while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                    {
                     tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                     n=n/2;

          }

                flag2=2;
                tableW2[1]=tableW2[2]+tableW2[3];

             }
          }
 }barrier;


 while(tableW[1]==0 && flag2!=2)
   {
    if($!=0 && $!=6 && $!=16)
{
if(pros==1 ){
 if (fail==0 ||  ( $!=5 && $!=7 && $!=8 && $!=13 && $!=14 && $!=15)){


n=($+(__STARTED_PROCS__-1)-1);
            n=n/(j/2);

            if(tableW[n/2]==0)
              {
               n=n/2;
               tableW[n]=tableW[n*2]+tableW[2*n+1];
               pros=2;


              }
 }
}
 else if(pros==2 && ( $!=16 && $!=5 && $!=7 && $!=8 && $!=13 && $!=14 && $!=15))
{
 fail=1;
            n=($+(__STARTED_PROCS__-1)-1);
            n=n/(j/2);

            if(tableW[n/2]==0)
              {
               n=n/2;
               tableW[n]=tableW[n*2]+tableW[2*n+1];
               pros=2;
              }
```

```
}
            j=j*2;
            }  barrier;
  }

  apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;



  //Kalesma algorithou parallilis Athroisis
  paralliliAthroisi(tableW4);

}




//phasi ergasias
async void phaseW3()
{
  int i;
  int apotelesmaW3;



  if($!=0 && $!=6 && $!=16)
  {
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;
     }
     barrier;

  phaseW4(tableW3);

  elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
  int i;


  phaseW4(tableW2); barrier;



barrier;
```

```
/// if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
  ///  {
    ///  pprintf("\nTo neo apotelesma einai..\n");
    ///  pprintf("_____\n");
 ///   }
/// barrier;



 /// for(i=1;i<2*(__STARTED_PROCS__-1);i++)
 ///   {
///      if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
///           {
///             pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
///           }
 ///   }
 /// barrier;

  elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;


 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);


//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW2[i]=0;
      tableEpeksergaston[i]=0;
    }  tableW2[1]=tableW3[1];
 barrier;
```

```
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
          for(j=1;j<128;j++)
            {
             tableEpeksergaston2[i][j]=0;
            }
    }
   barrier;

   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
    {
          if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
            {
             tableW2[prosorini*2]=0;
            }
          else
            {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
            }

          tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
          prosorini=prosorini+1;
    }

barrier;


////    if($==0 && flagW2==0)
  ////  {
///          pprintf("\nTo apotelesma tou w2 einai...\n");
///          pprintf("_____________________________________\n");
///          }
///    barrier;
 ///   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
 ///     {
///          if($==0 && flagW2==0)
///          {
///             pprintf("|%d| stin thesi %d \n",tableW2[i],i);
///             }
 ///    }

  barrier;
    flagW2=1;

barrier;



   p=1;

//Anathesi epeksergaston oi opoioi den exoun
//katareysei stis theseis
//tou pinaka pou exoume tin timi 0
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
   start{
```

```
   while(topicProcessor!=1)
    {
          tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
          tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

          topicProcessor=topicProcessor/2;



      if ((tableEpeksergaston[p*2+1])!=0 )
           {

             p=(p*2)+1;
           }
else       if((tableEpeksergaston[p*2])!=0)
           {
             p=p*2;
           }


    }
   }

   leftEpeksergastes=(__STARTED_PROCS__-1);


//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
   rightEpeksergastes=(__STARTED_PROCS__-1);

  ///  if($==0 && flagE==0)
   ///   {
///          pprintf("\nOi epeksergastes anatithente os eksis...\n");
///          pprintf("_____________________________________\n");
  ///    }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {
///          if($==0 && flagE==0)
///            {
             ///     pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

           /// }



                    if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
           {
            copyI=i;
              leftEpeksergastes=tableEpeksergaston[copyI];
              if(i>=(__STARTED_PROCS__-1)){

                break;

              }


           }
```

```
    }


  barrier;
  flagE=1;




  rightEpeksergastes=tableEpeksergaston[copyI+1];


  i=1;
  j=1;
  while(leftEpeksergastes!=0)
    {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
            {
             i++;
            }
          if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI][j]=i;
             j++;
             leftEpeksergastes--;
             i++;
             copy=i;
            }
    }


  i=copy;
  j=1;
 while(rightEpeksergastes!=0)
  {
   while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
     {
    i++;
     }
    if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI+1][j]=i;
             j++;
             rightEpeksergastes--;
             copy=i;
             i++;
            }
  }  barrier;
 ///if($==0)
 ///  {
 ///    pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
 ///  }
 ///for(i=1;i<2*(__STARTED_PROCS__-1);i++)
 ///  {
 ///    for(j=1;j<128;j++)
  ///     {
///           if(tableEpeksergaston2[i][j]!=0)
///             {
```

```
              ///   if($==0)
               ///    {
                        /// pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d
\n",tableEpeksergaston2[i][j],i);
              ///     }
////            }
  ///    }
///   }

}


//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;


  flag=1;
  flag2=1;


//Arxikopoiisi pinakon
   if($==0)
{
     for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
            {

              tableW1[i]=0;
            }


     for(i=1;i<128;i++)
            {
              diathesimoiEpeksergastes[i]=0;
            }
       }
 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($!=0 && $!=6 && $!=16 && $!=5 && $!=7 && $!=8 && $!=13 && $!=14 && $!=15 )//&& $!=1
&& $!=2 && $!=3 && $!=4)
    {
     tableW1[$+(__STARTED_PROCS__-1)-1]=1;


   } barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
  paralliliAthroisi(tableW1);
```

```
  barrier;

  apotelesmaW1=tableW1[1];




  if($!=6 && $!=16 && $!=5 && $!=7 && $!=8 && $!=13 && $!=14 && $!=15 )//&& $!=1 && $!=2
&& $!=3 && $!=4)
  {
     diathesimoiEpeksergastes[$]=1;
     }


  barrier;

   flagW1=1;

  flag=2;

}




async int main(void)
{
  int table[10000];
  int i;
  pr int flagW3=0;


  for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
     tableW3[i]=0;
    }

    for(i=1;i<100;i++)
    {
    dentriko_vima_Epeksergastes[i]=0;
     dentriko_vima_Xronos[i]=0;
    }


    barrier;




  //Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
  start{
  initTracing(100000);
  startTracing();
  starttime=getct();
  phaseW3();
  stoptime=getct();
  stopTracing();
  writeTraceFile("sum.trv","sum");
```

```
 }
 barrier;


//To apotelesma toy pinaka meta apo tin phasi W3 kai tin phasi W4
 if($==0 && flagW3==0)
   {
    pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai\n");
    pprintf("_____\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
          {
           pprintf("|%d| tin thesi %d\n",tableW3[i],i);
          }

    flagW3=1;
   }




 //An esto kai enas epeksergastis exei katarevsei tote ekteleitai
 //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
 //theoreite oti ekteloume ena dentriko bima


start{
 initTracing(100000);
 startTracing();}

i=1;

 while(elegxos!=(__STARTED_PROCS__-1))
   {
  starttime2=getct();
     phaseW1();  barrier;
    phaseW2();  barrier;
    flag=2;
    phase_W3(); barrier;
    flagPinaka=2;

    stoptime2=getct();

    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
 i++;

}


 start{
    stopTracing();
    writeTraceFile("sum.trv","sum");
    }

 barrier;

//  if($==0){
```

```
  for(i=1;i<100;i++){

   if(dentriko_vima_Epeksergastes[i]!=0){
   sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
  sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
   }
  else{break;}
   }

 //  pprintf("\n\n");
//            pprintf("=============================================================\n");
 //  pprintf("CPU Cycles               :: %d\n",stoptime-starttime+stoptime2-starttime2);
   pprintf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
   pprintf("Kostos apo Xrono           :: %d\n",(i-1)*(stoptime2-starttime2));
   pprintf("Kostos apo Epeksergastes   :: %d\n",sinolikoKostosEpeksergastes);
  // pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
            // pprintf("=============================================================\n");
            // pprintf("\n\n");

//}barrier;
 barrier;


return(0);
}
```

A.9 Αλγόριθμος W :: Σενάριο 3

```
/*
================================================================================
*/
/* W_AlgorithmSenario5.c                                        */
/*                                                              */
/*Algorithmos W:                                                */
/*                                                              */
/*Xrisimopoiei ena dianisma megethous 2n-1                      */
/*O algorithmos ektelei ena brogxo 4on faseon                   */
/*eos otou lithei to provlima.                                  */
/*Se ayto to Senario oi epeksergastes 3 kai 4 katarreoun        */
/*prin grapsoun tin timi 1 sta antistoixa filla                 */
/*Epeita kata tin proti epanalipsi katarreoun olio oi ipolipoi
*/
/*epeksergastes ektos apo ena epeksergasti, ton epeksergasti
*/
/*me prosopiko arithmo iso me 1                                                 */
/* Panagiota Nicoloaou                                          */
/*                                                              */
/*
================================================================================
*/



#include <math.h>
#include <io.h>
#include <fork.h>


sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh int starttime;
sh int stoptime;
sh int starttime2;
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
```

```
sh int sinolikoKostosEpeksergastes=0;


//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;


  j=2;
m=1;
 pros=1;

  if(flag==2)
    {
     j=2;
     if($!=0 && $==1)
            {
              if(tableW2[1]< (__STARTED_PROCS__-1))
                {
                 while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                        {
                         if(tableEpeksergaston2[copyI][m]==0)
                           {
                            flag=3;
                           }
                         m++;
                        }

                 if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                        {
                         id=copyI/2;

                         tableW2[copyI]=1;
                        }

                 barrier;

                 m=1;
                 flag=2;
                 while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                        {
                         if(tableEpeksergaston2[(copyI+1)][m]==0)
                           {
                            flag=3;
                           }
                         m++;
                        }

                 if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
```

```
                    {

                     id=copyI/2+1;
    tableW2[copyI+1]=1;

                    }
                barrier;
                n=id;


                while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                    {
                     tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                     n=n/2;

           }

                flag2=2;
                tableW2[1]=tableW2[2]+tableW2[3];

             }
           }
 }barrier;


 while(tableW[1]==0 && flag2!=2)
   {


    if($!=0 && $!=3 && $!=4)
{
if(pros==1 ){


 n=($+(__STARTED_PROCS__-1)-1);
            n=n/(j/2);

            if(tableW[n/2]==0)
              {
               n=n/2;
               tableW[n]=tableW[n*2]+tableW[2*n+1];
               pros=2;


              }

}
 else if(pros==2)
{
 fail=1;
            n=($+(__STARTED_PROCS__-1)-1);
            n=n/(j/2);

            if(tableW[n/2]==0)
              {
               n=n/2;
               tableW[n]=tableW[n*2]+tableW[2*n+1];
               pros=2;
```

```
            }


}
            j=j*2;
            }  barrier;
  }

  apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;

  //Kalesma algorithou parallilis Athroisis
  paralliliAthroisi(tableW4);

}




//phasi ergasias
async void phaseW3()
{
  int i;
  int apotelesmaW3;



  if($!=0 && $!=3 && $!=4)
  {
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;
     }
     barrier;

  phaseW4(tableW3);

  elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
  int i;


  phaseW4(tableW2); barrier;



barrier;
```

```
 ///if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
    ////{
      ///pprintf("\nTo neo apotelesma einai..\n");
     /// pprintf("_____\n");
    ////}
 ///barrier;



  ///for(i=1;i<2*(__STARTED_PROCS__-1);i++)
   /// {
     /// if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
///           {
            ///  pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
///           }
   /// }
  ///barrier;

  elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;


 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW2[i]=0;
      tableEpeksergaston[i]=0;
    }  tableW2[1]=tableW3[1];
 barrier;
```

```
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
          for(j=1;j<128;j++)
           {
            tableEpeksergaston2[i][j]=0;
           }
    }
   barrier;

   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
    {
          if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
           {
            tableW2[prosorini*2]=0;
           }
          else
           {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
           }

          tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
          prosorini=prosorini+1;
    }

barrier;


   /// if($==0 && flagW2==0)
   /// {
////        pprintf("\nTo apotelesma tou w2 einai...\n");
///         pprintf("_____________________________________\n");
///         }
  ///  barrier;
  ///  for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   ///   {

///         if($==0 && flagW2==0)
////        {
            ///    pprintf("|%d| stin thesi %d \n",tableW2[i],i);
            ///    }
   ///   }

  ///  barrier;
    flagW2=1;




   p=1;

//Anathesi epeksergaston oi opoioi den exoun
//katareysei stis theseis
//tou pinaka pou exoume tin timi 0
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
```

```
   start{
   while(topicProcessor!=1)
    {
          tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
          tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

          topicProcessor=topicProcessor/2;



      if ((tableEpeksergaston[p*2+1])!=0 )
          {

            p=(p*2)+1;
          }
else      if((tableEpeksergaston[p*2])!=0)
          {
            p=p*2;
          }


    }
   }

   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);

  //// if($==0 && flagE==0)
   /// {
          ///pprintf("\nOi epeksergastes anatithente os eksis...\n");
          ///pprintf("_____________________________________\n");
   ///  }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
///       if($==0 && flagE==0)
          ////{
            /// pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

          //// }


                if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
          {
           copyI=i;
            leftEpeksergastes=tableEpeksergaston[copyI];
            if(i>=(__STARTED_PROCS__-1)){

              break;

            }


          }
    }
```

```
   barrier;
   flagE=1;



//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
   rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
     {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
            {
             i++;
            }
          if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI][j]=i;
             j++;
             leftEpeksergastes--;
             i++;
             copy=i;
            }
     }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
   {
    while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
      {
    i++;
      }
    if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI+1][j]=i;
             j++;
             rightEpeksergastes--;
             copy=i;
             i++;
            }
   }  barrier;
//// if($==0)
  ///{
  ///  pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
  /// }
 ///for(i=1;i<2*(__STARTED_PROCS__-1);i++)
  /// {
  ///  for(j=1;j<128;j++)
  ///    {
          /// if(tableEpeksergaston2[i][j]!=0)
           ///  {
            ///   if($==0)
             ///    {
```

```
                    /// pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d
\n",tableEpeksergaston2[i][j],i);
              ///     }
             ///  }
   ///    }
  /// }

}

//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;


  flag=1;
  flag2=1;


 //Arxikopoiisi pinakon
   if($==0)
{
     for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
           {

             tableW1[i]=0;
           }


     for(i=1;i<128;i++)
           {
             diathesimoiEpeksergastes[i]=0;
           }


        }

 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($==1)
    {
     tableW1[$+(__STARTED_PROCS__-1)-1]=1;


   }

   barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
```

```
 paralliliAthroisi(tableW1);

 barrier;

apotelesmaW1=tableW1[1];




 if( $==1)
 {
    diathesimoiEpeksergastes[$]=1;
    }


 barrier;

  flagW1=1;

 flag=2;

}



async int main(void)
{
 int table[10000];
 int i;
 pr int flagW3=0;


 for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
    tableW3[i]=0;
  } barrier;

   for(i=1;i<100;i++)
   {
   dentriko_vima_Epeksergastes[i]=0;
    dentriko_vima_Xronos[i]=0;
  }


   barrier;


 //Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
 start{
 initTracing(100000);
 startTracing();
 starttime=getct();
 phaseW3();
 stoptime=getct();
 stopTracing();
 writeTraceFile("sum.trv","sum");
 }
```

```
 barrier;

 //To apotelesma toy pinaka meta apo tin phasi W3 kai tin phasi W4
 if($==0 && flagW3==0)
   {
    pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai\n");
    pprintf("_____________________________\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
          {
           pprintf("|%d| tin thesi %d\n",tableW3[i],i);
          }

    flagW3=1;
   }
 barrier;

 //An esto kai enas epeksergastis exei katarevsei tote ekteleitai
 //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
 //theoreite oti ekteloume ena dentriko bima

start{
 initTracing(100000);
 startTracing();}

i=1;
 while(elegxos!=(__STARTED_PROCS__-1))
   {
  starttime2=getct();
     phaseW1();  barrier;

   phaseW2();   barrier;
    flag=2;
    phase_W3(); barrier;
    flagPinaka=2;

stoptime2=getct();

    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
 i++;


}


//if($==0){



  for(i=1;i<100;i++){

  if(dentriko_vima_Epeksergastes[i]!=0){
  sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
 sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
  }
 else{break;}
  }

  pprintf("\n\n");
```

```
            pprintf("=============================================================\n");
  // printf("CPU Cycles                :: %d\n",stoptime-starttime+stoptime2-starttime2);
   pprintf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
   pprintf("Kostos apo Xrono           :: %d\n",(i-1)*(stoptime2-starttime2));
   pprintf("Kostos apo Epeksergastes   :: %d\n",sinolikoKostosEpeksergastes);
   pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
            pprintf("=============================================================\n");
            pprintf("\n\n");

//}
  barrier;


  return(0);
}
```

A.10 Αλγόριθμος W :: Σενάριο 4

```
/*
================================================================================
*/
/* W_AlgorithmSenario7.c                                        */
/*                                                              */
/*Algorithmos W:                                                */
/*                                                              */
/*Xrisimopoiei ena dianisma megethous 2n-1                      */
/*O algorithmos ektelei ena brogxo 4on faseon                   */
/*eos otou lithei to provlima.                                  */
/* Apo yin arxi katarreoun kai oi n-1 epeksergastes                                    */
/*kai paramenei o epejergasthw me id=5 gia na oloklirosei ton
*/
/*algorithmo                                                    */
/* Panagiota Nicoloaou                                          */
/*                                                              */
/*
================================================================================
*/


#include <math.h>
#include <io.h>
#include <fork.h>


sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh int starttime;
sh int stoptime;
sh int starttime2;
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
sh int sinolikoKostosEpeksergastes=0;
```

```
//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;


  j=2;
m=1;
 pros=1;

  if(flag==2)
    {
     j=2;
     if($!=0 && $==5)
            {
             if(tableW2[1]< (__STARTED_PROCS__-1))
               {
                while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[copyI][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

                if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                      {
                       id=copyI/2;

                       tableW2[copyI]=1;
                      }

                barrier;

                m=1;
                flag=2;
                while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[(copyI+1)][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

                if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                      {
```

```
                    id=copyI/2+1;
    tableW2[copyI+1]=1;

                  }
              barrier;
              n=id;


              while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                  {
                   tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                   n=n/2;

          }

              flag2=2;
              tableW2[1]=tableW2[2]+tableW2[3];

            }
          }
 }barrier;


 while(tableW[1]==0 && flag2!=2)
   {


    if($!=0 && $==5)
{
if(pros==1 ){


 n=($+(__STARTED_PROCS__-1)-1);
          n=n/(j/2);

          if(tableW[n/2]==0)
            {
              n=n/2;
              tableW[n]=tableW[n*2]+tableW[2*n+1];
              pros=2;


            }

}
 else if(pros==2)
{
 fail=1;
          n=($+(__STARTED_PROCS__-1)-1);
          n=n/(j/2);

          if(tableW[n/2]==0)
            {
              n=n/2;
              tableW[n]=tableW[n*2]+tableW[2*n+1];
              pros=2;
            }
```

```
}
            j=j*2;
            }  barrier;
 }

 apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
 int i;
  //Kalesma algorithou parallilis Athroisis
 paralliliAthroisi(tableW4);

}




//phasi ergasias
async void phaseW3()
{
 int i;
 int apotelesmaW3;



 if($!=0 && $==5)
 {
    tableW3[$+(__STARTED_PROCS__-1)-1]=1;
    }
    barrier;

 phaseW4(tableW3);

 elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
 int i;


 phaseW4(tableW2); barrier;



barrier;


 ///if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
```

```
  ///  {
   ///   pprintf("\nTo neo apotelesma einai..\n");
  ///    pprintf("_____________________________\n");
 ///   }
//// barrier;



 /// for(i=1;i<2*(__STARTED_PROCS__-1);i++)
  ///  {
   ///   if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
///           {
///             pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
///           }
 ///   }
 /// barrier;

  elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;


 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW2[i]=0;
      tableEpeksergaston[i]=0;
    }  tableW2[1]=tableW3[1];
 barrier;


    for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
      {
            for(j=1;j<128;j++)
              {
```

```
            tableEpeksergaston2[i][j]=0;
           }
     }
   barrier;

   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
    {
          if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
           {
            tableW2[prosorini*2]=0;
           }
          else
           {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
           }

          tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
          prosorini=prosorini+1;
     }

barrier;


///    if($==0 && flagW2==0)
 ///   {
///          pprintf("\nTo apotelesma tou w2 einai...\n");
///          pprintf("_____________________________\n");
///          }
  ///  barrier;
   /// for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   ///   {

//           if($==0 && flagW2==0)
//           {
//              pprintf("|%d| stin thesi %d \n",tableW2[i],i);
///             }
  ///    }

 ///   barrier;
    flagW2=1;




    p=1;

//Anathesi epeksergaston oi opoioi den exoun
//katareysei stis theseis
//tou pinaka pou exoume tin timi 0
    tableEpeksergaston[1]=tableW1[1];

    topicProcessor=(__STARTED_PROCS__-1);
    start{
    while(topicProcessor!=1)
     {
```

```
            tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
            tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

            topicProcessor=topicProcessor/2;



       if ((tableEpeksergaston[p*2+1])!=0 )
             {

               p=(p*2)+1;
             }
else         if((tableEpeksergaston[p*2])!=0)
             {
               p=p*2;
             }


     }
   }

   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);

///    if($==0 && flagE==0)
 ///     {
///          pprintf("\nOi epeksergastes anatithente os eksis...\n");
///          pprintf("_____________________________________\n");
 ///     }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {
///          if($==0 && flagE==0)
///            {
///               pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

            //// }



                    if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
            {
             copyI=i;
               leftEpeksergastes=tableEpeksergaston[copyI];
               if(i>=(__STARTED_PROCS__-1)){

                 break;

               }


            }
     }


   barrier;
   flagE=1;
```

```
//Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis

   rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
    {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
           {
            i++;
           }
          if( diathesimoiEpeksergastes[i]==1)
           {
            tableEpeksergaston2[copyI][j]=i;
            j++;
            leftEpeksergastes--;
            i++;
            copy=i;
           }
    }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
   while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
     {
    i++;
     }
    if( diathesimoiEpeksergastes[i]==1)
           {
            tableEpeksergaston2[copyI+1][j]=i;
            j++;
            rightEpeksergastes--;
            copy=i;
            i++;
           }
  }  barrier;
 ///if($==0)
 ///  {
 ///    pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
 ///  }
/// for(i=1;i<2*(__STARTED_PROCS__-1);i++)
 ///  {
  ///   for(j=1;j<128;j++)
  ///     {
            /// if(tableEpeksergaston2[i][j]!=0)
            ///   {
            ///     if($==0)
            ///       {
///                         pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d \n",tableEpeksergaston2[i][j],i);
            ///       }
            ///   }
```

```
///      }
///  }

}

//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;


  flag=1;
  flag2=1;

//Arxikopoiisi pinakon
   if($==0)
{
     for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
            {
              // diathesimoiEpeksergastes[i]=0;
              tableW1[i]=0;
            }


     for(i=1;i<128;i++)
            {
              diathesimoiEpeksergastes[i]=0;
            }


        }

 barrier;



//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($==5)
    {
     tableW1[$+(__STARTED_PROCS__-1)-1]=1;


   }

   barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei

  paralliliAthroisi(tableW1);

  barrier;
```

```
 apotelesmaW1=tableW1[1];




  if( $==5)
  {
     diathesimoiEpeksergastes[$]=1;
     }


  barrier;

   flagW1=1;

  flag=2;

}



async int main(void)
{
  int table[10000];
  int i;
  pr int flagW3=0;

 for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW3[i]=0;
    }

    for(i=1;i<100;i++)
    {
     dentriko_vima_Epeksergastes[i]=0;
      dentriko_vima_Xronos[i]=0;
    }


     barrier;


  //Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
  start{
  initTracing(100000);
  startTracing();
  starttime=getct();
  phaseW3();
  stoptime=getct();
  stopTracing();
  writeTraceFile("sum.trv","sum");
  }
  barrier;

  phaseW3(); barrier;
  if($==0 && flagW3==0)
    {
```

```
    pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai\n");
    pprintf("_____\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
          {
           pprintf("|%d| tin thesi %d\n",tableW3[i],i);
          }

    flagW3=1;
   }
 barrier;

 start{
 initTracing(100000);
 startTracing();}

i=1;
 while(elegxos!=(__STARTED_PROCS__-1))
  {
   starttime2=getct();
    phaseW1();  barrier;

  phaseW2();   barrier;
   flag=2;
   phase_W3(); barrier;
   flagPinaka=2;

stoptime2=getct();

    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
 i++;

}

 barrier;

  for(i=1;i<100;i++){

  if(dentriko_vima_Epeksergastes[i]!=0){
  sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
 sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
  }
 else{break;}
  }

  pprintf("\n\n");
          pprintf("=========================================================\n");
 // printf("CPU Cycles               :: %d\n",stoptime-starttime+stoptime2-starttime2);
  pprintf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
  pprintf("Kostos apo Xrono          :: %d\n",(i-1)*(stoptime2-starttime2));
  pprintf("Kostos apo Epeksergastes  :: %d\n",sinolikoKostosEpeksergastes);
  pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
          pprintf("=========================================================\n");
          pprintf("\n\n");

//}
 barrier;
 return(0);
```

A.11 Αλγόριθμος W :: Σενάριο 5

```
/*
================================================================================
*/
/* W_AlgorithmSenarioRandom.c                                          */
/*                                                                     */
/*Algorithmos W:                                                       */
/*                                                                     */
/*Xrisimopoiei ena dianisma megethous 2n-1                             */
/*O algorithmos ektelei ena brogxo 4on faseon                          */
/*eos otou lithei to provlima.                                         */
/*Ayto einai to scenario opou exoume dio elattomatikes kipselides
 */
/*Opoios epeksergastis paei stis theseis me aritmo 20 kai 21           */
/*katarreewi ektos apo ton teleytaio pou prepei na meinei
*/
/*aparaitita gia na oloklirosei ton algorithmo                                    */
/*                                                                     */
/* Panagiota Nicoloaou                                                 */
/*                                                                     */
/*
================================================================================
*/



#include <math.h>
#include <io.h>
#include <fork.h>
#include <stdlib.h>

sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh int FailProcessors[128];
sh int countProcessor=0;
sh int sumKostos=0;
sh int kostos[1000];
sh int loop=0;
sh int starttime;
sh int stoptime;
sh int starttime2;
```

```
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
sh int sinolikoKostosEpeksergastes=0;



//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;
pr int metritis=1;

  j=2;
m=1;
 pros=1;


  if(flag==2)
    {
     j=2;
     if(FailProcessors[$]!=1 && $!=0)
            {

              if(tableW2[1]< (__STARTED_PROCS__-1))
                {
                  while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                        {
                          if(tableEpeksergaston2[copyI][m]==0)
                            {
                              flag=3;
                            }
                          m++;
                        }

                  if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                        {
                          id=copyI/2;

     if((copyI!=20 && copyI!=21) || countProcessor>=13){

                          tableW2[copyI]=1;
            }
    else {FailProcessors[$]=1; countProcessor++;

    }

    }
```

```
                barrier;

                m=1;
                flag=2;
                while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                      {
                       if(tableEpeksergaston2[(copyI+1)][m]==0)
                         {
                          flag=3;
                         }
                       m++;
                      }

                if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                      {


    id=copyI/2+1;

   if((copyI+1!=20 && copyI+1!=21) || countProcessor>=13){
    tableW2[copyI+1]=1;

                       }
    else{

    FailProcessors[$]=1;
    countProcessor++;


  }
            }
                barrier;
                if(FailProcessors[$]!=1 ){
      n=id;


                while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                       {
                       if((n!=20 && n!=21 )|| countProcessor>=13){
                         tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                         n=n/2;
}

else {FailProcessors[$]=1;countProcessor++;}

            }
                if(FailProcessors[$]!=1){
                flag2=2;
                tableW2[1]=tableW2[2]+tableW2[3];
                }
               }
    }
            }
  }barrier;
```

```
  while(tableW[1]==0 && flag2!=2)
    {

     if($!=0 && FailProcessors[$]!=1 )
{
if(pros==1 ){

             n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

              if(((n/2)!=20 && (n/2)!=21) || countProcessor>=13){
              if(tableW[n/2]==0)
                {
                  n=n/2;
                  tableW[n]=tableW[n*2]+tableW[2*n+1];
                  pros=2;
             metritis++;

                }
}
else{FailProcessors[$]=1;countProcessor++;}


}


 else if(pros==2 )
{
if(metritis>=3)
{if(FailProcessors[$]!=1)
{

  fail=1;
              n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);

  metritis++;
   if(((n/2)!=20 && (n/2)!=21)|| countProcessor>=13){
              if(tableW[n/2]==0)
                {
                  n=n/2;
                  tableW[n]=tableW[n*2]+tableW[2*n+1];
                  pros=2;
                }
    }
    else{FailProcessors[$]=1;countProcessor++;}

    }
}

else if(metritis<3)
{fail=1;
              n=($+(__STARTED_PROCS__-1)-1);
              n=n/(j/2);
              metritis++;
              if(((n/2)!=20 && (n/2)!=21) || countProcessor>=13){
              if(tableW[n/2]==0)
                {
                  n=n/2;
```

```
                tableW[n]=tableW[n*2]+tableW[2*n+1];
                pros=2;
               }

     }
     else {FailProcessors[$]=1;countProcessor++;}
     }
}
             j=j*2;
            }
   barrier;
  }

  apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;

  paralliliAthroisi(tableW4);

}




//phasi ergasias
async void phaseW3()
{
  int i;
  int apotelesmaW3;



  if($!=0 && FailProcessors[$]!=1)
  {
  if(($+(__STARTED_PROCS__-1)-1)!=20 &&  ($+(__STARTED_PROCS__-1)-1)!=21){
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;
     }
     else FailProcessors[$]=1;

   }
     barrier;

  phaseW4(tableW3);

  elegxos=apotelesmaW4;

}

//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
  int i;
```

```
  start initTracing(100000);
  start startTracing();
  starttime2=getct();

  phaseW4(tableW2); barrier;

   stoptime2=getct();
  start stopTracing();
  start writeTraceFile("sum.trv","sum");

barrier;

if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
    {
     pprintf("\nTo neo apotelesma einai..\n");
      pprintf("_____________________________\n");
    }

 barrier;



for(i=1;i<2*(__STARTED_PROCS__-1);i++)
    {
      if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
             {
               pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
             }
    }
  barrier;

  elegxos=apotelesmaW4;

}




//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;


 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];
```

```
   }

 }
  barrier;
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
     tableW2[i]=0;
     tableEpeksergaston[i]=0;
   }  tableW2[1]=tableW3[1];
 barrier;


   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {
          for(j=1;j<128;j++)
            {
              tableEpeksergaston2[i][j]=0;
            }
     }
   barrier;

   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
     {
          if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
            {
              tableW2[prosorini*2]=0;
            }
          else
            {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
            }

          tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
          prosorini=prosorini+1;
     }

barrier;


   if($==0 && flagW2==0)
   {
          pprintf("\nTo apotelesma tou w2 einai...\n");
          pprintf("_____________________________\n");
          }
   barrier;

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {

          if($==0 && flagW2==0)
          {
             pprintf("|%d| stin thesi %d \n",tableW2[i],i);
             }
     }

   barrier;
   flagW2=1;
```

```
   p=1;

   //Anathesi epeksergaston
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
   start{
   while(topicProcessor!=1)
     {
          tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
          tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

          topicProcessor=topicProcessor/2;



      if ((tableEpeksergaston[p*2+1])!=0 )
            {

              p=(p*2)+1;
            }
else       if((tableEpeksergaston[p*2])!=0)
            {
              p=p*2;
            }


     }
   }

   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);

   if($==0 && flagE==0)
     {
          pprintf("\nOi epeksergastes anatithente os eksis...\n");
          pprintf("_____________________________________\n");
     }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {

          if($==0 && flagE==0)
            {
               pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

            }



                  if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
          {
           copyI=i;
             leftEpeksergastes=tableEpeksergaston[copyI];
             if(i>=(__STARTED_PROCS__-1)){

               break;
```

```
            }


          }
     }


   barrier;
   flagE=1;



  //Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
   rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
     {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
            {
             i++;
            }
          if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI][j]=i;
             j++;
             leftEpeksergastes--;
             i++;
             copy=i;
            }
     }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
    while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
      {
    i++;
      }
    if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI+1][j]=i;
             j++;
             rightEpeksergastes--;
             copy=i;
             i++;
            }
  }  barrier;

 if($==0)
  {
    pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
```

```
  }
for(i=1;i<2*(__STARTED_PROCS__-1);i++)
  {
   for(j=1;j<128;j++)
     {
          if(tableEpeksergaston2[i][j]!=0)
            {

             if($==0)
               {
                    pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d \n",tableEpeksergaston2[i][j],i);
               }


    }
     }
 }

}

//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

 int i;


 flag=1;
 flag2=1;


//Arxikopoiisi pinakon
  if($==0)
{
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
          {

            tableW1[i]=0;
          }


    for(i=1;i<128;i++)
          {
            diathesimoiEpeksergastes[i]=0;
          }


       }

barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($!=0 && FailProcessors[$]!=1 )
   {
```

```
    tableW1[$+(__STARTED_PROCS__-1)-1]=1;


   }

   barrier;


//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei

  paralliliAthroisi(tableW1);

  barrier;

 apotelesmaW1=tableW1[1];
kostos[loop]=apotelesmaW1;




  if(FailProcessors[$]!=1)
  {
     diathesimoiEpeksergastes[$]=1;
     }


  barrier;

   flagW1=1;

  flag=2;

}




async int main(void)
{
  int table[10000];
  int i;
  pr int flagW3=0;

 for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
      tableW3[i]=0;
    }


      for(i=1;i<100;i++)
    {
     dentriko_vima_Epeksergastes[i]=0;
      dentriko_vima_Xronos[i]=0;
    }



 for(i=0;i<(__STARTED_PROCS__-1);i++){
```

```
 FailProcessors[i]=0;

 }

  barrier;


//Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
  start{
  initTracing(100000);
  startTracing();
  starttime=getct();
  phaseW3();
  stoptime=getct();
  stopTracing();
  writeTraceFile("sum.trv","sum");
  }
  barrier;

  if($==0 && flagW3==0)
    {
     //printf("stoptime -starttime %u kai %d mse \n",stoptime-starttime,(stoptime-starttime)*4);
  printf("\n\n\n\n\n\n");
      pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai\n");
      pprintf("_____\n");
      for(i=1;i<2*(__STARTED_PROCS__-1);i++)
             {
              pprintf("|%d| tin thesi %d\n",tableW3[i],i);

  }

      flagW3=1;
    }

  barrier;

  //An esto kai enas epeksergastis exei katarevsei tote ekteleitai
  //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
  //theoreite oti ekteloume ena dentriko bima
 start{
  initTracing(100000);
  startTracing();}

 i=1;



  while(elegxos!=(__STARTED_PROCS__-1))
    {
   starttime2=getct();
       phaseW1();  barrier;

    phaseW2(); barrier;



      flag=2;
```

```
    phase_W3(); barrier;

    flagPinaka=2;



 //kostos[loop]=kostos[loop]*((stoptime-starttime)*4);


//loop++;
stoptime2=getct();

    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
  i++;




}
start{
    stopTracing();
    writeTraceFile("sum.trv","sum");
    }



//for(i=0;i<loop;i++){

 //sumKostos=kostos[i]+sumKostos;

 //}

 for(i=1;i<100;i++){

  if(dentriko_vima_Epeksergastes[i]!=0){
  sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
 sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
  }
 else{break;}
  }

  pprintf("\n\n");
            pprintf("=============================================================\n");
 // printf("CPU Cycles               :: %d\n",stoptime-starttime+stoptime2-starttime2);
  pprintf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
  pprintf("Kostos apo Xrono          :: %d\n",(i-1)*(stoptime2-starttime2));
  pprintf("Kostos apo Epeksergastes  :: %d\n",sinolikoKostosEpeksergastes);
  pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
            pprintf("=============================================================\n");
            pprintf("\n\n");


  barrier;


  return(0);
}
```

A.12 Αλγόριθμος W :: Σενάριο 6

```
/*
================================================================================
*/
/* W_AlgorithmSenarioRandom.c                                                  */
/*                                                                             */
/*Algorithmos W:                                                               */
/*                                                                             */
/*Xrisimopoiei ena dianisma megethous 2n-1                                     */
/*O algorithmos ektelei ena brogxo 4on faseon                                  */
/*eos otou lithei to provlima.                                                 */
/*Se ayto to scenario epilegete random mia pithanotita gia ton
*/
/* kathe epeksergasti se kathe bima
 */
/*An ayth i pithanotita einai megaliteri apo mia global timi
*/
/*tote o epeksergastis tha katarreysei an oxi tote tha sinexisei
*/
/*tin ektelesi toy algorithmou                                                 */
/* Panagiota Nicoloaou                                                         */
/*                                                                             */
/*
================================================================================
*/

#include <math.h>
#include <io.h>
#include <fork.h>
sh float pithanotita1=0.4;
sh float pithanotita2=0.2;
sh float pithanotita3=1;
sh float pithanotita4=1;
sh float pithanotita5=1;
sh int kostos[1000];
sh int apotelesmaW1;
sh int apotelesmaW4;
sh int elegxos;
sh int tableW3[1000];
sh int tableW1[1000];
sh int leftEpeksergastes;
sh int rightEpeksergastes;
sh int copyI;
sh int copy;
sh int flag=1;
sh int flag2=1;
sh int epeksergastes;
sh int diathesimoiEpeksergastes[128];
sh int tableEpeksergaston2[1000][128];
sh int tableW2[1000];
sh int flagPinaka=1;
sh int flagE=0;
sh int flagW2=0;
sh int fail=0;
sh int flagW1=0;
sh float randomP[128];
sh int loop=0;
sh int sumKostos=0;
sh int starttime;
```

```
sh int stoptime;
sh int starttime2;
sh int stoptime2;
sh int athrismaKostous=1;
sh int dentriko_vima_Epeksergastes[100];
sh int dentriko_vima_Xronos[100];
sh int sinolikoKostosXrono=0;
sh int sinolikoKostosEpeksergastes=0;

//Algorithmos parrallilis athroisis
//Atroizei ta stoixeia toy pinaka
//Xrisimopoihte apo tis faseis W1 kai W4
async void paralliliAthroisi(int tableW[])
{
  int i;
  pr int j;
  pr int n;
  pr int a;
  pr int b;
  pr int m;
  pr int pros;
  pr int id;


  j=2;
m=1;
 pros=1;

  if(flag==2)
    {
     j=2;
     if($!=0 && randomP[$]<=pithanotita1 && randomP[$]<=pithanotita2 &&
randomP[$]<=pithanotita3 && randomP[$]<=pithanotita4)//&& $!=1 && $!=2 && $!=3 && $!=4)
            {
             if(tableW2[1]< (__STARTED_PROCS__-1) )
               {
                while(tableEpeksergaston2[copyI][m]!=$ && flag!=3)
                       {
                        if(tableEpeksergaston2[copyI][m]==0)
                          {
                           flag=3;
                          }
                        m++;
                       }

                if(tableEpeksergaston2[copyI][m]==$ && flag2!=2)
                       {id=copyI/2;

                        tableW2[copyI]=1;
                       }

                barrier;

                m=1;
                flag=2;
                while(tableEpeksergaston2[(copyI+1)][m]!=$ && flag!=3)
                       {
                        if(tableEpeksergaston2[(copyI+1)][m]==0)
                          {
```

```
                    flag=3;
                  }
                 m++;
                }

            if(tableEpeksergaston2[copyI+1][m]==$ && flag2!=2)
                { id=copyI/2+1;

                 tableW2[copyI+1]=1;

                }
            barrier;
            n=id;


            while((tableW2[n]!=tableW2[n*2]+tableW2[2*n+1]) && n>=1 &&
n<=((__STARTED_PROCS__-1)-1))
                {
                 tableW2[n]=tableW2[n*2]+tableW2[2*n+1];
                 n=n/2;


        }

            flag2=2;
            tableW2[1]=tableW2[2]+tableW2[3];

          }
        }
 }barrier;


 while(tableW[1]==0 && flag2!=2)
   {
    if($!=0 && randomP[$]<=pithanotita1 && randomP[$]<=pithanotita2 &&
randomP[$]<=pithanotita3 && randomP[$]<=pithanotita4)
{
if(pros==1 ){



 n=($+(__STARTED_PROCS__-1)-1);
          n=n/(j/2);

          if(tableW[n/2]==0)
            {
             n=n/2;
             tableW[n]=tableW[n*2]+tableW[2*n+1];
             pros=2;


            }

}
 else
{
 fail=1;
          n=($+(__STARTED_PROCS__-1)-1);
          n=n/(j/2);
```

```
            if(tableW[n/2]==0)
              {
                n=n/2;
                tableW[n]=tableW[n*2]+tableW[2*n+1];
                pros=2;
              }



}
            j=j*2;
            }  barrier;
  }

  apotelesmaW4=tableW[1];

}


//Phasi aksiologisis proodou
async void phaseW4(int tableW4[])
{
  int i;

  paralliliAthroisi(tableW4);

}




//phasi ergasias
async void phaseW3()
{
  int i;
  int apotelesmaW3;



  if($!=0 &&  randomP[$]<=pithanotita1 /*&& randomP[$]<=pithanotita2 */&&
randomP[$]<=pithanotita3 && randomP[$]<=pithanotita4)
  {
     tableW3[$+(__STARTED_PROCS__-1)-1]=1;


     }
     barrier;

  phaseW4(tableW3);

  elegxos=apotelesmaW4;

}


//phasi ergasias (Paromoia me thn phasi W3 alla me kapoies parallages)
async void phase_W3()
{
```

```
  int i;


  phaseW4(tableW2); barrier;



barrier;


 if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
    {
      pprintf("\nTo neo apotelesma einai..\n");
      pprintf("_____________________________\n");
    }
 barrier;



  for(i=1;i<2*(__STARTED_PROCS__-1);i++)
    {
      if( flag2==2 && $==0 && apotelesmaW4==(__STARTED_PROCS__-1))
             {
               pprintf("|%d| stin thesi  %d\n",tableW2[i],i);
             }
    }

  barrier;

  elegxos=apotelesmaW4;

}



//Anoxi sfalmaton meso isozigismenis anathesis
// epeksergaston se stoixeia tis listas
async void phaseW2()
{
  int tableEpeksergaston[1000];
  pr int p;
int i;
  int j;
  int prosorini;
  int topicProcessor;


 prosorini=1;
  topicProcessor=(__STARTED_PROCS__-1);

//Arxikopoiisi pinakon
  if( flagPinaka==2 && $==0){
    for(i=1;i<(2*(__STARTED_PROCS__-1));i++){
  tableW3[i]=tableW2[i];

    }

  }
  barrier;
```

```
for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
   {
     tableW2[i]=0;
     tableEpeksergaston[i]=0;
   }  tableW2[1]=tableW3[1];
 barrier;


   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
          for(j=1;j<128;j++)
            {
              tableEpeksergaston2[i][j]=0;
            }
    }
   barrier;

   while(((prosorini*2)+1)<=((2*(__STARTED_PROCS__-1))-1))
    {
          if((tableW3[prosorini*2]+tableW3[(prosorini*2)+1])==0)
            {
              tableW2[prosorini*2]=0;
            }
          else
            {

tableW2[prosorini*2]=(tableW2[prosorini]*tableW3[prosorini*2])/(tableW3[prosorini*2]+tableW3[proso
rini*2+1]);
            }

          tableW2[prosorini*2+1]=tableW2[prosorini]-tableW2[prosorini*2];
          prosorini=prosorini+1;
    }

barrier;


   if($==0 && flagW2==0)
   {
          pprintf("\nTo apotelesma tou w2 einai...\n");
          pprintf("_____________________________________\n");
          }
   barrier;
   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
          if($==0 && flagW2==0)
          {
             pprintf("|%d| stin thesi %d \n",tableW2[i],i);
             }
    }

   barrier;
   flagW2=1;




   p=1;
```

```
   //Anathesi epeksergaston
   tableEpeksergaston[1]=tableW1[1];

   topicProcessor=(__STARTED_PROCS__-1);
   start{
   while(topicProcessor!=1)
     {
          tableEpeksergaston[p*2]=(tableEpeksergaston[p]*((topicProcessor/2)-
tableW2[p*2]))/(topicProcessor-tableW2[p]);
          tableEpeksergaston[p*2+1]= tableEpeksergaston[p]- tableEpeksergaston[p*2];

          topicProcessor=topicProcessor/2;



      if ((tableEpeksergaston[p*2+1])!=0 )
           {

             p=(p*2)+1;
           }
else       if((tableEpeksergaston[p*2])!=0)
           {
            p=p*2;
           }


     }
   }

   leftEpeksergastes=(__STARTED_PROCS__-1);
   rightEpeksergastes=(__STARTED_PROCS__-1);


   if($==0 && flagE==0)
     {
          pprintf("\nOi epeksergastes anatithente os eksis...\n");
          pprintf("_____________________________________\n");
     }

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
     {

          if($==0 && flagE==0)
           {
              pprintf("arithmos epeksergaston |%d| stin thesi %d\n",tableEpeksergaston[i],i);

           }



                  if(tableEpeksergaston[i]!=0 && tableEpeksergaston[i]<(__STARTED_PROCS__-1) )
          {
            copyI=i;
              leftEpeksergastes=tableEpeksergaston[copyI];
              if(i>=(__STARTED_PROCS__-1)){

                break;
```

```
                }


            }
}


   barrier;
   flagE=1;



  //Analoga me to posoi epejergastes anatithontai se mia sigkekrimeni thesi
//briskoume se poia thesi tha paei o kathe epeksergastis
   rightEpeksergastes=tableEpeksergaston[copyI+1];


   i=1;
   j=1;
   while(leftEpeksergastes!=0)
    {
          while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
           {
            i++;
           }
          if( diathesimoiEpeksergastes[i]==1)
           {
            tableEpeksergaston2[copyI][j]=i;
            j++;
            leftEpeksergastes--;
            i++;
            copy=i;
           }
    }


   i=copy;
   j=1;
 while(rightEpeksergastes!=0)
  {
   while (i<128 &&  diathesimoiEpeksergastes[i]!=1)
     {
    i++;
     }
   if( diathesimoiEpeksergastes[i]==1)
            {
             tableEpeksergaston2[copyI+1][j]=i;
             j++;
             rightEpeksergastes--;
             copy=i;
             i++;
            }
  }  barrier;
 if($==0)
  {
   pprintf("-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n");
  }
```

```
 for(i=1;i<2*(__STARTED_PROCS__-1);i++)
   {
    for(j=1;j<128;j++)
      {
            if(tableEpeksergaston2[i][j]!=0)
              {
               if($==0)
                 {
                       pprintf("\n-O epeksergastis (%d) tha paei sin thesi %d \n",tableEpeksergaston2[i][j],i);
                 }
              }
      }
   }

}

//Evresi sfalmaton meso katametrisis epeksergaston
//s'ayth tin asi katametrite to plithos ton epeksergaston
//pou den exei katareysei
async void phaseW1(){

  int i;


  flag=1;
  flag2=1;

 //Arxikopoiisi pinakon
   if($==0)
{
     for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
            {

              tableW1[i]=0;
            }


     for(i=1;i<128;i++)
            {
              diathesimoiEpeksergastes[i]=0;
            }
        }
 barrier;


//Oloi oi epeksergastes ektos apo ton epeksergasti me prosopiko arithmo to 0
//kai tous epeksergastes pou den exoun katareysei
//katagrafoun sta filla tou dentrou tin timi 1 gia na mporesei na ginei i katametrisi
//gia to posoi exoun katareysei.
 if($!=0 && randomP[$]<=pithanotita1 && randomP[$]<=pithanotita2 && randomP[$]<=pithanotita3
&& randomP[$]<=pithanotita4 )//&& $!=1 && $!=2 && $!=3 && $!=4)
    {
     tableW1[$+(__STARTED_PROCS__-1)-1]=1;


   } barrier;
```

```
//Kalesma tou algorithmou tis parallilis athroisis gia ipologismo ton epeksergaston pou den
//exoun katareysei
  paralliliAthroisi(tableW1);

  barrier;

  apotelesmaW1=tableW1[1];

 kostos[loop]=apotelesmaW1;

  if(randomP[$]<=pithanotita1 && randomP[$]<=pithanotita2 && randomP[$]<=pithanotita3 &&
randomP[$]<=pithanotita4 )//&& $!=1 && $!=2 && $!=3 && $!=4)
  {
     diathesimoiEpeksergastes[$]=1;
     }


  barrier;

   flagW1=1;

  flag=2;

}




async int main(void)
{
  int table[10000];
  int i;
  pr int flagW3=0;

   for(i=1;i<(2*(__STARTED_PROCS__-1));i++)
    {
     tableW3[i]=0;
    }


     for(i=1;i<100;i++)
    {
    dentriko_vima_Epeksergastes[i]=0;
     dentriko_vima_Xronos[i]=0;
    }



    barrier;



  for(i=1;i<(__STARTED_PROCS__-1);i++){
  srand(i*100);
   randomP[i]=abs(rand()% 100);
  randomP[i]=randomP[i]/100;
}
  barrier;
```

```
//Ypologismos tou xronou pou xriazetai i phasi W3 na ektelestei ston aslgorithmo xoris sfalmata
 start{
 initTracing(100000);
 startTracing();
 starttime=getct();
 phaseW3();
 stoptime=getct();
 stopTracing();
 writeTraceFile("sum.trv","sum");
 }
 barrier;


 if($==0 && flagW3==0)
   {
    pprintf("\nTo apotelesma meta tin fasi w3 kai w4 einai\n");
    pprintf("_____________________________\n");
    for(i=1;i<2*(__STARTED_PROCS__-1);i++)
          {
           pprintf("|%d| tin thesi %d\n",tableW3[i],i);
printf("Tipose mou ton pinaka %f \n",randomP[i]);
          }

    flagW3=1;
   }
 barrier;

   barrier;

//An esto kai enas epeksergastis exei katarevsei tote ekteleitai
 //o vrogxos ton tessaron faseon kathe ektelesi tou vrogxou
 //theoreite oti ekteloume ena dentriko bima
start{
 initTracing(100000);
 startTracing();}

i=1;

 while(elegxos!=(__STARTED_PROCS__-1))
   {

 starttime2=getct();
 phaseW1();  barrier;


      phaseW2();  barrier;



//kostos[loop]=kostos[loop]*((stoptime-starttime)*4);

       barrier;

       flag=2;
    phase_W3(); barrier;
    flagPinaka=2;

  stoptime2=getct();
```

```
    dentriko_vima_Epeksergastes[i]= apotelesmaW1;
    dentriko_vima_Xronos[i]=stoptime2-starttime2;
 i++;

 // loop++;
}

 start{
    stopTracing();
    writeTraceFile("sum.trv","sum");
    }


barrier;



//if($==0){



  for(i=1;i<100;i++){

  if(dentriko_vima_Epeksergastes[i]!=0){
  sinolikoKostosXrono=sinolikoKostosXrono+dentriko_vima_Xronos[i];
 sinolikoKostosEpeksergastes= sinolikoKostosEpeksergastes+dentriko_vima_Epeksergastes[i];
  }
 else{break;}
  }

  pprintf("\n\n");
            pprintf("============================================================\n");
 // printf("CPU Cycles                :: %d\n",stoptime-starttime+stoptime2-starttime2);
  pprintf("Xronos (se msec)          :: %d\n",(sinolikoKostosXrono+(stoptime-starttime)));
  pprintf("Kostos apo Xrono           :: %d\n",(i-1)*(stoptime2-starttime2));
  pprintf("Kostos apo Epeksergastes   :: %d\n",sinolikoKostosEpeksergastes);
  pprintf("Arithmos dendrikon Bimaton:: %d\n",i-1);
            pprintf("============================================================\n");
            pprintf("\n\n");


 barrier;
 return(0);
}
```

Παράρτημα Β

Δείγματα από εκτελέσεις των διάφορων επιλεγμένων αλγορίθμων

Β.1 Αλγόριθμος παράλληλης άθροισης

Με είσοδο τα παρακάτω δεδομένα τα οποία βρίσκονται αποθηκευμένα μέσα σε αρχείο εκτελείται ο αλγόριθμος παράλληλης άθροισης με 16 επεξεργαστές.

```
16
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
```

Αλγόριθμος παράλληλης άθροισης

```
parralliliAthrisi.c   parralliliAthrisi.o
cs05np1@athos:~/bin/pram/pramsim>pramsim -p 17 -v 1 $EXAMPLES/../parralliliAthri
si
       ***** PRAM-SIMULATOR (v2.0) *****
          >>>>>>>> DAMN FAST <<<<<<<<
       (c) by hirbli & stefran  07.10.94
You have 17 physical processors with 1 vP's each

Relocating file /home/students/cs/2005/cs05np1/../../../../../local/cs021p1/exam
ples/diplomatiki/../parralliliAthrisi... done
Loading file /var/tmp/aaat9aap5.cod...Doing realloc
Doing realloc
 done
PRAM P0 = (p0, v0)> g
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set


_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_
To athroisma ton 16 stoixeion einai :: 256
_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_

CPU Cycles      :: 2127
Xronos (se msec):: 8508
Kostos          :: 136128
=====================================================


EXIT: vp=#0, pc=$0000061d
EXIT: vp=#1, pc=$0000061d
EXIT: vp=#2, pc=$0000061d
EXIT: vp=#3, pc=$0000061d
EXIT: vp=#4, pc=$0000061d
EXIT: vp=#5, pc=$0000061d
EXIT: vp=#6, pc=$0000061d
EXIT: vp=#7, pc=$0000061d
EXIT: vp=#8, pc=$0000061d
EXIT: vp=#9, pc=$0000061d
EXIT: vp=#10, pc=$0000061d
EXIT: vp=#11, pc=$0000061d
EXIT: vp=#12, pc=$0000061d
EXIT: vp=#13, pc=$0000061d
EXIT: vp=#14, pc=$0000061d
EXIT: vp=#15, pc=$0000061d
EXIT: vp=#16, pc=$0000061d
Stop nach 120197 Runden, 6009.850 kIps
061d 18137FFF  POPNG   R6, ffffffff, R1
PRAM P0 = (p0, v0)>
```

B.2 Αλγόριθμος παράλληλης μετάδοσης

Το παρακάτω αποτέλεσμα προέκυψε από την εκτέλεση του αλγορίθμου από 16 επεξεργαστές. Το στοιχείο που μεταδίδεται είναι ο αριθμός 1.

```
6c72 18137FFF  POPNG   R6, ffffffff, R1
PRAM PO = (pO, vO)> q
cs05np1@athos:~/bin/pram/pramsim>clear
cs05np1@athos:~/bin/pram/pramsim>pramsim -p 17 -v 1 $EXAMPLES/../broadcasting
       ***** PRAM-SIMULATOR (v2.0) *****
          >>>>>>>> DAMN FAST <<<<<<<<
       (c) by hirbli & stefran  07.10.94
You have 17 physical processors with 1 vP's each

Relocating file /home/students/cs/2005/cs05np1/../../../../../local/cs021p1/exam
ples/diplomatiki/../broadcasting... done
Loading file /var/tmp/aaaI5aOu5.cod...Doing realloc
Doing realloc
 done
PRAM PO = (pO, vO)> g
#0000# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>#0
000# Dose to plithos ton kipselidon pou theleis na metadothei to stoixeio:
32
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set


CPU Cycles      :: 4077
Xronos (se msec):: 16308
Kostos          :: 260928
=======================================================


EXIT: vp=#0, pc=$00000525
EXIT: vp=#1, pc=$00000525
EXIT: vp=#2, pc=$00000525
EXIT: vp=#3, pc=$00000525
EXIT: vp=#4, pc=$00000525
EXIT: vp=#5, pc=$00000525
EXIT: vp=#6, pc=$00000525
EXIT: vp=#7, pc=$00000525
EXIT: vp=#8, pc=$00000525
EXIT: vp=#9, pc=$00000525
EXIT: vp=#10, pc=$00000525
EXIT: vp=#11, pc=$00000525
EXIT: vp=#12, pc=$00000525
EXIT: vp=#13, pc=$00000525
EXIT: vp=#14, pc=$00000525
EXIT: vp=#15, pc=$00000525
EXIT: vp=#16, pc=$00000525
Stop nach 310281 Runden, 5494.559 kIps
0525 18137FFF  POPNG   R6, ffffffff, R1
PRAM PO = (pO, vO)>
```

```
Stin thesi 1 exei metadothei to stoixeio 1
Stin thesi 2 exei metadothei to stoixeio 1
Stin thesi 3 exei metadothei to stoixeio 1
Stin thesi 4 exei metadothei to stoixeio 1
Stin thesi 5 exei metadothei to stoixeio 1
Stin thesi 6 exei metadothei to stoixeio 1
Stin thesi 7 exei metadothei to stoixeio 1
Stin thesi 8 exei metadothei to stoixeio 1
Stin thesi 9 exei metadothei to stoixeio 1
Stin thesi 10 exei metadothei to stoixeio 1
Stin thesi 11 exei metadothei to stoixeio 1
Stin thesi 12 exei metadothei to stoixeio 1
Stin thesi 13 exei metadothei to stoixeio 1
Stin thesi 14 exei metadothei to stoixeio 1
Stin thesi 15 exei metadothei to stoixeio 1
Stin thesi 16 exei metadothei to stoixeio 1
Stin thesi 17 exei metadothei to stoixeio 1
Stin thesi 18 exei metadothei to stoixeio 1
Stin thesi 19 exei metadothei to stoixeio 1
Stin thesi 20 exei metadothei to stoixeio 1
Stin thesi 21 exei metadothei to stoixeio 1
Stin thesi 22 exei metadothei to stoixeio 1
Stin thesi 23 exei metadothei to stoixeio 1
Stin thesi 24 exei metadothei to stoixeio 1
Stin thesi 25 exei metadothei to stoixeio 1
Stin thesi 26 exei metadothei to stoixeio 1
Stin thesi 27 exei metadothei to stoixeio 1
Stin thesi 28 exei metadothei to stoixeio 1
Stin thesi 29 exei metadothei to stoixeio 1
Stin thesi 30 exei metadothei to stoixeio 1
Stin thesi 31 exei metadothei to stoixeio 1
Stin thesi 32 exei metadothei to stoixeio 1
```

B.3 Αλγόριθμος Write-All με την βοήθεια δύο επεξεργαστών

Ο αριθμός των στοιχείων που θέλουμε να εκτελεστεί ο αλγόριθμος και να μετατραπεί η τιμή του κάθε στοιχείου από 0 σε 1 είναι 8.

```
cs05np1@athos:~/bin/pram/pramsim>pramsim -p 3 -v 1 $EXAMPLES/../Write_all_2p
       ***** PRAM-SIMULATOR (v2.0) *****
          >>>>>>>> DAMN FAST <<<<<<<<
       (c) by hirbli & stefran  07.10.94
You have 3 physical processors with 1 vP's each

Relocating file /home/students/cs/2005/cs05np1/../../../../../local/cs02lp1/examples/diplomatiki/../Write_all_2p... done
Loading file /var/tmp/aaaYhaOy5.cod...Doing realloc
Doing realloc
 done
PRAM P0 = (p0, v0)> g
#0000# Dose to plithos ton stoixeion:
8
#0000#
8
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0001# O epeksergastis 1 eftase sto stoixeio 5
#0002# O epeksergastis 2 eftase sto stoixeio 5
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
|¬P1¬|
cpu cycles       :: 12111
xronos (se msec) :: 48.444
kostos           :: 5
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
|¬P2¬|
cpu cycles       :: 12127
xronos (se msec) :: 48.508
kostos           :: 5
<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
Sinolikos xronos :: 60.619
Sinoliko  kostos :: 10
EXIT: vp=#0, pc=$0000041d
EXIT: vp=#1, pc=$0000041d
EXIT: vp=#2, pc=$0000041d
Stop nach 127947 Runden, 3489.464 kIps
041d 18137FFF  POPNG   R6, ffffffff, R1
PRAM P0 = (p0, v0)>
```

B.4 Αλγόριθμος Χ’

Επιλέχθηκε να εκτελεστεί ο βέλτιστος αλγόριθμος γιατί σ’ αυτόν παρατηρείται σε ποιο μεγάλο βαθμό το φαινόμενο του ασυγχρονισμού.
Το παρακάτω αποτέλεσμα προέκυψε από την εκτέλεση του αλγορίθμου από 2 επεξεργαστές σε 8 στοιχεία. Όπως βλέπουμε ο επεξεργαστής 1 από κάποια φάση και μετά γίνεται ποιο γρήγορος από τον επεξεργαστή 2 και γράφει σε περισσότερες κυψελίδες την τιμή 1.

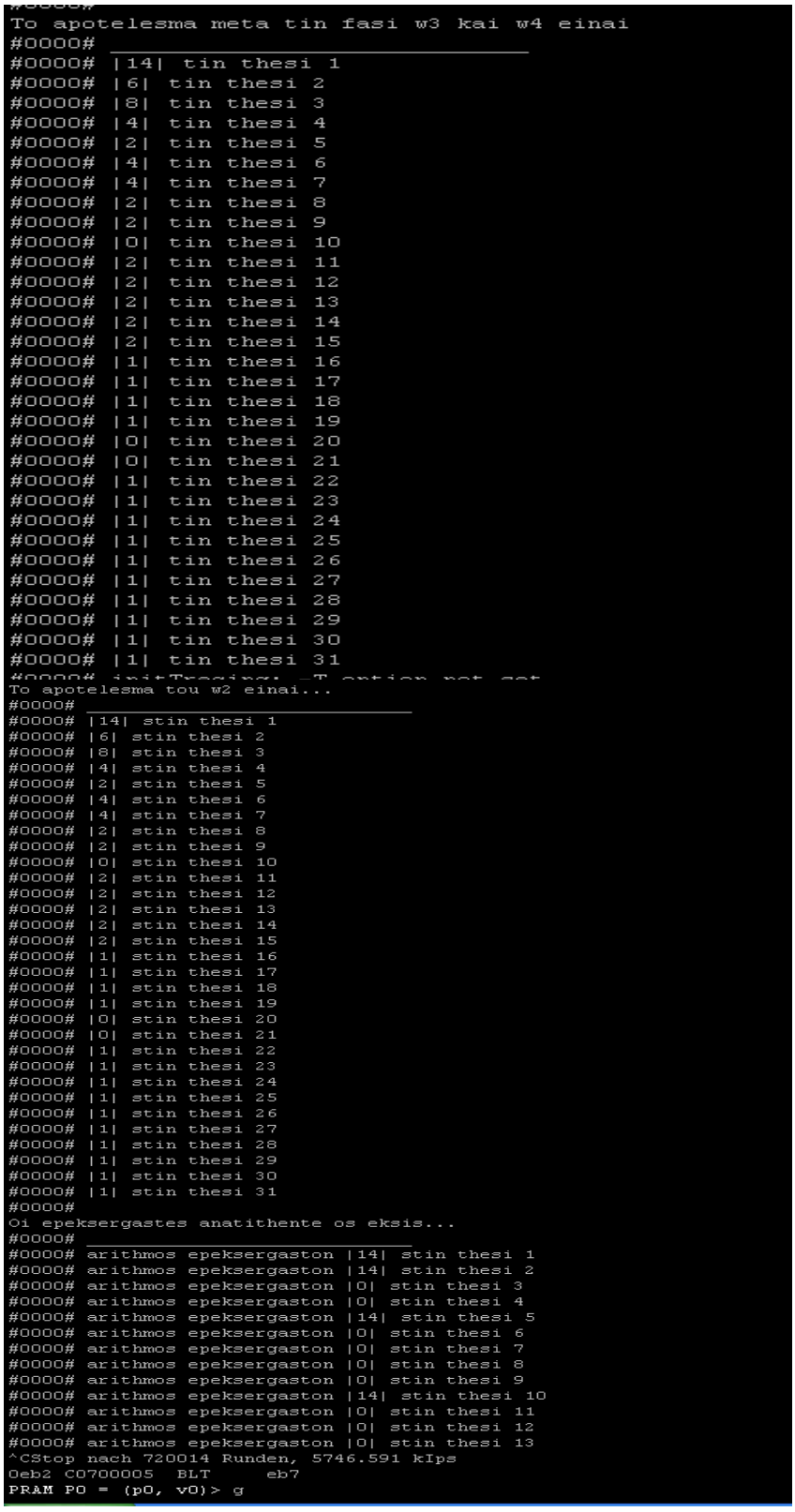
```
      ***** PRAM-SIMULATOR (v2.0) *****
         >>>>>>>> DAMN FAST <<<<<<<<
      (c) by hirbli & stefran  07.10.94
You have 3 physical processors with 1 vP's each

Relocating file /home/students/cs/2005/cs05np1/../../../../../local/cs02lp1/examples/diplomatiki/../X_newWithBit... done
Loading file /var/tmp/aaaZdayU5.cod...Doing realloc
Doing realloc
 done
PRAM P0 = (p0, v0)> g
#0000#            ALGORITHMOS X KAI X'
#0000# ---------------------------------------

#0000# Dose ta stoixeia pou theleis na exei o pinakas...
8
#0000# 8
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0001#
 @ O epeksergastis 1 egrapse stin 8 thesi tou pinaka#0002#
 @ O epeksergastis 2 egrapse stin 9 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 4 thesi tou pinaka #0002#
 @ O epeksergastis 2 egrapse stin 4 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 10 thesi tou pinaka#0002#
 @ O epeksergastis 2 egrapse stin 11 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 5 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 2 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 12 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 13 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 6 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 14 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 15 thesi tou pinaka#0001#
 @ O epeksergastis 1 egrapse stin 7 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 3 thesi tou pinaka #0001#
 @ O epeksergastis 1 egrapse stin 1 thesi tou pinaka #0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set
EXIT: vp=#1, pc=$000006b7
EXIT: vp=#2, pc=$000006b7
Stop nach 307013 Runden, 4385.900 kIps
4ea4 8061C002  ADD     R3, 2, R6
PRAM P0 = (p0, v0)>
```

B.4 Αλγόριθμος W:: Σενάριο5 (Όπου καταρρέουν οι κυψελίδες 20 και 21)

Το παρακάτω αποτέλεσμα προέκυψε από την εκτέλεση του αλγορίθμου από 16 επεξεργαστές σε 16 στοιχεία. Παρακάτω παρατηρούνται τα αποτελέσματα που προκύπτουν για την κάθε φάση.



```
UeB2 C0700005  BLT      eb7
PRAM PO = (pO, vO)> g
#0000# arithmos epeksergaston |O| stin thesi 14
#0000# arithmos epeksergaston |O| stin thesi 15
#0000# arithmos epeksergaston |O| stin thesi 16
#0000# arithmos epeksergaston |O| stin thesi 17
#0000# arithmos epeksergaston |O| stin thesi 18
#0000# arithmos epeksergaston |O| stin thesi 19
^CStop nach 106788 Runden, 5673.113 kIps
5beO C1800000  BMC      5beO
PRAM PO = (pO, vO)> g
#0000# arithmos epeksergaston |7| stin thesi 20
#0000# -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
#0000#
-O epeksergastis (1) tha paei sin thesi 20
#0000#
-O epeksergastis (2) tha paei sin thesi 20
#0000#
-O epeksergastis (3) tha paei sin thesi 20
#0000#
-O epeksergastis (4) tha paei sin thesi 20
#0000#
-O epeksergastis (7) tha paei sin thesi 20
#0000#
-O epeksergastis (8) tha paei sin thesi 20
#0000#
-O epeksergastis (9) tha paei sin thesi 20
#0000#
-O epeksergastis (10) tha paei sin thesi 21
#0000#
-O epeksergastis (11) tha paei sin thesi 21
#0000#
-O epeksergastis (12) tha paei sin thesi 21
#0000#
-O epeksergastis (13) tha paei sin thesi 21
#0000#
-O epeksergastis (14) tha paei sin thesi 21
#0000#
-O epeksergastis (15) tha paei sin thesi 21
#0000#
-O epeksergastis (16) tha paei sin thesi 21
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set
#0001# stopTracing: -T option not set
#0001# writeTraceFile: -T option not set
#0000# -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
#0000#
-O epeksergastis (16) tha paei sin thesi 21
#0000# initTracing: -T option not set
#0000# -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
#0000#
-O epeksergastis (16) tha paei sin thesi 20
#0000# initTracing: -T option not set
#0000# startTracing: -T option not set
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set
#0000#
To neo apotelesma einai..
#0016# stopTracing: -T option not set
#0000# ____________________________
#0016# writeTraceFile: -T option not set
#0000# |16| stin thesi  1
#0000# |8| stin thesi  2
#0000# |8| stin thesi  3
#0000# |4| stin thesi  4
#0000# |4| stin thesi  5
#0000# |4| stin thesi  6
#0000# |4| stin thesi  7
#0000# |2| stin thesi  8
#0000# |2| stin thesi  9
#0000# |2| stin thesi  10
#0000# |2| stin thesi  11
#0000# |2| stin thesi  12
#0000# |2| stin thesi  13
#0000# |2| stin thesi  14
#0000# |2| stin thesi  15
#0000# |1| stin thesi  16
#0000# |1| stin thesi  17
#0000# |1| stin thesi  18
#0000# |1| stin thesi  19
#0000# |1| stin thesi  20
#0000# |1| stin thesi  21
#0000# |1| stin thesi  22
#0000# |1| stin thesi  23
#0000# |1| stin thesi  24
#0000# |1| stin thesi  25
#0000# |1| stin thesi  26
#0000# |1| stin thesi  27
#0000# |1| stin thesi  28
#0000# |1| stin thesi  29
#0000# |1| stin thesi  30
#0000# |1| stin thesi  31
#0000# stopTracing: -T option not set
#0000# writeTraceFile: -T option not set
#0010#

#0011#
```