



© ΕΑΠ, 2009

Η παρούσα διατριβή, η οποία εκπονήθηκε στα πλαίσια της ΘΕ ΠΛΗ40, και τα λοιπά αποτελέσματα της αντίστοιχης Πτυχιακής Εργασίας (ΠΕ) αποτελούν συνιδιοκτησία του ΕΑΠ και του φοιτητή, ο καθένας από τους οποίους έχει το δικαίωμα ανεξάρτητης χρήσης και αναπαραγωγής τους (στο σύνολο ή τμηματικά) για διδακτικούς και ερευνητικούς σκοπούς, σε κάθε περίπτωση αναφέροντας τον τίτλο και το συγγραφέα και το ΕΑΠ όπου εκπονήθηκε η ΠΕ καθώς και τον επιβλέποντα και την επιτροπή κρίσης.



Τίτλος Διπλωματικής Εργασίας

Σχεδιασμός, Ανάπτυξη & Αξιολόγηση Παράλληλων Αλγορίθμων Επιστημονικού
Υπολογισμού σε Περιβάλλον Συστοιχίας Υπολογιστών Υψηλής Απόδοσης

Κρεμμύδας Δημήτριος

Μάμαλης Βασίλειος

Επ. Καθηγητής

ΤΕΙ Αθήνας

Σκόδρας Αθανάσιος

Καθηγητής

ΕΑΠ

Αλεφραγκής Παναγιώτης

Πανεπιστήμιο Πατρών

ΤΕΙ Μεσολογγίου

Περίληψη: Υλοποιήθηκε παράλληλα – με την χρήση της βιβλιοθήκης MPI (Message Passing Interface) και γλώσσας C – ο αλγόριθμος επίλυσης προβλημάτων γραμμικού προγραμματισμού «simplex» και πιο συγκεκριμένα η μέθοδος simplex της «πλήρους πινακοειδούς μορφής» (full tableau method). Επιλύσαμε 19 προβλήματα από το NETLIB και 4 τυχαία, διαφόρων μεγεθών στο εργαστήριο παράλληλης επεξεργασίας του Τμ. Πληροφορικής του ΤΕΙ Αθήνας (linux cluster με δικτυακή διασύνδεση τύπου Myrinet). Έγιναν εκτενές πειραματικές μετρήσεις όσον αφορά στους χρόνους εκτέλεσης υπολογισμών και υπολογίστηκε/μελετήθηκε η επιτάχυνση (speed-up) που επιφέρει η αύξηση του αριθμού των διεργαστών-επεξεργαστών που συμμετέχουν στην επίλυση. Ο σχεδιασμός και υλοποίηση έγινε κατόπιν εκτενούς διερεύνησης της βιβλιογραφίας και μελέτης αντίστοιχων υλοποίησεων, ενώ τα αποτελέσματα-μετρήσεις που καταγράφηκαν (επιτάχυνση κλπ) ήταν ιδιαίτερα ικανοποιητικά, ειδικότερα για προβλήματα μεγάλου μεγέθους.

Λέξεις-κλειδιά: Γραμμικός Προγραμματισμός, Παράλληλη Επεξεργασία, Simplex, MPI



Design, Implementation and Evaluation of Parallel Algorithms for Scientific Computing on a High-performance Cluster Environment

Kremmydas Dimitrios

Mamalis Vasilios
Ass. Professor
TEI of Athens

Skodras Athanasios
Professor
HOU

Alefragkis Panagiotis
University of Patras
TEI of Mesologgi

Abstract: We have implemented a parallel full tableau simplex solver for linear programming problems using MPI and C. We run the solver at the parallel processing laboratory of the Dept. of Informatics of the Technological University of Athens (a linux cluster with eight XEON hyperthreading processors and Myrinet-based interconnection network 2GB) for 19 NETLIB and 4 random problems. We have taken measurements for the computation times and calculated the final speedup of the problem for solving it in parallel. The results (speed-up etc.) were quite satisfactory, especially for large-size linear programming problems.

Keywords: Linear Programming, Parallel Processing, Simplex, MPI



Κρεμμύδας Δημήτριος, Σχεδιασμός, Ανάπτυξη & Αξιολόγηση Παράλληλων
Αλγορίθμων Επιστημονικού Υπολογισμού σε Περιβάλλον Συστοιχίας
Υπολογιστών Υψηλής Απόδοσης



Περιεχόμενα

| | | |
|-------|---|----|
| 1 | Εισαγωγή | 9 |
| 2 | Υπόβαθρο | 12 |
| 2.1 | Γραμμικός Προγραμματισμός και η μέθοδος Simplex..... | 12 |
| 2.1.1 | Εισαγωγικά | 12 |
| 2.1.2 | Το Γραμμικό Πρόβλημα..... | 12 |
| 2.1.3 | Η μέθοδος Simplex | 16 |
| 2.2 | Παράλληλος Υπολογισμός και βιβλιοθήκη MPI | 36 |
| 2.2.1 | Βασικές αρχές Παράλληλου Υπολογισμού | 36 |
| 2.2.2 | Η βιβλιοθήκη MPI | 42 |
| 2.2.3 | Η βιβλιοθήκη MPE | 50 |
| 3 | Σχεδιασμός και Υλοποίηση | 51 |
| 3.1 | Σχετικές Εργασίες | 51 |
| 3.1.1 | Άρθρο: Towards a practical parallelisation of the simplex method, J. Hall | 51 |
| 3.1.2 | Άρθρο: A Distribute, Scaleable Simplex Method, G. Yarmish & R. V. Slyke | 52 |
| 3.1.3 | Παρουσίαση σε Συνέδριο: Some Computational Results on MPI Parallel Implementation of Dense Simplex Method, El-Said Badr et al. | 53 |
| 3.2 | Παραλληλοποίηση της μεθόδου Simplex | 54 |
| 3.2.1 | Διαμοιρασμός των δεδομένων | 54 |
| 3.2.2 | Ο Αλγόριθμος | 56 |
| 3.3 | Υλοποίηση και Σχεδιαστικές Επιλογές..... | 58 |
| 3.3.1 | Περιβάλλον υλοποίησης | 58 |



| | | |
|-------|--|----|
| 3.3.2 | Περιβάλλον εκτέλεσης..... | 59 |
| 3.3.3 | Τα δεδομένα εισόδου | 59 |
| 3.3.4 | Οργάνωση των αρχείων (modules)..... | 60 |
| 3.3.5 | Το εκτελέσιμο αρχείο | 65 |
| 3.4 | Δυνατότητες βελτίωσης του τρέχοντος αλγορίθμου..... | 65 |
| 3.4.1 | Υπολογισμός της εξερχόμενης μεταβλητής | 65 |
| 3.4.2 | Προσθήκη κριτηρίων επιλογής εισερχόμενης μεταβλητής | 66 |
| 3.4.3 | Εφαρμογή προλυτικών (presolving) εργασιών..... | 66 |
| 3.4.4 | Αποτελεσματικότερη διαχείριση της φόρτωσης και διαμοιρασμού των αρχικών δεδομένων..... | 66 |
| 4 | Πειραματική Αξιολόγηση..... | 68 |
| 4.1 | Περιγραφή της Πειραματικής Διαδικασίας..... | 68 |
| 4.1.1 | Εκτέλεση πειράματος..... | 68 |
| 4.1.2 | Επεξεργασία αποτελεσμάτων | 72 |
| 4.2 | Παρουσίαση αποτελεσμάτων..... | 75 |
| 4.2.1 | Μέσος Όρος Επιτάχυνσης ανά Αριθμό Διεργασιών που συμμετείχαν στην λύση του προβλήματος | 75 |
| 4.2.2 | Χρόνος Εκτέλεσης Υπολογισμών..... | 82 |
| 4.2.3 | Μέσος Χρόνος Υπολογισμών ανά επανάληψη | 84 |
| 4.2.4 | Χρόνοι εκτέλεσης των βημάτων του αλγορίθμου Simplex | 85 |
| 4.3 | Συμπεράσματα | 90 |
| 5 | Παραρτήματα..... | 91 |
| 5.1 | Βιβλιογραφία..... | 91 |
| 5.2 | Πηγαίος Κώδικας | 93 |
| 5.2.1 | mpi_simplex_v2.h..... | 94 |



| | | |
|-------|------------------------------|-----|
| 5.2.2 | mpi_simplex_v2_main.c..... | 99 |
| 5.2.3 | mpi_load_mps.c..... | 108 |
| 5.2.4 | mpi_distr_lp_data.c..... | 117 |
| 5.2.5 | mpi_simplex.c..... | 128 |
| 5.2.6 | mpi_simplex_profiling.c..... | 146 |
| 5.2.7 | mpi_report.c..... | 148 |



1 Εισαγωγή

Θα μπορούσαμε να πούμε ότι η επίλυση των γραμμικών προβλημάτων με την μέθοδο simplex στηρίζεται σε τρείς πυλώνες: τον καθαρά θεωρητικό, τον αλγορίθμικό και τον εφαρμοσμένο. Ο καθένας από αυτούς εστιάζει σε διαφορετικό επίπεδο αφαίρεσης.

Στον πρώτο πυλώνα μπορούμε να εντάξουμε την μαθηματική θεωρία που στηρίζει και εξηγεί το πώς και το γιατί ο αλγόριθμος λύνει επιτυχώς τα προβλήματα. Σε αυτόν τον πυλώνα εντάσσονται ακόμα και οι διαφορετικές μαθηματικές οπτικές γωνίες μέσω των οποίων μπορεί να απεικονισθεί η simplex (π.χ. με την μορφή διανυσμάτων, γεωμετρική ερμηνεία κ.α.).

Ο δεύτερος πυλώνας, κάνοντας ένα βήμα πιο κοντά προς την πραγματικότητα, παρουσιάζει με ψευδοκώδικα και αναλύει με όρους αλγορίθμικής πολυπλοκότητας την αποδοτικότητα της μεθόδου και των επιμέρους παραλλαγών της. Σε αυτή η προσέγγιση δεν μας απασχολεί η μορφή εισόδου του προβλήματος, ούτε οι διαθέσιμες δομές δεδομένων ή οι διαθέσιμες συναρτήσεις.

Τέλος, ο τρίτος πυλώνας, ασχολείται με την υλοποίηση της μεθόδου σε κάποια γλώσσα προγραμματισμού. Είναι η πλέον εφαρμοσμένη προσέγγιση. Εδώ είναι δυσδιάκριτο το μαθηματικό υπόβαθρο καθώς μας απασχολεί αποκλειστικά το πρόβλημα της υλοποίησης του δεύτερου πυλώνα..

Στα πλαίσια αυτής εδώ της διπλωματικής δώσαμε έμφαση στον τελευταίο πυλώνα, χωρίς ωστόσο να παραγνωρίζουμε την σπουδαιότητα και την βαρύτητα των πρώτων δύο. Εστιάσαμε στην μέθοδο simplex για την επίλυση προβλημάτων γραμμικού προγραμματισμού και προσπαθήσαμε να καταδείξουμε την επιτάχυνση στους χρόνους εκτέλεσης που μπορεί να φέρει η παράλληλη επεξεργασία στην υλοποίηση της μεθόδου.

Η μέθοδος Simplex είναι ένας άπληστος αλγόριθμος που επινοήθηκε από τον G.Dantzig για την επίλυση γραμμικών προβλημάτων και δημοσιεύθηκε το 1947. Το παράδειγμα το οποίο χρησιμοποίησε ο Dantzig για να καταδείξει την χρησιμότητα της μεθόδου ήταν ο βέλτιστος επιμερισμός 70 εργασιών σε 70 διαφορετικούς υπαλλήλους. Ο αριθμός όλων των πιθανών



συνδυασμών είναι 70!, αριθμός αστρονομικός. Η λύση με απλή αναζήτηση, σύμφωνα με υπολογισμούς του, δεν θα είχε βρεθεί ακόμα κι αν ο υπολογιστής ξεκινούσε την επίλυση της συγχρόνως με το big bang, 15 εκατομμύρια χρόνια πριν την εποχή του. Αντίθετα η λύση του παραπάνω προβλήματος με την μέθοδο που πρότεινε ήταν σχεδόν στιγματία, ακόμα και με τις υπολογιστικές δυνατότητες των μηχανημάτων εκείνης της εποχής [1].

Ο αλγόριθμος simplex από την στιγμή της ανακάλυψης του χρησιμοποιήθηκε ευρέως για να λύσει προβλήματα γραμμικής φύσης. Το περιοδικό Computing in Science and Engineering τον συμπεριέλαβε μέσα στους 10 σημαντικότερους αλγορίθμους του 20^{ου} αιώνα [2]. Στο πέρασμα του χρόνου αναπτύχθηκαν διάφορες υλοποιήσεις της simplex, με πιο ευρέως γνωστή την αναθεωρημένη έκδοση (revised simplex method). Ωστόσο εμείς θα ασχοληθούμε με την αρχική έκδοση του αλγορίθμου που ονομάζεται μέθοδος του πλήρους πίνακα (full tableau method).

Ο τρόπος με τον οποίο δουλεύει ο αλγόριθμος μπορεί πολύ συνοπτικά να περιγραφεί ως εξής: Η κάθε διεργασία επιλέγει την τοπικά καλύτερη εισερχόμενη μεταβλητή με κριτήριο τον αρνητικότερο συντελεστή στην αντικειμενική συνάρτηση (κριτήριο Dantzig). Με καθολική μείωση (MPI_Reduce_All), όλες οι διεργασίες γνωρίζουν την στήλη με τον καθολικά αρνητικότερο συντελεστή και σε ποια διεργασία αυτή βρίσκεται Γίνεται μετάδοση (Bcast) της στήλης από την διεργασία που την κατέχει προς τις υπόλοιπες. Η διεργασία με την καθολικά εισερχόμενη μεταβλητή (δηλαδή την επιλεχθείσα στήλη) υπολογίζει την εξερχόμενη μεταβλητή με το κριτήριο του ελάχιστου λόγου και μεταδίδει τον αριθμό αυτής της γραμμής στις υπόλοιπες Όλες οι διεργασίες εκτελούν τους υπολογισμούς για το τμήμα του πίνακα που τους αναλογεί ώστε να εκτελεστεί η περιστροφή (pivot). Τα παραπάνω βήματα επαναλαμβάνονται μέχρι, είτε να μην υπάρχει αρνητικός συντελεστής στην αντικειμενική συνάρτηση (άριστο σημείο), είτε να μην μπορεί να βρεθεί εξερχόμενη μεταβλητή (μη φραγμένο πρόβλημα)

Το πείραμα εκτελέστηκε στο εργαστήριο παράλληλης επεξεργασίας του Τμ. Πληροφορικής του ΤΕΙ Αθήνας, το οποίο αποτελείται από μία συστοιχία (cluster) 8 υπολογιστών σε δίκτυο Myrinet. Εκτελέστηκαν συνολικά επτά τρεξίματα για κάθε περίπτωση ώστε τα τελικά αποτελέσματα να είναι στατιστικά σημαντικά και αξιόπιστα. Το λειτουργικό σύστημα υποστήριξης ήταν linux προσαρμοσμένης διανομής, ενώ οι οκτώ επεξεργαστές ήταν όλοι τύπου XEON με δυνατότητα hyperthreading.



Σε κάθε τρέξιμο εκτελέστηκαν 21 προβλήματα σε 1,2,3,4,5,6,7,8,16 διεργασίες και για 1 πρόβλημα σε 8 και 16 διεργασίες. Για το τελευταίο επιλέχθηκε αυτός ο συνδυασμός διεργασιών γιατί ο χρόνος εκτέλεσης με λιγότερες διεργασίες ήταν πρακτικά δύσκολος (χρόνος εκτέλεσης με 8 διεργασίες ~3200 δευτερόλεπτα).

Τα μετρικά τα οποία χρησιμοποιήθηκαν για να εκτιμήσουν την αποτελεσματικότητα της παράλληλης εκτέλεσης ήταν:

1. **Επιτάχυνση(n)** = (Χρόνος εκτέλεσης υπολογισμών με 1 διεργασία) / (Χρόνος εκτέλεσης υπολογισμών με n διεργασίες)
2. **Μέσος χρόνος υπολογισμών ανά επανάληψη**: Χρόνος εκτέλεσης υπολογισμών / (Επαναλήψεις Φάσης I + Επαναλήψεις Φάσης II)
3. **Χρόνος εκτέλεσης υπολογισμών**: Συνολικός χρόνος εκτέλεσης για την επίλυση των Φάσεων I και II

Επίσης έγινε 1 ακόμα τρέξιμο των 22 προβλημάτων σε 8 διεργασίες, στο οποίο με την χρήση της MPE βιβλιοθήκης καταγράφηκαν οι χρόνοι που η κάθε διεργασία εκτελούσε κάποια φάση του αλγορίθμου. Τα αποτελέσματα αυτά παρουσιάζονται γραφικά και συσχετίζονται περιγραφικά με τα παραπάνω μετρικά.

Τα αποτελέσματα κατέδειξαν ότι υπάρχει σχεδόν γραμμική με τον αριθμό των διεργασιών επιτάχυνση (speed-up) για προβλήματα σχετικά μεγάλα. Επίσης, η κλιμάκωση της επιτάχυνσης αρχίζοντας από προβλήματα με μεσαία μεγέθη και πηγαίνοντας προς προβλήματα μεγάλου μεγέθους είναι πολύ ικανοποιητική και οφείλεται σε μεγάλο βαθμό στην απόδοση της Myrinet διασύνδεσης. Γενικά, τα αποτελέσματα-μετρήσεις κρίνονται ιδιαίτερα ικανοποιητικά και στην πλειοψηφία τους ανάλογης έκτασης των αναγνωρισμένων δημοσιευμένων εργασιών της βιβλιογραφίας.



2 Υπόβαθρο

2.1 Γραμμικός Προγραμματισμός και η μέθοδος Simplex

2.1.1 Εισαγωγικά

Τα προβλήματα αριστοποίησης μίας συνάρτησης, δεδομένων κάποιων περιορισμών, συναντώνται πολύ συχνά σε πολλούς εφαρμοσμένους επιστημονικούς κλάδους. Για παράδειγμα στο ευρύτερο πεδίο των Οικονομικών, υπάρχει πλήθος από προβλήματα για τον προγραμματισμό της παραγωγής και την κατανομή των πόρων σε οποιουδήποτε είδους δραστηριότητα, τον προγραμματισμό μίας διαφημιστικής καμπάνιας, την διαχείριση των δρομολογίων ενός στόλου φορτηγών, την διαχείριση χρηματο-οικονομικών προϊόντων κλπ.

Ο γραμμικός προγραμματισμός με τον οποίο θα ασχοληθούμε σε αυτό εδώ το κεφάλαιο ασχολείται με το υποσύνολο των προβλημάτων εκείνων όπου τόσο η συνάρτηση προς αριστοποίηση (αντικειμενική) όσο και οι περιορισμοί είναι εξισώσεις ή ανισώσεις πρώτου βαθμού. [1].

2.1.2 Το Γραμμικό Πρόβλημα

2.1.2.1 Τα κύρια μέρη ενός Γραμμικού Προβλήματος

Ο γραμμικός προγραμματισμός ασχολείται με την αριστοποίηση -μεγιστοποίηση ή ελαχιστοποίηση- μίας συνάρτησης, δεδομένων κάποιων περιορισμών. Τόσο η συνάρτηση όσο και οι περιορισμοί θα πρέπει να χαρακτηρίζονται από γραμμικότητα ως προς τις μεταβλητές του προβλήματος.

Τα κύρια μέρη τα οποία απαρτίζουν ένα γραμμικό πρόβλημα είναι [3]:

Οι μεταβλητές επιλογής (variables, decision variables): Αναπαριστούν τις ποσότητες του προβλήματος που μπορούμε να μεταβάλλουμε.

Η αντικειμενική συνάρτηση: Είναι η σχέση που συνδέει τις μεταβλητές μας με τον στόχο μας. Εκφράζει την σχέση προς αριστοποίηση.

Οι περιορισμοί: Είναι σχέσεις που συνδέουν τις μεταβλητές με πιθανούς περιορισμούς στις τιμές τους ή στον συνδυασμό αυτών.



Η γενική μορφή ενός γραμμικού προβλήματος είναι:

Ελαχιστοποίησε ή Μεγιστοποίησε την: $Z = c_1x_1 + c_2x_2 + \dots + c_nx_n$

Υπό τους περιορισμούς: $a_{11}x_1 \pm a_{12}x_2 \pm \dots \pm a_{1n}x_n \{ \leq, \geq \} r_1$

$$a_{21}x_1 \pm a_{22}x_2 \pm \dots \pm a_{2n}x_n \{ \leq, \geq \} r_2$$

$$\vdots \quad \vdots \quad \vdots$$

$$a_{m1}x_1 \pm a_{m2}x_2 \pm \dots \pm a_{mn}x_n \{ \leq, \geq \} r_m$$

$$x_1, x_2, \dots, x_n \geq 0$$

Αν θελήσουμε να εκφράσουμε τις παραπάνω εξισώσεις σε μορφή πίνακα, θα ορίσουμε αρχικά τους παρακάτω πίνακες:

$$C \equiv \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad X \equiv \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad A \equiv \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad R \equiv \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}$$

και η αναπαράσταση του προβλήματος γίνεται:

Ελαχιστοποίησε ή Μεγιστοποίησε την $Z = C' \times X$

$$A \times X \{ \leq, \geq \} R$$

Υπό τους περιορισμούς: $x_j \geq 0$

2.1.2.2 To Δυϊκό πρόβλημα

Σε οποιοδήποτε γραμμικό πρόβλημα μεγιστοποίησης ή ελαχιστοποίησης υπάρχει το ισοδύναμο του πρόβλημα ελαχιστοποίησης ή μεγιστοποίησης. Τα δεξιά μέρη των περιορισμών του πρωτογενούς (primal) προβλήματος γίνονται συντελεστές της αντικειμενικής συνάρτησης του δυαδικού (dual) προβλήματος, ενώ οι περιορισμοί γίνονται μεταβλητές και οι μεταβλητές περιορισμοί.

Για παράδειγμα, για το γραμμικό πρόβλημα

Μεγιστοποίησε: $Z = 3x_1 + 4x_2 + 3x_3$



$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 12 \\ 42 \end{bmatrix}$$

Υπό τους περιορισμούς:

$$\text{και } x_1, x_2, x_3 \geq 0$$

Αντιστοιχεί το εξής πρόβλημα ελαχιστοποίησης:

$$\text{Ελαχιστοποίησε: } Z = 12y_1 + 42y_2$$

$$\begin{bmatrix} 1 & 2 \\ 1 & 4 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 3 \\ 4 \\ 3 \end{bmatrix}$$

Υπό τους περιορισμούς:

$$\text{και } x_1, x_2, x_3 \geq 0$$

Η ιδιότητα της δυϊκότητας των γραμμικών προβλημάτων μπορεί να αξιοποιηθεί σε υπολογιστικό επίπεδο αφού μπορούμε να την εκμεταλλευτούμε ώστε να μειώσουμε τον αριθμό των περιορισμών και τελικά τον χρόνο επίλυσης.

2.1.2.3 Ένα απλό παράδειγμα γραμμικού προβλήματος

Πριν προχωρήσουμε στην θεωρητική ανάλυση των Γραμμικών προβλημάτων, θεωρούμε σκόπιμο να παρουσιάσουμε ένα τέτοιο απλό και πρακτικό πρόβλημα από το βιβλίο του Alpha C. Chiang [4].

Έστω ότι για να διατηρήσει κάποιος την υγεία του θα πρέπει να καταναλώνει κάποιες ελάχιστες ποσότητες των τριών παρακάτω συστατικών: ασβέστιο, πρωτεΐνη και βιταμίνη. Έστω επίσης ότι η διατροφή κάποιου αποτελείται από δύο τρόφιμα: Τροφή A και Τροφή B. Στον παρακάτω πίνακα φαίνεται το κόστος των δύο τροφίμων καθώς και η περιεκτικότητα τους στα επιμέρους συστατικά ψ. Παρατίθεται επίσης και οι ελάχιστες καθημερινές ανάγκες κάποιου στα απαραίτητα συστατικά.

Πίνακας 1, Τιμές και περιεχόμενα Τροφών παραδείγματος

| Τροφή ^(ανά κιλό) | I | Τροφή ^(ανά κιλό) | II | |
|-----------------------------|------|-----------------------------|----|-------------------------------|
| Τιμές | 0,60 | 1,00 | | Ελάχιστες καθημερινές ανάγκες |



| | | | |
|-----------------------|----|---|----|
| Ασβέστιο (μονάδα *) | 10 | 4 | 20 |
| Πρωτεΐνη (μονάδα *) | 5 | 5 | 20 |
| Βιταμίνη A (μονάδα *) | 2 | 6 | 12 |

* Χρησιμοποιήσαμε υποθετικές μονάδες για να επιτρέψουμε τη χρηση ακεραίων αριθμών στο παράδειγμα

Ένα πρόβλημα γραμμικού προγραμματισμού μπορεί να είναι: Ποιος ο συνδυασμός των δύο προϊόντων που θα ικανοποιεί το καθημερινό διαιτολόγιο και θα συνεπάγεται ελάχιστο κόστος;

Η αντικειμενική συνάρτηση είναι η σχέση εκείνη η οποία καθορίζει το συνολικό κόστος αγοράς. Οι μεταβλητές είναι οι ποσότητες των προϊόντων A και B. Οι περιορισμοί είναι οι σχέσεις εκείνες που καθορίζουν τις ελάχιστες απαιτήσεις σε συστατικά. Πιο συγκεκριμένα:

$$\text{Ελαχιστοποίηση: } Z = 0,6x_1 + x_2 \text{ [συνάρτηση κόστους]}$$

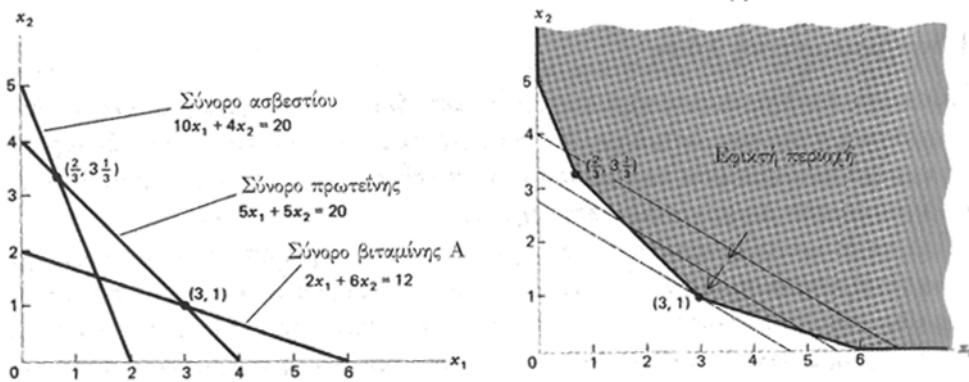
$$\text{Υπό τους περιορισμούς: } 10x_1 + 4x_2 \geq 20 \text{ [σύνορο ασβεστίου]}$$

$$5x_1 + 5x_2 \geq 20 \quad [\text{σύνορο πρωτεΐνης}]$$

$$2x_1 + 6x_2 \geq 12 \quad [\text{σύνορο βιταμίνης A}]$$

2.1.2.4 Διαγραμματική Επίλυση του παραδείγματος

Εφόσον στο πρόβλημα μας έχουμε δύο μεταβλητές απόφασης μπορούμε να το αναπαραστήσουμε και να το επιλύσουμε διαγραμματικά στην εικόνα 1.





Εικόνα 1, Διαγραμματική επίλυση Γραμμικού προβλήματος

Μπορούμε να παραστήσουμε τους τρείς περιορισμούς σε σχέση με τις τιμές των μεταβλητών απόφασης. Επίσης, αφού οι μεταβλητές αυτές μπορούν να πάρουν τιμές μόνο μεγαλύτερες αυτών των περιορισμών μπορούμε και να σχεδιάσουμε την περιοχή (εφικτή περιοχή) μέσα στην οποία επιτρέπεται να παίρνει τιμές το ζεύγος (x_1, x_2) (εφικτές λύσεις).

Επίσης στο ίδιο διάγραμμα φαίνονται και οι γραμμές του κόστους (της αντικειμενικής συνάρτησης). Όσο πιο κοντά προς την αρχή των αξόνων είναι μία τέτοια γραμμή τόσο μικρότερο κόστος μας δίνει ο συνδυασμός των (x_1, x_2) .

Συνδυάζοντας τα δύο παραπάνω στοιχεία (εφικτή περιοχή, γραμμές της αντικειμενικής συνάρτησης) μπορούμε βρούμε το σημείο το οποίο ικανοποιεί ταυτόχρονα τους περιορισμούς και ελαχιστοποιεί την συνάρτηση κόστους.

2.1.3 Η μέθοδος Simplex

2.1.3.1 Εισαγωγικά

Η μέθοδος Simplex είναι ένας άπληστος αλγόριθμος που επινοήθηκε από τον G.Dantzig για την επίλυση τέτοιου είδους γραμμικών προβλημάτων και δημοσιεύθηκε το 1947. Η μέση αλγορίθμική πολυπλοκότητα του αλγορίθμου είναι πολυωνυμική, ωστόσο η χειρότερη περίπτωση μας δίνει εκθετική πολυπλοκότητα.

Ο αλγόριθμος αυτός στηρίζεται στις εξής βασικές διαπιστώσεις για την φύση των προβλημάτων του γραμμικού προγραμματισμού:

Οι εφικτές λύσεις ενός γραμμικού προβλήματος n -μεταβλητών σχηματίζουν ένα κυρτό πολύγωνο στον n -χώρο

Η άριστη λύση του προβλήματος (ελάχιστη ή μέγιστη) βρίσκεται σε κάποια άκρη αυτού του πολυγώνου

Για την αντικειμενική συνάρτηση, κάθε τοπικό ελάχιστο ή μέγιστο είναι και ολικό.

Άρα η άριστη λύση του προβλήματος θα βρίσκεται σε κάποιο εξωτερικό σημείο του εφικτού χώρου, έτσι όπως αυτός καθορίζεται από το σύνολο των περιορισμών, και συγκεκριμένα σε κάποια τομή των υπέρ-επιπέδων που καθορίζουν οι επιμέρους περιορισμού. Ο αλγόριθμος simplex λοιπόν αναζητάει σε διαδοχικά γειτονικά τέτοια εξωτερικά σημεία την άριστη λύση του προβλήματος.



2.1.3.2 Υλοποιήσεις της μεθόδου Simplex

Υπάρχουν διάφορες παραλλαγές που εμπίπτουν στην κατηγορία των αλγορίθμων simplex. Πέραν της αρχικής μεθόδου του πλήρους πίνακα, υπάρχει η αναθεωρημένη μέθοδος (revised method), η μέθοδος Bartels – Golub, η μέθοδος Forrest-Tomlin, κ.α.

Αυτή με την οποία θα ασχοληθούμε σε αυτή εδώ την εργασία είναι η πλήρης πινακοειδής μορφή του αλγορίθμου που προτάθηκε από τον Dantzig (full tableau method). Είναι αποτελεσματική για προβλήματα όπου οι πίνακες του προβλήματος είναι πυκνοί, δηλαδή ο πίνακας των περιορισμών έχει λίγα σχετικά μηδενικά.

Σε κάθε περίπτωση, το πρόβλημα θα πρέπει να “εισαχθεί” στον αλγόριθμο με την «βασική μορφή» (standard form).

2.1.3.3 Η Βασική μορφή (αναπαράσταση) του προβλήματος

Μεγιστοποίηση την: $c_1x_1 + c_2x_2 + \dots + c_nx_n$

Υπό τους περιορισμούς: $a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq r_1$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq r_2$$

$$\vdots \quad \vdots \quad \vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq r_m$$

$$x_1, x_2, \dots, x_n \geq 0$$

Όπου:

x_1, x_2, \dots, x_n είναι οι n μεταβλητές του προβλήματος

c_j είναι ο συντελεστής της j -οστης μεταβλητής στην αντικειμενική συνάρτηση

a_{ij} είναι ο συντελεστής της j -οστης μεταβλητής στον i -οστό περιορισμό.

Παρατηρούμε λοιπόν, ότι σε αυτή την μορφή:

Η αντικειμενική συνάρτηση πρέπει να μεγιστοποιηθεί

Όλοι οι περιορισμοί πρέπει να είναι ανισώσεις τύπου \leq

Όλα τα δεξιά μέρη των περιορισμών πρέπει να είναι μη αρνητικοί αριθμοί

Όλες οι μεταβλητές πρέπει να λαμβάνουν τιμές ίσες ή μεγαλύτερες του μηδενός



Οποιοδήποτε πρόβλημα γραμμικού προβλήματος μπορεί να υποστεί τους κατάλληλους μετασχηματισμούς ώστε να αναπαρασταθεί με την μορφή αυτή. την οποία παρουσιάζουμε αμέσως παρακάτω.

2.1.3.4 Ο Μετασχηματισμός της αρχικής μορφής του γραμμικού προβλήματος στην βασική μορφή

Υπάρχουν οι εξής περιπτώσεις εξαιτίας των οποίων η μορφή του προβλήματος μας μπορεί να μην είναι η βασική [3]:

Το πρόβλημα να είναι πρόβλημα ελαχιστοποίησης: Ένα πρόβλημα ελαχιστοποίησης μπορεί να μετατραπεί σε πρόβλημα μεγιστοποίησης, αρκεί να πολλαπλασιάσουμε την αντικειμενική συνάρτηση με το -1.

Κάποιος ή κάποιοι από τους περιορισμούς να είναι του τύπου \geq : Εδώ αρκεί να εισάγουμε μία μεταβλητή πλεονάσματος (surplus variable) και θα έχουμε μετατρέψει την ανίσωση σε εξίσωση. Στην συνέχεια μετατρέπουμε τον περιορισμό με τον ίδιο τρόπο που μετατρέπουμε περιορισμούς – εξισώσεις (περίπτωση d).

Το δεξί μέρος κάποιου ή κάποιων περιορισμών να είναι αρνητικό: Εδώ αρκεί να πολλαπλασιάσουμε τον περιορισμό με -1. Επειδή θα αλλάξει η φορά της ανίσωσης, εάν τελικά προκύψει ανισότητα τύπου \geq , αρκεί να ακολουθήσουμε το βήμα για περιορισμούς τύπου \geq (περίπτωση b).

Κάποιος ή κάποιοι περιορισμοί δεν είναι ανίσωση αλλά εξίσωση: Σε αυτή την περίπτωση θα προσθέσουμε μία τεχνητή μεταβλητή (artificial variable), της οποίας ο μόνος ρόλος είναι να υποχρεώσει τον αλγόριθμο να επιλέξει κατά την φάση I σωστό αρχικό σημείο.

Για να κάνουμε τους παραπάνω μετασχηματισμούς πιο κατανοητούς, θα κάνουμε την μετατροπή του παραδείγματος μας στην βασική μορφή :

Το αρχικό πρόβλημα ήταν το εξής:

$$\text{Ελαχιστοποίηση: } Z = 0,6x_1 + x_2 \text{ [συνάρτηση κόστους]}$$

$$\text{Υπό τους περιορισμούς: } 10x_1 + 4x_2 \geq 20 \text{ [σύνορο ασβεστίου]}$$

$$5x_1 + 5x_2 \geq 20 \text{ [σύνορο πρωτεΐνης]}$$

$$2x_1 + 6x_2 \geq 12 \text{ [σύνορο βιταμίνης A]}$$



$$x_1 \geq 0, x_2 \geq 0$$

Πίνακας 2, Αρχικό πρόβλημα παραδείγματος

Το πρόβλημα αυτό θα υποστεί τους παρακάτω μετασχηματισμούς:

Πολλαπλασιάζουμε την αντικειμενική συνάρτηση με -1 για να μετατρέψουμε το πρόβλημα σε πρόβλημα μεγιστοποίησης

Προσθέτουμε και στους τρείς περιορισμούς μεταβλητές πλεονάσματος. Έτσι οι τρείς περιορισμοί γίνονται εξισώσεις

Προσθέτουμε στους τρείς νέους περιορισμούς που προκύψαν από το παραπάνω βήμα, τρείς τεχνητές μεταβλητές

Μετά τους μετασχηματισμούς που θα υποστεί θα είναι εκφρασμένο στην παρακάτω μορφή:

$$\text{Μεγιστοποίηση: } Z = -0,6x_1 - x_2 \quad [\text{συνάρτηση κόστους}]$$

$$\text{Υπό τους περιορισμούς: } 10x_1 + 4x_2 - s_1 + a_1 = 20 \quad [\text{σύνορο ασβεστίου}]$$

$$5x_1 + 5x_2 - s_2 + a_2 = 20 \quad [\text{σύνορο πρωτεΐνης}]$$

$$2x_1 + 6x_2 - s_3 + a_3 = 12 \quad [\text{σύνορο βιταμίνης A}]$$

$$x_1 \geq 0, x_2 \geq 0, s_1 \geq 0, s_2 \geq 0, s_3 \geq 0, a_1 = 0, a_2 = 0, a_3 = 0$$

Πίνακας 3, Πρόβλημα παραδείγματος στην Βασική μορφή

Στο μετασχηματισμένο πρόβλημα παρατηρούμε τα εξής:

Η μεταβλητή s_1 είναι μεταβλητή επιβράδυνσης

Η μεταβλητή s_2 είναι μεταβλητή πλεονάσματος. Η διαφορά της με μία μεταβλητή επιβράδυνσης είναι το αρνητικό της πρόσημο. Η διαφορά αυτή δηλώνει ότι η μεταβλητή πλεονάσματος εκφράζει την επιπλέον ποσότητα μεταβλητής που χρησιμοποιείται σε σχέση με την απαιτούμενη από τον περιορισμό.

Οι μεταβλητές a_1, a_2 και a_3 είναι τεχνητές μεταβλητές.



2.1.3.5 Βασική μορφή και η Αναπαράσταση του πίνακα (tableau)

Από την στιγμή που το πρόβλημα μας είναι στην βασική μορφή, με άλλα λόγια τα δεδομένα μας είναι σε μορφή αναγνωρίσιμη από τον αλγόριθμό μας, μπορούμε να τα αναπαραστήσουμε και με την μορφή του πίνακα (tableau). Η αναπαράσταση αυτή δεν είναι κρίσιμη για την εκτέλεση του αλγορίθμου, αλλά βοηθάει τον χειριστή να κρατήσει επαφή με την εξέλιξη της εκτέλεσης, αφού από τον πίνακα μπορεί να εξάγει άμεσα την τρέχουσα λύση. Επίσης με την βοήθεια αυτού του πίνακα μπορεί κάποιος να προσομοιάσει (simulate) την εκτέλεση του αλγορίθμου «χειροκίνητα».

Η μορφή αυτή φαίνεται στον Πίνακα 2. Για τον πίνακα αυτό θα πρέπει να παρατηρήσουμε τα εξής:

Στην πρώτη σειρά εμφανίζεται η αντικειμενική συνάρτηση. Η αρχική μορφή της αντικειμενικής συνάρτησης γίνεται: $Z - c_1x_1 - c_2x_2 - \dots - c_nx_n = 0$

Στις υπόλοιπες σειρές εμφανίζονται οι μεταβλητές που ανήκουν στις «βασικές μεταβλητές» (basic variables).

Στις στήλες παρατίθενται οι μεταβλητές

Η στήλη «Δεξί Μέρος της Εξίσωσης» αναπαριστά, για την πρώτη γραμμή την τρέχουσα τιμή της αντικειμενικής συνάρτησης ενώ για τις υπόλοιπες το δεξί μέρος των περιορισμών.

Η στήλη «Τεστ Ελάχιστου Λόγου» χρησιμοποιείται για να καταγράφουμε το αντίστοιχο τεστ, για το οποίο θα μιλήσουμε –μεταξύ των άλλων, στην ακριβώς αμέσως επόμενη ενότητα.

Η αναπαράσταση του πίνακα, για να είναι χρήσιμη στην διάρκεια της εκτέλεσης του simplex, θα πρέπει να είναι σε «κανονική μορφή» (proper form). Στην μορφή αυτή είναι ο πίνακας όταν έχει τα παρακάτω χαρακτηριστικά:

Σε κάθε εξίσωση (γραμμές 2-m) υπάρχει ακριβώς μία μη μηδενική στήλη βασικής μεταβλητής.

Σε κάθε στήλη βασικής μεταβλητής, η τιμή του κελιού αυτής της μεταβλητής είναι 1 και για όλα τα υπόλοιπα κελιά 0.

Πίνακας 4, Αναπαράσταση Πίνακα

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | .. | x_n | s_1 | .. | s_n | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|----|-------|-------|----|-------|---------------------|--------------------------|
|------------------|------------------|---|-------|----|-------|-------|----|-------|---------------------|--------------------------|



| Z | 0 | | | | | | | | | |
|--------------------|---|---|---|---|---|---|---|---|---|---|
| Βασική μεταβλητή 1 | 1 | | | | | | | | | |
| : | : | : | : | ⋮ | : | : | ⋮ | : | : | : |
| Βασική μεταβλητή m | m | | | | | | | | | |

2.1.3.6 Γενική περιγραφή του αλγορίθμου

Έχοντας προδιαγράψει την μορφή εισόδου του αλγορίθμου (βασική μορφή) και το πώς μπορούμε να μετασχηματίσουμε ένα πρόβλημα γραμμικού προγραμματισμού σε αυτή την μορφή, μπορούμε να προχωρήσουμε στην περιγραφή των βημάτων του αλγορίθμου. Ωστόσο πριν, είναι απαραίτητο να επεξηγήσουμε κάποιους όρους που θα χρησιμοποιούμε από εδώ και στο εξής συχνά:

Μεταβλητές: Όταν θα μιλάμε για μεταβλητές, στο εξής, πέρα των αρχικών μεταβλητών απόφασης, θα συμπεριλαμβάνουμε και τις μεταβλητές επιβράδυνσης και πλεονάσματος. Όχι όμως τις τεχνητές μεταβλητές.

Μη βασικές μεταβλητές: Όλες οι μεταβλητές που ο αλγόριθμος τους έχει δώσει την τιμή 0. Οι μεταβλητές αυτές ορίζουν τις πλευρές των περιορισμών που είναι ενεργές

Βασικές μεταβλητές: Όλες οι μεταβλητές που ο αλγόριθμος τους έχει δώσει τιμές μεγαλύτερες του μηδενός.

Βάση (Basis): Επειδή δεν μπορεί κάποιος να γνωρίζει με βεβαιότητα αν μία μεταβλητή ανήκει στις βασικές ή όχι μόνο και μόνο από την τιμή της, ο αλγόριθμος αποθηκεύει σε σύνολα το ποιες μεταβλητές είναι βασικές και ποιες όχι. Αυτά τα δύο σύνολο (σύνολο βασικών μεταβλητών, σύνολο μη βασικών μεταβλητών) ονομάζεται Βάση. Η Βάση του αλγορίθμου μεταβάλλεται κατά την διάρκεια των επαναλήψεων.

Προχωρώντας στην περιγραφή του αλγορίθμου simplex, αυτός μπορεί να χωριστεί σε δύο φάσεις εκτέλεσης:



2.1.3.7 Η Φάση I της Simplex

Κατά την φάση αυτή αναζητούμε κάποιο αρχικό σημείο που είναι στην άκρη της εφικτής περιοχής, από το οποίο θα μπορεί να ξεκινήσει ο αλγόριθμος. Σε συγκεκριμένου είδους προβλήματα (π.χ. μεγιστοποίησης, όπου όλοι οι περιορισμοί είναι ανισώσεις \leq και όλα τα δεξιά μέρη των περιορισμών είναι μεγαλύτερα του μηδενός) σαν αρχική λύση μπορούμε να θεωρήσουμε την αρχή των αξόνων των μεταβλητών απόφασης. Όμως η περίπτωση αυτή δεν καλύπτει όλα τα ενδεχόμενα προβλημάτων. Υπάρχει δηλαδή περίπτωση η αρχή των αξόνων να μην ανήκει στις εφικτές λύσεις.

Έτσι για την ασφαλή εύρεση μίας αρχικής λύσης, η αναζήτηση γίνεται αναδιατυπώνοντας το πρόβλημα ως προς τις τεχνητές μεταβλητές και λύνοντας το ξανά με simplex. Πιο συγκεκριμένα, το πρόβλημα είναι η ελαχιστοποίηση κάθε τεχνητής μεταβλητής. Έτσι αντικειμενική συνάρτηση του προβλήματος της Φάσης αυτής είναι το άθροισμα των τεχνητών μεταβλητών. Για το συγκεκριμένο πρόβλημα, η αρχή των αξόνων ανήκει στην εφικτή περιοχή οπότε μπορούμε να ξεκινήσουμε από εκεί.

Στην αναπαράσταση του Πίνακα, προσθέτουμε μία γραμμή, στην οποία αναπαρίσταται η αντικειμενική συνάρτηση της πρώτης φάσης.

Στο τέλος της Φάσης αυτής, οι τιμές των τεχνητών μεταβλητών θα έχουν μηδενισθεί και οι υπόλοιπες μεταβλητές θα έχουν πάρει τέτοιες τιμές ώστε ο αλγόριθμος να βρίσκεται σε ένα εφικτό σημείο του αρχικού προβλήματος.

Θα δούμε πιο συγκεκριμένα το πώς λειτουργεί ο αλγόριθμος στην φάση αυτή, όταν θα επιλύσουμε το παράδειγμα μας.

2.1.3.8 Η Φάση II της Simplex

Κατά την φάση αυτή επαναλαμβάνουμε τα εξής βήματα:

Αναζητούμε εάν υπάρχει διαθέσιμη μη βασική μεταβλητή που να μπορεί να μπει στις βασικές μεταβλητές. Εάν δεν υπάρχει, κι άρα η αντικειμενική συνάρτηση δεν μπορεί να αυξηθεί άλλο, τερματίζουμε τον αλγόριθμο, επιστρέφοντας την τρέχουσα λύση ως άριστη.

Αφού υπάρχει τουλάχιστον μία διαθέσιμη μη βασική μεταβλητή, επιλέγουμε αυτήν που προσδίδει στην αντικειμενική συνάρτηση την μεγαλύτερη αύξηση. Δηλαδή επιλέγουμε αυτή με τον πιο αρνητικό συντελεστή. Αυτή η μεταβλητή είναι η «μεταβλητή εισόδου».



Υπολογίζουμε ποια βασική μεταβλητή θα επιλεχθεί για αντικατάσταση. Αυτή η μεταβλητή θα ονομάζεται μεταβλητή εξόδου. Θα είναι αυτή η οποία περιορίζει περισσότερο την αύξηση της μεταβλητής εισόδου. Για να αποφασίσουμε ποια μεταβλητή είναι αυτή, αρκεί να διαιρέσουμε όλες τις βασικές μεταβλητές με τον συντελεστή της μεταβλητής εισόδου στην εξίσωση της κάθε βασικής μεταβλητής και να επιλέξουμε αυτήν με τον μικρότερο λόγο. Στην περίπτωση που κάποιος συντελεστής της μεταβλητής εισόδου είναι αρνητικός ή μηδέν, δεν επιλέγεται για αντικατάσταση η συγκεκριμένη βασική μεταβλητή.

Έχοντας βρει την μεταβλητή εισόδου και την μεταβλητή εξόδου αρκεί να επικαιροποιήσουμε τον πίνακα χρησιμοποιώντας γραμμοπράξεις για να τον φέρουμε στην κανονική του μορφή

2.1.3.9 Παράδειγμα εφαρμογής της Φάσης II της Simplex

Για γίνει πιο κατανοητή η φάση αυτή του αλγορίθμου θα παρουσιάσουμε αμέσως τώρα έναν αναλυτικό υπολογισμό σε ένα υποθετικό πρόβλημα. Έστω ότι το πρόβλημα μας, στην βασική μορφή του έχει ως εξής:

$$\text{Μεγιστοποίηση: } Z = 15x_1 + 10x_2$$

$$\text{Υπό τους περιορισμούς: } x_1 \leq 2$$

$$x_2 \leq 3$$

$$x_1 + x_2 \leq 4$$

$$x_1 \geq 0, x_2 \geq 0$$

Δημιουργώντας τις μεταβλητές περισσεύματος, το πρόβλημα μας λαμβάνει την εξής μορφή:

$$\text{Μεγιστοποίηση: } Z - 15x_1 - 10x_2 - 0s_1 - 0s_2 - 0s_3 = 0$$

$$\text{Υπό τους περιορισμούς: } x_1 + s_1 = 2$$

$$x_2 + s_2 = 3$$

$$x_1 + x_2 + s_3 = 4$$

$$x_1, x_2, s_1, s_2, s_3 \geq 0$$



Η αναπαράσταση του πίνακα για το παραπάνω πρόβλημα είναι:

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|---------------------|--------------------------|
| Z | 0 | 1 | -15 | -10 | 0 | 0 | 0 | 0 | |
| s_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | |
| s_3 | 3 | 0 | 1 | 1 | 0 | 0 | 1 | 4 | |

Σε αυτή την αναπαράσταση, ξεκινάμε από το σημείο $(x_1, x_2) = (0, 0)$ και οι βασικές μεταβλητές είναι οι s_1, s_2, s_3 .

Κύκλος 1^{ος}, Βήμα 1

Υπάρχει αρνητική μη-βασική μεταβλητή στον πίνακα, άρα δεν είμαστε στο μέγιστο σημείο

Κύκλος 1^{ος}, Βήμα 2

Η μεταβλητή εισόδου θα είναι αυτή με τον μεγαλύτερο αρνητικό συντελεστή. Δηλαδή η x_1 με συντελεστή -15

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|---------------------|--------------------------|
| Z | 0 | 1 | -15 | -10 | 0 | 0 | 0 | 0 | |
| s_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | |
| s_3 | 3 | 0 | 1 | 1 | 0 | 0 | 1 | 4 | |

Κύκλος 1^{ος}, Βήμα 3

Η μεταβλητή εξόδου θα επιλεχθεί με βάση το κριτήριο του Ελάχιστου λόγου. Με βάση το κριτήριο αυτό η βασική μεταβλητή με τον μικρότερο λόγο θα επιλεχθεί. Στον παρακάτω πίνακα φαίνονται οι υπολογισμοί και το ότι η μεταβλητή εξόδου είναι η s_1



| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξιά Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|----------------------|--------------------------|
| Z | 0 | 1 | -15 | -10 | 0 | 0 | 0 | 0 | - |
| s_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | $2/1=2$ |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | - |
| s_3 | 3 | 0 | 1 | 1 | 0 | 0 | 1 | 4 | $4/1=4$ |

Κύκλος 1^{ος}, Βήμα 4

Επικαιροποιούμε τον πίνακα ως εξής:

$$(Γραμμή 1) = (Γραμμή 1) - +15 * (Γραμμή 2)$$

$$(Γραμμή 4) = (Γραμμή 4) - (Γραμμή 2)$$

Αντικαθιστούμε στην πρώτη στήλη την μεταβλητή εξόδου s_1 με την μεταβλητή

εισόδου x_1

Ο νέος επικαιροποιημένος πίνακας φαίνεται παρακάτω. Παρατηρούμε ότι η τρέχουσα τιμή της αντικειμενικής συνάρτησης είναι 30 και το σημείο στο οποίο βρισκόμαστε είναι το $(x_1, x_2) = (2, 0)$:

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξιά Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|----------------------|--------------------------|
| Z | 0 | 1 | 0 | -10 | 15 | 0 | 0 | 30 | |
| x_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | |
| s_3 | 3 | 0 | 0 | 1 | -1 | 0 | 1 | 2 | |

Κύκλος 2^{ος}, Βήμα 1

Υπάρχει αρνητική μη-βασική μεταβλητή στον πίνακα, άρα δεν είμαστε στο μέγιστο σημείο.
Συνεχίζουμε.



Κύκλος 2^{ος}, Βήμα 2

Η μεταβλητή εισόδου θα είναι αυτή με τον μεγαλύτερο αρνητικό συντελεστή. Δηλαδή η x_2 με συντελεστή -10

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|---------------------|--------------------------|
| Z | 0 | 1 | 0 | -10 | 15 | 0 | 0 | 30 | |
| x_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | |
| s_3 | 3 | 0 | 0 | 1 | -1 | 0 | 1 | 2 | |

Κύκλος 1^{ος}, Βήμα 3

Η μεταβλητή εξόδου είναι η s_1

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|---------------------|--------------------------|
| Z | 0 | 1 | 0 | -10 | 15 | 0 | 0 | 30 | - |
| x_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | - |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | 3/1=3 |
| s_3 | 3 | 0 | 0 | 1 | -1 | 0 | 1 | 2 | 2/1=1 |

Κύκλος 2^{ος}, Βήμα 4

Επικαιροποιούμε τον πίνακα ως εξής:

$$(Γραμμή 1) = (Γραμμή 1) - +10 * (Γραμμή 5)$$

$$(Γραμμή 4) = (Γραμμή 4) - (Γραμμή 5)$$

Αντικαθιστούμε στην πρώτη στήλη την μεταβλητή εξόδου s_2 με την μεταβλητή εισόδου x_2



Στον νέο επικαιροποιημένος πίνακας παρατηρούμε ότι είμαστε στο σημείο $(x_1, x_2) = (2, 2)$ και η τιμή της αντικειμενικής συνάρτησης είναι 50.

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξιά Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-------|-------|-------|-------|-------|----------------------|--------------------------|
| Z | 0 | 1 | 0 | 0 | 5 | 0 | 10 | 50 | |
| x_1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | |
| s_2 | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | |
| x_2 | 3 | 0 | 0 | 1 | -1 | 0 | 1 | 2 | |

Κύκλος 3^{ος}, Βήμα 1

Δεν υπάρχει αρνητική μη-βασική μεταβλητή στον πίνακα, άρα είμαστε στο μέγιστο σημείο και τερματίζουμε τον αλγόριθμο, δίνοντας το τρέχον σημείο σαν το μέγιστο δυνατό σημείο και την τρέχουσα τιμή της Z σαν την μέγιστη τιμή.

2.1.3.10 Εφαρμογή της Simplex στην εκτενή μορφή του παραδείγματος

Ας λύσουμε ολοκληρωμένα το αρχικό μας παράδειγμα. Το γραμμικό πρόβλημα ήταν εκφρασμένο ως εξής:

$$\text{Ελαχιστοποίηση: } Z = 0,6x_1 + x_2 \text{ [συνάρτηση κόστους]}$$

$$\text{Υπό τους περιορισμούς: } 10x_1 + 4x_2 \geq 20 \text{ [σύνορο ασβεστίου]}$$

$$5x_1 + 5x_2 \geq 20 \quad [\text{σύνορο πρωτεΐνης}]$$

$$2x_1 + 6x_2 \geq 12 \quad [\text{σύνορο βιταμίνης A}]$$

$$x_1 \geq 0, x_2 \geq 0$$

Κάνοντας τους απαραίτητους μετασχηματισμούς το φέραμε στην βασική αναπαράσταση:

$$\text{Μεγιστοποίηση: } Z = -0,6x_1 - x_2 \text{ [συνάρτηση κόστους]}$$

$$\text{Υπό τους περιορισμούς: } 10x_1 + 4x_2 - s_1 + a_1 = 20 \text{ [σύνορο ασβεστίου]}$$



$$5x_1 + 5x_2 - s_2 + a_2 = 20 \quad [\text{σύνορο πρωτεΐνης}]$$

$$2x_1 + 6x_2 - s_3 + a_3 = 12 \quad [\text{σύνορο βιταμίνης A}]$$

$$x_1 \geq 0, x_2 \geq 0, s_1 \geq 0, s_2 \geq 0, s_3 \geq 0, a_1 \geq 0, a_2 \geq 0, a_3 \geq 0$$

Θα προχωρήσουμε στην πρώτη φάση του αλγορίθμου, κατά την οποία θα προσπαθήσουμε, μηδενίζοντας τις τεχνητές μεταβλητές, να βρεθούμε σε μία εφικτή αρχική λύση του προβλήματος.

Στην Φάση I, ο στόχος είναι η ελαχιστοποίηση της αντικειμενικής συνάρτησης

$$W = a_1 + a_2 + a_3$$

ή αλλιώς η μεγιστοποίηση της

$$W = -a_1 - a_2 - a_3$$

Έτσι προχωρώντας στην επίλυση του προβλήματος της φάσης I, έχουμε τον παρακάτω πίνακα:

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|--------------------------|
| -W | ph0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| -Z | ph1 | 0 | 1 | 0.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| a_1 | 1 | 0 | 0 | 10 | 4 | -1 | 0 | 0 | 1 | 0 | 0 | 20 | |
| a_2 | 2 | 0 | 0 | 5 | 5 | 0 | -1 | 0 | 0 | 1 | 0 | 20 | |
| a_3 | 3 | 0 | 0 | 2 | 6 | 0 | 0 | -1 | 0 | 0 | 1 | 12 | |

Έχουμε να παρατηρήσουμε τα εξής:

Διατηρούμε στον πίνακα την αρχική μας αντικειμενική συνάρτηση, την οποία και θα συνεχίσουμε να χρησιμοποιούμε στην Φάση II.

Ο παραπάνω πίνακας δεν είναι σε κανονική μορφή. Βλέπουμε ότι οι τεχνητές μεταβλητές δεν έχουν μόνο το μοναδιαίο στοιχείο. Θα κάνουμε γραμμοπράξεις για να τους φέρουμε σε κανονική μορφή



Κάνοντας τις πράξεις:

$$(Γραμμή 1) = (Γραμμή 1) - (Γραμμή 3)$$

$$(Γραμμή 1) = (Γραμμή 1) - (Γραμμή 4)$$

$$(Γραμμή 1) = (Γραμμή 1) - (Γραμμή 5)$$

Καταλήγουμε στον πίνακα που είναι σε κανονική μορφή:

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξιά Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|----------------------|--------------------------|
| -W | ph0 | -1 | 0 | -17 | -15 | 1 | 1 | 1 | 0 | 0 | 0 | -52 | |
| -Z | ph1 | 0 | 1 | 0.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| a_1 | 1 | 0 | 0 | 10 | 4 | -1 | 0 | 0 | 1 | 0 | 0 | 20 | |
| a_2 | 2 | 0 | 0 | 5 | 5 | 0 | -1 | 0 | 0 | 1 | 0 | 20 | |
| a_3 | 3 | 0 | 0 | 2 | 6 | 0 | 0 | -1 | 0 | 0 | 1 | 12 | |

Από εδώ ξεκινάμε τα βήματα της πλήρους πινακοειδής μορφής του simplex:

Κύκλος 1^{ος}, Βήμα 1

Υπάρχει αρνητική μη-βασική μεταβλητή στον πίνακα, άρα δεν είμαστε στο μέγιστο σημείο

Κύκλος 1^{ος}, Βήμα 2

Η μεταβλητή εισόδου θα είναι αυτή με τον μεγαλύτερο αρνητικό συντελεστή. Δηλαδή η x_1 με συντελεστή -17

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξιά Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|----------------------|--------------------------|
| -W | ph0 | -1 | 0 | -17 | -15 | 1 | 1 | 1 | 0 | 0 | 0 | -52 | - |
| -Z | ph1 | 0 | 1 | 0.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - |



| | | | | | | | | | | | | | |
|-------|---|---|---|----|---|----|----|----|---|---|---|----|--|
| a_1 | 1 | 0 | 0 | 10 | 4 | -1 | 0 | 0 | 1 | 0 | 0 | 20 | |
| a_2 | 2 | 0 | 0 | 5 | 5 | 0 | -1 | 0 | 0 | 1 | 0 | 20 | |
| a_3 | 3 | 0 | 0 | 2 | 6 | 0 | 0 | -1 | 0 | 0 | 1 | 12 | |

Κύκλος 1^{ος}, Βήμα 3

Η μεταβλητή εξόδου θα επιλεχθεί με βάση το κριτήριο του Ελάχιστου λόγου. Με βάση το κριτήριο αυτό η βασική μεταβλητή με τον μικρότερο λόγο θα επιλεχθεί. Στον παρακάτω πίνακα φαίνονται οι υπολογισμοί και το ότι η μεταβλητή εξόδου είναι η a_1

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|--------------------------|
| -W | ph0 | -1 | 0 | -17 | -15 | 1 | 1 | 1 | 0 | 0 | 0 | -52 | - |
| -Z | ph1 | 0 | 1 | 0.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - |
| a_1 | 1 | 0 | 0 | 10 | 4 | -1 | 0 | 0 | 1 | 0 | 0 | 20 | 2 |
| a_2 | 2 | 0 | 0 | 5 | 5 | 0 | -1 | 0 | 0 | 1 | 0 | 20 | 4 |
| a_3 | 3 | 0 | 0 | 2 | 6 | 0 | 0 | -1 | 0 | 0 | 1 | 12 | 6 |

Κύκλος 1^{ος}, Βήμα 4

Επικαιροποιούμε τον πίνακα ως εξής:

$$(Γραμμή 3) = (Γραμμή 3) / 10$$

$$(Γραμμή 1) = (Γραμμή 1) + 17 * (Γραμμή 3)$$

$$(Γραμμή 2) = (Γραμμή 2) - 0,6 * (Γραμμή 3)$$

$$(Γραμμή 4) = (Γραμμή 4) - 5 * (Γραμμή 3)$$

$$(Γραμμή 5) = (Γραμμή 5) - 2 * (Γραμμή 3)$$

Αντικαθιστούμε στην πρώτη στήλη την μεταβλητή εξόδου a_1 με την μεταβλητή εισόδου x_1



| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|--------------------------|
| -W | ph0 | -1 | 0 | 0 | -8.2 | -0.7 | 1 | 1 | 1.7 | 0 | 0 | -18 | |
| -Z | ph1 | 0 | 1 | 0 | 0.76 | 0.06 | 0 | 0 | -0.06 | 0 | 0 | -1.2 | |
| x_1 | 1 | 0 | 0 | 1 | 0.4 | -0.1 | 0 | 0 | 0.1 | 0 | 0 | 2 | |
| a_2 | 2 | 0 | 0 | 0 | 3 | 0.5 | -1 | 0 | -0.5 | 1 | 0 | 10 | |
| a_3 | 3 | 0 | 0 | 0 | 5.2 | 0.2 | 0 | -1 | -0.2 | 0 | 1 | 8 | |

Κύκλος 2^{ος}, Βήμα 1

Υπάρχει αρνητική μη-βασική μεταβλητή στον πίνακα, άρα δεν είμαστε στο μέγιστο σημείο

Κύκλος 2^{ος}, Βήμα 2

Η μεταβλητή εισόδου θα είναι αυτή με τον μεγαλύτερο αρνητικό συντελεστή. Δηλαδή η x_2 με συντελεστή -8,2

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|--------------------------|
| -W | ph0 | -1 | 0 | 0 | -8.2 | -0.7 | 1 | 1 | 1.7 | 0 | 0 | -18 | - |
| -Z | ph1 | 0 | 1 | 0 | 0.76 | 0.06 | 0 | 0 | -0.06 | 0 | 0 | -1.2 | - |
| x_1 | 1 | 0 | 0 | 1 | 0.4 | -0.1 | 0 | 0 | 0.1 | 0 | 0 | 2 | |
| a_2 | 2 | 0 | 0 | 0 | 3 | 0.5 | -1 | 0 | -0.5 | 1 | 0 | 10 | |
| a_3 | 3 | 0 | 0 | 0 | 5.2 | 0.2 | 0 | -1 | -0.2 | 0 | 1 | 8 | |

Κύκλος 2^{ος}, Βήμα 3



Η μεταβλητή εξόδου θα επιλεχθεί με βάση το κριτήριο του Ελάχιστου λόγου. Με βάση το κριτήριο αυτό η βασική μεταβλητή με τον μικρότερο λόγο θα επιλεχθεί. Στον παρακάτω πίνακα φαίνονται οι υπολογισμοί και το ότι η μεταβλητή εξόδου είναι η a_3

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστο υ Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|-------|---------------------|---------------------------|
| -W | ph0 | -1 | 0 | 0 | -8.2 | -0.7 | 1 | 1 | 1.7 | 0 | 0 | -18 | |
| -Z | ph1 | 0 | 1 | 0 | 0.76 | 0.06 | 0 | 0 | -0.06 | 0 | 0 | -1.2 | |
| x_1 | 1 | 0 | 0 | 1 | 0.4 | -0.1 | 0 | 0 | 0.1 | 0 | 0 | 2 | 5 |
| a_2 | 2 | 0 | 0 | 0 | 3 | 0.5 | -1 | 0 | -0.5 | 1 | 0 | 10 | 3,33 |
| a_3 | 3 | 0 | 0 | 0 | 5.2 | 0.2 | 0 | -1 | -0.2 | 0 | 1 | 8 | 1,53 |

Κύκλος 2^{ος}, Βήμα 4

Επικαιροποιούμε τον πίνακα ως εξής:

$$(Γραμμή 5) = (Γραμμή 5) / 5,2$$

$$(Γραμμή 4) = (Γραμμή 4) - 3^* (Γραμμή 5)$$

$$(Γραμμή 3) = (Γραμμή 3) - 0,4^* (Γραμμή 5)$$

$$(Γραμμή 2) = (Γραμμή 2) - 0,76^* (Γραμμή 5)$$

$$(Γραμμή 1) = (Γραμμή 1) + 8,2^* (Γραμμή 5)$$

Αντικαθιστούμε στην πρώτη στήλη την μεταβλητή εξόδου a_3 με την μεταβλητή

εισόδου x_2

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστο υ Λόγου |
|------------------|------------------|----|---|-------|-------|-------|-------|-------|-------|-------|--------|---------------------|---------------------------|
| -W | ph0 | -1 | 0 | 0 | 0 | -0.38 | 1 | -0.57 | 1.38 | 0 | 1.5769 | -5.38 | |
| -Z | ph1 | 0 | 1 | 0 | 0 | 0.03 | 0 | 0.14 | -0.0 | 0 | -0.1 | -2.36 | |



| | | | | | | | | | | | | | |
|-------|---|---|---|---|---|-----------|----|-----------|-----------|---|-----------|------|--|
| | | | | | | | | | 3 | | 4 | | |
| x_1 | 1 | 0 | 0 | 1 | 0 | -0.1 1 | 0 | 0.0 7 | 0.1 1 | 0 | -0.0 7 | 1.38 | |
| a_2 | 2 | 0 | 0 | 0 | 0 | 0.3 8 | -1 | 0.5 7 | -0.3 8 | 1 | -0.5 7 | 5.38 | |
| x_2 | 3 | 0 | 0 | 0 | 1 | 0.0 3 | 0 | -0.1 9 | -0.0 3 | 0 | 0.1 9 | 1.5 | |

Για λόγους οικονομίας χώρου, χωρίς να κάνουμε αναλυτικά τα υπόλοιπα βήματα, παρουσιάζουμε τους διαδοχικούς πίνακες simplex:

Κύκλος 3^{ος}

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | W | Z | x_1 | x_2 | s_1 | s_2 | s_3 | a_1 | a_2 | a_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστον Λόγου |
|------------------|------------------|----|---|-----------|-----------|------------|------------|-----------|-----------|------------|------------|---------------------|--------------------------|
| -W | ph0 | -1 | 0 | 0.0 00 | 0.0 00 | 0.0 00 | 0.0 00 | 0.0 00 | 1.0 00 | 1.0 00 | 1.0 00 | 0.000 | |
| -Z | ph1 | 0 | 1 | 0.0 00 | 0.0 00 | -0.0 67 | 0.2 53 | 0.0 00 | 0.0 67 | -0.2 53 | 0.0 00 | -3.734 | |
| x_1 | 1 | 0 | 0 | 1.0 00 | 0.0 00 | -0.1 67 | 0.1 33 | 0.0 00 | 0.1 67 | -0.1 33 | 0.0 00 | 0.667 | |
| s_3 | 2 | 0 | 0 | 0.0 00 | 0.0 00 | 0.6 67 | -1.7 33 | 1.0 00 | 0.6 67 | 1.7 33 | -1.0 00 | 9.334 | |
| x_2 | 3 | 0 | 0 | 0.0 00 | 1.0 00 | 0.1 67 | -0.3 33 | 0.0 00 | 0.1 67 | 0.3 33 | 0.0 00 | 3.333 | |

Στον παραπάνω πίνακα δεν υπάρχουν αρνητικές μη βασικές μεταβλητές, γεγονός που μας υποδεικνύει ότι η Φάση I τελείωσε.

Προχωράμε στην Φάση II, αφαιρώντας την πρώτη γραμμή και συνεχίζοντας κανονικά την simplex:

Ο πίνακας από τον οποίον θα ξεκινήσει η Φάση II θα είναι:



| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-----------|-----------|-----------|----------------|-----------|---------------------|--------------------------|
| -Z | ph1 | 1 | 0.0 00 | 0.0 00 | - 67 | 0.2 53 | 0.0 00 | -3.73 | |
| x_1 | 1 | 0 | 1.0 00 | 0.0 00 | - 67 | 0.1 33 | 0.0 00 | 0.67 | |
| s_3 | 2 | 0 | 0.0 00 | 0.0 00 | 0.6 67 | - 1.7 33 | 1.0 00 | 9.33 | |
| x_2 | 3 | 0 | 0.0 00 | 1.0 00 | 0.1 67 | - 0.3 33 | 0.0 00 | 3.33 | |

Και χωρίς πάλι να μπούμε σε λεπτομερή ανάλυση, βλέποντας ότι η μη βασική μεταβλητή s_1 είναι αρνητική, συνεχίζουμε:

Κύκλος 1^{ος}

| Βασική Μεταβλητή | Αριθμός Εξίσωσης | Z | x_1 | x_2 | s_1 | s_2 | s_3 | Δεξί Μέρος Εξίσωσης | Κριτήριο Ελάχιστου Λόγου |
|------------------|------------------|---|-----------|-----------|-----------|----------------|----------------|---------------------|--------------------------|
| -Z | ph1 | 1 | 0.0 00 | 0.0 00 | 0.0 00 | 0.0 80 | 0.1 00 | -2.800 | |
| x_1 | 1 | 0 | 1.0 00 | 0.0 00 | 0.0 00 | - 0.3 00 | 0.2 50 | 3.001 | |
| s_1 | 2 | 0 | 0.0 00 | 0.0 00 | 1.0 00 | - 2.6 00 | 1.5 00 | 14.000 | |
| x_2 | 3 | 0 | 0.0 00 | 1.0 00 | 0.0 00 | 0.1 00 | - 0.2 50 | 1.000 | |

Αφού δεν υπάρχει αρνητικός συντελεστής στις μη βασικές μεταβλητές, έχει τελειώσει και η Φάση II της simplex. Βρισκόμαστε στο σημείο $(x_1, x_2) = (3, 1)$ και η τιμή της αντικειμενικής συνάρτησης είναι 2.8



2.1.3.11 Ειδικές Περιπτώσεις του αλγορίθμου *Simplex*

Κατά την υλοποίηση του αλγορίθμου σε εκτελέσιμο πρόγραμμα θα πρέπει να αντιμετωπιστούν οι εξής ειδικές περιπτώσεις:

- «Ισοπαλία» στο Κριτήριο Ελάχιστου Λόγου για δύο βασικές μεταβλητές: Στην περίπτωση αυτή κινδυνεύει ο αλγόριθμος μας να επιλέγει κυκλικά και συνέχεια ένα σετ μεταβλητών και θα πρέπει να εφαρμοστεί κάποια τεχνική που να το αποφεύγει.
- Στο Κριτήριο Ελάχιστου Λόγου οι τιμές που προκύπτουν είναι όλες αρνητικές: Στην περίπτωση αυτή δεν υπάρχει κάποιος περιορισμός που να εμποδίζει την αύξηση της βασικής μεταβλητής. Δηλαδή το πρόβλημα μας είναι μη φραγμένο.
- «Ισοπαλία» στο κριτήριο επιλογής μεταβλητής εισόδου: Το γεγονός αυτό μας υποδεικνύει ότι όποια και από τις δύο μη-βασικές μεταβλητές να επιλέξουμε ως μεταβλητή εισόδου, θα έχουμε την ίδια αύξηση στην αντικειμενική συνάρτηση. Μία «στρατηγική» εξόδου από το πρόβλημα είναι να επιλέξουμε τυχαία μία από τις δύο μεταβλητές.
- Ύπαρξη μηδενικών συντελεστών μη-βασικών μεταβλητών στην αντικειμενική συνάρτηση, ενώ είμαστε στο μέγιστο σημείο: Το γεγονός αυτό μας υποδεικνύει ότι τα σημεία των λύσεων είναι περισσότερα του ενός.



2.2 Παράλληλος Υπολογισμός και βιβλιοθήκη MPI

2.2.1 Βασικές αρχές Παράλληλου Υπολογισμού

2.2.1.1 Μοντέλα και μέθοδοι σχεδίασης παράλληλων αλγορίθμων

Κατά τα τελευταία είκοσι πέντε και πλέον έτη έχουν αναπτυχθεί και υιοθετηθεί διάφορα μοντέλα παράλληλου υπολογισμού, αντίστοιχα μοντέλα παράλληλου προγραμματισμού, καθώς επίσης και πιο εξειδικευμένες υλοποιήσεις για διαφόρων μορφών πραγματικές/ρεαλιστικές παράλληλες μηχανές.

Όσον αφορά τα μοντέλα παράλληλου υπολογισμού, ένα πολύ δημοφιλές θεωρητικό μοντέλο για το σχεδιασμό παράλληλων αλγορίθμων είναι το πρότυπο PRAM, το οποίο περιγράφει έναν εξιδανικευμένο τρόπο λειτουργίας για σύγχρονα πολυεπεξεργαστικά συστήματα διαμοιραζόμενης ή κοινής μνήμης [21]. Οι αλγόριθμοι που προκύπτουν βασιζόμενοι στον συγκεκριμένο τρόπο λειτουργίας, πολύ συχνά δεν έχουν ωστόσο την αναμενόμενη απόδοση σε πραγματικά συστήματα για τεχνικούς λόγους. Το αυξημένο ενδιαφέρον για ασύγχρονους πολυεπεξεργαστές διαμοιρασμένης μνήμης χαμηλού κόστους τύπου cluster έχει ωθήσει προσπάθειες για μεταφορά των PRAM αλγορίθμων σε αυτά τα συστήματα. Λόγω του αργού δικτύου επικοινωνίας τους, οι clusters αποδίδουν ακόμα χειρότερα σε τέτοιου είδους υλοποιήσεις και γίνεται επιτακτική η ανάγκη για άλλα θεωρητικά μοντέλα. Μια ιδέα που γεφυρώνει το κενό μεταξύ PRAM αλγορίθμων και ασύγχρονου υλικού είναι το Bulk Synchronous Parallel μοντέλο (BSP), όπως προτάθηκε από τον Valiant [22,23]. Ένα BSP σύστημα αποτελείται από ένα πλήθος επεξεργαστών που ο καθένας έχει τη δική του τοπική μνήμη και συνδέονται με ένα δίκτυο δρομολόγησης το οποίο μπορεί να μεταφέρει μηνύματα από ένα σημείο σε ένα άλλο. Δεν δίνει τη δυνατότητα για μαζική μετάδοση σε πλήθος επεξεργαστών ή άλλους τρόπους επικοινωνίας.

Μια ακόμα πιο πρακτική προσέγγιση βασισμένη στο μοντέλο BSP είναι το πρότυπο Coarse Grain Multicomputer (CGM) όπως σχεδιάστηκε και εξελίχθηκε από τον Dehne με διάφορες ερευνητικές ομάδες [24]. Το μοντέλο CGM υποθέτει (επιπροσθέτως του BSP) πως η ταχύτητα μεταφοράς δεδομένων στο δίκτυο επικοινωνίας είναι πολύ μικρότερη από την ταχύτητα των τοπικών υπολογισμών.



Αντίστοιχα, τα τρία βασικότερα μοντέλα/πρωτόκολλα παράλληλου προγραμματισμού είναι ο παραλληλισμός δεδομένων, ο κοινός χώρος διευθύνσεων και η μεταβίβαση μηνυμάτων (*message passing*). Το πρώτο μοντέλο ονομάζεται και σύγχρονος παράλληλος προγραμματισμός ενώ τα άλλα δύο ονομάζονται και μοντέλα ασυγχρόνιστου παράλληλου προγραμματισμού, αλλά και μοντέλα παραλληλισμού ελέγχου. Επίσης ο δικτυακός παράλληλος προγραμματισμός (*cluster computing*) ορίζεται ως το είδος του παράλληλου / κατανεμημένου προγραμματισμού σε ένα τοπικό δίκτυο υπολογιστών. Είναι ουσιαστικά παράλληλος προγραμματισμός όπου οι ανεξάρτητοι υπολογιστές του δικτύου παίζουν το ρόλο των επεξεργαστών μίας *MIMD* παράλληλης μηχανής. Λόγω της κατανεμημένης αυτής αρχιτεκτονικής, ο δικτυακός παράλληλος προγραμματισμός γίνεται κυρίως με χρήση του μοντέλου μεταβίβασης μηνυμάτων και οδηγεί σε χονδρόκοκκα προγράμματα προκειμένου να αποφεύγονται οι (χρονοβόρες) επικοινωνίες του δικτύου.

Στα παραπάνω πλαίσια, μία πολύ γνωστή και ευρέως διαδεδομένη μέθοδος (η οποία ακολουθήθηκε σε σημαντικό βαθμό και στα πλαίσια της εργασίας αυτής – καθώς ταιριάζει ιδιαίτερα στην περίπτωση πολυεπεξεργαστικών συστημάτων κατανεμημένης μνήμης) είναι η μέθοδος ανάπτυξης των τεσσάρων βημάτων του Foster [25]:

1. Partitioning:

Αρχικά, διαμοιράζουμε (χωρίζουμε) το πρόβλημά μας σε μικρότερα tasks τα οποία μπορούν (κρίνουμε αρχικά ότι) να εκτελεστούν παράλληλα. Οι πιο συνήθεις μέθοδοι διαχωρισμού είναι η ‘data decomposition’ (αποσύνθεση/διαμοιρασμός δεδομένων) και η ‘functional decomposition’ (αποσύνθεση/διαμοιρασμός υπολογιστικού φόρτου).

2. Communication:

Σε μία ιδανική κατάσταση θα περιμέναμε τα παραπάνω διαχωρισμένα tasks να μπορούν να εκτελεστούν παράλληλα-ανεξάρτητα εντελώς μεταξύ τους. Στην πραγματικόττα ωστόσο απαιτούν συνήθως δεδομένα από άλλα tasks ή αποτελούν τα ίδια απαραίτητες εισόδους για άλλα tasks. Αυτές τις απαιτήσεις επικοινωνίας που υπάρχουν μεταξύ των διαχωρισμένων tasks πρέπει να τις αναγνωρίσουμε και να τις ταξινομήσουμε κατάλληλα σε απαιτήσεις τοπικής (local communication) και σφαιρικό επικοινωνίας (global communication)

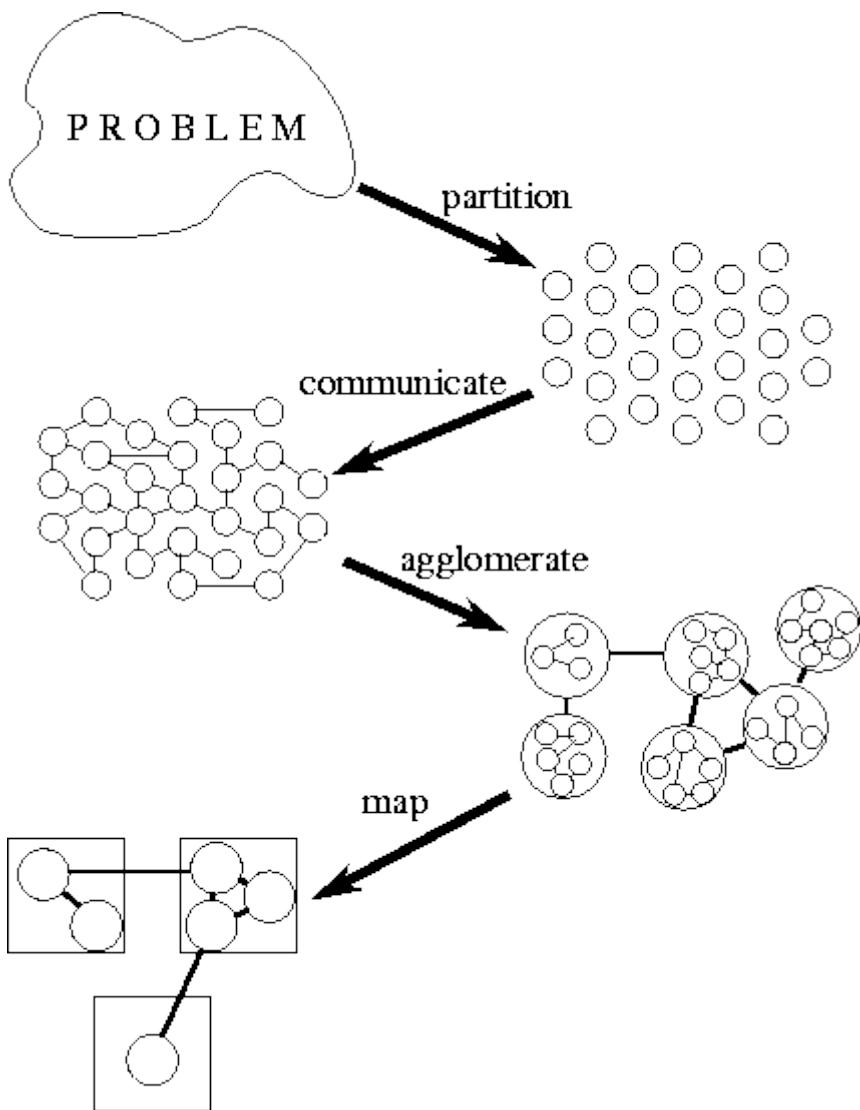
3. Agglomeration:

Σε αυτή τη φάση ομαδοποιούμε πολλά μικρά tasks σε μεγαλύτερα κατά τέτοιον τρόπο ώστε να ελαχιστοποιείται κατά το δυνατόν το συνολικό κόστος επικοινωνίας (communication cost – ομαδοποιώντας π.χ. tasks που πρέπει να επικοινωνήσουν μεταξύ τους μαζί), καθώς επίσης

και να αμβλυνθεί/βελτιωθεί το επίπεδο/επιβάρυνση από το μεγέθος/διαχείριση πολλαπλών μηνυμάτων (ομαδοποιώντας π.χ. tasks τα οποία επικοινωνούν συχνά μεταξύ τους)

4. Mapping:

Μέχρι αυτή τη φάση έχουμε διαμοιράσει το πρόβλημα σε λογικά σε μέγεθος και απαιτήσεις (με τα μέχρι τώρα κριτήρια) tasks αλλά όχι σε επεξεργαστές καθ' αυτούς. Αντικείμενο της τελευταίας αυτής φάσης είναι ακριβώς αυτό, να αντιστοιχίσουμε δηλαδή τα tasks σε επεξεργαστές, πράγμα το οποίο συνήθως δεν είναι δύσκολο αν έχουμε στατικά προβλήματα, αν έχουμε ωστόσο δυναμικά προβλήματα (στα οποία δεν γνωρίζουμε εξαρχής τον αριθμό των tasks κλπ) μπορεί να είναι ιδιαίτερα δύσκολο και χρονοβόρο.





2.2.1.2 Απόδοση παράλληλων αλγορίθμων - Επιτάχυνση

Όταν σχεδιάζουμε και υλοποιούμε παράλληλους αλγόριθμους θέλουμε να τρέχουν γρηγορότερα από τους αντίστοιχους σειριακούς. Η «επιτάχυνση» (speedup) μετράει ακριβώς αυτή την παράμετρο. Ορίζεται ως ο λόγος του χρόνου εκτέλεσης του σειριακού αλγορίθμου προς τον λόγο εκτέλεσης του παράλληλου αλγορίθμου [5].

$$\psi(n, p) = \frac{\text{Χρόνος Σειριακής Υλοποίησης}}{\text{Χρόνος Παράλληλης Υλοποίησης}}$$

Θέλοντας να αναλύσουμε λίγο περαιτέρω την επιτάχυνση μίας παράλληλης υλοποίησης, γνωρίζουμε ότι κάθε παράλληλος αλγόριθμος περιέχει εντολές που:

Υλοποιούνται μόνο σειριακά

Μπορούν να υλοποιηθούν παράλληλα

Προσδίδουν καθυστέρηση επικοινωνίας ή κάνουν άσκοπους υπολογισμούς

Έστω τώρα ότι θα συμβολίζουμε την επιτάχυνση ενός προβλήματος μεγέθους n που λύνεται από p επεξεργαστές ως $\psi(n, p)$. Επίσης με $\sigma(n)$ δηλώνουμε το κομμάτι του αλγορίθμου που μπορεί να υλοποιηθεί μόνο σειριακά και με $\varphi(n)$ το κομμάτι που μπορεί να υλοποιηθεί παράλληλα, ενώ με $\kappa(n, p)$ τις καθυστερήσεις επικοινωνίας, αφού αυτές εκτός από το μέγεθος του προβλήματος επηρεάζονται και από το πλήθος των επεξεργαστών.

Με βάση τους παραπάνω ορισμούς, ένα σειριακό πρόγραμμα θα εκτελείται σε $\sigma(n) + \varphi(n)$ χρόνο, αφού εκτελείται από έναν μόνο επεξεργαστή και δεν υπάρχει καθυστέρηση επικοινωνίας.

Αντίστοιχα, ο παράλληλος υπολογισμός του παραπάνω προγράμματος θα χρειάζεται $\sigma(n) + (\varphi(n)/p) + \kappa(n, p)$, αφού το παράλληλο κομμάτι του αλγορίθμου μοιράζεται σε p επεξεργαστές. Ωστόσο υπάρχει και μία αύξηση χρόνου από την παραλληλοποίηση του αλγορίθμου, ανάλογη με τον αριθμό των επεξεργαστών.

Μπορούμε λοιπόν να ξαναγράψουμε τον τύπο της επιτάχυνσης ως εξής:

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + (\varphi(n)/p) + \kappa(n, p)}$$

γιατί πολλές φορές ο παράλληλος υπολογισμός δεν μπορεί να διαχωρίσθει τέλεια μεταξύ των επεξεργαστών.



2.2.1.3 Απόδοση παράλληλων αλγορίθμων – Αποδοτικότητα

Η αποδοτικότητα (efficiency) ενός παράλληλου αλγορίθμου είναι ένα μέτρο του βαθμού χρήσης των επεξεργαστών. Ορίζεται ως ο λόγος της επιτάχυνσης προς τον αριθμό των επεξεργαστών που χρησιμοποιούνται.

$$\varepsilon(n, p) = \frac{\psi(n, p)}{p}$$

2.2.1.4 Απόδοση παράλληλων αλγορίθμων – Ο Νόμος του Amdahl

Το ποσοστό των διεργασιών του αλγορίθμου που μπορούν να εκτελεστούν μόνο σειριακά είναι:

$$f = \frac{\sigma(n)}{\sigma(n) + \varphi(n)}$$

Εκτελώντας διάφορους υπολογισμούς, καταλήγουμε στην σχέση του παραπάνω λόγου με την επιτάχυνση της παραλληλοποίησης:

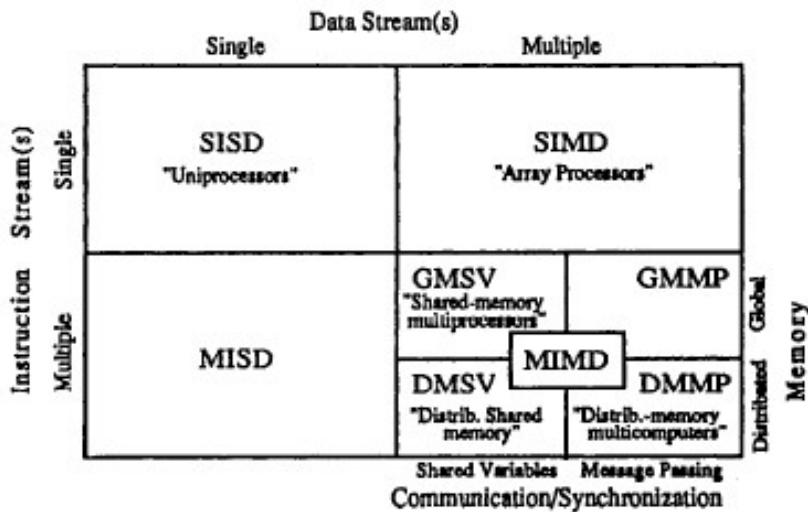
$$\psi(n, p) \leq \frac{1}{f + (1-f)/p}$$

Η παραπάνω σχέση μας, λαμβάνοντας σαν δεδομένο ότι προσπαθούμε να λυνόσουμε ένα πρόβλημα μεγέθους n , μας δίνει την σχέση που υπάρχει μεταξύ της επιτάχυνσης και του πλήθους των επεξεργαστών που χρησιμοποιούμε. Επίσης μπορεί να χρησιμοποιηθεί για να βρούμε την ασυμπτωτική σε σχέση με το πλήθος των επεξεργαστών ($p \rightarrow \infty$) συμπεριφορά της επιτάχυνσης.

Ωστόσο ο νόμος του Amdahl επειδή εξαλείφει τον όρο του παρανομαστή που εκφράζει την καθυστέρηση επικοινωνίας, τείνει να υπερεκτιμά την επιτάχυνση που μπορεί να επιτευχθεί αυξάνοντας τον αριθμό των επεξεργαστών.

2.2.1.5 Ταξινόμηση Υπολογιστικών Συστημάτων Παράλληλης Επεξεργασίας

Το παρακάτω σχήμα δίνει μία ταξινόμηση των υπολογιστικών συστημάτων από τον J. Flynn με βάση την ροή εντολών και την ροή δεδομένων προς τον επεξεργαστή: [Parhami, 1999]



Πίνακας 5, Ταξινόμηση Υπολογιστικών Συστημάτων κατά J. Flynn

Η κλάση SISD περιγράφει τα γνωστά και συνήθη απλά υπολογιστικά συστήματα με έναν επεξεργαστή. Η κλάση SIMD περιγράφει συστήματα με πολλούς επεξεργαστές, που οι εντολές προς αυτούς δίνονται από μία κεντρική μονάδα (array processors) και η κλάση MISD περιέχει συστήματα που δεν χρησιμοποιούνται ευρέως.

Η κλάση συστημάτων με την οποία θα ασχοληθούμε και θα χρησιμοποιήσουμε είναι αυτή των MIMD. Επειδή στην κλάση αυτή περιλαμβάνεται μεγάλος αριθμός συστημάτων, ο E.Johnson πρότεινε την περαιτέρω κατηγοριοποίηση τους με βάση την δομή της μνήμης τους (κοινή ή κατανεμημένη) και τον μηχανισμό που χρησιμοποιείται για την επικοινωνία μεταξύ των επεξεργαστών (κοινές μεταβλητές ή ανταλλαγή μηνυμάτων).

Με βάση λοιπόν αυτή την περαιτέρω κατηγοριοποίηση, τα συστήματα που εμείς θα ασχοληθούμε ανήκει στην κλάση DMMP. Ο παράλληλος υπολογισμός σε ένα τέτοιο σύστημα γίνεται με ανταλλαγή μηνυμάτων, δηλαδή δεδομένων, μεταξύ των επεξεργαστών ενώ ο κάθε επεξεργαστής έχει την δική του μνήμη.



2.2.2 Η βιβλιοθήκη MPI

2.2.2.1 Εισαγωγικά

Για την χρήση των συστημάτων MIMD έχει αναπτυχθεί η πρότυπη βιβλιοθήκη συναρτήσεων Message Parsing Interface (MPI, η οποία καθορίζει όλες εκείνες τις απαραίτητες συναρτήσεις για τον χειρισμό ενός τέτοιου συστήματος.

Το πρότυπο MPI στηρίζεται στο μοντέλο της επεξεργασίας μέσω ανταλλαγής μηνυμάτων (message-passing model), το οποίο όπως προαναφέρθηκε στην αρχή του παρόντος κεφαλαίου, αποτελεί το πλέον ίσως διαδεδομένο μοντέλο παράλληλου προγραμματισμού για παράλληλα περιβάλλοντα κατανεμημένης μνήμης.

Επειδή όπως αναφέρθηκε το MPI είναι ουσιαστικά ένα πρότυπο του μοντέλου ανταλλαγής μηνυμάτων, υπάρχουν πολλές υλοποιήσεις του, με πιο διαδεδομένες τις εκδόσεις MPICH, CHIMP, LAM MPI κ.α. Γενικά, το πρότυπο αυτό έχει κερδίσει ευρεία αποδοχή και σχεδόν κάθε εμπορικό σύστημα τύπου DMMP υποστηρίζει κάποια υλοποίηση του. Επίσης υπάρχουν αρκετές δωρεάν υλοποιήσεις του. Εμείς θα χρησιμοποιήσουμε μία τέτοια δωρεάν υλοποίηση, και πιο συγκεκριμένα την υλοποίηση MPI-2 της έκδοσης MPICH (MPICH-2) για τις γλώσσες C, C++ και Fortran [<http://www.mcs.anl.gov/research/projects/mpich2/>, Argonne National Laboratory] σε περιβάλλον LINUX.

Το πρότυπο MPI περιλαμβάνει περίπου 120 συναρτήσεις, αλλά πολλά παράλληλα προγράμματα μπορούν να γραφούν με λίγες μόνο βασικές συναρτήσεις του. Επομένως, δίνει τη δυνατότητα της ευελιξίας όταν απαιτείται εκτεταμένη λειτουργικότητα, αλλά ταυτόχρονα δε είναι απαραίτητο να γνωρίζει κανείς το MPI στο σύνολό του για να το χρησιμοποιήσει.

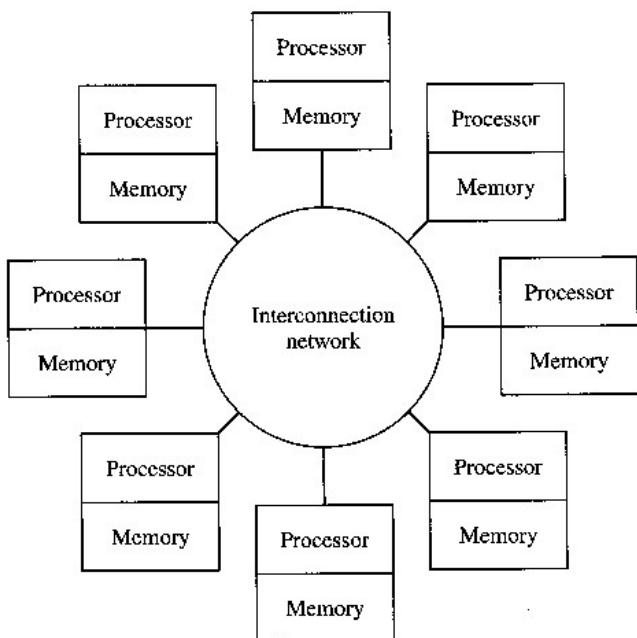
Ωστόσο, ο βασικός στόχος του MPI, όπως συμβαίνει με τα περισσότερα πρότυπα, είναι η μεταφερσιμότητα σε διαφορετικά μηχανήματα. Αυτό σημαίνει πως ο ίδιος κώδικας μπορεί να εκτελεστεί σε μια ποικιλία μηχανών εφόσον είναι διαθέσιμη η βιβλιοθήκη MPI. Και παρόλο που η ανταλλαγή μηνυμάτων αφορά συχνότερα παράλληλους υπολογιστές κατανεμημένης μνήμης, ο ίδιος κώδικας μπορεί να τρέξει εξίσου καλά και σε παράλληλο υπολογιστή διαμοιραζόμενης μνήμης. Μπορεί να τρέξει σε δίκτυο ή ακόμα και ως σύνολο διεργασιών στον ίδιο μεμονωμένο υπολογιστή. Επίσης, ένας άλλος τύπος συμβατότητας που προσφέρει το MPI είναι η δυνατότητα να εκτελεστεί κώδικας σε ετερογενή συστήματα,



δηλαδή σε συλλογές επεξεργαστών με διαφορετικές αρχιτεκτονικές. Ο χρήστης δε χρειάζεται να ανησυχεί αν ο κώδικας στέλνει μηνύματα μεταξύ επεξεργαστών της ίδιας ή διαφορετικής αρχιτεκτονικής καθώς επίσης και αν τα μηνύματα ανταλλάσσονται μεταξύ διεργασιών που βρίσκονται στον ίδιο ή σε διαφορετικούς επεξεργαστές.

2.2.2.2 Μοντέλο επεξεργασίας μέσω ανταλλαγής μηνυμάτων

Στο μοντέλο ανταλλαγής μηνυμάτων [5] όπως φαίνεται και στην ακόλουθη εικόνα θεωρούμε ότι υπάρχει ένα σύνολο επεξεργαστών, ο καθένας με την δική του μνήμη. Ο κάθε επεξεργαστής έχει προσβαση μόνο στα εντολές και τα δεδομένα της δικής του μνήμης. Ωστόσο υπάρχει ένα δίκτυο σύνδεσης που επιτρέπει στους επεξεργαστές να ανταλλάσσουν μηνύματα μεταξύ τους.



Εικόνα 2, Το μοντέλο επεξεργασίας μέσω ανταλλαγής μηνυμάτων

Ο χρήστης καθορίζει τον αριθμό των επεξεργαστών που θα εκτελούν παράλληλα το πρόγραμμα του όταν δίνει την εντολή εκτέλεσης του, και ο αριθμός αυτών παραμένει σταθερός μέχρι το τέλος του. Ο κάθε επεξεργαστής εκτελεί το ίδιο πρόγραμμα, αλλά επειδή έχει διαφορετικό ID, μπορεί ο προγραμματιστής με εντολές ελέγχου να επιλέγει κομμάτια κώδικα που θα εκτελούνται από συγκεκριμένους επεξεργαστές.

Ο κάθε επεξεργαστής εκτελεί τους υπολογισμούς του προγράμματος στις τοπικές του μεταβλητές και μέσω εντολών του προγράμματος μπορεί να τις μεταδώσει στους άλλους.



2.2.2.3 Τύποι δεδομένων της βιβλιοθήκης

| Name | C type |
|--------------------|----------------|
| MPI_CHAR | signed char |
| MPI_DOUBLE | double |
| MPI_FLOAT | float |
| MPI_INT | int |
| MPI_LONG | long |
| MPI_LONG_DOUBLE | long double |
| MPI_SHORT | short |
| MPI_UNSIGNED_CHAR | unsigned char |
| MPI_UNSIGNED | unsigned int |
| MPI_UNSIGNED_LONG | unsigned long |
| MPI_UNSIGNED_SHORT | unsigned short |

Πίνακας 6, Τύποι δεδομένων MPI

2.2.2.4 Οι συναρτήσεις της βιβλιοθήκης

Η βιβλιοθήκη MPI ορίζει [6] :

Συναρτήσεις για την ρύθμιση του περιβάλλοντος παράλληλου υπολογισμού (π.χ. MPI_Init, MPI_Finalize, MPI_Comm_rank κ.α.).

Συναρτήσεις για την αποστολή και λήψη ενός μηνύματος μεταξύ δύο επεξεργαστών (π.χ. MPI_Send, MPI_Receive κ.α.).

Συναρτήσεις για συλλογική επικοινωνία μεταξύ των επεξεργαστών (π.χ. MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Reduce κ.α.)

Άλλες παράγωγες συναρτήσεις (για ομαδοποίηση δεδομένων, ορισμό groups επεξεργαστών, ορισμό εικονικών τοπολογιών κ.α.).

Τύπους δεδομένων (MPI datatypes) για την υποστήριξη αντίστοιχων τύπων της γλώσσας προγραμματισμού (C στην περίπτωσή μας).

Παρακάτω θα παρουσιάσουμε σύντομα τις συναρτήσεις εκείνες της βιβλιοθήκης MPI που θα χρησιμοποιήσουμε στην υλοποίηση του προγράμματος μας.



2.2.2.5 Η συνάρτηση ***MPI_Init***

Ο τύπος της συνάρτησης με τα ορίσματα της είναι : `MPI_Init(&argc, &argv);`

Η συνάρτηση αυτή γίνεται σε κάθε επεξεργαστή και επιτρέπει στο σύστημα να προετοιμαστεί ώστε να αρχίσει να δέχεται κλήσεις μηνυμάτων MPI από τους υπόλοιπους επεξεργαστές.

Η κλήση αυτής της συνάρτησης θα πρέπει να γίνεται πρίν από οποιαδήποτε άλλη κλήση συνάρτησης τύπου MPI.

2.2.2.6 Το αντικείμενο ***MPI_COMM_WORLD***

Το `MPI_COMM_WORLD` είναι ένα αντικείμενο (τύπου communicator) στο οποίο κάθε ενεργός επεξεργαστής είναι μέλος. Όταν αρχικοποιείται το περιβάλλον MPI με την κλήση `MPI_Init` τότε, αυτόματα, κάθε επεξεργαστής περιλαμβάνεται στο `MPI_COMM_WORLD`. Όπως θα δούμε παρακάτω, μέσω του αντικειμένου αυτού μπορούμε να έχουμε πρόσβαση σε ιδιότητες του cluster των επεξεργαστών μας, όπως το πλήθος αυτών ή το αναγνωριστικό του κάθε επεξεργαστή.

2.2.2.7 Οι συναρτήσεις ***MPI_Comm_rank*** και ***MPI_Comm_size***

Ο τύπος των συναρτήσεων αυτών με τα ορίσματα τους είναι:

`MPI_Comm_rank(MPI_COMM_WORLD, &id)`

`MPI_Comm_size(MPI_COMM_WORLD, &p)`

Η `MPI_Comm_rank` τοποθετεί στην μεταβλητή `id` το αναγνωριστικό του τρέχοντος επεξεργαστή, το οποίο μπορεί να έχει τιμές από 0 εώς το πλήθος των επεξεργαστών - 1. Συνήθως χρησιμοποιούμε το επιστρεφόμενο `id` σε συνδυασμό με κάποια δομή ελέγχου για να δώσουμε στον τρέχοντα επεξεργαστή να καταλάβει ποιο κομμάτι των δεδομένων ή του υπολογισμού είναι δικό του.

Η `MPI_Comm_size` τοποθετεί στην μεταβλητή `p` το πλήθος των ενεργών επεξεργαστών που υπάρχουν στο `MPI_COMM_WORLD`, δηλαδή στο σύνολο του cluster μας.

2.2.2.8 Η συνάρτηση ***MPI_Finalize***

Εκτελείται αφού μία διεργασία έχει τελειώσει τις κλήσεις MPI ώστε να απελευθερωθούν πόροι (π.χ. μνήμη)



2.2.2.9 Η συνάρτηση *MPI_Reduce*

Ο τύπος της συνάρτησης με τα ορίσματα της είναι :

MPI_Reduce (

```
void    *operand,  
void    *result,  
int     count,  
MPI_Datatype type,  
MPI_Op      operator,  
int     root,  
MPI_Comm   comm. )
```

Τα ορίσματα και η λειτουργία τους :

void *operand: είναι ο δείκτης του στοιχείου το οποίο θα υποστεί την πράξη.

void *result: είναι ο δείκτης του τελικού αποτελέσματος της πράξης. Είναι ορισμένος μόνο για τον επεξεργαστή με id root.

int count: εάν ο αριθμός αυτός είναι μεγαλύτερος του 1, τότε η πράξη γίνεται σε μία συνεχόμενη περιοχή της μνήμης.

MPI_Datatype type: είναι ο τύπος των δεδομένων πάνω στο οποίον θα εκτελεστεί η πράξη. Οι διάφοροι τύποι δεδομένων που καθορίζονται από την βιβλιοθήκη MPI φαίνονται στο παράρτημα.

MPI_Op operator: είναι η πράξη που θα εκτελεσθεί στα δεδομένα. Στο παράρτημα δίνεται ο πίνακας ε τις σχετικές δυνατές πράξεις.

int root: είναι το αναγνωριστικό (id) του επεξεργαστή που θα πάρει το τελικό αποτέλεσμα *result.

MPI_Comm comm: είναι το αντικείμενο τύπου *MPI_Comm* που ορίζει στους επεξεργαστές ποιας ομάδας θα γίνει η πράξη της μείωσης.

Η συνάρτηση αυτή χρησιμοποιείται για να υπολογιστεί για κάθε τοπική μεταβλητή μία καθολική πράξη (άθροισμα, γινόμενο, κλπ). Για παράδειγμα εάν ο κάθε επεξεργαστής κατέχει το μέρος ενός αθροίσματος το οποίο θα πρέπει να υπολογιστεί και να επιστραφεί σαν απάντηση, αντί ο κάθε επεξεργαστής να στείλει στον root επεξεργαστή την τοπική του



μεταβλητή, ώστε αυτός να υπολογίσει από το σύνολο των τοπικών μεταβλητών το καθολικό άθροισμα, με την MPI_Reduce υπολογίζεται βέλτιστα και δενδροειδώς το άθροισμα και επιστρέφεται στον root επεξεργαστή.

| Name | Meaning |
|------------|---------------------------------|
| MPI_BAND | Bitwise and |
| MPI_BOR | Bitwise or |
| MPI_BXOR | Bitwise exclusive or |
| MPI_LAND | Logical and |
| MPI_LOR | Logical or |
| MPI_LXOR | Logical exclusive or |
| MPI_MAX | Maximum |
| MPI_MAXLOC | Maximum and location of maximum |
| MPI_MIN | Minimum |
| MPI_MINLOC | Minimum and location of minimum |
| MPI_PROD | Product |
| MPI_SUM | Sum |

Πίνακας 7, Πράξεις Μείοσης για την MPI_Reduce

2.2.2.10 Η συνάρτηση MPI_Bcast

Ο τύπος της συνάρτησης με τα ορίσματα της είναι :

```
MPI_Bcast (
```

void *message,

int count,

MPI_Datatype datatype,

int root,

MPI_Comm comm.)

Τα ορίσματα και η λειτουργία τους :

void *message: είναι ο δείκτης της αρχικής θέσης μνήμης του στοιχείου που θα μεταδοθεί.

int count: είναι ο αριθμός των θέσεων μνήμης που θα μεταδωθεί.

MPI_Datatype type: είναι ο τύπος των δεδομένων Οι διάφοροι τύποι δεδομένων που καθορίζονται από την βιβλιοθήκη MPI φαίνονται στο παράρτημα.

int root: είναι το αναγνωριστικό (id) του επεξεργαστή από τον οποίον θα μεταδωθεί το μήνυμα.



MPI_Comm comm: είναι το αντικείμενο τύπου MPI_Comm που ορίζει στους επεξεργαστές ποιας ομάδας θα γίνει η μετάδοση.

Με αυτή την συνάρτηση, ένας επεξεργαστής μπορεί να αποστείλει σε όλους τους υπόλοιπους επεξεργαστές κάποια τοπική του μεταβλητή. Ωστόσο η συνάρτηση θα πρέπει να κληθεί από όλους τους επεξεργαστές της ομάδας που εμπλέκονται σε αυτή την συλλογική επικοινωνία.

2.2.2.11 Οι συναρτήσεις MPI_Wtime και MPI_Wtick

double MPI_Wtime (void)

double MPI_Wtick (void)

Με τις συναρτήσεις αυτές μπορούμε να μετρήσουμε τον χρόνο που πέρασε για να εκτελεστεί ένα κομμάτι κώδικα.

Η συνάρτηση MPI_Wtime μετράει τον χρόνο που πέρασε από κάποια στγμή στο παρελθόν. Καλώντας την στην αρχή και στο τέλος του εκτελέσιμου κώδικα που θέλουμε να χρονομετρήσουμε και υπολογίζοντας την διαφορά των δύο επιτρεφόμενων τιμών, έχουμε τον χρόνο που μεσολάβησε.

Η συνάρτηση MPI_Wtick επιστρέφει την ακρίβεια της MPI_Wtime.

2.2.2.12 Η συνάρτηση MPI_Barrier

Η συνάρτηση αυτή είναι: int MPI_Barrier (MPI_Comm comm).

Καλώντας αυτή την συνάρτηση, όλοι οι επεξεργαστές του comm γκρούπ συγχρονίζονται. Δηλαδή η επόμενη εντολή από την MPI_Barrier ξεκινάει την ίδια στιγμή για όλους.

2.2.2.13 Η συνάρτηση MPI_Send

Ο τύπος της συνάρτησης με τα ορίσματα της είναι :

MPI_Send (

 void *message,

 int count,

 MPI_Datatype type,



```
int      dest,  
int      tag,  
MPI_Comm comm)
```

Τα ορίσματα και η λειτουργία τους :

void *message: είναι ο δείκτης του στοιχείου το οποίο θα σταλεί.

int count: είναι ο αριθμός των θέσεων μνήμης που θα μεταδωθεί.

MPI_Datatype type: είναι ο τύπος των δεδομένων .

int dest: είναι το αναγνωριστικό του επεξεργαστή που περιμένουμε να λάβει τα δεδομένα.

int tag: είναι ένας αριθμός που μπορεί να καθορίσει ο προγραμματιστής, και ο οποίος μπορεί να χρησιμεύσει για να διαφοροποιήσει τα δεδομένα που θα σταλούν σε περισσότερες από μία κλήσεις MPI_Send μεταξύ τους. Στο MPI_Receive θα πρέπει να δωθεί το ίδιο tag.

MPI_Comm comm: είναι η ομάδα επεξεργαστών στην οποία ανήκει ο dest.

Η προφανής λειτουργία αυτής της συνάρτησης, όπως και της MPI_Receive είναι η απευθείας ανταλλαγή δεδομένων μεταξύ δύο επεξεργαστών.

2.2.2.14 Η συνάρτηση ***MPI_Receive***

Ο τύπος της συνάρτησης με τα ορίσματα της είναι :

```
MPI_Receive (   
    void    *message,  
    int     count,  
    MPI_Datatype type,  
    int     source,  
    int     tag,  
    MPI_Comm   comm,  
    MPI_Status  *status )
```

Τα ορίσματα και η λειτουργία τους :

void *message: είναι ο δείκτης του στοιχείου στο οποίο θα αποθηκευθούν τα δεδομένα που θα ερθουν.



int count: είναι ο αριθμός των θέσεων μνήμης που θα καταλαμβάνουν τα δεδομένα που θα έρθουν.

MPI_Datatype type: είναι ο τύπος των δεδομένων

int source: είναι το αναγνωριστικό του επεξεργαστή που περιμένουμε να στείλει τα δεδομένα.

int tag: όπως στην MPI_Send

MPI_Comm comm: είναι η ομάδα επεξεργαστών στην οποία ανήκει ο source

MPI_Status *status: είναι μία μεταβλητή τύπου MPI_Status που περιέχει πληροφορίες για την έκβαση της λήψης των δεδομένων, αφού έχει ολοκληρωθεί.

Όπως και στην MPI_Send, εάν κάτι δεν πάει καλά, υπάρχει πρόβλημα να έχουμε κόλλημα (deadlock) του προγράμματος μας. Για παράδειγμα εάν υπάρξει εντολή MPI_Receive για κάποιον επεξεργαστή χωρίς να υπάρχει κατάλληλη εντολή MPI_Send, ο πρώτος επεξεργαστής θα κολλήσει.

2.2.3 Η βιβλιοθήκη MPE

Η MPE είναι μία βιβλιοθήκη για profiling και tracing σε προγράμματα MPI και συμπεριλαμβάνεται ως μέρος της διανομής της έκδοσης MPICH (MPE2 και MPICH2 πλέον). Λειτουργεί επίσης και με άλλες υλοποιήσεις MPI και για το λόγο αυτό μπορεί κανείς να το κατεβάσει και εγκαταστήσει μόνο του (ανεξάρτητα του MPICH). Η MPE καλείται κατά το τρέξιμο (εντολή τρεξίματος μαζί με κατάλληλες παραμέτρους και χρήση συναρτήσεων ανάλογα με το τι ζητείται) ενός προγράμματος MPI και παράγει για περαιτέρω διαγνωστικό έλεγχο κλπ συγκεκριμένα logfiles. Τα logfiles αυτά μπορεί κανείς για να το ερμηνεύσει και κατανοήσει καλύτερα (από άποψη πληροφοριών σε σχέση με τους χρόνους οι οποίοι επιτυγχάνονται σε κάθε επεξεργαστή για ‘υπολογισμούς’ (computation) και ‘επικοινωνία’ (communication) να το ανοίξει και μελετήσει οπτικά (visually) με χρήση του γραφικού εργαλείου ‘Jumpshot’ (logfile visualiser) το οποίο διατίθεται επίσης μαζί με την αντίστοιχη έκδοση του MPI. Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να ανατρέξει στα [26] και [27].



3 Σχεδιασμός και Υλοποίηση

3.1 Σχετικές Εργασίες

Σε αυτή την ενότητα θα παρουσιάσουμε τις πηγές από τις οποίες αντλήσαμε κατά κύριο λόγο την θεωρητική και αλγοριθμική τεκμηρίωση της παραλληλοποίησης της μεθόδου Simplex με την μέθοδο του πλήρους πίνακα.

3.1.1 Άρθρο: Towards a practical parallelisation of the simplex method, J. Hall

Στο άρθρο αυτό αφού γίνεται μία γρήγορη παρουσίαση της μεθόδου simplex τόσο στην μορφή του πίνακα όσο και στην αναθεωρημένη μορφή. Αναλύονται οι ιδιαιτερότητες που παρουσιάζουν τα περισσότερα πραγματικά γραμμικά προβλήματα και γίνεται μία λεπτομερής ανασκόπηση (review) της έως τώρα προόδου στην παραλληλοποίηση της μεθόδου simplex. Τα κυριότερα σημεία που επισημαίνονται είναι:

Τα περισσότερα πρακτικά – πραγματικά γραμμικά προβλήματα αντιστοιχούν σε αραιούς πίνακες (sparse matrixes) και για αυτό η αναθεωρημένη έκδοση του αλγορίθμου είναι και η ταχύτερη.

Εκτός από την πυκνότητα του πίνακα του προβλήματος, σημαίνοντα ρόλο στην ανάπτυξη αποδοτικών αλγορίθμων έχει και η δομή του πίνακα (π.χ. άνω / κάτω τριγωνικός, κλπ).

Οι περισσότερες προσπάθειες παραλληλοποίησης αποσκοπούσαν στο να αναπτυχθούν τεχνικές οι οποίες να είναι αποδοτικές και όχι να μελετηθεί η αλγοριθμική πολυπλοκότητα που υπέκρυπτε το εγχείρημα (Little thought, if any, has been given to the underlying computational scheme in terms of serial efficiency and numerical robustness).

Η συντριπτική πλειοψηφία των προσπαθειών παραλληλοποίησης χρησιμοποιεί σαν αφετηρία την μέθοδο του πλήρους πίνακα και όχι την αναθεωρημένη έκδοση.

Αντού του είδους οι υλοποιήσεις, με τις απλές δομές δεδομένων και την δυνατότητα που υπάρχει για αύξηση της ταχύτητας με την αύξηση των επεξεργαστών (scalability), είναι μία αρκετά καλή εξάσκηση στον παράλληλο υπολογισμό.



Ωστόσο, η ταχύτητα που προκύπτει από τις παραπάνω προσπάθειες παραλληλοποίησης (με την μέθοδο του πλήρους πίνακα) είναι συγκρίσιμη με την σειριακή simplex της αναθεωρημένης έκδοσης.

Η πραγματική πρόκληση στην παραλληλοποίηση της simplex είναι η ανάπτυξη παράλληλων αλγορίθμων που θα βασίζεται στην αναθεωρημένη έκδοση.

3.1.2 Άρθρο: A Distribute, Scaleable Simplex Method, G. Yarmish & R. V. Slyke

Στο άρθρο αυτό παρουσιάζεται μία απλή, επεκτάσιμη (scaleable) και κατανεμημένη εφαρμογή του αλγορίθμου simplex για την επίλυση μεγάλων γραμμικών προβλημάτων. Η παραλληλοποίηση γίνεται με **coarsed grain** τρόπο και είναι σχεδιασμένη για να εκτελείται σε αίθουσες υπολογιστών (optimized for loosely coupled workstations). Η γενική περιγραφή του αλγορίθμου που παρουσιάζεται έχει ως εξής:

Εάν απλοποιήσουμε την σειριακή μέθοδο simplex στα τρία παρακάτω βήματα:

- Επιλογή στήλης (μεταβλητής εισόδου)
- Επιλογή γραμμής (μεταβλητής εξόδου)
- Εναλλαγή μεταβλητών και περιστροφή του πίνακα (pivot)

τότε ο προτεινόμενος αλγόριθμος εκτελεί τα παρακάτω:

- Καταρχήν διαμοιράζει τις στήλες του πίνακα στους επεξεργαστές.
- Κάθε επεξεργαστής εκτελεί τοπική επιλογή της καλύτερης στήλης του.
- Όλοι οι επεξεργαστές επικοινωνούν μεταξύ τους την τιμή (pricing) της καλύτερης στήλης τους. Στο τέλος του βήματος αυτού όλοι οι επεξεργαστές θα γνωρίζουν ποιος επεξεργαστής έχει την «καλύτερη» στήλη.
- Ο επεξεργαστής με την καλύτερη στήλη την μεταδίδει στους υπόλοιπους και στην συνέχεια γίνεται στους επεξεργαστές η επιλογή της γραμμής.
- Οι επεξεργαστές κάνουν την εναλλαγή των μεταβλητών και τους υπολογισμούς των στηλών τις οποίες κατέχουν.

Επίσης, στο άρθρο αυτό παρουσιάζεται και ένα μοντέλου θεωρητικού υπολογισμού της απόδοσης του προτεινόμενου αλγορίθμου συναρτήσει του μεγέθους του προβλήματος και του αριθμού των επεξεργαστών.



3.1.3 Παρουσίαση σε Συνέδριο: Some Computational Results on MPI Parallel Implementation of Dense Simplex Method, El-Said Badr et al.

Στην παρουσίαση αυτή δίνεται η υλοποίηση ενός παράλληλου simplex αλγορίθμου με την χρήση MPI, σε μορφή ψευδοκώδικα και παρουσιάζεται η επιτάχυνση της επίλυσης του προβλήματος σε σχέση με τον αριθμό των επεξεργαστών και το μέγεθος του προβλήματος.

Ο διαχωρισμός των δεδομένων γίνεται ανά γραμμές του πίνακα. Ο επεξεργαστής 0 είναι υπεύθυνος για τους συντελεστές των μεταβλητών της αντικειμενικής συνάρτησης, δηλαδή για την επιλογή της εισερχόμενης μεταβλητής καθώς και για τον υπολογισμό των νέων συντελεστών κατά της περιστροφή του πίνακα..

Όσον αφορά τα αποτελέσματα, οι μετρήσεις γίνανε σε τυχαία προβλήματα μεγέθους 200x200, 300x300 και 400x400, και διαπιστώθηκε ότι αυξανομένου του μεγέθους του προβλήματος, η επιτάχυνση μεγαλώνει.



3.2 Παραλληλοποίηση της μεθόδου Simplex

Ο παράλληλος αλγόριθμός τον οποίο αναπτύξαμε για την επίλυση ενός προβλήματος γραμμικού προγραμματισμού που είναι σε κανονική μορφή, μοιάζει σε μεγάλο βαθμό με αυτόν που ανέπτυξε ο Yarmish στο άρθρο του [7]. Ο διαμοιρασμός του πίνακα των περιορισμών και του διανύσματος των συντελεστών της αντικειμενικής συνάρτησης γίνεται κατά στήλες στις διεργασίες.

3.2.1 Διαμοιρασμός των δεδομένων

3.2.1.1 Επιλογές

Υπήρχαν δύο επιλογές ως προς τον διαμερισμό των δεδομένων [5] :

- **Interleaved Data Decomposition:** Σε αυτόν το τύπο διαμερισμού δεδομένων, η διεργασία k από σύνολο p διεργασιών είναι υπεύθυνη για τα στοιχεία $k, k+p, k+2p, k+3p \dots k+np$. Για το στοιχείο του προβλήματος με δείκτη i γνωρίζουμε ότι η διεργασία η οποία είναι υπεύθυνη για αυτόν είναι $i \bmod p$.
- **Block Data Decomposition:** Εδώ, τα δεδομένα χωρίζονται σε συνεχόμενα μπλοκ, περίπου ίδιου μεγέθους. Εάν n είναι ο συνολικός αριθμός των δεδομένων και p ο αριθμός των επεξεργαστών, το πρώτο στοιχείο που ελέγχεται από τον επεξεργαστή k είναι ο κάτω ακέραιος του $i \bmod p$ και το τελευταίο στοιχείο είναι ο κάτω ακέραιος του $(i+1)n/p - 1$. Το i -οστό στοιχείο ελέγχεται από τον επεξεργαστή: κάτω ακέραιος του $(p(i+1)-1)/n$.

3.2.1.2 Διαμέριση των μεταβλητών του προβλήματος

Επιλέξαμε ο τρόπος διαμέρισης και διαμοιρασμού των δεδομένων (data decomposition) του προβλήματος να είναι ο δεύτερος, καθώς υπερέχει του πρώτου σε αποδοτικότητα όταν θέλουμε να διαμοιράσουμε δεδομένα.

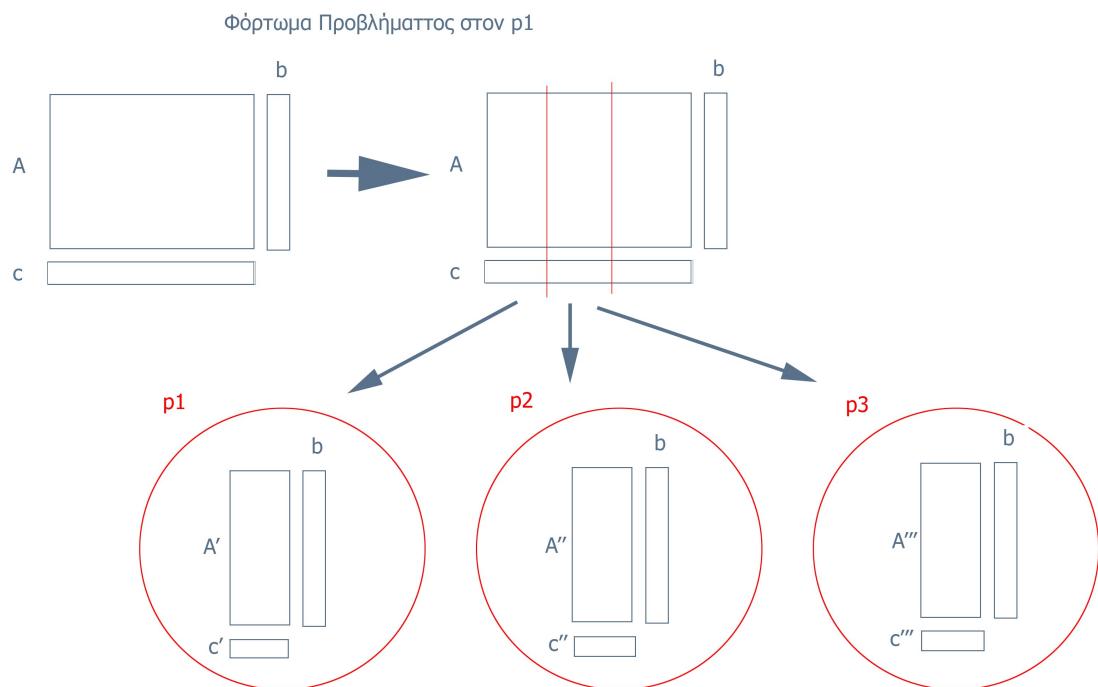


Έτσι ο πίνακας των περιορισμών μοιράζεται σε κάθε μία από τις διεργασίες κατά στήλες και κατά Block data decomposition. Το διάνυσμα των συντελεστών της αντικειμενικής συνάρτησης σπάει και αυτό και μοιράζεται στις διεργασίες με τον ίδιο τρόπο.

Το δεξί μέρος των περιορισμών, επειδή χρειάζεται ολόκληρο από όλους τους επεξεργαστές, θα ήταν ασύμφορο να σπάσει καθώς θα προκαλούσε πολύ συχνή επικοινωνία. Έτσι μοιράζεται εξ' αρχής ολόκληρο σε όλες τις διεργασίες και η κάθε διεργασία το υπολογίζει σε κάθε επανάληψη. Άλλωστε η υπολογιστική πολυπλοκότητα μίας πράξης είναι η ίδια είτε συμβαίνει σε μία διεργασία είτε ταυτόχρονα σε η διεργασίες.

Το ίδιο συμβαίνει και με το διάνυσμα (σετ) των βασικών και μη βασικών μεταβλητών καθώς και με το διάνυσμα που αποθηκεύει το είδος της μεταβλητής (απόφαση, πλεονάσματος ή τεχνητή).

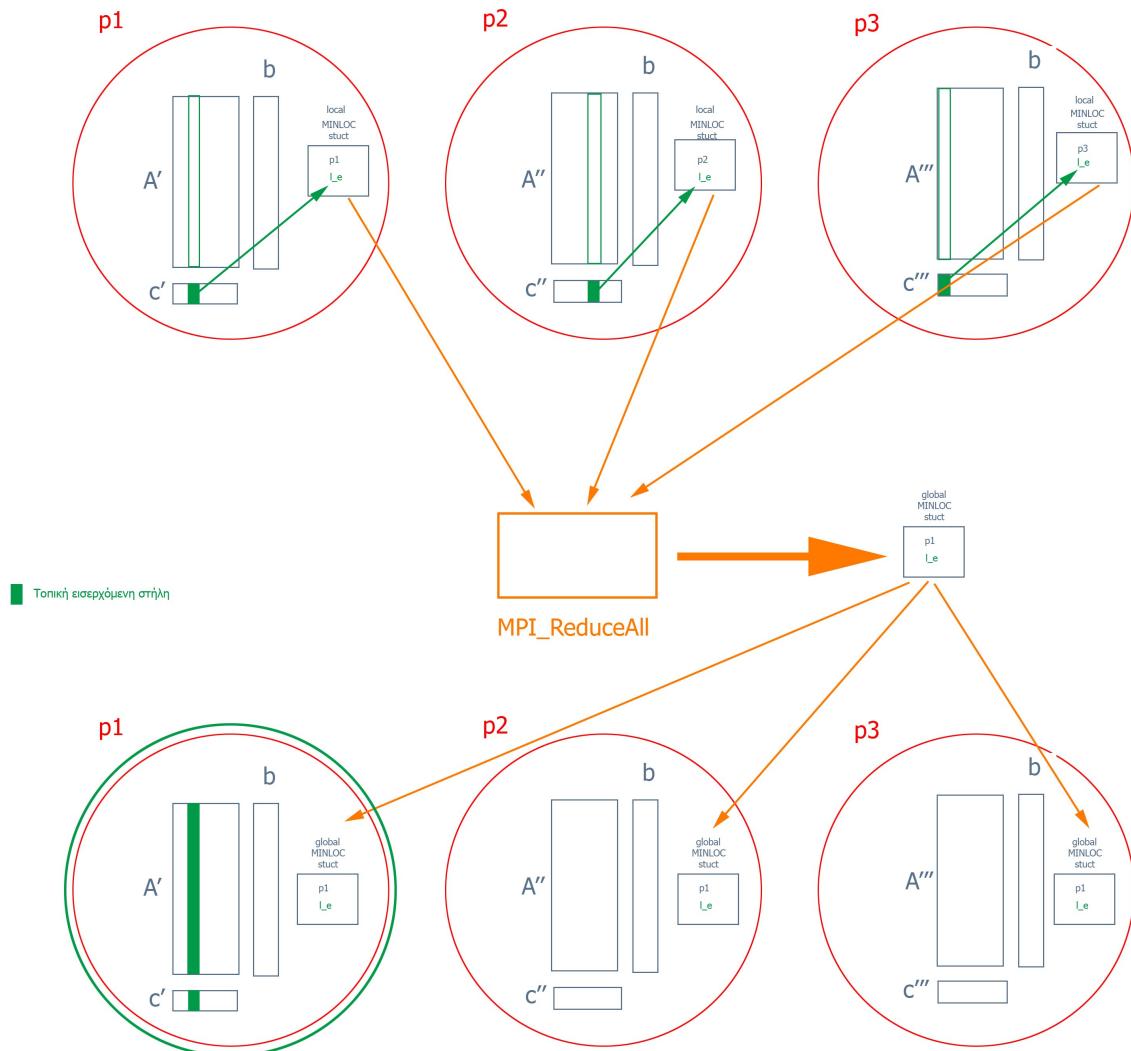
Παρακάτω, φαίνεται σχηματικά ο τρόπος διαμοιρασμού των δεδομένων:



3.2.2 Ο Αλγόριθμος

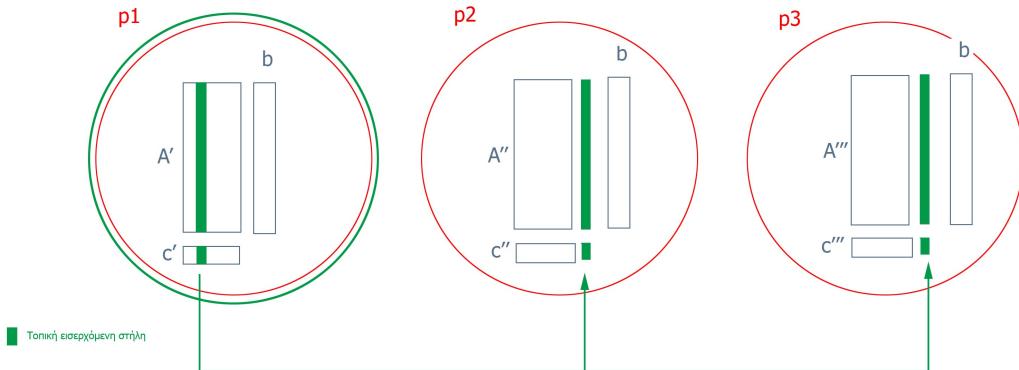
Σε γενικές γραμμές ο αλγόριθμός, μετά τον διαμοιρασμό των δεδομένων, έχει ως εξής:

- Η κάθε διεργασία επιλέγει την τοπικά καλύτερη εισερχόμενη μεταβλητή με κριτήριο τον αρνητικότερο συντελεστή στην αντικειμενική συνάρτηση (κριτήριο Dantzig)
- Με καθολική μείωση (MPI_Reduce_All), όλες οι διεργασίες γνωρίζουν την στήλη με τον καθολικά αρνητικότερο συντελεστή και σε ποια διεργασία αυτή βρίσκεται

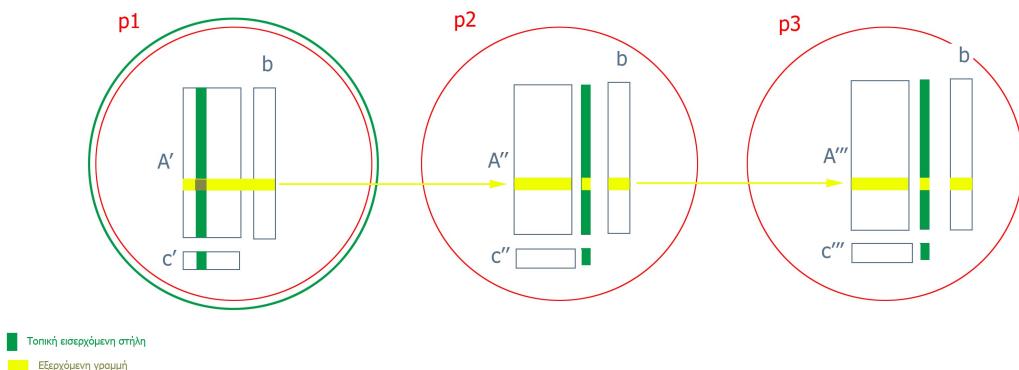




- Γίνεται μετάδοση (Bcast) της στήλης από την διεργασία που την κατέχει προς τις υπόλοιπες.



- Η διεργασία με την καθολικά εισερχόμενη μεταβλητή(δηλαδή την επιλεχθείσα στήλη) υπολογίζει την εξερχόμενη μεταβλητή με το κριτήριο του ελάχιστου λόγου και μεταδίδει τον αριθμό αυτής της γραμμής στις υπόλοιπες



- Όλες οι διεργασίες εκτελούν τους υπολογισμούς για το τμήμα του πίνακα που τους αναλογεί ώστε να εκτελεστεί η περιστροφή (pivot)
- Τα παραπάνω βήματα επαναλαμβάνονται μέχρι, είτε να μην υπάρχει αρνητικός συντελεστής στην αντικειμενική συνάρτηση (άριστο σημείο), είτε να μην μπορεί να βρεθεί εξερχόμενη μεταβλητή (μη φραγμένο πρόβλημα)



3.3 Υλοποίηση και Σχεδιαστικές Επιλογές

Στην ενότητα αυτή θα παρουσιάσουμε την υλοποίηση του παράλληλου αλγορίθμου simplex και διάφορες σχεδιαστικές επιλογές. Στο παράρτημα 4.3 υπάρχει ο κώδικας, τεκμηριωμένος με εκτενή σχόλια.

Θα πρέπει εδώ να παρατηρήσουμε ότι η υλοποίηση το αλγορίθμου simplex δεν είναι μία καθαρή υλοποίηση πλήρους πίνακα, καθώς στα παρακάτω σημεία έχουν επιλεγεί προσεγγίσεις που χρησιμοποιούνται στην αναθεωρημένη έκδοση [3]:

- Μη αποθήκευση των στηλών των βασικών μεταβλητών, αφού εξ ορισμού σχηματίζουν έναν μοναδιαίο πίνακα
- Μη ενημέρωση των συντελεστών της αντικειμενικής συνάρτησης για τις εξερχόμενες μεταβλητές αφού εξ ορισμού θα είναι ίσοι με μηδέν

3.3.1 Περιβάλλον υλοποίησης

Σαν γλώσσα προγραμματισμού χρησιμοποιήθηκε η ANSI C και σαν compiler ο GNU GCC. Οι επιλογές αυτές έγιναν για τους παρακάτω λόγους:

- Ο compiler παράγει αποδοτικό κώδικα
- Προτιμήθηκε σε σχέση με την Java γιατί η υλοποίηση της MPI βιβλιοθήκης είναι πιο αποδοτική
- Προτιμήθηκε σε σχέση με την C++, γιατί η δομή της είναι πιο απλή, οπότε θα είχαμε τον απαιτούμενο χρόνο να ασχοληθούμε με τον ίδιο τον αλγόριθμο.

Μόνο εμφανές μειονέκτημα της επιλογής της C ως γλώσσα υλοποίησης είναι η δυσκολία επαναχρησιμοποίησης του κώδικα, τόσο για την ανάπτυξη περαιτέρω εφαρμογών όσο και χρήσης του από τρίτους. Ωστόσο, η μικρή σχετικά έκταση του αλγορίθμου κάνει εφικτή, χωρίς ιδιαίτερο κόπο, την μετατροπή του υφιστάμενου κώδικα C σε κώδικα C++ με αντικειμενοστρεφή κατεύθυνση, εα μελλοντικά χρειαστεί.



Σαν περιβάλλον προγραμματιστικής ανάπτυξης χρησιμοποιήσαμε το Netbeans 6.5 και το Plugin C/C++. Όλη η εφαρμογή αναπτύχθηκε σε περιβάλλον Linux, έτσι ώστε να υπάρχει συμβατότητα με το τελικό περιβάλλον εκτέλεσης που θα ήταν και αυτό σε Linux.

Σαν βιβλιοθήκη MPI χρησιμοποιήθηκε η MPICH2, που είναι μία από τις γνωστότερες υλοποιήσεις της βιβλιοθήκης MPI για τις γλώσσες C, C++ και Fortran [<http://www.mcs.anl.gov/research/projects/mpich2/>, Argonne National Laboratory] και υπάρχει για αυτήν μεγάλης κλίμακας τεκμηρίωση και σχετικό υλικό υποστήριξης στο διαδίκτυο.

3.3.2 Περιβάλλον εκτέλεσης

Το περιβάλλον στο οποίο έγιναν οι υπολογισμοί μας, ήταν το εργαστήριο παράλληλης επεξεργασίας του TEI Αθήνας, το οποίο αποτελείται από μία συστοιχία (cluster) 8 υπολογιστών INTEL XEON (με hyperthreading – πράγμα που επιτρέπει ανάλογα με τη φύση της εφαρμογής αποδοτική προσομοίωση έως και 16 επεξεργαστών υπό προϋποθέσεις) που είναι συνδεδεμένοι μεταξύ τους σε δίκτυο Myrinet υψηλής ταχύτητας (2GBps).

3.3.3 Τα δεδομένα εισόδου

Σαν αρχεία εισόδου επιλέξαμε να είναι αρχεία τύπου MPS. Περισσότερες πληροφορίες για την δομή και την ιστορία αυτών των αρχείων περιγραφής γραμμικών προβλημάτων στο site του LP_Solve και στον [8].

Επιλέξαμε να χρησιμοποιήσουμε την δουλειά που ήδη είχε γίνει από το LP_Solve για το διάβασμα των αρχείων αυτών. Αντί να χρησιμοποιήσουμε τον πηγαίο κώδικα του LP_Solve, γεγονός που θα προκαλούσε διαχειριστικά προβλήματα, αποφασίσαμε να κάνουμε dynamic linking το πρόγραμμα μας με την βιβλιοθήκη (shared object library) του LP_Solve.

Επίσης για την δημιουργία των τυχαίων προβλημάτων χρησιμοποιήσαμε το πρόγραμμα matgen [9], με το οποίο δημιουργήσαμε αραιούς πίνακες δαφόρων μεγεθών. Στην συνέχεια τους εισαγάγαμε στο matlab και από εκεί, θέτοντας σαν διανύσματα του δεξιού μέρους των περιορισμών και των συντελεστών της αντικειμενικής συνάρτησης τα μοναδιαία διανύσματα, τα εξαγάγαμε σε αρχείο τύπου MPS.



Επίσης χρησιμοποιήσαμε και προβλήματα διαφόρων μεγεθών από την βιβλιοθήκη NetLib.
Στον παρακάτω πίνακα φαίνονται τα προβλήματα που χρησιμοποιήσαμε:

3.3.4 Οργάνωση των αρχείων (modules)

Καθώς αρκετές μεταβλητές ήταν κοινές για όλες τις συναρτήσεις όλων των αρχείων που αναπτύξαμε, θεωρήσαμε πιο λειτουργικό να ορίσουμε σαν καθολικά εξωτερικές (extern) μεταβλητές τις εξής:

- Τον αριθμό της τρέχουσας διεργασίας και τον συνολικό αριθμό διεργασιών.
- Τον αριθμό των βημάτων της Φάσης I και II και το σε ποια Φάση βρισκόμαστε.
- Τον τοπικό και καθολικό αριθμό των στηλών του προβλήματος καθώς και τον καθολικό αριθμό γραμμών αυτού.
- Τον τοπικό πίνακα των περιορισμών και τα τοπικά διανύσματα των συντελεστών της αντικειμενικής συνάρτησης(Φάσεις I και II). Το καθολικό διάνυσμα του δεξιού μέρους των περιορισμών και τις τρέχουσες αποτυμήσεις των αντικειμενικών συνάρτησεων.
- Τα σετ του ποιες μεταβλητές είναι βασικές και μη βασικές καθώς και το είδος της κάθε μεταβλητής (surplus, slack, artificial, decision)
- Το επίπεδο εκσφαλμάτωσης και το εάν θα παράγονται MPE στατιστικά

Πέραν αυτών των εξωτερικών μεταβλητών, τα αρχεία κώδικα και η λειτουργία τους είναι τα εξής:

3.3.4.1 *mpi_simplex_v2.h*

Στο αρχείο αυτό:

- Δηλώνονται τα header files που περιέχουν συναρτήσεις βιβλιοθηκών (π.χ. stdio.h, stdlib.h, math.h κ.α.)
- Γίνονται define τιμές που χρησιμοποιούνται από όλα τα υπόλοιπα αρχεία (π.χ. EPS_VALUE, ποιος επεξεργαστής είναι ο ROOT, κ.α.).
- Γίνονται define κάποιες χρήσιμες MACRO (π.χ. double_equals για το εάν ένας αριθμός είναι 0 ή όχι, BLOCK_LOW που δηλώνει για κάποια διεργασία, ποιο είναι το μικρότερο στοιχείο από τον καθολικό πίνακα ή διάνυσμα που της αντιστοιχεί)



- Ορίζονται structures που θα μας είναι χρήσιμες (lp_general_form, lp_standard_form)

3.3.4.2 mpi_simplex_v2_main.c

Είναι το αρχείο το οποίο περιέχει την main, δηλαδή την συνάρτηση από την οποία ξεκινάει η εκτέλεση του κώδικα. σε αυτό το αρχείο, με την σειρά εκτέλεσης. Μέσα σε αγκύλη η εμβέλεια του κώδικα:

1. [Όλες] Δηλώνονται οι καθολικές μεταβλητές του προγράμματος μας,
2. [Όλες] Αρχικοποιείται το MPI και το MPE περιβάλλον,
3. [ROOT] Φορτώνεται το MPS πρόβλημα στην δομή lp_lp_standard_form
4. [Όλες] Γίνεται ο διαμοιρασμός των δεδομένων από την ROOT στις υπόλοιπες διεργασίες
5. [Όλες] Προχωράει στην επίλυση του προβλήματος για την Φάση I
6. [Όλες] Ο ROOT μεταδίδει το αποτέλεσμα της Φάσης I σε όλες τις υπόλοιπες διεργασίες
7. [Όλες] Ελέγχει την επιστροφή της διαδικασίας επίλυσης της Φάσης I. Εάν είναι αποδεκτή (1) τότε προχωράει στην επίλυση του προβλήματος για την Φάση II,
8. [Όλες] Τερματίζεται το περιβάλλον MPI και MPE

3.3.4.3 mpi_load_mps.c

Στο αρχείο αυτό περιέχονται όλες οι συναρτήσεις που είναι υπεύθυνες για την φόρτωση του προβλήματος MPS σε μορφή τέτοια ώστε να είναι δυνατή η επίλυση του. Πιο συγκεκριμένα:

- mpi_load_mps: που εκτελεί τις παρακάτω διαδικασίες,
- mpi_read_mps: διαβάζει το MPS αρχείο σε ένα struct τύπου lp_general_form
- mpi_convert_to_std_form: μετατρέπει το πρόβλημα τύπου lp_general_form σε κανονική μορφή (lp_standard_form), προσθέτοντας ότι surplus, slack ή artificial μεταβλητές είναι απαραίτητο και κανονικοποιούντας τον πίνακα των περιορισμών.



3.3.4.4 *mpi_distr_lp_data.c*

Στο αρχείο αυτό περιέχονται οι συναρτήσεις που είναι υπεύθυνες για τον διαμοιρασμό των δεδομένων του προβλήματος (που είναι στην μορφή lp_standard_form) στις διεργασίες του Communicator. Πιο συγκεκριμένα:

- **mpi_distr_lp_data:** Είναι η βασική συνάρτηση αυτού του module. Αρχικά ο ROOT μεταδίδει σε όλες τις διεργασίες το μέγεθος του προβλήματος, ώστε να γίνουν τα απαραίτητα memory allocation του πίνακα και των διανυσμάτων. Στην συνέχεια ο ROOT, με Send-Receive, μεταδίδει τον πίνακα των περιορισμών και το διάνυσμα των συντελεστών της αντικειμενικής συνάρτησης στις αντίστοιχες διεργασίες.
- Υπάρχουν κάποιες άλλες επικουρικές συναρτήσεις, στις οποίες έχουν ομαδοποιηθεί διεργασίες, π.χ. η αποστολή ενός πίνακα ή ενός διανύσματος από τον ROOT στις υπόλοιπες διεργασίες όταν η αποδόμηση των δεδομένων γίνεται με την μέθοδο column-wise d:atablock.

3.3.4.5 *mpi_simplex.c*

Το αρχείο αυτό περιέχει την κυρίως λογική για την επίλυση του προβλήματος με την μέθοδο simplex. Πιο συγκεκριμένα:

- η συνάρτηση MPI_Simplex:
 - Αρχικά επιλέγεται η τοπικά εισερχόμενη μεταβλητή (στήλη), με την συνάρτηση MPI_ChooseEnteringVariable
 - Με MPI_All_Reduce επιλέγεται η καθολικά εισερχόμενη μεταβλητή (και η διεργασία στην οποία αναλογεί)
 - Στην συνέχεια επιλέγεται από την διεργασία που κατέχει την καθολικά εισερχόμενη μεταβλητή, η καθολικά εξερχόμενη μεταβλητή (γραμμή) με την συνάρτηση MPI_ChooseLeavingVariableIndex και μεταδίδεται προς όλες τις άλλες διεργασίες.
 - Επίσης από αυτή την διεργασία μεταδίδεται και ολόκληρη η στήλη που έχει επιλεγεί για εισερχόμενη, καθώς και οι παράμετροι της αντικειμενικής συνάρτησης ης Φάσης I και II σε αυτή την στήλη.
 - Εάν η γραμμή που έχει επιλεγεί είναι έγκυρη (δηλαδή δεν έχουμε μ φραγμένο πρόβλημα), η κάθε διεργασία εκτελεί το MPI_Pivot για το κομμάτι του πίνακα και των διανυσμάτων που της αναλογούν.



- Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι να μην μπορεί να επιλεγεί εισερχόμενη μεταβλητή.
- Όταν συμβεί το παραπάνω, εάν βρισκόμαστε στην Φάση I, ελέγχει η ROOT διεργασία εάν η τιμή της αντικειμενικής συνάρτησης της Φάσης αυτής ισούται με μηδέν και τότε επιστρέφει 1 (επιτυχία), αλλιώς 0 (αποτυχία). Εάν βρισκόμαστε στην Φάση II, επιστρέφει 1 (επιτυχία)
- η συνάρτηση MPI_ChooseEnteringVariable: Επιλέγει την τοπικά εισερχόμενη μεταβλητή ως εξής:
 - Για κάθε στήλη που ανήκει σε μία διεργασία:
 - Εάν είμαστε στην Φάση 2 και αυτή είναι τεχνητή μεταβλητή, προχωράμε στην επόμενη
 - Επιλέγουμε τον συντελεστή της κατάλληλης αντικειμενικής συνάρτησης (Φάσης I ή Φάσης II)
 - Εάν ο συντελεστής είναι αρνητικός και μικρότερος από το προηγούμενο προσωρινό ελάχιστο, τότε τον θέτουμε ως προσωρινό ελάχιστο (κριτήριο Dantzig)
 - Στο τέλος, αφού έχουμε διατρέξει όλες τις στήλες, έχουμε την θέση και τον συντελεστή του κομματιού της αντικειμενικής συνάρτησης για την οποία είναι υπεύθυνη η διεργασία. Με βάση αυτή την πληροφορία, ενημερώνουμε το struct obj_Coef, το οποίο θα χρησιμοποιηθεί στην καθολική μείωση.
 - Επιστρέφουμε το index του μικρότερου στοιχείου
- η συνάρτηση MPI_ChooseLeavingVariableIndex: Επιλέγει την καθολικά, αφού η επιλογή γίνεται στην διεργασία που είναι υπεύθυνη για αυτό, εξερχόμενη μεταβλητή, ως εξής:
 - Για κάθε γραμμή:
 - Εάν είμαστε στην Φάση II και η μεταβλητή είναι τεχνητή, προχωράμε στην επόμενη γραμμή
 - Εάν το στοιχείο της υποψήφιας γραμμής και της εισερχόμενη στήλης είναι μεγαλύτερο του μηδενός και το στοιχείο του δεξιού μέρους του περιορισμού είναι ίσο ή μεγαλύτερο του μηδενός, υπολογίζεται το πηλίκο των δύο παραπάνω μεγεθών



- Εάν το πηλίκο αυτό είναι μικρότερο από το προσωρινά μικρότερο πηλίκο, τότε το τελευταίο αντικαθίστανται από το πρώτο
- Τελικά, αφού έχουμε περάσει από όλες τις γραμμές, επιστρέφουμε την γραμμή με το μικρότερο πηλίκιο δεξιού μέρους προς στοιχείο πίνακα περιορισμών.
- Η συνάρτηση MPI_Pivot: Εκτελεί την περιστροφή για το κομμάτι που αντιστοιχεί στην κάθε διεργασία. Λαμβάνει σαν εισόδους την καθολικά εξερχόμενη μεταβλητή, την καθολικά και τοπικά εισερχόμενη μεταβλητή, τον επεξεργαστή που κατέχει την εισερχόμενη στήλη, την εισερχόμενη στήλη και τους συντελεστές της αντικειμενικής συνάρτησης σε αυτήν..Η λειτουργία της συνάρτησης έχει ως εξής::
 - Αρχικά, στην οδηγό διεργασία (αυτή που κατέχει την εισερχόμενη στήλη), κατανέμεται χώρος για την στήλη που θα αντικαταστήσει την εισερχόμενη και τους νέους συντελεστές
 - Η εξερχόμενη γραμμή διαιρείται με το στοιχείο περιστροφής (πίνακας περιορισμών, διάνυσμα δεξιού μέρους και νέα στήλη)
 - Γίνονται οι απαραίτητες γραμμοπράξεις ώστε να μηδενισθούν τα στοιχεία της εισερχόμενης στήλης πλην του στοιχείου περιστροφής, το οποίο ισούται ήδη με μονάδα. Οι γραμμοπράξεις γίνονται στον πίνακα των περιορισμών και τα διανύσματα της αντικειμενικής συνάρτησης (ανάλογα με την Φάση), του δεξιού μέρους των περιορισμών και της νέας στήλης και στην τιμή των αντικειμενικών συναρτήσεων.
 - Αντικαθίστανται η παλιά εισερχόμενη στήλη που πλέον είναι της μορφής [0 0 1 0 0] με την νέα, που ουσιαστικά πρόκειται για την στήλη της εξερχόμενης μεταβλητής που μέχρι πριν τους υπολογισμούς ήταν και αυτή της μορφής [0 0 0 1 0 0].
 - Ενημερώνονται τα στοιχεία του Βασικού και του μη-Βασικού σετ.



3.3.5 Το εκτελέσιμο αρχείο

Με το compile των επιμέρους αρχείων και το linking των object files, δημιουργείται το εκτελέσιμο αρχείο. Επιλέξαμε το εκτελέσιμο αρχείο να λαμβάνει τις εξής εισόδους:

- Την διαδρομή του mps αρχείο, δηλαδή του προβλήματος
- Το εάν πρόκειται για πρόβλημα ελαχιστοποίησης (0) ή μεγιστοποίησης (1)
- Τον βαθμό που θέλουμε να εμφανίζονται μηνύματα αποσφαλμάτωσης (0-6)
- Το εάν θέλουμε να χρησιμοποιηθεί το MPE για να δημιουργηθούν τα CLOG αρχεία (1) ή όχι (0)

3.4 Δυνατότητες βελτίωσης του τρέχοντος αλγορίθμου

Θα μπορούσαν να γίνουν οι εξής βελτιωτικές κινήσεις στον τρέχοντα παράλληλο αλγόριθμο simplex, οι οποίες θα μπορούσαν να γίνουν προσθετικά, ώστε κάθε φορά να μετράται η μείωση του χρόνου εκτέλεσης και η αύξηση της επιτάχυνσης.

3.4.1 Υπολογισμός της εξερχόμενης μεταβλητής

Στην τρέχουσα υλοποίηση, ο καθορισμός της εξερχόμενης μεταβλητής γίνεται στην διεργασία στην οποία έχει επιλεγεί η εισερχόμενη μεταβλητή. Στην συνέχεια ο αριθμός της γραμμής που έχει επιλεγεί μεταδίδεται και στις υπόλοιπες διεργασίες. Ωστόσο ο τρόπος αυτός, πέραν του ότι υποχρεώνει, όσο η διεργασία οδηγός κάνει τον υπολογισμό, τις υπόλοιπες διεργασίες σε αδράνεια, προσθέτει καθυστέρηση επικοινωνίας στην όλη διαδικασία.

Οι καθυστερήσεις αυτές θα μπορούσαν να αποφευχθεί εάν όλες οι διεργασίες υπολογίζανε την εξερχόμενη μεταβλητή μετά την λήψη της εισερχόμενης στήλης. Όπως έχουμε επισημάνει παραπάνω, ο ταυτόχρονος υπολογισμός δεν αυξάνει την αλγορίθμική πολυπλοκότητα.



3.4.2 Προσθήκη κριτηρίων επιλογής εισερχόμενης μεταβλητής

Θα είχε ενδιαφέρον να δούμε την διαφοροποίηση στην απόδοση του αλγορίθμου από την χρήση εναλλακτικών μεθόδων επιλογής της εισερχόμενης μεταβλητής. Η μέθοδος επιλογής με το κριτήριο της πιο κατηφορικής ακμής (steepest edge column choice) ενδείκνυται για περιβάλλον παράλληλης επεξεργασίας καθώς ο επιπλέον χρόνος υπολογισμού αντισταθμίζεται από την μείωση του αριθμού των επαναλήψεων, όπως επισημαίνει ο Yarmish [10].

3.4.3 Εφαρμογή προλυτικών (presolving) εργασιών

Κάθε εμπορική υλοποίηση του simplex περιλαμβάνει επιλογές για ενεργοποίηση προλυτικών διαδικασιών οι οποίες αυξάνουν την ακρίβεια της τελικής λύσης αλλά και μειώνουν το μέγεθος του αρχικού πίνακα από το οποίο θα κληθεί να ξεκινήσει ο αλγόριθμος. Τέτοιες προλυτικές εργασίες είναι [8] :

- Τεχνικές κλιμάκωσης, κατά τις οποίες τα στοιχεία του πίνακα των περιορισμών γίνονται πιο κοντινά σε τάξη μεγέθους μεταξύ τους
- Τεχνικές απλοποίησης του αρχικού προβλήματος κατά τις οποίες παράγεται ένα ισοδύναμο πρόβλημα με μικρότερες διαστάσεις. Ενδεικτικά, τέτοιες είναι ο εντοπισμός και η διαγραφή κενών γραμμών και στηλών, γραμμών με ένα μη μηδενικό στοιχείο, στηλών με ένα μη μηδενικό στοιχείο κ.α.

Η διερεύνηση και εφαρμογή των παραπάνω μεθόδων σε περιβάλλον παράλληλου υπολογισμού και η σχέση τους με τον τελικό χρόνο υπολογισμού θα είχε ιδιαίτερο ενδιαφέρον.

3.4.4 Αποτελεσματικότερη διαχείριση της φόρτωσης και διαμοιρασμού των αρχικών δεδομένων

Η τρέχουσα διαδικασία φόρτωσης των αρχικών δεδομένων μπορεί να περιγραφεί συνοπτικά ως εξής:

- Η ROOT διεργασία, μέσω της βιβλιοθήκης του lp_solve, διαβάζει το MPS αρχείο και φορτώνει τους πίνακες στην μνήμη, σε προσωρινές μεταβλητές. Στην συνέχεια κάνει τις απαραίτητες μετατροπές ώστε να είναι το πρόβλημα σε κανονική μορφή.



- Η ROOT διεργασία, διαμοιράζει τα δεδομένα από τις προσωρινές μεταβλητές στις υπόλοιπες διεργασίες, ενώ για τον εαυτό της, αντιγράφει τα κομμάτια που της αντιστοιχούν από τις προσωρινές μεταβλητές στις μεταβλητές που θα χρησιμοποιηθούν κατά την διάρκεια εκτέλεσης του αλγορίθμου.

Η παραπάνω διαδικασία είναι σπάταλη σε μνήμη. Πιο οικονομικό θα ήταν εάν η ROOT διεργασία διάβαζε κάθε φορά μόνο το κομμάτι εκείνο που ανήκε σε μία άλλη διεργασία και της το έστελνε άμεσα. Με τον τρόπο αυτό δεν θα χρειαζόταν μνήμη από την ROOT διεργασία ίση με το μέγεθος του προβλήματος, αλλά υποπολλαπλάσια αυτού.

Επίσης θα είχε νόημα να εξεταστεί κι το εάν η φάση της κανονικοποίησης του προβλήματος θα μπορούσε να γίνεται, όχι στην διεργασία που εκτελεί την φόρτωση, αλλά ετά την φάση του διαμοιρασμού, στις επιμέρους διεργασίες.



4 Πειραματική Αξιολόγηση

4.1 Περιγραφή της Πειραματικής Διαδικασίας

4.1.1 Εκτέλεση πειράματος

Όπως ειπώθηκε και παραπάνω, εκτελέστηκαν 7 συνολικά τρεξίματα 23 προβλημάτων διαφόρων μεγεθών σε 1,2,3,4,5,6,7,8,16 διεργασίες και ενός προβλήματος σε 8 και 16 διεργασίες. 19 από τα προβλήματα ανήκουν στην κατηγορία NETLIB ενώ τα υπόλοιπα 4 είναι τυχαία προβλήματα που δημιουργήσαμε εμείς. Όλα τα τρεξίματα έγιναν στο περιβάλλον συστοιχίας (linux cluster) του Τμ. Πληροφορικής του ΤΕΙ Αθήνας το οποίο περιλαμβάνει 8 dedicated υπολογιστές (επεξεργαστές XEON με δυνατότητα hyperthreading) διασυνδεδεμένους με εξοπλισμό Myrinet 2GB. Στον παρακάτω πίνακα φαίνονται τα χαρακτηριστικά των προβλημάτων που επιλέξαμε για να εκτελέσουμε το πείραμα.

Πίνακας 8, Προβλήματα τα οποία χρησιμοποιήθηκαν στο πείραμα

| Πρόβλημα | Περιορισμοί | Μεταβλητές | Μη μηδενικά στοιχεία | Πυκνότητα | Πηγή |
|----------|-------------|------------|----------------------|-----------|--------|
| ADLITTLE | 57 | 97 | 465 | 8.4% | netlib |
| AFIRO | 28 | 32 | 88 | 9.8% | netlib |
| BEACONFD | 174 | 262 | 3476 | 7.6% | netlib |
| BLEND | 75 | 83 | 521 | 8.4% | netlib |
| BRANDY | 221 | 249 | 2150 | 3.9% | netlib |
| LOTFI | 154 | 308 | 1086 | 2.3% | netlib |
| SC105 | 106 | 103 | 281 | 2.6% | netlib |
| SC205 | 206 | 203 | 552 | 1.3% | netlib |
| SC50A | 51 | 48 | 131 | 5.4% | netlib |
| SC50B | 51 | 48 | 119 | 4.9% | netlib |



| | | | | | |
|----------|------|------|--------|-------|--------|
| SCAGR7 | 130 | 140 | 553 | 3.0% | netlib |
| SHARE1B | 118 | 225 | 1182 | 4.5% | netlib |
| SHARE2B | 97 | 79 | 730 | 9.5% | netlib |
| STOCFOR1 | 118 | 111 | 474 | 3.6% | netlib |
| AGG | 489 | 163 | 2541 | 3.2% | netlib |
| AGG2 | 517 | 302 | 4515 | 2.9% | netlib |
| AGG3 | 517 | 302 | 4531 | 2.9% | netlib |
| BANDM | 306 | 472 | 2659 | 1.8% | netlib |
| SCFXM3 | 991 | 1371 | 7846 | 0.6% | netlib |
| RANDOM50 | 1000 | 700 | 140000 | 20.0% | random |
| RANDOM51 | 3000 | 2000 | 120000 | 2.0% | random |
| RANDOM53 | 1000 | 500 | 75000 | 15.0% | random |
| RANDOM54 | 500 | 1000 | 75000 | 15.0% | random |

Παρόλο που η εκτέλεση έγινε για 1,2,3,4,5,6,7,8,16 διεργασίες, εξαιτίας της ρύθμισης του MPI περιβάλλοντος, οι επεξεργαστές που πήραν μέρος στην εκτέλεση του αλγορίθμου για τις παραπάνω διεργασίες έχουν ως εξής:

| Διεργασίες | Επεξεργαστές |
|------------|--------------|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 3 |
| 6 | 3 |
| 7 | 4 |
| 8 | 4 |
| 16 | 8 |



4.1.1.1 Script για την εκτέλεση ενός τρεξίματος

Για την αυτοματοποίηση του κάθε τρεξίματος δημιουργήθηκε ένα shell script το οποίο εκτελούσε το τρέξιμο, κάθε πρόβλημα για όλους τους συνδυασμούς των επεξεργαστών.

```
#!/bin/sh
for fn in ./mps_bench/*.mps; do
    echo "$fn ,1 proc"
    mpirun -n 1 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 2 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 3 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 4 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    echo "$fn ,4 proc"
    mpirun -n 5 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 6 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 7 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    mpirun -n 8 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
    echo "$fn ,16 proc"
    mpirun -n 16 ./dist/Debug/GNU-Linux-x86/mpi_simplex_v2 $fn 0 0 0
done
```

Εικόνα 3. Το αρχείο run_tests.sh

Το script αυτό αναλάμβανε να τρέξει όλα τα αρχεία mps που υπάρχουν στον φάκελο mps_bench για 1-8 και 16 επεξεργαστές, με σκοπό την ελαχιστοποίησης της αντικειμενικής συνάρτησης, με επίπεδο εκασφαλμάτωσης 0, χωρίς την παραγωγή μηνυμάτων MPE. Δημιουργήσαμε ακόμα ένα αρχείο run_tests_mpe.sh το οποίο τρέχει τα προβλήματα με παραγωγή MPE μηνυμάτων.

Στο τέλος κάθε εκτέλεσης του mpi_simplex_v2, έτσι ώστε να μην επηρεάζεται ο χρόνος εκτέλεσης, προστίθενται (append) στο αρχείο times.csv πληροφορίες σχετικές με τους χρόνους και τα χαρακτηριστικά αυτής. Στο τέλος λοιπόν της εκτέλεσης του script θα υπάρχουν $21(\text{προβλήματα}) \times 9(\text{τύποι επεξεργαστών}) = 189$ εγγραφές στο αρχείο αυτό.

4.1.1.2 Αρχείο καταγραφής δεδομένων εκτέλεσης times.csv

Κάθε εγγραφή του αρχείου times.csv περιλαμβάνει τα εξής πεδία:

1. Όνομα αρχείου MPS
2. Αριθμός Επεξεργαστών οι οποίοι έλυσαν (1,2,3,4,5,6,7,8,16) παράλληλα το πρόβλημα
3. Αριθμός επαναλήψεων μέχρι την επίλυση της Φάσης I



4. Κωδικός επιστροφής της Φάσης I. Εάν στην Φάση I βρέθηκε μία εφικτή αρχική λύση τότε ο κωδικός αυτός ισούται με 1 αλλιώς με 0
5. Η τελική τιμή της αντικειμενικής συνάρτησης της Φάσης I
6. Αριθμός επαναλήψεων μέχρι την επίλυση της Φάσης II
7. Κωδικός επιστροφής της Φάσης II
8. Η τελική τιμή της αντικειμενικής συνάρτησης της Φάσης II. Δηλαδή η λύση του προβλήματος. Συγκρίνοντας το πεδίο αυτό με την γνωστή τιμή επίλυσης, μπορούμε να ελέγξουμε τα κατά πόσο ο αλγόριθμος επιλύει σωστά το πρόβλημα
9. Ο χρόνος που μεσολάβησε από την στιγμή που ξεκινάει το πρόγραμμα (αμέσως μετά το MPI_Init) μέχρι την στιγμή που ξεκινάει να φορτώνει το MPS αρχείο
10. Ο χρόνος που διήρκεσε το φόρτωμα του MPS αρχείου
11. Ο χρόνος που μεσολάβησε από την στιγμή που τελείωσε το φόρτωμα των δεδομένων μέχρι την στιγμή που ξεκίνησε ο διαμοιρασμός τους
12. Ο χρόνος που διήρκησε ο διαμοιρασμός (distribute) των δεδομένων του προβλήματος
13. Ο χρόνος που μεσολάβησε από την στιγμή που τελείωσε ο διαμοιρασμός των δεδομένων μέχρι την έναρξη επίλυσης της Φάσης I
14. Ο χρόνος που διήρκησε η Φάση I
15. Ο χρόνος που μεσολάβησε από την στιγμή που τελείωσε η Φάση I μέχρι την έναρξη της Φάσης II
16. Ο χρόνος που διήρκησε η Φάση II

Αφού λοιπόν έχει τελειώσει το τρέξιμο των 21 προβλημάτων, κρατάμε ένα αντίγραφο του times.csv και τρέχουμε ξανά το script. Στο τέλος όλων των τρεξιμάτων θα έχουμε 7 αρχεία times.csv, τα οποία και θα επεξεργαστούμε για να υπολογίσουμε τα μετρικά

4.1.1.3 Δεδομένα καταγραφής MPE

Επιπλέον των δεδομένων που κρατήσαμε για τους χρόνους και τα χαρακτηριστικά της εκτέλεσης των προβλημάτων σε διαφορετικό αριθμό επεξεργαστών, εκτελέσαμε και από μία φορά τα προβλήματα σε περιβάλλον 8 επεξεργαστών, αποθηκεύοντας αυτή την φορά δεδομένα, μέσω του περιβάλλοντος MPE, για τους χρόνους εκτέλεσης των διαφορετικών βημάτων του αλγορίθμου (επιλογή εισερχόμενης / εξερχόμενης μεταβλητής, περιστροφή πίνακα).Το περιβάλλον MPE καταγράφει αυτά τα στοιχεία σε αρχεία τύπου CLOG2.



4.1.2 Επεξεργασία αποτελεσμάτων

Αρχικά επιλέξαμε η επεξεργασία των αποτελεσμάτων να γίνει στο Excel. Όμως τα δεδομένα μας κινούνται σε τρείς διαστάσεις. Στην πρώτη καταγράφαμε τον αριθμό του προβλήματος, στην δεύτερη τον αριθμό του πεδίου του times.csv ενώ στην τρίτη διάσταση ήταν ο αριθμός της μέτρησης, αφού είχαμε επτά times.csv. Έτσι γρήγορα διαπιστώσαμε πως το περιβάλλον επεξεργασίας ήταν αρκετά ανελαστικό, τόσο για τον υπολογισμό των μετρικών όσο και για την στατιστική επεξεργασία τους.

Φτιάξαμε λοιπόν απλά script για το φόρτωμα και την επεξεργασία των δεδομένων στο MATLAB.

4.1.2.1 Φόρτωμα των δεδομένων

Η συνάρτηση process_times(filename) λαμβάνει σαν είσοδο το όνομα του αρχείου καταγραφής, π.χ. ‘times1.csv’ και δίνει σαν έξοδο έναν πίνακα τριών διαστάσεων, όπως φαίνεται και στην παρακάτω εικόνα.

```
Editor - \\hall\\f8\\Documents and Settings\\jkr\\My Documents\\Projects\\2008\\PLH40\\Results\\process_times.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function [data] = process_times(filename)
2 %
3 % data(i,j,k):
4 %     i: number of problem
5 %     j: number of procs [1=1, ..., 8-8,9=16]
6 %     k: metriko
7 %         1=load data time
8 %         2=distr data time
9 %         3=P1 time
10 %         4=P2 time
11 %         5=P1 iter,
12 %         6=P2 iter
13 %         7=total time
14 %
15
16 csvdata=dlmread(filename,';',0,1);
17
18 for i=1:1:21
19     for j=1:1:9
20         csv_row = (i-1)*9 + j;
21         data(i,j,1) = csvdata(csv_row,9);
22         data(i,j,2) = csvdata(csv_row,11);
23         data(i,j,3) = csvdata(csv_row,13);
24         data(i,j,4) = csvdata(csv_row,15);
25         data(i,j,5) = csvdata(csv_row,2);
26         data(i,j,6) = csvdata(csv_row,5);
27         for p=10:1:15
28             data(i,j,7) += csvdata(csv_row,p);
29         end
30     end
31 end
32
```

Εικόνα 4, η συνάρτηση process_times



4.1.2.2 Υπολογισμός Μετρικών – επιτάχυνση

Για το μετρικό **Χρόνος εκτέλεσης υπολογισμών δημιουργήσαμε το script calc_speedup:**

The screenshot shows a MATLAB editor window titled "Editor - F:\Documents and Settings\jkr\My Documents\Projects\2008\PLH40\Results\calc_speedup.m*". The code in the editor is as follows:

```
function [speedup] = calc_speedup(data)
%
% speedup(i,j):
%   i: number of problem
%   j: number of procs [1=1, ..., 8-8,9=16]
%
for i=1:1:21
    for j=1:1:9
        speedup(i,j) = (data(i,1,3)+data(i,1,4)) / (data(i,j,3)+data(i,j,4));
    end
end
```

The editor interface includes toolbars, menus like File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, Help, and a Stack dropdown set to Base. Below the editor are tabs for process_times.m, calc_speedup.m, and calc_speedup.m*, with calc_speedup.m* being the active tab. The status bar at the bottom shows Ln 6 Col 1 OVR.

Εικόνα 5, calc_speedup

Για κάθε τρέξιμο εκτελούμε την εντολή $\text{speedup}(:,:,n)=\text{calc_speedup}(\text{dataN})$, όπου n είναι ο αριθμός των τρεξίματος και dataN είναι ο πίνακας που έχουμε υπολογίσει για το n-οστό τρέξιμο με το process_times.

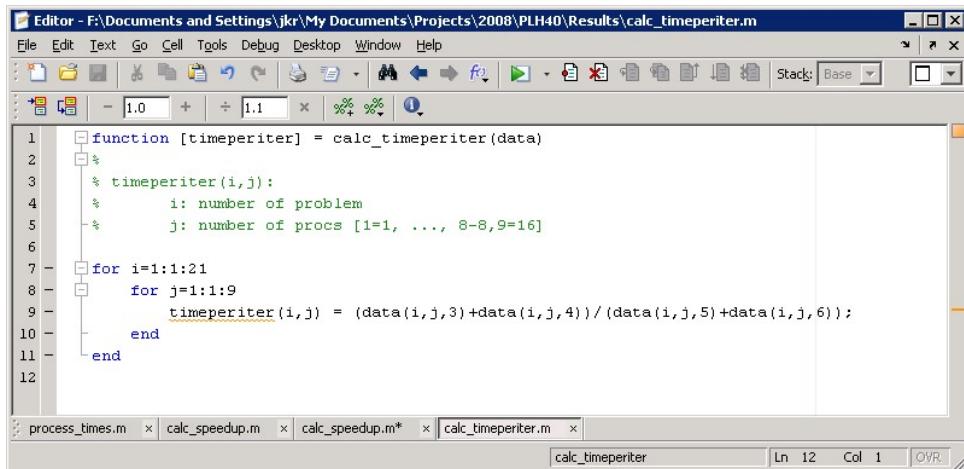
Έτσι στο τέλος, για να υπολογίσουμε τον μέσο όρο και την τυπική απόκλιση του μετρικού αυτού, αρκεί να εκτελέσουμε τις εντολές

```
mean(speedup,3)
```

```
std(speedup,0,3)
```

4.1.2.3 Υπολογισμός Μετρικών – Μέσος χρόνος υπολογισμών ανά επανάληψη

Για το μετρικό **Μέσος χρόνος υπολογισμών ανά επανάληψη** δημιουργήσαμε το script calc_timeperiter.m:

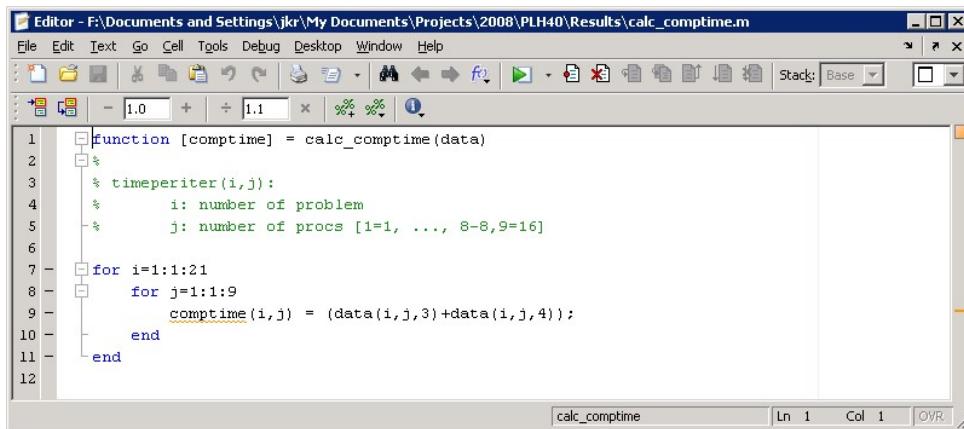


The screenshot shows the MATLAB Editor window with the script `calc_timeperiter.m`. The code calculates the time per iteration for a problem of size $i \times j$ using the formula $\text{timeperiter}(i,j) = (\text{data}(i,j,3)+\text{data}(i,j,4))/(\text{data}(i,j,5)+\text{data}(i,j,6))$. It uses nested loops to iterate over i from 1 to 21 and j from 1 to 9.

```
function [timeperiter] = calc_timeperiter(data)
%
% timeperiter(i,j):
%   i: number of problem
%   j: number of procs [1=1, ..., 8-8,9=16]
%
for i=1:1:21
    for j=1:1:9
        timeperiter(i,j) = (data(i,j,3)+data(i,j,4))/(data(i,j,5)+data(i,j,6));
    end
end
```

4.1.2.4 Υπολογισμός Μετρικών – Χρόνος εκτέλεσης υπολογισμών

Για το μετρικό Χρόνος εκτέλεσης υπολογισμών δημιουργήσαμε το script `calc_comptime`:



The screenshot shows the MATLAB Editor window with the script `calc_comptime.m`. The code is identical to `calc_timeperiter.m`, calculating the time per iteration for a problem of size $i \times j$ using the formula $\text{comptime}(i,j) = (\text{data}(i,j,3)+\text{data}(i,j,4))/(\text{data}(i,j,5)+\text{data}(i,j,6))$. It uses nested loops to iterate over i from 1 to 21 and j from 1 to 9.

```
function [comptime] = calc_comptime(data)
%
% timeperiter(i,j):
%   i: number of problem
%   j: number of procs [1=1, ..., 8-8,9=16]
%
for i=1:1:21
    for j=1:1:9
        comptime(i,j) = (data(i,j,3)+data(i,j,4));
    end
end
```

4.1.2.5 Χρήση του Jumpshot για την επεξεργασία των CLOG αρχείων

Τα αρχεία CLOG που παράγονται με την ενεργοποίηση του MPE διαβάζονται από το λογισμικό Jumpshot, το οποίο παράγει γραφικές παραστάσεις με τους χρόνους μεταξύ των σημείων που έχουμε ορίσει στον κώδικα μας. Το Jumpshot είναι γραμμένο σε JAVA και διανέμεται μαζί με το MPICH για windows η για Linux.



4.2 Παρονσίαση αποτελεσμάτων

4.2.1 Μέσος Όρος Επιτάχυνσης ανά Αριθμό Διεργασιών που συμμετείχαν στην λύση του προβλήματος

4.2.1.1 Πίνακας

| Μέγεθος | Πρόβλημα | Αριθμός διεργασιών (επεξεργαστών) | | | | | | | | |
|----------------|----------|-----------------------------------|-------|------|------|------|------|------|------|-------|
| | | 1 (1) | 2 (1) | 3(2) | 4(2) | 5(3) | 6(3) | 7(4) | 8(4) | 16(8) |
| 57x97(8.41%) | ADLITTLE | 1.00 | 0.45 | 0.39 | 0.31 | 0.28 | 0.28 | 0.25 | 0.23 | 0.17 |
| 28x32(9.82%) | AFIRO | 1.00 | 0.11 | 0.05 | 0.04 | 0.03 | 0.03 | 0.03 | 0.03 | 0.02 |
| 174x262(7.62%) | BEACONFD | 1.00 | 0.62 | 0.82 | 0.93 | 1.01 | 1.08 | 1.08 | 1.01 | 0.90 |
| 75x83(8.37%) | BLEND | 1.00 | 0.44 | 0.38 | 0.30 | 0.26 | 0.27 | 0.24 | 0.21 | 0.16 |
| 221x249(3.91%) | BRANDY | 1.00 | 0.61 | 0.84 | 0.93 | 1.01 | 1.09 | 1.11 | 1.06 | 0.99 |
| 154x308(2.29%) | LOTFI | 1.00 | 0.66 | 0.90 | 0.99 | 1.08 | 1.20 | 1.20 | 1.15 | 1.05 |
| 106x103(2.57%) | SC105 | 1.00 | 0.44 | 0.47 | 0.38 | 0.36 | 0.36 | 0.33 | 0.29 | 0.22 |
| 206x203(1.32%) | SC205 | 1.00 | 0.71 | 0.93 | 1.01 | 1.08 | 1.14 | 1.12 | 1.04 | 0.93 |
| 51x48(5.35%) | SC50A | 1.00 | 0.21 | 0.13 | 0.10 | 0.08 | 0.08 | 0.07 | 0.07 | 0.05 |
| 51x48(4.86%) | SC50B | 1.00 | 0.20 | 0.12 | 0.10 | 0.08 | 0.08 | 0.07 | 0.06 | 0.05 |
| 130x140(3.04%) | SCAGR7 | 1.00 | 0.39 | 0.50 | 0.45 | 0.44 | 0.44 | 0.41 | 0.37 | 0.29 |
| 118x225(4.45%) | SHARE1B | 1.00 | 0.53 | 0.71 | 0.70 | 0.72 | 0.76 | 0.74 | 0.66 | 0.54 |
| 97x79(9.53%) | SHARE2B | 1.00 | 0.30 | 0.29 | 0.23 | 0.21 | 0.21 | 0.19 | 0.16 | 0.13 |
| 118x111(3.62%) | STOCFOR1 | 1.00 | 0.38 | 0.43 | 0.38 | 0.36 | 0.36 | 0.33 | 0.30 | 0.23 |



| | | Αριθμός διεργασιών (επεξεργαστών) | | | | | | | | |
|-----------------|----------|-----------------------------------|-------|------|------|------|------|------|------|-------|
| Μέγεθος | Πρόβλημα | 1 (1) | 2 (1) | 3(2) | 4(2) | 5(3) | 6(3) | 7(4) | 8(4) | 16(8) |
| 489x163(3.19%) | AGG | 1.00 | 0.85 | 1.18 | 1.38 | 1.58 | 1.78 | 1.86 | 1.78 | 1.84 |
| 517x302(2.89%) | AGG2 | 1.00 | 0.88 | 1.29 | 1.56 | 1.81 | 2.09 | 2.21 | 2.28 | 2.83 |
| 517x302(2.9%) | AGG3 | 1.00 | 0.88 | 1.29 | 1.54 | 1.81 | 2.08 | 2.19 | 2.28 | 2.84 |
| 306x472(1.84%) | BANDM | 1.00 | 0.69 | 1.03 | 1.21 | 1.43 | 1.62 | 1.70 | 1.79 | 2.16 |
| 991x1371(0.58%) | SCFXM3 | 1.00 | 1.12 | 1.65 | 2.10 | 2.65 | 3.14 | 3.65 | 4.09 | 7.50 |
| 1000x700(20%) | RANDOM50 | 1.00 | 1.02 | 1.51 | 1.99 | 2.47 | 2.94 | 3.41 | 3.87 | 6.65 |
| 3000x2000(2%) | RANDOM51 | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a | n/a |
| 1000x500(15%) | RANDOM53 | 1.00 | 1.00 | 1.48 | 1.99 | 2.47 | 2.96 | 3.43 | 3.79 | 6.32 |
| 500x1000(15%) | RANDOM54 | 1.00 | 1.01 | 1.49 | 1.99 | 2.49 | 2.94 | 3.38 | 3.76 | 6.06 |

4.2.1.2 Σχολιασμός Πίνακα – απονοσία μέτρησης επιτάχυνσης για RANDOM51

Για το πρόβλημα RANDOM51 εκτιμήσαμε τον χρόνο επίλυσης μόνο για 8 και 16 διεργασίες. Ο χρόνος επίλυσης με 8 διεργασίες ήταν περίπου 6083 δευτερόλεπτα ενώ για 16 διεργασίες ήταν περίπου 3160 δευτερόλεπτα (δηλαδή λίγο λιγότερο από 1 ώρα – βλ. παρακάτω πίνακα παραγράφου 4.2.2). Κρίθηκε πως δεν ήταν πρακτικό να κάνουμε μετρήσεις για το συγκεκριμένο πρόβλημα για λιγότερες διεργασίες. Δεν είμαστε όμως σε θέση να εκτιμήσουμε την επιτάχυνση.

Ωστόσο είναι εμφανές ότι για το συγκεκριμένο πρόβλημα (RANDOM51) το αναμενόμενο speed-up θα μπορούσε να είναι ακόμα μεγαλύτερο από το 7,5 (το οποίο παρουσιάζεται ως το μέγιστο επιτευχθέν και είναι σε κάθε περίπτωση ιδιαίτερα ικανοποιητικό – βλ. πρόβλημα SCFXM3 16 διεργασίες σε 8 πραγματικούς επεξεργαστές) καθώς ο σχετικός λόγος χρόνων απόκρισης από τις 16 στις 8 διεργασίες (1,92) είναι ο μεγαλύτερος μεταξύ όλων των προβλημάτων (βλ. αναλυτικότερα παρακάτω πίνακα παραγράφου 4.2.2) για τις ίδιες διεργασίες (για το πρόβλημα SCFXM3 π.χ. είναι 1,83 ενώ για το RANDOM50 είναι 1,72).



4.2.1.3 Σχολιασμός Πίνακα – Κατάταξη προβλημάτων σε σχέση με την επιτάχυνση

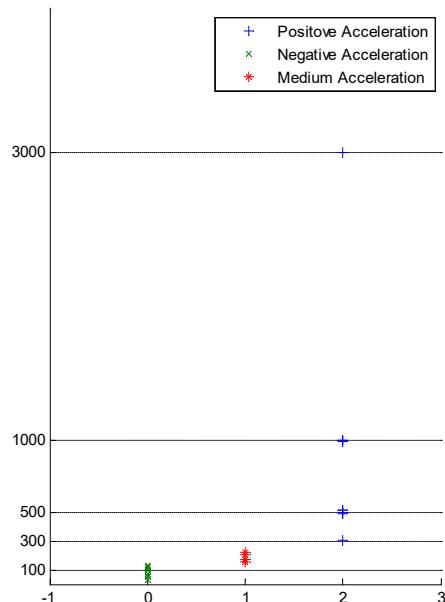
Με κριτήριο την επιτάχυνση στον παραπάνω πίνακα, θα μπορούσαμε να κατατάξουμε τα προβλήματα σε τρείς κατηγορίες:

- Αυτά που συμπεριφέρονται αρνητικά στην παράλληλη επίλυση. Τα προβλήματα αυτά θα τα ορίσουμε ως αυτά που δεν παρουσιάζουν επιτάχυνση, ανεξαρτήτως του αριθμού των διεργασιών. Τα προβλήματα αυτά είναι τα : ADLIITLE, AFIRO, BLEND, SC50A, SC50B, SC105, STOCFOR1, SHARE1B, SHARE2B κ.α.
- Αυτά που συμπεριφέρονται μέτρια στην παράλληλη επίλυση. Κριτήριο για να ενταχθεί ένα πρόβλημα σε αυτή την κατηγορία θα είναι να παρουσιάζει και επιβράδυνση και επιτάχυνση (για κάποιο εύρος επεξεργαστών στην κάθε περίπτωση). Τα προβλήματα αυτά είναι τα : BEACONFD, BRANDY, SC205, LOTFI.
- Αυτά που συμπεριφέρονται θετικά και απόλυτα θετικά στην παράλληλη επίλυση. Εδώ εντάσσονται τα προβλήματα που παρουσιάζουν σε όλες τις περιπτώσεις ικανοποιητική ή πολύ ικανοποιητική επιτάχυνση. Τα προβλήματα αυτά είναι τα : AGG, AGG2, AGG3, BANDM, SCFXM3, RANDOM50, RANDOM53, RANDOM54.

Με βάση λοιπόν αυτές τις παραπάνω κατηγορίες, μπορούμε να δούμε κάποια περιγραφικά στατιστικά για τα περιγραφικά τους χαρακτηριστικά. Τα παρουσιάζουμε στον παρακάτω πίνακα:

| Μέσος Όρος ± Τυπική απόκλιση | Γραμμές | Στήλες | Μη μηδενικά στοιχεία |
|------------------------------|-----------------|-----------------|----------------------|
| Αρνητική Απόκριση | 83,10 ± 35,27 | 96,6 ± 55,87 | 454,4 |
| Μέτρια Απόκριση | 188,75 ± 30,34 | 255,5 ± 43,19 | 1816 |
| Θετική Απόκριση | 924,44 ± 822,19 | 756,66 ± 601,92 | 48,010 |

Για να γίνει καλύτερα κατανοητός ο τρόπος που επηρεάζουν τα παραπάνω χαρακτηριστικά την απόκριση στην παράλληλη εκτέλεση, παρουσιάζουμε στο επόμενο γράφημα την κατανομή των γραμμών των προβλημάτων της κάθε κατηγορίας:

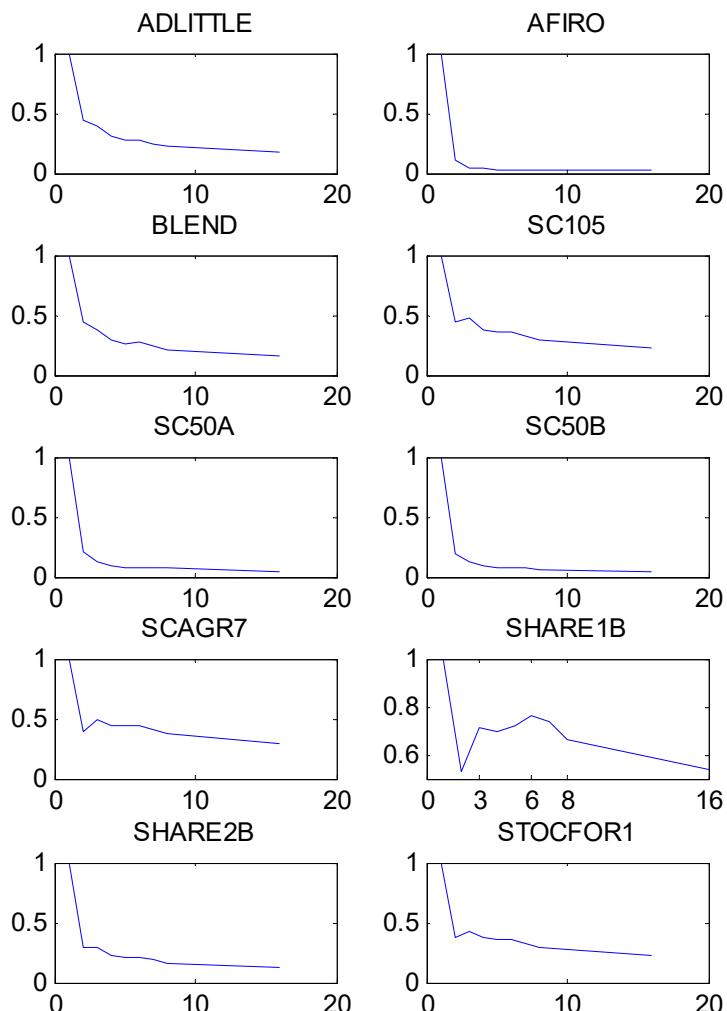


Χωρίς να κάνουμε τεστ στατιστικής σημαντικότητας, με μία πρόχειρη ματιά, μπορούμε να βγάλουμε το συμπέρασμα ότι το μέγεθος του προβλήματος παίζει ρόλο στην επιτάχυνση.

4.2.1.4 Σχολιασμός Πίνακα – Επιτάχυνση σε προβλήματα αρνητικής απόκρισης

Στο παρακάτω γράφημα φαίνεται η επιτάχυνση για τα προβλήματα αυτής της κατηγορίας:

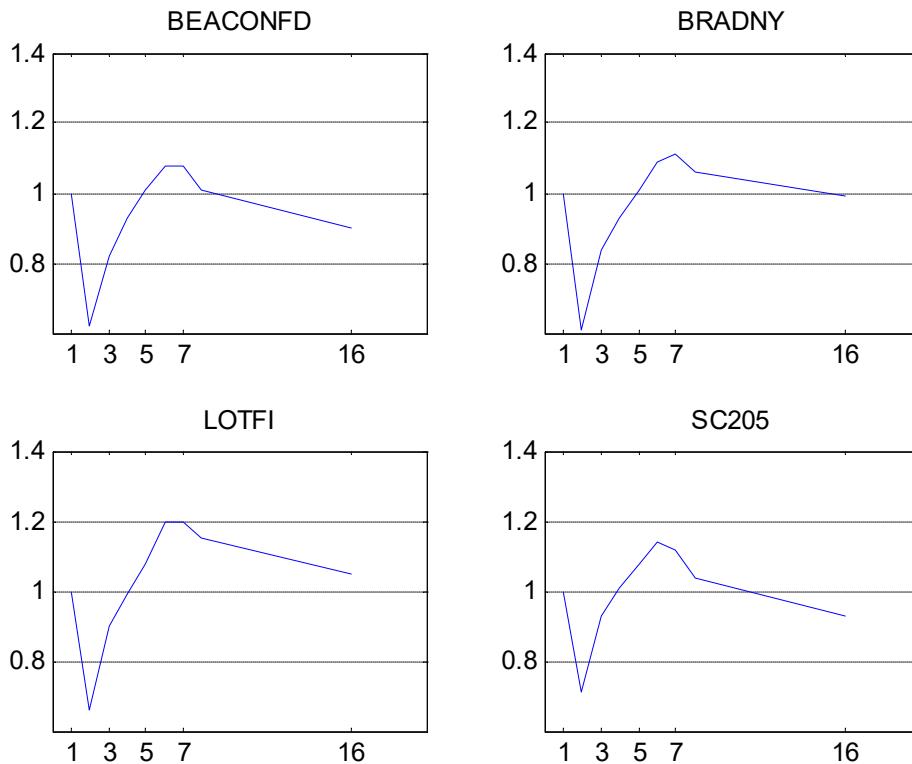
[είναι συνολικά δέκα προβλήματα: ADLIITLE, AFIRO, BLEND, SC50A, SC50B, SC105, STOCFOR1, SHARE1B, SHARE2B, SCAGR7]



Παρατηρούμε ότι σε όλα αυτά τα προβλήματα η παράλληλη επεξεργασία δεν προκαλεί επιτάχυνση αλλά αντίθετα επιβράδυνση. Το μικρό μέγεθος των προβλημάτων αυτών και ο μικρός χρόνος επίλυσης σε έναν επεξεργαστή έχουν σαν αποτέλεσμα να μειώνεται ο λόγος (χρόνος υπολογισμού) / (χρόνος επικοινωνίας) όσο αυξάνει ο αριθμός των διεργασιών και εν τέλει να υπάρχει αυτή η αρνητική αντίδραση στην παράλληλη επεξεργασία.

4.2.1.5 Σχολιασμός Πίνακα – Επιτάχυνση σε προβλήματα μέσης απόκρισης

Στο παρακάτω γράφημα φαίνονται οι λεπτομέρειες για τα προβλήματα με μέση απόκριση:

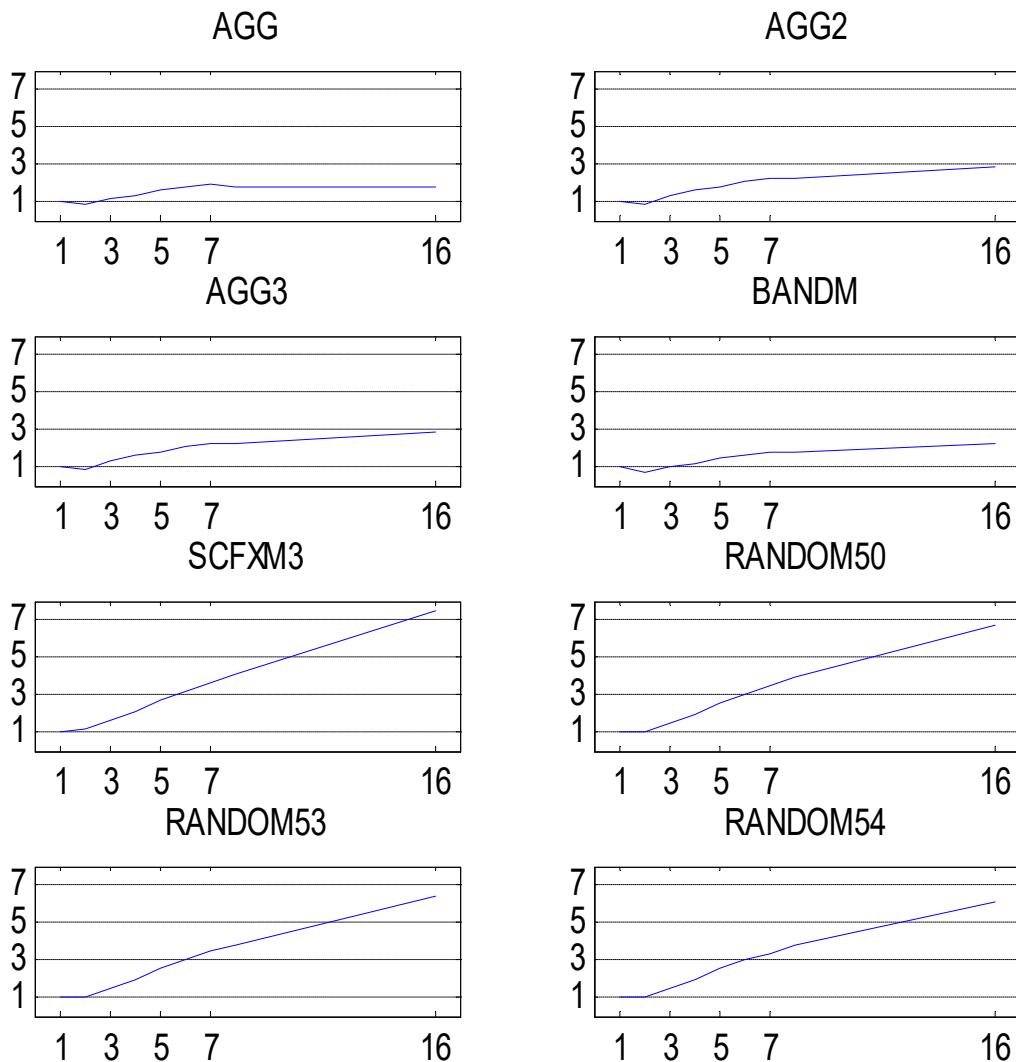


Όπως παρατηρούμε, στα προβλήματα αυτά, η επιτάχυνση συμβαίνει για κάποιο εύρος αριθμού επεξεργαστών και δεν είναι ικανοποιητική – υπάρχει δηλαδή επιτάχυνση η οποία υπερβαίνει το ένα σε αρκετές περιπτώσεις, με καλύτερο αυτό του προβλήματος LOTFI λόγω του συγκριτικά λίγο μεγαλύτερου μεγέθους από τα υπόλοιπα.

4.2.1.6 Σχολιασμός Πίνακα – Επιτάχυνση σε προβλήματα θετικής απόκρισης

Στο παρακάτω διάγραμμα εμφανίζεται η επιτάχυνση για τα προβλήματα που εμφανίζουν θετική (AGG, AGG2, AGG3, BANDM) και απόλυτα θετική απόκριση (SCFXM3 και RANDOM50, RANDOM53, RANDOM54) στην παράλληλη επεξεργασία:

[έχει παραληφθεί το πρόβλημα RANDOM51 – για το οποίο ισχύουν τα όσα αναφέρονται στην παράγραφο 4.2.1.2 παραπάνω]



Όπως παρατηρούμε, τον ουσιαστικό ρόλο στην επιτάχυνση παίζει ο αριθμός των επεξεργαστών και όχι ο αριθμός των διεργασιών (π.χ. δύο ανά επεξεργαστή με hyperthreading). Βλέπουμε για παράδειγμα πως στην επίλυση των RANDOM προβλημάτων με 16 διεργασίες η επιτάχυνση φτάνει σχεδόν ίση με τον αριθμό των επεξεργαστών (8).

[πιο συγκεκριμένα φτάνει μέχρι και 7,5 για 8 επεξεργαστές με δύο διεργασίες ανά επεξεργαστή / hyperthreading), επιτάχυνση η οποία μπορεί να θεωρηθεί πολύ ικανοποιητική και σε κάθε περίπτωση ανάλογης τουλάχιστον έκτασης των αναγνωρισμένων δημοσιευμένων εργασιών της βιβλιογραφίας για μεγάλου μεγέθους προβλήματα]



4.2.2 Χρόνος Εκτέλεσης Υπολογισμών

4.2.2.1 Πίνακας

| Μέγεθος | Πρόβλημα | Αριθμός διεργασιών | | | | | | | | |
|----------------|----------|--------------------|------|------|------|------|------|------|------|------|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 |
| 57x97(8.41%) | ADLITTLE | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 | 0.06 | 0.06 | 0.07 | 0.09 |
| 28x32(9.82%) | AFIRO | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 174x262(7.62%) | BEACONFD | 0.13 | 0.21 | 0.16 | 0.14 | 0.13 | 0.12 | 0.12 | 0.13 | 0.15 |
| 75x83(8.37%) | BLEND | 0.01 | 0.03 | 0.04 | 0.04 | 0.05 | 0.05 | 0.06 | 0.06 | 0.08 |
| 221x249(3.91%) | BRANDY | 0.53 | 0.87 | 0.63 | 0.57 | 0.52 | 0.49 | 0.48 | 0.50 | 0.54 |
| 154x308(2.29%) | LOTFI | 0.15 | 0.23 | 0.17 | 0.15 | 0.14 | 0.13 | 0.13 | 0.13 | 0.14 |
| 106x103(2.57%) | SC105 | 0.02 | 0.04 | 0.04 | 0.05 | 0.05 | 0.05 | 0.05 | 0.06 | 0.08 |
| 206x203(1.32%) | SC205 | 0.17 | 0.25 | 0.19 | 0.17 | 0.16 | 0.15 | 0.16 | 0.17 | 0.19 |
| 51x48(5.35%) | SC50A | 0.00 | 0.01 | 0.01 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.03 |
| 51x48(4.86%) | SC50B | 0.00 | 0.01 | 0.01 | 0.02 | 0.02 | 0.02 | 0.03 | 0.03 | 0.04 |
| 130x140(3.04%) | SCAGR7 | 0.04 | 0.11 | 0.08 | 0.09 | 0.10 | 0.09 | 0.10 | 0.11 | 0.14 |
| 118x225(4.45%) | SHARE1B | 0.21 | 0.40 | 0.30 | 0.30 | 0.29 | 0.28 | 0.29 | 0.32 | 0.39 |
| 97x79(9.53%) | SHARE2B | 0.01 | 0.04 | 0.04 | 0.05 | 0.06 | 0.06 | 0.06 | 0.07 | 0.10 |
| 118x111(3.62%) | STOCFOR1 | 0.02 | 0.05 | 0.04 | 0.05 | 0.05 | 0.05 | 0.06 | 0.06 | 0.08 |
| 489x163(3.19%) | AGG | 0.16 | 0.19 | 0.14 | 0.12 | 0.10 | 0.09 | 0.09 | 0.09 | 0.09 |
| 517x302(2.89%) | AGG2 | 0.57 | 0.65 | 0.44 | 0.37 | 0.31 | 0.27 | 0.26 | 0.25 | 0.20 |
| 517x302(2.9%) | AGG3 | 0.67 | 0.76 | 0.52 | 0.44 | 0.37 | 0.32 | 0.31 | 0.30 | 0.24 |



| Μέγεθος | Πρόβλημα | Αριθμός διεργασιών | | | | | | | | | |
|-----------------|----------|--------------------|--------|--------|--------|-------|-------|-------|---------|---------|--|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | |
| 306x472(1.84%) | BANDM | 5.91 | 8.53 | 5.75 | 4.87 | 4.12 | 3.65 | 3.47 | 3.30 | 2.74 | |
| 991x1371(0.58%) | SCFXM3 | 80.00 | 71.65 | 48.50 | 38.09 | 30.23 | 25.51 | 21.91 | 19.56 | 10.66 | |
| 1000x700(20%) | RANDOM50 | 15.93 | 15.69 | 10.53 | 8.02 | 6.46 | 5.41 | 4.68 | 4.11 | 2.40 | |
| 3000x2000(2%) | RANDOM51 | - | - | - | - | - | - | - | 6083.01 | 3160.42 | |
| 1000x500(15%) | RANDOM53 | 8.84 | 8.80 | 5.91 | 4.44 | 3.55 | 3.01 | 2.61 | 2.35 | 1.45 | |
| 500x1000(15%) | RANDOM54 | 233.73 | 233.98 | 157.57 | 117.36 | 94.73 | 78.90 | 68.21 | 61.60 | 37.01 | |

4.2.2.2 Σχολιασμός Πίνακα

Σαν χρόνο υπολογισμού θεωρούμε την διάρκεια επίλυσης των Φάσεων I και II. Δεν περιλαμβάνεται το φόρτωμα και η διανομή των πινάκων και διανυσμάτων των προβλημάτων.

Παρατηρούμε πόσο δραματικά μειώνεται ο χρόνος υπολογισμού για προβλήματα που διαρκούν πάνω από 5 δευτερόλεπτα. Π.χ. για πρόβλημα το οποίο απαιτεί 80sec σε 1 επεξεργαστή (SCFXM3) ο χρόνος απόκρισης εκμεταλλευόμενοι πλήρως το σύστημά μας πέφτει κοντά στα 10sec.

Πολύ περισσότερο αν αναλογιστούμε προβλήματα αρκετά μεγαλύτερα (π.χ. RANDOM 51) όπου ο χρόνος σε 1 επεξεργαστή είναι απαγορευτικός (κατ' εκτίμηση περίπου κοντά στις 7 ώρες) στο σύστημά μας πέφτει με παράλληλη εκτέλεση σε χρόνο κάτω από 1 ώρα. Γενικά, καθίσταται εφικτή η επίλυση μεγάλων προβλημάτων σε αρκετά ικανοποιητικό πραγματικό χρόνο.

Είναι επίσης εμφανές ότι για την πληρέστερη κατανόηση και αξιοποίηση των δυνατοτήτων του συστήματος (cluster) θα βοηθούσε η εκτέλεση ακόμα μεγαλύτερων σε μέγεθος προβλημάτων καθώς επίσης και με μεταβαλλόμενη πυκνότητα.



4.2.3 Μέσος Χρόνος Υπολογισμών ανά επανάληψη

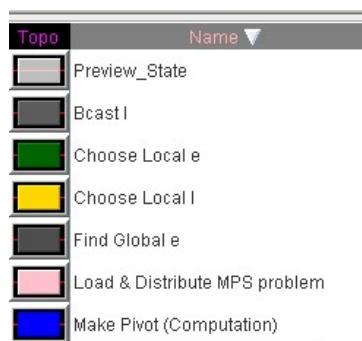
Παρακάτω επίσης δίνονται για πληρέστερη κατανόηση από τον αναγνώστη και οι μέσοι χρόνοι υπολογισμών για κάθε επανάληψη (για όλους τους διαφορετικούς αριθμούς διεργασιών που δοκιμάστηκαν στα πειράματα), καθώς είναι χρήσιμο να τονιστεί και εδώ ότι (α) η πηγή της επιτάχυνσης – ή επιβράδυνσης – είναι σε κάθε περίπτωση οι επιμέρους υπολογισμοί κάθε επανάληψης καθώς και (β) το ότι ο τελικός χρόνος απόκρισης διαμορφώνεται πάντα από τον συνολικό αριθμό επαναλήψεων του κάθε προβλήματος.

| Μέγεθος | Πρόβλημα | Αριθμός διεργασιών | | | | | | | | 16 |
|----------------|----------|--------------------|---------|---------|---------|---------|---------|---------|---------|---------|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 57x97(8.41%) | ADLITTLE | 0.00011 | 0.00024 | 0.00028 | 0.00035 | 0.00039 | 0.00039 | 0.00044 | 0.00049 | 0.00065 |
| 28x32(9.82%) | AFIRO | 0.00001 | 0.00014 | 0.00029 | 0.00038 | 0.00045 | 0.00045 | 0.00051 | 0.00057 | 0.00076 |
| 174x262(7.62%) | BEACONFD | 0.00071 | 0.00114 | 0.00086 | 0.00077 | 0.00070 | 0.00066 | 0.00065 | 0.00070 | 0.00079 |
| 75x83(8.37%) | BLEND | 0.00011 | 0.00024 | 0.00028 | 0.00036 | 0.00040 | 0.00040 | 0.00044 | 0.00051 | 0.00067 |
| 221x249(3.91%) | BRANDY | 0.00080 | 0.00131 | 0.00095 | 0.00086 | 0.00079 | 0.00073 | 0.00072 | 0.00075 | 0.00081 |
| 154x308(2.29%) | LOTFI | 0.00084 | 0.00126 | 0.00093 | 0.00084 | 0.00077 | 0.00069 | 0.00070 | 0.00073 | 0.00080 |
| 106x103(2.57%) | SC105 | 0.00016 | 0.00035 | 0.00033 | 0.00041 | 0.00044 | 0.00044 | 0.00047 | 0.00054 | 0.00069 |
| 206x203(1.32%) | SC205 | 0.00074 | 0.00105 | 0.00080 | 0.00073 | 0.00069 | 0.00065 | 0.00066 | 0.00071 | 0.00079 |
| 51x48(5.35%) | SC50A | 0.00003 | 0.00016 | 0.00026 | 0.00034 | 0.00040 | 0.00040 | 0.00045 | 0.00051 | 0.00069 |
| 51x48(4.86%) | SC50B | 0.00003 | 0.00016 | 0.00026 | 0.00034 | 0.00040 | 0.00040 | 0.00045 | 0.00050 | 0.00068 |
| 130x140(3.04%) | SCAGR7 | 0.00021 | 0.00053 | 0.00041 | 0.00046 | 0.00047 | 0.00046 | 0.00050 | 0.00056 | 0.00071 |
| 118x225(4.45%) | SHARE1B | 0.00039 | 0.00073 | 0.00055 | 0.00055 | 0.00054 | 0.00051 | 0.00052 | 0.00059 | 0.00072 |
| 97x79(9.53%) | SHARE2B | 0.00009 | 0.00029 | 0.00029 | 0.00038 | 0.00042 | 0.00042 | 0.00046 | 0.00053 | 0.00068 |
| 118x111(3.62%) | STOCFOR1 | 0.00016 | 0.00042 | 0.00037 | 0.00042 | 0.00045 | 0.00045 | 0.00049 | 0.00054 | 0.00071 |



| Μέγεθος | Πρόβλημα | Αριθμός διεργασιών | | | | | | | | 16 |
|-----------------|----------|--------------------|---------|---------|---------|---------|---------|---------|---------|---------|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 489x163(3.19%) | AGG | 0.00193 | 0.00226 | 0.00163 | 0.00140 | 0.00122 | 0.00108 | 0.00103 | 0.00108 | 0.00105 |
| 517x302(2.89%) | AGG2 | 0.00342 | 0.00389 | 0.00266 | 0.00220 | 0.00189 | 0.00164 | 0.00155 | 0.00150 | 0.00121 |
| 517x302(2.9%) | AGG3 | 0.00344 | 0.00389 | 0.00267 | 0.00223 | 0.00190 | 0.00165 | 0.00157 | 0.00151 | 0.00121 |
| 306x472(1.84%) | BANDM | 0.00221 | 0.00320 | 0.00215 | 0.00182 | 0.00154 | 0.00137 | 0.00130 | 0.00124 | 0.00103 |
| 991x1371(0.58%) | SCFXM3 | 0.03450 | 0.03090 | 0.02091 | 0.01643 | 0.01304 | 0.01100 | 0.00945 | 0.00843 | 0.00460 |
| 1000x700(20%) | RANDOM50 | 0.02493 | 0.02456 | 0.01648 | 0.01256 | 0.01011 | 0.00847 | 0.00732 | 0.00644 | 0.00375 |
| 3000x2000(2%) | RANDOM51 | | | | | | | | | |
| 1000x500(15%) | RANDOM53 | | | | | | | | | |
| 500x1000(15%) | RANDOM54 | | | | | | | | | |

4.2.4 Χρόνοι εκτέλεσης των βημάτων του αλγορίθμου Simplex



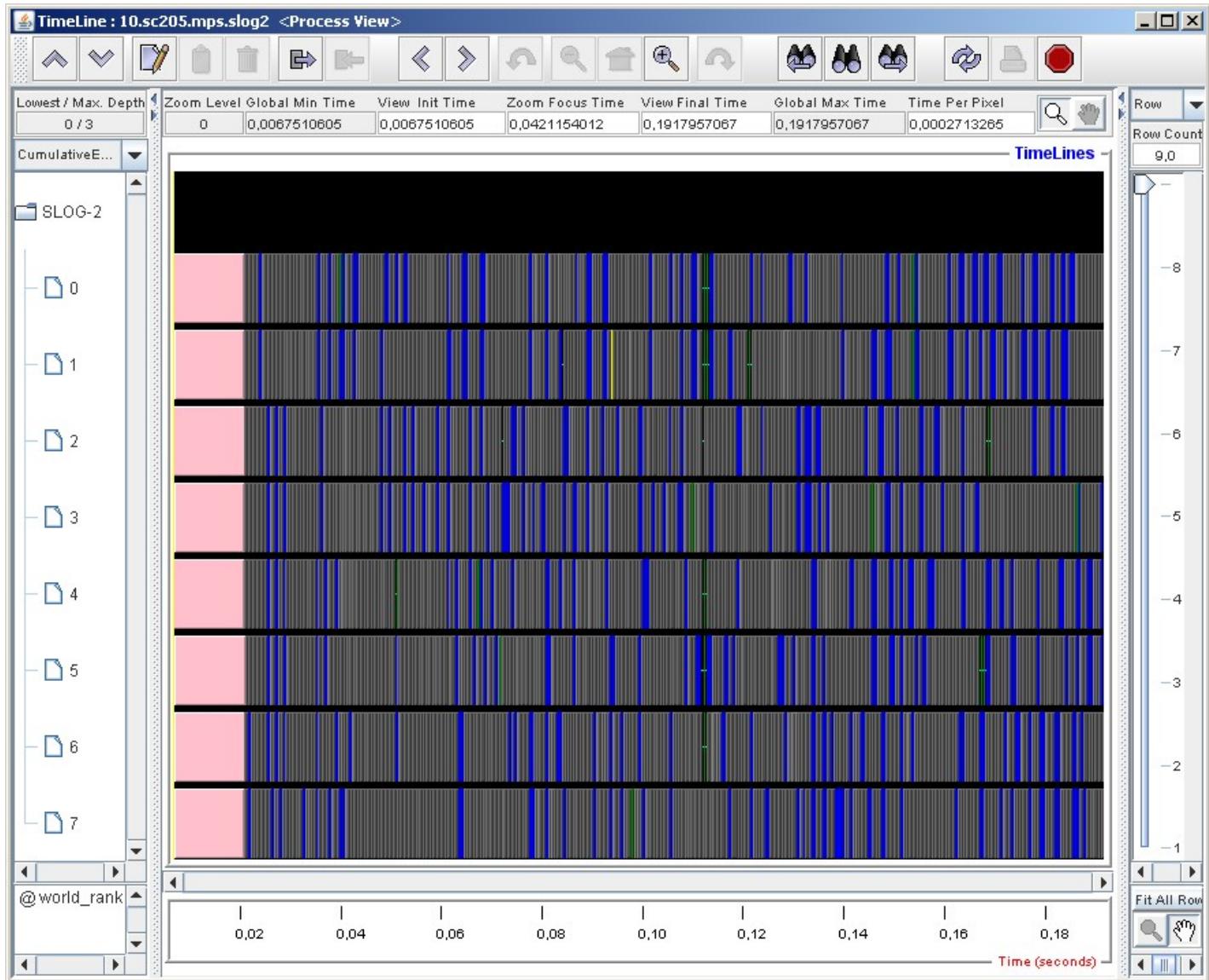
Θα παρουσιάσουμε ενδεικτικά, 1 προβλήματα στα οποία η παράλληλη επεξεργασία προκάλεσε επιβράδυνση και 1 στα οποία προκάλεσε επιτάχυνση. Δίπλα εμφανίζεται το υπόμνημα των χρωματικών διαβαθμίσεων:

Με πράσινο, κίτρινο και μπλέ χρώμα απεικονίζονται βήματα υπολογισμού, ενώ με γκρι η απαραίτητη επικοινωνία.

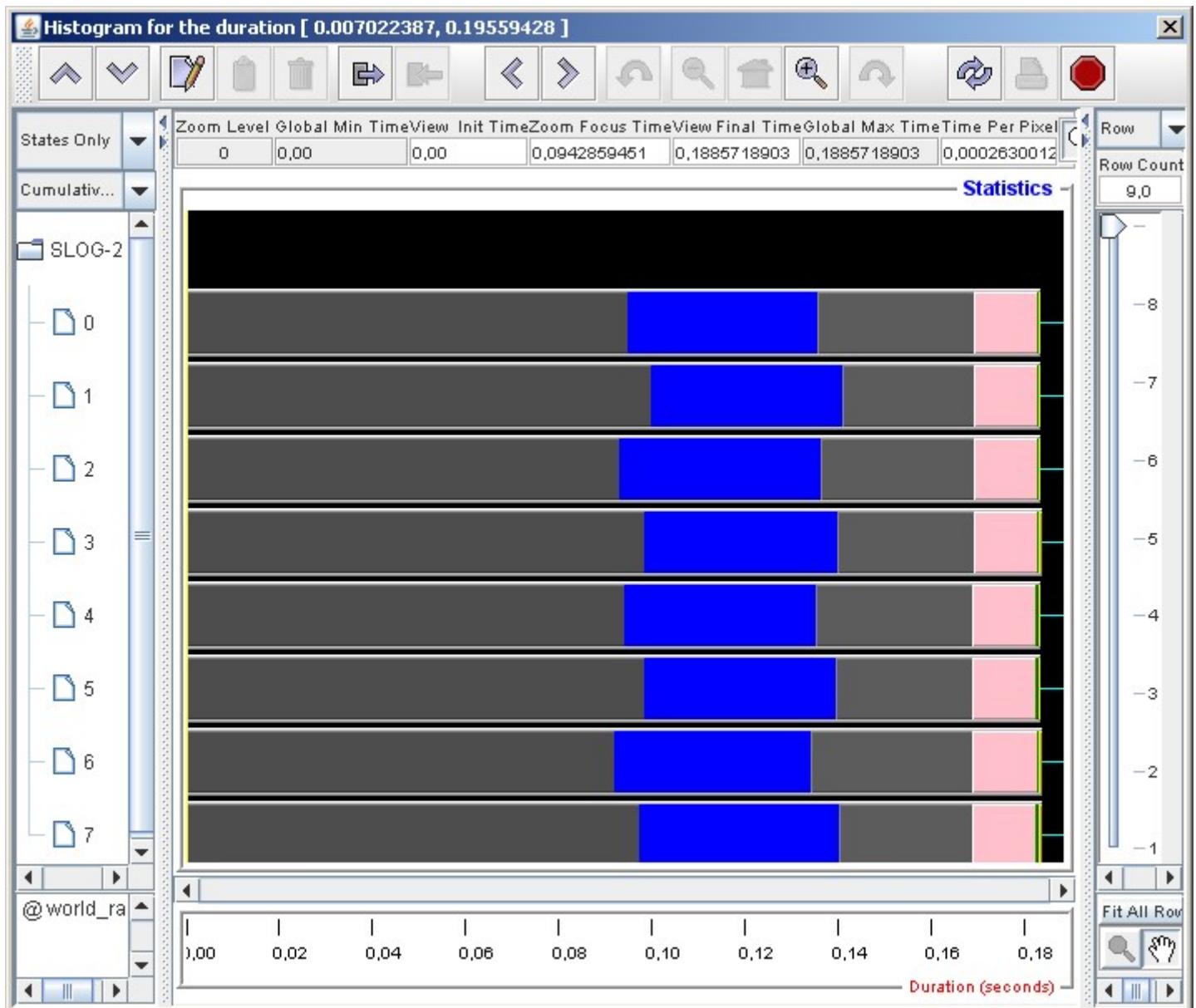
Σε κάθε πρόβλημα εμφανίζουμε και τον αθροιστικό χρόνο που εκτελέστηκε η κάθε φάση του αλγορίθμου. Όλες οι μετρήσεις και παραστάσεις που ακολουθούν διαμορφώθηκαν με χρήση της MPE βιβλιοθήκης γραφικών του MPI και του εργαλείου γραφικής απεικόνισης ‘Jumpshot’ το οποίο επίσης παρέχεται από το MPI για την απεικόνιση των αποτελεσμάτων.



4.2.4.1 Επιβράδυνση - Πρόβλημα SC205 -

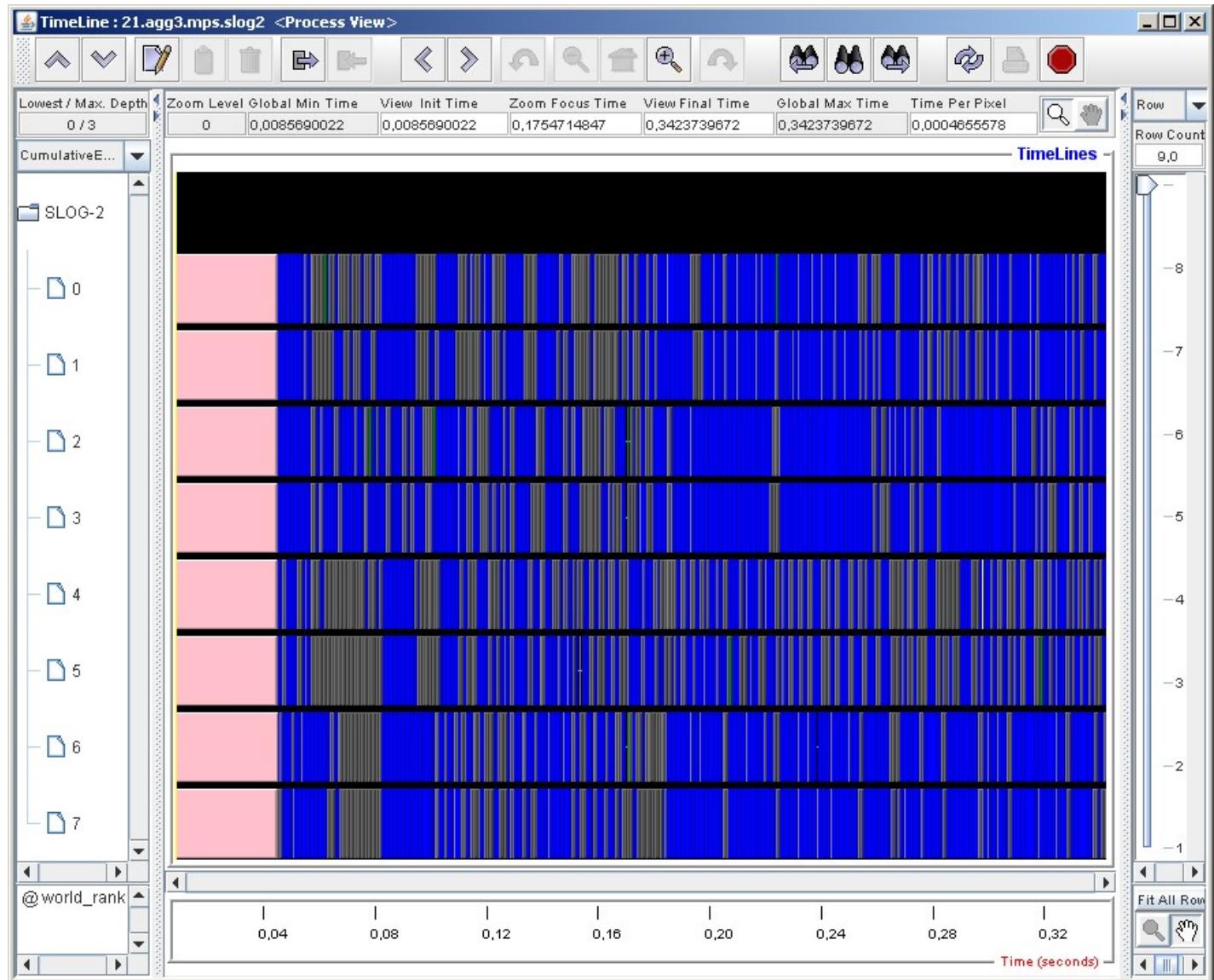


Παρατηρούμε πόσο μικρός είναι ο χρόνος υπολογισμού σε σχέση με τις διαδικασίες επικοινωνίας. Για αυτό και στο πρόβλημα από δεν υπάρχει κάποιου είδους επιτάχυνση στην παράλληλη επεξεργασία. Αυτό φαίνεται ξεκάθαρα και στο συγκεντρωτικό διάγραμμα των χρόνων αυτών, παρακάτω:



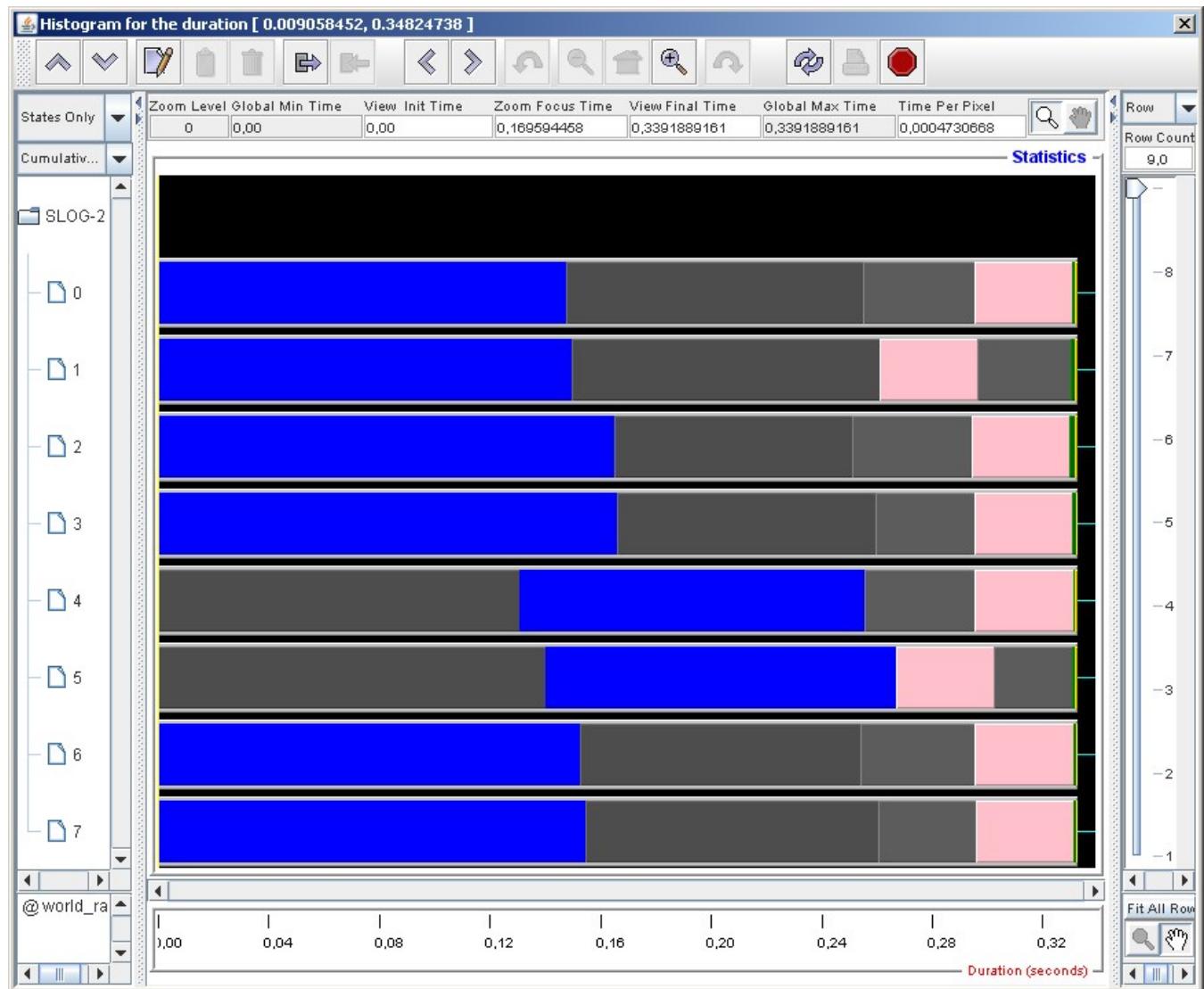


4.2.4.2 Επιτάχυνση – Πρόβλημα AGG3



Εδώ ο υπολογισμός στις επιμέρους διεργασίες επικρατεί της επικοινωνίας. Αυτό οφείλεται στο μέγεθος του προβλήματος που είναι τέτοιο ώστε οι επιμέρους διεργασίες να εκτελούν την περιστροφή του πίνακα σε χρόνο σημαντικό αναλογικά με τον χρόνο που σπαταλούν για την επικοινωνία.

Παρακάτω φαίνεται ο συνολικός χρόνος που οι διεργασίες εκτελούν τα επιμέρους βήματα του αλγορίθμου.





4.3 Συμπεράσματα

Από την παραπάνω ανάλυση των αποτελεσμάτων θα μπορούσαμε να συνοψίσουμε στα εξής συμπεράσματα:

- Το μέγεθος του προβλήματος παίζει ουσιώδη ρόλο στην απόκριση του στην παράλληλη επεξεργασία και στην επιτάχυνση που αυτή προκαλεί ή δεν προκαλεί συναρτήσει του αριθμού των επεξεργαστών. Στα προβλήματα με τα οποία εμείς ασχοληθήκαμε, θετική απόκριση είχαν προβλήματα που η μία τους διάσταση (γραμμές ή στήλες) ήταν τουλάχιστον 400.
- Η κλιμάκωση της επιτάχυνσης αρχίζοντας από προβλήματα με μεσαία μεγέθη και πηγαίνοντας προς προβλήματα μεγάλου μεγέθους είναι πολύ ικανοποιητική και οφείλεται σε μεγάλο βαθμό στην απόδοση της Myrinet διασύνδεσης.
- Παρατηρήσαμε ότι την μεγαλύτερη βαρύτητα στην επιτάχυνση την έχει ο αριθμός «διαφορετικών» επεξεργαστών που συμμετέχει στην επίλυση και όχι τόσο πολύ ο αριθμός διεργασιών (π.χ. δύο ανά επεξεργαστή).
- Ήταν ξεκάθαρο και επιβεβαιώνεται από τα διαγράμματα MPE πως ο βαθμός της επιτάχυνσης επηρεάζεται από τον λόγο υπολογισμός/επικοινωνία. Είναι ένα σημείο όπου επίσης φαίνεται η ικανοποιητική απόδοση της Myrinet διασύνδεσης.
- Συμπεραίνουμε έτσι, πως πέραν της παράλληλης επεξεργασίας, υπάρχουν άλλα κρίσιμα ζητήματα υλοποίησης (δομή δεδομένων, πρλυτικές διαδικασίες, κ.α.) που επηρεάζουν κρίσιμα τον τελικό χρόνο επίλυσης.
- Τα αποτελέσματα-μετρήσεις όσον αφορά στην επιτάχυνση (speed-up) για μεγάλα σε μέγεθος προβλήματα ήταν πολύ ικανοποιητικά (έως και 7,5 για 8 επεξεργαστές με δύο διεργασίες ανά επεξεργαστή / hypethreading) και ανάλογης έκτασης των αναγνωρισμένων δημοσιευμένων εργασιών της βιβλιογραφίας.
- Οι χρόνοι απόκρισης είναι επίσης αρκετά ικανοποιητικοί και – λόγω και της επιτευχθείσας επιτάχυνσης – καθιστούν εφικτή την επίλυση μεγάλων προβλημάτων σε αρκετά ικανοποιητικό πραγματικό χρόνο.
- Τα αποτελέσματα είναι εμφανές επίσης ότι θα μπορούσαν να κλιμακωθούν ανάλογα αν εκτελούντο προβλήματα ακόμα μεγαλύτερου μεγέθους, ενώ είναι εμφανής και η αναμενόμενη κλιμάκωση σε clusters περισσοτέρων επεξεργαστών (π.χ. 32, 128 κλπ), αρκεί να είναι διαθέσιμο ένα επαρκούς απόδοσης δίκτυο διασύνδεσης όπως το δίκτυο τύπου Myrinet που χρησιμοποιήθηκε στις παρούσες μετρήσεις.



5 Παραρτήματα

5.1 Βιβλιογραφία

- 1 Dantzig, G.B., and Thapa, M.N.: ‘Linear Programming, Introduction’ (Springer, 1997. 1997)
- 2 http://en.wikipedia.org/wiki/Simplex_algorithm
- 3 Chinneck, J.W.: ‘Practical Optimization: A Gentle Introduction’, in Editor (Ed.)^(Eds.): ‘Book Practical Optimization: A Gentle Introduction’ (2000, edn.), pp.
- 4 Chiang, A.C.: ‘Fundamental methods of mathematical economics’ (McGraw Hill, 1984, 3 edn. 1984)
- 5 Quinn, M.: ‘Parallel Programming in C with MPI and OpenMP’ (McGraw Hill, 2003, 1 edn. 2003)
- 6 Pacheco, P.S.: ‘A User's Guide to MPI’, in Editor (Ed.)^(Eds.): ‘Book A User's Guide to MPI’ (1998, edn.), pp.
- 7 Yarmish, G., and Slyke, R.V.: ‘A Distributed, Scaleable Simplex Method’, The Journal of Supercomputing, 2008
- 8 Αμπατζόγλου, Α.: ‘Υλοποίηση αναθεωρημένου αλγορίθμου simplex : για το γενικό γραμμικό πρόβλημα’, 2005
- 9 <http://matgen.sourceforge.net/2009>
- 10 Yarmish, G.: ‘A Distributed Implementation of the Simplex Method’, Polytechnic University, Michigan, USA, 2001
- 11 Hall, J. (2007). Towards a practical parallelisation of the simplex method. www.optimization-online.org.
- 12 Yarmish, G. and R. V. Slyke (2008). "A Distributed, Scaleable Simplex Method." The Journal of Supercomputing.
- 13 Badr, E.-S., N. Samaras, et al. (2006). Some Computational Results on MPI Parallel Implementation of Dense Simplex Method. PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY.



- 14 Yarmish, G. (2001). A Distributed Implementation of the Simplex Method, Polytechnic University, Michigan, USA.
- 15 Chinneck, J. W. (2000). Practical Optimization: A Gentle Introduction.
- 16 Cormen, T. H., C. E. Leiserson, et al. (2001). Introduction to Algorithms, MIT Press.
- 17 Kernighan, B. W. and D. M. Ritchie (1988). The C programming language, Prentice Hall.
- 18 "How to install C/C++ IDE for Netbeans." 2008, from <http://www.netbeans.org/community/releases/60/cpp-setup-instructions.html>.
- 19 Bezanson, J. "Understanding Floating Point Numbers", from http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point_representation.html.
- 20 . "LP Solve." 2008, from <http://lpsolve.sourceforge.net/>
- 21 Γ. Πάντζιου και Α. Τομαράς. Στοιχεία Παράλληλου Υπολογισμού. Εκδόσεις Νέων Τεχνολογιών, 2003.
- 22 L. G. Valiant et. al. General Purpose Parallel Architectures. In Handbook of Theoretical Computer Science, edited by J. van Leeuwen, MIT Press/Elsevier, Cambridge, MA/Amsterdam, 1990. pp. 943-972.
- 23 A. V. Gerbessiotis and L. G. Valiant. Direct Bulk-Synchronous Parallel Algorithms. Proc. Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science. Vol. 621. Springer-Verlag. Berlin, 1992. pp. 1-18.
- 24 F. Dehne (editor). Special Issue on “Coarse Grained Parallel Algorithms”. Algorithmica. 24(3/4), 1999.
- 25 I. Foster, Designing and Building Parallel Programs, (book) Addison-Wesley, 1995.
- 26 MPE: <http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm>
- 27 Jumpsot: <http://www.mcs.anl.gov/projects/perfvis/software/viewers/index.htm>



5.2 Πηγαίος Κώδικας

Η γενική διαρθρωση του προγράμματος αποτελείται από τα εξής αρχεία

1. mpi_simplex_v2.h
2. mpi_simplex_v2_main.c
3. mpi_load_mps.c
4. mpi_distr_lp_data.c
5. mpi_simplex.c
6. mpi_simplex_profiling.c
7. mpi_report.c

Για λόγους οικονομίας χώρου, ο πηγαίος κώδικας δίνεται για όλα τα παραπάνω αρχεία, πλήν του 7 και εκτενής σχολιασμός γίνεται μόνο για τα 1,2,4,5, καθώς τα υπόλοιπα αρχεία δεν πάιζουν ουσιαστικό ρόλο στην επίλυση του προβλήματος. Για το 7 δίνονται μόνο οι συναρτήσεις που κάνουν την μέτρηση των χρονικών διαστημάτων.



5.2.1 mpi_simplex_v2.h

```
1  /*
2   * File:  mpi_simplex_v2.h
3   * Author: jkr
4   *
5   * Created on March 13, 2009, 2:33 PM
6
7   * Στο αρχείο αυτό υπάρχουν τα includes και τα definitions
8   * των μεταβλητών και των δομών του προγράμματος που
9   * χρησιμοποιούνται από όλα τα υπόλοιπα αρχεία
10  */
11
12 //Τα includes
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <math.h>
16 #include <string.h>
17 #include <mpi.h>
18 #include <mpe.h>
19
20 //Για την ομιοόμορφη εκτύπωση των πινάκων (σε DEBUG mode)
21 #define PRINTF_FLOAT_DELIMITER "%+5.4f"
22 #define PRINTF_FLOAT_DELIMITER_CSV "%+5.4f;"
23
24 //Ορισμοί των τύπων των περιορισμών
25 #define kindofDECISION 0
```



```
26 #define kindofSLACK1
27 #define kindofSURPLUS 2
28 #define kindofARTIFICIAL 3
29
30 //Ορισμός των μέγιστων επαναλήψεων πριν θεωρηθεί ότι
31 //κόλλησε η simplex
32 #define MAX_ITERATIONS 2500
33
34 //Ορισμός του ποια εργασία είναι ο ROOT και του γενικού
35 //tag που θα χρησιμοποιείται για την επικοινωνία
36 #define ROOT 0
37 #define MYTAG 66
38
39 //Άλλοι ορισμοί
40 #define INF -1
41 #define TRUE 1
42 #define FALSE 0
43
44 //Ορισμός του EPSILON, δηλαδή της τιμής μικρότερης της οποίας
45 //θα θεωρείται η διαφορά δύο αριθμών μηδέν. Αυτό είναι απαραίτητα
46 //για να διαφυλαχθεί η αριθμητική σταθερότητα (numerical stability)
47 //του προγράμματος. Ορίζουμε και ένα πιο «χαλαρό» EPSILON έτσι
48 //ώστε να ελέγχεται η τιμή της αντικειμενικός συνάρτησης της Φάσης 1
49 #define EPS_VALUE 1e-7
50 #define EPS_VALUE_LOOSE 5
51 //Ορίζεται το πότε δύο αριθμοί τύπου double θα είναι ίσοι
52 #define double_equals(a,b) (fabs((a)-(b))<EPS_VALUE)
53 #define double_equals_loose(a,b) (fabs((a)-(b))<EPS_VALUE_LOOSE)
```



54

55 //Με αυτούς τους τύπους, μπορούμε να βρούμε πως θα διαμοιράζονται

56 //n στήλες στην εργασία με αριθμό id όταν υπάρχουν p συνολικά

57 //εργασίες (Quinn, p. 485)

58

59 //Το πρώτο στοιχείο του επεξεργαστή id (από 0 εώς n-1) στην συνολική

60 //αρίθμηση του πίνακα

61 #define BLOCK_LOW(id,p,n) ((id)*(n)/(p))

62 //Το τελευταίο στοιχείο του επεξεργαστή id (από 0 εώς n-1) στην συνολική

63 //αρίθμηση του πίνακα

64 #define BLOCK_HIGH(id,p,n) (BLOCK_LOW((id)+1,p,n)-1)

65 //Ο αριθμός των στηλών του επεξεργαστή id (από 1 εώς n) από το σύνολο αυτών

66 #define BLOCK_SIZE(id,p,n) (BLOCK_LOW((id)+1,p,n)-BLOCK_LOW(id,p,n))

67 //Ποιος είναι ο αριθμός του επεξεργαστή που είναι υπεύθυνος για την

68 //index στήλη ;

69 #define BLOCK_OWNER(index,p,n) (((p)*((index)+1)-1)/(n))

70

71 //Η δομή που αποθηκεύεται το αποτέλεσμα της επιλογής εισερχόμενης στήλης

72 //value είναι η τιμή της παραμέτρου της αντικειμενικής συνάρτησης,

73 //index είναι ο επεξεργαστής ο οποίος την κατέχει

74 struct obj_Coef {

75 double value;

76 int index;

77 };

78

79 //Η δομή ενός γενικού προβλήματος LP έτσι όπως φορτώνεται από

80 //το LP_SOLVE

81 /*



```
82      * 0 row is full of zeros
83      * 1st row is first constraint
84      */
85  struct lp_general_form {
86      int num_of_decision_variables ;
87      int num_of_constraints ;
88      int *row_type;
89      double **Constraints_CoEfficients;
90      double *ConstraintsRHS;
91      double *Objective_CoEfficients;
92      int type_is_maxim;
93  }; //τέλος της δομής
94
95  //Η δομή του LP προβλήματος που βρίσκεται σε κανονική μορφή
96  /*
97      * 0 row is first constraint
98      */
99  struct lp_standard_form {
100     int vars_decision_count;
101     int vars_constraints_count;
102     int vars_artificials_count;
103     int vars_all_count;
104     int *vars_KindOf; //0=decision, 1=slack, 2=artifical
105     int *NonBasicVarsIndexes; //N set
106     int NonBasicVars_count;
107     int *BasicVarsIndexes; //B set
108     int BasicVars_count;
109     double **Constraints_CoEfficients; //A matrix
```



```
110     double *ConstraintsRHS; //b vector
111     double *Objective_CoEfficients; //c vector
112     double CurrentValue_Objective; //u value
113     double *P1_Objective_CoEfficients;
114     double P1_CurrentValue_Objective;
115     int type_is_maxim; //1=maximize, 0=minimize
116     int P1_steps;
117     int P2_steps;
118 }; //τέλος της δομής
119
120
```



5.2.2 mpi_simplex_v2_main.c

```
1  /*
2   * File: mpi_simplex_v2_main.c
3   * Author: jkr
4   * Created on March 13, 2009, 2:38 PM
5
6   Πρόκειται για το αρχείο από το οποίο ξεκινάει η εκτέλεση του
7   προγράμματος.
8  */
9
10 #include "mpi_simplex_v2.h"
11
12 /*
13   Εδώ δηλώνονται όλες οι μεταβλητές οι οποίες χρειάζονται σε όλα τα
14   υπόλοιπα κομμάτια του προγράμματος. Θεώρησα ότι είναι καλύτερο το
15   να δηλωθούν ως global (extern) γιατί ο κώδικας θα ήταν πολύ πιο
16   καθαρός. Επιπλέον, σχεδόν όλες οι επιμέρους συναρτήσεις χρειάζονται
17   τις μεταβλητές αυτές. Η μη αντικειμενοστρεφής προσέγγιση της γλώσσας
18   C κάνει λίγο περιέργο το να δηλώνονται τόσες μεταβλητές ως global.
19   Κάνει επίσης δύσκολη την χρήση των συναρτήσεων που αναπτύχθηκαν
20   στα πλαίσια αυτού του προγράμματος και σε άλλες εφαρμογές.
21   Θεώρησα όμως ότι η καθαρότητα και η ευκρίνεια του κώδικα, προέχει σε
22   αυτή την περίπτωση
23  */
24
25 //Το αναγνωριστικό της τρέχουσας εργασίας (process)
```



```
26     int myid;  
27  
28     //Το πλήθος των επεξεργαστών  
29     int numprocs;  
30  
31     //Το επιστρεφόμενο status από τις MPI κλήσεις  
32     MPI_Status status;  
33  
34     //Η τρέχουσα Φάση της Simplex. Αρχικοποιείται σε 0  
35     int Phase = 0;  
36  
37     //Ο αριθμός βημάτων της Φάσης 1 και Φάσης 2  
38     int step_P1=0;  
39     int step_P2=0;  
40  
41     //Ο αριθμός των συνολικών στηλών του προβλήματος  
42     int g_total_cols;  
43  
44     //Ο αριθμός των στηλών που κατέχει η τρέχουσα εργασία  
45     int l_total_cols;  
46  
47     //Ο αριθμός των συνολικών γραμμών του προβλήματος.  
48     //Είναι κοινός για όλες τις εργασίες. Είναι περιττό δηλαδή  
49     // το να υπάρχει η μεταβλητή l_total_rows  
50     int g_total_rows;  
51  
52     //Ο δυσδιάστατος πίνακας των περιορισμών. Προτιμήθηκε  
53     //δυναμικά οριζόμενος πίνακας (με pointers δηλαδή) γιατί
```



```
54 //δινει ευελιξία
55 double **A;
56
57 //Το διάνυσμα των συντελεστών της αντικειμενικής συνάρτησης
58 //των Φάσεων 1(c_P1) και 2 (c)
59 double *c;
60 double *c_P1;
61
62 //Το διάνυσμα του δεξιού μέρους των περιορισμών
63 double *b;
64
65 //Η τρέχουσα τιμή της αντικειμενικής συνάρτησης των
66 //Φάσεων 1 (u_P1) και 2 (u)
67 double u=0.0;
68 double u_P1=0.0;
69
70 //Το σύνολο στο οποίο δηλώνεται η κάθε μεταβλητή τί τύπου είναι.
71 //Δηλαδή εαν είναι μεταβλητή απόφασης, πλεονάσματος ή slack,
72 //ή τεχνητή. Χρησιμεύει κατα την Φάση 2, όπου θέλουμε να λαμβά-
73 //νουμε υπόψιν μόνο τα τρία πρώτα είδη μεταβλητών
74 int *kindOf;
75
76 //Το σετ των μη βασικών μεταβλητών.
77 //Π.χ. το τρίτο στοιχείο του σετ, δηλώνει ποιά μεταβλητή απεικονίζεται
78 //στην τρίτη στήλη
79 int *N;
80
81 //Το σετ των βασικών μεταβλητών
```



```
82 //Π.χ. το τρίτο στοιχείο του σετ, δηλώνει ποιά μεταβλητή απεικονίζεται
83 //στην τρίτη σειρά
84 int *B;
85
86 //Μία μεταβλητή που δηλώνει το βαθμό που εκτυπώνονται βοηθητικές
87 //στην εκσφαλμάτωση πληροφορίες. Δίνεται σαν παράμετρος στο εκτελέ-
88 //σιμο αρχείο (μέσω *argv)
89 int DEBUG=0;
90
91 //Μία μεταβλητή που δηλώνει εάν θα παραχθούν MPE
92 logging μηνύματα. Δίνεται σαν παράμετρος στο εκτελέ-
93 //σιμο αρχείο (μέσω *argv)
94 int MPE_DEBUG;
95
96 //Είναι τα MPE Events
97 //load and distribute data
98 int _mpe_load_s, _mpe_load_e;
99
100 //local e
101 int _mpe_choose_local_e_s, _mpe_choose_local_e_f;
102 int _mpe_bcast_local_e_s, _mpe_bcast_local_e_f;
103
104 //local l
105 int _mpe_choose_local_l_s, _mpe_choose_local_l_f;
106 int _mpe_bcast_local_l_s, _mpe_bcast_local_l_f;
107
108 //computation
109 int _mpe_make_pivot_s, _mpe_make_pivot_f;
```



```
110 //  
111 double times[10];  
112  
113 int main(int argc, char** argv) {  
114  
115     //η δομή μέσα στην οποία διαβάζεται το MPS αρχείο  
116     struct lp_standard_form my_std_lp;  
117  
118     //οι μεταβλητές στις οποίες αποθηκεύονται τα αποτελέσματα της  
119     //simplex για τις φάσεις 1, 2  
120     int P1_res,P2_res;  
121  
122     //Αρχικοποίηση περιβιβάλλοντος MPI και MPE  
123     MPI_Init(&argc,&argv);  
124     MPI_Init_log();  
125     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);  
126     MPI_Comm_rank(MPI_COMM_WORLD,&myid);  
127  
128     //Αρχικοποίηση μεταβλητών MPE  
129     mpe_go();  
130  
131     //Αρχικά διαβάζεται από την ROOT εργασία το πρόβλημα  
132     if(myid==ROOT) {  
133         //Μεταδίδεται και στα υπόλοιπα μέλη της ομάδας το επίπεδο εκσφαλμάτωσης  
134         DEBUG = (int)atoi(argv[3]);  
135         MPE_DEBUG = (int)atoi(argv[4]);  
136         if(numprocs>1) {  
137             MPI_Bcast(&DEBUG,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);
```



```
138         MPI_Bcast(&MPE_DEBUG,1,MPI_INT,ROOT,MPI_COMM_WORLD);  
139     }  
140     }  
141 //και λαμβάνεται από τις υπόλοιπες εργασίες το επίπεδο εκφραλμάτωσης  
142 else {  
143     if(numprocs>1) {  
144         //Λήψη επιπέδου εκφραλμάτωσης  
145         MPI_Bcast(&DEBUG,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);  
146         MPI_Bcast(&MPE_DEBUG,1,MPI_INT,ROOT,MPI_COMM_WORLD);  
147     }  
148 }  
149  
150 //Αρχίζει το φόρτωμα των δεδομένων. Κρατάμε σχετικό MPE μήνυμα  
151 if(MPE_DEBUG) MPE_Log_event(_mpe_load_s,0,"Load & distr MPS file start");  
152 if(myid==ROOT) {  
153     log_time(1,FALSE);  
154     mpi_load_mps(&my_std_lp, argv[1],strcmp(argv[2],"0"));  
155     log_time(2,FALSE);  
156 }  
157  
158 //Διαμοιρασμός των δεδομένων  
159 log_time(3,TRUE);  
160 mpi_distr_lp_data(&my_std_lp);  
161 log_time(4,TRUE);  
162 if(MPE_DEBUG) MPE_Log_event(_mpe_load_e,0,"Load & distr MPS file finish");  
163  
164 //Μπαίνουμε στην Φάση 1  
165 log_time(5,TRUE);
```



```
166     Phase = (int)1;  
167  
168     if(DEBUG>=3) print_tableau_format_MPI_csv();  
169  
170     //Ελέγχουμε εάν η τρέχους λύση είναι σε εφικτό1  
171     //σημείο του προβλήματος. Εάν ναι, δεν είναι ανάγκη  
172     // να πάμε στην Φάση I. Άλλιώς εκτελούμε την Φασή 1  
173     //και αποθηκεύουμε το αποτέλεσμα της  
174     if(double_equals_loose(u_P1,0.0)) {  
175         P1_res = 1;  
176     }  
177     else {  
178         P1_res = MPI_Simplex();  
179     }  
180     log_time(6,TRUE);  
181  
182     //Αυτό μεταδίδεται και στις άλλες εργασίες  
183     if(numprocs>1) MPI_Bcast(&P1_res,1,MPI_INT,ROOT,MPI_COMM_WORLD);  
184  
185     //Εαν το αποτέλεσμα είναι επιτυχής κατάληξη (feasibility) της Φάσης 1  
186     //τότε προχώρα  
187     log_time(7,TRUE);  
188     if(P1_res==1) {  
189         //εκτύπωσε από τον ROOT ενημερωτικό μήνυμα  
190         if(myid==ROOT) { printf("\n%d, Phase 1 did return feasible solution after %d  
191         steps. P2 objective = %g. Going to Phase 2.",myid, step_P1,u ); }  
192  
193     //Μπήκαμε στην Φάση 2
```



```
194     Phase = (int)2;  
195  
196     // Εκτελούμε την Φασή 2 και αποθηκεύουμε το  
197     // αποτέλεσμα της  
198     P2_res = MPI_Simplex();  
199  
200     //Το εκτυπώνει μόνο ο ROOT, είτε θετικό είτε αρνητικό1  
201     if(myid==ROOT) {  
202         if(P2_res==1)  
203             printf("\nP%d, Phase 2 finished in %d steps. Objective Function  
204             value: %g",myid,step_P2,u);  
205         else  
206             printf("\nPhase 2 did not return feasible solution.\nNum of  
207             iterations: %d\nPhase 2 obj function value:%g\nReturn code: %d.",step_P2,u,P2_res) ;  
208     } //τέλος εάν είναι ο ROOT  
209     } //τέλος εάν επιτυχής η Φάση 1  
210     log_time(8,TRUE);  
211  
212     //εαν δεν κατέληξε σε εφικτή λύση η Φάση 1, τερματισε. Ο ROOT δίνει αιτιολογία  
213     else {  
214         if(myid==ROOT) {  
215             printf("\nPhase 1 did not return feasible solution.\nNum of iterations:  
216             %d\nPhase 1 obj function value:%g\nReturn code: %d.",step_P1,u_P1,P1_res) ;  
217         }  
218     } //τέλος εάν αποτυχής η Φάση 1  
219  
220     //Εκτύπωσε τον πίνακα της simplex, ανάλογα με επίπεδο εκφραλμάτωσης  
221     if(DEBUG>=3) print_tableau_format_MPI ()
```



222

```
223     if(MPE_DEBUG)MPE_Finish_log(argv[1]);  
224     MPI_Finalize();  
225  
226     if(myid==ROOT) {  
227         print_times_to_file("times.csv",argv[1],P1_res,P2_res);  
228         if(DEBUG>=1) {  
229             print_times_to_file("stdout",argv[1],P1_res,P2_res);  
230             printf("\n");  
231         }  
232     }  
233  
234 } //τέλος main  
235
```



5.2.3 mpi_load_mps.c

```
1  /*
2   * File:  mpi_simplex_v2_main.c
3   * Author: jkr
4   *
5   * Created on March 13, 2009, 3:09 PM
6   */
7
8  extern DEBUG, MPE_DEBUG;
9
10 #define LPSOLVEAPIFROMLIB
11 #define LPSOLVELIB "lp_solve_inc/liblpsolve55.so"
12
13 #include "lp_solve_inc/lp_lib.h"
14 #include "lp_solve_inc/lp_explicit.h"
15 #include "lp_solve_inc/lpkit.h"
16 #include "./mpi_simplex_v2.h"
17
18 static hlpssolve hlpssolve_ = NULL;
19
20 # define init_lpsolve_lib() ((hlpssolve_ != NULL) || (((hlpssolve_ = open_lpsolve_lib(LPSOLVELIB)) != NULL) &&
21 init_lpsolve(hlpssolve_)))
22 # define exit_lpsolve_lib() { if (hlpssolve_ != NULL) close_lpsolve_lib(hlpssolve_); hlpssolve_ =
23 NULL; }
24
25 //read mps and convert to standard form
26 int mpi_load_mps(struct lp_standard_form *my_std_lp, char *filename, int is_maximize) {
```



27

```
28     struct lp_general_form mylp;
29
30     mpi_read_mps(&mylp,filename,is_maximize);
31
32
33 //read from file into an lp problem
34 int mpi_read_mps(struct lp_general_form *mylp, char *filename, int is_maximize) {
35     int i=0,j=0;
36     lprec *lp=NULL;
37
38     if(! init_lpsolve_lib()) {
39         fprintf(stderr, "Unable to load lpsolve shared library (%s).\nIt is probably not in the correct path.\n", LPSOLVELIB);
40         return 0;
41     }
42
43     lp = read_MPS(filename,3);
44
45     if(lp == NULL) {
46         fprintf(stderr, "Unable to read mps file: ");
47         return 0;
48     }
49
50     //set maxim or minim
51
52     if(is_maximize) {mylp->type_is_maxim=1;}
53     else {mylp->type_is_maxim=0;}
54
55     //copy num of rows
56
57     mylp->num_of_decision_variables = get_Norig_columns(lp);
58
59     //copy num of constraints
60
61     mylp->num_of_constraints = get_Norig_rows(lp);
62
63     //malloc
```



```
55     mylp->ConstraintsRHS = malloc(((mylp->num_of_constraints)+1)*sizeof(double));  
56     mylp->Objective_CoEfficients           =                         malloc(((mylp-  
57     >num_of_decision_variables)+1)*sizeof(double));  
58     mylp->Constraints_CoEfficients = malloc((mylp->num_of_constraints+1)*sizeof(double));  
59     for(i=0;i<=mylp->num_of_constraints;i++) {  
60         mylp->Constraints_CoEfficients[i]=malloc((mylp-  
61     >num_of_decision_variables+1)*sizeof(double));  
62     }  
63     //copy RHS  
64     memcpy(mylp->ConstraintsRHS,lp->orig_rhs,sizeof(double)*(1+mylp->num_of_constraints));  
65     //copy constraint coefficients  
66     for(i=1;i<=(mylp->num_of_constraints);i++) {  
67         for(j=1;j<=(mylp->num_of_decision_variables);j++) {  
68             mylp->Constraints_CoEfficients[i][j] = (double)mat_getitem(lp->matA,i,j);  
69         }  
70     }  
71     //copy Objective_CoEfficients  
72     memcpy(mylp->Objective_CoEfficients,lp->orig_obj,(mylp-  
73     >num_of_decision_variables+1)*sizeof(double));  
74     //copy row_type  
75     mylp->row_type = malloc((mylp->num_of_constraints+1)*sizeof(int));  
76     memcpy(mylp->row_type,lp->row_type,sizeof(int)*(1+mylp->num_of_constraints));  
77     free(lp);  
78 } //end if read success  
79     return 1;  
80 } //end function  
81  
82 //convert from a general lp problem to a standard lp problem
```



```
83     int mpi_convert_to_std_form( struct lp_general_form *mylp,struct lp_standard_form *my_std_lp) {  
84         int i=0,j=0,mult=0,tot_cols_needed=0;  
85  
86         //copy things that do not need any process  
87         //look for negative rhs on mylp and convert accordingly  
88         for(i=0;i<mylp->num_of_constraints;i++) {  
89             if(mylp->ConstraintsRHS[i]<0.0) {  
90                 //make it positive  
91                 mylp->ConstraintsRHS[i] *= (-1.0);  
92                 //reverse equality type  
93                 if( mylp->row_type[i] ==LE) { mylp->row_type[i] = GE;}  
94                 else if (mylp->row_type[i]==GE){mylp->row_type[i] = LE;}  
95                 //reverse sign on variables  
96                 for(j=0;j<mylp->num_of_decision_variables;j++) {  
97                     if(! double_equals(mylp->Constraints_CoEfficients[i][j],0.0)) {  
98                         mylp->Constraints_CoEfficients[i][j] *= (-1.0);  
99                     }  
100                }  
101            }  
102        }  
103        //initialize Steps  
104        my_std_lp->P1_steps = 0;  
105        my_std_lp->P2_steps = 0;  
106        //set Basic - NonBasic Sets count to 0  
107        my_std_lp->BasicVars_count = 0;  
108        my_std_lp->NonBasicVars_count = 0;  
109        //copy MAXIM or MINIM  
110        my_std_lp->type_is_maxim = mylp->type_is_maxim;
```



```
111 //set Phase 1 Artificial Vars number to 0
112 my_std_lp->vars_artificials_count=0;
113 //copy num of decision variables
114 my_std_lp->vars_decision_count = mylp->num_of_decision_variables;
115 //copy num of constraints
116 my_std_lp->vars_constraints_count = mylp->num_of_constraints;
117 //calculate total columns needed
118 tot_cols_needed = calc_Constraints_CoEfficients_cols(mylp);
119 //Initialize objective value
120 my_std_lp->CurrentValue_Objective = 0.0;
121 my_std_lp->P1_CurrentValue_Objective =0.0;
122 //initialize RHS, Con_Coef, Obj_Coef, P1_Ob_Coef with malloc
123 my_std_lp->ConstraintsRHS = malloc((mylp->num_of_constraints)*sizeof(double));
124 memset(my_std_lp->ConstraintsRHS,0.0,(mylp->num_of_constraints)*sizeof(double));
125 my_std_lp->Objective_CoEfficients = malloc(tot_cols_needed*sizeof(double));
126 memset(my_std_lp->Objective_CoEfficients,0.0,tot_cols_needed*sizeof(double));
127 my_std_lp->P1_Objective_CoEfficients = malloc(tot_cols_needed*sizeof(double));
128 memset(my_std_lp->P1_Objective_CoEfficients,0.0,tot_cols_needed*sizeof(double));
129 //Constraint Coefficients
130 my_std_lp->Constraints_CoEfficients = malloc(my_std_lp-
131 >vars_constraints_count*sizeof(double));
132 for(i=0;i<mylp->num_of_constraints;i++) {
133     my_std_lp->Constraints_CoEfficients[i]=malloc(tot_cols_needed*sizeof(double));
134     memset(my_std_lp->Constraints_CoEfficients[i],0.0,tot_cols_needed*sizeof(double));
135 }
136 //initialize Basic, Non-Basic Sets, vars_Kind_Of
137 my_std_lp->BasicVarsIndexes = malloc(my_std_lp->vars_constraints_count*sizeof(int)); ;
138 my_std_lp->NonBasicVarsIndexes = malloc(tot_cols_needed*sizeof(double));
```



```
139     my_std_lp->vars_KindOf           =           malloc((my_std_lp-
140   >vars_constraints_count+tot_cols_needed)*sizeof(int));
141   if(my_std_lp->type_is_maxim){mult=-1;} else {mult=1;}
142   //copy Objective_CoEfficients
143   for(j=1;j<=(mylp->num_of_decision_variables);j++) {
144     my_std_lp->Objective_CoEfficients[j-1] = mult*mylp->Objective_CoEfficients[j];
145   }
146   //Create all Basic-NonBasic Sets
147   int BasicIndex_current=-1; int NonBasicIndex_current=-1; int generalVarsIndex=-1;
148   //Add all decision variables to non-basic set and to DecisionVarsIndexes
149   for(i=0;i<my_std_lp->vars_decision_count;i++) {
150     generalVarsIndex++;
151     my_std_lp->NonBasicVarsIndexes[++NonBasicIndex_current]=generalVarsIndex;
152     my_std_lp->vars_KindOf[generalVarsIndex]=kindofDECISION;
153   }
154   for(i=1;i<=mylp->num_of_constraints;i++) { //loop through rows
155     generalVarsIndex++;
156     if (mylp->row_type[i]==EQ) {
157       //add artificial
158       my_std_lp->vars_artificials_count++;
159       my_std_lp->vars_KindOf[generalVarsIndex]=kindofARTIFICIAL; //add to artificials
160       my_std_lp->BasicVarsIndexes[++BasicIndex_current]=generalVarsIndex;
161     } //end if EQ
162     else if (mylp->row_type[i]==LE) {
163       //add to slacks
164       my_std_lp->vars_KindOf[generalVarsIndex]=kindofSLACK;
165       my_std_lp->BasicVarsIndexes[++BasicIndex_current]=generalVarsIndex;
166     } //end if LE
```



```
167     else if (mylp->row_type[i]==GE) {  
168         //add the surplus  
169         my_std_lp->vars_KindOf[generalVarsIndex]=kindofSURPLUS;  
170         my_std_lp->NonBasicVarsIndexes[++NonBasicIndex_current]=generalVarsIndex;  
171         (my_std_lp->Constraints_CoEfficients[i-1])[NonBasicIndex_current] = -1.0;  
172         //add the artificial  
173         generalVarsIndex++;  
174         (my_std_lp->vars_artificials_count)++;  
175         my_std_lp->vars_KindOf[generalVarsIndex]=kindofARTIFICIAL;  
176         my_std_lp->BasicVarsIndexes[++BasicIndex_current]=generalVarsIndex;  
177     } //end if GE  
178 } //end loop rows  
179 //set Basic - NonBaasic Sets Length  
180 my_std_lp->NonBasicVars_count = NonBasicIndex_current+1;  
181 my_std_lp->BasicVars_count = BasicIndex_current+1;  
182 my_std_lp->vars_all_count = generalVarsIndex+1;  
183 //for each row: copy Constraints, copy RHS  
184 for(i=1;i<=mylp->num_of_constraints;i++) { //loop through rows  
185     if (mylp->row_type[i]==LE || mylp->row_type[i]==EQ ) { //if row is equal type  
186         //copy Constraint Coefficients  
187         for(j=1;j<=mylp->num_of_decision_variables;j++)  
188             my_std_lp->Constraints_CoEfficients[i-1][j-1] = mylp->Constraints_CoEfficients[i][j];  
189         //copy RHS original  
190         my_std_lp->ConstraintsRHS[i-1]=mylp->ConstraintsRHS[i];  
191     } //end if row is equal or LE type  
192     else if (mylp->row_type[i]==GE ) {  
193         //copy Constraint Coefficients  
194         for(j=1;j<=my_std_lp->vars_decision_count;j++) {
```



```
195     my_std_lp->Constraints_CoEfficients[i-1][j-1]      =      ((double)(-1.0))*(mylp-
196   >Constraints_CoEfficients[i][j]);
197   //copy RHS original
198   my_std_lp->ConstraintsRHS[i-1]=((double)(-1.0))*mylp->ConstraintsRHS[i];
199 }
200 } // end if GE
201 } //end looping throug constraints
202 //normalize tablaeu for P1
203 for(i=0;i<my_std_lp->BasicVars_count;i++) {
204   if(my_std_lp->vars_KindOff[my_std_lp->BasicVarsIndexes[i]] == kindofARTIFICIAL) {
205     //alter objective coefficients of P1
206     for(j=0;j<my_std_lp->NonBasicVars_count;j++) {
207       if(double_equals(my_std_lp->P1_Objective_CoEfficients[j],my_std_lp-
208 >Constraints_CoEfficients[i][j])) {
209         my_std_lp->P1_Objective_CoEfficients[j] = 0.0;
210       }
211     else {
212       my_std_lp->P1_Objective_CoEfficients[j]      -=      my_std_lp-
213 >Constraints_CoEfficients[i][j];
214     }
215   }
216   if(double_equals(
217 >ConstraintsRHS[i])){
218     my_std_lp->P1_CurrentValue_Objective=0.0;
219   }
220   else {
221     my_std_lp->P1_CurrentValue_Objective -= my_std_lp->ConstraintsRHS[i];
222   }
223 } //end if is Artificial
```



```
224      } //end of loop of basic set
225      return 1;
226  };
227
228
229  int calc_Constraints_CoEfficients_cols(struct lp_general_form *mylp) {
230      int i;
231      int ret = mylp->num_of_decision_variables;
232      for(i=1;i<=mylp->num_of_constraints;i++) {
233          if(mylp->row_type[i]==GE)
234              ret++;
235      }
236      return ret;
237  }
```



5.2.4 mpi_distr_lp_data.c

```
1  /*
2   * File:  mpi_simplex_distr.c
3   * Author: jkr
4   * * Created on March 14, 2009, 11:00 PM
5
6   * Είναι το αρχείο στο οποίο εκτελείται το σπάσιμο των δεδομένων (κατα
7   * βάση των στηλών) και το στάλσιμο τους στις επιμέρους εργασίες.
8   * Σαν μέθοδος διαμοιρασμού των δεδομένων έχει χρησιμοποιηθεί
9   * ο κατα στήλη συνεχόμενη διαμερισμός (columnwise block-striped
10  decomposition).
11  */
12
13 #include "mpi_simplex_v2.h"
14
15 //Λαμβάνουμε τις εξωτερικές μεταβλητές
16 extern int myid, numprocs,DEBUG;
17 extern int g_total_cols, l_total_cols, g_total_rows;
18 extern double **A;
19 extern double *b,*c,*c_P1;
20 extern double u,u_P1;
21 extern int *kindOf,*N,*B;
22 MPI_Status status;
23
24 extern int _mpe_load_s, _mpe_load_e;
25 extern int _mpe_broadcast_s, _mpe_broadcast_e;
```



```
26     extern int _mpe_compute_s, _mpe_compute_e;
27     extern int _mpe_reduce_all_s, _mpe_reduce_all_e;
28     extern int _mpe_barrier_s, _mpe_barrier_e;
29     extern char *msg_s, *msg_e;
30
31
32     int mpi_distr_lp_data( struct lp_standard_form *CurrentLP )
33     {
34         //Τοπικές μεταβλητές.
35         //Γενικά όπου i=τρέχουσα γραμμή, j=τρέχουσα στήλη, p=τρέχουσα εργασία
36         int i,j,p;
37
38         //Αρχικά μεταδίδουμε τα γενικά χαρακτηριστικά του προβλήματος
39         strcpy(msg_s,"Broadcast Gen.Prob start"); strcpy(msg_s,"Broadcast Gen.Prob end");
40
41         if(myid==ROOT) {
42             //Μετάδοση Καθολικού Αριθμού γραμμών (και ανάθεση στην αντίστοιχη τοπική
43             μεταβλητή του ROOT)
44             if(numprocs>1)                               MPI_Bcast(&(CurrentLP-
45             >BasicVars_count),1,MPI_INT,ROOT,MPI_COMM_WORLD);
46             g_total_rows = CurrentLP->BasicVars_count;
47             //Μετάδοση Καθολικού Αριθμού στηλών (και ανάθεση στην αντίστοιχη τοπική
48             μεταβλητή του ROOT)
49             if(numprocs>1)                               MPI_Bcast(&(CurrentLP-
50             >NonBasicVars_count),1,MPI_INT,ROOT,MPI_COMM_WORLD);
51             g_total_cols=CurrentLP->NonBasicVars_count;
52
53             //Υπολογισμός και Ανάθεση του Τοπικού Αριθμού Στηλών για τον ROOT
54             if(numprocs>1) l_total_cols = BLOCK_SIZE(myid,numprocs,g_total_cols);
```



```
55         else l_total_cols = g_total_cols;
56     } //τέλος η μετάδοση από ROOT
57     else {
58         //Οι υπόλοιπες εργασίες λαμβάνουν τα γενικά χαρκτηριστικά του προβλήματος
59         if(numprocs>1) {
60             //Λήψη Καθολικού Αριθμού γραμμών
61             MPI_Bcast(&g_total_rows,1,MPI_INT,ROOT,MPI_COMM_WORLD);
62             //Λήψη Καθολικού Αριθμού στηλών
63             MPI_Bcast(&g_total_cols,1,MPI_INT,ROOT,MPI_COMM_WORLD);
64             //Υπολογισμός του Τοπικού Αριθμού Στηλών
65             l_total_cols = BLOCK_SIZE(myid,numprocs,g_total_cols);
66         } //τέλος εάν οι εργασίες > 1
67     } //τέλος εάν η εργασία <> ROOT
68
69     if(DEBUG>=2)
70         printf("\nP%d:      mpi_distr_lp_data:      l_total_cols=%d,      g_total_cols=%d,
71 g_total_rows=%d, Columns from %d to %d, size is %d", myid,l_total_cols,g_total_cols,g_total_rows,
72 BLOCK_LOW(myid,numprocs,g_total_cols),
73 BLOCK_HIGH(myid,numprocs,g_total_cols),BLOCK_SIZE(myid,numprocs,g_total_cols) );
74
75     //Αρχικοποίησεις των δεδομένων του προβλήματος για την κάθε εργασία
76     if(MPE_DEBUG)      MPE_Log_event(_mpe_compute_s,0,"mpi_distr_lp_data:Allocating
77 arrays");
78
79     b = malloc(g_total_rows*(sizeof(double)));
80     memset(b,0.0,g_total_rows*sizeof(double));
81
82     c = malloc(l_total_cols*sizeof(double));
83     memset(c,0.0,l_total_cols*sizeof(double));
```



84

```
85     c_P1 = malloc(l_total_cols*sizeof(double));
86     memset(c_P1,0.0,l_total_cols*sizeof(double));
87
88     A = malloc(l_total_cols*sizeof(double));
89     for(j=0;j<l_total_cols;j++) {
90         A[j]=(void *) malloc(g_total_rows*sizeof(double));
91         memset(A[j],0.0,g_total_rows*sizeof(double));
92     }
93
94     kindOf = malloc((g_total_rows+g_total_cols)*sizeof(int));
95     N = malloc(g_total_cols*sizeof(int));
96     B = malloc(g_total_rows*sizeof(int));
97
98     if(MPE_DEBUG)      MPE_Log_event(_mpe_compute_e,0,"mpi_distr_lp_data:Allocating
99     arrays");
100
101    //Εδώ γίνεται η μετάδοση
102    if(myid==ROOT) {
103
104        //b
105        if(numprocs>1) {
106            MPI_Bcast(CurrentLP-
107            >ConstraintsRHS,g_total_rows,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);
108        }
109        memcpy(b,CurrentLP->ConstraintsRHS,g_total_rows*sizeof(double));
110
111        //c
```



```
112         if(numprocs>1) {  
113             Send_Vector_to_all(CurrentLP-  
114             >Objective_CoEfficients,g_total_cols,MYTAG);  
115         }  
116         memcpy(c,(CurrentLP-  
117             >Objective_CoEfficients)+BLOCK_LOW(myid,numprocs,g_total_cols),  
118             l_total_cols*sizeof(double));  
119  
120         //c_P1  
121         if(numprocs>1) {  
122             Send_Vector_to_all(CurrentLP-  
123             >P1_Objective_CoEfficients,g_total_cols,MYTAG);  
124         }  
125         memcpy(c_P1,(CurrentLP-  
126             >P1_Objective_CoEfficients)+BLOCK_LOW(myid,numprocs,g_total_cols),  
127             l_total_cols  
*sizeof(double));  
128  
129         //A  
130         if(numprocs>1) {  
131             Send_2d_Matrix_to_all(CurrentLP-  
132             >Constraints_CoEfficients,g_total_rows,g_total_cols,MYTAG);  
133         }  
134  
135         for(j=BLOCK_LOW(myid,numprocs,g_total_cols);j<=BLOCK_HIGH(myid,numprocs,g_to  
136             tal_cols);j++){  
137             for(i=0;i<g_total_rows;i++){  
138                 A[j][i]=CurrentLP->Constraints_CoEfficients[i][j];  
139             }  
140         }  
141
```



```
142          //u
143          u = CurrentLP->CurrentValue_Objective;
144          if(numprocs>1)
145      {MPI_Bcast(&u,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);}
146
147          //u_P1
148          u_P1 = CurrentLP->P1_CurrentValue_Objective;
149          if(numprocs>1)
150      {MPI_Bcast(&u_P1,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);}
151
152          //N
153          N = CurrentLP->NonBasicVarsIndexes;
154          if(numprocs>1) {
155              MPI_Bcast(CurrentLP-
156      >NonBasicVarsIndexes,g_total_cols,MPI_INT,ROOT,MPI_COMM_WORLD);
157          }
158
159          //B
160          B = CurrentLP->BasicVarsIndexes;
161          if(numprocs>1) {
162              MPI_Bcast(CurrentLP-
163      >BasicVarsIndexes,g_total_rows,MPI_INT,ROOT,MPI_COMM_WORLD);
164          }
165
166          //kindOf
167          kindOf = CurrentLP->vars_KindOf;
168          if(numprocs>1) {
169              MPI_Bcast(CurrentLP-
170      >vars_KindOf,g_total_rows+g_total_cols,MPI_INT,ROOT,MPI_COMM_WORLD);
```



```
171 }  
172 } // τέλος μετάδοσης από ROOT  
173  
174 //λήψη από τις υπόλοιπες εργασίες  
175 else {  
176     if(numprocs>1) {  
177         //b  
178  
179         MPI_Bcast(b,g_total_rows,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);  
180         //c  
181         Recv_Vector(c,g_total_cols,MYTAG);  
182         //c_P1  
183         Recv_Vector(c_P1,g_total_cols,MYTAG);  
184         //A  
185         Recv_2d_Matrix(A,g_total_rows,g_total_cols,MYTAG);  
186         //u  
187         MPI_Bcast(&u,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);  
188         //u_P1  
189         MPI_Bcast(&u_P1,1,MPI_DOUBLE,ROOT,MPI_COMM_WORLD);  
190         //N  
191         MPI_Bcast(N,g_total_cols,MPI_INT,ROOT,MPI_COMM_WORLD);  
192         //B  
193         MPI_Bcast(B,g_total_rows,MPI_INT,ROOT,MPI_COMM_WORLD);  
194         //kindOf  
195  
196         MPI_Bcast(kindOf,g_total_rows+g_total_cols,MPI_INT,ROOT,MPI_COMM_WORLD);  
197     } //τέλος εάν υπάρχουν πάνω από 1 διεργασίες  
198 } //τέλος λήψης από άλλες εργασίες
```



```
199         return 1;_distr
200     } //τέλος συνάρτησης mpi_simplex_distr
201
202
203     /*
204     Γενικού σκοπού συναρτήσεις, για την αποστολή και λήψη διανυσμάτων ή πινάκων.
205     Χρησιμοποιούν κατα στήλη συνεχόμενο διαμερισμό.
206     */
207
208     //Εκτελεί τις απαραίτητες MPI_Send εντολές για να μοιραστεί ένας δυσδιάστατος πίνακας
209     //που υπάρχει στην εργασία 0 στις υπόλοιπες εργασίες
210     //Πρώτα αντιστρέφει τον πίνακα, ώστε να δουλέψει με στήλες, αφού
211     //ο τρόπος που ο 2D πίνακας είναι φτιαγμένος στην c είναι κατα γραμμές
212     int Send_2d_Matrix_to_all(double **matrix, int total_rows, int total_cols, int tag ) {
213         int i=0,j=0,p=0;
214         double *transpose_buffer = malloc((total_rows)*sizeof(double));
215
216         for(p=1;p<numprocs;p++) {
217             for(j=BLOCK_LOW(p,numprocs,total_cols);j<=BLOCK_HIGH(p,numprocs,total_cols);j++){
218                 for(i=0;i<total_rows;i++){
219                     transpose_buffer[i]= matrix[i][j];
220                 }
221                 MPI_Send(transpose_buffer,total_rows,MPI_DOUBLE,p,tag,MPI_COMM_WORLD); //send matrix
222             }
223         } //end send to all processors
224     }
225
226     //Εκτελεί τις απαραίτητες MPI_Recv εντολές για να πάρει η εργασία 0, έναν δυσδιάστατο πίνακα
```



```
227 //που υπάρχει κατανεμημένος σε columnwise block-striped decomposition στις υπόλοιπες εργασίες
228 int Recv_2d_Matrix_from_all(double **matrix, int total_rows, int total_cols, int tag ) {
229     int i=0,j=0,p=0;
230     double *transpose_buffer = malloc((total_rows)*sizeof(double));
231     for(p=1;p<numprocs;p++) {
232         for(j=BLOCK_LOW(p,numprocs,total_cols);j<=BLOCK_HIGH(p,numprocs,total_cols);j++) {
233             MPI_Recv(transpose_buffer,total_rows,MPI_DOUBLE,p,tag,MPI_COMM_WORLD,&status);
234         //send matrix
235         for(i=0;i<total_rows;i++){
236             matrix[i][j]=transpose_buffer[i];
237         }
238     }
239     } //end send to all processors
240 }
241
242 //Εκτελεί τις απαραίτητες MPI_Recv εντολές για να πάρει μία εργασία, το κομμάτι
243 //που της αναλογεί από έναν δυσδιάστατο πίνακα
244 //Ουσιαστικά κάνει τόσα MPI_Recv, όσες είναι οι στήλες που πρέπει να λάβει
245 int Recv_2d_Matrix(double **matrix, int total_rows, int total_cols, int tag ) {
246     int j=0,i=0;
247     for(j=0;j<BLOCK_SIZE(myid,numprocs,total_cols);j++) {
248         MPI_Recv(matrix[j],total_rows,MPI_DOUBLE,ROOT,tag,MPI_COMM_WORLD,&status);
249     } //end for
250 }
251
252 //Εκτελεί τις απαραίτητες MPI_Send εντολές για να στείλει μία εργασία, το κομμάτι
253 //που κατέχει από τον διαμοιρασμένο δυσδιάστατο πίνακα στον ROOT
254 int Send_2d_Matrix(double **matrix, int total_rows, int total_cols, int tag ) {
```



```
255     int j=0,i=0;  
  
256     for(j=0;j<BLOCK_SIZE(myid,numprocs,total_cols);j++){  
  
257         MPI_Send(matrix[j],total_rows,MPI_DOUBLE,ROOT,tag,MPI_COMM_WORLD);  
  
258     } //end for  
  
259 }  
  
260  
  
261 //Εκτελούνται οι απαραίτητες MPI_send από τον ROOT για να κατανείμει ένα διάνυσμα  
262 //στις υπόλοιπες εργασίες  
  
263 int Send_Vector_to_all(double *vector,int total_el, int tag){  
  
264     //Block Data Dicomposition, p. 120, Quinn  
  
265     int p=0;  
  
266     for(p=1;p<numprocs;p++) {  
  
267         MPI_Send(vector+BLOCK_LOW(p,numprocs,total_el),BLOCK_SIZE(p,numprocs,total_el),MPI_D  
268         OUBLE,p,tag,MPI_COMM_WORLD); //send matrix  
  
269     } //end send to all processors  
  
270 }  
  
271  
  
272 //Εκτελούνται οι απαραίτητες MPI_Recv από τον ROOT για να λάβει ένα κατανεμημένο διάνυσμα  
273 //στις υπόλοιπες εργασίες  
  
274 int Recv_Vector_from_all(double *vector,int total_el, int tag){  
  
275     //Block Data Dicomposition, p. 120, Quinn  
  
276     int p=0;  
  
277     for(p=1;p<numprocs;p++) {  
  
278         MPI_Recv(vector+BLOCK_LOW(p,numprocs,total_el),BLOCK_SIZE(p,numprocs,total_el),MPI_D  
279         OUBLE,p,tag,MPI_COMM_WORLD,&status); //send matrix  
  
280     } //end send to all processors  
  
281 }  
  
282
```



```
283 //Λαμβάνει μία εργασία το κομμάτι που της αναλογεί από τον ROOT
284 int Recv_Vector(double *vector,int total_el, int tag) {
285
286 MPI_Recv(vector,BLOCK_SIZE(myid,numprocs,total_el),MPI_DOUBLE,ROOT,tag,MPI_COMM_
287 WORLD,&status);
288 }
289
290 //Στέλνει μία εργασία το κομμάτι που της αναλογεί στον ROOT
291 int Send_Vector(double *vector,int total_el, int tag) {
292
293 MPI_Send(vector,BLOCK_SIZE(myid,numprocs,total_el),MPI_DOUBLE,ROOT,tag,MPI_COMM_
294 WORLD);
295 }
```



5.2.5 mpi_simplex.c

```
1  /*
2   * File:  mpi_simplex.c
3   * Author: jkr
4   *
5   * Created on March 27, 2009, 11:38 AM
6
7   To κομμάτι αυτό, περιλαμβάνει όλες τις λειτουργίες που εκτελούνται
8   για να επιλυθεί ένα γραμμικό πρόβλημα που είναι σε κανονική μορφή
9   */
10
11 #include "mpi_simplex_v2.h"
12
13 //Δηλώνονται οι εξωτερικές μεταβλητές
14 extern int myid, numprocs,DEBUG;
15 extern int Phase, step_P1,step_P2;
16 extern int g_total_cols, l_total_cols, g_total_rows;
17 extern double **A;
18 extern double *b,*c,*c_P1;
19 extern double u,u_P1;
20 extern int *kindOf,*N,*B;
21 MPI_Status status;
22
23
24 int MPI_Simplex() {
25
```



```
26      // Ο τοπικός εξερχόμενος αριθμός στήλης (local entering)
27      int l_e;
28
29      //Ο καθολικός εξερχόμενος αριθμός στήλης (global entering)
30      int g_e;
31
32      //Ο καθολικός εισερχόμενος αριθμός γραμμής (global leaving)
33      int g_l;
34
35      //Η δομή στην οποία αποθηκεύεται η πληροφορία
36      //του αποτελέσματος της επιλογής για εισερχόμενη
37      //μεταβλητή και η οποία χρησιμοποιείται για την
38      //καθολική μείωση (δηλαδή, όλες οι εργασίες επιλέγουν
39      //εκείνη την στήλη η οποία έχει ην καθολικά μικρότερη
40      //τιμή στην παράμετρο της αντικειμενικής συνάρτησης)
41      struct obj_Coef l_min_c, g_min_c;
42
43      //Οι παράμετροι των αντικειμενικών συναρτήσεων (Φάσης 1 & 2)
44      //στην καθολικά εισερχόμενη μεταβλητή
45      double c_min_value, c_P1_min_value;
46
47      //Το διάνυσμα της καθολικά εισερχόμενης μεταβλητής
48      double *my_e_col=malloc(g_total_rows*sizeof(double)); \
49
50      //Αρχικά αναζητούμε για κάθε διεργασία την τοπικά εισερχόμενη μεταβλητή
51      //(επιλογή στήλης)
52      l_e=MPI_ChooseEnteringVariable(&l_min_c);
53
```



```
54 //Αφού λοιπόν έχει επιλεγεί τοπικά η στήλη, και ο αριθμός
55 //των διεργασιών >1, τότε προχωρούμε στην επιλογή της
56 //καθολικά εισερχόμενης μεταβλητής.
57 //Το MPI_Allreduce παίρνει μία δομή με index και value και
58 //επιστρέφει την δομή με κάποιου είδους καθολική πράξη.
59 //Στην περίπτωση μας κάνουμε εύρεση του minimum.
60 //Ετσι κάθε εργασία δίνει την τοπική δομή της l_min_c και στο
61 //τέλος του Reduce παίρνει μία δομή g_min_c με value την μικρότερη
62 //τιμή και index τον αριθμό της εργασίας που είχε αυτή την τιμή
63 if(numprocs>1)
64
65     MPI_Allreduce(&l_min_c,&g_min_c,1,MPI_DOUBLE_INT,MPI_MINLOC,MPI_COMM_
66 WORLD);
67
68     else
69
70         //Εάν η εργασία είναι αυτή με την καθολικά επιλεγμένη στήλη, τότε
71         if(myid==g_min_c.index) {
72
73             if(numprocs>1) //στείλε τον αριθμό της τοπικά επιλεγμένη στήλη σε όλους
74
75                 MPI_Bcast(&l_e,1,MPI_INT,myid,MPI_COMM_WORLD);
76
77                 //ενώ για αυτή την εργασία τάντισε την καθολική στήλη με την τοπική
78                 g_e = l_e;
79
80             }
81
82         else {
83
84             if(numprocs>1) //Λαβε τον αριθμό της καθολικής στήλης σαν g_e
85
86                 MPI_Bcast(&g_e,1,MPI_INT,g_min_c.index,MPI_COMM_WORLD);
87
88             }
89
90
91
```



```
82     MPI_Barrier(MPI_COMM_WORLD);  
83  
84     //Ξεκινάει το loop της simplex, όσο υπάρχει στήλη για επιλογή  
85     while( g_e != -1) {  
86  
87         // εργασία με την καθολική στήλη, επιλέγει και γραμμή  
88         if(myid==g_min_c.index) {  
89             g_l=MPI_ChooseLeavingVariableIndex(l_e);  
90             c_min_value = c[l_e];  
91             c_P1_min_value = c_P1[l_e];  
92             //Γίνεται η μετάδοση του αριθμού τη γραμμής  
93             if(numprocs>1) {  
94                 MPI_Bcast(&g_l,1,MPI_INT,myid,MPI_COMM_WORLD);  
95                 //Γίνεται η μετάδοση της επιλεγμένης στήλης  
96  
97                 MPI_Bcast(A[l_e],g_total_rows,MPI_DOUBLE,myid,MPI_COMM_WORLD);  
98                 memcpy(my_e_col,A[l_e],g_total_rows*sizeof(double));  
99                 //Γίνεται η μετάδοση της παραμέτρου της αντικειμενικής  
100                συνάρτησης  
101                //στην επιλεγμένη στήλη  
102  
103                MPI_Bcast(&c_min_value,1,MPI_DOUBLE,myid,MPI_COMM_WORLD);  
104                //εάν η τρέχουσα φάση είναι η 1, τότε μετέδωσε και την  
105                παράμετρο  
106                //της αντικειμενικής συνάρτησης αυτού του προβλήματος  
107                if(Phase==1)  
108  
109                MPI_Bcast(&c_P1_min_value,1,MPI_DOUBLE,myid,MPI_COMM_WORLD);  
110            } //τέλος εάν υπάρχουν περισσότερες από 1 εργασίες
```



```
111 } //τέλος μετάδοσης από εργασία με εισερχόμενη στήλη
112 else {
113     //Λήψη από εργασία με επιλεγμένη στήλη
114     if(numprocs>1) {
115
116         MPI_Bcast(&g_l,1,MPI_INT,g_min_c.index,MPI_COMM_WORLD);
117
118         MPI_Bcast(my_e_col,g_total_rows,MPI_DOUBLE,g_min_c.index,MPI_COMM_WORLD)
119     ;
120
121         MPI_Bcast(&c_min_value,1,MPI_DOUBLE,g_min_c.index,MPI_COMM_WORLD);
122         if(Phase==1)
123
124             MPI_Bcast(&c_P1_min_value,1,MPI_DOUBLE,g_min_c.index,MPI_COMM_WORLD);
125         }
126     } //τέλος ανταλλαγής στοιχείων επιλογής
127
128     //εάν δεν υπάρχει επιλεγμένη γραμμή, το πρόβλημα
129     //είναι χωρίς όριο. Η Simplex επιτρέφει
130     if(g_l==-1) {
131         return 3;
132     }
133
134     if(DEBUG>=2 && myid==ROOT)
135         printf("\nP%d: Step %d/ph%d: MPI_Simplex: Before Pivotng,
136         g_e=%d@P%d, l_e=%d, g_l=%d,pivot_el=%g
137         ",myid,Phase==1?step_P1:step_P2,Phase,g_e,g_min_c.index,l_e,g_l,my_e_col[g_l]);
138
139     //Αύξηση του αριθμού βημάτων (ανάλογα με Φάση)
```



```
140     Phase==1?step_P1++:step_P2++;
141     //Εκτέλεση του Pivot
142     MPI_Pivot(g_l,l_e,g_e,g_min_c.index,my_e_col,c_min_value,c_P1_min_value);
143
144     if(DEBUG>=3)
145         print_tableau_format_MPI_csv ((int)1);
146
147     if(DEBUG>=1 && myid==ROOT)
148         printf("\nP%d: Step %d/ph%d: MPI_Simplex: P1 Obj Value= %g, P2 Obj
149     Value=%g",myid, Phase==1?step_P1:step_P2,Phase,u_P1,u);
150
151     //έλεγχος ότι η simplex δεν συνεχίζεται επ αόριστον
152     if( (Phase==1 && step_P1>MAX_ITERATIONS) || (Phase==2 &&
153 step_P2>MAX_ITERATIONS) )
154         return 2;
155     }
156     else {
157         //Εκτελείται η ίδια αρχική διαδικασία επιλογής εισερχόμενης μεταβλητής
158         l_e=MPI_ChOOSEENTERINGVariable(&l_min_c);
159         if(numprocs>1)
160
161             MPI_Allreduce(&l_min_c,&g_min_c,1,MPI_DOUBLE_INT,MPI_MINLOC,MPI_COMM_
162 WORLD);
163         else
164             memcpy(&g_min_c,&l_min_c,sizeof(struct obj_Coef));
165         if(myid==g_min_c.index) {
166             if(numprocs>1)
167                 MPI_Bcast(&l_e,1,MPI_INT,myid,MPI_COMM_WORLD);
```



```
169          g_e = l_e;  
170      }  
171      else {  
172          if(numprocs>1)  
173      }  
174      MPI_Bcast(&g_e,1,MPI_INT,g_min_c.index,MPI_COMM_WORLD);  
175  }  
176  } //τέλος ελέγχου για ατέρμονη simplex  
177  }//τέλος loop simplex  
178  
179  //Ο ROOT αναλαμβάνει να επιστρέψει την κατηγορία αποτελέσματος  
180  //Εάν η φάση είναι η 1 τότε επιτυχία επιστρέφεται μόνο εάν έχει  
181  //μηδενιστεί η αντικειμενική συνάρτηση της Φάσης αυτής.  
182  //Εάν η φάση είναι η 2, τότε επιστρέφει επιτυχία  
183  //Όλες οι υπόλοιπες εργασίες, επιστρέφουν επιτυχία.  
184  if(myid==ROOT) {  
185      if(Phase==1) {  
186          if(double_equals_loose(u_P1,0.0))  
187              return 1;  
188          else  
189              return 0;  
190      }  
191      else  
192          return 1;  
193  }  
194  else {  
195      return 1;  
196  }
```



197

198 } //τέλος mpi_simplex

199

200

201

202

203 /*

204 Στην συνάρτηση αυτή, δίνεται από την κάθε εργασία, σαν είσοδος:

205 Η καθολική εξερχόμενη γραμμή (g_l),

206 η καθολική εισερχόμενη στήλη (g_e),

207 η τοπική εισερχόμενη στήλη (l_e),

208 ο επεξεργαστής που έχει την καθολική εισερχόμενη στήλη (g_e_proc),

209 το διάνυσμα της καθ. εισ. στήλης (g_e_col),

210 οι μέγιστες τιμές των παραμέτρων των αντικειμενικών συναρτήσεων, φάσης 1 & 2 στην θέση τις
211 εισερχόμενης μεταβλητής.

212 Το διάνυσμα του δεξιού μέρους του περιορισμού είναι κοινό σε όλες τις εργασίες, όπως επίσης και οι
213 τρέχουσες τιμές των αντικειμενικών συναρτήσεων.

214 Με βάση όλες τις παραπάνω πληροφορίες, η κάθε εργασία εκτελεί το pivot για το κομμάτι που της
215 αναλογεί στον πίνακα των περιορισμών και το διάνυσμα των παραμέτρων των αντικειμενικών
216 συναρτήσεων.

217 */

218 int MPI_Pivot(int g_l, int l_e, int g_e, int g_e_proc, double *g_e_col, double g_e_c_value, double
219 g_e_c_P1_){

220 //Τα i,j χρησιμοποιούνται σαν μετρητές για την τρέχουσα στήλη / γραμμή

221 int i,j;

222 //Προσωρινή μεταβλητή για την αποθήκευση αποτελεσμάτων

223 double temp;

224 //Το διάνυσμα της στήλης που θα αντικαταστήσει την στήλη της εισερχόμενης μεταβλ.

225 double *new_A_e;



```
226          //Οι τιμές των παραμέτρων που θα αντικαταστήσουν τις αντίστοιχες της εισερχόμενης
227      μεταβλ.  

228      double new_c_P1_e,new_c_e;  

229
230      //ο επεξεργαστής που έχει την εισερχόμενη μεταβλητή, θα πρέπει να κάνει τους
231      //υπολογισμούς για την νέα στήλη που είναι της μορφής[ 0 0 .... 1 0 0 0 ...0]
232      if(myid==g_e_proc) {
233          new_A_e = malloc(g_total_rows * sizeof(double)); //the leaving column
234          memset(new_A_e,0.0,g_total_rows * sizeof(double));
235          new_A_e[g_l]=1.0;
236          //καθώς και των παραμέτρων των αντικειμενικών συναρτήσεων στην θέση αυτή
237          if(Phase == 1)
238              new_c_P1_e=(double)0.0; //the new c_P1[e]
239              new_c_e=(double)0.0; //the new obj[e]
240      } //τέλος υπολογισμών επεξεργαστή με εισερχόμενη μεταβλ.
241
242      //Τώρα διαιρούμε την γραμμή της εξερχόμενης μεταβλητής
243      //με το pivot στοιχείο, δηλαδή την τομή εισερχόμενης στήλης
244      //και εξερχόμενης μεταβλητής. Επειδή η εισερχόμενη στήλη δεν
245      //υπάρχει σε κάθε εργασία, έχουμε φροντίσει να την μεταδώσουμε
246      //νωρίτερα και να την περάσουμε σαν παράμετρο σε αυτή εδώ
247      //την συνάρτηση
248
249      //διαιρούμε το δεξί μέρος
250      temp = g_e_col[g_l];
251      if(double_equals (b[g_l],temp)) { b[g_l] = 1.0;}
252      else { b[g_l] /= temp;}
253      //εδώ διαιρούμε το κομμάτι που αντιστοιχεί στην εργασία. Αυτό
```



```
254 //γίνεται εώς εξής: Εάν δεν είμαστε στην Φάση 2 κάνουμε την διαίρεση του στοιχείου
255 //#[στήλη j, γραμμή g_l] με το pivot_element. Εάν είμαστε στην Φάση 2, ελέγχουμε
256 //να μην είναι τεχνητή η μεταβλητή
257 for(j=0;j<l_total_cols;j++) {
258     if(Phase==2
259         kindOf[N[j+BLOCK_LOW(myid,numprocs,g_total_cols)]]==kindofARTIFICIAL) {continue;}
260     else {
261         if(double_equals ( A[j][g_l], g_e_col[g_l])) {
262             A[j][g_l] = 1.0;
263         else
264             A[j][g_l] /= g_e_col[g_l];
265     }
266 } //τέλος διαίρεσης γραμμής
267
268 //διαιρούμε και το στοιχείο της νέας στήλης, φυσικά μόνο στον
269 //επεξεργαστή με την εισερχόμενη μεταβλητή
270 if(myid==g_e_proc)
271     new_A_e[g_l] = (1.0)/g_e_col[g_l]; //make new_A[e][l]
272
273 //τώρα κάνουμε τους μετασχηματισμούς στον πίνακα των
274 //περιορισμών για να μετατρέψουμε την εισερχόμενη στήλη
275 //στην μορφή [0 0 ... 1 ... 0 0]. Κάνουμε πάλι τον έλεγχο για να
276 //μην χρησιμοποιούμε τεχνητές μεταβλητές στην Φάση 2
277 for(i=0;i<g_total_rows;i++){
278     if(Phase==2 && kindOf[B[i]]==kindofARTIFICIAL) {continue;}
279     else {
280         //δεν θέλουμε να πειράξουμε την εξερχόμενη γραμμή
281         if(i!=g_l) {
```



```
282         for(j=0;j<l_total_cols;j++) {  
283             if(Phase==2) &&  
284                 kindOf[N[j+BLOCK_LOW(myid,numprocs,g_total_cols)]]==kindofARTIFICIAL) {continue;}  
285             else {  
286                 temp = g_e_col[i]*A[j][g_l];  
287                 if(double_equals(A[j][i],temp)) {A[j][i] = 0.0;}  
288                 else {A[j][i] -= temp;}  
289             } //τέλος ελέγχου φάσης στήλης  
290         } //τέλος loop στηλών  
291  
292         //Φτιάχνουμε και την νέα στήλη  
293         if(myid==g_e_proc) {  
294             temp = g_e_col[i]*new_A_e[g_l];  
295             if(double_equals(new_A_e[i],temp)) {new_A_e[i] =  
296                 0.0;}  
297             else {new_A_e[i] -= temp;}  
298         }  
299     } //τέλος εάν γραμμή <-> εξερχόμενη  
300 } //τέλος ελέγχου Φάσης & τεχνητής μεταβλητής  
301 } //τέλος loop γραμμών  
302  
303         //Φτιάχνουμε τις παραμέτρους της αντικειμενικής  
304         for(j=0;j<l_total_cols;j++) {  
305             //πρώτα για την Φάση 1  
306             if(Phase == 1){  
307                 temp = g_e_c_P1_value*(A[j][g_l]);  
308                 if(double_equals(c_P1[j],temp)) {c_P1[j] = 0.0;}  
309                 else {c_P1[j] -= temp;}
```



```
310          }
311      //Υστερα για την Φάση 2
312      if(Phase==2                               &&
313      kindOf[N[j+BLOCK_LOW(myid,numprocs,g_total_cols)]]==kindofARTIFICIAL) {continue;}
314      else {
315          temp = g_e_c_value*(A[j][g_l]);
316          if(double_equals(c[j],temp)) {c[j] = 0.0;}
317          else {c[j] -= temp;}
318      }
319  } //τέλος loop στηλών
320
321  //Φτιάχνουμε τις νέες παραμέτρους της αντικειμενικής
322  if(myid==g_e_proc) {
323      if(Phase == 1){
324          temp = g_e_c_P1_value*new_A_e[g_l];
325          if(double_equals(0.0,temp)) {new_c_P1_e = 0.0;}
326          else {new_c_P1_e = -temp;}
327      }
328      temp = g_e_c_value*new_A_e[g_l];
329      if(double_equals(0.0,temp)) {new_c_e = 0.0;}
330      else {new_c_e = -temp;}
331  } //τέλος υπολογισμού νέων παραμέτρων
332
333  //Φτιάχνουμε το δεί μέρος των περιορισμών.
334  //Εκτελείται το ίδιο κομμάτι σε όλους τους επεξεργαστές.
335  //Αυτό το κάνουμε γιατί είτε ένας επεξεργαστής εκτελέσει μία διαδικασί
336  //είτε πολλοί, δεν αυξάνεται η πολυπλοκότητα
337  for(i=0;i<g_total_rows;i++){
```



```
338 //Όχι της εισερχόμενης γραμμής, αφού έχουμε κάνει υπολογισμούς
339     if(i!=g_l) {
340         temp = g_e_col[i]*b[g_l];
341         if(double_equals(b[i],temp)) {b[i] = 0.0;}
342         else {b[i] -= temp;}
343     }
344 } //τέλος υπολογισμού rhs
345
346 //Υπολογίζουμε τις νέες τιμές των αντικειμενικών συναρτήσεων
347 if(Phase == 1){
348     temp = g_e_c_P1_value*b[g_l];
349     if(double_equals(u_P1,temp)) {u_P1 = 0.0;}
350     else {u_P1 -= temp;}
351 }
352 temp = g_e_c_value*b[g_l];
353 if(double_equals(u,temp)) {u = 0.0;}
354 else {u -= temp;}
355
356 //Αντικαθιστούμε την στήλη που φεύγει
357 if(myid==g_e_proc) {
358     memcpy(A[l_e],new_A_e,g_total_rows * sizeof(double));
359     if(Phase==1) { c_P1[l_e] = new_c_P1_e;}
360     c[l_e] = new_c_e;
361 }
362
363 //Ανανεώνουμε το ποιος ανήκει στο Βασικό και Μη-Βασικό Σετ
364 //Αυτό γίνεται σε όλους τους επεξεργαστές
365 int s,abs_e;
```



```
366     abs_e = BLOCK_LOW(g_e_proc,numprocs,g_total_cols) + g_e;
367     s = B[g_l];
368     B[g_l] = N[abs_e];
369     N[abs_e] = s;
370
371 }; //τέλος συνάρτησης mpi_simplex
372
373 /*
374 Στην συνάρτηση αυτή επιλέγεται η τοπική εισερχόμενη μεταβλητή
375 και αποθηκένεται στην δομή l_min_c τύπου obj_Coef.
376 Το σκεπτικό της όλης διαδικασίας είναι να επιλεγεί το μικρότερο
377 εκείνο στοιχείο του διανύσματος των παραμέτων της αντικειμενικής συνάρτησης
378 ,το οποίο επιπλέον είναι αρνητικό και υπάρχει τουλάχιστον ένα έγκυρο
379 στοιχείο εξερχόμενης στήλης.
380 */
381 int MPI_ChooseEnteringVariable( struct obj_Coef *l_min_c ) {
382     int i,j;
383     //χρησιμοποιείται για να δούμε εάν υπάρχει έγκυρο στοιχείο γραμμής στην στήλη
384     int valid;
385     //αποθηκεύουν την προσωρινά μικρότερη εισερχόμενη στήλη
386     int cur_min_idx=-1;
387     double cur_min_val=0.0;
388     //Ποιά αντικειμενική συνάρτηση θα χρησιμοποιήσουμε ανάλογα με την Φάση
389     double val_to_chk=0.0;
390
391     for(j=0;j<l_total_cols;j++) {
392         if( Phase==2
393             kindOf[N[j+BLOCK_LOW(myid,numprocs,g_total_cols)]]==kindofARTIFICIAL) {continue;}
```



```
394     else {  
395         //ανάλογα με την φάση ελέγχουμε την παράμετρο της αντίστοιχης  
396         //αντικειμενικής  
397         if(Phase == 1) val_to_chk = c_P1[j];  
398         else val_to_chk = c[j];  
399  
400         //θεωρούμε ότι δεν υπάρχει έγκυρη εξερχόμενη γραμμή  
401         valid=0;  
402         //εάν η παράμετρος είναι αρνητική και μικρότερη από την τρέχουσα  
403     μικρότερη  
404         if( val_to_chk<0.0 && cur_min_val>val_to_chk){  
405             //τσεκάρουμε για έγκυρη γραμμή  
406             for(i=0;i<g_total_rows;i++) {  
407                 if(Phase==2 && kindOf[B[i]]==kindofARTIFICIAL)  
408             {continue;}  
409             else {  
410                 //εάν το στοιχείο της γραμμής i της στήλης είναι  
411                 //μεγαλύτερο του μηδενός και το δεξί μέρος του  
412                 //περιορισμού είναι ίσο ή μεγαλύτερο του  
413     μηδενός  
414                 //τότε βρήκαμε έγκυρο στοιχείο και η στήλη  
415     μπορεί  
416                 //να είναι η νέα υποψήφια εισερχόμενη  
417                 if(  
418                     (! double_equals(A[j][i],0.0)) &&  
419                     (A[j][i] > 0.0) &&  
420                     (b[i] > 0.0 || (double_equals(b[i],0.0)))  
421                 ) {  
422                     valid=1;
```



```
423                                         break;  
424                                         } //τέλος ελέγχου έγκυρου στοιχείο  
425                                         } //τέλος ελέγχου Φάσης γραμμής  
426                                         } //τέλος loop γραμμής  
427                                         } //τέλος εάν τιμή αντικειμενικής <= 0  
428                                         } //τέλος ελέγχου Φάσης στήλης  
429  
430                                         //Εάν είναι έγκυρο το στοιχείο, κάνουμε την αντικατάσταση  
431                                         if(valid) {  
432                                         cur_min_idx = j;  
433                                         cur_min_val = val_to_chk;  
434                                         }  
435                                         } //τέλος loop στηλών  
436  
437                                         //το τελικό αποτέλεσμα το περνάμε στην δομή  
438                                         l_min_c->index = myid;  
439                                         l_min_c->value = cur_min_val;  
440                                         return cur_min_idx;  
441  
442                                         } //τέλος συνάρτησης  
443  
444  
445                                         /*  
446                                         Στην συνάρτηση αυτή επιλέγεται η εξερχόμενη μεταβλητή ως εξής:  
447                                         Για την εισερχόμενη στήλη του πίνακα περιορισμών ελέγχεται με το  
448                                         ελάχιστο κριτήριο (Δεξί μέρος περιορισμού / Στοιχείο Πίνακα).  
449                                         επιστρέφει δηλαδή την γραμμή που έχει τον ελάχιστο παραπάνω λόγο  
450                                         */
```



```
451     int MPI_ChooseLeavingVariableIndex(int e) {  
452  
453         int i=0;  
454         double cur_theta=0.0, test_theta=0.0;  
455         int cur_min_idx=-1;  
456         int first=TRUE;  
457  
458         //Για κάθε γραμμή  
459         for(i=0;i<g_total_rows;i++) {  
460             if( Phase==2 && kindOf[B[i]]==kindofARTIFICIAL) {continue;}  
461             else {  
462                 //θέλουμε η γραμμή εξερχόμενης μεταβλητής να πληρεί  
463                 //τα κριτήρια  
464                 if(  
465                     ( (!double_equals(A[e][i],0.0)) && (A[e][i] > 0.0)) &&  
466                     (b[i] > 0.0 || double_equals(b[i],0.0))  
467                 ) {  
468                     //υπολογίζουμε τον λόγο  
469                     if(double_equals(b[i],A[e][i]))  
470                         test_theta = 1.0;  
471                     else  
472                         test_theta = b[i]/A[e][i];  
473                     //εάν ο λόγος είναι μικρότερος από τον προηγούμενο  
474                     //ή δεν υπάρχει προηγούμενος λόγος, τότε επιλέγουμε  
475                     //την τρέχουσα μεταβλητή σαν προσωρινά επιλεγμένη  
476                     if( cur_theta > test_theta || first ) {  
477                         first=FALSE;  
478                         cur_theta = test_theta;
```



```
479           cur_min_idx = i;  
480       }  
481   }  
482 } //Τέλος ελέγχου Φάσης  
483 } //τέλος για κάθε γραμμή  
484  
485 //επιστρέφει το index της τρέχουσας μικρότερης γραμμής  
486 return cur_min_idx;  
487 }; //τέλος συνάρτησης
```



5.2.6 mpi_simplex_profiling.c

```
1  /*
2   * File:  mpi_simplex_profiling.c
3   * Author: jkr
4   *
5   * Created on April 03, 2009, 11:38 AM
6   */
7
8  #include "mpi_simplex_v2.h"
9
10 extern int myid, numprocs, DEBUG, MPE_DEBUG;
11 extern int _mpe_load_s, _mpe_load_e;
12 extern int _mpe_choose_local_e_s, _mpe_choose_local_e_f;
13 extern int _mpe_bcast_local_e_s, _mpe_bcast_local_e_f;
14 extern int _mpe_choose_local_l_s, _mpe_choose_local_l_f;
15 extern int _mpe_bcast_local_l_s, _mpe_bcast_local_l_f;
16 extern int _mpe_make_pivot_s, _mpe_make_pivot_f;
17 extern int _mpe_barrier_s, _mpe_barrier_e;
18 extern char *msg_s, *msg_e;
19
20
21 int mpe_go() {
22     _mpe_load_s=MPE_Log_get_event_number();
23     _mpe_load_e=MPE_Log_get_event_number();
24     _mpe_choose_local_e_s=MPE_Log_get_event_number();
25     _mpe_choose_local_e_f=MPE_Log_get_event_number();
```



```
26     _mpe_bcast_local_e_s=MPE_Log_get_event_number();  
27     _mpe_bcast_local_e_f=MPE_Log_get_event_number();  
28     _mpe_choose_local_l_s=MPE_Log_get_event_number();  
29     _mpe_choose_local_l_f=MPE_Log_get_event_number();  
30     _mpe_bcast_local_l_s=MPE_Log_get_event_number();  
31     _mpe_bcast_local_l_f=MPE_Log_get_event_number();  
32     _mpe_make_pivot_s=MPE_Log_get_event_number();  
33     _mpe_make_pivot_f=MPE_Log_get_event_number();  
34     _mpe_barrier_s=MPE_Log_get_event_number();  
35     _mpe_barrier_e=MPE_Log_get_event_number();  
36     if(myid==ROOT) {  
37         MPE_Describe_state(_mpe_load_s,_mpe_load_e,"Load & Distribute MPS problem","pink");  
38         MPE_Describe_state(_mpe_choose_local_e_s,_mpe_choose_local_e_f,"Choose Local e",  
39                           "DarkGreen");  
40         MPE_Describe_state(_mpe_bcast_local_e_s,_mpe_bcast_local_e_f,"Find Global e","gray30");  
41         MPE_Describe_state(_mpe_choose_local_l_s,_mpe_choose_local_l_f,"Choose Local l","gold1");  
42         MPE_Describe_state(_mpe_bcast_local_l_s,_mpe_bcast_local_l_f,"Bcast l","gray36");  
43         MPE_Describe_state(_mpe_make_pivot_s,_mpe_make_pivot_f,"Make Pivot (Computation)","blue");  
44         MPE_Describe_state(_mpe_barrier_s,_mpe_barrier_e,"Barrier","yellow");  
45     }  
46     msg_s =malloc(35*sizeof(char));  
47     msg_e =malloc(35*sizeof(char));  
48 }
```



1 5.2.7 mpi_report.c.

```
1    //Για οικονομία χώρου, δίνονται μόνο οι συναρτήσεις για την καταγραφή των χρόνων
2    //εκτέλεση
3
4    int log_time(int tm, int doBarrier) {
5       if(doBarrier)
6           MPI_Barrier(MPI_COMM_WORLD);
7       times[tm] = MPI_Wtime();
8   }
9
10   int print_times_to_file(char *filename, char *problemname, int P1_res, int P2_res) {
11       FILE *file;
12       int i;
13       if(strcmp(filename,"stdout")==0)
14           file=stdout;
15       else
16           file = fopen(filename,"a");
17       if(file!=NULL) {
18           fprintf(file,"%s;%d;%d;%d;%g;%d;%d;%g;",problemname,numprocs,step_P1,
19           P1_res,u_P1,step_P2,P2_res,u);
20           for(i=1;i<10;i++) {
21               fprintf(file,"%16.16f",times[i]-times[i-1]);
22               fprintf(file,";");
23           }
24           fprintf(file,"\n");
25           fclose(file);
26       }
```



```
27     else
28         printf("\nFailure opening %s",filename);
29 }
```