Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΑΠΟΔΟΤΙΚΩΝ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΕ ΤΟ ΛΟΓΙΣΜΙΚΟ PUBSP

Μαργαρίτα Αντωνάκη

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάρτιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΑΠΟΔΟΤΙΚΩΝ ΠΑΡΑΛΛΗΛΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΕ ΤΟ ΛΟΓΙΣΜΙΚΟ PUBSP

Μαργαρίτα Αντωνάκη

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

Στα πλαίσια ανάπτυξης και ολοκλήρωσης της διπλωματικής μου εργασίας ένα σημαντικό άτομο το οποίο με βοήθησε ιδιαίτερα με τις συμβουλές του και με τη βοήθεια του ήταν ο επιβλέπων καθηγητής Δρ. Χρύσης Γεωργίου, Επίκουρος καθηγητής του τμήματος πληροφορικής του Πανεπιστημίου Κύπρου.

Ως επιβλέπων καθηγητής μου έδωσε τις κατευθυντήριες γραμμές ώστε να καταφέρω να τελειώσω την διπλωματική μου εργασία σε ένα καθορισμένο χρονικό διάστημα. Επιπλέον έχοντας πολλή εμπειρία και γνώση όσο αφορά την παράλληλη επεξεργασία, αφού είναι ερευνητής στην θεωρία και πρακτική του παράλληλου υπολογισμού με βοήθησε να κατανοήσω κάποια σημαντικά θέματα όσο αφορά τον παράλληλο υπολογισμό και να μπορέσω να προχωρίσω με την εργασία μου. Σε δύσκολες στιγμές και αδιέξοδα, με συναντήσεις και συζητήσεις μαζί του, αλλά και με προσωπική μελέτη και έρευνα κατάφερνα τελικά να τα ξεπεράσω και να προχωρίσω με την εργασία μου.

Τελικά πέραν της ολοκλήρωσης της διπλωματικής εργασίας, κατάφερα με την βοήθεια του Δρ. Χρύση Γεωργίου να εμβαθύνω στην θεωρία του παράλληλου υπολογισμού, ενός θέματος πολύ σημαντικού στον τομέα της πληροφορικής στις μέρες μας. Ασχολήθηκα με πολύ ενδιαφέροντες αλγορίθμους και έμαθα την λειτουργία του μοντέλου BSP.

ΠΕΡΙΛΗΨΗ

Κατά την ανάπτυξη και ολοκλήρωση της ερευνητικής μου εργασίας ερευνήθηκαν διάφορα θέματα που σχετίζονται με τον παράλληλο υπολογισμό και πιο συγκεκριμένα μελετήθηκε το μοντέλο παράλληλου υπολογισμού BSP και εκτελέστηκαν διάφοροι αλγόριθμοι σύμφωνα με αυτό. Για τους αλγόριθμους αυτούς έγινε στην συνέχεια πειραματική αξιολόγηση ώστε να εξαχθούν διάφορα συμπεράσματα.

Στόχος της εργασίας ήταν η μελέτη και εξοικείωση με το μοντέλο BSP και με διάφορους αλγόριθμους. Στην συνέχεια η ανάπτυξη των αλγορίθμων αυτών με βάση το μοντέλο αυτό.

Το BSP αποτελεί ένα υπολογιστικό μοντέλο παράλληλου υπολογισμού, το οποίο περιέχει κάποια βασικά χαραχτηριστικά. Αρχικά αποτελείται από ένα σύνολο από επεξεργαστές, οι οποίοι είναι υπεύθυνοι να διαμοιραστούν την εργασία που πρέπει να εκτελεστεί για την πιο εύκολη και γρήγορη εκτέλεση της. Επίσης περιέχει ένα δίκτυο επικοινωνίας που μεταφέρει μηνύματα από σημείο σε σημείο και ένα μηχανισμό για αποδοτικό φραγμένο συγχρονισμό για όλες ή ένα μέρος των διαδικασιών. Σημαντικό χαρακτηριστικό του μοντέλου είναι τα υπερβήματα στα οποία είναι χωρισμένοι οι διάφοροι αλγορίθμοι, κατά την διάρκεια του κάθενος γίνονται τοπικοί υπολογισμοί από τους επεξεργαστές και μερικές φορές ανταλλάσονται μηνύματα μεταξύ των επεξεργαστών, όταν αυτό είναι απαραίτητο. Κάθε υπερβήμα τελειώνει με ένα φραγμένο συγχρονισμό. Το μοντέλο αυτό έιναι πολύ σημαντικό διότι αποτελεί ένα ενοποιητικό μοντέλο για παράλληλο υπολογισμό. Μπορούν να γράφονται εύκολα τα διάφορα προγράμματα που δημιουργούνται σύμφωνα με αυτό και επιπρόσθετα είναι ανεξάρτητα από συγκεκριμένες αρχιτεχτονικές. Ακόμη η απόδοση των προγραμμάτων που είναι υλοποιημένα με βάση αυτό το μοντέλο είναι προβλέψιμη.

Μετά την μελέτη του μοντέλου αυτού ακολουθεί η υλοποίηση διάφορων αλγορίθμων και η πειραματική αξιολόγηση τους που οδήγησε σε διάφορα συμπεράσματα όσο αφορά την συμπεριφορά των αλγορίθμων αλλά και του μοντέλου BSP.

Με την ανάπτυξη και ολοκλήρωση της εργασίας αυτής κατάφερα να αποκτήσω αρκετές εμπειρίες και χρήσιμες γνώσεις για την χρήση του μοντέλου BSP αλλά και γενικότερα για την παράλληλη επεξεργασία. Επιπλέον συνεργάστηκα με έναν έμπειρο ερευνητή και

καθηγητή του Πανεπιστημίου Κύπρου το Δρ. Χρύση Γεωργίου, από τον οποίο κατάφερα να κερδίσω πολλές χρήσιμες γνώσεις για τους παράλληλους αλγορίθμους και το μοντέλο BSP.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Σκοπός του Κεφαλαίου 1 είναι να παρουσιάσει τους λόγους που οδήγησαν σε αυτή την μελέτη και την διαδικασία που ακολουθήθηκε για την ολοκλήρωση της. Στο πρώτο και δεύτερο Υποκεφάλαιο 1.1, 1.2 παρουσιάζεται το κίνητρο και ο στόχος της διπλωματικής εργασίας. Δηλαδή αναφέρεται το θέμα της εργασίας και γίνεται μια περιγραφή του μοντέλου παράλληλης επεξεργασίας BSP. Τονίζεται ο στόχος της διπλωματικής εργασίας και η χρησιμότητα γενικά του παράλληλου υπολογισμού και ειδικότερα του μοντέλου BSP. Στην συνέχεια στο Υποκεφάλαιο 1.3 παρουσιάζονται τα βήματα που ακολουθήθηκαν για την αποπεράτωση της εργασίας. Με λίγα λόγια παρουσιάζεται η μεθοδολογία που ακολουθήθηκε και η λεπτομερής ανάλυση των βημάτων, με περιγραφή όλων όσων έγιναν κατά την διάρκεια ολοκλήρωσης της εργασίας. Τέλος στο Υποκεφάλαιο 1.4 παρουσιάζεται το τι θα ακολουθήσει στην συνέχεια του εγγράφου.

1.1 Κίνητρο διπλωματικής εργασίας

Ο παράλληλος υπολογισμός θεωρείται ότι είναι πολύ σημαντικός και επίσης ότι αποτελεί το μέλλον της επεξεργασίας σε υπολογιστικά συστήματα. Δύο κύριοι λόγοι που τον κάνουν όχι μόνο να είναι σημαντικός , αλλά και απαραίτητος, είναι πρώτον το γεγονός ότι σήμερα δημιουργούνται πολύπλοκοι και μεγάλοι υπολογισμοί, που ένας σειριακός υπολογιστής δεν μπορεί να τους διεκπεραιώσει και δεύτερον οι φυσικοί περιορισμοί που υπάρχουν στα κυκλώματα VLSI (Very Large Scale Integration). Επιπρόσθετα, το κόστος των διάφορων παράλληλων συστημάτων έχει πλέον μειωθεί αρκετά, οπόταν η χρήση του παράλληλου υπολογισμού είναι αρκετά συμφέρουσα στις μέρες μας [**2**].

Οι δεκαετίες εμπειρίας έδειξαν όμως ότι η δημιουργία ενός αποδοτικού και φορητού (portable) παράλληλου λογισμικού είναι μια διαδικασία αρκετά περίπλοκη και αυτό οφείλεται σε ένα μεγάλο βαθμό στην ποικιλομορφία των υπαρχόντων παράλληλων συστημάτων και των παράλληλων μοντέλων (parallel computer models).

Παρατηρούμε ότι κάποια χρησιμοποιούν τεχνικές διαμοιρασμού μνήμης (shared-memory techniques) και κάποια άλλα τεχνικές μετάδοσης δεδομένων (message passing). Κάποια εκμεταλλεύονται την τοπικότητα των διχτύων, ενώ κάποια άλλα όχι. Ως αποτέλεσμα της πιο πάνω χαοτικής κατάστασης, κάθε παράλληλος κώδικας που γράφεται, γράφεται σύμφωνα με τον υπολογιστή στον οποίο θα τρέξει. Συνεπώς όταν επιθυμούμε η συγκεκριμένη εφαρμογή να τρέξει σε διαφορετικό παράλληλο υπολογιστή, απαιτείται μια θεμελιώδης αναπροσαρμογή του κώδικα.

Επιζητώντας να κάνουμε την παράλληλη επεξεργασία πιο εύχρηστη, πρέπει να δημιουργηθεί ένα ενοποιητικό μοντέλο για παράλληλο υπολογισμό (unifying model for parallel programming). Ένα αποδοτικό ενοποιητικό μοντέλο πρέπει να έχει και κάποια χαρακτηριστικά. Αρχικά πρέπει να είναι αρκετά απλό, ώστε να μπορούν να το χρησιμοποιούν εύκολα οι προγραμματιστές και να γράφουν κώδικες. Επίσης πρέπει να είναι τέτοιο, ώστε να μπορούν εύκολα οι αρχιτέκτονες υλικού υπολογιστών (hardware) να σχεδιάζουν αποδοτικό υλικό, που να υποστηρίζει κώδικες γραμμένους σε αυτό το μοντέλο. Τέλος πρέπει να μπορεί να βοηθά στην ανάπτυξη και ανάλυση αλγόριθμων.

Το Bulk-Synchronous Parallel (BSP) model, που προτάθηκε από τον Leslie Valiant, είναι ένα υποσχόμενο υποψήφιο ενοποιητικό μοντέλο για παράλληλο υπολογισμό [**3,8**].

1.2 Στόχος διπλωματικής εργασίας

Στα πλαίσια αυτής της διπλωματικής εργασίας, μελετήθηκε το μοντέλο παράλληλου προγραμματισμού BSP (Bulk Synchronous Parallel computing model) [**2,6,13**] και επίσης η βιβλιοθήκη PUB [**1**].

Στο μοντέλο BSP υπάρχει ένας παράλληλος υπολογιστής, που αποτελείται από ένα σύνολο από ασυγχρόνιστους επεξεργαστές, όπου ο καθένας έχει την δική του τοπική μνήμη (local memory). Επίσης υπάρχει ένας μηχανισμός αλληλοσύνδεσης που επιτρέπει την επικοινωνία από σημείο σε σημείο (point to point) και ένας μηχανισμός φραγμένου συγχρονισμού (barier style synchronazation). Αυτός ο παράλληλος υπολογιστής είναι με λίγα λόγια η BSP μηχανή [**3**].

Παράμετροι λογισμικού σε αυτή την μηχανή είναι το p, που εκφράζει το πλήθος των διαθέσιμων επεξεργαστών, το g (gap) που αποτελεί τον χρόνο που χρειάζεται για να παραδοθεί ένα byte δεδομένων μέσα στον μηχανισμό αλληλοσύνδεσης, και τέλος το L που είναι ο χρόνος που απαιτείται από την μηχανή για να γίνει ο φραγμένος συγχρονισμός. Για συγκεκριμένες μηχανές το g και L καθορίζονται πειραματικά [**12**].

Η εκτέλεση ενός αλγόριθμου στο BSP μοντέλο είναι μια ακολουθία από υπερβήματα (supersteps), που το κάθε ένα τερματίζει με ένα φραγμένο συγχρονισμό. Σε κάθε υπερβήμα οι επεξεργαστές εκτελούν τοπικούς υπολογισμούς και στέλνουν μηνύματα σε άλλους επεξεργαστές. Τα μηνύματα αυτά είναι διαθέσιμα στους επεξεργαστές, όταν τελειώσει ο επόμενος συγχρονισμός.

Το μοντέλο BSP χρησιμοποιεί την βιβλιοθήκη PUB, η οποία είναι μια βιβλιοθήκη της C, που υποστηρίζει την ανάπτυξη και υλοποίηση παράλληλων αλγόριθμων, σχεδιασμένων σύμφωνα με το μοντέλο BSP. Η βιβλιοθήκη αυτή προσφέρει ασυγχρόνιστη μετάδοση δεδομένων μεταξύ των επεξεργαστών, που είναι οργανωμένη σε υπερβήματα. Στην αρχή κάθε υπερβήματος οι επεξεργαστές παραλαμβάνουν μηνύματα που στάλθηκαν στο προηγούμενο υπερβήμα. Η βιβλιοθήκη PUB προσφέρει επιπρόσθετα λειτουργίες για ανταλλαγή μηνυμάτων από σημείο σε σημείο, συναρτήσεις για μετάδοση δεδομένων (broadcast) και λειτουργίες παράλληλου προθεματικού υπολογισμού (parallel prefix) μεταξύ ζευγαριών επεξεργαστών. Επιπρόσθετα είναι αρκετά ευέλικτη, αφού επιτρέπει την δημιουργία ανεξάρτητων BSP αντικειμένων, όπου το καθένα αντιπροσωπεύει ένα εικονικό

BSP υπολογιστή. Τέλος υποστηρίζει εικονικούς επεξεργαστές για να τρέξει παράλληλα προγράμματα [**11,13**].

Βασικός στόχος σε αυτή την εργασία ήταν να χρησιμοποιηθεί το λογισμικό PubBSP για την ανάπτυξη παράλληλων αλγόριθμων με βάση το μοντέλο BSP. Στην συνέχεια, πολύ σημαντικό κομμάτι της εργασίας, ήταν η σύγκριση και η πειραματική αξιολόγηση των αλγόριθμων που αναπτύχθηκαν με βάση τις παραμέτρους που χαρακτηρίζουν το μοντέλο αυτό.

Έτσι με βάση τα πιο πάνω τέθηκαν από την αρχή της εργασίας κάποιοι στόχοι, οι οποίοι έπρεπε να υλοποιηθούν σε καθορισμένο χρονικό διάστημα. Οι στόχοι αυτοί ήταν να γίνει μελέτη του παράλληλου μοντέλου BSP και η εκμάθηση της γλώσσας προγραμματισμού PubBSP, στόχοι οι οποίοι ήταν πολύ σημαντικοί στην επιτυχημένη ολοκλήρωση της εργασίας, λόγω του ότι η κατανόηση του μοντέλου και η εκμάθηση της γλώσσας θα βοηθούσε στην εκπλήρωση των στόχων που ακολουθούσαν. Στη συνέχεια ένας πολύ βασικός στόχος ήταν η μελέτη και υλοποίηση βασικών παράλληλων αλγόριθμων, και τέλος η πειραματική αξιολόγηση των αλγόριθμων αυτών, η οποία μας βοήθησε να φτάσουμε σε συμπεράσματα, όσο αφορά το μοντέλο και τις διάφορες παραμέτρους του.

1.3 Μεθοδολογία που ακολουθήθηκε

Στα πλαίσια ολοκλήρωσης της ερευνητικής μου εργασίας απέκτησα πολλές χρήσιμες εμπειρίες και μου δόθηκε η δυνατότητα να εμβαθύνω σε θέματα παράλληλων αλγόριθμων και συγκεκριμένα στην χρήση και λειτουργία του παράλληλου μοντέλο BSP και να αντιληφθώ την πραγματική αξία της παράλληλης επεξεργασίας. Επίσης εξάσκησα και βελτίωσα τις ικανότητες μου στον προγραμματισμό αλγόριθμων και εξοικειώθηκα με τον τρόπο διεξαγωγής της πειραματικής αξιολόγησης αλγόριθμων. Τέλος είχα την τύχη να συνεργαστώ με τον επίκουρο καθηγητή Χρύση Γεωργίου, από τον οποίο αποκόμισα πολύτιμες συμβουλές.

Για την αποπεράτωση της ερευνητικής μου εργασίας χρειάστηκε να γίνουν 5 βασικά βήματα :

1. Μελέτη και κατανόηση του παράλληλου μοντέλου BSP.
2. Εκμάθηση της γλώσσας προγραμματισμού PuBsp.
3. Μελέτη και κατανόηση συγκεκριμένων παράλληλων αλγόριθμων.
4. Υλοποίηση αλγόριθμων.
5. Πειραματική αξιολόγηση.

Στην **πρώτη φάση** μελετήθηκε το μοντέλο παράλληλου υπολογισμού BSP, το οποίο είναι ένα πολύ βασικό μοντέλο, διότι σε αντίθεση με άλλα μοντέλα παράλληλου υπολογισμού μπορεί να υλοποιηθεί εύκολα και στην πράξη. Για τη μελέτη αυτή χρησιμοποιήθηκαν διάφορα επιστημονικά έγγραφα τα οποία δόθηκαν από τον Δρ. Χρύση Γεωργίου, ή εντοπίστηκαν μέσω αναζήτησης στο διαδίκτυο. Τα έγγραφα αυτά φάνηκαν πολύ χρήσιμα, διότι περιέγραφαν αναλυτικά το μοντέλο και περιείχαν πολύ κατανοητά παραδείγματα.
Στην **δεύτερη φάση** έγινε η εκμάθηση της γλώσσας PuBsp με μελέτη του εγχειριδίου [**5**]. Ενώ ακολούθως έγινε η εγκατάσταση της βιβλιοθήκης PUB στον προσωπικό μου λογαριασμό του Πανεπιστημίου Κύπρου. Οδηγίες για εγκατάσταση της βιβλιοθήκης υπάρχουν στο Παράρτημα Γ. Με την εγκατάσταση της βιβλιοθήκης και χρησιμοποιώντας το λειτουργικό σύστημα Linux, εκτελέστηκαν μικρά παραδείγματα προγραμμάτων για την καλύτερη κατανόηση και εξοικείωση με την γλώσσα προγραμματισμού PuBsp.
Ακολούθως στην **τρίτη φάση** έγινε η μελέτη εις βάθος και κατανόηση συγκεκριμένων αλγόριθμων προς υλοποίηση. Αυτοί ήταν τρεις διαφορετικοί αλγόριθμοι για μετάδοση δεδομένων, τέσσερις διαφορετικοί αλγόριθμοι για πολλαπλασιασμό πινάκων, ο βέλτιστος αλγόριθμος για παράλληλη προθεματική άθροιση, ο βέλτιστος αλγόριθμος για παράλληλη άθροιση, ένας αλγόριθμος για παράλληλη προθεματική άθροιση σε τετραγωνικό δίχτυο αλληλοσύνδεσης και ένας αλγόριθμος παράλληλης άθροισης, επίσης σε τετραγωνικό δίκτυο αλληλοσύνδεσης.
Στην **τέταρτη φάση** υλοποιήθηκαν αυτοί οι αλγόριθμοι σύμφωνα με το μοντέλο BSP και χρησιμοποιώντας την γλώσσα προγραμματισμού PuBsp. Οι κώδικες υλοποίησης των αλγόριθμων στην γλώσσα PuBsp βρίσκονται στο Παράρτημα Α.
Τέλος στην **πέμπτη** και τελευταία φάση έγινε η πειραματική αξιολόγηση των αλγόριθμων που υλοποιήθηκαν, με την εκτέλεση τους σε διάφορα πλήθη εικονικών επεξεργαστών σε

ένα υπολογιστή του Πανεπιστημίου Κύπρου, την λήψη μετρήσεων και την εξαγωγή συμπερασμάτων.

1.4 Οργάνωση Κειμένου

Στην συνέχεια ακολουθεί το Κεφάλαιο 2 που παρουσιάζει τον παράλληλο υπολογισμό και ειδικότερα το μοντέλο παράλληλου υπολογισμού BSP, στοιχεία πάνω στα οποία βασίστηκε η ολοκλήρωση της διπλωματικής μου εργασίας. Ακολούθως στο Κεφαλαίου 3 γίνεται η παρουσίαση της προγραμματιστικής γλώσσας PuBSP που χρησιμοποιήθηκε στην δημιουργία των αλγόριθμων. Στο Κεφάλαιο 5 παρουσιάζεται το πρόβλημα του παράλληλου προθεματικού υπολογισμού και πειραματική αξιολόγηση για τον αλγόριθμο που υλοποιήθηκε. Στη συνέχεια στο Κεφαλαίου 6 παρουσιάζεται το πρόβλημα της παράλληλης άθροισης, επίσης με την πειραματική αξιολόγηση για τον υλοποιημένο αλγόριθμο. Στο Κεφάλαιο 7 παρουσιάζεται η πειραματική αξιολόγηση για δύο αλγόριθμους υλοποιημένους σε τετραγωνικό πλέγμα (mesh). Ο πρώτος αλγόριθμος που παρουσιάζεται είναι ο αλγόριθμος για παράλληλη άθροιση και ο δεύτερος για παράλληλη προθεματική άθροιση. Στο Κεφάλαιο 8 παρουσιάζεται το πρόβλημα του πολλαπλασιασμού πινάκων και αλγόριθμοι για την επίλυση του, μαζί με πειραματική αξιολόγηση. Τέλος στο Κεφάλαιο 9 ολοκληρώνεται η ανάλυση της διπλωματικής μου εργασίας, παραθέτοντας κάποια τελικά συμπεράσματα.

ΚΕΦΑΛΑΙΟ 2

Περιγραφή του Μοντέλου Φραγμένου Συγχρονισμού BSP

Το Κεφάλαιο 2 έχει σαν στόχο να παρουσιάσει τον παράλληλο υπολογισμό και ειδικότερα το μοντέλο παράλληλου υπολογισμού BSP, στοιχεία πάνω στα οποία θα βασιστεί η ολοκλήρωση της διπλωματικής εργασίας. Στο Υποκεφάλαιο 2.1 παρουσιάζονται κάποια βασικά στοιχεία για τον παράλληλο υπολογισμό, στο Υποκεφάλαιο 2.2 παρουσιάζονται κάποιες σχετικές έννοιες, συγκεκριμένα τα μοντέλα χρονισμού και στο Υποκεφάλαιο 2.3 γίνεται μια αναλυτική περιγραφή του μοντέλου BSP, που είναι και το βασικό θέμα που κυριαρχεί στην ερευνητική αυτή εργασία. Στο Υποκεφάλαιο 2.4 γίνεται μια περαιτέρω ανάλυση διάφορων σχετικών θεμάτων με το μοντέλο, για την διευκρίνιση μερικών βασικών χαρακτηριστικών του μοντέλου.

2.1 Παράλληλος υπολογισμός

Ο παράλληλος υπολογισμός αποτελεί την συνεταιριστική και ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από ένα επεξεργαστές που αποσκοπεί στη γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων [**2,4**].

Ένας παράλληλος υπολογιστής είναι μια συλλογή από επεξεργαστές, ικανή να εκτελεί παράλληλους υπολογισμούς. Οι επεξεργαστές βρίσκονται σε «κοντινή» απόσταση, είναι συνήθως επεξεργαστές του ίδιου τύπου (ομοιογενείς) και επιλύουν μαζί κάθε πρόβλημα. Έτσι έχουν τεράστια υπολογιστική δύναμη.

Επιπρόσθετα η βασική διαφορά μεταξύ του παράλληλου και σειριακού υπολογισμού έγκειται στο γεγονός ότι σε ένα συνηθισμένο ακολουθιακό επεξεργαστή η επεξεργασία μπορεί να γίνεται κάθε φορά σε μια φυσική τοποθεσία. Σε μια παράλληλη όμως μηχανή η επεξεργασία μπορεί να γίνει ταυτόχρονα σε πολλές τοποθεσίες. Πολλές υπολογιστικές λειτουργίες μπορούν να γίνουν ανά δευτερόλεπτο [**12**].

2.2 Μοντέλα χρονισμού

Πιο κάτω παρουσιάζονται τα μοντέλα χρονισμού [**2**]:

Συγχρονισμένο: Σε ένα τέτοιο μοντέλο όλοι οι επεξεργαστές μεταβαίνουν από το ένα βήμα στο επόμενο με τον ίδιο ρυθμό και η έννοια του βήματος είναι κοινή για όλους τους επεξεργαστές. Με αυτό τον τρόπο η επεξεργασία είναι συγχρονισμένη.

Ασυγχρόνιστο: Σε ένα τέτοιο μοντέλο παράλληλου υπολογισμού κάθε επεξεργαστής κινείται με διαφορετικό ρυθμό και σε διαφορετικές ταχύτητες επεξεργασίας.

Συγχρονισμός: Είναι η διαδικασία εξαναγκασμού ασυγχρόνιστων επεξεργαστών να συγχρονίσουν τα βήματά τους.

2.3 Περιγραφή μοντέλου BSP

Αρκετά μοντέλα παράλληλου υπολογισμού, όπως παραδείγματος χάριν το **PRAM** [**2**], έχουν δύσκολη άμεση υλοποίηση σε αλγόριθμων σε πραγματικά παράλληλα συστήματα. Καταρχήν το μοντέλο PRAM απαιτεί απόλυτο συγχρονισμό, ο οποίος στοιχίζει και όσος περισσότερος συγχρονισμός απαιτείται, τόσο πιο δύσκολη γίνεται η υλοποίηση. Ακόμη στο μοντέλο αυτό γίνονται κάποιες υποθέσεις. Συγκεκριμένα θεωρούμε ότι οι επεξεργαστές έχουν πρόσβαση στην κοινόχρηστη μνήμη σε σταθερό χρόνο και ποτέ δεν παθαίνουν κάποια βλάβη. Κατά συνέπεια το μοντέλο αυτό είναι το πιο κατάλληλο σε θεωρητικό επίπεδο, δηλαδή κατάλληλο για μελέτη και σχεδιασμό παράλληλων αλγόριθμων, όμως δεν υλοποιείται εύκολα σε πραγματικά παράλληλα συστήματα.

Σε αντίθεση, το **BSP** δεν είναι απλά ένα θεωρητικό μοντέλο, είναι ένα παράδειγμα για προγραμματισμό παράλληλων υπολογιστών. Υπάρχουν διάφορα μοντέλα που μπορούν να χρησιμοποιηθούν σε πραγματικά παράλληλα συστήματα, όμως εμείς θα ασχοληθούμε με το BSP.

Κάποιες ιδιότητες του μοντέλου είναι ότι μπορούμε εύκολα να γράφουμε προγράμματα στο μοντέλο αυτό, τα οποία είναι ανεξάρτητα από την αρχιτεκτονική των υπολογιστών που εκτελούνται. Επίσης η απόδοση του μοντέλου είναι προβλέψιμη, λόγω του ότι υπάρχει κάποιος μαθηματικός τύπος, με τον οποίο μπορούμε να βρούμε στο περίπου το χρόνο και το κόστος των αλγόριθμων, όπως επίσης λόγω του ότι υπάρχουν κάποιοι παράμετροι του μοντέλου που μπορούμε να τις υπολογίσουμε εύκολα. Οπόταν, σύμφωνα με τα πιο πάνω στοιχεία, μπορούμε εύκολα να γνωρίζουμε από την αρχή την απόδοση των αλγόριθμων μας.

Στο μοντέλο BSP υπάρχει ένα σύνολο από ασυγχρόνιστους επεξεργαστές, που επικοινωνούν μέσω ενός μηχανισμού αλληλοσύνδεσης (π.χ. δίκτυο), μέσω του οποίου επιτρέπεται η ανταλλαγή μηνυμάτων.
Σημαντικό στοιχείο είναι ότι ο κάθε επεξεργαστής έχει την δική του τοπική μνήμη, στην οποία αποθηκεύονται δεδομένα, που μόνο ο ίδιος επεξεργαστής γνωρίζει. Κατά την

διάρκεια εκτέλεσης των αλγόριθμων στο μοντέλο αυτό, γίνονται διάφοροι τοπικοί υπολογισμοί, στους οποίους κάθε επεξεργαστής επεξεργάζεται δεδομένα που βρίσκονται στην δική του τοπική μνήμη.
Ακόμη στο μοντέλο αυτό υπάρχει και ένας μηχανισμός φραγμένου συγχρονισμού, σύμφωνα με τον οποίο εξαναγκάζονται όλοι οι ασυγχρόνιστοι επεξεργαστές να συγχρονιστούν.

Το μοντέλο BSP θεωρείται ο συνδυασμός τριών χαρακτηριστικών τα οποία είναι τα ακόλουθα [**13**]:

1. Μια συλλογή από συνιστώσες, που η κάθε μια υλοποιεί επεξεργασία (processor-memory).
2. Ένα δίκτυο επικοινωνίας, που μπορεί να μεταφέρει μηνύματα από σημείο σε σημείο μεταξύ των συνιστώσων. Για αυτόν το σκοπό, δηλαδή για την μεταφορά των μηνυμάτων, είναι υπεύθυνος ένας δρομολογητής.
3. Η δυνατότητα για καθολικό συγχρονισμό, σε όλες ή ένα υποσύνολο των συνιστώσων, σε κανονικά διαστήματα L μονάδων χρόνου.

Έτσι σημαντικοί παράμετροι του μοντέλου είναι οι εξής:

p – Αριθμός των επεξεργαστών.

L – Χρόνος που απαιτείται για συγχρονισμό.

g – Η απώλεια (καθυστέρηση) του μηχανισμού αλληλοσύνδεσης.

h – σχέση – Ο μέγιστος αριθμός μηνυμάτων που στέλνει ή παραλαμβάνει ένας επεξεργαστής σε μια επικοινωνία, γεγονός που εξαρτάται από το συγκεκριμένο αλγόριθμο.

Η εκτέλεση ενός αλγόριθμου στο μοντέλο **BSP(p,L,g)** αποτελείται από **υπερβήματα,** όπου καθένα από αυτά τερματίζει με ένα φραγμένο συγχρονισμό. Ένα υπερβήμα είναι ένα πολύ σημαντικό και χαρακτηριστικό στοιχείο για το μοντέλο BSP και αποτελεί ένα κομμάτι υπολογισμού, όπου κάθε επεξεργαστής υλοποιεί μια συγκεκριμένη διεργασία χρησιμοποιώντας τα δικά του τοπικά δεδομένα. Μια τέτοια διεργασία, που αναλαμβάνει να

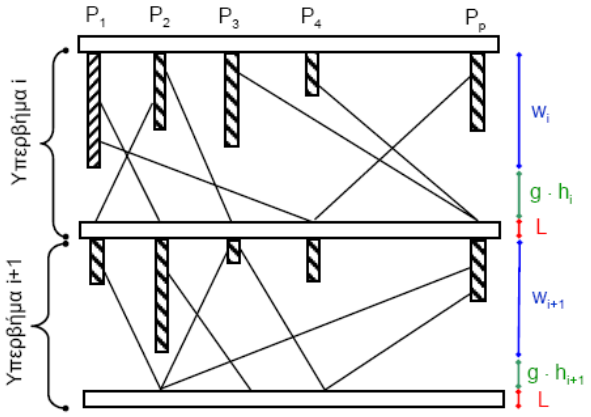
κάνει ο επεξεργαστής, συμπεριλαμβάνει τοπικούς υπολογισμούς από τους επεξεργαστές και αποστολή μηνυμάτων, κατά το επικοινωνιακό μέρος, από κάποιους επεξεργαστές σε άλλους. Τα μηνύματα αυτά είναι διαθέσιμα μέχρι το επόμενο υπερβήμα. Έτσι ο κάθε επεξεργαστής που έχει παραλάβει κάποιο μήνυμα σε κάποιο υπερβήμα του αλγόριθμου, αν δεν το αποθηκεύσει στην τοπική του μνήμη, τότε με την αρχή του καινούριου υπερβήματος το μήνυμα θα χαθεί και ο επεξεργαστής δεν θα γνωρίζει πλέον κάτι για αυτό.

Το μοντέλο BSP θεωρείται μερικώς συγχρονισμένο, διότι παρ΄ όλο του ότι οι επεξεργαστές είναι ασυγχρόνιστοι μεταξύ τους, υπάρχει κάποιος συγχρονισμός στο τέλος κάθε υπερβήματος.

Ο χρόνος εκτέλεσης των αλγόριθμων σε κάθε υπερβήμα καθορίζεται από την εξής μαθηματική σχέση: $\mathbf{w_t + g.\ h_t + L}$ [**10**], όπου **t** είναι το υπερβήμα στο οποίο βρίσκεται η εκτέλεση του αλγόριθμου, $\mathbf{w_t}$ είναι ο χρόνος που χρειάστηκε για τους τοπικούς υπολογισμούς κάθε επεξεργαστή και $\mathbf{h_t}$ είναι το μέγιστο πλήθος μηνυμάτων που στάλθηκαν ή παραλήφθηκαν από κάποιο επεξεργαστή. Για να βρούμε τον χρόνο εκτέλεσης όλου του αλγόριθμου πρέπει να αθροίσουμε προσεχτικά το χρόνο εκτέλεσης των διαφόρων υπερβημάτων που αποτελούν τον αλγόριθμο.

Σχήμα 2.1 [2,4]

(Στο σχήμα αυτό παρουσιάζεται η εκτέλεση ενός αλγόριθμου στο μοντέλο BSP**)**



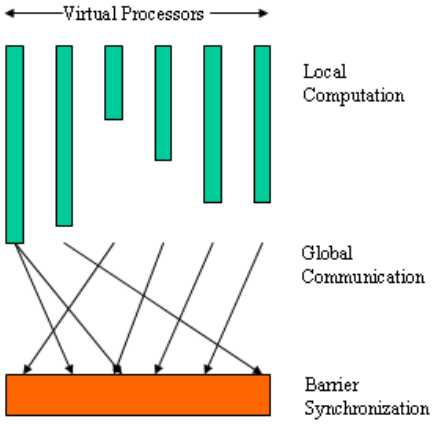
Το μπλε βέλος στο σχήμα παρουσιάζει το χρόνο που χρειάζεται για τους τοπικούς υπολογισμούς που κάνει ο κάθε επεξεργαστής πάνω στα δεδομένα που έχει αποθηκευμένα στην τοπική του μνήμη. Παρατηρούμε ότι ο χρόνος αυτός μπορεί να είναι διαφορετικός στον κάθε επεξεργαστή, στο κάθε υπερβήμα. Με το πράσινο βέλος παρουσιάζεται ο χρόνος που απαιτείται για την επικοινωνία και με το κόκκινο φαίνεται ο χρόνος που απαιτείται για το συγχρονισμό μεταξύ των επεξεργαστών στο τέλος κάθε υπερβήματος.

2.4 Ανάλυση επιπλέον θεμάτων για το μοντέλο BSP

2.4.1 Προγραμματιστική μορφή στο μοντέλο BSP

Στο μοντέλο αυτό υπάρχουν δύο δομές, η κάθετη (vertical structure) και η οριζόντια (horizontal structure) [**6**].

Σχήμα 2.2 (Κάθετη και οριζόντια δομή) [**6**]



Στην **κάθετη δομή** ανήκει η ακολουθιακή σύνθεση των υπερβημάτων. Δηλαδή συμπεριλαμβάνονται όλοι οι τοπικοί υπολογισμοί των επεξεργαστών, η επικοινωνία μεταξύ τους καθώς και ο φραγμένος συγχρονισμός που γίνεται στο τέλος κάθε υπερβήματος.

Στην **οριζόντια δομή** υπάρχει η συνεργασία μεταξύ ενός σταθερού πλήθους εικονικών επεξεργαστών. Επιπλέον οι διεργασίες δεν βρίσκονται σε κάποια συγκεκριμένη σειρά και η τοπικότητα δεν παίζει κανένα ρόλο στον διαμοιρασμό των διεργασιών στους επεξεργαστές (p= πλήθος επεξεργαστών).

2.4.2 Ο τρόπος με τον οποίο διεξάγεται η επικοινωνία

Υπάρχει η δυνατότητα να περιορίζεται ο χρόνος που χρειάζεται για την μετάδοση δεδομένων μεταξύ των επεξεργαστών μέσα σε κάποια χρονικά όρια και σκεπτόμενοι τις επικοινωνιακές ενέργειες ενός υπερβήματος σαν ένα σύνολο.
Το h εκφράζει μέγιστο πλήθος μηνυμάτων που στέλνονται ή παραλαμβάνονται από τους επεξεργαστές, και η επικοινωνία που υπάρχει μεταξύ τους χαρακτηρίζεται σαν μια h-σχέση.
Το BSP μοντέλο δεν βρίσκει διαφορά μεταξύ της αποστολής ενός μηνύματος μεγέθους m και m μηνυμάτων μεγέθους 1.

2.4.3 Το κόστος του φραγμένου συγχρονισμού

Γενικά υπάρχει η θεωρία ότι είναι πολύ ακριβός και είναι προτιμότερο να αποφεύγεται. Παρά όλα αυτά αρκετοί από τους δημιουργούς του BSP ισχυρίζονται ότι ο φραγμένος συγχρονισμός δεν είναι τόσο ακριβώς όσο πιστεύεται [**6**].
Το κόστος ενός φραγμένου συγχρονισμού έχει δύο μέρη. Το κόστος που οφείλεται στην ποικιλία των χρόνων των υπολογιστικών βημάτων που γίνονται, και το κόστος για την επίτευξη μιας καθολικής συνεπής κατάστασης (globally-consistent state) σε όλους τους επεξεργαστές

2.4.4 Στο μοντέλο BSP υπάρχουν δύο μορφές προγραμματισμού

- **Αυτόματος τρόπος (automatic mode)**
  Το πρόγραμμα είναι γραμμένο σε γλώσσα υψηλού επιπέδου, που αποκρύπτει από το χρήστη θέματα που αφορούν τη μνήμη [**3,8**]. Είναι και ο

τρόπος που χρησιμοποιήσαμε για την δημιουργία των αλγόριθμων μας, διότι ήταν πολύ βασικό το στοιχείο ότι δεν ασχολείται με πληροφορίες του χαμηλού επιπέδου της αρχιτεκτονικής του υπολογιστή και είναι αρκετά φιλικός τρόπος, λόγω του ότι τα προγράμματα γράφονται σε γλώσσες υψηλού επιπέδου.

- **Άμεσος τρόπος (direct mode)**

  Χρησιμοποιώντας αυτή την μορφή προγραμματισμού ο προγραμματιστής διατηρεί έλεγχο για την δέσμευση της μνήμης. Επιπρόσθετα χρησιμοποιώντας τον **άμεσο τρόπο** για προγραμματισμό, μικροί σταθεροί όροι πολλαπλασιασμού είναι σημαντικοί κατά την ώρα της εκτέλεσης.

  Ο άμεσος τρόπος είναι χρήσιμος σε περιπτώσεις, στις οποίες μικροί πολλαπλασιαστικοί σταθεροί παράγοντες είναι σημαντικοί και όταν μικρά στιγμιότυπα προβλημάτων μπορούν να εκτελεστούν πιο αποδοτικά με τον άμεσο τρόπο. Όταν η διαθέσιμη μηχανή που θα χρησιμοποιηθεί έχει υψηλό g τότε επίσης προτιμάται ο άμεσος τρόπος [**3,8**]**.**

2.4.5 Τυπικές τιμές για τα g και l για παράλληλους υπολογιστές

Στους παράλληλους υπολογιστές παρατηρούμε ότι είναι δύσκολο να πετύχουμε αποδοτική επικοινωνία όταν στέλλονται πολλά δεδομένα σε πολλούς προορισμούς. Ας μελετήσουμε το τι συμβαίνει όταν έχουμε μια p- σχέση. Καμία αποδοτική αρχιτεκτονική δεν μπορεί να παρέχει p^2 καλώδια, επειδή είναι πολύ ακριβό. Έτσι χρησιμοποιούνται πιο λίγες συνδέσεις. Η τιμή του g αποτελεί ένα καλό μέτρο του χαμηλότερου ορίου του ρυθμού επικοινωνίας μιας αρχιτεκτονικής.

Οι τιμές που παίρνει το g σε μια 1-σχέση και μια p-σχέση διαφέρουν ελάχιστα. Αυτό συνεπάγεται ότι η απόδοση του δικτύου όταν έχουμε 1-σχέση δεν είναι και τόσο καλή.

Πίνακας 2.1 (Παρουσιάζει τις διάφορες τιμές που μπορεί να πάρει το g και το l στις διάφορες παράλληλες μηχανές **[7]**)

Machine	Mflops			p	l		g(local)		g(total exch.)		n1/2
					flops	µs	flop/word	µs/word	flop/word	µs/word	words
	⌊s⌋	⌈s⌉	s								
SGI PowerChallenge	53	94	74	1	226	3.1	0.5	0.007	0.5	0.007	80
				2	1132	15.3	9.8	0.13	10.2	0.14	12
				3	1496	20.2	8.9	0.12	9.5	0.13	12
				4	1902	25.7	9.8	0.13	9.3	0.13	12
Gray T3D	5	19	12	1	68	5.6	0.3	0.02	0.3	0.02	94
				2	164	13.5	0.7	0.06	1.0	0.08	71
				4	168	13.9	0.7	0.06	0.8	0.65	66
				8	175	14.4	0.8	0.07	0.8	0.65	59
				9	383	31.7	0.9	0.07	1.2	0.10	39
				16	181	14.9	0.9	0.07	1.0	0.08	61
				25	486	40.2	1.1	0.09	1.5	0.13	26
				32	201	16.6	1.1	0.09	1.4	0.12	28
				64	148	12.3	1.0	0.09	1.7	0.14	27
				128	301	24.9	1.1	0.09	1.8	0.15	20
				256	387	32.1	1.2	0.11	2.4	0.19	15
IBM SP2 (switch)	25	27	26	1	244	9.4	1.3	0.05	1.3	0.05	7
				2	1903	73.2	6.3	0.24	7.8	0.30	6
				4	3583	137.8	6.4	0.25	8.0	0.31	7
				8	5412	208.2	6.9	0.27	11.4	0.43	6
Multiprocessor Sun	3.8	16.4	10.1	1	24	2.4	0.4	0.04	0.4	0.04	7
				2	54	5.3	3.0	0.29	3.4	0.34	7
				3	74	7.4	2.9	0.29	4.1	0.41	8
				4	118	11.7	3.3	0.32	4.1	0.41	11
Hitachi SR2001	2.3	8.5	5.4	1	31	5.6	0.2	0.05	0.2	0.05	16
				2	1165	216.1	2.6	0.50	3.0	0.54	8
				4	2299	426.1	2.8	0.53	3.0	0.56	8
				8	3844	712.1	3.0	0.54	3.1	0.59	8
				16	4638	911.4	3.0	0.55	3.0	0.55	8
				32	6906	1321.7	4.7	0.90	4.9	0.92	6
Convex Exemplar			10.5	1	60	5.8	0.16	0.02			18
				2	21373	2035	8.3	0.8			6
				4	64457	6138	9.2	0.9			7
				8	194476	18521	11.3	1.2			9
Digital Alpha Farm			10.1	1	29	2.9	0.3	0.03			17
				2	17202	1703.1	81.1	8.0			4
				3	34356	3401.6	83.0	8.2			4
				4	47109	4664.3	81.3	8.1			4
Parsytec GC			19.3	1	98	5.1	1.0	0.05	1.0	0.05	16
				2	6309	325	109	5.6	113	5.9	3
				4	23538	1219	190	9.9	143	7.4	3
				8	29080	1506	252	13.1	254	13.2	3
				16	224977	11600	253	13.1	342	17.7	3
				32	130527	6700	272	14.1	658	34.1	3
IBM SP2 (ethernet)	25	27	26	1	241	9.3	1.3	0.05	1.3	0.05	8
				2	18759	721.5	182.1	7.0	183.6	7.1	3
				4	39025	1500.9	388.2	14.9	628.2	24.2	5
				8	88795	3415.2	1246.6	47.3	1224.1	47.1	2

ΚΕΦΑΛΑΙΟ 3

Περιγραφή γλώσσας προγραμματισμού PuBSP

Στόχος του Κεφαλαίου 3 είναι η παρουσίαση της προγραμματιστικής γλώσσας PuBSP που θα χρησιμοποιηθεί στην δημιουργία των αλγόριθμων. Στο Υποκεφάλαιο 3.1 παρουσιάζεται η βιβλιοθήκη **PUB-LIBRARY** [**1**] και γίνεται επεξήγηση διάφορων προγραμματιστικών εντολών που υπάρχουν σε αυτή. Ακολούθως στο Υποκεφάλαιο 3.2 παρουσιάζεται ο ψευδοκώδικας μιας υλοποιημένης συνάρτησης, στα πλαίσια της διπλωματικής εργασίας και επεξηγείται η προγραμματιστική γλώσσα που χρησιμοποιείται στον αλγόριθμο που παρουσιάστηκε προηγουμένως. Για σκοπούς καλύτερης εξοικείωσης με τη γλώσσα παρουσιάζονται και επεξηγόνται και κάποιες επιπρόσθετες εντολές.

3.1 Βιβλιοθήκη PUB-LIBRARY και επεξήγηση προγραμματιστικών εντολών

Στόχος στο σημείο αυτό είναι να γίνει κάποια παρουσίαση της γλώσσας προγραμματισμού που χρησιμοποιείται. Συγκεκριμένα χρησιμοποιείται η γλώσσα προγραμματισμού C με κάποιες επιπρόσθετες εντολές που υπάρχουν στην βιβλιοθήκη PUB-Library, η οποία είναι μια από τις βιβλιοθήκες που υπάρχουν για υλοποίηση αλγόριθμων σύμφωνα με το μοντέλο παράλληλου υπολογισμού BSP.

Η βιβλιοθήκη PUB-Library (Paderborn University BSP-Library [**1**]) είναι μια βιβλιοθήκη της C (C-library) η οποία περιέχει συναρτήσεις επικοινωνίας (communication routines). Χρησιμοποιώντας αυτές τις συναρτήσεις είναι δυνατή η υλοποίηση αλγόριθμων που είναι σχεδιασμένοι σύμφωνα με το μοντέλο BSP. Οι αλγόριθμοι που είναι υλοποιημένοι σύμφωνα με το μοντέλο BSP διαιρούνται σε διάφορα κομμάτια που ονομάζονται υπερβήματα (supersteps). Σε κάθε υπερβήμα οι επεξεργαστές μπορούν να δουλεύουν σε τοπικά δεδομένα και να στέλνουν μηνύματα. Στο τέλος κάθε υπερβήματος γίνεται ένας φραγμένος συγχρονισμός (barrier synchronization), κατά τον οποίο όλοι οι επεξεργαστές παίρνουν τα μηνύματα που πήραν από κάποιους άλλους επεξεργαστές, στο προηγούμενο υπερβήμα.

Η PUB περιέχει κάποιες συναρτήσεις για μετάδοση μηνυμάτων (message passing), αλλά και για απομακρυσμένη πρόσβαση μνήμης (remote memory access). Επιπρόσθετα, επιτρέπει την δημιουργία ανεξάρτητων BSP αντικειμένων, όπου το καθένα από αυτά αντιπροσωπεύει έναν εικονικό BSP υπολογιστή. Αυτή η ιδιότητα που μόλις προαναφέραμε κάνει την βιβλιοθήκη πολύ ευέλικτη.

Χάριν παραδείγματος, πιο κάτω παρουσιάζεται ο ψευδοκώδικας για την συνάρτηση που υλοποιεί τον αλγόριθμο μετάδοσης δεδομένων (broadcast). Σύμφωνα με αυτόν τον αλγόριθμο, ένας επεξεργαστής, συγκεκριμένα ο επεξεργαστής με ταυτότητα ίση με το 0 (pid=0), δηλαδή ο πρώτος επεξεργαστής, στέλνει το δεδομένο που κρατά στην τοπική του μνήμη σε όλους τους υπόλοιπους επεξεργαστές. Σε αυτό τον αλγόριθμο υπάρχει μόνο ένα

υπερβήμα, στο οποίο όλοι οι επεξεργαστές ενημερώνονται για την πληροφορία που έχει αποθηκευμένη στην τοπική του μνήμη ο επεξεργαστής με ταυτότητα 0.
Με την παρουσίαση του πιο κάτω αλγόριθμου, θα παρουσιαστούν και κάποιες σημαντικές εντολές που υπάρχουν στην βιβλιοθήκη PUB, ώστε να εξοικειωθούμε με την βιβλιοθήκη αυτή και με αλγόριθμους υλοποιημένους σύμφωνα με το μοντέλο παράλληλης επεξεργασίας BSP.

3.2 Ψευδοκώδικας για συνάρτηση που υλοποιεί μετάδοση δεδομένων:

```
int argc;
char** argv;
FILE *f1;

void exclusive_main( t_bsp* bsp, void* param ) {

doTests( bsp, argc, argv );
int pid, nprocs;
int i;
t_bspmsg *msg;
int *data1;
nprocs=bsp_nprocs(bsp);
pid=bsp_pid(bsp);
int data=pid;
int cmsg=0;

f1 = fopen ("out.lis", "wt");
if (pid==0)
  {
   bsp_gsend(bsp, 0, bsp_nprocs(bsp)-1, & data, sizeof(int));
   cmsg= bsp_nprocs(bsp)-1;
```

```
  }
   bsp_sync(bsp);

for (i=0; i<bsp_nmsgs (bsp); i++) {
  msg = bsp_getmsg (bsp, i);
  data1=(int *) bspmsg_data (msg);
  fprintf(f1,"Epeksergastis %d data=%d\n", pid,*data1); }

bsp_sync(bsp);

if(pid==0)
  fprintf(f1,"Stalthikan %d minimata sto ipervima!!\n",cmsg);

bsp_threadsafe_done( bsp );
}
```

Οι εντολές που παρουσιάζονται μαυρισμένες, είναι χαρακτηριστικές εντολές της βιβλιοθήκης PUB, και επεξηγούνται αμέσως πιο κάτω.

nprocs=bsp_nprocs(bsp);

Γενική μορφή: **int bsp_nprocs(t_bsp* bsp)**

Αυτή η συνάρτηση επιστρέφει τον αριθμό των κόμβων μέσα στην ομάδα που είναι ορισμένη από το BSP αντικείμενο [**9**].

Πίνακας 3.1

Παράμετρος	**Τύπος**	**Περιγραφή**
bsp	t_bsp	Δείχτης στο BSP αντικείμενο

pid=bsp_pid(bsp);

Γενική μορφή: **int bsp_pid(t_bsp* bsp)**

Αυτή η συνάρτηση επιστρέφει την ταυτότητα id των κόμβων στο διάστημα 0,...,bsp_nprocs-1 [**9**].

Πίνακας 3.2

Παράμετρος	**Τύπος**	**Περιγραφή**
bsp	t_bsp	Δείχτης στο BSP αντικείμενο

bsp_gsend(bsp, 0, bsp_nprocs(bsp)-1, & data, sizeof(int)); Είναι μια συνάρτηση, η οποία χρησιμοποιείται για την αποστολή μηνυμάτων μεταξύ των επεξεργαστών. Πιο κάτω θα παρουσιαστεί η γενική μορφή της κλήσης της συνάρτησης, μαζί με άλλες παρόμοιες συναρτήσεις.

void bsp_gsend (t_bsp* bsp, int proc_lo, t_proc_id proc_hi, void* buffer, int size)
void bsp_gsendmsg (t_bsp* bsp, int proc_lo, t_proc_id proc_hi, t_bspmsg* msg, int size)

void bsp_hpgsend (t_bsp* bsp, int proc_lo, t_proc_id proc_hi, void* buffer, int size)
void bsp_lsend (t_bsp* bsp, int* dest_procs, int ndest, void* buffer, int size)
void bsp_lsendmsg (t_bsp* bsp, int* dest_procs, int ndest, t_bspmsg* msg, int size)
void bsp_hplsend (t_bsp* bsp, int* dest_procs, int ndest, void* buffer, int size)

Πίνακας 3.3

Παράμετρος	**Τύπος**	**Περιγραφή**
bsp	t_bsp*	Δείχτης στο BSP αντικείμενο.
proc_lo	int	Η μικρότερη ταυτότητα επεξεργαστή (id) από το πλήθος επεξεργαστών στους οποίους θα σταλεί το δεδομένο.
proc_hi	int	Η μεγαλύτερη ταυτότητα επεξεργαστή (id) από το πλήθος επεξεργαστών στους οποίους θα σταλεί το δεδομένο.
dest_procs	int*	Ο πίνακας με τους επεξεργαστές στους οποίους θα σταλεί το δεδομένο.
ndest	int	Αριθμός ταυτοτήτων (ids) στο πλήθος των επεξεργαστών που θα σταλεί κάποιο δεδομένο.
buffer	void*	Δείχτης στο σύνολο (block) στοιχείων που θα σταλούν.
msg	t_bspmsg*	Δείχτης στο μήνυμα.
size	int	Μέγεθος του block στη μνήμη ή του μηνύματος.

Αυτές οι συναρτήσεις στέλνουν ένα μήνυμα σε ένα σύνολο από επεξεργαστές. Το σύνολο των επεξεργαστών που παραλαμβάνουν κάποιο μήνυμα καθορίζεται από το διάστημα [`proc_lo`:`proc_hi`] ή είναι μια λίστα που περιέχει ταυτότητες επεξεργαστών. Τα διάφορα μηνύματα είναι διαθέσιμα στους παραλήπτες μετά από κάποιο συγχρονισμό (**bsp_sync** ή **bsp_oblsync**) [**9**].

Οι δύο συναρτήσεις **bsp_gsend** και **bsp_lsend** στέλνουν το σύνολο στοιχείων της μνήμης (block) που έχει πάνω του κάποιο δείχτη και έχει ένα συγκεκριμένο μέγεθος (size). Αφού κληθούν αυτές οι συναρτήσεις, ο δείχτης στο σύνολο (block) στοιχείων που θα σταλούν (`buffer`), μπορεί να επαναχρησιμοποιηθεί από το χρήστη αμέσως.

Οι δύο συναρτήσεις **bsp_gsendmsg** και **bsp_lsendmsg** στέλνουν ένα μήνυμα (`msg`) με τύπο t_bspmsg*, ο οποίος δημιουργείται από την συνάρτηση **bsp_createmsg.** Εφόσον καλέσουμε μια από τις δύο αυτές συναρτήσεις το μήνυμα (`msg`) δεν είναι διαθέσιμο πλέον για τον χρήστη.

Οι δύο συναρτήσεις **bsp_hpgsend** και **bsp_hplsend** στέλνουν ένα block της μνήμης όπως οι συναρτήσεις **bsp_gsend** και **bsp_lsend**,. Όμως μετά από αυτή την κλήση ο δείχτης στο σύνολο (block) στοιχείων που θα σταλούν (`buffer`), δεν πρέπει να ανανεώνεται από το χρήστη μέχρι τον επόμενο συγχρονισμό.

bsp_sync(bsp);

Γενική μορφή: **void bsp_sync(t_bsp* bsp)**

Πίνακας 3.5

Παράμετρος	**Τύπος**	**Περιγραφή**
Bsp	t_bsp	Δείχτης στο BSP αντικείμενο

Η πιο πάνω συνάρτηση συγχρονίζει το BSP σύνολο από επεξεργαστές. Συγκεκριμένα μπλοκάρει μια συγκεκριμένη απειλή μέχρι [**9**]:

- Όλοι οι κόμβοι στο σύνολο να έχουν καλέσει την συνάρτηση bsp_sync.
- Όλα τα μηνύματα που στάλθηκαν σε κάποιο κόμβο μέσα σε ένα υπερβήμα να έχουν παραληφθεί.
- Όλες οι prefix-λειτουργίες να έχουν τελειώσει.

t_bspmsg *msg;

Γενική μορφή: **t_bspmsg *msg**

Με την πιο πάνω εντολή δημιουργείται το αντικείμενο BSP [**9**].

data1=(int *) bspmsg_data (msg);

Γενική μορφή: **void* bspmsg_data(t_bspmsg* msg)**

Πίνακας 3.7

Παράμετρος	**Τύπος**	**Περιγραφή**
msg	t_bspmsg*	Δείχτης στο BSP αντικείμενο

Αυτή η συνάρτηση επιστρέφει έναν δείχτη προς τα δεδομένα του μηνύματος [**9**].

bsp_threadsafe_done(bsp);

Γενική μορφή: **void bsp_threadsafe_done(t_bsp* bsp)**

Πίνακας 3.8

Παράμετρος	**Τύπος**	**Περιγραφή**
bsp	t_bsp	Δείχτης στο BSP αντικείμενο

Αυτή η συνάρτηση καταστρέφει το υποσύνολο (subgroup) που δημιουργείται από την συνάρτηση **bsp_threadsafe_dup** και στην συνέχεια ελευθερώνει το BSP αντικείμενο [**9**].

Σημείωση: Ποτέ δεν πρέπει να στέλνεται μήνυμα σε ένα είδη κατεστραμμένο αντικείμενο BSP. Η συμπεριφορά της βιβλιοθήκης PUB (PUB-Library) σε τέτοια περίπτωση, θα είναι απροσδιόριστη.

Παρουσίαση επιπρόσθετων εντολών

Πιο κάτω θα παρουσιαστούν ακόμη κάποιες επιπρόσθετες αλλά σημαντικές εντολές που παρουσιάζονται στην βιβλιοθήκη PUB και μπορούν να φανούν χρήσιμες στα προγράμματα, που υλοποιούνται σύμφωνα με το μοντέλο BSP [**9**].

Οι πιο κάτω δύο συναρτήσεις χρησιμοποιούνται για αρχικοποίηση της βιβλιοθήκης.

Γενική μορφή: **void bsplib_saveargs(int* argc, char*** argv)**

Πίνακας 3.9

παράμετρος	**τύπος**	**Περιγραφή**
argc	int*	αριθμός τον παραμέτρων του προγράμματος
argv	char***	δείχτης σε παραμέτρους του προγράμματος

Σε μερικά συστήματα (mpi, tcpip) η PUB χρειάζεται να γνωρίζει τις παραμέτρους του προγράμματος. Σε αυτά λοιπόν τα συστήματα πρέπει να καλέσουμε την πιο πάνω συνάρτηση πριν καλεστεί η συνάρτηση **bsplib_init.** Στα υπόλοιπα συστήματα, αυτή η συνάρτηση δεν χρησιμεύει πουθενά, όμως μπορούμε να την γράφουμε πάντα πριν την **bsplib_init** σε όλες τις αρχιτεκτονικές, ώστε να έχουμε με τον τρόπο αυτό ευέλικτο κώδικα προς όλες τις αρχιτεκτονικές.

Γενική μορφή: **void bsplib_init(t_bsplib_params* parameter, t_bsp* bsp)**

Πίνακας 3.10

παράμετρος	**τύπος**	**Περιγραφή**
parameter	t_bsplib_params*	Παράμετρος
bsp	t_bsp*	Δείχτης σε αντικείμενο bsp για αρχικοποίηση

Αυτή η συνάρτηση χρησιμοποιείται για την αρχικοποίηση της συνάρτησης PUB. Καλείται πάντα πριν οποιαδήποτε άλλη συνάρτηση της βιβλιοθήκης, εκτός βέβαια από τις συναρτήσεις bsplib_param_init και **bsplib_saveargs** [**9**].

Ως παράμετρο στην συνάρτηση αυτή έχουμε το BSPLIB_STDPARAMS ή ένα δείχτη στην δομή **t_bsplib_params** που αρχικοποιείται από την συνάρτηση bsplib_params_init.

Σε μερικά συστήματα όπως το mpi και το tcpip η βιβλιοθήκη PUB χρειάζεται να γνωρίζει τις παραμέτρους του προγράμματος και για αυτό το λόγο χρησιμοποιείται και η συνάρτηση **bsplib_saveargs** που αναφέραμε πιο πάνω.

Γενική μορφή: **void bsplib_done()**

Αυτή η συνάρτηση "καθαρίζει" όλες τις πηγές της PUB (PUB resources). Αφού καλεστεί αυτή η συνάρτηση, δεν μπορούμε πλέον να χρησιμοποιήσουμε οποιαδήποτε άλλη συνάρτηση της βιβλιοθήκης [**9**].

Οι πιο κάτω δύο συναρτήσεις που θα παρουσιαστούν χρησιμοποιούνται για μετάδοση μηνυμάτων (message passing).

Γενική μορφή: **t_bspmsg* bsp_findmsg (t_bsp* bsp, int proc_id, int index)**
Γενική μορφή: **t_bspmsg* bsp_getmsg (t_bsp* bsp, int index)**

Πίνακας 3.11

παράμετρος	**τύπος**	**Περιγραφή**
bsp	t_bsp*	δείχτης στο BSP αντικείμενο
proc_id	t_proc_id	Ταυτότητα του επεξεργαστή που αποστέλλει το μήνυμα
index	int	Δείχτης στο μήνυμα (που ξεκινά από το 0)

Αυτές οι συναρτήσεις επιστρέφουν δείχτη προς κάποιο μήνυμα το οποίο παραλήφθηκε στο προηγούμενο υπερβήμα. Όλα τα μηνύματα είναι διαθέσιμα μέχρι τον επόμενο φραγμένο συγχρονισμό. Οπόταν όταν επιθυμούμε να το κρατήσουμε αυτό το μήνυμα και για αργότερα, πρέπει να το αντιγράψουμε στην τοπική μνήμη του επεξεργαστή μας. Για επεξεργασία των μηνυμάτων χρησιμοποιούνται οι εξής συναρτήσεις: bspmsg_ data, bspmsg_ size και bspmsg_ src .

bsp_get: Όλα τα μηνύματα που βρίσκονται μέσα στην λίστα μηνυμάτων αριθμούνται από το 0 μέχρι το bsp_nmsgs-1, όπου είναι το πλήθος των επεξεργαστών μειωμένο κατά ένα. Επιπλέον χρησιμοποιώντας την συνάρτηση **bsp_get** , παίρνεις το μήνυμα στην θέση index.

bsp_find: Με την συνάρτηση αυτή μπορούμε να ελέγξουμε την λίστα μηνυμάτων μας που προέρχεται από τον επεξεργαστή με ταυτότητα proc_id. Αν μέσα στην λίστα αυτή υπάρχουν n μηνύματα αριθμούνται από το 0 μέχρι το n-1.

Πιο κάτω θα παρουσιαστούν συναρτήσεις οι οποίες χρησιμοποιούνται για μακρινή πρόσβαση μνήμης (remote memory access).

Γενική μορφή: **void bsp_pop_reg(t_bsp* bsp, void* ident)**

Πίνακας 3.12

παράμετρος	**τύπος**	**Περιγραφή**
bsp	t_bsp*	δείχτης προς το BSP αντικείμενο
ident	void*	τοπική διεύθυνση της μεταβλητής, η ίδια με αυτήν στην κλήση της συνάρτησης bsp_push_reg

Αυτή η συνάρτηση διαγράφει την μεταβλητή που καταχωρήθηκε από την συνάρτηση **bsp_push_reg** για καθολική πρόσβαση.

Γενική μορφή: **void bsp_push_reg(t_bsp* bsp, void* ident, int size)**

Πίνακας 3.13

παράμετρος	**τύπος**	**Περιγραφή**
bsp	t_bsp*	Δείχτης σε BSP αντικείμενο
ident	void*	Τοπική μνήμη για μεταβλητή
size	int	Μέγεθος μεταβλητής το οποίο μπορεί να είναι διαφορετικό σε διαφορετικούς επεξεργαστές.

Αυτή η συνάρτηση καταχωρεί την μεταβλητή για καθολική πρόσβαση. Ο κάθε επεξεργαστής μπορεί να έχει πρόσβαση στις μεταβλητές άλλων επεξεργαστών χρησιμοποιώντας την συνάρτηση bsp_get (**bsp_hpget**) και **bsp_put** (**bsp_hpput**) [**9**].

Γενική μορφή: **void bsp_put (t_bsp* bsp, int destPID, void* src, void* dest, int offset, int nbytes)**
Γενική μορφή: **void bsp_hpput (t_bsp* bsp, int destPID, void* src, void* dest, int offset, int nbytes)**

Πίνακας 3.14

παράμερος	**τύπος**	**Περιγραφή**
bsp	t_bsp*	Δείχτης σε BSP αντικείμενο
destPID	int	Ταυτότητα του επεξεργαστή προορισμού
src	void*	Δείχτης στην μνήμη πηγή
dest	void*	Δείχτης στην μεταβλητή προορισμός
offset	int	offset στην μεταβλητή προορισμό
nbytes	int	Αριθμός των bytes που πρέπει να αντιγραφούν

Αυτή η συνάρτηση αντιγράφει nbytes bytes από την τοπική μνήμη του επεξεργαστή προορισμού, μέσα στην μεταβλητή ενός επεξεργαστή με ταυτότητα destPID και ο οποίος είναι ο επεξεργαστής προορισμός [**9**].

bsp_put: Η αντιγραφή ολοκληρώνεται μετά τον συγχρονισμό και οποιεσδήποτε κλήσεις της συνάρτησης **bsp_get.**

bsp_hpput: Αυτή η συνάρτηση είναι μια ειδική περίπτωση της πιο πάνω συνάρτησης. Η αντιγραφή γίνεται ασυγχρόνιστα, οπόταν και ολοκληρώνεται μετά την επόμενη κλήση της συνάρτησης **bsp_sync** ή της συνάρτησης **bsp_oblsync**. Να σημειώσουμε ότι η μνήμη – πηγή δεν μπορεί να αλλαχτεί προηγουμένως.

Γενική μορφή: **void bsp_get (t_bsp* bsp, int srcPID, void* src, int offset, void* dest, int nbytes)**

Γενική μορφή: **void bsp_hpget (t_bsp* bsp, int srcPID, void* src, int offset, void* dest, int nbytes)**

Πίνακας 3.15

παράμετρος	**τύπος**	**Περιγραφή**
bsp	t_bsp*	Δείχτης στο BSP αντικείμενο
srcPID	int	Ταυτότητα του επεξεργαστή πηγή
src	void*	Δείχτης στην μεταβλητή πηγή
offset	int	Υπερχείλιση (offset) της μεταβλητής πηγή
dest	void*	Δείχτης προς την μνήμη προορισμός
nbytes	int	Αριθμός των bytes που θα αντιγραφούν

Αυτή η συνάρτηση αντιγράφει `nbytes` bytes από την μεταβλητή πηγή `src` στον επεξεργαστή με ταυτότητα `srcPID`, στην τοπική του μνήμη `dest`.

bsp_get: Η μεταβλητή πηγή αντιγράφεται όταν ο επεξεργαστής που έχει το ρόλο της πηγής srcPID καλέσει την συνάρτηση calls bsp_sync ή την συνάρτηση **bsp_oblsync** και πριν οποιαδήποτε εκτέλεση της συνάρτησης **bsp_put**.

bsp_hpget: Αυτή η συνάρτηση είναι μια ειδική περίπτωση της πιο πάνω. Σύμφωνα με αυτή την συνάρτηση η πηγή και ο προορισμός δεν μπορούν να αλλάξουν το δεδομένο που κρατούν υποθηκευμένο μέχρι τον επόμενο συγχρονισμό.

ΚΕΦΑΛΑΙΟ 4

Αλγόριθμοι Μετάδοσης Δεδομένων

Στο Κεφάλαιο 4 παρουσιάζεται το πρόβλημα της μετάδοσης μηνύματος/στοιχείου μεταξύ των επεξεργαστών. Για την επίλυση του προβλήματος αυτού υλοποιήθηκαν τρεις διαφορετικοί αλγόριθμοι μετάδοσης. Έτσι το κεφάλαιο αυτό αποτελείται από το Υποκεφάλαιο 4.1 στο οποίο γίνεται μια περιγραφή του προβλήματος με το οποίο θα καταπιαστούμε. Στο Υποκεφάλαιο 4.2 περιγράφεται η παράλληλη λύση του προβλήματος και οι αλγόριθμοι που μπορούν να χρησιμοποιηθούν για την επίλυση αυτού του προβλήματος. Να σημειώσουμε στο σημείο αυτό ότι οι κώδικες για τους αλγόριθμους βρίσκονται στο Παράρτημα Α. Στο Υποκεφάλαιο 4.3 γίνονται κάποιες παρατηρήσεις γύρω από την αναμενόμενη συμπεριφορά των αλγόριθμων και στα 4.4 και 4.5 Υποκεφάλαια παρουσιάζονται τα αποτελέσματα των μετρήσεων και οι γραφικές παραστάσεις. Ακολούθως στο 4.6 γράφονται τα συμπεράσματα των μετρήσεων και γραφικών που πάρθηκαν από την εκτέλεση των αλγόριθμων σε διάφορες συνθήκες εκτέλεσης τους. Τέλος θέλοντας να ελεγχθεί η συμπεριφορά των αλγόριθμων και σε μεγαλύτερα μεγέθη σταλμένων μηνυμάτων ακολουθεί το Υποκεφάλαιο 4.7 με τα Υποκεφάλαια του 4.7.1, 4.7.2

και 4.7.3, στα οποία και πάλι παρουσιάζονται μετρήσεις, γραφικές και συμπεράσματα αντίστοιχα.

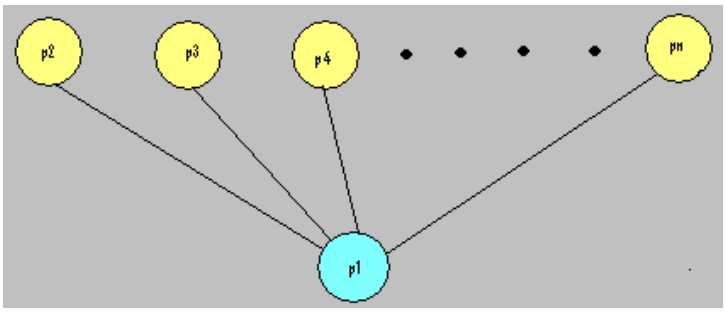
4.1 Πρόβλημα μετάδοσης δεδομένων: Κάποιο μήνυμα είναι αποθηκευμένο στην τοπική μνήμη ενός επεξεργαστή. Πρέπει όμως να γίνει γνωστό σε όλους τους επεξεργαστές. Οπόταν οι επεξεργαστές που κρατούν αυτό το μήνυμα πρέπει να το στείλουν και στους υπόλοιπους.

Για το μοντέλο BSP θα παρουσιάσουμε τρεις διαφορετικούς τρόπους που μπορούμε να το επιτύχουμε αυτό (μετάδοση δεδομένων μεταξύ επεξεργαστών), ο καθένας από τους οποίους χρειάζεται διαφορετικό χρόνο εκτέλεσης και μεταδίδει με διαφορετικό τρόπο τα διάφορα μηνύματα μεταξύ των επεξεργαστών.

4.2 Παράλληλη λύση

Αλγόριθμος 1: *Ένας επεξεργαστής, έστω ο P_1 στέλνει το μήνυμα/στοιχείο σε όλους τους υπόλοιπους επεξεργαστές.*

Σχήμα 4.1: (έστω p, Παρουσιάζεται το υπερβήμα στο οποίο ο πρώτος επεξεργαστής στέλνει το μήνυμα/στοιχείο σε όλους τους υπόλοιπους)



Για το συγκεκριμένο αλγόριθμο χρειάζεται να γίνει μόνο ένα υπερβήμα, μετά τη ολοκλήρωση του οποίου, όλοι πλέον οι επεξεργαστές θα έχουν στην τοπική τους μνήμη το μήνυμα/στοιχείο.

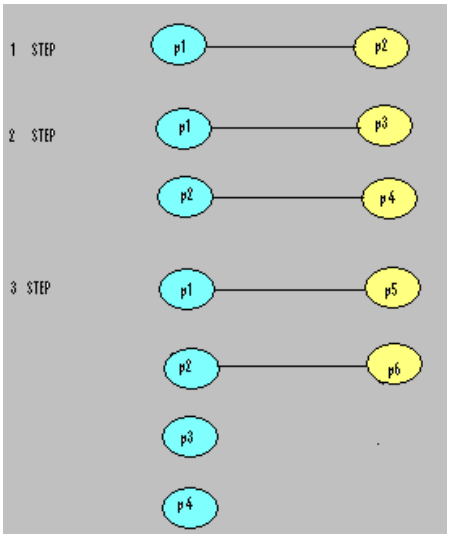
Ο χρόνος που θα χρειαστεί για την ολοκλήρωση του αλγόριθμου είναι ***(p-1)g+L***. Αυτό συμβαίνει διότι γίνεται μόνο ένα υπερβήμα στο οποίο στέλνονται (p-1) μηνύματα και άρα το h=p-1 και συνεπώς έχουμε μια p-1 σχέση.

Υπολογιστικό μέρος: 0
Επικοινωνιακό μέρος: (p-1)g ((p-1)-σχέση, h = p-1)
Συγχρονισμός: L

Συνολικός χρόνος υπολογισμού: (p-1)g + L

Αλγόριθμος 2: Ο επεξεργαστής P_j όπου $j<=2^{i-1}$ στέλνει το μήνυμα/στοιχείο στον επεξεργαστή $P_{j+2}{}^{i-1}$.

Σχήμα 4.2 (έστω p=6, Παρουσιάζονται τα υπερβήματα κατά την εκτέλεση του δεύτερου αλγόριθμου)



Για την ολοκλήρωση αυτού του αλγόριθμου χρειάζονται να γίνουν log p υπερβήματα, όπου p είναι το πλήθος των επεξεργαστών που είναι διαθέσιμοι για την εκτέλεση του αλγόριθμου αυτού. Ο χρόνος εκτέλεσης του αλγόριθμου είναι ***(g+L)log p*** .

Στον αλγόριθμο αυτό υπάρχει μια 1-σχέση, λόγω του ότι σε κάθε υπερβήμα στέλλεται ένα μήνυμα μόνο από κάποιους επεξεργαστές, οπόταν το h=1. Για την ολοκλήρωση του αλγόριθμου χρειάζονται logp υπερβήματα.

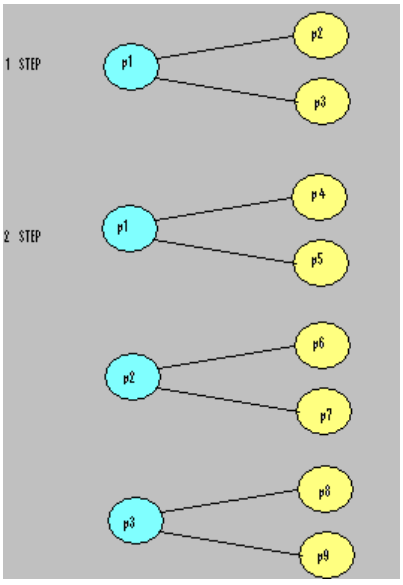
Χρόνος για broadcast2:

Υπολογιστικό μέρος: 0
Επικοινωνιακό μέρος: g (1-σχέση, h = 1)
Συγχρονισμός: L

Συνολικός χρόνος υπολογισμού: (g + L) log p

Αλγόριθμος 3: Ο επεξεργαστής P_j, όπου $j<=k^i$ στέλνει το μήνυμα/στοιχείο σε k-1 διαφορετικούς επεξεργαστές, $P_{j+ki\ \lambda}$ όπου $1<=\lambda<=k-1$. Στο τέλος κάθε υπερβήματος i υπάρχουν k^i+1 επεξεργαστές που γνωρίζουν το μήνυμα.

Σχήμα 4.3: (έστω p=9 και k=3, παρουσιάζονται τα υπερβήματα κατά την εκτέλεση του τρίτου αλγόριθμου)



Για την ολοκλήρωση του αλγόριθμου πρέπει να γίνουν $\log_k p$ υπερβήματα, όπου p είναι το πλήθος των επεξεργαστών που είναι διαθέσιμοι για την εκτέλεση του αλγόριθμου αυτού και k είναι η σταθερά μετάδοσης του αλγόριθμου. Σε κάθε υπερβήμα, κάθε επεξεργαστής που γνωρίζει το μήνυμα/στοιχείο μετάδοσης το στέλνει σε άλλους k-1 επεξεργαστές, οπόταν έχουμε μια σχέση k-1 (h=k-1). Έτσι ο χρόνος εκτέλεσης είναι **$((k-1)g+L)\log_k p$,** αφού χρειάζονται να γίνουν $\log_k p$ υπερβήματα.

Υπολογιστικό μέρος: 0
Επικοινωνιακό μέρος: (κ-1)g ((κ-1)-σχέση, h = κ-1)
Συγχρονισμός: L

Συνολικός χρόνος υπολογισμού: $((\kappa-1)g + L) \log_\kappa p$

Μπορούμε να παρατηρήσουμε ότι ο τρίτος αλγόριθμος είναι μια γενική περίπτωση που συμπεριλαμβάνει τους δύο πρώτους αλγόριθμου για την μετάδοση στοιχείων. Δηλαδή για συγκεκριμένες τιμές του k ο αλγόριθμος αυτός γίνεται ο ίδιος με τους δύο άλλους. Συγκεκριμένα όταν το k είναι ίσο με 2, τότε ο αλγόριθμος αυτός είναι ο ίδιος με το δεύτερο αλγόριθμο που παρουσιάστηκε, και όταν το k ισούται με το πλήθος των επεξεργαστών τότε γίνεται ο ίδιος με τον πρώτο αλγόριθμο που παρουσιάστηκε.

4.3 Παρατηρήσεις

- Από τις εξισώσεις του χρόνου εκτέλεσης κάθε αλγόριθμου, φαίνεται ότι και οι τρεις εξαρτώνται από το πλήθος των επεξεργαστών. Συγκεκριμένα όσο αυξάνεται το πλήθος των επεξεργαστών, περιμένουμε να αυξάνεται και ο χρόνος, με τη διαφορά όμως ότι στον πρώτο αλγόριθμο για μετάδοση η αύξηση είναι γραμμική, ενώ στους άλλους είναι λογαριθμική.
- Όταν χρησιμοποιούνται για καθένα από τους τρεις αλγόριθμους περισσότεροι επεξεργαστές, θα στέλνονται περισσότερα μηνύματα μεταξύ των επεξεργαστών και αυτό είναι φυσικό και αναμενόμενο, διότι περισσότεροι επεξεργαστές συνεπάγεται περισσότερη επικοινωνία για να μπορέσει να επιτευχθεί ο σκοπός του αλγόριθμου.

- Στον τρίτο αλγόριθμο που παρουσιάσαμε για μετάδοση δεδομένων, παρατηρούμε ότι υπάρχει μια σταθερά **k,** η οποία αντιπροσωπεύει κάθε φορά τον τρόπο με τον οποίο θα γίνεται η επικοινωνία, και πόσα μηνύματα θα στέλνονται σε κάθε υπερβήμα. Με την αλλαγή αυτής της σταθεράς αναμένουμε να αλλάζει και ο χρόνος που χρειάζεται για την εκτέλεση του ο αλγόριθμος. Πιο συγκεκριμένα, όσο μεγαλώνει αυτή η σταθερά, αναμένουμε να μικραίνει ο χρόνος, διότι στέλνονται στο κάθε υπερβήμα περισσότερα μηνύματα, οπόταν σε λιγότερα υπερβήματα και συνεπώς σε λιγότερο χρόνο γίνεται η μετάδοση σε όλους τους επεξεργαστές.
- Στον πρώτο αλγόριθμο για μετάδοση που παρουσιάσαμε πιο πάνω, παρατηρούμε ότι ο πρώτος επεξεργαστής στέλνει το δεδομένο σε όλους τους επεξεργαστές από το πρώτο υπερβήμα. Αυτός ο αλγόριθμος περιμένουμε να μας δώσει πιο λίγο χρόνο εκτέλεσης, αφού από το πρώτο υπερβήμα τελειώνει η μετάδοση και ολοκληρώνεται ο αλγόριθμος.
- Στον πρώτο αλγόριθμο στέλνονται πάρα πολλά μηνύματα στο πρώτο και μοναδικό του υπερβήμα (μηνύματα προς όλους τους επεξεργαστές), ενώ στους υπόλοιπους περιμένουμε σαν σύνολο να στέλνονται τα ίδια μηνύματα, αλλά σε κάθε υπερβήμα να στέλνονται λιγότερα μηνύματα από ότι στον πρώτο.
- Στον τρίτο αλγόριθμο μπορούμε να παρατηρήσουμε ότι αν το **k** είναι ίσο με το πλήθος των επεξεργαστών τότε και πάλι θα τελειώσει ο αλγόριθμος αυτός σε ένα υπερβήμα, άρα θα χρειάζεται λιγότερος χρόνο εκτέλεσης. Ο χρόνος εκτέλεσης του σε αυτή την τιμή του k, περιμένουμε να είναι περίπου ίσος με του πρώτου αλγόριθμου, όταν χρησιμοποιείται και στους δύο ίδιο πλήθος επεξεργαστών. Αυτό διότι ουσιαστικά ο τρίτος αλγόριθμος γίνεται ο ίδιος με τον πρώτο.
- Με παρόμοιο τρόπο παρατηρούμε ότι αν στον τρίτο αλγόριθμο το k πάρει την τιμή 2, τότε θα μετατραπεί στον δεύτερο αλγόριθμο για μετάδοση που παρουσιάσαμε πιο πάνω. Οπόταν και πάλι αναμένουμε να έχουμε ίδιους χρόνους εκτέλεσης (ίσως παρόμοιοι χρόνοι λόγω κάποιων αλγοριθμικών λεπτομερειών), εφόσον χρησιμοποιείται το ίδιο πλήθος επεξεργαστών και από τους δύο αλγόριθμους. Επίσης όσο αφορά τα μηνύματα αναμένουμε να στέλνονται ακριβώς τα ίδια μηνύματα σε κάθε υπερβήμα.

4.4 Μετρήσεις

Πιο κάτω παρουσιάζονται οι μετρήσεις που αφορούν διάφορες παραμέτρους των αλγόριθμων μας σε διαφορετικές συνθήκες εκτέλεσης τους.
Πρέπει να αναφερθεί ότι κατά την διαδικασία λήψης των μετρήσεων, τα πειράματα έγιναν πάνω σε μια μηχανή , οπόταν χρησιμοποιήθηκαν εικονικοί επεξεργαστές. Επίσης κάθε φορά που παίρναμε τον χρόνο εκτέλεσης του αλγόριθμου για ένα συγκεκριμένο στιγμιότυπο του, τρέχαμε το αλγόριθμο 5 φορές και παίρναμε τον μέσο όρο των χρόνων αυτών , ώστε να έχουμε πιο ρεαλιστικά αποτελέσματα.
Στις μετρήσεις που ακολουθούν πρέπει να σημειωθεί ότι το μέγεθος των μηνυμάτων είναι το μέγεθος ενός ακέραιου αριθμού (4bytes).

Ακολουθούν μετρήσεις για χρόνους εκτέλεσης και πλήθος σταλμένων μηνυμάτων, για διαφορετικό πλήθος επεξεργαστών.

Αλγόριθμος μετάδοσης 1

Πίνακας 4.1

Επεξεργαστές	**Χρόνος**	**Σταλμένα μηνύματα**
2	0.00052	1
4	0.00096	3
8	0.00184	7
10	0.00228	9
20	0.00454	19
30	0.00678	29
50	0.01139	49
70	0.01623	69
90	0.01824	89
100	0.02292	99
150	0.03504	149
175	0.04095	174
190	0.04501	189
195	0.04583	194
199	0.04681	198

Αλγόριθμος μετάδοσης 2

Πίνακας 4.2

Επεξεργαστές	**Χρόνος**				**Σταλμένα μηνύματα**				
		1 υπερβήμα	2 υπερβήμα	3 υπερβήμα	4 υπερβήμα	5 υπερβήμα	6 υπερβήμα	7 υπερβήμα	8 υπερβήμα
2	0.00019	1							
4	0.00085	1	2						
8	0.00236	1	2	4					
10	0.00358	1	2	4	2				
20	0.0086	1	2	4	8	4			
30	0.01298	1	2	4	8	14			
50	0.025	1	2	4	8	16	18		
70	0.03972	1	2	4	8	16	32	6	
90	0.05173	1	2	4	8	16	32	26	
100	0.05764	1	2	4	8	16	32	36	
150	0.09669	1	2	4	8	16	32	64	22
175	0.11293	1	2	4	8	16	32	64	47
190	0.12417	1	2	4	8	16	32	64	62
195	0.12674	1	2	4	8	16	32	64	67
199	0.12953	1	2	4	8	16	32	64	71

Αλγόριθμος μετάδοσης 3 Πίνακας 4.3

Επεξεργαστές	**Χρόνος**				**Σταλμένα μηνύματα**				
		1 υπερβήμα	2 υπερβήμα	3 υπερβήμα	4 υπερβήμα	5 υπερβήμα	6 υπερβήμα	7 υπερβήμα	8 υπερβήμα
2	0.00032	1							
4	0.00012	1	2						
8	0.00296	1	2	4					
10	0.00423	1	2	4	2				
20	0.00973	1	2	4	8	4			
30	0.01474	1	2	4	8	14			
50	0.02793	1	2	4	8	16	18		
70	0.04446	1	2	4	8	16	32	6	
90	0.05728	1	2	4	8	16	32	26	
100	0.06402	1	2	4	8	16	32	36	
150	0.10675	1	2	4	8	16	32	64	22
175	0.12535	1	2	4	8	16	32	64	47
190	0.1362	1	2	4	8	16	32	64	62
195	0.13938	1	2	4	8	16	32	64	67
199	0.14173	1	2	4	8	16	32	64	71

Πιο κάτω θα παρουσιαστούν μετρήσεις για τον τρίτο αλγόριθμο μετάδοσης δεδομένων που παρουσιάστηκε πιο πάνω. Στις μετρήσεις αυτές το πλήθος των επεξεργαστών διατηρείται σταθερό (p=199) και έχουμε την αλλαγή της σταθεράς k κάθε φορά. Στόχος μας είναι να ελέγξουμε πώς επηρεάζεται ο χρόνος εκτέλεσης του αλγόριθμου σε σχέση με το k.

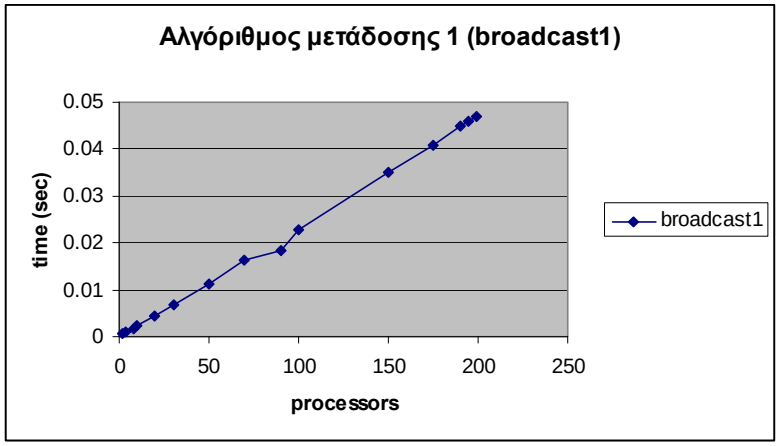
Πίνακας 4.4

k	**Χρόνος**
2	0.10675
5	0.06721
10	0.05801
25	0.04806
40	0.04883
55	0.04887
80	0.04998
100	0.05030
125	0.05094
130	0.05089
150	0.05099
199	0.04577

4.5 Ανάλυση αποτελεσμάτων

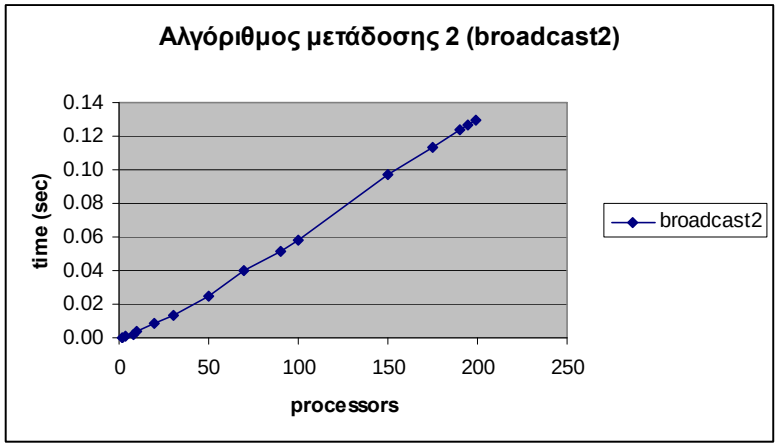
Πιο κάτω παρουσιάζονται οι γραφικές παραστάσεις για τις μετρήσεις που πάρθηκαν για τους τρεις αλγόριθμους μετάδοσης δεδομένων.

Γραφική 4.1



Η γραφική αυτή δείχνει πώς επηρεάζεται ο χρόνος εκτέλεσης του πρώτου αλγόριθμου για μετάδοση που παρουσιάσαμε (broadcast1), σε σχέση με το πλήθος των επεξεργαστών που χρησιμοποιούνται. Παρατηρούμε ότι η γραφική μας είναι μια καμπύλη, που όσο αυξάνονται οι τιμές στον άξονα των τετμημένων, τόσο κατευθύνεται ανοδικά η καμπύλη. Έτσι μπορούμε να συμπεράνουμε, ότι όσο περισσότεροι είναι οι επεξεργαστές που χρησιμοποιούνται από τον αλγόριθμο, τόσο αυξάνεται και ο χρόνος για μετάδοση του δεδομένου, που γνωρίζει αρχικά μόνο ο πρώτος επεξεργαστής, και μεταδίδεται στην συνέχεια και σε όλους τους υπόλοιπους επεξεργαστές. Ακόμη παρατηρούμε ότι η γραφική παράσταση δεν είναι γραμμική, όπως θα περιμέναμε να ήταν, λόγω του ότι ο χρόνος είναι γραμμικός ως προς το πλήθος των επεξεργαστών. Αυτό οφείλεται στο γεγονός ότι τα πειράματα έγιναν σε εικονικούς επεξεργαστές και συνεπώς τα αποτελέσματα δεν είναι και τόσο ρεαλιστικά.

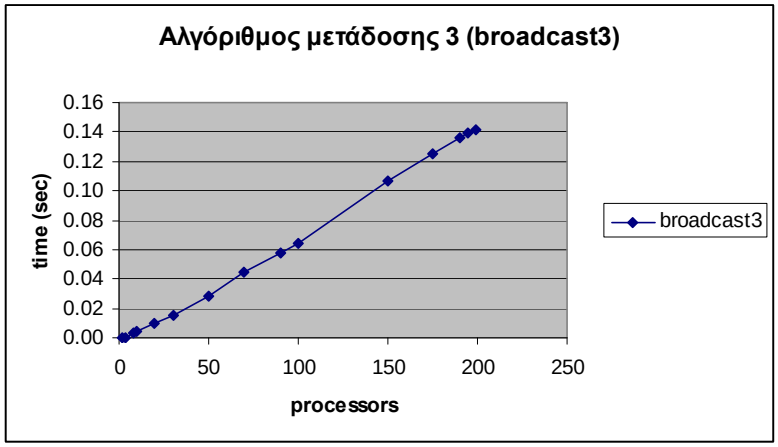
Γραφική 4.2



Η γραφική αυτή δείχνει πώς επηρεάζεται ο χρόνος εκτέλεσης του δεύτερου αλγόριθμου για μετάδοση που παρουσιάσαμε (broadcast2), σε σχέση με το πλήθος των επεξεργαστών που χρησιμοποιούνται. Αναπαριστά μια καμπύλη, που καθώς μεγαλώνουν οι τιμές στον άξονα των x, μεγαλώνουν και οι τιμές στον άξονα των y, έτσι η καμπύλη έχει μια ανοδική πορεία. Έτσι μπορούμε να συμπεράνουμε ότι όσο περισσότεροι είναι οι επεξεργαστές που χρησιμοποιούνται από τον αλγόριθμο, τόσο αυξάνεται και ο χρόνος για μετάδοση του δεδομένου, που γνωρίζει αρχικά μόνο ο πρώτος επεξεργαστής, και μεταδίδεται στην συνέχεια σε όλους τους υπόλοιπους επεξεργαστές.

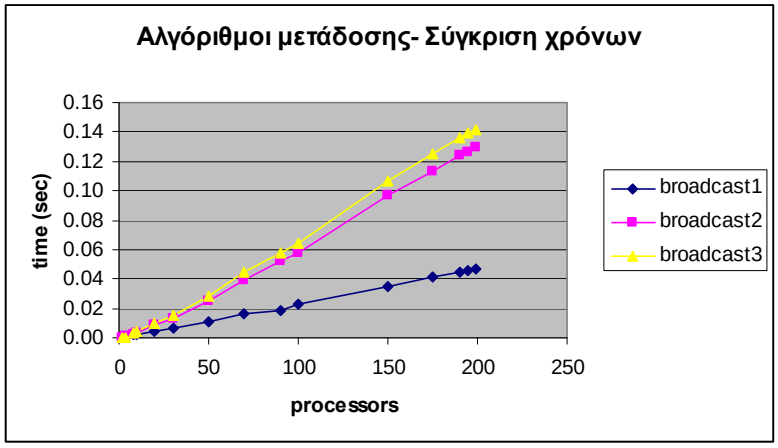
Προηγουμένως παρατηρήσαμε πως ο χρόνος εκτέλεσης του αλγόριθμου είναι λογαριθμικός ως προς το πλήθος των επεξεργαστών που χρησιμοποιούνται, έτσι θα περιμέναμε και η γραφική να ήταν λογαριθμική. Παρατηρούμε ότι δεν είναι ακριβώς λογαριθμική για το λόγο ότι οι μετρήσεις πάρθηκαν χρησιμοποιώντας εικονικούς επεξεργαστές, ενώ η θεωρητική ανάλυση του αλγόριθμου γίνεται βασισμένη στο γεγονός ότι χρησιμοποιούνται πραγματικοί επεξεργαστές.

Γραφική 4.3



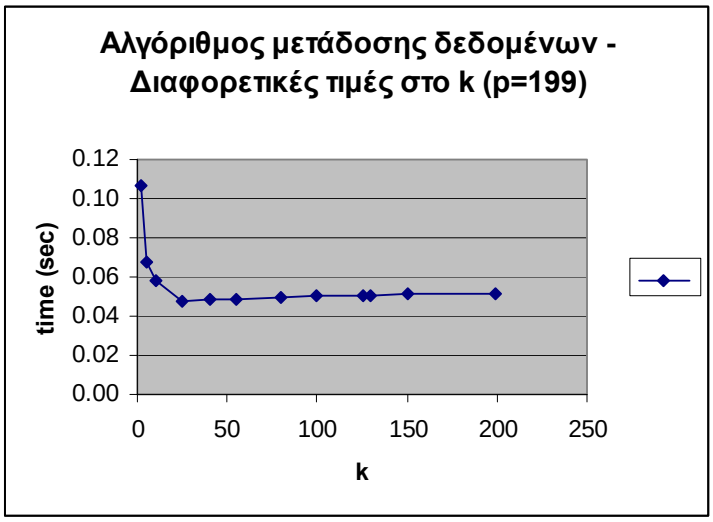
Η γραφική αυτή δείχνει πώς επηρεάζεται ο χρόνος εκτέλεσης του τρίτου αλγόριθμου για μετάδοση που παρουσιάσαμε (broadcast3), σε σχέση με το πλήθος των επεξεργαστών που χρησιμοποιούνται. Από την γραφική παρατηρούμε ότι όσο αυξάνονται οι τιμές στον άξονα των τετμημένων, αυξάνονται και οι τιμές στον άξονα των τεταγμένων. Έτσι μπορούμε να συμπεράνουμε ότι όσο περισσότεροι είναι οι επεξεργαστές που χρησιμοποιούνται από τον αλγόριθμο, τόσο αυξάνεται και ο χρόνος για μετάδοση του δεδομένου, που γνωρίζει αρχικά μόνο ο πρώτος επεξεργαστής, και μεταδίδεται στην συνέχεια σε όλους τους υπόλοιπους επεξεργαστές. Η σχέση του χρόνου εκτέλεσης με το πλήθος των επεξεργαστών είναι λογαριθμική, παρ' όλα αυτά η γραφική δεν είναι λογαριθμική και ο λόγος είναι ότι χρησιμοποιήθηκαν εικονικοί επεξεργαστές, και όχι πραγματικοί επεξεργαστές.

Γραφική 4.4



Η πιο πάνω γραφική παρουσιάζει τον χρόνο που χρειάζονται οι τρεις διαφορετικοί αλγόριθμοι για μετάδοση δεδομένων σε σχέση με το πλήθος των επεξεργαστών που χρησιμοποιούνται κάθε φορά για την εκτέλεση των αλγόριθμων. Παρατηρούμε ότι και οι τρεις αλγόριθμοι για μικρές τιμές των επεξεργαστών, έχουν σχεδόν ίσες τιμές για τον χρόνο εκτέλεσης, όμως στην συνέχεια ο δεύτερος αλγόριθμος για μετάδοση χρειάζεται περισσότερο χρόνο εκτέλεσης από τους υπόλοιπους δύο αλγόριθμους, ενώ ο πρώτος αλγόριθμος χρειάζεται τον λιγότερο χρόνο εκτέλεσης. Αυτό είναι λογικό, διότι στον πρώτο αλγόριθμο στέλλονται όλα τα μηνύματα στο πρώτο υπερβήμα, ενώ στους άλλους δύο στέλλονται σταδιακά σε περισσότερα υπερβήματα. Οπόταν χρειάζεται περισσότερος χρόνος, ο οποίος οφείλεται στον συγχρονισμό των επεξεργαστών που χρειάζεται στο κάθε υπερβήμα. Στον πρώτο αλγόριθμο μόνο μια φορά χρειάζεται να συγχρονιστούν οι επεξεργαστές. Αν παρατηρήσουμε τους χρόνους που χρειάζονται ο δεύτερος και τρίτος αλγόριθμος, θα δούμε ότι έχουν σχεδόν ίσους χρόνους εκτέλεσης και ο λόγος είναι διότι το k στον τρίτο αλγόριθμο που παρουσιάσαμε είναι ίσο με 2. Οπόταν τώρα ο τρίτος αλγόριθμος κάνει ακριβώς το ίδιο πράγμα που κάνει και ο δεύτερος. Ο χρόνος όμως δεν είναι ακριβώς ο ίδιος, διότι χρησιμοποιούνται εικονικοί επεξεργαστές αντί πραγματικών επεξεργαστών.

Γραφική 4.5

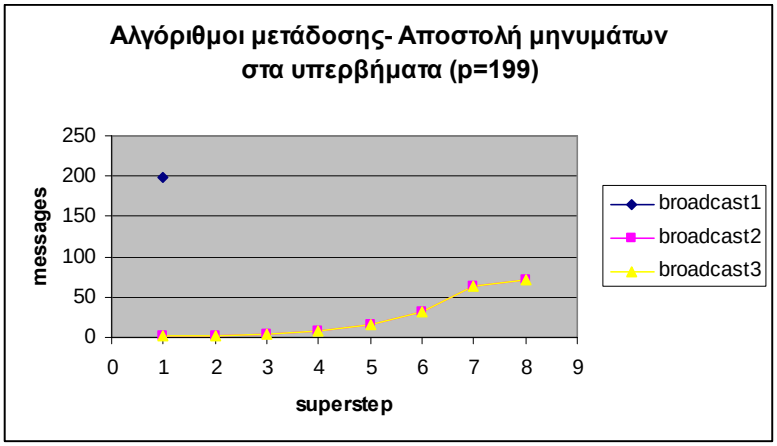


Η πιο πάνω γραφική δείχνει το πώς μεταβάλλεται ο χρόνος εκτέλεσης του τρίτου αλγόριθμου για μετάδοση δεδομένων σε σχέση με το k. Δηλαδή με διαφορετικά k ελέγχουμε πώς επηρεάζεται ο χρόνος εκτέλεσης τους αλγόριθμου. Για τις μετρήσεις που λήφθηκαν για την δημιουργία αυτής της γραφικής, το πλήθος των επεξεργαστών που χρησιμοποιήθηκαν ήταν 199. Η γραφική είναι μια καμπύλη που έχει κλίση προς τα κάτω. Παρατηρούμε ότι οι τιμές για το χρόνο έχουν τάση να μειώνονται, καθώς αυξάνεται το k που χρησιμοποιείται από τον αλγόριθμο. Βέβαια από μια τιμή του k και μετά, ο χρόνος εκτέλεσης του αλγόριθμου για διαφορετικά k είναι περίπου ο ίδιος. Αυτό συμβαίνει διότι για εκείνο το διάστημα τιμών του k, ο αλγόριθμος χρειάζεται δύο υπερβήματα για να ολοκληρωθεί, οπόταν γι αυτό και ο χρόνος είναι παρόμοιος. Αντίθετα, στις προηγούμενες μικρότερες τιμές για το k χρειάζονταν περισσότερα υπερβήματα για την ολοκλήρωση του αλγόριθμου και επομένως περισσότερος χρόνος. Τέλος, όταν το k ήταν ίσο με το πλήθος των επεξεργαστών, ο χρόνος εκτέλεσης ήταν μικρότερος, αφού χρειάζεται ένα υπερβήμα ολοκλήρωσης του αλγόριθμου.

Έτσι μπορούμε να συμπεράνουμε ότι καθώς αυξάνεται το k στον αλγόριθμο, δηλαδή όσο στέλνονται περισσότερα μηνύματα σε κάθε βήμα από τους επεξεργαστές που γνωρίζουν το δεδομένο, τόσο μειώνεται ο χρόνος εκτέλεσης του αλγόριθμου. Αυτό είναι λογικό διότι θα χρειαστούν πιο λίγα υπερβήματα για να τερματίσει ο αλγόριθμος, οπόταν και πιο λίγος χρόνος για συγχρονισμό μεταξύ των επεξεργαστών.

Ακόμη παρατηρούμε ότι όταν είχαμε ίσο πλήθος επεξεργαστών με το k (199), ο χρόνος ήταν ο πιο μικρός, καθώς επίσης ήταν περίπου ο ίδιος με τον χρόνο εκτέλεσης του πρώτου αλγόριθμου, όταν χρησιμοποιούσε 199 επεξεργαστές. Αυτό συμβαίνει διότι στέλλονται όλα τα μηνύματα από το πρώτο υπερβήμα σε όλους τους επεξεργαστές, οπόταν πλέον ο τρίτος αλγόριθμος κάνει ακριβώς την ίδια δουλεία με τον πρώτο αλγόριθμο. Ο χρόνος όμως παρατηρούμε ότι είναι σχεδόν ο ίδιος, και όχι ο ίδιος όπως θα αναμέναμε. Ο λόγος είναι η χρήση των εικονικών επεξεργαστών για την εκτέλεση των αλγόριθμων, που σίγουρα δεν μας δίνει τόσο ρεαλιστικά αποτελέσματα, όπως στην περίπτωση των πραγματικών επεξεργαστών. Οι εικονικοί επεξεργαστές δημιουργούνται πάνω σε ένα πραγματικό επεξεργαστή και επομένως είναι φυσικό ότι θα υπάρχει κάποια επιπλέον καθυστέρηση, λόγω υπερβολικής δουλειάς που αναγκάζεται να κάνει ο ένας επεξεργαστής.

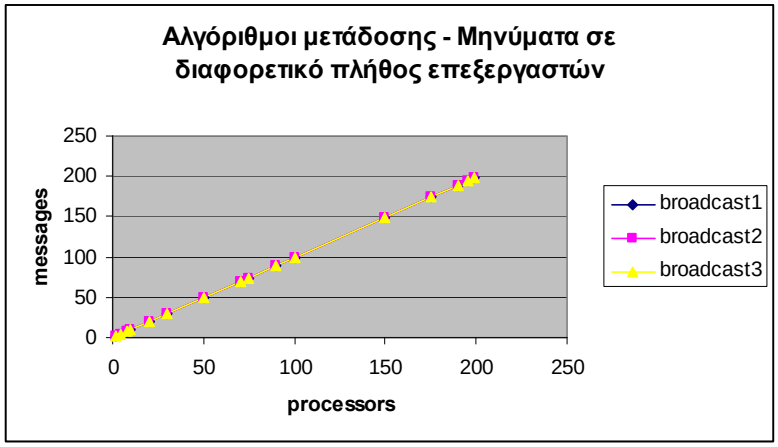
Γραφική 4.6



Η γραφική παρουσιάζει το πώς μεταβάλλεται το πλήθος των σταλμένων μηνυμάτων στα διάφορα υπερβήματα του αλγόριθμου, όταν οι επεξεργαστές είναι ίσοι με 199. Παρατηρούμε ότι για τον πρώτο αλγόριθμο μετάδοσης, χρειάστηκε μόνο ένα υπερβήμα, στο οποίο στάλθηκαν όλα τα μηνύματα, γι αυτό και στάλθηκαν πάρα πολλά μηνύματα. Σε αντίθεση στους άλλους δύο χρειάστηκε να γίνουν 8 υπερβήματα, στα οποία καθώς αυξάνεται ο δείχτης τους, αυξάνεται το πλήθος των μηνυμάτων που στέλνονται σε αυτά.

Αυτό είναι φυσικό, διότι περισσότεροι επεξεργαστές γνωρίζουν το δεδομένο, καθώς προχωρούν τα υπερβήματα, από τους οποίους είναι δυνατόν όλοι να χρειαστεί να στείλουν k-1 μηνύματα σε κάποιους άλλους, αφού στον αλγόριθμο αυτό έχουμε μια k-1 σχέση (h=k-1).

Γραφική 4.7



Στην γραφική αυτή παρουσιάζεται το πώς μεταβάλλονται τα μηνύματα που στέλνονται στους τρεις αλγόριθμους, σε σχέση με το πλήθος των επεξεργαστών που χρησιμοποιούνται. Παρατηρούμε ότι η καμπύλη κατευθύνεται προς τα πάνω, δηλαδή καθώς αυξάνεται το πλήθος των επεξεργαστών αυξάνεται και το πλήθος των μηνυμάτων που στέλνονται. Συγκεκριμένα σαν σύνολο, το πλήθος των μηνυμάτων που στέλνονται κάθε φορά είναι και όσο το πλήθος των επεξεργαστών μειωμένο κατά ένα. Αυτό είναι αναμενόμενο διότι ο πρώτος επεξεργαστής γνωρίζει το δεδομένο, οπόταν απομένουν άλλοι *πλήθος_επεξεργαστών -1* επεξεργαστές να το μάθουν. Όλοι αυτοί οι επεξεργαστές για να μάθουν το δεδομένο πρέπει να πάρουν κάποιο μήνυμα που να τους το μεταδίδει.

Επιπρόσθετα βλέπουμε ότι και στους τρεις αλγόριθμους στέλνεται ίσο πλήθος μηνυμάτων, για ίσο πλήθος επεξεργαστών. Αυτό είναι φυσικό διότι αφού χρησιμοποιείται το ίδιο πλήθος επεξεργαστών σημαίνει πως θα σταλούν τόσα μηνύματα όσοι και οι επεξεργαστές. Γνωρίζουμε ότι κάθε επεξεργαστής για να ενημερωθεί για το δεδομένο που μεταδίδεται πρέπει να πάρει κάποιο μήνυμα.

4.6 Συμπεράσματα

Στο σημείο αυτό θα παρουσιαστούν τα συμπεράσματα που προέκυψαν από της μετρήσεις και τις γραφικές παραστάσεις των μετρήσεων. Τα συμπεράσματα αφορούν και τους τρεις αλγόριθμους για μετάδοση δεδομένων.

1. Παρατηρούμε ότι μεγαλύτερο πλήθος επεξεργαστών και στους τρεις αλγόριθμους οδηγεί σε μεγαλύτερο χρόνο εκτέλεσης. Αυτό συμβαίνει διότι πρέπει να υπάρξει μεγαλύτερη επικοινωνία για να ενημερωθούν όλοι οι επεξεργαστές. Όπου σίγουρα αυτό χρειάζεται μεγαλύτερο χρόνο.
2. Σαν σύνολο το πλήθος των μηνυμάτων που στέλνονται είναι το ίδιο σε όλους τους αλγόριθμους και είναι ίσο με το *πλήθος των επεξεργαστών -1*, αφού ο πρώτος επεξεργαστής, έστω pid=0, γνωρίζει από την αρχή το μήνυμα που πρέπει να σταλεί. Συνεπώς απομένουν ακόμη *πλήθος των επεξεργαστών -1* επεξεργαστές να μάθουν το δεδομένο. Κάθε επεξεργαστής για να μάθει το δεδομένο πρέπει να παραλάβει ένα μήνυμα, από ένα άλλο επεξεργαστή που να το δίνει το δεδομένο μετάδοσης. Επομένως συνολικά θα σταλούν τόσα μηνύματα όσο και το πλήθος των επεξεργαστών που δεν γνωρίζουν το μήνυμα.
3. Σε κάθε υπερβήμα στέλνονται πολύ πιο λίγα μηνύματα στον δεύτερο και τρίτο αλγόριθμο από ότι στον πρώτο. Στον πρώτο στέλνονται όλα τα μηνύματα από το πρώτο υπερβήμα αφού έχουμε μια nprocs-1 σχέση στον αλγόριθμο αυτό (h=nprocs-1). Στους άλλους δύο αλγόριθμους έχουμε μια 1-σχέση, οπόταν τα μηνύματα στέλλονται σταδιακά στα διάφορα υπερβήματα, και έτσι σε κάθε υπερβήμα στέλλονται πιο λίγα μηνύματα.
4. Στον δεύτερο και τρίτο τα μηνύματα στέλνονται ακριβώς με τον ίδιο τρόπο, αφού στις μετρήσεις αυτές το k για τον τρίτο αλγόριθμο είναι ίσο με 2 και άρα ο τρίτος αλγόριθμος μετάδοσης γίνεται ο δεύτερος.
5. Παρατηρούμε ακόμη από τις μετρήσεις, ότι για διαφορετικά k στον τρίτο αλγόριθμο μετάδοσης, όσο μεγαλώνει το k ο χρόνος εκτέλεσης μειώνεται. Αυτό είναι λογικό διότι στέλνονται περισσότερα μηνύματα σε ένα υπερβήμα και

επομένως γίνεται η μετάδοση μέσα σε πιο λίγα υπερβήματα και επομένως πιο λίγο χρόνο.

6. Ακόμη παρατηρούμε από την ίδια μέτρηση ότι όταν είχαμε ίσο πλήθος επεξεργαστών με το k (199) τότε ο χρόνος ήταν ο πιο μικρός καθώς επίσης ήταν περίπου ο ίδιος με τον χρόνο εκτέλεσης του πρώτου αλγόριθμου, όταν χρησιμοποιούσε 199 επεξεργαστές. Αυτό συμβαίνει διότι πλέον ο τρίτος αλγόριθμος γίνεται ο ίδιος με τον πρώτο αλγόριθμο μετάδοσης, αφού στο πρώτο υπερβήμα ο επεξεργαστής με ταυτότητα 0 (pid=0) στέλνει το μήνυμα σε k-1 επεξεργαστές, και επομένως σε 198 επεξεργαστές. Οπόταν από το πρώτο υπερβήμα στέλνεται το μήνυμα σε όλους τους επεξεργαστές και τελειώνει η εκτέλεση του αλγόριθμου. Ο χρόνος είναι περίπου ο ίδιος και όχι ο ίδιος, λόγω του ότι η εκτέλεση αλγόριθμων γίνεται σε εικονικούς και όχι σε πραγματικούς επεξεργαστές.
7. Οι χρόνοι που πήραμε από τις παραπάνω μετρήσεις αναμέναμε σύμφωνα και με την θεωρητική ανάλυση να είναι οι ίδιοι για το δεύτερο και τρίτο αλγόριθμο, λόγω του ότι στον τρίτο αλγόριθμο το k είναι ίσο με 2, οπόταν είναι σαν να έχουμε τον δεύτερο αλγόριθμο. Παρατηρούμε ότι οι χρόνοι εκτελέσεις των δύο αυτών αλγόριθμων είναι περίπου οι ίδιοι και όχι οι ίδιοι. Αυτό οφείλεται στο γεγονός ότι χρησιμοποιούμε εικονικούς επεξεργαστές για την εκτέλεση των αλγόριθμων και την λήψη των μετρήσεων. Στην πραγματικότητα η θεωρία βασίζεται στο γεγονός ότι χρησιμοποιούμε πραγματικούς επεξεργαστές. Με την χρήση εικονικών δημιουργούμε ουσιαστικά φανταστικούς επεξεργαστές σε ένα πραγματικό, οπόταν είναι φυσικό ότι με αυτή την υπερφόρτωση ενός επεξεργαστή, δεν μπορούμε να πάρουμε και πολύ ρεαλιστικά αποτελέσματα, λόγω καθυστερήσεων που δημιουργούνται σε αυτό τον επεξεργαστή.

4.7 Μετάδοση μηνυμάτων μεγαλύτερου μεγέθους (πινάκων)

Πιο κάτω θα παρουσιαστούν μετρήσεις, γραφικές και συμπεράσματα για τους πιο πάνω τρεις αλγόριθμους μετάδοσης, με τη διαφορά όμως ότι το στοιχείο που μεταδίδεται μεταξύ των επεξεργαστών είναι ένας πίνακας. Δηλαδή μεταδίδονται μηνύματα μεγαλύτερου μεγέθους.
Επομένως αυτό που θα ελέγξουμε είναι το κατά πόσο επηρεάζει τον χρόνο εκτέλεσης των αλγόριθμων, το μέγεθος των μηνυμάτων που στέλνονται.

4.7.1 Μετρήσεις

Πιο κάτω παρουσιάζονται οι μετρήσεις που έγιναν για την μετάδοση μηνυμάτων μεγαλύτερου μεγέθους, χρησιμοποιώντας τους αλγόριθμους broadcast1, broadcast2 broadcast3.

Πίνακας 4.5

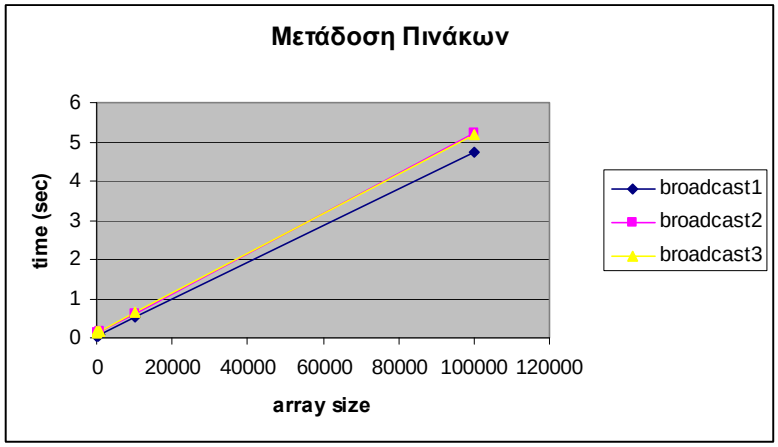
Μέγεθος πινάκων	1	200	400	1000	10000	100000
Χρόνος για broadcast1	0.04681	0.05124	0.06118	0.09079	0.52984	4.74278
Χρόνος για broadcast2	0.12953	0.13178	0.14084	0.17209	0.6019	5.24227
Χρόνος για broadcast3	0.14173	0.14365	0.15284	0.18107	0.63685	5.17249

Το πλήθος των επεξεργαστών είναι σταθερό σε αυτές τις μετρήσεις και είναι ίσο με 199.

4.7.2 Ανάλυση αποτελεσμάτων

Πιο κάτω παρουσιάζονται μια γραφική παράσταση για τις μετρήσεις που παρουσιάστηκαν προηγουμένως. Συγκεκριμένα δείχνει την σχέση μεταξύ του μεγέθους των μηνυμάτων και του χρόνου εκτέλεσεις των αλγόριθμων.

Γραφική 4.8



Η πιο πάνω γραφική παρουσιάζει την σχέση μεταξύ του μεγέθους των μηνυμάτων που στέλλονται και του χρόνου εκτέλεσης, καθενώς από τους τρεις αλγόριθμους μετάδοσης δεδομένων.

Παρατηρούμε ότι η γραφική για τους τρεις αλγόριθμους είναι μια καμπύλη που κατευθύνεται προς τα πάνω. Δηλαδή όσο μεγαλώνει το μέγεθος των μηνυμάτων αυξάνεται και ο χρόνος εκτέλεσης των αλγόριθμων.

Αυτό συμβαίνει διότι με την αποστολή μεγαλύτερου μεγέθους μηνυμάτων, παραδείγματος χάριν μεγέθους h, έχει το ίδιο κόστος με την αποστολή h μηνυμάτων μεγέθους 1[7].

4.7.3 Συμπεράσματα

1. Το μοντέλο BSP δεν κάνει διάκριση μεταξύ του κόστους για μια διαδικασία που αποστέλλει h μηνύματα με μέγεθος 1 και μια διαδικασία που αποστέλλει ένα μήνυμα μεγέθους h. Και οι δύο επικοινωνίες είναι μια h-σχέση με κόστος h*g [**7**]. Από εδώ μπορούμε να καταλάβουμε ότι όταν στέλουμε ίδιο πλήθος μηνυμάτων σε δύο διαδικασίες, αλλά στην μια από τις δύο στέλουμε μεγαλύτερου μεγέθους μηνύματα είναι φυσικό ότι θα χρειάζεται περισσότερος χρόνος για την εκτέλεση της διαδικασίας. Μάλιστα όσο μεγαλύτερη είναι η διαφορά του μεγέθους των μηνυμάτων, τόσο περισσότερο θα μεγαλώνει η διαφορά στον χρόνο εκτέλεσης των δύο διαδικασιών. Με λίγα λόγια όσο μεγαλύτερο είναι το μέγεθος των μηνυμάτων που ανταλλάσονται μεταξύ των επεξεργαστών, τόσο μεγαλύτερος είναι και ο χρόνος εκτέλεσης του αλγόριθμου.

ΚΕΦΑΛΑΙΟ 5

Παράλληλος προθεματικός υπολογισμός

Στο Κεφάλαιο 5 παρουσιάζεται το πρόβλημα του παράλληλου προθεματικού υπολογισμού ξεκινώντας με το Υποκεφάλαιο 5.1 στο οποίο γίνεται η περιγραφή του προβλήματος. Ακολουθεί το υποκεφάλαιο 5.2 που δίνει την σειριακή λύση του προβλήματος και το Υποκεφάλαιο 5.3 που δίνει την παράλληλη λύση. Στην συνέχεια στο Υποκεφάλαιο 5.4 περιγράφονται οι αλγόριθμοι που υλοποιήθηκαν για το πρόβλημα αυτό και στο 5.5 γίνονται κάποιες παρατηρήσεις για την αναμενόμενη συμπεριφορά του αλγόριθμου. Επιπλέον το Κεφάλαιο 5 περιέχει τα Υποκεφάλαια 5.6 και 5.7 για την παρουσίαση των μετρήσεων και των γραφικών παραστάσεων και τέλος περιέχει το Υποκεφάλαιο 5.8 στο οποίο γράφονται τα συμπεράσματα.

5.1 Πρόβλημα Παράλληλου Προθεματικού Υπολογισμού

Έστω X = (x1,x2,…,x*n*) ένα διατεταγμένο σύνολο στοιχείων ενός συνόλου Ψ και ⊗ ένας *δυαδικός (*x*i* ⊗ x*j*)*, προσεταιριστικός (* (x*i* ⊗ x*j*) ⊗ x*k* = x*i* ⊗ (x*j* ⊗ x*k*)*), κλειστός* (x*i* ⊗ x*j* ∈ Ψ) *τελεστής* στο Ψ [**2**].

Ο Τελεστής ⊗ απαιτεί χρόνο Θ(1).

Μερικά παραδείγματα για τον τελεστή είναι τα πιο κάτω:

• Αριθμοί: +, · , MIN, MAX → Ψ: σύνολο ακεραίων, πραγματικών

• Λογικές πράξεις: AND, OR, XOR → Ψ: {0,1}

• Χαρακτήρες/Λέξεις: συνένωση (concatenation) → Ψ: {a,b}*

K^{οστό} πρόθεμα (prefix) καλείται το αποτέλεσμα x1 ⊗ x2 ⊗ … ⊗ xk.

Το **πρόβλημα** που πρέπει να λύσουμε λοιπόν σε αυτό το σημείο είναι να υπολογίσουμε όλα τα **n** προθέματα για τον τελεστή (+), όπου η διαδικασία αυτή ονομάζεται **ολικός προθεματικός υπολογισμός** (all-prefix computation).

Παράδειγμα της λύσης του προβλήματος:

1ο πρόθεμα π1 = x1

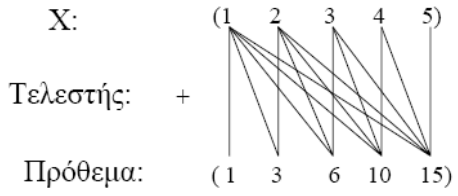
2ο πρόθεμα π2 = x1 + x2

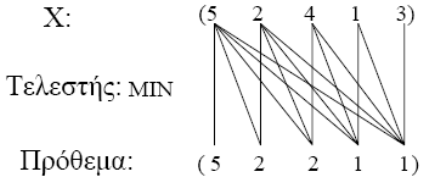
3ο πρ…όθεμα …π3 = x1 + x2 + x3

nοστό πρόθεμα π**n** = x1 + x2 + … + x**n**

Ο προθεματικός υπολογισμός με χρήση παράλληλων συστημάτων καλείται **παράλληλος προθεματικός υπολογισμός**. Η λύση του πιο πάνω προβλήματος θα γίνει χρησιμοποιώντας το μοντέλο BSP, όπου βεβαίως και θα θεωρείται παράλληλος προθεματικός υπολογισμός.

<u>Σχήμα 5.1</u>: (Παράδειγμα προθεματικού υπολογισμού για τελεστή (+) ή (MIN)) [**2,4**]





Άλλοι τελεστές που μπορούν να χρησιμοποιηθούν είναι:

- Για αριθμούς: +,-,MIN,MAX ,όπου το Ψ μπορεί να είναι το σύνολο ακεραίων ή πραγματικών αριθμών.
- Λογικές πράξεις: AND,OR,XOR, όπου το σύνολο Ψ είναι το {0,1}.

5.2 Σειριακή λύση

Δεδομένου του συνόλου X = {x1,x2,…,x**n**}, πρέπει να υπολογιστούν τα n προθέματα (π1,π2,..,π**n**) που μπορούν να παραχθούν από αυτό το σύνολο X, με τον τελεστή +.

Σύντομη περιγραφή: Θα χρησιμοποιηθεί στη λύση αυτή ένας επεξεργαστής όπου με ένα βρόγχο θα αθροίζει με σειρά όλα τα στοιχεία του συνόλου και κάθε φορά που αθροίζει δύο θα τυπώνει το αποτέλεσμα και πάνω σε αυτό θα αθροίζει το επόμενο στοιχείο του συνόλου. Οι πράξεις ξεκινούν να γίνονται από την αρχή του συνόλου, δηλαδή από το στοιχείο x1.

Για την σειριακή λύση χρειάζεται γραμμικός χρόνος ως προς το πλήθος των στοιχείων που πρέπει να αθροιστούν → T(n)=Θ(n), όπου n πλήθος στοιχείων μέσα στο σύνολο X.

5.3 Παράλληλη λύση

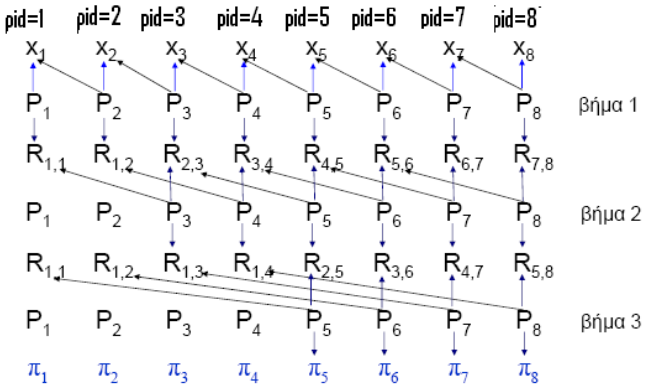
Στην παράλληλη λύση χρησιμοποιείται η τεχνική του **αναδρομικού διπλασιασμού.** Δηλαδή γίνεται ένας συνδυασμός ζευγαριών στοιχείων και στην συνέχεια ζευγαριών ζευγαριών και ούτω κάθε εξής.
Επίσης χρησιμοποιούνται n επεξεργαστές, όσο είναι δηλαδή και το πλήθος των στοιχείων στο σύνολο X. Στο τέλος του αλγόριθμου το π_κ , δηλαδή το κάθε πρόθεμα, βρίσκεται στην τοπική μνήμη του αντίστοιχου επεξεργαστή, pid=k.
Για το **αλγόριθμο παράλληλου προθεματικού αθροίσματος στο μοντέλο BSP** σε κάθε υπερβήμα παίρνουν κάποιοι επεξεργαστές το δεδομένο που έχουν στην τοπική τους μνήμη κάποιοι άλλοι επεξεργαστές. Αρχικά όλοι οι επεξεργαστές παίρνουν δεδομένο από τον επεξεργαστή που έχει την αμέσως μικρότερη ταυτότητα από την δική τους. Ακολούθως σε κάθε υπερβήμα διπλασιάζεται η διαφορά μεταξύ ταυτοτήτων των επεξεργαστών που συσχετίζονται. Δηλαδή ένας επεξεργαστής θα πάρει δεδομένο από έναν άλλο, που η ταυτότητα τους έχει διπλάσια διαφορά από την διαφορά που είχε η ταυτότητα του συγκεκριμένου επεξεργαστή που θα πάρει δεδομένο, από την ταυτότητα του προηγούμενου επεξεργαστή από τον οποίο πηρέ στοιχείο στο προηγούμενο υπερβήμα. Καθώς εκτελείται ο αλγόριθμος παρατηρούμε ότι στα διάφορα του υπερβήματα μειώνεται ο αριθμός των επεξεργαστών που παίρνουν δεδομένο από κάποιο άλλο επεξεργαστή. Συγκεκριμένα κάθε φορά το πλήθος των επεξεργαστών που σταματούν να παίρνουν δεδομένο από κάποιους άλλους, γιατί έχουν είδη υπολογίσει το προθεματικό τους άθροισμα, διπλασιάζεται.

Πιο κάτω φαίνεται ο κώδικας για τον αλγόριθμο προθεματικού αθροίσματος στο μοντέλο BSP.

```
while(distance<nprocs){
   if(distance<=pid){
     bsp_get(bsp,pid-distance,&result,0,&extra,sizeof(int));
```

```
    cmsg[pid]++;
  }
bsp_sync(bsp);
result+=extra;
extra=0;
distance=2*distance;
```

Σχήμα 5.2: (έστω επεξεργαστές=8 και |X|=n=8 στοιχεία, παρουσιάζει την εκτέλεση της παράλληλης λύσης που παρουσιάστηκε πιο πάνω) [**2,4**]



Στο πιο πάνω σχήμα τα βέλη δείχνουν από πιο επεξεργαστή θα πάρει κάποιο δεδομένο κάποιος άλλος επεξεργαστής. Όλα τα στοιχεία στην στήλη κάθε επεξεργαστή είναι στοιχεία που βρίσκονται στην τοπική του μνήμη.

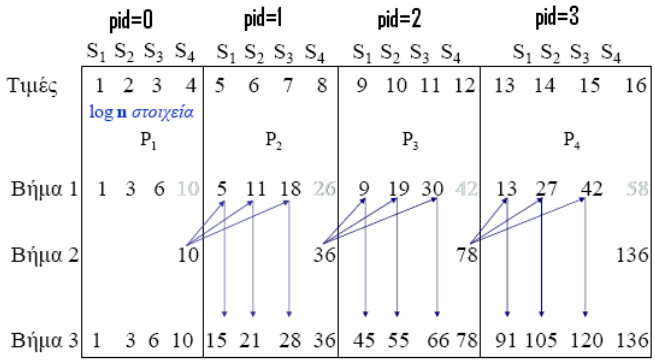
Υπάρχει όμως και βέλτιστος αλγόριθμος ο οποίος διατηρεί ασυμπτωτικά τον χρόνο σταθερό και βελτιώνει το κόστος του αλγόριθμου. Στη βέλτιστη αυτή λύση θα χρησιμοποιηθούν πιο λίγοι επεξεργαστές, συγκεκριμένα n/logn όταν το πλήθος των στοιχείων είναι ίσο με n.

Παράλληλη λύση για βέλτιστο αλγόριθμο

Πιο κάτω θα δοθεί η περιγραφή για τον βελτιστο αλγόριθμο προθεματικού αρθοίσματος, στην οποία μειώνουμε το πλήθος των επεξεργαστών, χρησιμοποιώντας n/logn επεξεργαστές αντί n, αλλά διατηρούμε τον ασυμπτωτικό χρόνο της προηγούμενης λύσης. Έτσι ο αλγόριθμος που ακολούθεί έχει πιο λίγο κόστος εκτέλεσης.

Κατά την εκτέλεση αυτού του αλγόριθμου, αρχικά οι επεξεργαστές μοιράζουν μεταξύ τους τα στοιχεία, συγκεκριμένα ο κάθε ένας παίρνει logn στοιχεία στην τοπική του μνήμη, για τα οποία βρίσκει το προθεματικό άθροισμα σειριακά, χρησιμοποιώντας την σειριακή λύση που παρουσιάσαμε πιο πάνω. Στην συνέχεια χρησιμοποιώντας ο κάθε επεξεργαστής το τελευταίο στοιχείο που έχει αποθηκευμένο στην τοπική του μνήμη, εφαρμόζεται η μη βέλτιστη παράλληλη λύση για τον υπολογισμό του προθεματικού αθροίσματος, που παρουσιάσαμε επίσης πιο πάνω, αφού υπάρχουν πλέον ίσο πλήθος επεξεργαστών και στοιχείων (logn επεξεργαστές και logn στοιχεία). Τέλος ο κάθε επεξεργαστής, εκτός από τον πρώτο (pid=0), αθροίζουν σειριακά τα στοιχεία που κρατούν στην τοπική τους μνήμη, εκτός όμως το τελευταίο, με το τελευταίο στοιχείο που κρατά ο προηγούμενος επεξεργαστής.

Σχήμα 5.3 (έστω επεξεργαστές=n/logn =4, |X|=n=16 στοιχεία, παράδειγμα εκτέλεσης βέλτιστου αλγόριθμου προθεματικής άθροισης) [**2,4**]



5.4 Περιγραφή αλγόριθμων

Παράλληλος προθεματικός υπολογισμός: (βέλτιστος αλγόριθμος χρησιμοποιώντας το μοντέλο BSP)

Βήμα 1: Ο κάθε επεξεργαστής από τους n/logn υπολογίζει σειριακά το ολικό προθεματικό άθροισμα των στοιχείων logn που κρατά φυλαγμένα στην τοπική του μνήμη.

Βήμα 2: Όλοι οι επεξεργαστές εκτελούν το μη βέλτιστο αλγόριθμο παράλληλου προθεματικού υπολογισμού, χρησιμοποιώντας τα τελευταία στοιχεία που φυλάσσει ο καθένας στην τοπική του μνήμη από το προηγούμενο βήμα.

Βήμα 3: Όλοι οι επεξεργαστές εκτός από τον πρώτο (pid=0), εκτελούν σειριακά άθροιση του τελευταίου στοιχείου των στοιχείων που αποθηκεύει ο προηγούμενος επεξεργαστής με τα στοιχεία της ομάδας τους εκτός του τελευταίου.

Χρόνος βέλτιστου αλγόριθμου για παράλληλο προθεματικό άθροισμα

Βήμα 1:

Υπολογιστικό μέρος: **logn** (Γίνονται n/logn σειριακές αθροίσεις από κάθε επεξεργαστή).

Επικοινωνιακό μέρος: **0**

Συγχρονισμός: **0**

Βήμα 2:

Υπολογιστικό μέρος: **1**

Επικοινωνιακό μέρος**:g**

Συγχρονισμός**:L**

Το βήμα 2 είναι ένας βρόχος που επαναλαμβάνεται logp φορές, όπου p είναι το πλήθος των επεξεργαστών.

Βήμα 3:

Υπολογιστικό μέρος: : **logn**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός: **L**

Συνολικός χρόνος υπολογισμού: 2logn+ (g+1+L)logp+ g+L (όπου p=n/logn)

5.5 Παρατηρήσεις

1. Όσο αυξάνονται τα στοιχεία που πρέπει να αθροιστούν, θα πρέπει να μεγαλώνει και ο χρόνος που χρειάζεται ο αλγόριθμος.
2. Επίσης όσο περισσότεροι είναι οι επεξεργαστές τόσο μεγαλύτερος θα είναι ο χρόνος και η επικοινωνία. Οπόταν θα είναι και περισσότερα τα μηνύματα που θα στέλνονται μεταξύ των επεξεργαστών.

5.6 Μετρήσεις

Πιο κάτω παρουσιάζονται οι μετρήσεις που πάρθηκαν για τις διάφορες παραμέτρους του αλγόριθμου αυτού. Για μετρήσεις του χρόνου παίρναμε κάθε φορά 5 διαφορετικές για το ίδιο στιγμιότυπο του αλγόριθμου και παίρναμε τον μέσο όρο αυτών ως τελικό χρόνο εκτέλεσης του αλγόριθμου.

Πίνακας 5.1:

Πλήθος στοιχείων	Πλήθος επεξεργαστών	Χρόνος	Σταλμένα μηνύματα
16	4	0.00241	8
256	32	0.04457	160

Στο πιο πάνω πίνακα ισχύει η σχέση ότι n είναι το πλήθος των στοιχείων και n/ logn είναι το πλήθος των επεξεργαστών (βέλτιστος αλγόριθμος προθεματικής άθροισης).

Πίνακας 5.2:

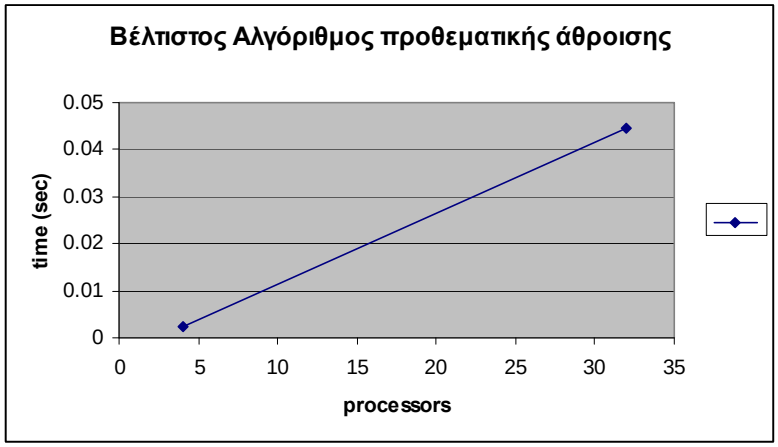
Πλήθος στοιχείων	**Πλήθος επεξεργαστών**	**Χρόνος**	**Σταλμένα μηνύματα**
16	**16**	**0.01134**	**64**

Παρατήρηση: Στον αλγόριθμο για την προθεματική άθροιση παρατηρούμε ότι αν χρησιμοποιήσουμε ίσο πλήθος επεξεργαστών με τα στοιχεία μας τότε θα μετατραπεί στον μη βέλτιστο αλγόριθμο προθεματικού υπολογισμού. Θεωρητικά γνωρίζουμε ότι ο βέλτιστος αλγόριθμος έχει λιγότερο κόστος, και ασυμπτωτικά τον ίδιο χρόνο με τον μη βέλτιστο. Πιστεύουμε όμως ότι χρειάζεται περισσότερο απόλυτο χρόνο εκτέλεσης αφού χρησιμοποιούνται πιο λίγοι επεξεργαστές, και χρειάζεται να γίνει και κάποιο σειριακό κομμάτι για τον υπολογισμό του προθεματικού αθροίσματος. Πρακτικά όμως βλέπουμε πως δεν ισχύει αυτό, σύμφωνα με την μέτρηση που πήραμε για 16 επεξεργαστές και 16 στοιχεία. Αυτό συμβαίνει διότι είναι πολύ μικρό το πλήθος των στοιχείων και των επεξεργαστών οπόταν το σειριακό μέρος δεν έχει πολλές επαναλήψεις και άρα δεν χρειάζεται πολύ χρόνο. Επιπρόσθετα το παράλληλο μέρος χρειάζεται περισσότερο χρόνο στον μη βέλτιστο αλγόριθμο, αφού χρησιμοποιούνται περισσότεροι επεξεργαστές και άρα θα γίνουν περισσότερες επαναλήψεις. Σε συνδυασμό των δύο πιο πάνω, τελικά καταλήγουμε να έχουμε πιο μεγάλο χρόνο εκτέλεσης στον μη βέλτιστο αλγόριθμο. Σε μεγάλες όμως τιμές των στοιχείων και των επεξεργαστών αναμένουμε να ισχύει η πιο πάνω παρατήρηση, δηλαδή η απόλυτη τιμή του χρόνου εκτέλεσης του μη βέλτιστου αλγόριθμου να είναι πιο μικρή από τον χρόνο εκτέλεσης του βέλτιστου αλγόριθμου για παράλληλο προθεματικό υπολογισμό.

5.7 Ανάλυση αποτελεσμάτων

Πιο κάτω παρουσιάζονται οι γραφικές για τις μετρήσεις που παρουσιάστηκαν πιο πάνω.

Γραφική 5.1

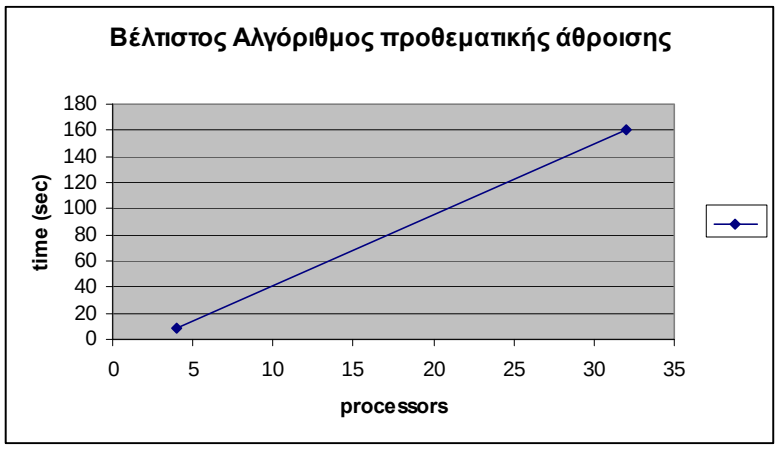


Στην πιο πάνω γραφική φαίνεται το πως επηρεάζεται ο χρόνος εκτέλεσης του αλγόριθμου από το πλήθος των επεξεργαστών που χρησιμοποιούνται και κατά συνέπεια και από το πλήθος των στοιχείων που χρησιμοποιεί ο αλγόριθμος, σε κάθε εκτέλεση του.

Παρατηρούμε ότι η γραφική είναι μια ευθεία στην οποία οι τιμές του y μεγαλώνουν όσο μεγαλώνουν οι τιμές για το x. Το πλήθος των επεξεργαστών σχετίζεται λογαριθμικά με τον χρόνο εκτέλεσης του αλγόριθμου, οπόταν περιμένουμε και η γραφική να είναι λογαριθμική. Η γραφική όμως είναι ευθεία λόγω του ότι μόνο δύο μετρήσεις μπορούσαμε να πάρουμε, επειδή χρησιμοποιούμε εικονικούς επεξεργαστές και το πλήθος που μπορούμε να χρησιμοποιήσουμε είναι περιορισμένο.

Οπόταν από εδώ φαίνεται ότι όσο περισσότεροι είναι οι επεξεργαστές τόσο μεγαλύτερος είναι ο χρόνος εκτέλεσης του αλγόριθμου. Αυτό ισχύει διότι περισσότεροι επεξεργαστές συνεπάγεται περισσότερα στοιχεία (n= πλήθος στοιχείων και n/logn= πλήθος επεξεργαστών), οπόταν σίγουρα με περισσότερα στοιχεία θα χρειάζεται περισσότερος χρόνος για να γίνει η προθεματική άθροιση.

Γραφική 5.2



Στην πιο πάνω γραφική φαίνεται η σχέση μεταξύ των επεξεργαστών, και του πλήθους στοιχείων κατά συνέπεια, με το πλήθος μηνυμάτων που στέλνονται μεταξύ των επεξεργαστών.

Βλέπουμε ότι η γραφική είναι μια ευθεία γραμμή, στην οποία μεγαλώνουν οι τιμές των τεταγμένων όσο μεγαλώνουν οι τιμές των τετμημένων.

Συνεπώς όσο αυξάνεται το πλήθος των επεξεργαστών, άρα και των στοιχείων που θα χρησιμοποιηθούν για την προθεματική άθροιση, τόσο περισσότερα μηνύματα ανταλλάσσονται μεταξύ των επεξεργαστών. Αυτό είναι αναμενόμενο διότι όσο μεγαλύτερο είναι το πλήθος των επεξεργαστών χρειάζονται περισσότερα μηνύματα για της επικοινωνία τους.

5.8 Συμπεράσματα

1. Παρατηρούμε ότι όσο μεγαλώνει το πλήθος των επεξεργαστών και των στοιχείων που χρησιμοποιούμε, αυξάνεται και ο χρόνος που χρειαζόμαστε για την εκτέλεση του αλγόριθμου. Αυτό οφείλεται στο γεγονός ότι θα χρειαστεί περισσότερος χρόνος για συγχρονισμό μεταξύ τον επεξεργαστών αφού θα αυξηθεί και το πλήθος των υπερβημάτων που εκτελούνται στον αλγόριθμο.

2. Επίσης τα μηνύματα που στέλνονται αυξάνονται μαζί με την αύξηση του πλήθους των επεξεργαστών και των στοιχείων, και αυτό είναι φυσικό διότι τότε χρειάζεται να γίνει περισσότερη επικοινωνία και συνεπώς έχουμε μεγαλύτερη ανταλλαγή μηνυμάτων.

Σημείωση: Χρησιμότητα Προθεματικού υπολογισμού [2,4]

Ο Προθεματικός Υπολογισμός βρίσκει αρκετές **εφαρμογές**, για παράδειγμα:

- στην επίλυση αναδρομικών εξισώσεων,
- αποτίμηση πολυωνύμων,
- λεξιλογική σύγκριση χαρακτήρων,
- αναζήτηση κανονικών εκφράσεων,
- υλοποίηση της Radixsort και Quicksort και
- διαγραφή συγκεκριμένων στοιχείων μιας συστοιχίας.

Ο προθεματικός υπολογισμός περιέχεται ως αρχέγονη εντολή σε πολλά υπολογιστικά συστήματα.

ΚΕΦΑΛΑΙΟ 6

Παράλληλη άθροιση

Στόχος του Κεφαλαίου 6 είναι η παρουσίαση του προβλήματος της παράλληλης άθροισης. Με τα Υποκεφάλαια 6.1, 6.2 και 6.3 παρουσιάζεται το πρόβλημα με την σειριακή και παράλληλη του λύση. Ακολούθως υπάρχουν τα Υποκεφάλαια 6.4, 6.5, 6.6, 6.7 και 6.8 στα οποία γίνεται περιγραφή του αλγόριθμου βέλτιστης παράλληλης άθροισης n στοιχείων που χρησιμοποιεί n/logn επεξεργαστές, γίνονται κάποιες παρατηρήσεις για τον αλγόριθμο της παράλληλης άθροισης, και παρουσιάζονται τα αποτελέσματα μετρήσεων, οι γραφικές παραστάσεις και τα συμπεράσματα.

6.1 Πρόβλημα Παράλληλης Άρθοισης Στόχος μας στην Παράλληλη Άθροιση είναι το *άθροισμα n αριθμών*

Έτσι μας δίδεται μια λίστα n αριθμών, $\alpha_1,\alpha_2,.....,\alpha_n$ και μας ζητείται να υπολογίσουμε το άθροισμα S των αριθμών.

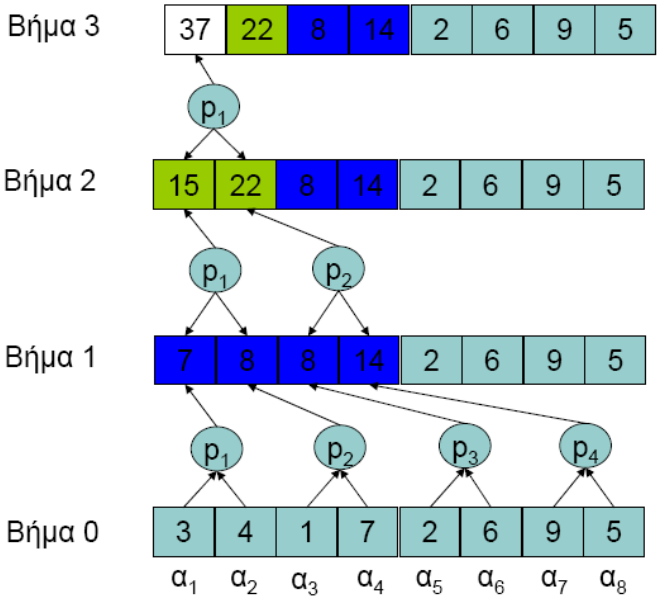
6.2 Σειριακή λύση Ο επεξεργαστής εκτελεί n-1 διαδοχικές προσθέσεις. [**2,4**]

Set $S=a_1$
For j=2 to n do
    $S=S+a_j$
end do

Η **πολυπλοκότητα** αυτής της σειριακής λύσης για παράλληλη άθροιση είναι $T(n)=C(n)=\Theta(n)$. Όπου T(n) είναι ο χρόνος εκτέλεσης και C(n) είναι το κόστος του αλγόριθμου.

6.3 Παράλληλη λύση Οι επεξεργαστές που χρησιμοποιούνται μπορούν να είναι λιγότεροι, ίσοι, ή περισσότεροι από το πλήθος των στοιχείων που θα αθροιστούν. Οι επεξεργαστές μοιράζουν τα στοιχεία που πρέπει να αθροιστούν (σε περίπτωση όπου το πλήθος των επεξεργαστών είναι λιγότερο από το πλήθος των στοιχείων) και σειριακά ο καθένας αθροίζει τα δικά του στοιχεία (πλήθος στοιχείων / πλήθος επεξεργαστών). Στην συνέχεια κάποιοι συγκεκριμένοι επεξεργαστές στέλνουν το άθροισμα που κρατούν σε κάποιο άλλον συγκεκριμένο επεξεργαστή. Συγκεκριμένα κάθε φορά το πλήθος των επεξεργαστών που στέλνουν μήνυμα σε άλλους μειώνεται στο μισό. Κάθε φορά επεξεργαστές με μεγαλύτερη ταυτότητα στέλνουν το άθροισμα που κρατούν σε επεξεργαστές με μικρότερη ταυτότητα. Κάθε επεξεργαστής που παίρνει δεδομένο από κάποιο άλλο επεξεργαστή, το αθροίζει στο δικό του τοπικό άθροισμα. Στο τέλος το τελικό αποτέλεσμα το κρατά ο επεξεργαστής με την μικρότερη ταυτότητα.

Σχήμα 6.1 (Παράδειγμα εκτέλεσης παράλληλης άθροισης) [**2,4**]



6.4 Περιγραφή Αλγόριθμου (Βέλτιστη Παράλληλη άθροιση n στοιχεία για άθροιση και n/logn επεξεργαστές. Τα βήματα που παρουσιάζονται δεν αποτελούν τα υπερβήματα του αλγόριθμου. Κάποια από αυτά τα βήματα χρειάζονται να γίνουν περισσότερες από μια φορές. Οπόταν το πλήθος των υπερβημάτων είναι το συνολικό πλήθος εκτέλεσης βημάτων που παρουσιάζουμε πιο κάτω.)

Βήμα 1: Τα στοιχεία μοιράζονται ίσα στους επεξεργαστές (array size=n/ num of processors =n/logn σε κάθε επεξεργαστή).

Βήμα 2: Ο κάθε επεξεργαστής αθροίζει σειριακά τα στοιχεία της ομάδας του (logn στοιχεία).

Βήμα 3: Ορίζουμε μια βοηθητική μεταβλητή **temp= nprocs/2^i** η οποία θα μειώνει συνεχώς στο μισό το πλήθος των επεξεργαστών.

Βήμα 4: Όσο η temp είναι μεγαλύτερη ή ίση με το ένα:

Βήμα 5: Οι επεξεργαστές που έχουν ταυτότητα (pid) μεγαλύτερη από το temp στέλνουν το άθροισμα που κρατούν στο επεξεργαστή με ταυτότητα (pid) ίση με pid-temp, ο οποίος με την σειρά του θα προσθέσει το άθροισμα που πήρε στο δικό του άθροισμα.

Τελικά το τελικό αποτέλεσμα, δηλαδή το άθροισμα όλων των στοιχείων θα το κρατά ο πρώτος επεξεργαστής, δηλαδή ο επεξεργαστής με την ταυτότητα 0.

Χρόνος του αλγόριθμου για παράλληλη άθροιση

Βήμα 1:

Υπολογιστικό μέρος: **arraysize/p=n/n/logn =logn**
Επικοινωνιακό μέρος: **0**
Συγχρονισμός:**0**

Βήμα 2:

Υπολογιστικό μέρος: **arraysize/p= n/n/logn =logn**
Επικοινωνιακό μέρος: **0**
Συγχρονισμός:**0**

Βήμα 3:

Υπολογιστικό μέρος: **1**
Επικοινωνιακό μέρος: **0**
Συγχρονισμός:**0**

Βήμα 4:

logp επαναλήψεις

Βήμα 5:

Υπολογιστικό μέρος: **1**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός:**L**

Συνολικός χρόνος υπολογισμού: 2(arraysize/p)+(g+L+1)logp+1=

2logn +(g+L+1)log(n/logn)+1

6.5 Παρατηρήσεις για την παράλληλη άθροιση

1. Ο χρόνος που χρειάζεται ο αλγόριθμος αυτός για την παράλληλη άθροιση πρέπει να μεγαλώνει όσο μεγαλώνει το μέγεθος της εισόδου.
2. Όσο λιγότεροι είναι οι επεξεργαστές που χρησιμοποιούνται τόσο λιγότερος πρέπει να είναι και ο χρόνος που απαιτείται, διότι ο χρόνος που χρειάζεται ο αλγόριθμος είναι λογαριθμικός ως προς το πλήθος των επεξεργαστών που χρησιμοποιούνται (logp).

6.6 Μετρήσεις

Πιο κάτω παρουσιάζονται τα αποτελέσματα των μετρήσεων για τον αλγόριθμο της παράλληλης άθροισης.

Πίνακας 6.1

Αρ. Στοιχείων	Αρ. Επεξεργαστών	Χρόνος(sec)	Σταλμένα μηνύματα
16	4	0.00518	3
256	32	0.02594	18

Ο πιο πάνω πίνακας δείχνει τους χρόνους του βέλτιστου αλγόριθμου όταν οι επεξεργαστές είναι n/logn και τα στοιχεία n.

Πίνακας 6.2

Αρ. Στοιχείων	Αρ. Επεξεργαστών	Χρόνος(sec)	Σταλμένα μηνύματα
16	**16**	**0.02088**	**15**

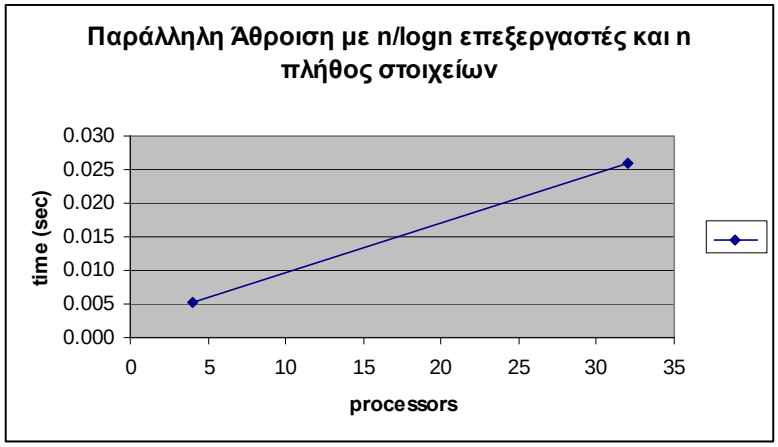
Επίσης πήραμε ακόμη μια μέτρηση στην οποία έχουμε ίσο αριθμό στοιχείων και επεξεργαστών, δημιουργώντας έτσι τον μη βέλτιστο αλγόριθμο παράλληλης άθροισης. Χρησιμοποιήσαμε συγκεκριμένα 16 στοιχεία και επεξεργαστές. Ο χρόνος εκτέλεσης που χρειάστηκε ήταν 0.02088 sec, οπόταν παρατηρήσαμε ότι χρειάζεται περισσότερο χρόνο από το βέλτιστο, ενώ εμείς αναμέναμε το αντίθετο, λόγω του ότι στον μη βέλτιστο δεν χρειάζεται να υλοποιηθεί το σειριακό κομμάτι αφού έχουμε ίσο αριθμό επεξεργαστών και στοιχείων. Αυτό συμβαίνει διότι είναι πολύ μικρό το πλήθος των στοιχείων και των επεξεργαστών οπόταν το σειριακό μέρος δεν έχει πολλές επαναλήψεις και άρα δεν χρειάζεται πολύ χρόνο. Επιπρόσθετα το παράλληλο μέρος χρειάζεται περισσότερο χρόνο στον μη βέλτιστο αλγόριθμο, αφού χρησιμοποιούνται περισσότεροι επεξεργαστές και άρα θα γίνουν περισσότερες επαναλήψεις. Σε συνδυασμό με τα δύο πιο πάνω, τελικά είναι πιο μεγάλος ο χρόνος εκτέλεσης του μη βέλτιστου αλγόριθμου. Σε μεγάλες όμως τιμές των στοιχείων και των επεξεργαστών αναμένουμε να ισχύει η πιο πάνω παρατήρηση, δηλαδή η

απόλυτη τιμή του χρόνου εκτέλεσης του μη βέλτιστου αλγόριθμου να είναι πιο μικρή από τον χρόνο εκτέλεσης του βέλτιστου αλγόριθμου για παράλληλη άθροιση.

6.7 Ανάλυση αποτελεσμάτων

Πιο κάτω παρουσιάζονται οι γραφικές για τις μετρήσεις που παρουσιάστηκαν πιο πάνω.

Γραφική 6.1

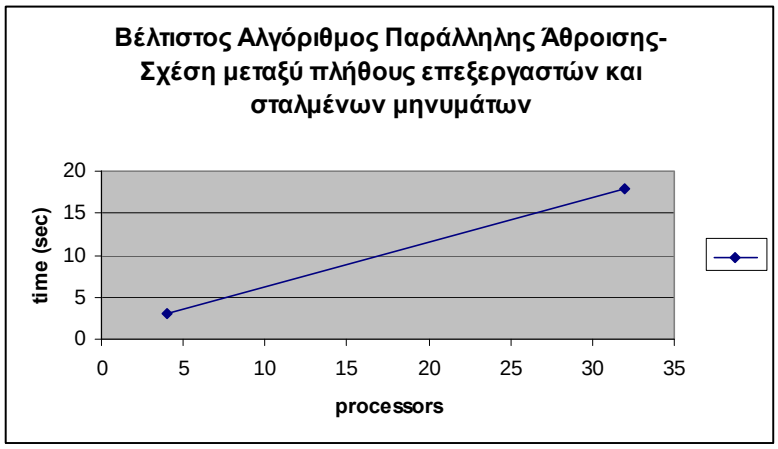


Η πιο πάνω γραφική χρησιμοποιεί τιμές πλήθους των στοιχείων και επεξεργαστών ώστε να έχουμε την εκτέλεση του βέλτιστου αλγόριθμου παράλληλης άθροισης με n/logn επεξεργαστές και n πλήθος στοιχείων.

Η γραφική είναι μια ευθεία όπου μεγαλώνουν οι τιμές του χρόνου όσο μεγαλώνει το μέγεθος του πίνακα και επομένως το πλήθος των στοιχείων. Η σχέση μεταξύ του χρόνου εκτέλεσης και του πλήθους των επεξεργαστών που χρησιμοποιούνται είναι λογαριθμική. Οπόταν θα περιμέναμε και η γραφική παράσταση να ήταν λογαριθμική. Η γραφική όμως είναι μια ευθεία διότι έχουμε πάρει μόνο δύο μετρήσεις, λόγω του περιορισμού που προέρχεται από την χρήση εικονικών επεξεργαστών.

Παρατηρούμε ότι για μεγαλύτερο πλήθος επεξεργαστών και στοιχείων ο χρόνος εκτέλεσης του αλγόριθμου αυξάνεται. Αυτό είναι αναμενόμενο διότι είναι περισσότερα τα στοιχεία που θα αθροιστούν και επίσης περισσότεροι οι επεξεργαστές, οπόταν θα υπάρχει μεγαλύτερη επικοινωνία και επομένως καθυστέρηση, όπως επίσης θα σταλούν και περισσότερα μηνύματα.

Γραφική 6.2



Στην πιο πάνω γραφική παρουσιάζεται η σχέση ανάμεσα στο πλήθος των επεξεργαστών και των σταλμένων μηνυμάτων στον βέλτιστο αλγόριθμο για παράλληλη άθροιση.

Παρατηρούμε ότι η γραφική είναι μια ευθεία που όσο μεγαλώνουν οι τιμές των τετμημένων μεγαλώνουν και οι τιμές των τεταγμένων.

Επομένως παρατηρούμε ότι όσο περισσότερους επεξεργαστές χρησιμοποιούμε τόσο μεγαλύτερος είναι το πλήθος των μηνυμάτων που στέλνονται κατά την διάρκεια του αλγόριθμου.

Αυτό είναι αναμενόμενο, διότι περισσότεροι επεξεργαστές οδηγεί στο να χρειαζόμαστε περισσότερη επικοινωνία, άρα και αποστολή περισσότερων μηνυμάτων.

Ακόμη λόγω της σχέσης που υπάρχει μεταξύ του πλήθους των επεξεργαστών και των στοιχείων για άθροιση, μπορούμε να παρατηρήσουμε ότι περισσότεροι επεξεργαστές συνεπάγονται και περισσότερα στοιχεία για άθροιση, γεγονός που και πάλι οδηγεί σε περισσότεροι επικοινωνία, οπόταν περισσότερα μηνύματα.

6.8 Συμπεράσματα

Πιο κάτω παρουσιάζονται τα συμπεράσματα από την εκτέλεση του αλγόριθμου παράλληλης άθροισης και τις μετρήσεις που πάρθηκαν.

1. Όντως όσο περισσότεροι είναι οι επεξεργαστές που χρησιμοποιούνται, τόσο μεγαλύτερος είναι ο χρόνος που χρειάζεται ο αλγόριθμος, διότι ο χρόνος εκτέλεσης του αλγόριθμου είναι λογαριθμικός ως προς το πλήθος των επεξεργαστών. Όσοι περισσότεροι είναι οι επεξεργαστές χρειάζεται να γίνουν περισσότερα υπερβήματα και χρειάζεται περισσότερος χρόνος για συγχρονισμό των επεξεργαστών. Οπόταν συνολικά ο χρόνος εκτέλεσης είναι μεγαλύτερος.
2. Όσο περισσότεροι είναι οι επεξεργαστές τόσο περισσότερα είναι και τα μηνύματα που στέλνονται κατά την διάρκεια της άθροισης διότι πρέπει να επικοινωνήσουν περισσότεροι επεξεργαστές μεταξύ τους για να γίνει η άθροιση.
3. Όσο περισσότερα είναι τα στοιχεία που θα αθροιστούν τόσο μεγαλώνει και ο χρόνος που χρειαζόμαστε για την άθροιση. Αυτό οφείλεται στο ότι οι επεξεργαστές πρέπει να κάνουν περισσότερη δουλειά, διότι είναι περισσότερα τα στοιχεία που πρέπει να αθροιστούν, συνεπώς δικαιολογημένα χρειάζεται περισσότερος χρόνος.

ΚΕΦΑΛΑΙΟ 7

Δίκτυα Αλληλοσύνδεσης ΔΑ
Παράλληλη Άθροιση και Παράλληλος Προθεματικός Υπολογισμός

Στο Κεφάλαιο 7 παρουσιάζονται δύο αλγόριθμοι υλοποιημένοι σε τετραγωνικό πλέγμα (mesh). Ο πρώτος αλγόριθμος που παρουσιάζεται είναι ο αλγόριθμος για παράλληλη άθροιση και ο δεύτερος για παράλληλη προθεματική άθροιση. Έτσι στο Υποκεφάλαιο 7.1 επεξήγονται τα δίχτυα αλληλοσύνδεσης, και ακολούθως ακολουθεί η αναλυτική περιγραφή των δύο αυτών αλγόριθμων. Στα Υποκεφάλαια 7.2.1, 7.2.2 και 7.2.3 περιγράφεται το πρόβλημα της παράλληλης άθροισης σε τετραγωνικό δίχτυο αλληλοσύνδεσης, η παράλληλη της λύση και ο αλγόριθμός της, και στα Υποκεφάλαια 7.2.4, 7.2.5 και 7.2.6 περιγράφεται το πρόβλημα της παράλληλης προθεματικής άθροισης

σε τετραγωνικό δίκτυο αλληλοσύνδεσης, η παράλληλη λύση του προβλήματος και η περιγραφή του αλγόριθμου. Ακολουθεί το Υποκεφάλαιο 7.2.7 στο οποίο γίνονται παρατηρήσεις για τους δύο αλγόριθμους και στην συνέχεια στα Υποκεφάλαια 7.2.8, 7.2.9, 7.2.10 παρουσιάζονται οι μετρήσεις, οι γραφικές παραστάσεις και τα συμπεράσματα.

7.1 Δίκτυα Αλληλοσύνδεσης (ΔΑ)

Τα δίκτυα αλληλοσύνδεσης χρησιμοποιούνται ως ένας τρόπος επικοινωνίας μεταξύ επεξεργαστών σε παράλληλους αλγόριθμους. Συγκεκριμένα υπάρχουν n πανομοιότυποι επεξεργαστές, όπου ο κάθε ένας έχει την δική του τοπική μνήμη και διαφορετικό ρεύμα δεδομένων. Πιο κάτω θα παρουσιαστούν κάποια σημαντικά χαρακτηριστικά για τα δίκτυα αλληλοσύνδεσης και δύο αλγόριθμοι βασισμένοι στον τρόπο λειτουργίας των δικτύων αλληλοσύνδεσης.

Παραδείγματα διχτύων αλληλοσύνδεσης σε SIMD είναι:

- Γραμμικό πλέγμα (Line Array), Δαχτύλιος (Ring)
- Αστέρι (Star)
- Δυαδικό δέντρο (Binary Tree)
- Ορθογώνιο Πλέγμα (2-Dimensional Array or Mesh)
- Κύβος (Cube), Υπερκύβος (Hypercube)

Βασικά θέματα για σχεδιασμό μοντέλων ΔΑ:

Τα μοντέλα ΔΑ μπορούμε να τα αναπαραστήσουμε σαν μη κατευθυνόμενους γράφους G=(V,E). Κάθε κόμβος αυτού του γράφου (όπου $i \in V$) είναι ένας επεξεργαστής στο σύστημά μας. Επίσης κάθε ακμή $(i,j) \in E$ αντιπροσωπεύει ένα αμφικατευθυνόμενο σύνδεσμο μεταξύ των επεξεργαστών i και j.

Τα δεδομένα του προβλήματος είναι διαμοιρασμένα στους επεξεργαστές οι οποίοι επικοινωνούν μεταξύ τους, ανταλλάσοντας μηνύματα. Σε ένα βήμα, ο κάθε επεξεργαστής μπορεί να επικοινωνήσει μόνο με τους γειτονικούς κόμβους.

Σημαντικές παράμετροι ΔΑ [2]

Διάμετρος επικοινωνίας: Αποτελεί την διάμετρο του γράφου που αναπαριστά το ΔΑ, και είναι η μεγαλύτερη «απόσταση» μεταξύ δύο κόμβων. Συγκεκριμένα για κάθε ζεύγος κόμβων, υπολογίζουμε την μικρότερη τους απόσταση. Η διάμετρος είναι η μεγαλύτερη μικρότερη απόσταση μεταξύ όλων των ζευγαριών. Επιπρόσθετα αν η διάμετρος είναι d, τότε η επικοινωνία μεταξύ των κόμβων που απέχουν περισσότερο μεταξύ τους μέσα στο δίχτυο, παίρνει χρόνο Ω(d). Οπόταν συμπεράνουμε ότι ο χρόνος που χρειάζεται για οποιοδήποτε υπολογισμό, όπου τα δεδομένα εισόδου είναι διαμοιρασμένα στους επεξεργαστές είναι Ω(d).

Μέγιστο βάθος: Αποτελεί τον μέγιστο αριθμό γειτονικών κόμβων οποιουδήποτε κόμβου.

Αριθμός Συνδέσεων: Είναι ο συνολικός αριθμός των αμφικατευθυνόμενων συνδέσμων που υπάρχουν στο δίκτυο.

7.2 Περιγραφή αλγόριθμων

7.2.1 Άθροιση σε τετραγωνικό πλέγμα (mesh)

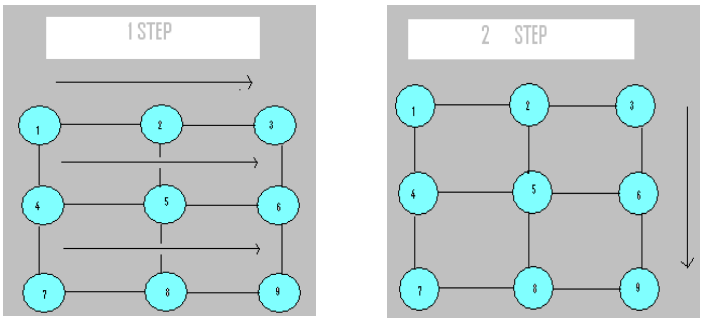
Πρόβλημα της άθροισης σε τετραγωνικό πλέγμα: Άθροιση δεδομένων που έχουν αποθηκευμένα στην τοπική τους μνήμη **n** επεξεργαστές οι οποίοι βρίσκονται τοποθετημένοι σε διάταξη όπου σχηματίζει ένα τετραγωνικό πλέγμα. Ο κάθε επεξεργαστής δεν μπορεί να επικοινωνήσει με οποιοδήποτε από τους άλλους επεξεργαστές. Μπορεί να επικοινωνεί μόνο με γειτονικούς επεξεργαστές, δηλαδή επεξεργαστές που δεν έχουν μεταξύ τους άλλους επεξεργαστές.

7.2.2 Παράλληλη λύση: Για την παράλληλη αυτή λύση χρησιμοποιούνται **n** επεξεργαστές όταν το δίχτυο μας είναι ένα $\sqrt{\mathbf{n}}$ **x** $\sqrt{\mathbf{n}}$ τετράγωνο. Κάθε επεξεργαστής, που είναι πρώτος

στην σειρά του, έχει αποθηκευμένο στην μνήμη του ένα στοιχείο το οποίο αρχικά πρέπει να το στείλει στον γείτονα του στα δεξιά. Ο γείτονας του παίρνοντας το στοιχείο το αθροίζει στα δικά του δεδομένα και με την σειρά του, στέλνει το στοιχείο που κρατά στο γείτονα του στα δεξιά. Η διαδικασία αυτή επαναλαμβάνεται μέχρι ο τελευταίος επεξεργαστής που πήρε δεδομένο να μην έχει στα δεξιά του άλλο επεξεργαστή. Επιπρόσθετα αυτή η διαδικασία επαναλαμβάνεται παράλληλα σε όλες τις σειρές του διχτύου - τετραγώνου. Ακολούθως μεταφέρονται με παρόμοιο τρόπο τα δεδομένα κάθε επεξεργαστή που βρίσκεται στην τελευταία στήλη του δικτύου – τετραγώνου, προς τον γειτονικό του επεξεργαστή που βρίσκεται αμέσως από κάτω του. Με αυτό τον τρόπο τελικά, ο επεξεργαστής που βρίσκεται στη χαμηλότερη θέση της τελευταίας στήλης του δικτύου- τετραγώνου, αποθηκεύει στην τοπική του μνήμη το άθροισμα όλων των αριθμών.

Σχήμα 7.1

(n=9, 3x3 mesh) (παράδειγμα εκτέλεσης παράλληλης άθροισης σε τετραγωνικό πλαίσιο)



7.2.3 Περιγραφή Αλγόριθμου για παράλληλη λύση

Βήμα 1: Ενόσω υπάρχει επόμενος επεξεργαστής στα δεξιά του επεξεργαστή:

Βήμα 2: Ο κάθε επεξεργαστής στέλνει στον επεξεργαστή με την αμέσως μεγαλύτερη ταυτότητα αν υπάρχει το στοιχείο του.

Βήμα 3: Οι επεξεργαστές που παίρνουν κάποιο στοιχείο από κάποιο άλλο επεξεργαστή το αθροίζουν στο δικό τους τοπικό άθροισμα.

Βήμα 4: Ενόσω υπάρχει μεγαλύτερος επεξεργαστής που είναι ο τελευταίος της επόμενης ομάδας:

Βήμα 5: Κάθε τελευταίος επεξεργαστής από κάθε ομάδα, ξεκινώντας από την πρώτη ομάδα και προχωρώντας σειριακά, στέλνει το τοπικό του άθροισμα στον τελευταίο επεξεργαστή της επόμενης ομάδας, ο οποίος το αθροίζει στο δικό του τοπικό άθροισμα..

Ο επεξεργαστής με την μεγαλύτερη ταυτότητα κρατά στο τέλος στην τοπική του μνήμη το αποτέλεσμα της άθροισης.

Χρόνος του αλγόριθμου για παράλληλη άθροιση:

Βήμα 1

Βρόγχος που επαναλαμβάνεται $\sqrt{p}$ φορές.

Βήμα 2

Υπολογιστικό μέρος: **0**
Επικοινωνιακό μέρος: **g**
Συγχρονισμός: **L**

Βήμα 3

Υπολογιστικό μέρος: **1**
Επικοινωνιακό μέρος: **0**
Συγχρονισμός: **0**

Βήμα 4

Βρόγχος που επαναλαμβάνεται √p φορές.

Βήμα 5

Υπολογιστικό μέρος: **1**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός: **L**

Συνολικός Χρόνος παράλληλης άθροισης σε τετραγωνικό πλέγμα: $2\sqrt{p}((g+L)+1)$

7.2.4 Προθεματική άθροιση σε τετραγωνικό πλέγμα (mesh)

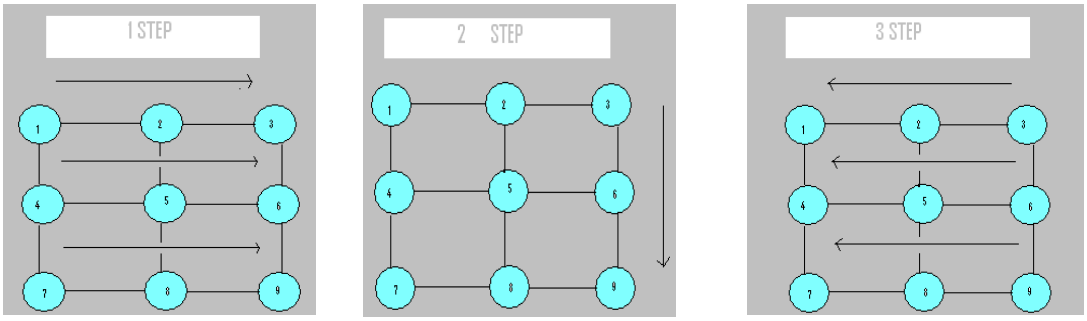
Πρόβλημα της προθεματικής άθροισης σε τετραγωνικό πλέγμα: Προθεματική άθροιση δεδομένων που έχουν αποθηκευμένα στην τοπική τους μνήμη **n** επεξεργαστές οι οποίοι βρίσκονται τοποθετημένοι σε διάταξη όπου σχηματίζει ένα τετραγωνικό πλέγμα. Ο κάθε επεξεργαστής δεν μπορεί να επικοινωνήσει με οποιοδήποτε από τους άλλους επεξεργαστές. Μπορεί να επικοινωνεί μόνο με γειτονικούς επεξεργαστές, δηλαδή επεξεργαστές που δεν έχουν μεταξύ τους άλλους επεξεργαστές.

7.2.5 Παράλληλη λύση: Για την παράλληλη αυτή λύση χρησιμοποιούνται **n** επεξεργαστές όταν το δίχτυο μας είναι ένα $\sqrt{\mathbf{n}}$ **x** $\sqrt{\mathbf{n}}$ τετράγωνο. Κάθε επεξεργαστής, που είναι πρώτος στην σειρά του, έχει αποθηκευμένο στην μνήμη του ένα στοιχείο το οποίο αρχικά πρέπει να το στείλει στον γείτονα του στα δεξιά. Ο γείτονας του παίρνοντας το στοιχείο το αθροίζει στα δικά του δεδομένα και με την σειρά τους στέλνει το στοιχείο που κρατά στο γείτονα του στα δεξιά. Η διαδικασία αυτή επαναλαμβάνεται μέχρι ο τελευταίος επεξεργαστής που πήρε δεδομένο να μην έχει στα δεξιά του άλλο επεξεργαστή. Επιπρόσθετα αυτή η διαδικασία επαναλαμβάνεται παράλληλα σε όλες τις σειρές του διχτύου - τετραγώνου. Ακολούθως μεταφέρονται με παρόμοιο τρόπο τα δεδομένα κάθε επεξεργαστή στην τελευταία στήλη του διχτύου – τετραγώνου, και κάθε φορά που ένας

επεξεργαστής παραλαμβάνει δεδομένο το αθροίζει στα δικά του δεδομένα. Τέλος επαναλαμβάνεται μια διαδικασία όμοια με το πρώτο βήμα όμως τώρα με την αντίθετη φορά κίνησης των μηνυμάτων. Δηλαδή τώρα μεταφέρονται παράλληλα σε κάθε στήλη τα δεδομένα από τον δεξιότερο επεξεργαστή κάθε σειράς προς τον αριστερότερο. Σε κάθε βήμα κάθε επεξεργαστής που παραλαμβάνει κάποιο στοιχείο το αθροίζει στα δεδομένα του.

Σχήμα 7.2:

(n=9, 3x3 mesh) (παράδειγμα εκτέλεσης προθεματικής άθροισης σε τετραγωνικό πλαίσιο)



7.2.6 Περιγραφή Αλγόριθμου για παράλληλη λύση

Βήμα 1 Ενόσω υπάρχει επόμενος επεξεργαστής στα δεξιά του επεξεργαστή:

Βήμα 2 Ο κάθε επεξεργαστής στέλνει στον επεξεργαστή με την αμέσως μεγαλύτερη ταυτότητα αν υπάρχει, το στοιχείο του.

Βήμα 3 Οι επεξεργαστές που παίρνουν κάποιο στοιχείο από κάποιο άλλο επεξεργαστή το αθροίζουν στο δικό τους τοπικό άθροισμα.

Βήμα 4 Ενόσω υπάρχει μεγαλύτερος επεξεργαστής που είναι ο τελευταίος της επόμενης ομάδας:

Βήμα 5 Κάθε τελευταίος επεξεργαστής από κάθε ομάδα, ξεκινώντας από την πρώτη ομάδα και προχωρώντας σειριακά, στέλνει το τοπικό του άθροισμα στον τελευταίο επεξεργαστή της επόμενης ομάδας, ο οποίος το αθροίζει στο δικό του τοπικό άθροισμα..

Βήμα 6 Οι τελευταίοι επεξεργαστές κάθε ομάδας, εκτός από αυτόν της πρώτης ομάδας, στέλνει το τοπικό του άθροισμα στο επεξεργαστή που έχει την αμέσως μικρότερη ταυτότητα και βρίσκεται στην ομάδα του.

Βήμα 7 Ενόσω υπάρχει προηγούμενος επεξεργαστής στα αριστερά του επεξεργαστή που παρέλαβε μήνυμα:

Βήμα 8 Κάθε επεξεργαστής που παίρνει στοιχείο από άλλον επεξεργαστή το αθροίζει στο τοπικό του άθροισμα και το στέλνει το τελικό άθροισμα στον επεξεργαστή που έχει την αμέσως μικρότερη ταυτότητα και βρίσκεται στην ομάδα του.

Χρόνος του αλγόριθμου για παράλληλη άθροιση:

Βήμα 1

Βρόγχος που επαναλαμβάνεται $\sqrt{p}$ φορές.

Βήμα 2

Υπολογιστικό μέρος: **0**
Επικοινωνιακό μέρος: **g**
Συγχρονισμός: **L**

Βήμα 3

Υπολογιστικό μέρος: **1**
Επικοινωνιακό μέρος: **0**
Συγχρονισμός: **0**

Βήμα 4

Βρόγχος που επαναλαμβάνεται √p φορές.

Βήμα 5

Υπολογιστικό μέρος: **1**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός: **L**

Βήμα 6

Υπολογιστικό μέρος: **0**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός: **L**

Βήμα 7

Βρόγχος που επαναλαμβάνεται √p φορές.

Βήμα 8

Υπολογιστικό μέρος: **1**

Επικοινωνιακό μέρος: **g**

Συγχρονισμός: **L**

Συνολικός Χρόνος παράλληλης άθροισης σε τετραγωνικό πλέγμα:

$3\sqrt{p}((g+L)+1) +(g+L)$

7.2.7 Παρατηρήσεις για την παράλληλη άθροιση και τον προθεματικό υπολογισμό στο ΔΑ mesh

1. Ασυμπτωτικά ο χρόνος που χρειάζονται οι δύο αλγόριθμοι είναι ο ίδιος **O(√p(g+L)).** Οπόταν ασυμπτωτικά χρειάζονται τον ίδιο χρόνο, όμως ο χρόνος που χρειάζονται σε απόλυτες τιμές είναι διαφορετικός. Συγκεκριμένα η παράλληλη

άθροιση χρειάζεται λιγότερο χρόνο και αυτό είναι και φυσικό διότι σε αυτό τον αλγόριθμο γίνονται λιγότερα βήματα για την ολοκλήρωση του. Στην ουσία οι δύο αλγόριθμοι μέχρι κάποιο σημείο υλοποιούν ακριβώς το ίδιο πράγμα, ενώ στην συνέχεια ο αλγόριθμος του προθεματικού υπολογισμού περιέχει ακόμη κάποια επιπρόσθετα βήματα.

2. Από την εξίσωση για τον χρόνο τον δυο αλγόριθμων συμπεραίνουμε ότι ο χρόνος τους πρέπει να αυξάνεται όσο το πλήθος των επεξεργαστών και άρα όσο το μέγεθος του διχτύου αλληλοσύνδεσης μεγαλώνει.
3. Μέχρι και το βήμα 4 οι δυο αλγόριθμοι στέλνουν ακριβώς το ίδιο πλήθος μηνυμάτων. Τελικά βέβαια στην προθεματική άθροιση θα σταλούν περισσότερα μηνύματα.

7.2.8 Μετρήσεις

Πιο κάτω ακολουθούν μετρήσεις για τις διαφορετικές παραμέτρους των αλγόριθμων Παράλληλης Άθροισης και Προθεματικής Παράλληλης Άθροισης.

Πίνακας 7.2.1:

Αρ. Επεξεργαστών	Προθεματική Άθροιση- Χρόνος	Άθροιση- Χρόνος
4	0,00297	0,00236
25	0,03509	0,02552
36	0,05881	0,04342
49	0,10057	0,06661
64	0,1356	0,10194
81	0,19357	0,14559
100	0,26662	0,18958
121	0,34505	0,25074
144	0,44526	0,31864

169	0,56791	0,40488

Στο πιο πάνω πίνακα φαίνονται οι χρόνοι εκτέλεσης για τους δύο αλγόριθμους.

Παρατηρούμε ότι στην προθεματική άθροιση ο χρόνος είναι μεγαλύτερος. Αυτό είναι φυσικό διότι χρειάζονται να γίνουν σε αυτή κάποια επιπρόσθετα βήματα τα οποία θα χρειάζονται σίγουρα και περισσότερο χρόνο.

Πίνακας 7.2.2

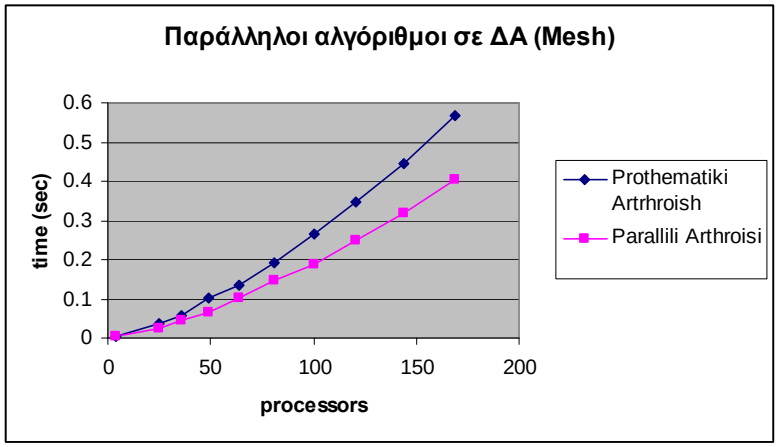
Αρ. Επεξεργαστών	**Προθεματική Άθροιση- Πλήθος Μηνυμάτων**		**Άθροιση- Πλήθος μηνυμάτων**
4	(3+4)	7	3
25	(54+70)	124	54
36	(95+120)	215	95
49	(153+189)	342	153
64	(231+280)	511	231
81	(332+396)	728	332
100	(459+540)	999	459
121	(615+715)	1330	615
144	(803+924)	1727	803
169	(1026+1170)	2196	1026

Στον πίνακα πιο πάνω φαίνεται το πλήθος των σταλμένων μηνυμάτων και η σχέση του με το πλήθος των επεξεργαστών που χρησιμοποιούνται στον κάθε ένα από τους δύο αλγόριθμους.

7.2.9 Ανάλυση Αποτελεσμάτων

Ακολουθούν γραφικές για τις μετρήσεις που παρουσιάστηκαν πιο πάνω.

Γραφική 7.2.1

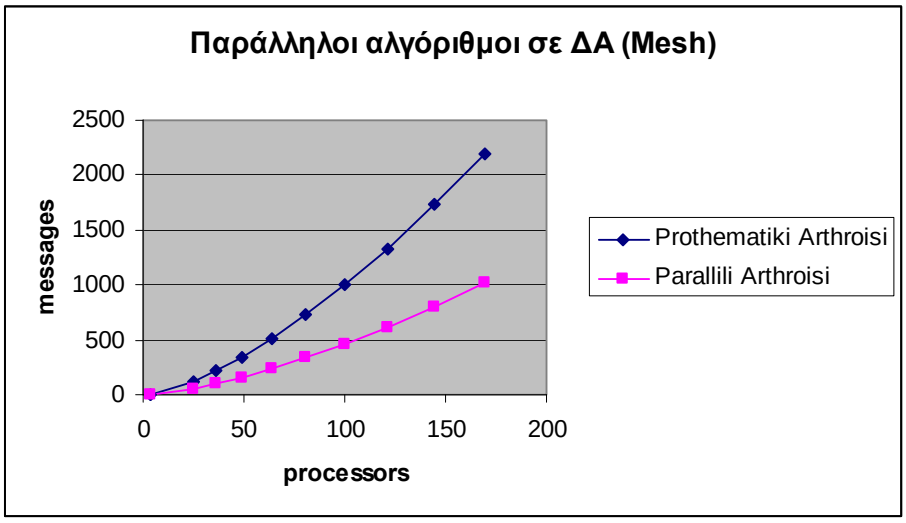


Η πιο πάνω γραφική δείχνει την σχέση μεταξύ του πλήθους των επεξεργαστών και του χρόνου εκτέλεσης. Παρατηρούμε ότι και για τους δύο αλγόριθμους η γραφική είναι μια καμπύλη που κατευθύνεται προς τα πάνω. Αυτό δείχνει ότι όσο περισσότεροι είναι οι επεξεργαστές τόσο περισσότερος είναι ο χρόνος εκτέλεσης. Βέβαια ο χρόνος παρατηρώντας την μαθηματική σχέση που παρουσιάσαμε πιο πάνω περιμένουμε να είναι ανάλογος με την τετραγωνική ρίζα του πλήθους των επεξεργαστών. Αυτό δεν ισχύει όπως μπορούμε να δούμε από την γραφική και οφείλεται στο γεγονός ότι τα πειράματα έγιναν σε εικονικούς επεξεργαστές που δημιουργούνται πάνω σε ένα πραγματικό επεξεργαστή, ενώ οι αρχικές παρατηρήσεις και τα αναμενόμενα αποτελέσματα αναφέρονται στην χρήση πραγματικών επεξεργαστών.

Ακόμη παρατηρούμε ότι ο χρόνος για την προθεματική άθροιση είναι πάντα περισσότερος από τον χρόνο εκτέλεσης της παράλληλης άθροισης στο ΔΑ mesh. Αυτό ισχύει διότι στην προθεματική άθροιση αρχικά γίνονται τα ίδια βήματα με την παράλληλη άθροιση, όμως

ακολούθως χρειάζεται να γίνουν κάποια επιπρόσθετα βήματα για την ολοκλήρωση του αλγόριθμου αυτού.

Γραφική 7.2.2



Στην γραφική αυτή φαίνεται η σχέση του πλήθους των επεξεργαστών σε σχέση με τα μηνύματα που στέλνονται.

Παρατηρούμε ότι και στους δύο αλγόριθμους η γραφική είναι μια καμπύλη που κατευθύνεται προς τα πάνω.

Αυτό μας δείχνει ότι όσο περισσότερους επεξεργαστές χρησιμοποιούμε, το πλήθος των μηνυμάτων που στέλνονται στους αλγόριθμους μεγαλώνει. Οπόταν υπάρχει περισσότερη επικοινωνία όταν έχουμε περισσότερους επεξεργαστές.

Ακόμη βλέπουμε ότι ανταλλάσσονται περισσότερα μηνύματα στην προθεματική άθροιση, αφού υπάρχουν κάποια επιπρόσθετα βήματα τα οποία πρέπει να γίνουν για την ολοκλήρωση του αλγόριθμου.

Τέλος από τους πίνακες πιο πάνω παρατηρούμε επίσης ότι τα μηνύματα στέλνονται ακριβώς με τον ίδιο τρόπο στους δύο αλγόριθμους, όταν έχουμε ίσο πλήθος επεξεργαστών, και μέχρι κάποιο σημείο του αλγόριθμου για προθεματική άθροιση. Στην συνέχεια υπάρχουν επιπρόσθετα βήματα στον αλγόριθμο για προθεματική

άθροιση, οπόταν για αυτό στέλνονται περισσότερα μηνύματα. Επίσης για τον ίδιο λόγω ο χρόνος είναι μεγαλύτερος στην προθεματική άθροιση.

7.2.10 Συμπεράσματα

Πιο κάτω ακολουθούν συμπεράσματα για τις μετρήσεις από την εκτέλεση των δύο αλγόριθμων που αναφέρονται πιο πάνω.

1. Όντως από τις μετρήσεις παρατηρούμε ότι όσο το πλήθος των επεξεργαστών μεγαλώνει (n), αυξάνεται και ο χρόνος που χρειάζονται οι δύο αλγόριθμοι. Αυτό συμβαίνει διότι το πλήθος των επεξεργαστών σχετίζεται με τον χρόνο εκτέλεσης του αλγόριθμου. Συγκεκριμένα ο χρόνος είναι ανάλογος προς το √p (τετραγωνική ρίζα των επεξεργαστών).
2. Ο χρόνος για την άθροιση είναι μικρότερος από τον χρόνο που χρειάζεται ο αλγόριθμος για την προθεματική άθροιση, λόγω του ότι στον αλγόριθμο για προθεματική άθροιση γίνεται επιπλέον δουλειά για την ολοκλήρωση του. Τα βήματα είναι αρχικά τα ίδια και στους δύο αλγόριθμους. Από ένα σημείο και μετά όμως ο αλγόριθμος της παράλληλης άθροισης ολοκληρώνεται ενώ στον αλγόριθμο προθεματικής άθροισης γίνονται επιπλέον βήματα.
3. Τα μηνύματα που στέλνονται κατά την προθεματική άθροιση μεταξύ των επεξεργαστών, μέχρι κάποιο σημείο είναι ακριβώς τα ίδια με αυτά στη παράλληλη άθροιση διότι η προθεματική άθροιση είναι η παράλληλη άθροιση με κάποια επιπρόσθετα βήματα.

ΚΕΦΑΛΑΙΟ 8

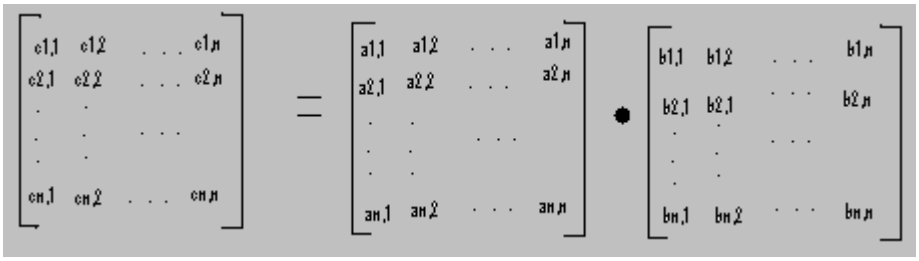
Πολλαπλασιασμός Πινάκων

Στο Κεφάλαιο 8 παρουσιάζεται το πρόβλημα του πολλαπλασιασμού πινάκων και αλγόριθμοι για την επίλυση του. Αρχίζοντας με το Υποκεφάλαιο 8.1 παρουσιάζεται η περιγραφή του προβλήματος και ακολουθούν τα Υποκεφάλαια 8.2, 8.3 στα οποία περιγράφεται η παράλληλη λύση του προβλήματος. Στην συνέχεια στο Υποκεφάλαιο 8.4 γίνονται κάποιες παρατηρήσεις όσο αφορά την συμπεριφορά των τεσσάρων αλγόριθμων για πολλαπλασιασμό πινάκων, που υλοποιήθηκαν. Στην συνέχεια ακολουθούν μετρήσεις. Γραφικές παραστάσεις και συμπεράσματα στα Υποκεφάλαια 8.5, 8.6 και 8.7.

8.1 Πρόβλημα Πολλαπλασιασμού Πινάκων Μας δίνονται δύο πίνακες μεγέθους **n** x **n**, έστω **A** και **B**, και μας ζητείται να υπολογίσουμε το γινόμενο τους **C=AB**, δηλαδή να υπολογίσουμε τα n^2 στοιχεία:

$$c_{i,j}=\Sigma_{k=1-n}\ a_{i,k}\ .b_{k,j} \quad i<=i,j,k<=n$$

<u>Σχήμα 8.1:</u> (παράδειγμα πινάκων για πολλαπλασιασμό)



8.2 Παράλληλη λύση

Πιο κάτω θα παρουσιαστούν **4 διαφορετικοί αλγόριθμοι** για πολλαπλασιασμό πινάκων.
Οι τρεις από αυτούς υλοποιούν τον αλγόριθμο πολλαπλασιασμού πινάκων με την διαφορά ότι ο καθένας από τους τρεις χρησιμοποιεί διαφορετικό αλγόριθμο για την μετάδοση των στοιχείων (έναν από τους τρεις που παρουσιάστηκαν πιο πάνω), και ο τέταρτος υλοποιεί τον αλγόριθμο πολλαπλασιασμού πινάκων με διαφορετικό τρόπο.

8.3 Περιγραφή Παράλληλης λύσης

Ο κάθε επεξεργαστής εκτός από την ταυτότητα του pid θα ανήκει και σε μια ομάδα έστω k. Επίσης νοητά οι επεξεργαστές τοποθετούνται σε διάταξη όπως είναι τα στοιχεία των πινάκων που πρόκειται να πολλαπλασιάσουν. Με αυτό τον τρόπο δίνουμε δυο χαρακτηριστικά στους επεξεργαστές το i και το j, ώστε να μπορούμε στην συνέχεια να

αντιστοιχίσουμε το στοιχεία των πινάκων με τους επεξεργαστές και να κάνουμε κατάλληλα τον πολλαπλασιασμό. Στο σημείο αυτό θα αναφέρουμε ότι οι πίνακες που χρησιμοποιούμε είναι τετραγωνικοί (nxn) και το πλήθος των επεξεργαστών μας πρέπει να είναι ανάλογο ως προς το μέγεθος των πινάκων, συγκεκριμένα το πλήθος των επεξεργαστών πρέπει να είναι n^3.

Κατά την διάρκεια της εκτέλεσης του αλγόριθμου αρχικά ο κάθε [i, k, *] επεξεργαστής παίρνει το στοιχείο $a_{i,k}$ του πίνακα A και αντίστοιχα ο κάθε [*, k, j] επεξεργαστής παίρνει το $b_{k,j}$ του πίνακα B. Τότε πολλαπλασιάζει τα στοιχεία που πήρε $a_{i,k}$ και $b_{k,j}$. Το γινόμενο τους χρησιμοποιείται για να υπολογιστούν τα στοιχεία $c_{i,j}$. Τα στοιχεία αυτά που αποτελούν και τον τελικό πίνακα C υπολογίζονται χρησιμοποιώντας τον αλγόριθμο παράλληλης άθροισης με **n** επεξεργαστές. Πρέπει επίσης να σημειωθεί ότι για τον τέταρτο αλγόριθμο για πολλαπλασιασμό πινάκων υπάρχει διαφορά από τους άλλους αλγόριθμους στο ότι μπορούν ταυτόχρονα να διαβάζουν από άλλους επεξεργαστές δεδομένα που υπάρχουν στη τοπική τους μνήμη.

Παράδειγμα 8.1

Με το παράδειγμα αυτό φαίνεται ο τρόπος λειτουργίας του αλγόριθμου.

Σχήμα 8.1

(n=2 οπόταν θα έχουμε p=8 επεξεργαστές, πίνακες για πολλαπλασιασμό)

A **B**

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Μετάδοση των κατάλληλων στοιχείων $a_{i,k}$ και $b_{k,j}$ στους κατάλληλους επεξεργαστές:

(0,0,*)	(0,0,0) pid=0	(*,0,0)	(0,0,0) pid=0
	(0,0,1) pid=1		(1,0,0) pid=2
(0,1,*)	(0,1,0) pid=4	(*,0,1)	(0,0,1) pid=1
	(0,1,1) pid=5		(1,0,1) pid=3

(1,0,*) (1,0,0) pid=2 (*,1,0) (0,1,0) pid=4

(1,0,1) pid=3 (1,1,0) pid=6

(1,1,*) (1,1,0) pid=6 (*,1,1) (0,1,1) pid=5

(1,1,1) pid=7 (1,1,1) pid=7

Πολλαπλασιασμός των στοιχείων ai,k και bk,j**:**

pid=0 A(0,0)*B(0,0)=1*1=1

pid=1 A(0,0)*B(0,1)=1*2=2

pid=2 A(1,0)*B(0,0)=3*1=3

pid=3 A(1,0)*B(0,1)=3*2=6

pid=4 A(0,1)*B(1,0)=2*3=6

pid=5 A(0,1)*B(1,1)=2*4=8

pid=6 A(1,1)*B(1,0)=4*3=12

pid=7 A(1,1)*B(1,1)=4*4=16

Ακολουθεί παράλληλη άθροιση:

pid=0, pid=4 c0,0=1+6=7

pid=1, pid=5 c0,1=2+8=10

pid=2, pid=6 c1,0=3+12=15

pid=3, pid=7 c1,1=6+16=22

Αποτέλεσμα

Σχήμα 8.2: (αποτέλεσμα πολλαπλασιασμού)

C **A** **B**

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Παρουσίαση 4 αλγόριθμων για πολλαπλασιασμό πινάκων

Πιο κάτω παρουσιάζονται **τρεις διαφορετικοί αλγόριθμοι** που υλοποιήθηκαν για πολλαπλασιασμό πινάκων. Συγκεκριμένα παρουσιάζουμε ένα αλγόριθμο διότι οι τρεις αυτοί αλγόριθμοι, έχουν την μόνηθ διαφορά ότι χρησιμοποιούν διαφορετική συνάρτηση για μετάδοση δεδομένων. Όμως όλος ο υπόλοιπος κώδικας είναι ακριβώς ο ίδιος.

Βήμα 1: Το στοιχείο $a_{i,k}$ μεταδίδεται σε όλους τους επεξεργαστές [i,k,*].
Το στοιχείο b_{kj} μεταδίδεται σε όλους τους επεξεργαστές [*,k,j].
Ο επεξεργαστής [i,k,j] διαβάζει τα $a_{i,k}$ και $b_{k,j}$ και υπολογίζει το γινόμενο τους.

Βήμα 2: Το κάθε $c_{i,j}$ υπολογίζεται χρησιμοποιώντας τον αλγόριθμο παράλληλης άθροισης με n επεξεργαστές.

Στον αλγόριθμο πολλαπλασιασμού πινάκων χρησιμοποιούνται αλγόριθμοι όπως οι τρεις αλγόριθμοι για μετάδοση στοιχείων και ο αλγόριθμος παράλληλης άθροισης, που παρουσιάστηκαν πιο πάνω. Ο αλγόριθμος για μετάδοση στοιχείων χρησιμοποιείται όταν θα χρειάζεται ένας επεξεργαστής να πάρει κάποιο στοιχείο που το κρατά στην τοπική του μνήμη κάποιος άλλος επεξεργαστής. Ο αλγόριθμος παράλληλης άθροισης χρησιμοποιείται για τον υπολογισμό των τελικών $c_{i,j}$ στοιχείων του πίνακα αποτέλεσμα C.

Πιο κάτω θα παρουσιαστεί ο χρόνος εκτέλεσης για καθένα από αυτούς τους αλγόριθμους.

Ξεκινώντας με το αλγόριθμο ο οποίος χρησιμοποιεί τον πρώτο αλγόριθμο για μετάδοση δεδομένων, βλέπουμε ότι γίνονται ουσιαστικά δύο βήματα. Στο πρώτο βήμα παρατηρούμε ότι γίνεται η μετάδοση των δεδομένων μεταξύ των επεξεργαστών και επομένως ο χρόνος που χρειάζεται είναι **1+(p-1)g+L,** (υπολογιστικό μέρος: **1**, επικοινωνιακό μέρος: **(p-1)g**, συγχρονισμός: **L.** Ακολούθως στο δεύτερο βήμα στόχος είναι ο υπολογισμός των

στοιχείων που απαρτίζουν τον πίνακα C χρησιμοποιώντας την παράλληλη άθροιση. Για την παράλληλη άθροιση υπάρχει περιγραφή των βημάτων και των χρόνων πιο πάνω. Ο συνολικός χρόνος που χρειάζεται για τον υπολογισμό του καινούριου πίνακα χρησιμοποιώντας την παράλληλη άθροιση είναι **2(arraysize/p) +(g+L+1)logp+1**, όπου arraysize είναι το πλήθος των στοιχείων που πρέπει να αθροιστούν. Έτσι ο χρόνος εκτέλεσης του αλγόριθμου αυτού είναι (2(n/p) +(g+L+1)logp+1) + **((p-1)g+1+L).**

Ακολούθως παρατηρούμε ότι ο αλγόριθμος ι οποίος χρησιμοποιεί τον δεύτερο αλγόριθμο που παρουσιάσαμε, για μετάδοση δεδομένων, έχει συνολικό χρόνο εκτέλεσης (2(n/p) +(g+L+1)logp+1) + **((g+L) logp+1).** Στο πρώτο βήμα του αλγόριθμου χρειάζεται **(g+L) logp+1** χρόνο για την μετάδοση δεδομένων και υπολογισμό γινομένων (υπολογιστικό μέρος: **1**, επικοινωνιακό μέρος: **g**, συγχρονισμός: **L**). Στο δεύτερο βήμα χρησιμοποιείται η παράλληλη άθροιση γιαυτό και χρειάζεται χρόνος **2(n/p) +(g+L+1)logp+1.** Τέλος ο τρίτος αλγόριθμος χρησιμοποιεί τον τρίτο αλγόριθμο για μετάδοση δεδομένων, οπόταν στο πρώτο βήμα χρειάζεται **((κ-1)g + L) $\log_{\kappa}$p+1** χρόνο (υπολογιστικό μέρος: **1**, επικοινωνιακό μέρος: **(κ-1)g ((κ-1)-σχέση, h = κ-1)**, συγχρονισμός: **L**). Στο δεύτερο βήμα επίσης χρειαζόμαστε **2(n/p) +(g+L+1)logp+1** χρόνο διότι καλείται ο αλγόριθμος για παράλληλη άθροιση που παρουσιάσαμε πιο πάνω. Επομένως ο συνολικός χρόνος είναι (2(n/p) +(g+L+1)logp+1) + **(((κ-1)g + L) $\log_{\kappa}$p+1).**

Πιο κάτω παρουσιάζεται ο τέταρτος αλγόριθμος για πολλαπλασιασμό πινάκων.

Βήμα 1: Το στοιχείο $a_{i,k}$ μεταδίδεται σε όλους τους επεξεργαστές [i,k,*].
Το στοιχείο b_{kj} μεταδίδεται σε όλους τους επεξεργαστές [*,k,j].
Ο επεξεργαστής [i,k,j] διαβάζει τα $a_{i,k}$ και $b_{k,j}$ και υπολογίζει το γινόμενο τους.

Βήμα 2: Το κάθε $c_{i,j}$ υπολογίζεται χρησιμοποιώντας τον αλγόριθμο παράλληλης άθροισης με n επεξεργαστές.

Είναι ο ίδιος αλγόριθμος με τους προηγούμενους για πολλαπλασιασμό πινάκων , με την διαφορά ότι όταν κάποιος επεξεργαστής χρειάζεται κάποιο δεδομένο που το κρατά κάποιος

άλλος επεξεργαστής, το παίρνει **ταυτόχρονα** με τους υπόλοιπους επεξεργαστές που χρειάζονται το ίδιο στοιχείο. Και πάλι στον αλγόριθμο αυτό χρησιμοποιείται ο αλγόριθμος παράλληλης άθροισης που προαναφέραμε. Δεν χρησιμοποιούνται οι τρεις αλγόριθμοι μετάδοσης στοιχείων, που παρουσιάστηκαν πιο πάνω.

Ο χρόνος εκτέλεσης για το αλγόριθμο αυτό είναι (2(n/p) +(g+L+1)logp+1) + **(g+L+1).** Στο πρώτο βήμα του αλγόριθμου όπου γίνεται η μετάδοση δεδομένων μεταξύ των επεξεργαστών και ο υπολογισμός γινομένων χρειάζεται χρόνο (**g+L+1),** διότι ο κάθε επεξεργαστής όταν χρειάζεται κάποιο δεδομένο από την τοπική μνήμη κάποιου άλλου, το παίρνει χωρίς να περιμένει να του το στείλει (υπολογιστικό μέρος: **1**, επικοινωνιακό μέρος: **g** , συγχρονισμός: **L**). Στο δεύτερο βήμα γίνεται επίσης παράλληλη άθροιση με συνολικό χρόνο **2(n/p) +(g+L+1)logp+1**.

8.4 Παρατηρήσεις για αλγόριθμους για πολλαπλασιασμό πινάκων

1. Και οι τέσσερις αλγόριθμοι παρατηρούμε έχουν χρόνο εκτέλεσης που αυξάνεται όσο αυξάνεται το πλήθος των επεξεργαστών.
2. Οι τρεις πρώτοι αλγόριθμοι παρατηρούμε ότι έχουν τον ίδιο χρόνο με την διαφορά ότι χρησιμοποιούν διαφορετικό αλγόριθμο για την μετάδοση στοιχείων, οπότε χρειάζονται διαφορετικό χρόνο για αυτό τον σκοπό.
3. Επίσης με βάση την παραπάνω παρατήρηση αναμένουμε η διαφορά μεταξύ αυτών των τριών αλγόριθμων να είναι η ίδια που υπήρχε και μεταξύ των τριών αλγόριθμων για μετάδοση δεδομένων.
4. Ο τέταρτος αλγόριθμος περιμένουμε να είναι πιο γρήγορος λόγω του ότι δεν υπάρχει σε αυτόν μετάδοση δεδομένων. Ο κάθε αλγόριθμος όταν χρειάζεται κάποιο δεδομένο που το κρατά κάποιος άλλος επεξεργαστής μπορεί να το πάρει ακριβώς την στιγμή που θέλει, χωρίς να περιμένει κάποιους άλλους επεξεργαστές.

5. Όσο αυξάνονται οι επεξεργαστές που χρησιμοποιούνται περιμένουμε ότι θα αυξάνονται και τα μηνύματα που στέλνονται μεταξύ των επεξεργαστών. Αυτό ισχύει και για τους τέσσερις αλγόριθμους.
6. Τα μηνύματα που στέλνονται στον δεύτερο και τρίτο αλγόριθμο περιμένουμε να είναι ακριβώς τα ίδια λόγω του ότι στον τρίτο αλγόριθμο χρησιμοποιείται ο αλγόριθμος για μετάδοση δεδομένων όπου χρησιμοποιεί την σταθερά k και στην οποία έχει δώσει την τιμή 2.

8.5 Μετρήσεις

Ακολουθούν οι μετρήσεις που πάρθηκαν για τους τέσσερις αυτούς αλγόριθμους. Για τις μετρήσεις όσο αφορά τον χρόνο εκτέλεσης των αλγόριθμων παίρναμε τον μέσο όρο πέντε διαφορετικών μετρήσεων για το ίδιο στιγμιότυπο του κάθε αλγόριθμου. Οι μετρήσεις είναι περιορισμένες λόγω του ότι τα πειράματα γίνονται σε εικονικούς επεξεργαστές όπου είναι περιορισμένο το πλήθος που μπορούμε να δημιουργήσουμε σε ένα πραγματικό επεξεργαστή.

Στους πίνακες που ακολουθούν φαίνονται οι μετρήσεις για τον χρόνο που χρειάζεται ο κάθε αλγόριθμος και το πλήθος των σταλμένων μηνυμάτων, ανάλογα με το πλήθος των επεξεργαστών που χρησιμοποιούνται για την εκτέλεση του αλγόριθμου.

Αλγόριθμος 1

Πίνακας 8.1

Μέγεθος Πινάκων	**Χρόνος**	**Σταλμένα μηνύματα**
2	0,03177	3,4 ,… (4 φορές)
3	0,25329	5,6,… (9 φορές)
4	1,09833	7,8,… (16 φορές)
5	3,99249	9,10,… (25 φορές)

Αλγόριθμος 2

Πίνακας 8.2

Μέγεθος Πινάκων	**Χρόνος**	**Σταλμένα μηνύματα**
2	0,03931	1,1,2,2,... (8 φορές)
3	0,31695	2,2,3,3,.. (18 φορές)
4	1,65089	3,3,4,4,.. (32 φορές)
5	5,62496	4,4,5,5,.. (50 φορές)

Αλγόριθμος 3

Πίνακας 8.3

Επεξεργαστές	**Χρόνος**	**Σταλμένα μηνύματα**
2	0,03046	1,1,2,2,... (8 φορές)
3	0,30832	2,2,3,3,.. (18 φορές)
4	1,63812	3,3,4,4,.. (32 φορές)
5	5,62496	4,4,5,5,.. (50 φορές)

Αλγόριθμος 4

Για τον αλγόριθμο αυτό δεν υπάρχουν σταλμένα μηνύματα διότι, χρησιμοποιείται παράλληλος προθεματικός υπολογισμός.

Πίνακας 8.4

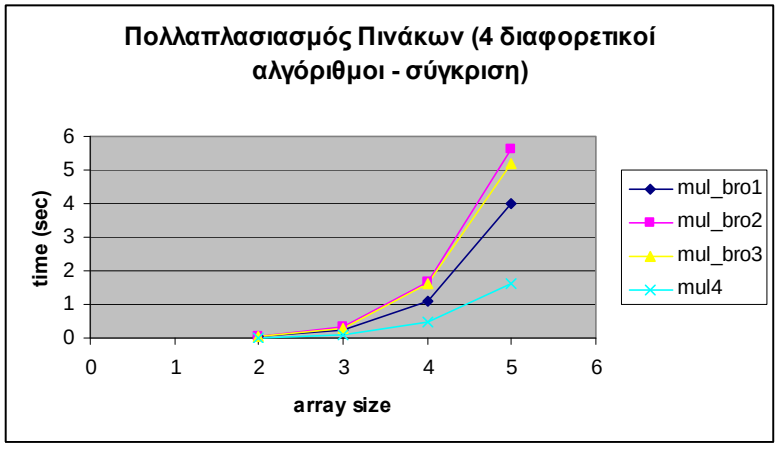
Επεξεργαστές	**Χρόνος**
2	0,01077
3	0,09623

4	0,47626
5	1,63703

8.6 Ανάλυση αποτελεσμάτων

Στο σημείο αυτό θα παρουσιαστούν οι γραφικές παραστάσεις των πιο πάνω μετρήσεων, για να μας βοηθήσουμε να καταλήξουμε σε κάποια συμπεράσματα.

Γραφική 8.1



Στην πιο πάνω γραφική παράσταση φαίνεται η σχέση του μεγέθους των πινάκων για τον πολλαπλασιασμό, με τον χρόνο που χρειάζονται οι τέσσερις αλγόριθμοι για πολλαπλασιασμό πινάκων, που παρουσιάστηκαν πιο πάνω.

Στο σημείο αυτό να σημειωθεί ότι παράλληλα βρίσκουμε και την σχέση του πλήθους των επεξεργαστών που χρησιμοποιούνται σε σχέση με τον χρόνο εκτέλεσης των αλγόριθμων, διότι οι επεξεργαστές που χρησιμοποιούνται είναι ανάλογοι με το μέγεθος των πινάκων. Συγκεκριμένα αν έχουμε πίνακα μεγέθους n χρησιμοποιούμε n^3 επεξεργαστές.

Επιπρόσθετα βλέπουμε ότι η γραφική και των τεσσάρων αλγόριθμων είναι μια καμπύλη που κατευθύνεται προς τα πάνω. Δηλαδή μπορούμε να παρατηρήσουμε ότι και στους τέσσερις αλγόριθμους υπάρχει σχέση μεταξύ του χρόνου με το μέγεθος των πινάκων και το πλήθος των επεξεργαστών. Και συγκεκριμένα παρατηρούμε ότι όσο αυξάνεται το μέγεθος των πινάκων (αυξάνεται το πλήθος επεξεργαστών αφού αν το μέγεθος των πινάκων είναι n τότε το πλήθος των επεξεργαστών είναι n^3), τόσο αυξάνεται και ο χρόνος εκτέλεσης των

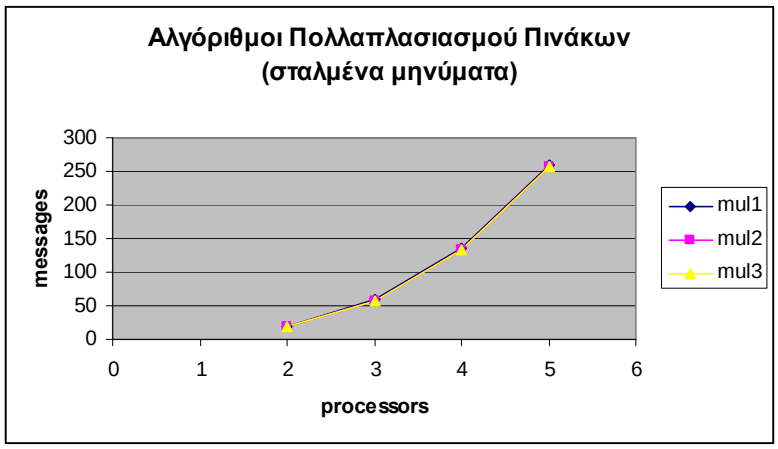
αλγόριθμων. Από τις μαθηματικές σχέσεις που παρουσιάσαμε πιο πάνω για τους χρόνους εκτελέσεις των αλγόριθμων παρατηρούμε ότι ο χρόνος είναι λογαριθμικός ως προς το πλήθος των επεξεργαστών που χρησιμοποιούνται και επομένως και του μεγέθους των πινάκων. Οι γραφικές παραστάσεις παρατηρούμε ότι δεν έχουν λογαριθμική μορφή λόγω του ότι τα πειράματα έγιναν σε εικονικούς επεξεργαστές και έτσι οι μετρήσεις δεν είναι πολύ ρεαλιστικές.

Αν συγκρίνουμε τώρα μεταξύ τους, τους χρόνους των τεσσάρων αλγόριθμων, μπορούμε να δούμε ότι ο τέταρτος αλγόριθμος για πολλαπλασιασμό πινάκων που υλοποιήθηκε, είναι ο αλγόριθμος που χρειάζεται τον λιγότερα χρόνο, διότι οι επεξεργαστές μπορούν να διαβάσουν κάποιο δεδομένο από την μνήμη κάποιου άλλου επεξεργαστή, ταυτόχρονα και με άλλους επεξεργαστές. Οπόταν δεν χρειάζεται ο ένας επεξεργαστής να περιμένει τον άλλο για να μπορέσει να συνεχίσει με την δική του δουλειά. Επίσης στον αλγόριθμο αυτό δεν υπάρχει η ανάγκη για χρήση του αλγόριθμου μετάδοσης δεδομένων. Μπορεί ο κάθε επεξεργαστής να διαβάζει από μόνος του το στοιχείο που επιθυμεί από κάποιον άλλο επεξεργαστή. Οπόταν σίγουρα χρειαζόμαστε και λιγότερο χρόνο για συγχρονισμό μεταξύ των αλγόριθμων και επομένως ο συνολικός χρόνος για τον αλγόριθμο αυτό είναι μικρότερος από τους υπόλοιπους.

Από τους άλλους αλγόριθμους παρατηρούμε ότι στον αλγόριθμο που χρησιμοποιεί τον πρώτο αλγόριθμο για μετάδοση θέλει τον λιγότερο χρόνο ενώ στους άλλους δύο θέλει σχεδόν παρόμοιο χρόνο. Αυτό οφείλεται στο γεγονός ότι η μόνη διαφορά των τριών αυτών αλγόριθμων είναι ο αλγόριθμος μετάδοσης που χρησιμοποιούν. Οπόταν και με βάση τα συμπεράσματα για τους τρεις διαφορετικούς αλγόριθμους για μετάδοση που παρουσιάστηκαν πιο πάνω, πρέπει να ισχύουν τα πιο πάνω.

Οι αλγόριθμοι που χρησιμοποιούν τον δεύτερο και τρίτο αλγόριθμο για μετάδοση δεδομένων έπρεπε να έχουν τον ίδιο χρόνο εκτέλεσης για ίδιο πλήθος επεξεργαστών και μέγεθος πινάκων, λόγω του ότι το k στο τρίτο αλγόριθμο μετάδοσης είναι ίσο με 2. Οπόταν οι δύο αλγόριθμοι μετατρέπονται στον ίδιο αλγόριθμο. Ο χρόνος τελικά δεν είναι ακριβώς ο ίδιος και στους δύο αλγόριθμους και αυτό οφείλεται στο γεγονός ότι τα πειράματα εκτελούνται σε εικονικούς επεξεργαστές, όπου έχει σαν αποτέλεσμα να μην παίρνουμε πολύ ρεαλιστικά αποτελέσματα.

Γραφική 8.2



Η γραφική παρουσιάζει την σχέση μεταξύ του μεγέθους των πινάκων που χρησιμοποιούνται για πολλαπλασιασμό και του πλήθους των σταλμένων μηνυμάτων.

Παρατηρούμε ότι είναι μια καμπύλη που κατευθύνεται προς τα πάνω. Δηλαδή όσο μεγαλώνει το μέγεθος των πινάκων που θα πολλαπλασιαστούν μεγαλώνει και το πλήθος των μηνυμάτων που στέλνονται μεταξύ των επεξεργαστών, δηλαδή μεγαλώνει η επικοινωνία. Αυτό είναι φυσικό και αναμενόμενο διότι μεγαλύτερο μέγεθος πινάκων συνεπάγεται περισσότερα δεδομένα, και μεγαλύτερο πλήθος επεξεργαστών, όπου για να επικοινωνήσουν χρειάζονται να στείλουν περισσότερα μηνύματα.

Το συνολικό πλήθος μηνυμάτων που στέλνονται μεταξύ των επεξεργαστών είναι το ίδιο και στους τρεις αλγόριθμους, επειδή έχουμε το ίδιο πλήθος επεξεργαστών και μεγέθους πινάκων.

Αν παρατηρήσουμε και τους πίνακες μετρήσεων αυτών των αλγόριθμων, που παρουσιάζονται πιο πάνω θα παρατηρήσουμε ότι στους δύο που χρησιμοποιούν τον δεύτερο και τρίτο αλγόριθμο μετάδοσης τα μηνύματα στέλνονται ακριβώς με τον ίδιο τρόπο. Ο λόγος είναι και πάλι το k=2 στον αλγόριθμο που χρησιμοποιεί των τρίτο αλγόριθμο για μετάδοση δεδομένων.

8.6 Συμπεράσματα

Με τις μετρήσεις και την δημιουργία γραφικών παραστάσεων καταφέραμε να φτάσουμε σε κάποια συμπεράσματα όσο αφορά τους αλγόριθμους αυτούς, τα οποία παρουσιάζονται αμέσως πιο κάτω.

1. Όντως παρατηρούμε ότι και στους τέσσερις αλγόριθμους ο χρόνος αυξάνεται όσο αυξάνεται το πλήθος των επεξεργαστών, γιατί γίνονται περισσότερα υπερβήματα για την ολοκλήρωση του αλγόριθμου και χρειάζεται περισσότερος χρόνος για επικοινωνία και συγχρονισμό μεταξύ των αλγόριθμων.
2. Επίσης παρατηρούμε ότι ο χρόνος εκτέλεσης που χρειάζεται ο τέταρτος αλγόριθμος είναι πολύ πιο μικρός από τους χρόνους των υπόλοιπων αλγόριθμων, λόγω του ότι δεν υπάρχει χρήση αλγόριθμου για μετάδοση δεδομένων, ούτε χρειάζεται κάποιος επεξεργαστής να περιμένει κάποιους άλλους για να πάρει δεδομένα.
3. Οι χρόνοι για τον δεύτερο και τρίτο αλγόριθμο είναι περίπου οι ίδιοι λόγω του ότι στον τρίτο που χρησιμοποιεί τον τρίτο αλγόριθμο μετάδοσης, το k είναι ίσο με 2 και επομένως μετατρέπεται στον δεύτερο αλγόριθμο μετάδοσης. Έτσι η διαφορά μεταξύ των δύο αλγόριθμο για πολλαπλασιασμό χάνεται, και στην ουσία πλέον οι δύο αλγόριθμοι υλοποιούν ακριβώς το ίδιο πράγμα. (Γενικά θα μπορούσαμε να πούμε ότι ο τρίτος αλγόριθμος για μετάδοση δεδομένων είναι μια γενική περίπτωση όπου μπορεί με k=2 στην συνάρτηση για μετάδοση δεδομένων, να μετατραπεί στο δεύτερο αλγόριθμο και αντίστοιχα, αν το k ισούται με το πλήθος των επεξεργαστών να μετατραπεί στον πρώτο αλγόριθμο για μετάδοση δεδομένων.)
4. Τα μηνύματα που στέλνονται στον δεύτερο και τρίτο αλγόριθμο είναι ακριβώς τα ίδια για το λόγω που προαναφέραμε πιο πάνω (k=2).
5. Τα μηνύματα που στέλνονται στον πρώτο αλγόριθμο είναι περισσότερα στο πρώτο υπερβήμα, διότι χρησιμοποιείται ο πρώτος αλγόριθμος μετάδοσης στον οποίο στο πρώτο υπερβήμα στέλνονται όλα τα δεδομένα από έναν επεξεργαστή, προς όλους τους υπόλοιπους που πρέπει να παραλάβουν τα δεδομένα.

6. Λόγω του ότι στον πρώτο αλγόριθμο χρησιμοποιείται ο πρώτος αλγόριθμος μετάδοσης παρατηρούμε ότι ο χρόνος του είναι μικρότερος από ότι τον χρόνο του δεύτερου και τρίτου αλγόριθμου.

ΚΕΦΑΛΑΙΟ 9

Συμπεράσματα

Με το Κεφάλαιο αυτό ολοκληρώνεται η ανάλυση της διπλωματικές μου εργασίας, παραθέτοντας κάποια τελικά συμπεράσματα. Στο Υποκεφάλαιο 9.1 αναφέρονται κάποιοι περιορισμοί που υπήρχαν κατά την διάρκεια ολοκλήρωσης της εργασίας, οι οποίοι περιόρισαν κάποια σημεία της εργασίας ή δεν επέτρεψαν να πραγματοποιηθούν κάποια άλλα. Ακολούθως στο Υποκεφάλαιο 9.2 αναφέρονται κάποια γενικά συμπεράσματα που προέρχονται από την υλοποίηση και ανάλυση των αλγόριθμων που αναφέρθηκαν σε προηγούμενα κεφάλαια. Στο Υποκεφάλαιο 9.3 καθορίζονται κάποιοι μελλοντικοί στόχοι, δηλαδή κάποια μελλοντική εργασία που θα ήταν καλό να γίνει και που είναι σχετική με την ερευνητική μας εργασία. Τέλος στο Υποκεφάλαιο 9.4 καταγράφονται οι εμπειρίες που αποκτήθηκαν κατά την εκπόνηση της ερευνητικής εργασίας.

9.1 Περιορισμοί αδυναμίες και ενδεχόμενες παραδοχές

Στο τελευταίο στάδιο της ερευνητικής εργασίας έγινε πειραματική αξιολόγηση των αλγόριθμων που υλοποιήθηκαν σε προηγούμενο στάδιο, με την δημιουργία μετρήσεων και γραφικών και την εξαγωγή συμπερασμάτων. Αυτή η πειραματική αξιολόγηση έγινε για να μελετήσουμε την πρακτικότητα των αλγόριθμων. Δηλαδή για να ελέγξουμε διάφορες παραμέτρους τους, σε διαφορετικές συνθήκες εκτέλεσης. Παραδείγματος χάριν σε περίπτωση μεγαλύτερης εισόδου, πως θα άλλαζε ο χρόνος εκτέλεσης του αλγόριθμου. Ακόμη βασική ήταν και η σύγκριση παρόμοιων αλγόριθμων που είχαν όμως κάποιες μικρές διαφορές. Οι διάφοροι αλγόριθμοι εκτελέστηκαν σε ένα υπολογιστή-επεξεργαστή του Πανεπιστημίου Κύπρου, χρησιμοποιώντας βέβαια ένα πλήθος εικονικών επεξεργαστών που δημιουργούνταν πάνω στον πραγματικό επεξεργαστή. Σίγουρα πολύ πιο ρεαλιστικά αποτελέσματα θα μπορούσαμε να πάρουμε εάν είχαμε υλοποιήσει τον κάθε αλγόριθμο, παράλληλα σε πολλούς επεξεργαστές-υπολογιστές. Δυστυχώς υπήρξε αυτός ο περιορισμός λόγω του ότι αντιμετωπίσαμε πρόβλημα στην εγκατάσταση της έκδοσης της βιβλιοθήκης PUB για παράλληλη επεξεργασία σε πολλούς επεξεργαστές (tspip communication layer and release mode). Έγινε προσπάθεια επικοινωνίας με την ομάδα προγραμματιστών για την βιβλιοθήκη PUB, ώστε να μας βοηθήσει σε αυτή την δυσκολία, όμως δυστυχώς δεν υπήρξε ανταπόκριση, οπόταν συνεχίστηκε η δουλεία σε εικονικούς επεξεργαστές.

Σε περίπτωση που θα εκτελούσαμε τους αλγόριθμους σε πολλούς πραγματικούς επεξεργαστές, σίγουρα θα παίρναμε πιο ρεαλιστικά αποτελέσματα και θα αποφεύγαμε κάποιους περιορισμούς που οφείλονται στην χρήση εικονικών επεξεργαστών. Για παράδειγμα παρατηρούμε ότι δεν μπορούσαμε να δημιουργήσουμε περισσότερους από 199 εικονικούς επεξεργαστές για την εκτέλεση όλων των αλγόριθμων μας. Γεγονός που πολλές φορές δεν μας επέτρεπε να βγάλουμε πολύ σωστά συμπεράσματα, λόγω του ότι περιορίζονταν οι μετρήσεις που μπορούσαν να γίνονται.

Επιπλέον λόγω του πιο πάνω περιορισμού σε κάποιον από τους αλγόριθμους που υλοποιήσαμε, αναγκαστήκαμε να υποθέσουμε ότι ο λόγος που δεν πήραμε τα αναμενόμενα αποτελέσματα ήταν λόγω του ότι μπορούσαμε να πάρουμε μετρήσεις μόνο για μικρές τιμές. Άρα συμπεραίνουμε ότι τα πειραματικά αποτελέσματα, αν γίνονταν κάτω από ρεαλιστικές συνθήκες και μεγάλο αριθμό επεξεργαστών, ίσως μας αποκάλυπταν πιο ενδιαφέροντα αποτελέσματα που ίσως έδειχναν καλύτερα την πρακτικότητα των αλγόριθμων.

9.2 Συμπεράσματα από την εκτέλεση των αλγόριθμων

Για αυτήν την ερευνητική εργασία υλοποιήθηκαν μερικοί παράλληλοι αλγόριθμοι σύμφωνα με το μοντέλο παράλληλης επεξεργασίας BSP και χρησιμοποιώντας την βιβλιοθήκη PUB. Υλοποιήθηκαν τρεις διαφορετικοί αλγόριθμοι για μετάδοση δεδομένων μεταξύ των επεξεργαστών (broadcast1,broadcast2,broadcast3). Ακολούθως υλοποιήθηκε ο βέλτιστος αλγόριθμος για παράλληλη άθροιση και ο βέλτιστος αλγόριθμος για παράλληλο προθεματικό υπολογισμό. Επίσης υλοποιήθηκαν τέσσερις παραλλαγές αλγόριθμων για τον μη βέλτιστο αλγόριθμο για πολλαπλασιασμό πινάκων. Τέλος υλοποιήθηκαν δύο αλγόριθμοι για παράλληλη άθροιση και προθεματικό υπολογισμό σε τετραγωνικό δίχτυο αλληλοσύνδεσης (mesh).

Στην συνέχεια με την εκτέλεση των αλγόριθμων πάρθηκαν διάφορες μετρήσεις και δημιουργήθηκαν γραφικές παραστάσεις με τα δεδομένα των μετρήσεων, οι οποίες μας βοήθησαν να βγάλουμε πιο εύκολα συμπεράσματα για τις διάφορες παραμέτρους που αφορούν τους αλγόριθμους μας.

Γενικά παρατηρήθηκε ότι τα συμπεράσματα που εξάγονταν από τις μετρήσεις μας συμφωνούσαν αρκετά με την θεωρία. Σε όλους τους αλγόριθμους, όσο αυξανόταν το πλήθος των επεξεργαστών αυξανόταν και ο χρόνος εκτέλεσης των αλγόριθμων , όπως επίσης και το πλήθος των σταλμένων μηνυμάτων. Και τα δύο επακόλουθα της αύξησης του πλήθους των επεξεργαστών, είναι αποτέλεσμα της αύξησης της επικοινωνίας. Γενικά σε κάθε αλγόριθμο που δημιουργείται με βάση το μοντέλο αυτό υπάρχει άμεση συσχέτιση μεταξύ του πλήθους των επεξεργαστών με το χρόνο εκτέλεσης και το πλήθος των σταλμένων μηνυμάτων. Μια άλλη γενική παρατήρηση από τα αποτελέσματα ήταν ότι το

μέγεθος των μηνυμάτων που στέλνονται, σε αλγόριθμους μετάδοσης επηρεάζει τον χρόνο εκτέλεσης των αλγόριθμων.

Η υλοποίηση αλγόριθμων στο μοντέλο αυτό έχει μια πολύ απλή δομή, οπόταν είναι δυνατόν να γράφονται σε αυτό πολύ εύκολα κώδικες. Συγκεκριμένα αποτελείται από υπερβήματα, που το κάθε ένα τελειώνει με ένα φραγμένο συγχρονισμό.

Επίσης η απόδοση του μοντέλου είναι προβλέψιμη, λόγω του ότι υπάρχει κάποιος μαθηματικός τύπος ($\mathbf{w_t + g.\ h_t + L}$) με τον οποίο μπορούμε να βρούμε στο περίπου τον χρόνο και το κόστος των αλγόριθμων. Έπίσης λόγω του ότι υπάρχουν κάποιοι παράμετροι του μοντέλου που μπορούμε να τις υπολογίσουμε εύκολα. Οπόταν με τα πιο πάνω στοιχεία μπορούμε εύκολα να γνωρίζουμε από την αρχή την απόδοση των αλγόριθμων μας. Όντως από τις μετρήσεις οι χρόνοι συμφωνούσαν αρκετά με αυτό που αναμέναμε από τον τύπο αυτό πριν την εκτέλεση του αλγόριθμου.

Ακολούθως παρουσιάζονται τα συμπεράσματα από την πειραματική αξιολόγηση των υλοποιημένων αλγόριθμων.

- Αρχικά μελετώντας τους **τρεις αλγόριθμους για μετάδοση δεδομένων** συμπεράναμε ότι όσο περισσότερη είναι η επικοινωνία μεταξύ των επεξεργαστών τόσος περισσότερος είναι ο χρόνος που χρειάζεται και τόσα περισσότερα είναι τα μηνύματα που χρειάζεται να σταλούν. Οπόταν αύξηση του πλήθους των επεξεργαστών οδηγεί στην ανάγκη για μεγαλύτερη επικοινωνία άρα σε μεγαλύτερο χρόνο εκτέλεσης και μεγαλύτερο πλήθος μηνυμάτων.
- Το συνολικό πλήθος των μηνυμάτων που στέλνονται είναι το ίδιο σε όλους τους αλγόριθμους και είναι ίσο με το *πλήθος των επεξεργαστών -1*, αφού ο πρώτος επεξεργαστής είναι ο μόνος που γνωρίζει από την αρχή το μήνυμα που πρέπει να σταλεί. Συνεπώς απομένουν ακόμη *πλήθος των επεξεργαστών -1* επεξεργαστές να μάθουν το δεδομένο. Κάθε επεξεργαστής για να μάθει το δεδομένο πρέπει να παραλάβει ένα μήνυμα, από ένα άλλο επεξεργαστή. Επομένως συνολικά θα σταλούν τόσα μηνύματα όσο και το πλήθος των επεξεργαστών που δεν γνωρίζουν το μήνυμα και αυτό θα συμβεί και στους τρεις αλγόριθμους.
- Τα μηνύματα όμως στέλλονται με διαφορετικό τρόπο στον κάθε αλγόριθμο, οπόταν έχουμε διαφορετικό πλήθος μηνυμάτων στα διάφορα υπερβήματα των αλγόριθμων. Στον πρώτο αλγόριθμο που παρουσιάσαμε έχουμε ένα υπερβήμα στο

οποίο στέλλονται nprocs-1 μηνύματα (h=nprocs-1). Στους υπόλοιπους δύο στέλλονται πιο λίγα μηνύματα στο κάθε υπερβήμα. Συγκεκριμένα στέλλεται ένα μήνυμα από κάθε επεξεργαστή που γνωρίζει το δεδομένο, προς κάποιο άλλο που δεν το γνωρίζει 1-σχέση.

- Συμπεράναμε ακόμη ότι ο τρίτος αλγόριθμος για μετάδοση δεδομένων ανάλογα με την τιμή που δίνουμε στο k, μπορεί να μεταδίδει με διαφορετικό τρόπο τα μηνύματα και κάθε φορά να χρειάζεται διαφορετικό χρόνο για τον σκοπό αυτό. Όταν το k είναι ίσο με δύο τότε εκτελείται ακριβώς με τον ίδιο τρόπο που εκτελείται ο δεύτερος αλγόριθμος που παρουσιάσαμε, διότι έχουμε μια 1-σχέση όπως και στο αλγόριθμο αυτό. Όταν το k είναι ίσο με το πλήθος των επεξεργαστών τότε έχουμε μια nprocs-1 σχέση, οπόταν μετατρέπεται στον πρώτο αλγόριθμο που παρουσιάσαμε για μετάδοση δεδομένων και τερματίζει στο ένα υπερβήμα. Οι μετρήσεις που κάναμε για διαφορετικά k μας έδειξαν ότι όσο μεγαλώνει το k ο χρόνος εκτέλεσης μειώνεται. Αυτό είναι λογικό διότι στέλνονται περισσότερα μηνύματα σε ένα υπερβήμα και επομένως γίνεται η μετάδοση μέσα σε πιο λίγα υπερβήματα και επομένως πιο λίγο χρόνο.
- Ακόμη πρέπει να σημειωθεί ότι καταφέραμε να καταλήξουμε σε συμπεράσματα όσο αφορά την σχέση μεταξύ του μεγέθους των μηνυμάτων που μεταδίδονται και του χρόνου εκτέλεσης του αλγόριθμου. Συγκεκριμένα αυτό που ισχυεί είναι ότι το μοντέλο BSP δεν κάνει διάκριση μεταξύ του κόστους για μια διαδικασία που αποστέλλει h μηνύματα με μέγεθος 1 και μια διαδικασία που αποστέλλει ένα μήνυμα μεγέθους h. Και οι δύο επικοινωνίες είναι μια h-σχέση με κόστος h*g [7]. Από εδώ μπορούμε να καταλάβουμε ότι όταν στέλουμε ίδιο πλήθος μηνυμάτων σε δύο διαδικασίες, αλλά στην μια από τις δύο στέλουμε μεγαλύτερου μεγέθους μηνύματα είναι φυσικό ότι θα χρειάζεται περισσότερος χρόνος για την εκτέλεση της διαδικασίας. Μάλιστα όσο μεγαλύτερη είναι η διαφορά του μεγέθους των μηνυμάτων, τόσο περισσότερο θα μεγαλώνει η διαφορά στον χρόνο εκτέλεσης των δύο διαδικασιών. Με λίγα λόγια όσο μεγαλύτερο είναι το μέγεθος των μηνυμάτων που ανταλλάσονται μεταξύ των επεξεργαστών, τόσο μεγαλύτερος είναι και ο χρόνος εκτέλεσης του αλγόριθμου.

- Ακόμη με την συνέχιση των πειραμάτων μας ελέξαμε την σχέση μετάξυ του πλήθους των μηνυμάτων που αποστέλλονται και και του χρόνου εκτέλεσης του αλγόριθμου. Καταλήξαμε στο συμπέρασμα ότι όσα περισσότερα μηνύματα στέλλονται τόσος περισσότερος χρόνος χρειάζεται. Αυτό είναι φυσικό διότι παραδείγματως χάρην στην περίπτωση μας με την αποστολή ένα προς ένα των στοιχειών ενώς πίνακα, υπάρχει μεγαλύτερη καθυστέρηση αφού σειριακά ένας επεξεργαστής στέλλει περισσότερα από ένα μηνύματα σε κάποιον άλλο επεξεργαστή.
- Ακολούθως υλοποιώντας των αλγόριθμο του **προθεματικού αθροίσματος** καταλήξαμε στο συμπέρασμα ότι επίσης όταν χρησιμοποιούσαμε μεγαλύτερο πλήθος επεξεργαστών και στοιχείων, ο χρόνος εκτέλεσης του αλγόριθμου αυξανόταν όπως επίσης και τα σταλμένα μηνύματα. Αυτό οφείλεται στο γεγονός ότι μεγαλώνει η επικοινωνία που πρέπει να γίνεται και επομένως αυξάνονται τα μηνύματα που στέλλονται, και ο χρόνος για συγχρονισμό μεταξύ των επεξεργαστών.
- Μελετώντας τον αλγόριθμο **παράλληλης άθροισης** παρατηρήσαμε ότι ισχύει η πιο πάνω παρατήρηση που αναφέραμε και στον προθεματικό υπολογισμό. Ο χρόνος εκτέλεσης του αλγόριθμου είναι λογαριθμικός ως προς το πλήθος των επεξεργαστών. Όσοι περισσότεροι είναι οι επεξεργαστές χρειάζεται να γίνουν περισσότερα υπερβήματα και χρειάζεται περισσότερος χρόνος για συγχρονισμό των επεξεργαστών. Οπόταν συνολικά ο χρόνος εκτέλεσης είναι μεγαλύτερος.
- Όσο περισσότεροι είναι οι επεξεργαστές τόσο περισσότερα είναι και τα μηνύματα που στέλνονται κατά την διάρκεια της άθροισης διότι πρέπει να επικοινωνήσουν περισσότεροι επεξεργαστές μεταξύ τους για να γίνει η άθροιση.
- Όσο περισσότερα είναι τα στοιχεία που θα αθροιστούν τόσο μεγαλώνει και ο χρόνος που χρειαζόμαστε για την άθροιση. Αυτό οφείλεται στο ότι οι επεξεργαστές πρέπει να κάνουν περισσότερη δουλειά, διότι είναι περισσότερα τα στοιχεία που πρέπει να αθροιστούν, συνεπώς δικαιολογημένα χρειάζεται περισσότερος χρόνος.
- Με την εκτέλεση και πειραματική αξιολόγηση στους **δύο αλγόριθμους για παράλληλη άθροιση και προθεματικό υπολογισμό σε τετραγωνικό πλέγμα** οδηγηθήκαμε στα πιο κάτω συμπεράσματα. Όσο το πλήθος των επεξεργαστών

μεγαλώνει (n), αυξάνεται και ο χρόνος που χρειάζονται οι δύο αλγόριθμοι. Αυτό συμβαίνει διότι το πλήθος των επεξεργαστών σχετίζεται με τον χρόνο εκτέλεσης του αλγόριθμου. Συγκεκριμένα ο χρόνος είναι ανάλογος προς το √p (τετραγωνική ρίζα των επεξεργαστών).

- Ο χρόνος για την άθροιση είναι μικρότερος από τον χρόνο που χρειάζεται ο αλγόριθμος για την προθεματική άθροιση, λόγω του ότι στον αλγόριθμο για προθεματική άθροιση γίνεται επιπλέον δουλειά για την ολοκλήρωση του. Τα βήματα είναι αρχικά τα ίδια και στους δύο αλγόριθμους. Από ένα σημείο και μετά όμως ο αλγόριθμος της παράλληλης άθροισης ολοκληρώνεται ενώ στον αλγόριθμο προθεματικής άθροισης γίνονται επιπλέον βήματα.
- Όσο αφορά τα μηνύματα που στέλνονται κατά την προθεματική άθροιση μεταξύ των επεξεργαστών, παρατηρήσαμε ότι μέχρι κάποιο σημείο είναι ακριβώς τα ίδια με αυτά στη παράλληλη άθροιση διότι η προθεματική άθροιση είναι η παράλληλη άθροιση με κάποια επιπρόσθετα βήματα.
- Τέλος με την υλοποίηση **τεσσάρων διαφορετικών αλγόριθμων για πολλαπλασιασμό πινάκων** παρατηρήσαμε ότι και στους τέσσερις ο χρόνος αυξάνεται όσο αυξάνεται το πλήθος των επεξεργαστών, γιατί γίνονται περισσότερα υπερβήματα για την ολοκλήρωση του αλγόριθμου και χρειάζεται περισσότερος χρόνος για επικοινωνία και συγχρονισμό μεταξύ των αλγόριθμων.
- Επίσης παρατηρούμε ότι ο χρόνος εκτέλεσης που χρειάζεται ο τέταρτος αλγόριθμος είναι πολύ πιο μικρός από τους χρόνους των υπόλοιπων αλγόριθμων, λόγω του ότι δεν υπάρχει χρήση αλγόριθμου για μετάδοση δεδομένων, ούτε χρειάζεται κάποιος επεξεργαστής να περιμένει κάποιους άλλους για να πάρει δεδομένα.
- Οι χρόνοι για τον δεύτερο και τρίτο αλγόριθμο είναι περίπου οι ίδιοι λόγω του ότι στον τρίτο που χρησιμοποιεί τον τρίτο αλγόριθμο μετάδοσης, το k είναι ίσο με 2 και επομένως μετατρέπεται στον δεύτερο αλγόριθμο μετάδοσης. Έτσι η διαφορά μεταξύ των δύο αλγόριθμο για πολλαπλασιασμό χάνεται, και στην ουσία πλέον οι δύο αλγόριθμοι υλοποιούν ακριβώς το ίδιο πράγμα. Λόγω του ότι στον πρώτο αλγόριθμο χρησιμοποιείται ο πρώτος αλγόριθμος μετάδοσης παρατηρούμε ότι ο

χρόνος του είναι μικρότερος από ότι τον χρόνο του δεύτερου και τρίτου αλγόριθμου.

- Τα μηνύματα που στέλνονται στον δεύτερο και τρίτο αλγόριθμο είναι ακριβώς τα ίδια για το λόγω που προαναφέραμε πιο πάνω (k=2). Τα μηνύματα που στέλνονται στον πρώτο αλγόριθμο είναι περισσότερα στο πρώτο υπερβήμα που εκτελείται κατά την κλήση της συνάρτησης για μετάδοση δεδομένων, διότι χρησιμοποιείται ο πρώτος αλγόριθμος μετάδοσης στον οποίο στο πρώτο υπερβήμα στέλνονται όλα τα δεδομένα από έναν επεξεργαστή, προς όλους τους υπόλοιπους που πρέπει να παραλάβουν τα δεδομένα.

9.3 Μελλοντική εργασία

Ένας πολύ σημαντικός στόχος που δεν καταφέραμε να υλοποιήσουμε, λόγω του προβλήματος που προαναφέραμε, ήταν να εκτελεστούν οι αλγόριθμοι σε πολλούς πραγματικούς επεξεργαστές, γεγονός που θα βοηθούσε στην εξαγωγή καλύτερων και πιο ρεαλιστικών συμπερασμάτων. Οπόταν θα ήταν καλό να λυθεί το πρόβλημα με την εγκατάσταση της έκδοσης της βιβλιοθήκης PUB για παράλληλη επεξεργασία σε πολλούς επεξεργαστές (tspip communication layer and release mode), ώστε να εκτελεστούν οι αλγόριθμοι σε πραγματικούς επεξεργαστές και να παρθούν καλύτερες και περισσότερες μετρήσεις για εξαγωγή πιο ρεαλιστικών συμπερασμάτων. Συνεπώς με την αντιμετώπιση της πιο πάνω δυσκολίας θα εκτελεστούν περισσότερα σενάρια και κάποια πράγματα που υποθέσαμε θα μπορέσουμε να τα αποδείξουμε.

Για παράδειγμα στην περίπτωση με τον αλγόριθμο για παράλληλη προθεματική άθροιση βρήκαμε ότι ο χρόνος εκτέλεσης του βέλτιστου αλγόριθμου ήταν μικρότερος από τον χρόνο εκτέλεσης του μη βέλτιστου, γεγονός που αντιφάσκει με αυτό που αναμέναμε. Αυτό υποθέσαμε ότι ισχύει λόγω του ότι με τους περιορισμούς που υπάρχουν στην χρήση εικονικών επεξεργαστών μπορέσαμε να πάρουμε μόνο μια μέτρηση για πολύ μικρές τιμές επεξεργαστών και στοιχείων. Με την χρήση πραγματικών επεξεργαστών δεν θα υπάρχει αυτός ο περιορισμός και θα μπορέσουμε να οδηγηθούμε σε καλύτερα συμπεράσματα.

Επιπλέον λόγω της χρήσης εικονικών επεξεργαστών δεν ολοκληρώθηκε η προσπάθεια για δημιουργία του βέλτιστου παράλληλου αλγόριθμου, για ταξινόμηση (merge sort) [**2**], λόγω του ότι υπήρχε μεγάλη δυσκολία στην υλοποίηση αναδρομικών κλήσεων χρησιμοποιώντας εικονικούς επεξεργαστές. Οπόταν με την χρήση πραγματικών επεξεργαστών ένας καλός μελλοντικός στόχος είναι η υλοποίηση αυτού του αλγόριθμου και η εξαγωγή συμπερασμάτων.

9.4 Εμπειρίες που αποκτήθηκαν

Στην προσπάθεια ολοκλήρωσης της διπλωματικής μου εργασίας εξοικειώθηκα με την ερευνητική μεθοδολογία η οποία χαρακτηρίζεται από κάποια βασικά βήματα τα οποία παρουσιάζονται παρακάτω: Αρχικά πρέπει να γίνει διερεύνηση και μελέτη χρησιμοποιώντας διάφορα άρθρα και δημοσιεύσεις που θα οδηγήσουν στην πλήρη κατανόηση του θέματος. Ακολούθως γίνεται ο πειραματισμός και η εξοικείωση με εργαλεία ή με γλώσσα προγραμματισμού που θα χρησιμοποιηθούν και στην συνέχεια η υλοποίηση αλγόριθμων χρησιμοποιώντας αυτά. Αργότερα γίνεται η πειραματική αξιολόγηση κατά την οποία γίνονται μετρήσεις με την εκτέλεση των αλγόριθμων , και ακολούθως δημιουργούνται κατάλληλες γραφικές χρησιμοποιώντας αυτές τις μετρήσεις. Τέλος έχουμε την εξαγωγή διαφόρων συμπερασμάτων.

Ακολουθώντας την πιο πάνω μεθοδολογία κατάφερα να αποκομίσω πολύ χρήσιμες γνώσεις και εμπειρίες γύρω από διάφορους τομείς της πληροφορικής με τους οποίους δεν είχα την δυνατότητα προηγουμένως να εμπλακώ.

Με την μελέτη διαφόρων άρθρων και δημοσιεύσεων κατάφερα να κατανοήσω καλύτερα το μοντέλο BSP αλλά και γενικότερα εξοικειώθηκα περισσότερο με τους παράλληλους αλγόριθμους. Με αποτέλεσμα να καταλάβω την πραγματική χρησιμότητά τους και την αναγκαιότητα τους στην υλοποίηση των διάφορων συστημάτων στις μέρες μας.

Επιπρόσθετα το μοντέλο BSP (**Bulk-Synchronous Parallel model)**, το οποίο μελέτησα σε βάθος, είναι πραγματικά ένα πολύ σπουδαίο μοντέλο παράλληλου υπολογισμού, το οποίο είναι ένα υποσχόμενο υποψήφιο ενοποιητικό μοντέλο για παράλληλο υπολογισμό, που

μπορεί να αντιμετωπίσει την δυσκολία στην δημιουργία ενός αποδοτικού και φορητού (portable) παράλληλου λογισμικού.

Ακόμη αλγόριθμους τους οποίους είχα μελετήσει θεωρητικά, τους υλοποίησα και έτσι τους μελέτησα σε βάθος, με αποτέλεσμα να τους κατανοήσω καλύτερα και να μπορέσω να ελέγξω και πρακτικά την συμπεριφορά τους στις διάφορες συνθήκες εκτέλεσης τους. Επίσης μέσα από την υλοποίηση των αλγόριθμων και την πειραματική αξιολόγηση κατάφερα να ελέγξω την σχέση μεταξύ της θεωρίας και της πραγματικότητας, καταλήγοντας ότι αυτά που διατυπώνονται θεωρητικά, ισχύουν σε γενικές γραμμές και πρακτικά.

Ένα άλλο σημαντικό στοιχείο που κατάφερα να κερδίσω ήταν ότι κατάφερα να βελτιώσω τις προγραμματιστικές μου ικανότητες με την υλοποίηση των αλγόριθμων. Επίσης έμαθα την γλώσσα προγραμματισμού PuBsp και τον τρόπο γραφής παράλληλων αλγόριθμων σύμφωνα με το μοντέλο BSP.

Μέσα στα πλαίσια ολοκλήρωσης της ερευνητικής μου εργασίας, ίσως η σημαντικότερη εμπειρία που απόκτησα ήταν η εμπλοκή μου στην διαδικασία και μεθοδολογία της έρευνας και η συνεργασία μου με έμπειρο ερευνητή του Πανεπιστημίου Κύπρου, το Δρ. Χρύση Γεωργίου.

Κλείνοντας, πιστεύω πως οι εμπειρίες που απόκτησα κατά την διάρκεια εκπόνησης αυτής της ερευνητικής εργασίας, θα μου φανούν πολύ χρήσιμες στην μελλοντική μου επαγγελματική σταδιοδρομία.

Βιβλιογραφία

1. Olaf Bonorden**,** Miroslaw Dynia, Joachim Gehweiler, Rolf Wanka , Pub-Library, User Guide and Function Reference, http://wwwcs.uni-paderborn.de/~bsp/docu/pub8/index.html, Release 8.1-pre, July 2003.

2. Χρύσης Γεωργίου , Σημειώσεις Μαθήματος ΕΠΛ 431 – Σύνθεση Παράλληλων Αλγόριθμων, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2008, http://www2.cs.ucy.ac.cy/~chryssis/EPL431/

3. A.V.Gerbessiotis , Άρθρο CIS 668,10-16-2000, A Brief Introduction to the BSP Model, http://web.njit.edu/~alexg/courses/cis668/Fall2000/handsub5.pdf , Fall 2000 Handout 10, BSP Model Intro.

4. JaJa Joseph, An Introduction to Parallel Algorithms, Addison-Wesley Publishing Company, Reprinted November 1992

5. Amit Petkar, Read Integers from a Text File Using a C-program, http://www.nabble.com/read-integers-from-a-text-file-using-a-c-program-td9491187.html, March 20,2007.

6. Michael C. Scherger, BSP Programming Style: An Overview of the BSP Model of Parallel Computation, Department of Computer Science, Kent State University, http://www.cs.kent.edu/~jbaker/PDA-Sp07/slides/bsp%20model.ppt#272.

7. D.B. Skillicorn, Jonathan M.D. Hill, W.F. McColl, Programming Research Group, Questions And Answers About BSP, Revised 11th November 1996.

8. Leslie G. Valiant, A Bridging Model for Parallel Computation, Communications of the ACM ,August 1990, Vol.33, No.8.

9. BSP Worldwide, http://www.bsp-worldwide.org/

10. Installation of Pub Library , http://wwwcs.uni-paderborn.de/~bsp/docu/pub8/node4.html

11. Paderborn University BSP-Library, http://wwwcs.uni-paderborn.de/~bsp/

12. Παράλληλος Υπολογισμός, http://www.webopedia.com/TERM/P/parallel_processing.html

13. The BSP Model, http://wwwcs.uni-paderborn.de/fachbereich/AG/agmadh/WWW/bono/paper/nestedbsp/node6.html

Παράρτημα Α

Παρουσίαση κώδικων υλοποιημένων αλγορίθμων [5]

Α.1 Αλγόριθμος για μετάδοση δεδομένων 1

```
/* ========================================================== */
/*                                                            */
/*   broadcast1.c                                             */
/* Margarita Antonaki                                         */
/*                                                            */
/* ========================================================== */


/*Algorithoms1: Enas epeksergastis, esto o P1 steli to minima/stixio      */
/*se olous tous ipoloipous epeksergastes(xronos (p-1)g+L)                 */
//93 lines

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
double  duration,time1,time2;
FILE *f1;


void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

doTests( bsp, argc, argv );
t_bspmsg *msg;
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int i,*data1;
int data=pid, cmsg=0;

//ANOIGMA ARXEIOU

f1 = fopen ("out.lis", "wt");
```

```
time1=bsp_time(bsp);

//O EPEKSERGASTIS ME PID=0 STELLEI MINIMA SE OLOUS TOUS IPOLOIPOUS

if (pid==0)
  {
   bsp_gsend(bsp, 1, nprocs-1, & data, sizeof(int));
   cmsg= bsp_nprocs(bsp)-1;
  }
   bsp_sync(bsp);


//O KATHE EPEKSERGASTIS POU EXEI PARALAVEI MINIMA TO APOTHIKEVEI
//STIN DIKI TOU TOPIKI MNIMI

for (i=0; i<bsp_nmsgs (bsp); i++) {
  msg = bsp_getmsg (bsp, i);
  data1=(int *) bspmsg_data (msg);
  fprintf(f1,"Epeksergastis %d data=%d\n", pid,*data1);
  }

bsp_sync(bsp);

time2=bsp_time(bsp);

if(pid==0)
  fprintf(f1,"Stalthikan %d minimata sto ipervima!!\n",cmsg);

bsp_threadsafe_done( bsp );
}



int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   int numofpro;

   printf("Dose ton arithmo ton epeksergaston: _");
   scanf("%d",&numofpro);
   printf("\n");

   argc = _argc; argv = _argv;
   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", numofpro);
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );
```

```
    bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );
    fprintf(f1, "\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf defterolepta\n",time2-time1
);
    fclose (f1);

 bsplib_done();
    free( bsp );
#ifdef MEMLOG
    checkMemLog();
#endif
    return 0;
}
```

Α.2 Αλγόριθμος για μετάδοση δεδομένων 2

```
/* ========================================================== */
/*                                                            */
/*   broadcast2.c                                             */
/* Margarita Antonaki                                         */
/*                                                            */
/* ========================================================== */


/* Algorithmos 2: O epeksergastis Pj  opou j<=2i-1 stelei to minima/stoixio   */
/*ston epeksergasti Pj+2  i-1. (xronos : (g+L)log p)                          */

 //lines 238
#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <math.h>

#define main oldmain
#include "testbsp.c"
#undef main


int argc;
char** argv;
double  duration,time1,time2;
FILE *f1;
int cmsg[200];

void exclusive_main( t_bsp* bsp, void* param ) {
```

```
//DILWSI METAVLITWN

doTests( bsp, argc, argv );
t_bspmsg *msg;
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int i,data=pid;
int *data1;
float rep=ceil(log(nprocs)/log(2));
int extra2=1;
int printed=0;
int vcmsg=0,c;


//ANOIGMA ARXEIOU

f1 = fopen ("out.lis", "wt");

time1=bsp_time(bsp);

  for(i=1;i<=rep;i++){
              if(pid<pow(2,i-1)){
               if((pow(2,i-1)+pid)<nprocs){
                        bsp_send( bsp,pow(2,i-1)+pid, &data, sizeof(int));
                        cmsg[pid]++;
                        }
               }

  bsp_sync( bsp );


          if (pid>=pow(2,i-1))
  {
    msg = bsp_getmsg (bsp, 0);
    if(msg!=NULL)
    {
    data1 = (int*) bspmsg_data (msg);
    data=*data1;
    }
  }

if(pid==0){
          for(c=0;c<nprocs;c++)
            vcmsg=vcmsg+cmsg[c];
          if(vcmsg!=0)
  fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
  vcmsg=0;
}
cmsg[pid]=0;

bsp_sync( bsp );
if(data==0 && printed==0){
  fprintf(f1,"Epeksergastis %d data=%d\n", pid,data);
  printed=1;
  }
}
```

```
  time2=bsp_time(bsp);

  bsp_threadsafe_done( bsp );
}


int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    int numofpro;

    printf("Dose ton arithmo ton epeksergaston: _");
    scanf("%d",&numofpro);
    printf("\n");

    argc = _argc; argv = _argv;

    bsplib_saveargs( &argc, &argv );
    vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
    if( vprocs==0 )
        vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
    if( vprocs==0 )
        vprocs = 4;

    bsplib_init( BSPLIB_STDPARAMS, bsp );

    if( bsp_pid(bsp)== 0)
        printf( "creating %d virtual procs\n", vprocs );


    bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

    fprintf( f1,"\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf defterolepta\n",time2-time1
);
    fclose (f1);


 bsplib_done();
    free( bsp );
#ifdef MEMLOG
    checkMemLog();
#endif
    return 0;
}
```

Α.3 Αλγόριθμος για μετάδοση δεδομένων 3

```
/* ========================================================== */
/*                                                            */
/*   broadcast3.c                                             */
/* Margarita Antonaki                                         */
/*                                                            */
/* ========================================================== */


/*Algorithmos3: O epeksergastis Pj, opou j<=ki  stelei to minima/stoixio se  k-1*/
/*diaforetikous epeksergastes Pj+ki ? , 1<=?<=k-1. Sto telos kathe iperbimatos i*/
/*iparxoun ki+1  epeksergastes pou kseroun to minima.(xronos: ((k-1)g+L)logk p) */

//lines 343

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <math.h>

#define main oldmain
#include "testbsp.c"
#undef main


int argc;
char** argv;
double  duration,time1,time2;
FILE *f1;
int cmsg[200];

void exclusive_main( t_bsp* bsp, void* param ) {

  //DILWSI METAVLITWN

  doTests( bsp, argc, argv );
  t_bspmsg *msg;
  int k=2;
  int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
  int i=1,l=1,kpro=k-1;
  int count,data=pid;
  int *data1;
  int temp;
  float rep=ceil(log(nprocs)/log(k));
```

```
int printed=0;
int c,vcmsg=0;

//ANOIGMA ARXEIOU

f1 = fopen ("out.lis", "wt");

//OTAN OI EPEKSERGASTES EINAI LIGOTEROI APO TO K
//DEN XREIAZETAI NA ELEKSOUME AFTI TIN PERITPOSI

if(nprocs<k || k==1 ){
  if (pid==0)
  printf("Lathos stin eisodo!!!\n");
exit(-1);
}

time1=bsp_time(bsp);

//EPIDI KSEKINOUN K EPEKSERGASTES NA STELOUN MINIMA KAI OXI MONO ENAS
//O EPEKSERGASTIS ME pid=0 STELLEI MINIMA SE OLOUS TOUS YPOLOIPOUS

if (pid==0)
  {
    bsp_gsend(bsp, 1, k-1, & data, sizeof(int));
    cmsg[pid]= k-1;
  }
  bsp_sync(bsp);


//O KATHE EPEKSERGASTIS POU PARELAVE MINIMA TO APOTHIKEVEI
//STIN TOPIKI TOU MNIMI
for (i=0; i<bsp_nmsgs (bsp); i++) {
  msg = bsp_getmsg (bsp, i);
  data1=(int *) bspmsg_data (msg);
  data=*data1;
  }

bsp_sync(bsp);

i=1;
if(pid==0){
          for(c=0;c<nprocs;c++)
            vcmsg=vcmsg+cmsg[c];
          if(vcmsg!=0)
  fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
  vcmsg=0;
}
cmsg[pid]=0;

bsp_sync( bsp );
```

```
if(data==0 && printed==0){
  fprintf(f1,"Epeksergastis %d data=%d\n", pid,data);
  printed=1;
  }

if(k!=nprocs){
  while(i<=rep){
            if(pid<pow(k,i)){
              while(l<=kpro){
                temp=pow(k,i) *l +pid;
       if(temp<nprocs){
                          bsp_send( bsp,temp, &data, sizeof(int));
                          cmsg[pid]++;
                          }
                        l++;
                }
              l=1;
             }

bsp_sync( bsp );


          for (count=0; count<bsp_nmsgs (bsp); count++) {
            msg = bsp_getmsg (bsp, count);
            data1 = (int*) bspmsg_data (msg);
            data= *data1;
            }

          i++;

          if(pid==0){
           for(c=0;c<nprocs;c++)
             vcmsg=vcmsg+cmsg[c];
            if(vcmsg!=0)
    fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
  vcmsg=0;
}
cmsg[pid]=0;

bsp_sync( bsp );
if(data==0 && printed==0){
  fprintf(f1,"Epeksergastis %d data=%d\n", pid,data);
  printed=1;
  }
}}

time2=bsp_time(bsp);
bsp_threadsafe_done( bsp );
}
```

```
int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   int numofpro;

   printf("Dose ton arithmo ton epeksergaston: _");
   scanf("%d",&numofpro);
   printf("\n");

   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

   fprintf( f1,"\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf
defterolepta\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

Α.4 Αλγόριθμος για μετάδοση δεδομένων 1 – μετάδοση πινάκων (μεγαλύτερο μέγεθος μηνυμάτων)

```
//*=============================================================*/
/*                                                             */
/*   broadcast1Array.c                                         */
/* Margarita Antonaki                                          */
/*                                                             */
/* ============================================================ */


/*Algorithoms1: Enas epeksergastis, esto o P1 steli to minima/stixio      */
/*se olous tous ipoloipous epeksergastes(xronos (p-1)g+L)                 */
//93 lines

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int n;
double  duration,time1,time2;
FILE *f1;

void exclusive_main( t_bsp* bsp, void* param ) {

  //DILWSI METAVLITWN

  doTests( bsp, argc, argv );
   t_bspmsg *msg;
  int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
  int i,c;
  int *data1;
  int data[n];
  int cmsg=0;

  //ANOIGMA ARXEIOU

  f1 = fopen ("out.lis", "wt");

  for(c=0;c<n;c++)
```

```
  data[c]=pid;

 time1=bsp_time(bsp);

 //O EPEKSERGASTIS ME pid=0 STELLEI MINIMA SE OLOUS TOUS YPOLOIPOUS

 if (pid==0){
    bsp_gsend(bsp, 0, bsp_nprocs(bsp)-1,data, sizeof(int)*n);
  cmsg= bsp_nprocs(bsp)-1;
 }
 bsp_sync(bsp);


 //O KATHE EPEKSERGASTIS POU PARELAVE MINIMA TO APOTHIKEVEI
 //STIN DIKI TOU TOPIKI MNIMI


  msg = bsp_getmsg (bsp, 0);
  if(msg!=NULL) {
    data1 = (int*) bspmsg_data (msg);
    for(c=0;c<n;c++)
    data[c]=data1[c];
    }


 bsp_sync(bsp);

 time2=bsp_time(bsp);

 fprintf(f1,"Epeksergastis %d kai o arithmos minimaton pou estile einai %d \n\n", pid,cmsg);
 for(c=0;c<n;c++)
  fprintf(f1,"%d ",data[c]);
 fprintf(f1,"\n");

 bsp_threadsafe_done( bsp );
 }



int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   int numofpro;

   printf("Dose ton arithmo ton epeksergaston: _");
   scanf("%d",&numofpro);
   printf("\n");

   printf("Dose ton megethos tou pinaka: _");
   scanf("%d",&n);
   printf("\n");


   argc = _argc; argv = _argv;
```

```
   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", numofpro);
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );


   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );


    fprintf(f1, "\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf defterolepta\n",time2-
time1 );
    fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

Α.5 Αλγόριθμος για μετάδοση δεδομένων 2 – μετάδοση πινάκων (μεγαλύτερο μέγεθος μηνυμάτων)

```
/* ============================================================== */
/*                                                                */
/*   broadcast2Array.c                                            */
/* Margarita Antonaki                                             */
/*                                                                */
/* ============================================================== */


/* Algorithmos 2: O epeksergastis Pj  opou j<=2i-1 stelei to minima/stoixio   */
/*ston epeksergasti Pj+2  i-1. (xronos : (g+L)log p)                          */



#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <math.h>
```

```
#define main oldmain
#include "testbsp.c"
#undef main


int argc;
char** argv;
double  duration,time1,time2;
int n;
FILE *f1;
int cmsg[200];

void exclusive_main( t_bsp* bsp, void* param ) {

 //DILWSI METAVLITWN

 doTests( bsp, argc, argv );
 t_bspmsg *msg;
 int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
 int i,c;
 int count,data[n];
 int *data1;
 float rep=ceil(log(nprocs)/log(2));
 int printed=0;
 int vcmsg=0,datas=0;

 //ANOIGMA ARXEIOU

  f1 = fopen ("out.lis", "wt");

 for(c=0;c<n;c++)
  data[c]=pid;

 time1=bsp_time(bsp);

  for(i=1;i<=rep;i++){
              if(pid<pow(2,i-1)){
               if((pow(2,i-1)+pid)<nprocs){
                        bsp_send( bsp,pow(2,i-1)+pid,data, (sizeof(int)*n));
                        cmsg[pid]++;
                        }
               }

  bsp_sync( bsp );


          if (pid>=pow(2,i-1)){
    msg = bsp_getmsg (bsp, 0);
    if(msg!=NULL) {
      data1 = (int*) bspmsg_data (msg);
      for(c=0;c<n;c++)
        data[c]=data1[c];
    }
```

```
  datas=1;
  }

  if(pid==0){
            for(c=0;c<nprocs;c++)
              vcmsg=vcmsg+cmsg[c];
            if(vcmsg!=0)
   fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
   vcmsg=0;
  }
  cmsg[pid]=0;


  }

  time2=bsp_time(bsp);

  fprintf(f1,"Epeksergastis %d Pinakas:\n",pid);
  for(c=0;c<n;c++)
   fprintf(f1,"%d ",data[c]);
  fprintf(f1,"\n");

  bsp_threadsafe_done( bsp );
}


int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   int numofpro;

   printf("Dose ton arithmo ton epeksergaston: _");
   scanf("%d",&numofpro);
   printf("\n");

   printf("Dose to megethos toy pinaka: _");
   scanf("%d",&n);
   printf("\n");

   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );
```

```
    bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

    fprintf(f1, "\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf defterolepta\n",time2-time1
);
    fclose (f1);

   bsplib_done();
  free( bsp );
#ifdef MEMLOG
    checkMemLog();
#endif
    return 0;
}
```

Α.6 Αλγόριθμος για μετάδοση δεδομένων 3 – μετάδοση πινάκων (μεγαλύτερο μέγεθος μηνυμάτων)

```
/* ================================================================= */
/*                                                                   */
/*   broadcast3Array.c                                               */
/* Margarita Antonaki                                                */
/*                                                                   */
/* ================================================================= */


/*Algorithmos3: O epeksergastis Pj, opou j<=ki  stelei to minima/stoixio se  k-1*/
/*diaforetikous epeksergastes Pj+kiλ , 1<=λ<=k-1. Sto telos kathe iperbimatos i*/
/*iparxoun ki+1  epeksergastes pou kseroun to minima.(xronos: ((k-1)g+L)logk p) */



#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <math.h>

#define main oldmain
#include "testbsp.c"
#undef main
FILE *f1;


int argc;
char** argv;
double  duration,time1,time2;
int n;
int cmsg[200];
```

```
void exclusive_main( t_bsp* bsp, void* param ) {

 // DILWSI METAVLITWN

 doTests( bsp, argc, argv );
 t_bspmsg *msg;
 int k=2;
 int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
 int i=1,c,kpro=k-1,l=1;
 int a=1;
 int count,data[n];
 int *data1;
 int temp;
 float rep=ceil(log(nprocs)/log(k));
 int printed=0;
 int vcmsg=0;


 //ANOIGMA ARXEIOU

 f1 = fopen ("out.lis", "wt");

 for(c=0;c<n;c++)
  data[c]=pid;

 //OTAN OI EPEKSERGASTES EINAI LIGOTEROI APO TO K
 //DEN XREIAZETAI NA ELEKSOUME AFTI TIN PERIPTOSI

 if(nprocs<k || k==1 ){
  if (pid==0)
  printf("Lathos stin eisodo!!!\n");
 exit(-1);
 }


 //EPIDI KSEKINOUN K EPEKSERGSTES NA STELOUN MINIMA KAI OXI MONO ENAS

 time1=bsp_time(bsp);

 if (pid==0){
    bsp_gsend(bsp, 0, k-1, data, (sizeof(int)*n));
  cmsg[pid]= k-1;
 }
 bsp_sync(bsp);


    msg = bsp_getmsg (bsp, 0);
    if(msg!=NULL) {
    data1 = (int*) bspmsg_data (msg);
    for(c=0;c<n;c++)
      data[c]=data1[c];
    }


 i=1;
```

```
if(pid==0){
            for(c=0;c<nprocs;c++)
              vcmsg=vcmsg+cmsg[c];
            if(vcmsg!=0)
  fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
  vcmsg=0;
}
cmsg[pid]=0;

if(k!=nprocs){
  while(a<=rep){
              if(pid<pow(k,i)){
                while(l<=kpro){
                  temp=pow(k,i) *l +pid;
                   if(temp<nprocs){
                                bsp_send( bsp,temp, data, (sizeof(int)*n));
                               cmsg[pid]++;
                               }
                               l++;
                          }
                  l=1;
               }
           bsp_sync( bsp );



  msg = bsp_getmsg (bsp, 0);
  if(msg!=NULL) {
    data1 = (int*) bspmsg_data (msg);
    for(c=0;c<n;c++)
      data[c]=data1[c];
    }

           i++;
           a++;

           if(pid==0){
            for(c=0;c<nprocs;c++)
              vcmsg=vcmsg+cmsg[c];
            if(vcmsg!=0)
  fprintf(f1,"Stalthikan %d minimata sto %d ipervima\n",vcmsg,i);
  vcmsg=0;
}
cmsg[pid]=0;

}}

time2=bsp_time(bsp);

fprintf(f1,"Epeksergastis %d Pinakas:\n", pid);
for(c=0;c<n;c++)
  fprintf(f1,"%d ",data[c]);
fprintf(f1,"\n");
```

```
  bsp_threadsafe_done( bsp );
}


int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    int numofpro;

    printf("Dose ton arithmo ton epeksergaston: _");
    scanf("%d",&numofpro);
    printf("\n");

    printf("Dose ton megethos ton pinakon: _");
    scanf("%d",&n);
    printf("\n");

    argc = _argc; argv = _argv;

    bsplib_saveargs( &argc, &argv );
    vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
    if( vprocs==0 )
        vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
    if( vprocs==0 )
        vprocs = 4;

    bsplib_init( BSPLIB_STDPARAMS, bsp );

    if( bsp_pid(bsp)== 0)
        printf( "creating %d virtual procs\n", vprocs );


    bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

     fprintf(f1, "\n-->O xronos pou xriastike o algorithos gia na ektelesti einai %.6lf defterolepta\n",time2-
time1 );
     fclose (f1);

 bsplib_done();
    free( bsp );
#ifdef MEMLOG
    checkMemLog();
#endif
    return 0;
}
```

Α.7 Αλγόριθμος παράλληλη άρθροιση σε τετραγωνικό δίκτυο αλληλοσύνδεσης

```
/* ============================================================== */
/*                                                                */
/* Margarita Antonaki                                             */
/* MeshParallliliArthisi.c                                        */
/*  Parallili arthoisi oi epeksergastes prepei na ine isi me aritho pou exi */
/*  riza  px 16                                                   */
/*                                                                */
/* ============================================================== */

//165 lines

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include<time.h>
#include<math.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg=0;
FILE *f1;

void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

doTests( bsp, argc, argv );
t_bspmsg *msg;
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int *data;
int i;
int n=nprocs;
int temp;
int team;

//PINAKAS POU ANTIPROSOPEVEI TA SOIXEIA TOU MESH (TETRAGWNIKOU PLEGMATOS)
int mesh[n];
for(i=0;i<n;i++)
  mesh[i]=i+1;
temp=mesh[pid];
```

```
//ANOIGMA ARXEIOU
f1 = fopen ("out.lis", "wt");

////////////////////////////////TEAM//////////////////////////////////
//DIAMIRASMOS TON EPEKSERGASTON SE OMADES

int nft=sqrt(n);
for(i=1;i<=nft;i++){
  if(i!=1){
    if(pid<i*nft && pid>=(i-1)*nft)
      team=i;
     }
  else if(pid<i*nft)
    team=1;
}


//STELOUN OLOI OI EPEKSERGASTES TIS OMADAS TOUS SE EPEKSERGASTI STA DEKSIA AN
IPARXEI

int cou=1;
int took=1;
int times=0;

while(cou<=nft){
//PROTO BIMA STELEI O KATHENAS STON EPOMENO TOU MESA STIN OMADA
if ((pid+1)<(team*nft) && took==1){
  bsp_send( bsp,pid+1,&mesh[pid],sizeof(int));
  cmsg++;
  }
bsp_sync(bsp);
took=0;

msg = bsp_getmsg (bsp,0);
if(msg!=NULL){
  data= (int *) bspmsg_data (msg);
  if(times>=1){
    mesh[pid]=temp;
  }
  mesh[pid]=mesh[pid] + *data;
  took=1;
  times++;
  }
cou++;
}
if(pid==0){
  fprintf(f1,"1 Ipervima minimata: %d\n",cmsg);
  cmsg=0;
  }


//OI TELEFTAIOI EPEKSERGASTES KATHE OMADAS THA STELOUN ME SEIRA
//TA DEDOMENA TOUS STON TELEFTAIO TIS EPOMENIS OMADAS
```

```
temp=-1;
for(i=1;i<nft;i++){
  if(pid==(i*nft)-1){
    bsp_send( bsp,((i+1)*nft)-1,&mesh[pid],sizeof(int));
    cmsg++;
    }
  bsp_sync(bsp);

  msg = bsp_getmsg (bsp,0);
  if(msg!=NULL){
    data= (int *) bspmsg_data (msg);
    temp=*data;
    mesh[pid]=mesh[pid] + temp;
  }
}

if(pid==0)
  fprintf(f1,"2 Ipervima minimata: %d\n",cmsg);

if(pid==nprocs-1)
  fprintf(f1,"Processor %d To apotelesma tis artrhoisis einai %d\n",pid,mesh[pid]);


bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    double  duration,time1,time2;
    int numofpro,ok=0,temp;

    while(ok==0){
      printf("Dose ton arithmo ton epeksergaston,(to plithos ton epeksergaston prepei na exi riza tou dio): _");
      scanf("%d",&numofpro);
      printf("\n");

      temp=sqrt(numofpro);
      temp=temp*temp;
      if(temp==numofpro)
        ok=1;
      }

    argc = _argc; argv = _argv;

    bsplib_saveargs( &argc, &argv );
    vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
    if( vprocs==0 )
        vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
    if( vprocs==0 )
        vprocs = 4;

    bsplib_init( BSPLIB_STDPARAMS, bsp );
```

```
   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );


   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

   time2=bsp_time(bsp);
   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf defterolepta\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;}
```

Α.8 Αλγόριθμος προθεματικής άρθροισης για τετραγωνικό δίχτυο αλληλοσύνδεσης

```
//* ============================================================ */
/*                                                               */
/* MeshProthematikiArthisi.c                                    */
/* Margarita Antonaki                                           */
/*                                                              */
/* Prothematiki arthoisi oi epeksergastes prepei na ine isOi me aritho pou exi  */
/*  riza tou 2 px 16                                            */
/* ============================================================ */


//355 lines

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include<time.h>
#include<math.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg=0;
FILE *f1;
```

```
void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

doTests( bsp, argc, argv );
t_bspmsg *msg;
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int *data;
int i, n=nprocs;

//ANOIGMA ARXEIOU
f1 = fopen ("out.lis", "wt");

//PINAKAS POU ANTIPROSOPEVEI TA STOIXEIA TOU MESH (TETRAGWNIKOU PLEGMATOS)

int mesh[n];
for(i=0;i<n;i++)
  mesh[i]=i+1;

int temp=mesh[pid];

////////////////////////////////TEAM//////////////////////////////////
//DIAMIRASMOS TON EPEKSERGASTON SE OMADES

int team;
int nft=sqrt(n);
for(i=1;i<=nft;i++){
  if(i!=1){
    if(pid<i*nft && pid>=(i-1)*nft)
      team=i;
      }
  else if(pid<i*nft)
    team=1;
}



//STELOUN OLOI STON EPEKSERGASTI TIS OMADAS TOUS STA DEKSI AN IPARXEI

int cou=1;
int took=1;
int times=0;

while(cou<=nft){
//PROTO BIMA STELEI O EPEKSERGASTIS STON EPOMENO TOU MESA STIN OMADA

if ((pid+1)<(team*nft) && took==1){
  bsp_send( bsp,pid+1,&mesh[pid],sizeof(int));
  cmsg++;
  }
bsp_sync(bsp);
took=0;

msg = bsp_getmsg (bsp,0);
if(msg!=NULL){
```

```
  data= (int *) bspmsg_data (msg);
  if(times>=1){
    mesh[pid]=temp;
  }
  mesh[pid]=mesh[pid] + *data;
  took=1;
  times++;
  }
cou++;
}
if(pid==0){
  fprintf(f1,"1 Ipervima minimata: %d\n",cmsg);
  cmsg=0;
  }
////////////////////////////////////////////////////////////////////////////

//OI TELEFTAIOI EPEKSERGASTES KATHE OMADAS THA STILOUN ME SEIRA TA DEDOMENA
TOUS
// STON TELAFTAIO TIS EPOMENIS OMADAS

temp=-1;
for(i=1;i<nft;i++){
  if(pid==(i*nft)-1){
    bsp_send( bsp,((i+1)*nft)-1,&mesh[pid],sizeof(int));
    cmsg++;
    }
  bsp_sync(bsp);

  msg = bsp_getmsg (bsp,0);
  if(msg!=NULL){
    data= (int *) bspmsg_data (msg);
    temp=*data;
    mesh[pid]=mesh[pid] + temp;
  }
}

if(pid==0){
  fprintf(f1,"2 Ipervima minimata: %d\n",cmsg);
  cmsg=0;
  }
////////////////////////////////////////////////////////////////////////////

//EXTOS APO TIN PRWTI OMADA EPEKSERGASTWN I IPOLOIPOI, KATHE TELEFTAIOS APO
//KATHE OMADA THA STELEI STOUS PROIGOUMENOUS TIN TIMI POU PIRE //GIA NA TIN
//ARTRHROISOUN STA DIKA TOUS

cou=1;

while(cou<=nft){

if ((pid)>((team-1)*nft) && temp!=-1){
  bsp_send( bsp,pid-1,&temp,sizeof(int));
  cmsg++;
  temp=-1;
  }
```

```
bsp_sync(bsp);

msg = bsp_getmsg (bsp,0);
if(msg!=NULL){
  data= (int *) bspmsg_data (msg);
  temp=*data;
  mesh[pid]=mesh[pid] + temp;
  }
cou++;
}

if(pid==0)
  fprintf(f1,"3 Ipervima minimata: %d\n",cmsg);

fprintf(f1,"Processor %d me stoixeio %d\n",pid,mesh[pid]);
////////////////////////////////////////////////////////////////////////////

bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    double  duration,time1,time2;
    int numofpro,ok=0,temp;

    while(ok==0){
      printf("Dose ton arithmo ton epeksergaston,(to plithos ton epeksergaston prepei na exi riza tou dio): _");
      scanf("%d",&numofpro);
      printf("\n");

      temp=sqrt(numofpro);
      temp=temp*temp;
      if(temp==numofpro)
        ok=1;
      }

    argc = _argc; argv = _argv;

    bsplib_saveargs( &argc, &argv );
    vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
    if( vprocs==0 )
        vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
    if( vprocs==0 )
        vprocs = 4;

    bsplib_init( BSPLIB_STDPARAMS, bsp );

    if( bsp_pid(bsp)== 0)
        printf( "creating %d virtual procs\n", vprocs );


    time1=bsp_time(bsp);
    bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );
```

```
   time2=bsp_time(bsp);
   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf defterolepta\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

A.9 Βέλτιστος αλγόριθμος προθεματικής άρθροισης

```
/* ============================================================== */
/*                                                                */
/* Margarita Antonaki                                             */
/*  ProthematikiArthisi.c                                         */
/*                                                                */
/*  Prothematiki Artrhoisi                                        */
/*  1. mirasmos ton stoixeion stous epeksergastes.                */
/*  2. seiriakos prothematioks ipologismos, pou ginetai apo kathe epegergasti */
/*     sta stoixeia tou.                                          */
/*  3. Parallilos prothematikos ipologismos me ta teleftaia stoixeia */
/*     ton epeksergaston.                                         */
/*  4. Prosthesi ton telefteon stoixeion ton epeksergaston sta stoixeia */
/*        tou epomenou epeksergasti                               */
/* ============================================================== */


/*Dimiourgeis tous eikonikous epeksergastes kai oles i leitourgies pou theleis na kanoun
* ginontai stin sinartisi pou yparxei san parametros stin sinartisi
* bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true ), diladi tin
* exclusive_main.
*/
//isos arithmos epeksergaston kai stoixeion
//189 lines

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include <math.h>

#define main oldmain
#define MAX 16
```

```
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg[200];
FILE *f1;

void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

doTests( bsp, argc, argv );
t_bspmsg *msg;
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int  data, *data1;
int array[MAX];
int myarray[MAX];
int mydata=MAX/nprocs;
int temp= (mydata)*pid;
int count=0;
int smsum=0;
cmsg[pid]=0;
int i,c;
int distance=1;
int result=0;
int mod=MAX%nprocs;
int j;


//ANOIGMA ARXEIOU
f1 = fopen ("out.lis", "wt");

//STIN PERIPTOSI OPOU OI EPEKSERGASTES EINAI PERISSOTEROI APO TA //STOIXEIA
//XRISIMOPOIO OSOUS EPEKSERGASTES OSA EINAI TA STOIXEIA MOU

if(nprocs>MAX)
  nprocs=MAX;

for (i=0;i<MAX;i++)
  array[i]=i;


//GEMIZOUME TON PINAKA ME STOIXEIA APO 0 - max-1
for(i=temp;i<(temp+mydata);i++){
  myarray[count]=array[i];
  count++;
  }

//SE PERIPTOSI POU DEN DIEROUNTE AKRIVOS OI EPEKSERGASTES ME TA //STOIXEIA
//VAZOUME TA STOIXEIA STON TELEFTAIO EPEKSERGASTI

j=mod+i;
if(mod!=0 && pid==(nprocs-1))
  while(i<=j){
```

```
     myarray[count]=array[i];
     count++;
     i++;
   }

//SIRIAKOS IPOLOGISMOS PROTHEMATIKOU ARTHROISMATOS GIA myarray

for(i=1;i<mydata;i++)
   myarray[i]=myarray[i-1]+myarray[i];


//SIRIAKOS IPOLOGISMOS POU GINETAI GIA TA EPIPRISTHETA STOIXEIA
//POU EXEI O TELEFTAIOS EPEKSERGASTIS

j=i+mod;
if(mod!=0 && pid==nprocs-1)
  while(i<=j)
  {
    myarray[i]=myarray[i]+myarray[i-1];
    i++;
  }

//PARALLILOS IPOLOGISMOS PROTHEMATIKOU ARTHROISMATOS TON //TELEFTAIWN
STOIXEIWN KATHE EPEKSERGASTI

//O KATHE EPEKSERGASTIS ESI OS result TO TELEFTAIO STOIXEIO TOY //myarray TOY

result=myarray[mydata-1];

if(mod!=0 && pid==nprocs-1)
result=myarray[mydata+mod-1];


bsp_push_reg(bsp,&result,sizeof(int));
int extra=0;
int halfpro=nprocs/2;

//PARALLILOS PROTHEMATIKOS IPOLOGISMOS ME TA TELEFTAIA //STOIXEIA
//POU EXOUN OI EPEKSERGASTES

while(distance<nprocs)
  {
    if(distance<=pid)
    {
      bsp_get(bsp,pid-distance,&result,0,&extra,sizeof(int));
      cmsg[pid]++;
    }

    bsp_sync(bsp);
    result+=extra;
    extra=0;
    distance=2*distance;

}
```

```
//TOPOTHETISI STO result TO TELEFTAIO STOIXEIO TOU myarray KATHE //EPEKSERGASTI

if(pid!=nprocs-1)
myarray[mydata-1]=result;
else
myarray[mydata-1+mod]=result;

//ARTHROISI TELEFTAIOU STIXIOU ENOS EPEKSERGASTI STA UPOLOIPA //TOU EPOMENOU

if(pid!=0){
bsp_get(bsp,pid-1,&result,0,&extra,sizeof(int));
cmsg[pid]++;
}
bsp_sync(bsp);

//EPEKSERGASTES EXTOS TOU 0 KAI TOU TELEFTAIOU

if(pid!=0 && pid!=nprocs-1){
for(i=0;i<mydata-1;i++)
myarray[i]=myarray[i]+extra;
}

if(pid!=nprocs-1)
for(i=0;i<mydata;i++)
  if(pid<MAX)
    fprintf(f1,"Epeksergastis %d: To prothema mou einai %d\n", pid,myarray[i]);

//TELEFTAIOS EPEKSERGASTIS

if(pid==nprocs-1){
    for(i=0;i<mydata-1+mod;i++)
      myarray[i]=myarray[i]+extra;
    for(i=0;i<mydata+mod;i++)
      if(pid<MAX)
        fprintf(f1,"Epeksergastis %d To prothema mou einai %d \n", pid,myarray[i]);

    }

bsp_pop_reg(bsp,&result);

bsp_sync(bsp);

//IPOLOGISMOS TWN MINIMATWN POU STALIKAN

if(pid==0){
  for(c=0;c<nprocs;c++)
    smsum+=cmsg[c];
  fprintf(f1,"\nTo plithos minimaton pou stalikan einai %d.\n",smsum);
  }

bsp_threadsafe_done( bsp );
}
```

```
int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   int numofpro;
   double  duration,time1,time2;

   printf("Dose ton arithmo ton epeksergaston: _");
   scanf("%d",&numofpro);
   printf("\n");

   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );

   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );
    time2=bsp_time(bsp);

   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf seconds\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

Α.10 Βέλτιστος αλγόριθμος παράλληλης άρθροισης

```
//* ========================================================== */
/*                                                            */
/*   paralliliArthisi.c                                       */
/*   Margarita Antonaki                                       */
/*                                                            */
/* ========================================================== */

//174 lines
```

```
#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include <math.h>

#define main oldmain
#define MAXSIZE 16
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg[200];
FILE *f1;

void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

doTests( bsp, argc, argv );
int pid=bsp_pid(bsp), nprocs=bsp_nprocs(bsp);
int result;
int count,count2=0;
int table[MAXSIZE];
int smsum=0;
int c,sum=0;
FILE *fp;
//ARXIKOPOIISI TOU PLITHOUS MINIMATWN
cmsg[pid]=0;

//ANOIGMA ARXEIOU
f1 = fopen ("out.lis", "wt");


//ANOIGMA ARXEIOU
fp = fopen("file2.txt", "r");

if (fp == NULL) {
  if (pid==0)
    fprintf(f1,"Ipirxe provlima sto anigma tou arxiou.\n");
  exit(-1);
}
else{
  count=0;
  while (!feof(fp)){
    fscanf(fp,"%d", &table[count]);
    count++;
    if(count==MAXSIZE)
```

```
    break;
    }
}

//KLEISIMO ARXEIOU
fclose(fp);

int MAX=count;
int size=MAX/nprocs;
int privatearray[size];
int x=MAX/nprocs;
int start= (x)*pid;

//O KATHE EPEKSERGASTIS PERNEI TA DEDOMENA POU TOU ANTISTOIXOUN
//KAI TA APOTHIKEVEI STIN DIKI TOU TOPIKI MNINI

for(count=start; count<(start+(MAX/nprocs)); count++){
  privatearray[count2]=table[count];
  count2++;
}

int mod=MAX%nprocs;
if(mod!=0){
  if(pid<mod){
    privatearray[count2]=table[x*nprocs+pid];
  }
}



//O KATHE EPEKERGASTIS ARTHOIZEI TA DEDOMENA POU KRATEI
for (count=0; count<size; count++)
 sum+=privatearray[count];

 if(mod!=0)
{
  if(pid<mod)
    sum+=privatearray[count];
}

int i=0;
int temp=2,temp2;
if(nprocs<=MAX)
  temp2=nprocs;
else
  temp2=MAX;

int tsum=0;

bsp_push_reg(bsp,&tsum,sizeof(int));

//SE KATHE EPANALIPSI MEIONONTAI STO MISO OI EPEKSERGASTES POU KRATOUN
//DEDOMENA XRISIMA GIA TIN ARTHROISI. TELIKA TO APOTELESMA THA TO
//KRATA O EPEKSERGASTIS ME pid=0
```

```
while(temp>1)
{
  i++;
  temp=ceil(nprocs/pow(2,i));
  if(pid>=temp && pid<temp2){
    bsp_put(bsp,pid-temp,&sum,&tsum,0,sizeof(int));
    cmsg[pid]++;
    }

  bsp_sync( bsp );

 if(pid<temp){
  sum+=tsum;
  tsum=0;
  }
 temp2=temp;

}

 bsp_pop_reg(bsp,&tsum);

if(pid==0)
 fprintf(f1,"Pid %d: To arthoisma einai iso me %d.\n", pid,sum);

//IPOLOGISMOS TON MINIMATON POU STALIKAN
if(pid==0){
  for(c=0;c<nprocs;c++)
    smsum+=cmsg[c];
  fprintf(f1,"To plithos minimaton pou stalikan einai %d.\n",smsum);
  }


bsp_threadsafe_done( bsp );
}



int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    int numofpro;
    double  duration,time1,time2;

    printf("Dose ton arithmo ton epeksergaston: _");
    scanf("%d",&numofpro);
    printf("\n");

    argc = _argc; argv = _argv;

    bsplib_saveargs( &argc, &argv );
    vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
    if( vprocs==0 )
        vprocs = argupcaseoptint( argc, argv, "-vp", numofpro );
    if( vprocs==0 )
```

```
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );

   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );
    time2=bsp_time(bsp);

   fprintf(f1, "\n-->O xronos pou xriastike einai %.6lf sec\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

A.11 Αλγόριθμος για πολλαπλασιασμό πινάκων στο EREW (χρησιμοποιείται ο αλγόριθμος για μετάδοση δεδομένων 1)

```
/* ============================================================ */
/*                                                              */
/* Margarita Antonaki                                           */
/*mul1.c                                                        */
/* Pollaplasiasmos pinakon me broadcast 1                       */
/* broadcast -> stelei enas epeksergastis minima se olous tous ipoloipous */
/*                                                              */
/* ============================================================ */
//lines 394

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include<time.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg[200];
```

```
int sum=0,psum,c;
FILE *f1;

/*SINARTISI GIA BROADCAST1,O EPEKSERGASTIS ME TAFTOTITA 0
STELEI TA APARIATITA DEDOMENA SE OLOUS TOUS IPOLOIPOUS EPEKSERGASTES */

void broadcast( t_bsp* bsp, void* param,int A, int B, int prolistA[], int prolistB[],int ca, int cb )
{
 register int nprocs=bsp_nprocs(bsp);
 register int pid=bsp_pid(bsp);
 bsp_lsend(bsp,prolistA,ca,&A,sizeof(int));
 bsp_lsend(bsp,prolistB,cb,&B,sizeof(int));
 if(prolistA[0]!=0 || prolistB[0]!=0 )
  cmsg[pid]=cmsg[pid]+ca+cb;
 else
  cmsg[pid]=cmsg[pid]+ca+cb-2;

  if(pid==0){
    psum=sum;
    sum=0;
    for(c=0;c<nprocs;c++)
      sum+=cmsg[c];
    fprintf(f1,"Minimata pou stalikan kata to broadcast=%d\n",sum-psum);
   }


}


/*                    PARALLILI ARTHROISI                          */

int parallel_sum(t_bsp* bsp, void* param,int i, int j,int inmul,int n,int r, int c)
{
 register int nprocs=bsp_nprocs(bsp);
 register int pid=bsp_pid(bsp);
 int buffer=0;
 int temp2,temp3,cou,cou1,ix,tsum,arrayex[nprocs];
 int result=0;
 int ex=-1;
 int sim=0;
 int minusone=-1;
 int counter=0;

 //ARXIKOPOIISI PINAKA arrayex
 for(cou=0;cou<nprocs;cou++)
  arrayex[cou]=-1;

 //REMOTE MEMORY
 bsp_push_reg(bsp,&tsum,sizeof(int));
 bsp_push_reg(bsp,&ex,sizeof(int));
 bsp_push_reg(bsp,arrayex,sizeof(int));
 bsp_push_reg(bsp,&temp3,sizeof(int));
 bsp_push_reg(bsp,&counter,sizeof(int));
 bsp_push_reg(bsp,&result,sizeof(int));
```

```
 /* OI EPEKSERGASTES POU PREPEI NA PIASOUN KAPIO STOIXEIO APO TOUS PINAKES
EXOUN SIMAIA ISI ME 1 TIN OPOIA KANOUN GNWSTI STON EPEKSERGASTI 0, MESO TOU
PINAKA arrayex */

 if(i==r && j==c){
  sim=1;
  bsp_put(bsp,0,&sim,arrayex,pid*sizeof(int),sizeof(int));
  buffer=inmul;
  }
 bsp_sync( bsp );

 /* O EPEKSERGASTIS 0 DINEI MIA EXTRA TAFTOTITA STO KATHE ENA APO
 /* TOUS EPEKSERGASTES POU PREPEO NA PAROUN KAPOIO STOIXEIO, TIN
 /* OPOIA GNOSTOPOIOUN SE OLOUS TOUS EPEKSERGASTES MESO TOU   arrayex */

 if (pid==0)
  for(cou=0;cou<nprocs;cou++)
   if(arrayex[cou]==1){
    bsp_put(bsp,cou,&counter,&ex,0,sizeof(int));
    for(cou1=0;cou1<nprocs;cou1++){
     bsp_put(bsp,cou1,&counter,arrayex,cou*sizeof(int),sizeof(int));
     }
    counter=counter+1;
    }
  bsp_sync( bsp );

 ix=0;
 temp2=2;

 /* TO counter STO TELOS THA KRATA TO PLITHOS TON EPEKSERGASTON POU      */
 /* KRATOUN DEDOMENA TA OPOIA THA XRIASTOUN STIN ARTHROISI POU THA GINEI */

 if(pid==0)
  for(cou1=0;cou1<nprocs;cou1++){
   bsp_put(bsp,cou1,&counter,&temp3,0,sizeof(int));
   bsp_put(bsp,cou1,&counter,&counter,0,sizeof(int));
   }
  bsp_sync( bsp );
  tsum=0;

  /* PARALLILI ARTHROISI POU GINETAI APO TOUS n EPEKSERGASTES POU KRATOUSAN
DEDOMENA GIA ARTHROISI */

  while(temp2>1) {
   ix++;
   temp2=ceil(counter/pow(2,ix));
   if(sim==1){
    if(ex>=temp2 && ex<temp3){
     for(cou=0;cou<nprocs;cou++)
      if(arrayex[cou]==ex-temp2){
       bsp_put(bsp,cou,&buffer,&tsum,0,sizeof(int));
       }
     }
    }
   bsp_sync( bsp );
```

```
   if(sim==1){
    if(ex<temp2){
      buffer+=tsum;
      tsum=0;
      }
    temp3=temp2;
    }

  }
 bsp_sync( bsp );

  /* O EPEKSERGASTIS ME TIN extra TAFTOTITA 0 KRATA TO APOTELESMA TIS ARTHROISIS
KAI TO STELEI STO EPEKSERGASTI 0 */

 if(ex==0){
  result=buffer;
  bsp_put(bsp,0,&result,&result,0,sizeof(int));
  }
 bsp_sync( bsp );


 bsp_pop_reg(bsp,&tsum);
 bsp_pop_reg(bsp,&ex);
 bsp_pop_reg(bsp,arrayex);
 bsp_pop_reg(bsp,&temp3);
 bsp_pop_reg(bsp,&counter);
 bsp_pop_reg(bsp,&result);

 return result;
}


void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

int pid, nprocs;
doTests( bsp, argc, argv );
nprocs=bsp_nprocs(bsp);
pid=bsp_pid(bsp);

//TRIA STOIXEIA XARAXTIRISTIKA GIA KATHE EPEKSERGASTI
int k,i,j;
//DILWSI DIO PINAKWN MEGETHOUS nXn
int n=4;
int A[n][n],B[n][n],C[n][n];
int r,c,k1=1;
//PLITHOS EPEKSERGASTWN SE KATHE OMADA
int nopro=nprocs/n;
//METAVLITES GIA PROIGOUMENI OMADA KAI GIA OMADA POU EIMASTE TORA
int oldteam=0,nowteam=nopro;
//METRITIS OMADON- EXOUME TOSO PLITHOS OMADON OSO EINAI TO n
int teamc;
//METRITIS
```

```
int counter;
//STOIXEIA POU KRATA O KATHE EPEKSERGASTIS APO TOUS PINAKES
int *a,*b;
int sa,sb;
//STOIXEIA GIA TO broadcast
int i1;
t_bspmsg *msg;
//PINAKES POU KRATOUN PIOI EPEKSERGASTES PREPEI NA PAROUN STOIXEIO APO TO 0
KATHE STIGMI
int IDSA[nprocs];
int IDSB[nprocs];

//MONO O EPEKSERGASTIS ME pid=0 KRATA TOUS 3 PINAKES
if(pid==0)
{
  //ARXIKOPOIISI PINAKON A,B
  for(r=0;r<n;r++)
    for(c=0;c<n;c++)
    {
      A[r][c]=k1;
      B[r][c]=k1;
      C[r][c]=0;
      k1++;
    }
}


//TOPOTHETISI TON EPEKSERGASTON SE OMADES,ME TIN EVRESI TOU k GIA TON KATHENA
for(teamc=1;teamc<=n;teamc++)
  {
    if(pid>=oldteam && pid<nowteam){
      k=teamc-1;

      //EVRESI TON i KAI j TON EPEKSERGASTON
      //OI EPEKSERGASTES PERNOUN AAFTA TA STOIXEIA OPOS EINAI DOSMENA STON
PINAKA
      //ANALOGA ME TIN SEIRA TOUS px O 0 PERNI TO 1,1 ,o 2 to 1,2 ktl

      counter=0;
      for(r=0;r<n;r++)
        for(c=0;c<n;c++)
        {
          if(pid-((teamc-1)*nopro)==counter)
          {
            i=r;
            j=c;
          }
        counter++;
      }
    }

    oldteam=nowteam;
    nowteam=nowteam+nopro;

    }
```

```
//ALGORITHOS GIA POLLAPLASIASMO POU THA EXI broadcast KAI arthrisi

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA a
int prolistA[nprocs];
for(r=0;r<nprocs;r++)
  prolistA[r]=0;
//COUNTER GIA a
int ca=0;

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA b
int prolistB[nprocs];
for(r=0;r<nprocs;r++)
    prolistB[r]=0;
//counter gia b
int cb=0;

//FLAGS
int fa=0;
int fb=0;

//ARXIKOPOIISI PINAKON IDSA KAI IDSB
for(r=0;r<nprocs;r++){
    IDSA[r]=0;
    IDSB[r]=0;
  }

bsp_push_reg(bsp,IDSA,nprocs*sizeof(int));
bsp_push_reg(bsp,IDSB,nprocs*sizeof(int));

int temp;
int data=1;
int idzero=0;
int ta;
int atemp,btemp;
//METRITIS GIA MINIMATA
int cou=-1;


for(r=0;r<n;r++){
  for(c=0;c<n;c++){
    //O KATHE EPEKSERGASTIS DILONI AN THELAI NA PIASEI KAPOIO DEDOMENO
    if(i==r && k==c){
      bsp_put(bsp,idzero,&data,IDSA,pid*sizeof(int),sizeof(int));
      fa=1;
      cou++;
      }
    if(j==c && k==r){
      bsp_put(bsp,idzero,&data,IDSB,pid*sizeof(int),sizeof(int));
      fb=1;
      cou++;
      }
    bsp_sync(bsp);
```

```
if(pid==0){
 //O EPEKSERGASTIS 0 PROSPATHEI NA VREI PIOI EPEKSERGASTES THELOUN STOIXEIA
 //KRATA SE MIA LISTA SE POIOUS EPEKSERGASTES PREPEI NA STEILEI DEDOMENA
 ca=0;
 cb=0;
 for(temp=0;temp<nprocs;temp++){
  if(IDSA[temp]==1){
   prolistA[ca]=temp;
   ca++;
   }

  if(IDSB[temp]==1){
   prolistB[cb]=temp;
   cb++;
   }

  }

 //BROADCAST
 broadcast(bsp,param,A[r][c],B[r][c],prolistA,prolistB,ca,cb);

 //ARXIKOPOIISI PINAKON IDSA KAI IDSB
 for(ta=0;ta<nprocs;ta++){
  IDSA[ta]=0;
  IDSB[ta]=0;
  prolistA[ta]=0;
  prolistB[ta]=0;
  }

 }//TELOS if
 bsp_sync(bsp);

 //OSOI EPEKSERGASTES EPREPE NA PAROUN STOIXEIA APO TON 0 TA PERNOUN
 //KAI TA APOTHIKEVOUN STIN a KAI b METAVLITI

 //PERNEI DEDOMENO GIA a KAI b
 if(bsp_nmsgs (bsp)!=0 && fa==1 && fb==1){
  msg = bsp_getmsg (bsp, 0);
  a=(int *) bspmsg_data (msg);
  msg = bsp_getmsg (bsp, 1);
  b=(int *) bspmsg_data (msg);
  fa=0;
  fb=0;
  sa=*a;
  sb=*b;
  }

  //PERNEI DEDOMENO GIA a
 if(bsp_nmsgs (bsp)!=0 && fa==1){
  msg = bsp_getmsg (bsp, cou);
  a=(int *) bspmsg_data (msg);
  sa=*a;
  fa=0;
  }
```

```
        //PERNEI DEDOMENO GIA b
        if(bsp_nmsgs (bsp)!=0 && fb==1){
           msg = bsp_getmsg (bsp, cou);
           b=(int *) bspmsg_data (msg);
           sb=*b;
           fb=0;
           }

        cou=-1;
        }
       }

      bsp_sync(bsp);

      //POLAPLASIASMOS a ,b
      int inmul;
      inmul=sa * sb;

       for(r=0;r<n;r++)
        {
          for(c=0;c<n;c++)
            {
             C[r][c]=parallel_sum(bsp,param,i,j,inmul,n,r,c);
            }
        }

//EKTIPOSI PINAKON A KAI B
if(pid==0){
printf("Pinakas A: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",A[r][c]);
    printf("\n");
    }
}

if(pid==0){
printf("Pinakas B: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",B[r][c]);
    printf("\n");
    }
}

//IPOLOGISMOS PINAKA C KAI EKTIPOSI APO TON EPEKSERGASTI ME pid 0
if(pid==0)
printf("\nPinakas Apotelesmatvn apo ton polaplasiasmo ton pinakon A kai B:\n");
  for(r=0;r<n;r++)
    {
      for(c=0;c<n;c++)
       {
         C[r][c]=parallel_sum(bsp,param,i,j,inmul,n,r,c);
         if(pid==0)
           fprintf(" %d ",C[r][c]);
```

```
      }
      if(pid==0)
       printf("\n");
   }
  bsp_sync(bsp);

bsp_pop_reg(bsp,IDSA);
bsp_pop_reg(bsp,IDSB);

bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   double  duration,time1,time2;
   f1 = fopen ("out.lis", "wt");


   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", 64 );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );

   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

   time2=bsp_time(bsp);
   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf sec\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

A.12 Αλγόριθμος για πολλαπλασιασμό πινάκων στο EREW (χρησιμοποιείται ο αλγόριθμος για μετάδοση δεδομένων 2)

```
/* ============================================================ */
/*                                                              */
/*   Margatita Antonaki                                         */
/*   mul2.c                                                     */
/*   Pollaplasiasmos pinakon (n stin 3 epeksergastes)           */
/*   me parallilili arthoisi kai broadcast(2)                   */
/*                                                              */
/*                                                              */
/* ============================================================ */
//lines 547


#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include<time.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg[200];
int sum=0,psum,c;
FILE *f1;


/*SINARTISI GIA broadcast2*/

int broadcast( t_bsp* bsp, void* param,int datas,int ca,int simaia,int extra,int arrayextra[] )
{
  register int nprocs=bsp_nprocs(bsp);
  register int pid=bsp_pid(bsp);

  int  temps=ceil(log(ca+1)/log(2));
  int is;
  int *data1s;
  int cou;
  t_bspmsg *msg;
  int sa=0;

  for(is=1; is<=temps; is++){
    if(simaia==1 || pid==0){
      if(extra<pow(2,is-1))
```

```
      if((extra+pow(2,is-1))<=ca+1){
        for(cou=0;cou<nprocs;cou++)
          if(arrayextra[cou]==extra+pow(2,is-1)){
            bsp_send( bsp, cou, &datas, sizeof(int) );
            cmsg[pid]++;
            }
        }
      }
   bsp_sync( bsp );

   if(simaia==1 || pid==0){
    if (extra>=pow(2,is-1)){
      msg = bsp_getmsg (bsp, 0);
      if(msg!=NULL){
        data1s = (int*) bspmsg_data (msg);
        datas=*data1s;
        }
      }
    }
   bsp_sync( bsp );

   }


  if(pid==0){
    psum=sum;
    sum=0;
    for(c=0;c<nprocs;c++)
      sum+=cmsg[c];
    fprintf(f1,"Minimata pou stalikan kata to broadcast=%d\n",sum-psum);
   }

  if(simaia==1)
   sa=datas;

return sa;
}


/*                    PARALLILI ARTHROISI                          */

int parallel_sum(t_bsp* bsp, void* param,int i, int j,int inmul,int n,int r, int c)
{
  register int nprocs=bsp_nprocs(bsp);
  register int pid=bsp_pid(bsp);
  int buffer=0;
  int temp2,temp3,cou,cou1,ix,tsum,arrayex[nprocs];
  int result=0;
  int ex=-1;
  int sim=0;
  int minusone=-1;
  int counter=0;

  //ARXIKOPOIISI  PINAKA arrayex
  for(cou=0;cou<nprocs;cou++)
```

```
   arrayex[cou]=-1;

  //REMOTE MEMORY
  bsp_push_reg(bsp,&tsum,sizeof(int));
  bsp_push_reg(bsp,&ex,sizeof(int));
  bsp_push_reg(bsp,arrayex,sizeof(int));
  bsp_push_reg(bsp,&temp3,sizeof(int));
  bsp_push_reg(bsp,&counter,sizeof(int));
  bsp_push_reg(bsp,&result,sizeof(int));

  /* OI EPEKSERGASTES POU PREPEI NA PIASOUN KAPIO STOIXEIO APO TOUS PINAKES
EXOUN SIMAIA ISI ME 1 TIN OPOIA KANOUN GNOSTI STON EPEKSERGASTI 0, MESO TOU
PINAKA arrayex */

  if(i==r && j==c){
    sim=1;
    bsp_put(bsp,0,&sim,arrayex,pid*sizeof(int),sizeof(int));
    buffer=inmul;
    }
  bsp_sync( bsp );

  /* O EPEKSERGASTIS 0 DINEI MIA EXTRA TAFTOTITA STO KATHE ENA APO
  /* TOUS EPEKSERGASTES POU PREPEO NA PAROUN KAPOIO STOIXEIO, TIN
  /* OPOIA GNOSTOPOIOUN SE OLOUS TOUS EPEKSERGASTES MESO TOU arrayex */

  if (pid==0)
    for(cou=0;cou<nprocs;cou++)
      if(arrayex[cou]==1){
        bsp_put(bsp,cou,&counter,&ex,0,sizeof(int));
        for(cou1=0;cou1<nprocs;cou1++){
          bsp_put(bsp,cou1,&counter,arrayex,cou*sizeof(int),sizeof(int));
          }
        counter=counter+1;
        }
    bsp_sync( bsp );

  ix=0;
  temp2=2;

  /* TO counter STO TELOS THA KRATA TO PLITHOS TON EPEKSERGASTON POU    KRATOUN
DEDOMENA TA OPOIA THA XRIASTOUN STI ARTHROIS POU THA GINEI */

  if(pid==0)
    for(cou1=0;cou1<nprocs;cou1++){
      bsp_put(bsp,cou1,&counter,&temp3,0,sizeof(int));
      bsp_put(bsp,cou1,&counter,&counter,0,sizeof(int));
      }
    bsp_sync( bsp );
    tsum=0;

   /* PARALLILI ARTHROISI POU GINETAI APO TOUS n EPEKSERGASTES POU KRATOUSAN
,DEDOMENA GIA ARTHROISI */
    while(temp2>1) {
      ix++;
      temp2=ceil(counter/pow(2,ix));
```

```
    if(sim==1){
      if(ex>=temp2 && ex<temp3){
        for(cou=0;cou<nprocs;cou++)
          if(arrayex[cou]==ex-temp2){
            bsp_put(bsp,cou,&buffer,&tsum,0,sizeof(int));
            }
        }
      }
    bsp_sync( bsp );

   if(sim==1){
     if(ex<temp2){
       buffer+=tsum;
       tsum=0;
       }
     temp3=temp2;
     }

   }
  bsp_sync( bsp );

  /* O EPEKSERGASTIS ME TIN extra TAFTOTITA 0 KRATA TO APOTELESMA TIS ARTHROISIS
KAI TO STELEI STO EPEKSERGASTI 0 */
  if(ex==0){
    result=buffer;
    bsp_put(bsp,0,&result,&result,0,sizeof(int));
    }
  bsp_sync( bsp );


  bsp_pop_reg(bsp,&tsum);
  bsp_pop_reg(bsp,&ex);
  bsp_pop_reg(bsp,arrayex);
  bsp_pop_reg(bsp,&temp3);
  bsp_pop_reg(bsp,&counter);
  bsp_pop_reg(bsp,&result);

  return result;
}



void exclusive_main( t_bsp* bsp, void* param ) {


//DILWSI METAVLITWN

int pid, nprocs;
doTests( bsp, argc, argv );
nprocs=bsp_nprocs(bsp);
pid=bsp_pid(bsp);

//TRIA STOIXEIA XARAXTIRISTIKA GIA KATHE EPEKSERGASTI
int k,i,j;
//DILWSI DIO PINAKWN MEGETHOUS nXn
```

```
int n=2;
int A[n][n],B[n][n],C[n][n];
int r,c,k1=1;
//PLITHOS EPEKSERGASTWN SE KATHE OMADA
int nopro=nprocs/n;
//METAVLITES GIA PROIGOUMENI OMADA KAI GIA OMADA POU EIMASTE TORA
int oldteam=0,nowteam=nopro;
//METRITIS OMADON- EXOUME TOSO PLITHOS OMADON OSO EINAI TO n
int teamc;
//METRITIS
int counter;
//STOIXEIA POU KRATA O KATHE EPEKSERGASTIS APO TOUS PINAKES
int *a,*b;
int sa=0;
int tempsa;
int sb=0;
int tempsb;
//STOIXEIA GIA TO broadcast
int i1;
t_bspmsg *msg;
//PINAKES POU KRATOUN PIOI EPEKSERGASTES PREPEI NA PAROUN STOIXEIO APO TO 0
KATHE STIGMI
int IDSA[nprocs];
int IDSB[nprocs];
int tca,tcb;
int simaib=0;
int simaia=0;
int one=1;
int ma;
int arrayextra[nprocs];
int arrayextrb[nprocs];
int cou;
int  datas, *data1s,is;
int prolistA[nprocs];
//COUNTER GIA  a
int ca=0;
//FLAGS
int fa=0;
int fb=0;


//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA b
int prolistB[nprocs];
//COUNTER GIA b
int cb=0;
int temp;
int data=1;
int id=0;
int ta;
int atemp,btemp;

int extra;
int extrb;
int g,s;
float temps;
```

```
int inmul;
int datasb,*data1sb,isb;
int fla,flb;

//MONO O EPEKSERGASTIS ME pid=0 KRATA TOUS 3 PINAKES
if(pid==0)
{
 //ARXIKOPOIISI PINAKON A,B
  for(r=0;r<n;r++)
    for(c=0;c<n;c++)
    {
      A[r][c]=k1;
      B[r][c]=k1;
      C[r][c]=0;
      k1++;
    }
}
else

 //ARXIKOPOIISI PINAKON A,B
  for(r=0;r<n;r++)
    for(c=0;c<n;c++)
    {
      A[r][c]=0;
      B[r][c]=0;
      C[r][c]=0;
    }


//TOPOTHETISI TON EPEKSERGASTON SE OMADES,ME TIN EVRESI TOU k //GIA TON
KATHENA
for(teamc=1;teamc<=n;teamc++)
  {
    if(pid>=oldteam && pid<nowteam){
      k=teamc-1;


//EVRESI TON i KAI j TON EPEKSERGASTON OI EPEKSERGASTES
//PERNOUN AAFTA TA STOIXEIA OPOS EINAI   DOSMENA STON PINAKA //ANALOGA ME TIN
SEIRA TOUS px O 0 PERNI TO 1,1 ,o 2 to 1,2 ktl

      counter=0;
      for(r=0;r<n;r++)
        for(c=0;c<n;c++)
        {
          if(pid-((teamc-1)*nopro)==counter)
          {
            i=r;
            j=c;
          }
        counter++;
      }
    }

    oldteam=nowteam;
```

```
   nowteam=nowteam+nopro;

   }

//ALGORITHOS GIA POLLAPLASIASMO POU THA EXI broadcast KAI arthrisi

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA a
for(r=0;r<nprocs;r++)
   prolistA[r]=0;
//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA b
for(r=0;r<nprocs;r++)
   prolistB[r]=0;
//ARXIKOPOIISI PINAKON IDSA KAI IDSB
for(r=0;r<nprocs;r++)
  {
   IDSA[r]=0;
   IDSB[r]=0;
  }

bsp_push_reg(bsp,IDSA,nprocs*sizeof(int));
bsp_push_reg(bsp,IDSB,nprocs*sizeof(int));

//BROADCAST
for(r=0;r<n;r++)
  {
   for(c=0;c<n;c++)
     {
      for(ta=0;ta<nprocs;ta++)
        {
         IDSA[ta]=0;
         prolistA[ta]=0;
         IDSB[ta]=0;
         prolistB[ta]=0;
        }

      if(i==r && k==c)
        {
         bsp_put(bsp,id,&data,IDSA,pid*sizeof(int),sizeof(int));
         fa=1;

        }

      if(j==c && k==r)
        {
         bsp_put(bsp,id,&data,IDSB,pid*sizeof(int),sizeof(int));
         fb=1;
        }

      bsp_sync(bsp);

      simaib=0;
      simaia=0;
 //O EPEKSERGASTIS 0 PROSPATHEI NA VREI PIOI EPEKSERGASTES   //THELOUN STOIXEIA
      extra=-1;
      extrb=-1;
```

```
fla=0;
flb=0;

//GIA  TO 0 TO extra THA EINAI PANTA 0
if(pid==0){
  extra=0;
  extrb=0;
  }

//ARXIKOPOIISI PINAKON extra
for(cou=0;cou<nprocs;cou++){
  arrayextra[cou]=0;
  arrayextrb[cou]=0;
  }


bsp_push_reg(bsp,&extra,sizeof(int));
bsp_push_reg(bsp,&simaia,sizeof(int));
bsp_push_reg(bsp,&ca,sizeof(int));
bsp_push_reg(bsp,arrayextra,sizeof(int));
bsp_push_reg(bsp,&fla,sizeof(int));
bsp_push_reg(bsp,&extrb,sizeof(int));
bsp_push_reg(bsp,&simaib,sizeof(int));
bsp_push_reg(bsp,&cb,sizeof(int));
bsp_push_reg(bsp,arrayextrb,sizeof(int));
bsp_push_reg(bsp,&flb,sizeof(int));

if(pid==0 && IDSA[0]==1){
  fla=1;
  for(cou=0;cou<nprocs;cou++){
    bsp_put(bsp,cou,&fla,&fla,0,sizeof(int));
    }
  }
if(pid==0 && IDSB[0]==1){
  flb=1;
  for(cou=0;cou<nprocs;cou++){
    bsp_put(bsp,cou,&flb,&flb,0,sizeof(int));
    }
  }
bsp_sync (bsp);

if(pid==0){
  ca=0;
  cb=0;
  for(temp=0;temp<nprocs;temp++){
     if(IDSA[temp]==1){
       if(temp!=0){
         prolistA[ca]=temp;
         if(fla==0)
           tca=ca+1;
         else
           tca=ca;
         bsp_put(bsp,temp,&tca,&extra,0,sizeof(int));

         for(cou=0;cou<nprocs;cou++){
```

```
          bsp_put(bsp,cou,&tca,arrayextra,temp*sizeof(int),sizeof(int));
          }
        ca++;
        }
       else {
        simaia=1;
        ca++;
        }
       }
   }


 for(temp=0;temp<nprocs;temp++){
  if(IDSB[temp]==1){
   if(temp!=0){
    prolistB[cb]=temp;
    if(flb==0)
     tcb=cb+1;
    else
     tcb=cb;
    bsp_put(bsp,temp,&tcb,&extrb,0,sizeof(int));
     for(cou=0;cou<nprocs;cou++){
      bsp_put(bsp,cou,&tcb,arrayextrb,temp*sizeof(int),sizeof(int));
       }
     cb++;
    }
    else {
     simaib=1;
     cb++;
     }
   }
  }
 }//TELOS IF
 bsp_sync (bsp);

if(pid==0)
  for(temp=0;temp<ca;temp++){
    bsp_put(bsp,prolistA[temp],&one,&simaia,0,sizeof(int));
    bsp_put(bsp,prolistA[temp],&ca,&ca,0,sizeof(int));
  }
  bsp_sync(bsp);

if(pid==0)
  for(temp=0;temp<cb;temp++){
    bsp_put(bsp,prolistB[temp],&one,&simaib,0,sizeof(int));
    bsp_put(bsp,prolistB[temp],&cb,&cb,0,sizeof(int));
  }
  bsp_sync(bsp);

//AN KAI TO pid 0 PREPI NA PAREI STOIXEIO
if(simaia==1 && pid==0){
  sa=A[r][c];
  simaia=0;
  }
```

```
        if(simaib==1 && pid==0){
           sb=B[r][c];
           simaib=0;
           }

        datas=A[r][c];
        datasb=B[r][c];


  //OLI OI EPEKSERGASTES PREPI NA GNORIZOUN TO ca, PROSORINA TO
//ca EINAI 1
        if(simaia!=1 )
          ca=n;
        if(simaib!=1 )
          cb=n;

        tempsa= broadcast( bsp,param,datas,ca,simaia,extra,arrayextra);
        if(simaia==1)
          sa=tempsa;
        tempsb= broadcast( bsp,param,datasb,cb,simaib,extrb,arrayextrb);
        if(simaib==1)
          sb=tempsb;
   }
  }
bsp_sync(bsp);

//POLLAPLASIASMOS a ,b
inmul=sa*sb ;

//EKTIPOSI PINAKON A KAI B
if(pid==0){
printf("Pinakas A: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",A[r][c]);
    printf("\n");
    }
}

if(pid==0){
printf("Pinakas B: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",B[r][c]);
    printf("\n");
    }
}

//IPOLOGISMOS PINAKA C KAI EKTIPOSI APO TON EPEKSERGASTI ME pid 0
if(pid==0)
printf("\nPinakas Apotelesmaton apo ton polaplasiasmo ton pinakon A kai B:\n");
  for(r=0;r<n;r++)
    {
      for(c=0;c<n;c++)
       {
```

```
        C[r][c]=parallel_sum(bsp,param,i,j,inmul,n,r,c);
        if(pid==0)
          printf(" %d ",C[r][c]);
       }
      if(pid==0)
       printf("\n");
    }


bsp_pop_reg(bsp,IDSA);
bsp_pop_reg(bsp,&extra);
bsp_pop_reg(bsp,&simaia);
bsp_pop_reg(bsp,&ca);
bsp_pop_reg(bsp,arrayextra);
bsp_pop_reg(bsp,&fla);

bsp_pop_reg(bsp,IDSB);
bsp_pop_reg(bsp,&extrb);
bsp_pop_reg(bsp,&simaib);
bsp_pop_reg(bsp,&cb);
bsp_pop_reg(bsp,arrayextrb);
bsp_pop_reg(bsp,&flb);


bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   double time1,time2;
   f1 = fopen ("out.lis", "wt");


   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", 8 );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );


   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );

   time2=bsp_time(bsp);
   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf sec\n",time2-time1 );
```

```
   fclose (f1);
 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

A.13 Αλγόριθμος για πολλαπλασιασμό πινάκων στο EREW (χρησιμοποιείται ο αλγόριθμος για μετάδοση δεδομένων 3)

```
/* ========================================================== */
/*                                                            */
/*   Margarita Antonaki                                       */
/*   mul3.c                                                   */
/*   Pollaplasiasmos pinakon (n stin 3 epeksergastes)         */
/*   me parallilili arthoisi kai broadcast(3)                 */
/*                                                            */
/*                                                           */
/* ========================================================== */
//lines 605

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
#include<time.h>

#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;
int cmsg[200];
int sum=0,psum,c;
FILE *f1;

/*SINARTISI GIA broadcast3*/

int broadcast( t_bsp* bsp, void* param,int datas,int ca,int simaia,int extra,int arrayextra[] )
{
  register int nprocs=bsp_nprocs(bsp);
  register int pid=bsp_pid(bsp);
```

```
 //TO k PREPEI NA EINAI MIKROTERO ISO TOY n
 int k=2;
 int i=1,l=1,temp=k-1;
 int a=1;
 int count,data;
 t_bspmsg *msg;
 int *data1;
 int temp2;
 int sa=0;
 int cou;

 data=datas;

 float  temp3=ceil(log(ca+1)/log(k));
 int extra2=1;



if(simaia==1 || pid==0){
//OTAN OI EPEKSERGASTES EINAI LIGOTEROI APO TO k
//DEN XRIAZETAI NA ELEKSOUME AFTI TI PERIPTOSI

if(ca<k || k==1 ){
 if (pid==0)
   printf("Lathos stin eisodo!!!\n");
 exit(-1);
 }
 //EPIDI KSEKINOUN k EPEKSERGASTES NA STELOUN MINIMA KAI OXI MONO ENAS

//O EPEKSERGASTIS 0 THA STILEI SE k EPEKSERGASTES
// AFTOI I k EPEKSERGASTES PREPEI NA EINAI EPEKSERGASTES POU //THELOUN STOIXEIA
 while(extra2<k){
  if(pid==0){
    for(cou=0;cou<nprocs;cou++)
      if(arrayextra[cou]==extra2){
        bsp_send( bsp,cou, &data, sizeof(int));
        cmsg[pid]++;
        }
   }
  extra2++;
 }
 }
 bsp_sync( bsp );

 if(simaia==1 || pid==0){
 for (count=0; count<bsp_nmsgs (bsp); count++) {
   msg = bsp_getmsg (bsp, count);
   if(msg!=NULL){
     data1 = (int*) bspmsg_data (msg);
     data=*data1;
     }
   }
 }
```

```
while(a<=temp3)
{
 if(simaia==1 || pid==0){
  if(extra<pow(k,i))
    {
     while(l<=temp)
       {
        temp2=pow(k,i) *l +extra;
        if(temp2<=ca)
          for(cou=0;cou<nprocs;cou++)
            if(arrayextra[cou]==temp2){
              bsp_send( bsp,cou, &data, sizeof(int));
              cmsg[pid]++;
              }

        l++;
       }
    l=1;
  }
 }

 bsp_sync( bsp );

 if(simaia==1 || pid==0)
 for (count=0; count<bsp_nmsgs (bsp); count++) {
  msg = bsp_getmsg (bsp, count);
  if(msg!=NULL){
    data1 = (int*) bspmsg_data (msg);
    data= *data1;
  }
 }

  i++;
  a++;
}



  if(pid==0){
    psum=sum;
    sum=0;
    for(c=0;c<nprocs;c++)
      sum+=cmsg[c];
    fprintf(f1,"Minimata pou stalikan kata to broadcast=%d\n",sum-psum);
  }



 if(simaia==1)
  sa=data;

 return sa;
}
```

```
/*                    PARALLILI ARTHROISI                          */

int parallel_sum(t_bsp* bsp, void* param,int i, int j,int inmul,int n,int r, int c)
{
 register int nprocs=bsp_nprocs(bsp);
 register int pid=bsp_pid(bsp);
 int buffer=0;
 int temp2,temp3,cou,cou1,ix,tsum,arrayex[nprocs];
 int result=0;
 int ex=-1;
 int sim=0;
 int minusone=-1;
 int counter=0;

 //ARXIKOPOIISI  PINAKA arrayex
 for(cou=0;cou<nprocs;cou++)
  arrayex[cou]=-1;

 //REMOTE MEMORY
 bsp_push_reg(bsp,&tsum,sizeof(int));
 bsp_push_reg(bsp,&ex,sizeof(int));
 bsp_push_reg(bsp,arrayex,sizeof(int));
 bsp_push_reg(bsp,&temp3,sizeof(int));
 bsp_push_reg(bsp,&counter,sizeof(int));
 bsp_push_reg(bsp,&result,sizeof(int));

 /* OI EPEKSERGASTES POU PREPEI NA PIASOUN KAPIO STOIXEIO APO TOUS PINAKES
EXOUN SIMAIA ISI ME 1 TIN OPOIA KANOUN GNOSTI STON EPEKSERGASTI 0, MESO TOU
PINAKA arrayex */

 if(i==r && j==c){
  sim=1;
  bsp_put(bsp,0,&sim,arrayex,pid*sizeof(int),sizeof(int));
  buffer=inmul;
  }
 bsp_sync( bsp );

 /* O EPEKSERGASTIS 0 DINEI MIA EXTRA TAFTOTITA STO KATHE ENA APO
 /* TOUS EPEKSERGASTES POU PREPEO NA PAROUN KAPOIO STOIXEIO, TIN
 /* OPOIA GNOSTOPOIOUN SE OLOUS TOUS EPEKSERGASTES MESO TOU arrayex */

 if (pid==0)
  for(cou=0;cou<nprocs;cou++)
   if(arrayex[cou]==1){
    bsp_put(bsp,cou,&counter,&ex,0,sizeof(int));
    for(cou1=0;cou1<nprocs;cou1++){
     bsp_put(bsp,cou1,&counter,arrayex,cou*sizeof(int),sizeof(int));
     }
    counter=counter+1;
    }
  bsp_sync( bsp );

 ix=0;
```

```
 temp2=2;

 /* TO counter STO TELOS THA KRATA TO PLITHOS TON EPEKSERGASTON POU    KRATOUN
DEDOMENA TA OPOIA THA XRIASTOUN STI ARTHROIS POU THA GINEI */

 if(pid==0)
  for(cou1=0;cou1<nprocs;cou1++){
   bsp_put(bsp,cou1,&counter,&temp3,0,sizeof(int));
   bsp_put(bsp,cou1,&counter,&counter,0,sizeof(int));
   }
  bsp_sync( bsp );
  tsum=0;

  /* PARALLILI ARTHROISI POU GINETAI APO TOUS n EPEKSERGASTES POU KRATOUSAN
DEDOMENA GIA ARTHROISI */
  while(temp2>1) {
    ix++;
    temp2=ceil(counter/pow(2,ix));
    if(sim==1){
     if(ex>=temp2 && ex<temp3){
      for(cou=0;cou<nprocs;cou++)
       if(arrayex[cou]==ex-temp2){
        bsp_put(bsp,cou,&buffer,&tsum,0,sizeof(int));
        }
      }
     }
    bsp_sync( bsp );

  if(sim==1){
   if(ex<temp2){
    buffer+=tsum;
    tsum=0;
    }
   temp3=temp2;
   }

  }
 bsp_sync( bsp );

 /* O EPEKSERGASTIS ME TIN extra TAFTOTITA 0 KRATA TO APOTELESMA TIS ARTHROISIS
KAI TO STELEI STO EPEKSERGASTI 0 */
 if(ex==0){
  result=buffer;
  bsp_put(bsp,0,&result,&result,0,sizeof(int));
  }
 bsp_sync( bsp );


 bsp_pop_reg(bsp,&tsum);
 bsp_pop_reg(bsp,&ex);
 bsp_pop_reg(bsp,arrayex);
 bsp_pop_reg(bsp,&temp3);
 bsp_pop_reg(bsp,&counter);
 bsp_pop_reg(bsp,&result);
```

```
  return result;
}



void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

int pid, nprocs;
doTests( bsp, argc, argv );
nprocs=bsp_nprocs(bsp);
pid=bsp_pid(bsp);

//TRIA STOIXEIA XARAXTIRISTIKA GIA KATHE EPEKSERGASTI
int k,i,j;
//DILWSI DIO PINAKWN MEGETHOUS nXn
int n=4;
int A[n][n],B[n][n],C[n][n];
int r,c,k1=1;
//PLITHOS EPEKSERGASTWN SE KATHE OMADA
int nopro=nprocs/n;
//METAVLITES GIA PROIGOUMENI OMADA KAI GIA OMADA POU EIMASTE TORA
int oldteam=0,nowteam=nopro;
//METRITIS OMADON- EXOUME TOSO PLITHOS OMADON OSO EINAI TO n
int teamc;
//METRITIS
int counter;
//STOIXEIA POU KRATA O KATHE EPEKSERGASTIS APO TOUS PINAKES
int *a,*b;
int sa=0;
int tempsa;
int sb=0;
int tempsb;
//STOIXEIA GIA TO broadcast
int i1;
t_bspmsg *msg;
//PINAKES POU KRATOUN PIOI EPEKSERGASTES PREPEI NA PAROUN STOIXEIO APO TO 0
KATHE STIGMI
int IDSA[nprocs];
int IDSB[nprocs];
int tca,tcb;
int simaib=0;
int simaia=0;
int one=1;
int ma;
int arrayextra[nprocs];
int arrayextrb[nprocs];
int cou;
int  datas, *data1s,is;
int prolistA[nprocs];
//COUNTER GIA  a
int ca=0;
//FLAGS
int fa=0;
```

```
int fb=0;

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA b
int prolistB[nprocs];
//COUNTER GIA b
int cb=0;
int temp;
int data=1;
int id=0;
int ta;
int atemp,btemp;

int extra;
int extrb;
int g,s;
float temps;
int inmul;
int datasb,*data1sb,isb;
int fla,flb;

//MONO O EPEKSERGASTIS ME pid=0 KRATA TOUS 3 PINAKES
if(pid==0)
{
  //ARXIKOPOIISI PINAKON A,B
  for(r=0;r<n;r++)
    for(c=0;c<n;c++)
    {
      A[r][c]=k1;
      B[r][c]=k1;
      C[r][c]=0;
      k1++;
    }
}
else

  //ARXIKOPOIISI PINAKON A,B
  for(r=0;r<n;r++)
    for(c=0;c<n;c++)
    {
      A[r][c]=0;
      B[r][c]=0;
      C[r][c]=0;
    }


//TOPOTHETISI TON EPEKSERGASTON SE OMADES,ME TIN EVRESI TOU k //GIA TON
KATHENA
for(teamc=1;teamc<=n;teamc++)
  {
    if(pid>=oldteam && pid<nowteam){
      k=teamc-1;

  //EVRESI TON i KAI j TON EPEKSERGASTON OI EPEKSERGASTES //PERNOUN AAFTA TA
STOIXEIA OPOS EINAI DOSMENA STON PINAKA
//ANALOGA ME TIN SEIRA TOUS px O 0 PERNI TO 1,1 ,o 2 to 1,2 ktl
```

```
      counter=0;
      for(r=0;r<n;r++)
        for(c=0;c<n;c++)
        {
          if(pid-((teamc-1)*nopro)==counter)
          {
            i=r;
            j=c;
          }
        counter++;
      }
    }

    oldteam=nowteam;
    nowteam=nowteam+nopro;

    }

//ALGORITHOS GIA POLLAPLASIASMO POU THA EXI broadcast KAI arthrisi

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA a
for(r=0;r<nprocs;r++)
    prolistA[r]=0;

//ARXIKOPOIISI LISTAS ME EPEKSERGASTES GIA b
for(r=0;r<nprocs;r++)
    prolistB[r]=0;

//ARXIKOPOIISI PINAKON IDSA KAI IDSB
for(r=0;r<nprocs;r++)
  {
    IDSA[r]=0;
    IDSB[r]=0;
  }

bsp_push_reg(bsp,IDSA,nprocs*sizeof(int));
bsp_push_reg(bsp,IDSB,nprocs*sizeof(int));


for(r=0;r<n;r++)
  {
    for(c=0;c<n;c++)
      {
        for(ta=0;ta<nprocs;ta++)
          {
            IDSA[ta]=0;
            prolistA[ta]=0;
            IDSB[ta]=0;
            prolistB[ta]=0;
          }

        if(i==r && k==c)
          {
            bsp_put(bsp,id,&data,IDSA,pid*sizeof(int),sizeof(int));
```

```
      fa=1;

     }

    if(j==c && k==r)
     {
      bsp_put(bsp,id,&data,IDSB,pid*sizeof(int),sizeof(int));
      fb=1;
     }

    bsp_sync(bsp);

    simaib=0;
    simaia=0;
//O EPEKSERGASTIS 0 PROSPATHEI NA VREI PIOI EPEKSERGASTES //THELOUN STOIXEIA
    extra=-1;
    extrb=-1;
    fla=0;
    flb=0;

    //GIA  TO 0 TO extra THA EINAI PANTA 0
    if(pid==0){
     extra=0;
     extrb=0;
     }

    //ARXIKOPOIISI PINAKON extra
    for(cou=0;cou<nprocs;cou++){
     arrayextra[cou]=0;
     arrayextrb[cou]=0;
     }


    bsp_push_reg(bsp,&extra,sizeof(int));
    bsp_push_reg(bsp,&simaia,sizeof(int));
    bsp_push_reg(bsp,&ca,sizeof(int));
    bsp_push_reg(bsp,arrayextra,sizeof(int));
    bsp_push_reg(bsp,&fla,sizeof(int));
    bsp_push_reg(bsp,&extrb,sizeof(int));
    bsp_push_reg(bsp,&simaib,sizeof(int));
    bsp_push_reg(bsp,&cb,sizeof(int));
    bsp_push_reg(bsp,arrayextrb,sizeof(int));
    bsp_push_reg(bsp,&flb,sizeof(int));

    if(pid==0 && IDSA[0]==1){
     fla=1;
     for(cou=0;cou<nprocs;cou++){
      bsp_put(bsp,cou,&fla,&fla,0,sizeof(int));
      }
     }
    if(pid==0 && IDSB[0]==1){
     flb=1;
     for(cou=0;cou<nprocs;cou++){
      bsp_put(bsp,cou,&flb,&flb,0,sizeof(int));
      }
```

```
  }
bsp_sync (bsp);

if(pid==0){
  ca=0;
  cb=0;
  for(temp=0;temp<nprocs;temp++){
     if(IDSA[temp]==1){
       if(temp!=0){
         prolistA[ca]=temp;
         if(fla==0)
           tca=ca+1;
         else
           tca=ca;
         bsp_put(bsp,temp,&tca,&extra,0,sizeof(int));

         for(cou=0;cou<nprocs;cou++){
           bsp_put(bsp,cou,&tca,arrayextra,temp*sizeof(int),sizeof(int));
           }
         ca++;
         }
       else {
         simaia=1;
         ca++;
         }
       }
    }


  for(temp=0;temp<nprocs;temp++){
    if(IDSB[temp]==1){
      if(temp!=0){
        prolistB[cb]=temp;
        if(flb==0)
          tcb=cb+1;
        else
          tcb=cb;
        bsp_put(bsp,temp,&tcb,&extrb,0,sizeof(int));

         for(cou=0;cou<nprocs;cou++){
           bsp_put(bsp,cou,&tcb,arrayextrb,temp*sizeof(int),sizeof(int));
           }
         cb++;
        }
        else {
         simaib=1;
         cb++;
         }
     }
    }
  }//TELOS IF
  bsp_sync (bsp);

if(pid==0)
   for(temp=0;temp<ca;temp++){
```

```
            bsp_put(bsp,prolistA[temp],&one,&simaia,0,sizeof(int));
            bsp_put(bsp,prolistA[temp],&ca,&ca,0,sizeof(int));
           }
           bsp_sync(bsp);

       if(pid==0)
         for(temp=0;temp<cb;temp++){
           bsp_put(bsp,prolistB[temp],&one,&simaib,0,sizeof(int));
           bsp_put(bsp,prolistB[temp],&cb,&cb,0,sizeof(int));
           }
           bsp_sync(bsp);

       //AN KAI TO pid 0 PREPI NA PAREI STOIXEIO
       if(simaia==1 && pid==0){
          sa=A[r][c];
          simaia=0;
          }

       if(simaib==1 && pid==0){
          sb=B[r][c];
          simaib=0;
          }

       datas=A[r][c];
       datasb=B[r][c];

  //OLI OI EPEKSERGASTES PREPI NA GNORIZOUN TO ca, PROSORINA TO
//ca EINAI 1
       if(simaia!=1 )
         ca=n;
       if(simaib!=1 )
         cb=n;

       tempsa= broadcast( bsp,param,datas,ca,simaia,extra,arrayextra);
       if(simaia==1)
         sa=tempsa;
       tempsb= broadcast( bsp,param,datasb,cb,simaib,extrb,arrayextrb);
       if(simaib==1)
         sb=tempsb;
   }
  }
bsp_sync(bsp);

//POLLAPLASIASMOS a ,b
inmul=sa*sb ;



//EKTIPOSI PINAKON A KAI B
if(pid==0){
printf("Pinakas A: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",A[r][c]);
    printf("\n");
```

```
    }
}

if(pid==0){
printf("Pinakas B: \n");
  for(r=0;r<n;r++){
    for(c=0;c<n;c++)
       printf(" %d ",B[r][c]);
    printf("\n");
    }
}

//IPOLOGISMOS PINAKA C KAI EKTIPOSI APO TON EPEKSERGASTI ME pid 0
if(pid==0)
printf("\nPinakas Apotelesmatvn apo ton polaplasiasmo ton pinakon A kai B:\n");
  for(r=0;r<n;r++)
    {
      for(c=0;c<n;c++)
       {
         C[r][c]=parallel_sum(bsp,param,i,j,inmul,n,r,c);
         if(pid==0)
           printf(" %d ",C[r][c]);
       }
      if(pid==0)
       printf("\n");
    }


bsp_pop_reg(bsp,IDSA);
bsp_pop_reg(bsp,&extra);
bsp_pop_reg(bsp,&simaia);
bsp_pop_reg(bsp,&ca);
bsp_pop_reg(bsp,arrayextra);
bsp_pop_reg(bsp,&fla);

bsp_pop_reg(bsp,IDSB);
bsp_pop_reg(bsp,&extrb);
bsp_pop_reg(bsp,&simaib);
bsp_pop_reg(bsp,&cb);
bsp_pop_reg(bsp,arrayextrb);
bsp_pop_reg(bsp,&flb);

bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
    t_bsp* bsp = mem_new( t_bsp, 1 );
    int i, vprocs;
    int pid, nprocs;
    double  time1,time2;
    fl = fopen ("out.lis", "wt");


    argc = _argc; argv = _argv;
```

```
   bsplib_saveargs( &argc, &argv );
   vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", 64 );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );


   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );


   time2=bsp_time(bsp);
   fprintf( f1,"\n-->O xronos pou xriastike einai %.6lf sec\n",time2-time1 );
   fclose (f1);

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

A.14 Αλγόριθμος για πολλαπλασιασμό πινάκων στο PREW

```
/* ============================================================ */
/*                                                              */
/*  Margarita Antonaki                                          */
/*  prew_mul.c                                                  */
/*  Pollaplasiasmos pinakon (n stin 3 epeksergastes)            */
/* Xrisimopoiontas to PREW   kai tin sinartisi gia parallili arthisi */
/*                                                              */
/* ============================================================ */

//lines 274

#include <debug.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <arguments.h>
#include <threads.h>
#include <pub.h>
#include <mem.h>
#include <string.h>
```

```
#define main oldmain
#include "testbsp.c"
#undef main

int argc;
char** argv;

int sum(t_bsp* bsp, void* param,int i, int j,int inmul,int n,int r, int c)
{
 register int nprocs=bsp_nprocs(bsp);
 register int pid=bsp_pid(bsp);

  //*************PARALLILI ARTHROISI*****************************
    int abtemp[nprocs];
    int buffer=0;
    int ix;
    int temp2,temp3;
    int tsum;
    int result=0;

    bsp_push_reg(bsp,abtemp,nprocs*sizeof(int));
    bsp_push_reg(bsp,&tsum,sizeof(int));


       if(i==r && j==c)
         buffer=inmul;

       ix=0;
       temp2=2;
       temp3=nprocs;
       tsum=0;



       while(temp2>1)
         {
          ix++;
          temp2=ceil(nprocs/pow(2,ix));
          if(pid>=temp2 && pid<temp3){
          bsp_put(bsp,pid-temp2,&buffer,&tsum,0,sizeof(int));
          }

         bsp_sync( bsp );

         if(pid<temp2){
           buffer+=tsum;
           tsum=0;
          }
         temp3=temp2;

        }

        if(pid==0)
             result= buffer;
```

```
bsp_pop_reg(bsp,abtemp);
bsp_pop_reg(bsp,&tsum);

return result;
}



void exclusive_main( t_bsp* bsp, void* param ) {

//DILWSI METAVLITWN

int pid, nprocs;
doTests( bsp, argc, argv );
nprocs=bsp_nprocs(bsp);
pid=bsp_pid(bsp);

//TRIA STOIXEIA XARAXTIRISTIKA GIA KATHE EPEKSERGASTI
int k,i,j;
//DILWSI DIO PINAKWN MEGETHOUS nXn
int n=4;
int A[n][n],B[n][n],C[n][n];
int r,c,k1=1;
//PLITHOS EPEKSERGASTWN SE KATHE OMADA
int nopro=nprocs/n;
//METAVLITES GIA PROIGOUMENI OMADA KAI GIA OMADA POU EIMASTE TORA
int oldteam=0,nowteam=nopro;
//METRITIS OMADON- EXOUME TOSO PLITHOS OMADON OSO EINAI TO n
int teamc;
//METRITIS
int counter;
//STOIXEIA POU KRATA O KATHE EPEKSERGASTIS APO TOUS PINAKES
int *a,*b;
int sa=0;
int sb=0;
//STOIXEIA GIA TO broadcast
int i1;

//MONO O EPEKSERGASTIS ME pid=0 KRATA TOUS 3 PINAKES
if(pid==0)
{
  //ARXIKOPOIISI PINAKON A,B
 for(r=0;r<n;r++)
   for(c=0;c<n;c++)
   {
     A[r][c]=k1;
     B[r][c]=k1;
     C[r][c]=0;
     k1++;
   }
}
```

```
//TOPOTHETISI TON EPEKSERGASTON SE OMADES,ME TIN EVRESI TOU k GIA TON KATHENA
for(teamc=1;teamc<=n;teamc++)
  {
   if(pid>=oldteam && pid<nowteam){
     k=teamc-1;

  //EVRESI TON i KAI j TON EPEKSERGASTON
 //OI EPEKSERGASTES PERNOUN AAFTA TA STOIXEIA OPOS EINAI //DOSMENA STON PINAKA
//ANALOGA ME TIN SEIRA TOUS px O 0 PERNI TO 1,1 ,o 2 to 1,2 ktl

     counter=0;
     for(r=0;r<n;r++)
       for(c=0;c<n;c++)
       {
         if(pid-((teamc-1)*nopro)==counter)
         {
           i=r;
           j=c;
         }
       counter++;
     }
   }

   oldteam=nowteam;
   nowteam=nowteam+nopro;

   }


//O KATHE EPEKSERGASTIS PERNI TO STOIXEIO POU XREIAZETAI ME //TAFTOXRONI
ANAGNOSI


int data,datab;
bsp_push_reg(bsp,&data,sizeof(int));
bsp_push_reg(bsp,&datab,sizeof(int));

int temp;
int id=0;

  for(r=0;r<n;r++)
    {

       for(c=0;c<n;c++)
       {

         if(pid==0){
           data=A[r][c];
           datab=B[r][c];
           }

       if(i==r && k==c){
           bsp_get(bsp,id,&data,0,&sa,sizeof(int));
           }
```

```
      if(j==c && k==r){
         bsp_get(bsp,id,&datab,0,&sb,sizeof(int));
         }


      bsp_sync(bsp);


      }
     }


     //POLLAPLASIASMOS a ,b
     int inmul;
     inmul=sa * sb;


      //PARALLILI ARTHROISI
      for(r=0;r<n;r++)
       {
        for(c=0;c<n;c++)
          {
           C[r][c]=sum(bsp,param,i,j,inmul,n,r,c);
          }
       }



       //EKTIPOSI PINAKON A KAI B
     if(pid==0){
       printf("Pinakas A: \n");
     for(r=0;r<n;r++){
       for(c=0;c<n;c++)
       printf(" %d ",A[r][c]);
     printf("\n");
     }
    }

    if(pid==0){
      printf("\nPinakas B: \n");
    for(r=0;r<n;r++){
      for(c=0;c<n;c++)
        printf(" %d ",B[r][c]);
    printf("\n");
    }
}


     //EKTIPOSI TOU PINAKA C APO TON 0
     if(pid==0){
     printf("\nPinakas C: \n");
        for(r=0;r<n;r++)
         {
          for(c=0;c<n;c++)
```

```
            {
              printf(" %d ",C[r][c]);
            }
           printf("\n");
         }
         }

        bsp_sync(bsp);

bsp_pop_reg(bsp,&data);
bsp_pop_reg(bsp,&datab);



bsp_threadsafe_done( bsp );
}

int main( int _argc, char** _argv ) {
   t_bsp* bsp = mem_new( t_bsp, 1 );
   int i, vprocs;
   int pid, nprocs;
   double  duration,time1,time2;



   argc = _argc; argv = _argv;

   bsplib_saveargs( &argc, &argv );
  vprocs = argupcaseoptint( argc, argv, "--vprocs=", 0 );
   if( vprocs==0 )
      vprocs = argupcaseoptint( argc, argv, "-vp", 64 );
   if( vprocs==0 )
      vprocs = 4;

   bsplib_init( BSPLIB_STDPARAMS, bsp );

   if( bsp_pid(bsp)== 0)
      printf( "creating %d virtual procs\n", vprocs );


   time1=bsp_time(bsp);
   bsp_exclusive_dup( bsp, vprocs, 1024*4096, exclusive_main, (void*) 0, true );


   time2=bsp_time(bsp);
   printf( "\n-->O xronos pou xriastike einai %.6lf seconds\n",time2-time1 );

 bsplib_done();
   free( bsp );
#ifdef MEMLOG
   checkMemLog();
#endif
   return 0;
}
```

Παράρτημα Β

Παρουσίαση δειγμάτων εκτέλεσης αλγορίθμων

Β1 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 1

Χρησιμοποιούνται 15 επεξεργαστές, και τελικά στην τοπική τους μεταβλητή data έχουν αποθηκευμένη την ταυτότητα του πρώτου επεξεργαστή, δηλαδή το 0. Οπόταν έχουμε μετάδοση της τιμής της τοπικής μεταβλητής data του επεξεργαστή 0.

```
Epeksergastis 1 data=0
Epeksergastis 2 data=0
Epeksergastis 3 data=0
Epeksergastis 4 data=0
Epeksergastis 5 data=0
Epeksergastis 6 data=0
Epeksergastis 7 data=0
Epeksergastis 8 data=0
Epeksergastis 9 data=0
Epeksergastis 10 data=0
Epeksergastis 11 data=0
Epeksergastis 12 data=0
Epeksergastis 13 data=0
Epeksergastis 14 data=0
Stalthikan 14 minimata sto ipervima!!

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.008660
defterolepta
```

Β2 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 2

Χρησιμοποιούνται 15 επεξεργαστές και η μεταβλήτη data είναι αυτή που φυλάει το δεδομένο που στέλλεται. Συγκεκριμένα στέλλεται η τοπική μεταβλητή data του επεξεργαστή 0. Πιο κάτω παρουσιάζονται σε κάθε υπερβήμα πιοί επεξεργαστές έχουν πάρει ήδη το δεδομένο και πόσα μηνύματα έχουν σταλλεί.

```
Stalthikan 1 minimata sto 1 ipervima
Epeksergastis 0 data=0
Epeksergastis 1 data=0

Stalthikan 2 minimata sto 2 ipervima
Epeksergastis 2 data=0
Epeksergastis 3 data=0

Stalthikan 4 minimata sto 3 ipervima
Epeksergastis 4 data=0
Epeksergastis 5 data=0
Epeksergastis 6 data=0
Epeksergastis 7 data=0

Stalthikan 7 minimata sto 4 ipervima
Epeksergastis 8 data=0
Epeksergastis 9 data=0
```

```
Epeksergastis 10 data=0
Epeksergastis 11 data=0
Epeksergastis 12 data=0
Epeksergastis 13 data=0
Epeksergastis 14 data=0

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.017614
defterolepta
```

Β3 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 3

Χρησιμοποιούνται 15 επεξεργαστές και η μεταβλήτη data είναι αυτή που φυλάει το δεδομένο που στέλλεται. Συγκεκριμένα στέλλεται η τοπική μεταβλητή data του επεξεργαστή 0. Πιο κάτω παρουσιάζονται σε κάθε υπερβήμα πιοί επεξεργαστές έχουν πάρει ήδη το δεδομένο και πόσα μηνύματα έχουν σταλλεί.

```
Stalthikan 1 minimata sto 1 ipervima
Epeksergastis 0 data=0
Epeksergastis 1 data=0

Stalthikan 2 minimata sto 2 ipervima
Epeksergastis 2 data=0
Epeksergastis 3 data=0

Stalthikan 4 minimata sto 3 ipervima
Epeksergastis 4 data=0
Epeksergastis 5 data=0
Epeksergastis 6 data=0
Epeksergastis 7 data=0

Stalthikan 7 minimata sto 4 ipervima
Epeksergastis 8 data=0
Epeksergastis 9 data=0
Epeksergastis 10 data=0
Epeksergastis 11 data=0
Epeksergastis 12 data=0
Epeksergastis 13 data=0
Epeksergastis 14 data=0

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.013124
defterolepta
```

Β4 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 1 (μεταφορά πίνακα - μεγαλύτερο μέγεθος μηνυμάτων)

Χρησιμοποιούνται 15 επεξεργαστές και το στοιχείο που στέλλεται είναι ένας μονοδιάστατος πίνακας μεγέθους 4 του επεξεργαστή με ταυτότητα 0. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν σε κάθε υπερβήμα και ο πίνακας που είναι

αποθηκευμένος στην τοπική μνήμη κάθε επεξεργαστή στο τέλος της εκτέλεσης του αλγορίθμου.

```
Epeksergastis 0 kai o arithmos minimaton pou estile einai 14

0 0 0 0
Epeksergastis 1 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 2 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 3 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 4 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 5 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 6 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 7 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 8 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 9 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 10 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 11 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 12 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 13 kai o arithmos minimaton pou estile einai 0

0 0 0 0
Epeksergastis 14 kai o arithmos minimaton pou estile einai 0

0 0 0 0

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.003069
defterolepta
```

Β5 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 2 (μεταφορά πίνακα - μεγαλύτερο μέγεθος μηνυμάτων)

Χρησιμοποιούνται 15 επεξεργαστές και το στοιχείο που στέλλεται είναι ένας μονοδιάστατος πίνακας μεγέθους 4 του επεξεργαστή με ταυτότητα 0. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν σε κάθε υπερβήμα και ο πίνακας που είναι αποθηκευμένος στην τοπική μνήμη κάθε επεξεργαστή στο τέλος της εκτέλεσης του αλγορίθμου.

```
Stalthikan 1 minimata sto 1 ipervima
Stalthikan 2 minimata sto 2 ipervima
Stalthikan 4 minimata sto 3 ipervima
Stalthikan 7 minimata sto 4 ipervima
Epeksergastis 0 Pinakas:
0 0 0 0
Epeksergastis 1 Pinakas:
0 0 0 0
Epeksergastis 2 Pinakas:
0 0 0 0
Epeksergastis 3 Pinakas:
0 0 0 0
Epeksergastis 4 Pinakas:
0 0 0 0
Epeksergastis 5 Pinakas:
0 0 0 0
Epeksergastis 6 Pinakas:
0 0 0 0
Epeksergastis 7 Pinakas:
0 0 0 0
Epeksergastis 8 Pinakas:
0 0 0 0
Epeksergastis 9 Pinakas:
0 0 0 0
Epeksergastis 10 Pinakas:
0 0 0 0
Epeksergastis 11 Pinakas:
0 0 0 0
Epeksergastis 12 Pinakas:
0 0 0 0
Epeksergastis 13 Pinakas:
0 0 0 0
Epeksergastis 14 Pinakas:
0 0 0 0

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.004755
defterolepta
```

Β6 Δείγμα εκτέλεσης αλγορίθμου για μετάδοση δεδομένων 3 (μεταφορά πίνακα - μεγαλύτερο μέγεθος μηνυμάτων)

Χρησιμοποιούνται 15 επεξεργαστές και το στοιχείο που στέλλεται είναι ένας μονοδιάστατος πίνακας μεγέθους 4 του επεξεργαστή με ταυτότητα 0. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν σε κάθε υπερβήμα και ο πίνακας που είναι

αποθηκευμένος στην τοπική μνήμη κάθε επεξεργαστή στο τέλος της εκτέλεσης του αλγορίθμου.

```
Stalthikan 1 minimata sto 1 ipervima
Stalthikan 2 minimata sto 2 ipervima
Stalthikan 4 minimata sto 3 ipervima
Stalthikan 7 minimata sto 4 ipervima

Epeksergastis 0 Pinakas:
0 0 0 0
Epeksergastis 1 Pinakas:
0 0 0 0
Epeksergastis 2 Pinakas:
0 0 0 0
Epeksergastis 3 Pinakas:
0 0 0 0
Epeksergastis 4 Pinakas:
0 0 0 0
Epeksergastis 5 Pinakas:
0 0 0 0
Epeksergastis 6 Pinakas:
0 0 0 0
Epeksergastis 7 Pinakas:
0 0 0 0
Epeksergastis 8 Pinakas:
0 0 0 0
Epeksergastis 9 Pinakas:
0 0 0 0
Epeksergastis 10 Pinakas:
0 0 0 0
Epeksergastis 11 Pinakas:
0 0 0 0
Epeksergastis 12 Pinakas:
0 0 0 0
Epeksergastis 13 Pinakas:
0 0 0 0
Epeksergastis 14 Pinakas:
0 0 0 0

-->O xronos pou xriastike o algorithos gia na ektelesti einai 0.005837
defterolepta
```

B7 Δείγμα εκτέλεσης αλγορίθμου προθεματικής άρθροισης σε τετραγωνικό δίχτυο αλληλοσύνδεσης

Χρησιμοποιήθηκαν 4 επεξεργαστές. Ο κάθε επεξεργαστής κρατά στο τέλος ένα πρόθεμα από το προθεματικό άρθροισμα και το εκτυπώνει. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν σε κάθε υπερβήμα και το πρόθεμα που αποθηκεύει στην τοπική του μνήμη κάθε επεξεργαστής.

```
1 Ipervima minimata: 2
2 Ipervima minimata: 1
3 Ipervima minimata: 1

Processor 0 me stoixeio 1
Processor 1 me stoixeio 3
Processor 2 me stoixeio 6
```

```
Processor 3 me stoixeio 10

-->O xronos pou xriastike einai 0.008594 defterolepta
```

Β8 Δείγμα εκτέλεσης αλγορίθμου παράλληλης άρθροισης σε τετραγωνικό δίχτυο αλληλοσύνδεσης

Χρησιμοποιήθηκαν 4 επεξεργαστές. Ο τελευταίος επεξεργαστής τυπώνει το αποτέλεσμα της άρθροισης. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν σε κάθε υπερβήμα και το τελικό άρθοισμα που αποθηκεύει στην τοπική του μνήμη ο επεξεργαστής με την μεγαλύτερη ταυτότητα.

```
1 Ipervima minimata: 2
2 Ipervima minimata: 1

Processor 3 To apotelesma tis artrhoisis einai 10

-->O xronos pou xriastike einai 0.004947 defterolepta
```

Β9 Δείγμα εκτέλεσης βέλτιστου αλγορίθμου παράλληλης άρθροισης

Χρησιμοποιήθηκαν 4 επεξεργαστές για 16 στοιχεία. Ο πρώτος επεξεργαστής τυπώνει το αποτέλεσμα της άρθροισης. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν και το τελικό άρθοισμα που αποθηκεύει στην τοπική του μνήμη ο επεξεργαστής με την μικρότερη ταυτότητα.

```
Pid 0: To arthoisma einai iso me 29.
To plithos minimaton pou stalikan einai 3.

-->O xronos pou xriastike einai 0.013222 sec
```

Β10 Δείγμα εκτέλεσης βέλτιστου αλγορίθμου προθεματικής άρθροισης

Χρησιμοποιήθηκαν 4 επεξεργαστές για 16 στοιχεία. Ο κάθε επεξεργαστής κρατά στο τέλος ένα πλήθος προθεμάτων από το προθεματικό άρθροισμα και το εκτυπώνει. Πιο κάτω παρουσιάζονται τα μηνύματα που στάλληκαν και το πρόθεμα που αποθηκεύει στην τοπική του μνήμη ο κάθε επεξεργαστής.

```
Epeksergastis 0: To prothema mou einai 0
Epeksergastis 0: To prothema mou einai 1
Epeksergastis 0: To prothema mou einai 3
Epeksergastis 0: To prothema mou einai 6
```

```
Epeksergastis 1: To prothema mou einai 10
Epeksergastis 1: To prothema mou einai 15
Epeksergastis 1: To prothema mou einai 21
Epeksergastis 1: To prothema mou einai 28
Epeksergastis 2: To prothema mou einai 36
Epeksergastis 2: To prothema mou einai 45
Epeksergastis 2: To prothema mou einai 55
Epeksergastis 2: To prothema mou einai 66
Epeksergastis 3 To prothema mou einai 78
Epeksergastis 3 To prothema mou einai 91
Epeksergastis 3 To prothema mou einai 105
Epeksergastis 3 To prothema mou einai 120

To plithos minimaton pou stalikan einai 8.

-->O xronos pou xriastike einai 0.005261 seconds
```

Β11 Δείγμα εκτέλεσης πρώτου αλγορίθμου για πολλαπλασιασμό πινάκων με αλγόριθμο μετάδοσης δεδομένων 1

Χρησιμοποιήθηκαν πίνακες 4x4 και επομένως 64 επεξεργαστές. Τα αποτελέσματα τυπώνονται από τον πρώτο επεξεργαστή.

Pinakas A:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas B:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas Apotelesmatvn apo ton polaplasiasmo ton pinakon A kai B:
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600

Β12 Δείγμα εκτέλεσης δεύτερου αλγορίθμου για πολλαπλασιασμό πινάκων με αλγόριθμο μετάδοσης δεδομένων 2

Χρησιμοποιήθηκαν πίνακες 2x2 και επομένως 8 επεξεργαστές. Τα αποτελέσματα τυπώνονται από τον πρώτο επεξεργαστή.

Pinakas A:
1 2
3 4

Pinakas B:
1 2
3 4

Pinakas Apotelesmaton apo ton polaplasiasmo ton pinakon A kai B:
7 10
15 22

Β13 Δείγμα εκτέλεσης τρίτου αλγορίθμου για πολλαπλασιασμό πινάκων με αλγόριθμο μετάδοσης δεδομένων 3

Χρησιμοποιήθηκαν πίνακες 4x4 και επομένως 64 επεξεργαστές. Τα αποτελέσματα τυπώνονται από τον πρώτο επεξεργαστή.

Pinakas A:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas B:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas Apotelesmatvn apo ton polaplasiasmo ton pinakon A kai B:

90 100 110 120

202 228 254 280
314 356 398 440
426 484 542 600

B14 Δείγμα εκτέλεσης τέταρτου αλγορίθμου για πολλαπλασιασμό πινάκων

Χρησιμοποιήθηκαν πίνακες 4x4 και επομένως 64 επεξεργαστές. Τα αποτελέσματα τυπώνονται από τον πρώτο επεξεργαστή.

Pinakas A:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas B:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Pinakas C:
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600

-->O xronos pou xriastike einai 0.460111 seconds

Παράρτημα Γ

Οδηγίες εγκατάστασης της βιβλιοθήκης PUB στο λειτουργικό σύστημα LINUX [10]

1. Μπορείτε να πάρετε την βιβλιοθήκη PUB από την ακόλουθη ιστοσελίδα http://www.uni-paderborn.de/ pub/download.html, κατεβάζοντας το αρχείο pub-xxx.tgz, όπου xxx αντιπροσωπεύει τον αριθμό της έκδοσης της βιβλιοθήκης.
2. Αποσυμπιέστε το αρχείο σε μορφή tar:

    - Όταν το αρχείο έχει την μορφή GNU-tar εκτελέστε την ακόλουθη εντολή: `tar xvfz pub-xxx.tgz` ή
    - Σε οποιαδήποτε άλλη μορφή εκτελέστε : `gunzip <pub-xxx.tgz | tar xfv -`

3. Ξεκινήστε την εγκατάσταση της βιβλιοθήκης PUB με την εντολή `./configure`

    - Κατά την διαδικασία του configure θα ερωτηθείτε για τις βιβλιοθήκες επικοινωνίας που επιθυμείτε να χρησιμοποιηθούν. Και έτσι η βιβλιοθήκη θα δημιουργηθεί σύμφωνα με την επικοινωνία που επιλέξατε. Οπόταν μπορείτε να ελέγξετε τα προγράμματα σας με την ακολουθιακή έκδοση και χρησιμοποιώντας εικονικούς επεξεργαστές . Αργότερα όμως υπάρχει και η δυνατότητα να μεταγλωττιστούν τα προγράμματα σύμφωνα την πραγματική παράλληλη έκδοση.
    - Επιλέξετε την μορφή (mode) της βιβλιοθήκης. Υπάρχουν τρείς διαθέσιμες μορφές :

        1. **release** Αποτελεί μια γρήγορη έκδοση της Pub, αλλά δεν περιέχει έλεγχο λαθών και αποσφαλμάτωση (debug) κώδικα.
        2. **debug** Περιέχει αποσφαλμάτωση κώδικα
        3. **profile** Δημιουργεί πληροφορίες για profile για το εργαλείο profiling tool

4. Ξεκινήστε την μεταγλώττιση για την βιβλιοθήκη PUB.