Παράλληλος προγραμματισμός περιστροφικών αλγορίθμων εξωτερικών σημείων τύπου simplex

ΠΛΟΣΚΑΣ ΝΙΚΟΛΑΟΣ

Διπλωματική Εργασία Μεταπτυχιακού Προγράμματος στην Εφαρμοσμένη Πληροφορική

Κατεύθυνση: Συστήματα Υπολογιστών

Επιβλέπων Καθηγητής: Σαμαράς Νικόλαος

Εξεταστής: Παπαρρίζος Κωνσταντίνος

Πανεπιστήμιο Μακεδονίας

Θεσσαλονίκη

Ιούνιος 2009

Περίληψη

Ο αλγόριθμος simplex είναι συχνά η πιο αποτελεσματική μέθοδος για την επίλυση γραμμικών προβλημάτων (ΓΠ). Ο υπολογιστικός χρόνος των αλγορίθμων simplex επηρεάζεται σημαντικά από την αντιστροφή της βάσης που γίνεται σε κάθε επανάληψη. Η παραλληλοποίηση ενός αλγόριθμου simplex είναι μια μεγάλη πρόκληση, λόγω των πολύ πυκνών πινάκων και της μεγάλης επικοινωνίας. Ο λόγος υπολογισμού προς επικοινωνία είναι πολύ χαμηλός. Οπότε υπάρχει άμεση ανάγκη για προσεκτική επιλογή παράλληλων τεχνικών για την επίτευξη μιας καλής επιτάχυνσης.

Ο σκοπός αυτής της διπλωματικής εργασίας είναι να παρουσιάσει μια παράλληλη υλοποίηση του πρωτεύοντα αλγόριθμου εξωτερικών σημείων. Σε αυτήν την προσέγγιση, δόθηκε ιδιαίτερη έμφαση στην παραλληλοποίηση της βάσης. Ο πίνακας που περιέχει τη βάση κατανέμεται σε πολλούς υπολογιστές και η αντίστροφη υπολογίζεται πιο γρήγορα σε μεγάλα ΓΠ. Εκτός από την παράλληλη υλοποίηση, η εργασία αυτή παρουσιάζει και τα πειραματικά αποτελέσματα που δείχνουν την επιτάχυνση ανάμεσα στην ακολουθιακή και την παράλληλη εκδοχή σε τυχαία μεγάλα πυκνά βέλτιστα ΓΠ. Η υπολογιστική μελέτη πραγματοποιήθηκε με τη βοήθεια του περιβάλλοντος Matlab.

Λέξεις κλειδιά: Υπολογιστική μελέτη, Αλγόριθμος εξωτερικών σημείων, Γραμμικός Προγραμματισμός, Παράλληλος προγραμματισμός, Matlab.

Abstract

The simplex method is frequently the most efficient method of solving linear programming (LP) problems. The computation time of simplex algorithms relies on the basis inverse that occurs in each iteration. Parallelizing simplex algorithms is one of the most challenging problems. Because of very dense matrices and very heavy communication, the ratio of computation to communication is extremely low. It becomes necessary to carefully select parallel techniques, partitioning patterns and communication optimization to achieve a speedup.

The aim of this paper is to present a parallel implementation of the primal exterior point algorithm. In this approach the basis inverse is computed in parallel. The matrix that holds the basis is distributed among different workers and the computation is performed faster in large-scale LP problems. Apart from the parallel implementation, this paper presents a computational study that shows the speedup among the serial and parallel version in large-scale randomly generated full dense LP problems. The computationally study was performed with the help of the matrix programming language, Matlab.

Index Terms: Computational Study, Exterior Point Algorithm, Linear Programming, Parallel Programming, Matlab.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Αντικείμενο της εργασίας

Ο αλγόριθμος simplex έχει χαρακτηριστεί ως ένας από τους σημαντικότερους αλγορίθμους του 20^{ου} αιώνα. Στις περισσότερες των περιπτώσεων είναι η καλύτερη επιλογή για την επίλυση γραμμικών προβλημάτων (ΓΠ). Οι υπολογιστικές του ανάγκες σε πόρους έχουν οδηγήσει τους ερευνητές να εστιάσουν στην εύρεση μεθόδων παραλληλοποίησης του.

Η έρευνα στον τομέα αυτό έχει εστιαστεί στον αναθεωρημένο αλγόριθμο, επειδή εκμεταλλεύεται την αραιότητα που υπάρχει στα περισσότερα γραμμικά προβλήματα. Ωστόσο, δεν υπάρχουν πολλές παράλληλες ή κατανεμημένες υλοποιήσεις για τον αναθεωρημένο αλγόριθμο simplex που συγκλίνουν καλά. Οι Hall και McKinnon [21] δούλεψαν πάνω σε παράλληλες αναθεωρημένες δομές, ενώ οι Thomadakis και Liu [51] εργάστηκαν πάνω στον κλασικό αλγόριθμο. Οι Eckstein et al [15] έδειξαν ότι κάθε επανάληψη στον παράλληλο αναθεωρημένο αλγόριθμο απαιτεί σημαντικά περισσότερο χρόνο από ότι στην παραλληλοποιημένη εκδοχή με tableau.

Ο Stunkel [49] υλοποίησε τόσο τον κλασικό αλγόριθμο για αραιά προβλήματα όσο και τον αναθεωρημένο και πέτυχε μια επιτάχυνση 8 και 12 για μικρά προβλήματα από το σύνολο Netlib [18]. Οι Cvetanovic et al [10] ανέφεραν μια επιτάχυνση ίση με 12 όταν έλυσαν δύο μικρά προβλήματα χρησιμοποιώντας τον κλασικό αλγόριθμο. Πρόσφατα, οι Badr et al [3] παρουσίασαν μια υλοποίηση σε οχτώ επεξεργαστές, επιτυγχάνοντας ένα speedup της τάξης του 5 όταν λύθηκαν μικρά τυχαία πυκνά γραμμικά προβλήματα. Στην εργασία [48] περιγράφεται μια παραλληλοποίηση του αναθεωρημένου αλγόριθμου simplex και μιας εκδοχής που κάνει χρήση της LU παραγοντοποίησης για την αντιστροφή της βάσης. Στην πρώτη εκδοχή υπάρχει μια γραμμική επιτάχυνση, ενώ στη δεύτερη περίπτωση η παράλληλη εκδοχή δεν είναι πάντα καλύτερη από την ακολουθιακή.

Μετά από πειράματα, παρατηρήθηκε ότι ο κλασικός αλγόριθμος simplex υλοποιείται συνήθως χρησιμοποιώντας πυκνούς πίνακες. Ωστόσο, οι Lentini et al [31] υλοποίησαν μια παράλληλη υλοποίηση του κλασικού αλγορίθμου simplex αποθηκεύοντας το tableau ως αραιό πίνακα. Όταν έλυσαν μέτριου μεγέθους προβλήματα από το σύνολο Netlib σε τέσσερις υπολογιστές, αποκόμισαν μια επιτάχυνση μεταξύ 0.5 και 2.7.

Στον αναθεωρημένο αλγόριθμο simplex χρησιμοποιούνται πυκνές δομές για τη διατήρηση του B^{-1} στη μνήμη. Ωστόσο, η Shu [47] διαπίστωσε ότι το B^{-1} είναι πλήρης πίνακας και παραλληλοποίησε τον αναθεωρημένο αλγόριθμο simplex διατηρώντας το B^{-1} ως αραιό πίνακα. Στα πειραματικά της αποτελέσματα ανέφερε ότι πέτυχε μια επιτάχυνση της τάξεως του 17 σε μικρά προβλήματα από το σύνολο Netlib.

Η παρούσα εργασία ασχολείται με την παραλληλοποίηση του αλγορίθμου εξωτερικών σημείων τύπου simplex. Πιο συγκεκριμένα, η εργασία αυτή εστιάζει στην παραλληλοποίηση της ανανέωσης της βάσης που είναι και η πιο χρονοβόρα λειτουργία του αλγόριθμου simplex.

Ο προγραμματισμός τόσο της ακολουθιακής όσο και της παράλληλης έκδοσης του αλγορίθμου έγινε στο προγραμματιστικό περιβάλλον Matlab R2009a. Στις τελευταίες εκδόσεις αυτού του περιβάλλοντος περιλαμβάνεται μια πολύ χρήσιμη εργαλειοθήκη για τον παράλληλο προγραμματισμό διεργασιών. Μια αναλυτική περιγραφή της εργαλειοθήκης αυτής παρουσιάζεται στο παράρτημα της παρούσας εργασίας.

1.2 Διάρθρωση της εργασίας

Η υπόλοιπη εργασία είναι χωρισμένη σε 7 κεφάλαια. Στο κεφάλαιο 2 παρουσιάζεται αναλυτικά ο αλγόριθμος εξωτερικών σημείων τύπου simplex. Στο κεφάλαιο αυτό περιέχεται τόσο το μαθηματικό υπόβαθρο του αλγορίθμου όσο και ένα παράδειγμα για την καλύτερη κατανόηση του αλγορίθμου. Στο κεφάλαιο 3 περιλαμβάνεται μια εισαγωγή στις έννοιες του παράλληλου προγραμματισμού και τις πιο ευρέως χρησιμοποιούμενες μετρικές για υπολογιστικά πειράματα με παράλληλους αλγορίθμους.

Στο κεφάλαιο 4 περιγράφονται οι τεχνικές παράλληλου πολλαπλασιασμού πινάκων που υπάρχουν στη βιβλιογραφία, ενώ στο κεφάλαιο 5 περιγράφεται η παραλληλοποίηση της ανανέωσης βάσης, όπως υλοποιήθηκε στην παρούσα εργασία. Το κεφάλαιο 6 περιλαμβάνει τα αποτελέσματα της υπολογιστικής μελέτης που διεξήχθη για τη σύγκριση της ακολουθιακής και της παράλληλης εκδοχής του

αλγορίθμου. Τέλος, στο Κεφάλαιο 7 παρουσιάζονται πιθανές μελλοντικές επεκτάσεις και βελτιώσεις της παράλληλης υλοποίησης, σύμφωνα με τα συμπεράσματα που προέκυψαν από την εργασία.

1.3 Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου και Επίκουρο Καθηγητή του τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας, κ. Νικόλαο Σαμαρά τόσο για την πολύτιμη βοήθειά του κατά τη διάρκεια εκπόνησης της εργασίας αυτής όσο και για τη γενικότερη καθοδήγησή του όλα αυτά τα χρόνια που συνεργαζόμαστε. Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου που με ανέχτηκαν και με στήριξαν κατά τη διάρκεια όλης αυτής της προσπάθειας.

ΚΕΦΑΛΑΙΟ 2

ΑΛΓΟΡΙΘΜΟΙ ΕΞΩΤΕΡΙΚΩΝ ΣΗΜΕΙΩΝ

2.1 Ιστορική αναδρομή

Ο αλγόριθμος simplex έχει χαρακτηριστεί ως ένας από τους σημαντικότερους αλγορίθμους του $20^{ου}$ αιώνα. Η δημιουργία του κατέστη απαραίτητη από την ανάγκη για την επίλυση προβλημάτων βελτιστοποίησης. Ο αλγόριθμος simplex προτάθηκε από τον George Dantzig το 1947 [12], ο οποίος είναι γνωστός ως ο πατέρας του γραμμικού προγραμματισμού και ένας από τους σημαντικότερους μαθηματικούς του $20^{ου}$ αιώνα. Η πρώτη παρουσίαση του νέου αποτελέσματος δεν έπεισε για την αποτελεσματικότητά του. Δεν έγινε, όμως, το ίδιο στη δεύτερη παρουσίαση. Η επιστημονική κοινότητα πείστηκε τώρα ότι πρόκειται για κάποιο πολύ σημαντικό αποτέλεσμα. Ο αλγόριθμος δημοσιεύτηκε για πρώτη φορά το 1949 [11].

Μετά την πρώτη δημοσίευση του σημαντικού αυτού αποτελέσματος οι δρόμοι για την ανάπτυξη μιας νέας επιστήμης άνοιξαν διάπλατα. Η βιβλιογραφία των δεκαετιών του ΄50 και του ΄60 βρίθει από δημοσιεύσεις σημαντικότατων αποτελεσμάτων συνοδευμένων πολλές φορές με υπολογιστικά αποτελέσματα τα οποία αποδείκνυαν την πρακτική αποτελεσματικότητα του αλγορίθμου simplex. Όμως, ο ρυθμός αύξησης του μεγέθους των πρακτικών προβλημάτων ήταν μεγαλύτερος όχι μόνο του ρυθμού αύξησης της ταχύτητας των ηλεκτρονικών υπολογιστών αλλά και του ρυθμού παραγωγής νέων σημαντικών αποτελεσμάτων. Ο αλγόριθμος simplex άρχισε να αντιμετωπίζει προβλήματα χρόνου στην επίλυση μεγάλων προβλημάτων. Η αρχική ευφορία για την πρακτική αποτελεσματικότητα άρχισε σιγά σιγά να σβήνει. Το σημαντικότερο κτύπημα ήρθε το 1971 όταν οι μαθηματικοί Klee και Minty [28] απάντησαν αρνητικά στο πιο σημαντικό ανοικτό πρόβλημα της εποχής εκείνης, του χαρακτηρισμού της υπολογιστικής πολυπλοκότητας της χειρότερης περίπτωσης. Απέδειξαν ότι ο αλγόριθμος simplex είναι από θεωρητική άποψη το χειρότερο που μπορούσε να συμβεί, ένας εκθετικός αλγόριθμος. Αργότερα, όμως, ο Borgwardt [6], [7] απέδειξε ότι ο αναμενόμενος αριθμός των επαναλήψεων για την επίλυση ενός γραμμικού προβλήματος από έναν

αλγόριθμο τύπου simplex είναι πολυωνυμικός όταν χρησιμοποιείται σε πραγματικά προβλήματα.

Το αποτέλεσμα των Klee και Minty έστρεψε την έρευνα σε άλλες κατευθύνσεις. Ιδιαίτερα η ανακάλυψη πολυωνυμικού αλγορίθμου για το γραμμικό πρόβλημα έγινε το νέο διάσημο ανοικτό πρόβλημα, το οποίο ταλαιπώρησε την επιστημονική κοινότητα για περίπου 10 χρόνια. Το 1979 ο Ρώσος επιστήμονας L. Khachian [27] τροποποίησε έναν αλγόριθμο ο οποίος αναπτύχθηκε από άλλους Ρώσους επιστήμονες, I. I. Dikin [13] και N. Z. Shor [46], και απέδειξε ότι η τροποποίηση του είναι πολυωνυμική. Ήταν ένα από τα πιο σημαντικά αποτελέσματα όχι μόνο του μαθηματικού προγραμματισμού αλλά και της θεωρητικής πληροφορικής και των μαθηματικών γενικότερα.

Ο νέος αλγόριθμος, ο ελλιψοειδής ή ρώσικος αλγόριθμος, ο πρώτος πολυωνυμικός αλγόριθμος για το γραμμικό πρόβλημα, επεφύλαξε μια απίστευτη έκπληξη στους ερευνητές. Παρ' ότι ήταν ότι καλύτερο από θεωρητικής απόψεως έμελλε να μείνει γνωστός στην ιστορία και για τη μνημειώδη πρακτική αναποτελεσματικότητά του. Οι συγκρίσεις με τον αλγόριθμο simplex ήταν απογοητευτικές, γιατί δούλευε πάντοτε στη χειρότερη περίπτωσή του. Παρά την απογοητευτική πρακτική του εμφάνιση, ο ελλειψοειδής αλγόριθμος παραμένει ένα σημαντικότατο θεωρητικό αποτέλεσμα.

Οι επιστήμονες στράφηκαν στην ανακάλυψη πολυωνυμικών αλγορίθμων οι οποίοι όμως έπρεπε να είναι αποτελεσματικοί και στην πράξη. Ένας αλγόριθμος με όλα αυτά τα χαρακτηριστικά ανακαλύφθηκε από τον Karmarkar το 1983 [26]. Ο Karmarkar ανακοίνωσε στο συνέδριο της IEEE εκείνης της χρονιάς, ότι ο αλγόριθμος εσωτερικών σημείων που είχε κατασκευάσει είναι περίπου 50 φορές ταχύτερος του αλγόριθμου simplex. Οι μετέπειτα υπολογιστικές μελέτες απέδειξαν ότι οι ισχυρισμοί του Karmarkar ήταν διογκωμένοι.

Από τότε που το κλασικό άρθρο του Karmarkar δημοσιεύθηκε, πολλά άρθρα έχουν γραφεί για τις μεθόδους εσωτερικών σημείων [19, 20, 32]. Όπως δείχνουν οι θεωρητικές ιδιότητές τους, οι μέθοδοι εσωτερικών σημείων λειτουργούν καλά για τα πρακτικά προβλήματα στην πραγματική ζωή και ξεπερνούν τη μέθοδο simplex για τα πολύ μεγάλης κλίμακας προβλήματα [4].

Παρόλα αυτά, η ανάπτυξη περιστροφικών αλγορίθμων (pivoting algorithms), οι οποίοι να είναι ταχύτεροι του simplex παραμένει ένα θέμα με μεγάλο επιστημονικό ενδιαφέρον. Ένας τρόπος για την βελτίωση της υπολογιστικής συμπεριφοράς των αλγορίθμων τύπου simplex είναι η μετακίνηση σε μη-γειτονικές (non-adjacent) βελτιώνουσες κορυφές. Ένας τέτοιος αλγόριθμος ονομάζεται αλγόριθμος simplex εξωτερικών σημείων (exterior point simplex algorithm), ο οποίος θα αναλυθεί και αναλυτικότερα στο κεφάλαιο αυτό.

2.2 Βασικές έννοιες - Ορισμοί

Έστω το ακόλουθο πρόγραμμα γραμμικού προγραμματισμού:

$$\begin{aligned} \min \quad & c^T x \\ \text{μ.π.} \quad & Ax = b \qquad (\Pi 2.1) \\ & x >= b \end{aligned}$$

όπου $c, x \in R^n$, $b \in R^m$ και $A \in R^{mxn}$. Ο εκθέτης T σημαίνει αναστροφή. Το δυϊκό πρόβλημα του (Π2.1) είναι:

$$\begin{aligned} \max \quad & w^T b \\ \text{μ.π.} \quad & A^T w + s = 0 \qquad (\Pi 2.2) \\ & s >= b \end{aligned}$$

όπου $w \in R^n$ είναι οι δυϊκές μεταβλητές και $s \in R^n$ είναι το διάνυσμα των δυϊκών χαλαρών μεταβλητών.

Υποθέτουμε ότι η μήτρα A είναι πλήρους βαθμού. Η συνθήκη αυτή υποδηλώνει ότι η μήτρα A είναι αντιστρέψιμη. Επίσης, συνηθίζεται κατά τη μελέτη αλγορίθμων γραμμικού προγραμματισμού να υιοθετούνται οι ακόλουθες υποθέσεις:

1. Η εφικτή περιοχή δεν είναι κενή αλλά υπάρχει.

2. Κάθε κορυφή της εφικτής περιοχής προσδιορίζεται από ακριβώς n περιορισμούς. Αυτό σημαίνει ότι το γραμμικό πρόβλημα είναι μη εκφυλισμένο.

3. Η βέλτιστη λύση είναι μοναδική.

4. Μια εύκολα υπολογισμένη αρχική εφικτή λύση είναι διαθέσιμη.

Διαμερίζοντας τη μήτρα A ως A = (B, N) το πρόβλημα (Π2.1) γράφεται ως εξής:

$$\begin{aligned} \min \quad & c_B^T x_B + c_N^T x_N \\ \text{μ.π.} \quad & Bx_B + Nx_N = 0 \qquad (\Pi 2.3) \\ & x_B, \; x_N \geq 0 \end{aligned}$$

Η μήτρα B είναι μια τετραγωνική (m x m) αντιστρέψιμη υπομήτρα της A. Οι μεταβλητές οι οποίες ανήκουν στη B ονομάζονται βασικές ενώ αυτές που ανήκουν στη N μη-βασικές. Η λύση η οποία αντιστοιχεί στη βάση B είναι $x^T = (x_B, x_N)^T$ όπου $x_B = B^{-1}b$ και $x_N = 0$. Μια βάση B είναι πρωτεύοντος εφικτή αν ισχύει x >= 0, ενώ είναι δυικά εφικτή αν:

$$s = c - A^T w \geq 0 \qquad (2.1)$$

όπου $w^T = c^T B^{-1}$ είναι οι πολλαπλασιαστές simplex. Σε περίπτωση που ισχύει η 2.1 τότε οι πολλαπλασιαστές αποτελούν μία λύση στο πρόβλημα (Π2.2). Αν ισχύει $x \not\geq 0$ τότε η λύση ονομάζεται μη-εφικτή. Η μη-εφικτή βασική λύση αποτελείται από σημεία τα οποία είναι εξωτερικά σε σχέση με την εφικτή περιοχή. Μια βάση η οποία είναι και πρωτεύοντος και δυικά εφικτή ονομάζεται βέλτιστη. Στην περίπτωση αυτή οι λύσεις x και w είναι βέλτιστες λύσεις στα προβλήματα (Π2.1) και (Π2.2).

2.3 Βασικά χαρακτηριστικά αλγορίθμων εξωτερικών σημείων

Οι προσπάθειες κατασκευής αλγορίθμων που να είναι αποτελεσματικότεροι των εφικτών αλγόριθμων simplex δεν απέδωσαν για 35 χρόνια περίπου. Στη δεκαετία του '80 αναπτύχθηκαν οι αλγόριθμοι εσωτερικών σημείων οι οποίο είναι πράγματι ταχύτεροι των αλγόριθμων simplex σε προβλήματα μεγάλου μεγέθους. Παρ' όλα

αυτά, το ενδιαφέρον για τον αλγόριθμο simplex δεν μειώθηκε κυρίως εξαιτίας της προσαρμοστικότητάς του στην επίλυση πιο γενικών προβλημάτων. Πρόσφατα αναπτύχθηκαν νέοι αλγόριθμοι τύπου simplex. Οι νέοι αλγόριθμοι φαίνεται ότι υπερτερούν σημαντικά του κλασικού αλγορίθμου. Το χαρακτηριστικό τους γνώρισμα είναι ότι δημιουργούν δύο δρόμους που οδηγούν στη βέλτιστη λύση. Ένα άλλο χαρακτηριστικό τους γνώρισμα είναι ότι δημιουργούν εξωτερικά σημεία. Ένας τέτοιος αλγόριθμος ονομάζεται αλγόριθμος simplex εξωτερικών σημείων (exterior point simplex algorithm). Στη συνέχεια ο αλγόριθμος εξωτερικών σημείων θα αναφέρεται για λόγους συντομίας ως EPSA.

Ο πρώτος EPSA αναπτύχθηκε από τον Paparrizos [37] για το πρόβλημα χωροθέτησης (assignment problem). Στη συνέχεια ο ίδιος ερευνητής ανέπτυξε ένα γενικό EPSA για γραμμικά προβλήματα [38]. Ανεξάρτητα, οι Anstreicher, Terlaky [2] ανέπτυξαν έναν παρόμοιο πρωτεύων αλγόριθμο ο οποίος επιλύει ακολουθία από υποπροβλήματα. Ένα κοινό χαρακτηριστικό όλων σχεδόν των τύπου simplex αλγορίθμων είναι ότι μπορούν να ερμηνευτούν ως μια διαδικασία που ακολουθεί τύπου simplex μονοπάτια (simplex paths) τα οποία καταλήγουν στη βέλτιστη λύση. Οι αλγόριθμοι εξωτερικών σημείων διαφέρουν ριζικά από τον πρωτεύοντα αλγόριθμο simplex επειδή οι βασικές τους λύσεις δεν είναι εφικτές.

Σε κάθε επανάληψη ο πρωτεύον αλγόριθμος simplex, ο οποίος για λόγους συντομίας θα αναφέρεται ως PSA εναλλάσσει μια στήλη του B με μια γειτονική στήλη του N κατασκευάζοντας έτσι μια νέα βάση B. Γεωμετρικά αυτό σημαίνει ότι κινείται κατά μήκος των ακμών του πολυέδρου $P = \{x \mid Ax \leq b, x \geq 0\}$. Ένας τέτοιος δρόμος είναι γνωστός ως δρόμος simplex. Το πλήθος των διαφορετικών κορυφών και κατά συνέπεια όλων των βάσεων που μπορούν να κατασκευαστούν δίνεται από τη σχέση

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Από την άλλη μεριά οι αλγόριθμοι εξωτερικών σημείων δημιουργούν δυο δρόμους προς τη βέλτιστη λύση. Ο ένας δρόμος είναι μη εφικτός και ο άλλος είναι εφικτός. Για αυτό το λόγο δε χρειάζεται να προχωρούν εξετάζοντας τη μία μετά την άλλη ακμή κατά μήκος του πολυέδρου. Επομένως οι αλγόριθμοι εξωτερικών σημείων μπορούν να ακολουθήσουν συντομότερους δρόμους παρακάμπτοντας τα σύνορα της εφικτής περιοχής. Το σημαντικό για τους αλγόριθμους εξωτερικών σημείων είναι ότι δεν πρέπει σε καμία περίπτωση να χάσουν την επαφή με την εφικτή περιοχή. Σε αντίθετη περίπτωση η εύρεση της βέλτιστης κορυφής είναι πολύ δύσκολη. Για τη διατήρηση της επαφής με την εφικτή περιοχή φροντίζει ο δεύτερος δρόμος ο οποίος αποτελείται από εφικτά σημεία. Αυτός ο δρόμος δεν είναι όμως τύπου simplex.

Οι Paparrizos et al [41] έδειξαν ότι η γεωμετρία των αλγορίθμων εξωτερικών σημείων καθιστά φανερό ότι είναι ταχύτεροι από τον κλασικό αλγόριθμο simplex, γεγονός το οποίο επαληθεύτηκε σε πρωταρχικά υπολογιστικά αποτελέσματα σε

ειδικά δομημένα τυχαία γραμμικά προβλήματα, [1], [14]. Το συμπέρασμα αυτό επιβεβαιώθηκε από τους Paparrizos et al [41], οι οποίοι με μια εκτενή υπολογιστική μελέτη σε τυχαία αραιά και πυκνά γραμμικά προβλήματα έδειξαν ότι ο πρωτεύων EPSA είναι πάνω από 10 φορές ταχύτερος του κλασικού αλγορίθμου simplex. Άλλοι αποτελεσματικοί αλγόριθμοι τύπου simplex έχουν αναπτυχθεί από τους Anstreicher, Terlaky [2], Murty, Fathi [35] και Sherali et al [45]. Ο πιο αποτελεσματικός αλγόριθμος εξωτερικών σημείων έχει υλοποιηθεί από τον Samaras [44]. Μια πλήρης αναφορά όλων των κανόνων περιστροφής για το γραμμικό προγραμματισμό μπορεί να βρεθεί στην αναφορά [50].

Οι αλγόριθμοι εξωτερικών σημείων κατασκευάζουν μια ακολουθία βασικών λύσεων. Έστω B η τρέχουσα βάση. Στη συνέχεια γίνεται ο έλεγχος βελτιστότητας. Αν το παρών σημείο δεν είναι βέλτιστο τότε γίνεται προσπάθεια να βρεθούν οι δείκτες k και l, οι οποίοι αντιστοιχούν στην εξερχόμενη και εισερχόμενη μεταβλητή αντίστοιχα. Αν δεν μπορεί να προσδιοριστεί εξερχόμενη μεταβλητή τότε το πρόβλημα (Π2.1) είναι απεριόριστο. Ακολουθεί η αναλυτική περιγραφή του πρωτεύοντος αλγορίθμου εξωτερικών σημείων.

2.4 Ο πρωτεύων αλγόριθμος εξωτερικών σημείων

Ο πρωτεύων αλγόριθμος εξωτερικών σημείων θα περιγραφεί στη συνέχεια στο γραμμικό πρόβλημα (Π2.4). Ο αλγόριθμος ξεκινά με μια βασική εφικτή διαμέριση (B, N). Στη συνέχεια υπολογίζονται τα σύνολα δεικτών:

$$P = \{j \in N : s_j < 0\} \tag{2.2}$$

και

$$Q = \{j \in N : s_j \in 0\} \tag{2.3}$$

Αν $P = \varnothing$ τότε η τρέχουσα βάση B και η αντίστοιχη λύση $x^T = (x_B, x_N)$ είναι βέλτιστη στο πρόβλημα (Π2.1). Διαφορετικά, προσδιορίζεται η εξερχόμενη μεταβλητή $x_{B[r]} = x_k$ από τη σχέση

$$a = \frac{x_{B[r]}}{-d_{B[r]}} = \min\left\{\frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0\right\} \tag{2.4}$$

όπου d είναι μια βελτιώνουσα κατεύθυνση. Η κατεύθυνση αυτή κατασκευάζεται κατά τέτοιο τρόπο έτσι ώστε η ακτίνα $\{x + td : t > 0\}$ να τέμνει την εφικτή περιοχή του προβλήματος (Π2.4). Ο συμβολισμός d_B συμβαίνει ότι παίρνουμε εκείνες τις

συνιστώσες της d που αντιστοιχούν στις βασικές μεταβλητές. Η d_B υπολογίζεται από την ακόλουθη σχέση

$$d_B = -\sum_{j \in P} h_j \tag{2.5}$$

όπου $h_j = B^{-1}A_{.j}$.

Ο προσδιορισμός της εξερχόμενης μεταβλητής από τη σχέση (2.4) είναι ακριβώς ο ίδιος με αυτόν του αλγορίθμου simplex. Όπως και ο simplex, έτσι και ο EPSA σταματά αν $d_B \geq 0$. Στην περίπτωση αυτή το πρόβλημα (Π2.4) είναι απεριόριστο. Στη συνέχεια προσδιορίζεται η εισερχόμενη μεταβλητή x_l. Υπολογίζονται πρώτα οι σχέσεις:

$$\theta_2 = -\frac{-s_Q}{H_{rQ}} = \min\left\{\frac{-s_j}{H_{rj}} : H_{rj} > 0 \ \wedge \ j \in P\right\} \tag{2.6}$$

και

$$\theta_2 = -\frac{-s_Q}{H_{rQ}} = \min\left\{\frac{-s_j}{H_{rj}} : H_{rj} < 0 \wedge j \in Q\right\} \tag{2.7}$$

Αν ισχύει $\theta 1 \leq \theta 2$ τότε l = p, διαφορετικά l = q. Η μη βασική μεταβλητή x_l εισέρχεται στη βάση. Στη συνέχεια περιγράφεται ο πρωτεύων EPSA σε μορφή βημάτων.

Υπενθυμίζεται ότι η αντίστροφη της βάσης υπολογίζεται από τη σχέση:

$$\overline{B}^{-1} = E^{-1}B^{-1} \tag{2.8}$$

όπου E^{-1} είναι η αντίστροφη της eta-μήτρας, E.

Αναθεωρημένος Πρωτεύων Αλγόριθμος Εξωτερικών Σημείων

Βήμα 0. *(Αρχικοποίηση).* Ξεκίνα με μια εφικτή βασική διαμέριση (B, N). Υπολόγισε τη μήτρα B^{-1} και τα διανύσματα x_B, w και s_N. Το s_N υπολογίζεται από τη σχέση (2.1). Βρες τα σύνολα δεικτών P και Q από τις σχέσεις (2.2) και (2.3) αντίστοιχα. Επέλεξε αυθαίρετα ένα διάνυσμα $\lambda = (\lambda_1, \lambda_2, ..., \lambda_{|P|}) > 0$ και υπολόγισε το s_0 χρησιμοποιώντας τη σχέση:

$$s_0 = \sum_{j \in P} \lambda_j s_j$$

και την κατεύθυνση d_B από τη σχέση (2.5).

Βήμα 1. *(Έλεγχος τερματισμού).*

i. (Έλεγχος βελτιστότητας). Αν $P = \varnothing$, STOP. Το πρόβλημα (Π2.1) είναι βέλτιστο.

ii. (Επιλογή της εξερχόμενης μεταβλητής). Αν $d_B \geq 0$, STOP. Αν $s_0 = 0$ το πρόβλημα (Π2.1) είναι βέλτιστο. Αν $s_0 < 0$ το πρόβλημα (Π2.1) είναι απεριόριστο. Διαφορετικά, επέλεξε την εξερχόμενη μεταβλητή $x_{B[r]} = x_k$ χρησιμοποιώντας τη σχέση (2.4).

Βήμα 2. *(Επιλογή της εισερχόμενης μεταβλητής).* Υπολόγισε τα διανύσματα:

$$H_{rP} = (B^{-1})_r.A_P \text{ και } H_{rQ} = (B^{-1})_r.A_Q$$

Βρες επίσης τους λόγους θ_1 και θ_2 από τις σχέσεις (2.6) και (2.7) αντίστοιχα. Προσδιόρισε τους δείκτες t_1 και t_2 τέτοιους ώστε $P[t_1] = p$ και $Q[t_2] = q$. Αν $\theta_1 \leq \theta_2$, θέσε $l = q$. Η μη βασική μεταβλητή x_l εισέρχεται στη βάση.

Βήμα 3. *(Περιστροφή).* Θέσε $B[r] = l$. Αν $\theta 1 \leq \theta 2$, θέσε $P \leftarrow P\backslash\{l\}$ και $Q \leftarrow Q \cup \{k\}$. Διαφορετικά, θέσε $Q[t_2] = k$. Χρησιμοποιώντας τη νέα διαμέριση (B, N) όπου N = (P, Q), υπολόγισε τη μήτρα B^{-1} από τη σχέση (2.8) και ανανέωσε τα διανύσματα x_B, w και s_N. Επίσης ανανέωσε την νέα κατεύθυνση $\overline{d}_B$ χρησιμοποιώντας τη σχέση

$$\overline{d}_B = E^{-1} d_B \qquad (2.9)$$

Αν $l \in P$ θέσε $d_{B[r]} \leftarrow d_{B[r]} + \lambda_l$. Πήγαινε στο Βήμα 1.

Η απόδειξη της ορθότητας του EPSA μπορεί να βρεθεί στις αναφορές [39] και [41].

Παράδειγμα

Να λυθεί με τον αλγόριθμο εξωτερικών σημείων το γραμμικό πρόβλημα:

$$\begin{array}{lrcrcrcrcr} \min & -2x_1 & - & 3x_2 & + & x_3 & + & 4x_4 & & \\ \mu.\pi. & x_1 & + & 4x_2 & - & 2x_3 & - & x_4 & \leq & 8 \\ & x_1 & + & 3x_2 & - & 4x_3 & - & x_4 & \leq & 7 \\ & x_1 & + & 3x_2 & + & 2x_3 & - & x_4 & \leq & 10 \end{array}$$

$$x_j \geq 0, \quad (j = 1,2,3,4)$$

Προσθέτουμε τις χαλαρές μεταβλητές x_5, x_6 και x_7. Η βάση στην οποία οι χαλαρές μεταβλητές είναι όλες βασικές είναι εφικτή. Επομένως, μπορούμε να ξεκινήσουμε την επίλυση με τον αλγόριθμο εξωτερικών σημείων χρησιμοποιώντας σαν βάση ξεκινήματος B = [5, 6, 7] και N = [1, 2, 3, 4].

Τα δεδομένα του προβλήματος μετά την εισαγωγή των χαλαρών μεταβλητών είναι:

$$c^T = (-2,-3,1,4,0,0,0)$$

$$A = \begin{bmatrix} 1 & 4 & -2 & -1 & 1 & 0 & 0 \\ 1 & 3 & -4 & -1 & 0 & 1 & 0 \\ 1 & 3 & 2 & -1 & 0 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 8 \\ 7 \\ 10 \end{bmatrix}$$

Επανάληψη 1

Βήμα 0. (Αρχικοποίηση)

Χρησιμοποιώντας τη βάση $B = [a_5, a_6, a_7] = [e_1, e_2, e_3] = B^{-1}$ υπολογίζουμε εύκολα και βρίσκουμε ότι:

$$x_B = B^{-1} b = b$$

$$w^T = (c_B)^T B^{-1} = (0, 0, 0)\, B^{-1} = (0, 0, 0)$$

και

$$z = w^T b = 0$$

Ξεκινάμε με τα σύνολα των βασικών και μη βασικών δεικτών:

$$B = [5, 6, 7], \;\; N = [1, 2, 3, 4]$$

Υπολογίζουμε εύκολα τις χαλαρές δυικές μεταβλητές:

$$(s_N)^T = (c_N)^T - w^T N = (c_N)^T = (c_1, c_2, c_3, c_4) = (-2, -3, 1, 4)$$

Επομένως είναι:

$$P = \{ j \in N : s_j < 0 \} = [1,2] \quad \text{και} \quad Q=N \sim P=[3,4]$$

Είναι $h_j = a_j$ και επομένως θέτοντας $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = \lambda_4 = 0$ βρίσκουμε ότι $h_{00} = s_1 + s_2 = (-2) + (-3) = -5$ και $h_0 = a_1 + a_2$. Με τα στοιχεία αυτά κατασκευάζουμε τα στοιχεία του Πίνακα 1. Στο αριστερό του Πίνακα 1 είναι η στήλη $[-s_0, (h_0)^T]^T$.

	5	0	0	0	0	1	
5	5	1	0	0	8	-2	$\leftarrow r = 1$
6	4	0	1	0	7	-4	
7	4	0	0	1	10	2	

Πίνακας 1

Βήμα 1. (Έλεγχος τερματισμού)

Είναι $P = \varnothing$. Ο αλγόριθμος δε σταματά. Είναι $h_0 > 0$. Δε μπορούμε να συμπεράνουμε ότι το πρόβλημα είναι απεριόριστο. Προχωρούμε στον προσδιορισμό της εξερχόμενης μεταβλητής. Είναι:

$$a = \min\left\{\frac{x_5}{h_{10}}, \frac{x_6}{h_{20}}, \frac{x_7}{h_{30}}\right\} = \min\left\{\frac{8}{5}, \frac{7}{4}, \frac{10}{4}\right\} = \frac{8}{5} = \frac{x_5}{h_{10}}$$

Η εξερχόμενη μεταβλητή είναι η x_5 με r = l και k = B[l] = 5.

Βήμα 2. (Προσδιορισμός εισερχόμενης μεταβλητής)

Υπολογίζουμε τα διανύσματα:

$$H_{1P} = (B^{-1})_{1.}P \quad \text{και} \quad H_{1Q} = (B^{-1})_{1.}Q$$

Όπου:

$$H_{1P} = (h_{11}, h_{12}) = (1, 4)$$

$$H_{1Q} = (h_{13}, h_{14}) = (-2, -1)$$

Επομένως, έχουμε:

$$\theta_1 = \min\left\{\frac{-s_1}{h_{11}}, \frac{-s_2}{h_{12}}\right\} = \min\left\{\frac{2}{1}, \frac{3}{4}\right\} = \frac{3}{4} = \frac{-s_1}{h_{11}}$$

$$\theta_2 = \min\left\{\frac{-s_3}{h_{13}}, \frac{-s_4}{h_{14}}\right\} = \min\left\{\frac{-1}{-2}, \frac{-4}{-1}\right\} = \frac{1}{2} = \frac{-s_3}{h_{13}}$$

και p = 1, q = 3. Επειδή $\theta_1 > \theta_2$ θέτουμε l = q = 3. Είναι δε t = l αφού Q[l] = q = 3. Εισερχόμενη μεταβλητή είναι η x_3.

Βήμα 3. (Περιστροφή)

Επειδή είναι $\theta_1 > \theta_2$ θέτουμε B[r] = B[l] = l = 3 και Q[t] = Q[l] = k = 5. Τα νέα σύνολα B, P και Q είναι:

B = [3, 6, 7], P = [1, 2] και Q = [5, 4]

Υπολογίζουμε το h_3. Είναι $h_3 = B^{-1} a_3 = a_3$, όπως ακριβώς φαίνεται στον Πίνακα 0. Η περιστροφή στο στοιχείο h_{13} δίνει τα ανανεωμένα δεδομένα του Πίνακα 2. Επειδή είναι $\theta_1 > \theta_2$ η στήλη h_0 και το στοιχείο h_{00} ανανεώνονται όπως οι υπόλοιπες στήλες του Πίνακα 1. Προσέξετε ότι τώρα είναι $x_3 = -4 < 0$. Επομένως το τρέχον βασικό σημείο δεν είναι εφικτό, είναι ένα εξωτερικό σημείο. Οι νέες δυικές χαλαρές μεταβλητές είναι:

$$(s_N)^T = (s_1, s_2, s_4, s_r) = (c_N)^T - (w^T N) = (-2, -3, 4, 0) - (-1/2, 0, 0)N$$

$$= (-3/2, -1, ½, 0)$$

	5/2	-1/2	0	0	-4	**-1**
3	-5/2	-1/2	0	0	-4	-2
6	-6	-2	1	0	-9	-3
7	9	1	0	1	18	7 ← $r = 3$

Πίνακας 2

Επανάληψη 2

Βήμα 1. (Ελεγχος τερματισμού)

Δεν είναι P = ∅. Δεν είναι $h_0 \leq 0$. Επομένως, προχωρούμε στον προσδιορισμό της εξερχόμενης μεταβλητής. Το μοναδικό θετικό στοιχείο του h_0 είναι το $h_{30} = 9$. Επομένως, εξερχόμενη μεταβλητή είναι η $x_{B[3]} = x_9$ με r = 3 και k = 9.

Βήμα 2. (Προσδιορισμός εισερχόμενης μεταβλητής)

$$H_{3P} = (h_{31}, h_{32}) = (B^{-1})_{3P} = (2, 7)$$

$$H_{3Q} = (h_{35}, h_{34}) = (B^{-1})_{3Q} = (1, -2)$$

Επομένως, έχουμε:

$$\theta_1 = \min\left\{\frac{-s_1}{h_{31}}, \frac{-s_2}{h_{32}}\right\} = \min\left\{\frac{\frac{3}{2}}{2}, \frac{1}{7}\right\} = \frac{1}{7} = \frac{-s_2}{h_{32}}$$

$$\theta_2 = \min\left\{\frac{-s_5}{h_{35}}, x\right\} = \min\left\{\frac{-\frac{7}{2}}{-2}, x\right\} = \frac{7}{4} = \frac{-s_5}{h_{35}}$$

Είναι p = 2 και q = 2. Επειδή είναι $\theta_1 < \theta_2$ θέτουμε l = p = 2 και διαπιστώνουμε ότι t = 2 (γιατί Q[l] = 2). Εισερχόμενη μεταβλητή είναι η x_2.

Βήμα 3. (Περιστροφή)

Θέτουμε B[r] = B[3] = l = 2. Άρα είναι B = [3, 6, 2]. Είναι P = P ~ {2} = [l]. Τέλος, είναι Q = Q ∪ {k} = [4, 5, 7]. Για το διάνυσμα περιστροφής βρίσκουμε ότι $(h_2)^T = ((B^{-1})_2 Q_2)^T = (-2, -5, 7)$. Η περιστροφή στο στοιχείο $h_{32} = 7$ του Πίνακα 2 δίνει τα νέα στοιχεία του Πίνακα 3.

	17/4	-9/14	0	-1/7	-46/7
3	1/14	-3/14	0	2/7	8/7
6	3/7	-9/7	1	5/7	5/7
2	2/7	1/7	0	1/7	18/7

17/14
-1/14
3/7
2/7

← $r = 3$

Πίνακας 3

Τώρα είναι l = p ∈ Q. Γι' αυτό μετά την περιστροφή στον Πίνακα 2, αφαιρέσαμε το $\lambda_2 = 1$ από το στοιχείο $h_{30} = 9/7$ και έτσι στον Πίνακα 3 γράφτηκε το στοιχείο $h_{30} = 9/7 - 1 = 2/7$. Τώρα είναι:

$$(s_N)^T = (s_1, s_4, s_5, s_7) = \left(\frac{17}{14}, -\frac{45}{14}, -\frac{9}{14}, -\frac{1}{7}\right)$$

Επανάληψη 3

Είναι $P = [l] \neq \varnothing$. Επομένως, το τρέχον σημείο δεν είναι βέλτιστο (δεν είναι $h_0 \leq 0$). Ο έλεγχος ελαχίστου λόγου δίνει:

$$a = \min\left\{\frac{x_3}{h_{10}}, \frac{x_6}{h_{20}}, \frac{x_2}{h_{30}}\right\} = \min\left\{\frac{\frac{8}{7}}{\frac{1}{14}}, \frac{\frac{5}{7}}{\frac{3}{7}}, \frac{\frac{1}{7}}{\frac{2}{7}}\right\} = \frac{1}{2} = \frac{x_{B[3]}}{h_{30}}$$

Επομένως, εξερχόμενη μεταβλητή είναι η x_2 με k = 2 και r = 3. Είναι:

$$H_{3P} = h_{31} = (B^{-1})_{3.}\, a_1 = \frac{2}{7}$$

$$H_{3Q} = [h_{35}, h_{34}, h_{37}] = (B^{-1})_{3.}\, Q = \left(\frac{1}{7}, \frac{-2}{7}, \frac{1}{7}\right)$$

και επομένως

$$\theta_1 = \frac{-s_1}{h_{31}} = \frac{\frac{17}{14}}{\frac{2}{7}} = \frac{17}{4}$$

$$\theta_2 = \frac{-s_4}{h_{34}} = \frac{-\frac{45}{14}}{-\frac{2}{7}} = \frac{45}{4}$$

Άρα p = 1 και q = 4. Είναι $\theta_1 < \theta_2$. Επομένως l = p = 1 και t = 1. Είναι:

$$(h_1)^T = (B^{-1}\, a_1)^T = (h_0)^T = \left(\frac{1}{14}, \frac{3}{7}, \frac{2}{7}\right)$$

και

$$s_1 = c_1 - w^T\, a_1 = -\frac{17}{14}$$

Η περιστροφή στο στοιχείο $h_{31} = 2/7$ του Πίνακα 3 δίνει τον Πίνακα 4, ο οποίος διαπιστώνεται στην επόμενη επανάληψη ότι είναι βέλτιστος.

	0	-5/4	0	-3/4	-245/14
3	0	-1/4	0	1/4	1/2
6	0	-3/2	1	1/2	0
1	0	1/2	0	1/2	9

Πίνακας 4

Επανάληψη 4

P = ∅, άρα η βέλτιστη τιμή της αντικειμενικής συνάρτησης είναι -245/14, ενώ το βέλτιστο σημείο είναι εκφυλισμένο. Οι χαλαρές δυικές μεταβλητές είναι $s_2 = 17/4$, $s_4 = 2$, $s_5 = 5/4$ και $s_7 = ¾$.

Από τις τιμές των s_j προκύπτει ότι το τρέχον σημείο είναι πράγματι βέλτιστο.

ΚΕΦΑΛΑΙΟ 3

ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

3.1 Εισαγωγή

Η απαίτηση για γρηγορότερους και πιο αποτελεσματικούς υπολογισμούς σε επιστημονικές και εμπορικές εφαρμογές έχει αυξηθεί σημαντικά τα τελευταία χρόνια. Φυσικοί περιορισμοί και υψηλά κόστη καθιστούν αδύνατη την αύξηση της ταχύτητας των επεξεργαστών πέρα από συγκεκριμένα όρια. Για να ξεπεραστούν αυτές οι δυσκολίες νέες αρχιτεκτονικές έχουν εμφανιστεί εισάγοντας την παράλληλη επεξεργασία. Στις μέρες μας διαφορετικοί τύποι μηχανών με υψηλές επιδόσεις είναι διαθέσιμες και στηρίζονται στη συγκέντρωση πολλών επεξεργαστών μαζί. Ανάμεσα σε αυτά τα συστήματα τα σημαντικότερα είναι [30]:

- διανυσματικοί πολυεπεξεργαστές
- μαζικά παράλληλοι επεξεργαστές (εκατοντάδες έως χιλιάδες επεξεργαστών συνδεδεμένων μεταξύ τους με κοινή κατανεμημένη μνήμη) και
- υπολογιστές σε δίκτυο (σταθμοί εργασίας συνδεδεμένοι με ένα μεσαίας ή υψηλών ταχύτητας δίκτυο, οι οποίοι εργάζονται σαν μια παράλληλη εικονική μηχανή υψηλών επιδόσεων).

Ο παραλληλισμός οδηγεί στην ανάγκη για νέους αλγορίθμους, ειδικά σχεδιασμένους ώστε να εκμεταλλεύονται σύγχρονες διαδικασίες, λόγω του ότι πολλές φορές οι βέλτιστες παράλληλες λύσεις δεν επιτυγχάνονται με απλές τροποποιήσεις των ακολουθιακών αλγορίθμων [17, 29]. Σε αντίθεση με τα ακολουθιακά προγράμματα, οι παράλληλοι αλγόριθμοι εξαρτώνται ισχυρά από την αρχιτεκτονική υπολογιστών για τους οποίους έχουν σχεδιαστεί. Προγράμματα τα οποία υλοποιούν αυτούς τους αλγόριθμους έχουν αναπτυχθεί και εκτελεστεί σε παράλληλα υπολογιστικά περιβάλλοντα τα οποία αποτελούνται από έναν παράλληλο υπολογιστή και το σχετικό λογισμικό [9, 51]. Το λογισμικό αποτελείται από

παράλληλα προγραμματιστικά εργαλεία, εργαλεία απόδοσης και σχετικούς μεταγλωττιστές καθώς και ορισμένες βιβλιοθήκες που αναπτύχθηκαν για την επίλυση ειδικών κατηγοριών προβλημάτων [5, 33, 36, 42]. Τα προγραμματιστικά εργαλεία τυπικά παρέχουν υποστήριξη για την ανάπτυξη προγραμμάτων, τα οποία αποτελούνται από διεργασίες που ανταλλάσουν πληροφορίες κατά τη διάρκεια των εκτελέσεων.

Η απόδοση των συστημάτων που αναπτύσσονται με προγραμματιστικά εργαλεία και η διαθέσιμη υποστήριξη για επικοινωνία και συγχρονισμό μεταξύ των τμημάτων του παράλληλου προγράμματος, είναι μερικά από τα σημαντικά σενάρια του παράλληλου προγραμματισμού. Το προγραμματιστικό εργαλείο θα έπρεπε επίσης να είναι αντάξιο του προγραμματιστικού μοντέλου σύμφωνα με το οποίο υλοποιείται ο παράλληλος αλγόριθμος.

Η ευκολία και ο χρόνος για την ανάπτυξη ενός προγράμματος είναι πολλές φορές εξαιρετικά κρίσιμα. Τα εργαλεία μπορούν να βοηθήσουν τον προγραμματιστή ώστε να καταλάβει και να βελτιώσει παράλληλα προγράμματα με το να παρακολουθεί την εκτέλεση ενός παράλληλου προγράμματος και την παραγωγή δεδομένων τα οποία αναλύονται ώστε να εντοπιστούν και να γίνουν αντιληπτές παροχές χαμηλής απόδοσης.

Η διόρθωση παράλληλων προγραμμάτων είναι δύσκολο πρόβλημα. Παράλληλα εργαλεία διόρθωσης έχουν κατασκευαστεί και για παράλληλα προγράμματα και είναι συνήθως ιδιαίτερα αποτελεσματικά.

3.2 Παραλληλισμός δεδομένων

Η πλέον ισχυρή και ευρέως διαδεδομένη τεχνική για την οργάνωση παράλληλων προγραμμάτων είναι ο παραλληλισμός δεδομένων, που σε γενικές γραμμές μπορεί να καθορισθεί ως η ταυτόχρονη εκτέλεση της ίδιας λειτουργίας σε κάθε συστατικό μιας δομής δεδομένων. Η μεγάλη εφαρμογή των παράλληλων υπολογισμών στα επιστημονικά προβλήματα βασίζεται στη χρήση μεγάλων πολυδιάστατων πινάκων και άλλων μεγάλων δομών δεδομένων. Ο κεντρικός πυρήνας των περισσοτέρων παραδοσιακών αλγορίθμων αποτελείται από σειρές φωλιασμένων βρόχων, που εκτελούν πολύπλοκους χειρισμούς πινάκων για να παράγουν κάποιο αριθμητικό αποτέλεσμα.

Σήμερα ωστόσο, τα περισσότερα παράλληλα προγράμματα δημιουργούνται με την αναδιοργάνωση αυτών των σειριακών αλγόριθμων ώστε οι φωλιασμένοι βρόχοι να εκτελούνται παράλληλα. Συνήθως αυτό σημαίνει ότι εξαιτίας της παράλληλης εφαρμογής της ίδιας λειτουργίας σε διαφορετικά τμήματα του πίνακα δεδομένων, υπάρχει ο παραλληλισμός και στην παραγωγή των αποτελεσμάτων του προγράμματος.

Ο παραλληλισμός δεδομένων με απλά λόγια είναι ότι η δομή του παραλληλισμού ανταποκρίνεται στη δομή των δεδομένων. Σε ορισμένες περιπτώσεις, κάθε στοιχειώδης μονάδα δεδομένων υφίσταται παράλληλη επεξεργασία. Σε άλλες περιπτώσεις, η δομή των δεδομένων τμηματοποιείται σε ομάδες ανεξάρτητων μονάδων δεδομένων, όπου κάθε ομάδα υφίσταται παράλληλη επεξεργασία. Όλες αυτές οι περιοχές εφαρμογών χρησιμοποιούν δομές που περιέχουν μεγάλους αριθμούς παρόμοιων μονάδων δεδομένων.

Για να μπορέσουμε να επεξεργαστούμε αυτά τα παράλληλα προγράμματα θα πρέπει να υπάρχει και το σωστό περιβάλλον. Το περιβάλλον που μπορούμε να χρησιμοποιήσουμε είναι πολλοί επεξεργαστές (είτε διαφορετικού τύπου είτε του ίδιου τύπου) οι οποίοι μπορεί να βρίσκονται σε σχετικά μικρή απόσταση είτε να τους χωρίζει απόσταση και οι οποίοι μπορούν να ανταλλάσσουν πληροφορίες και δεδομένα μέσω δικτύων.

Η σημαντικότητα του παράλληλου υπολογισμού έγκειται στο ότι μπορούμε να διεκπεραιώσουμε πολύπλοκους και μεγάλους υπολογισμούς που ένας σειριακός υπολογιστής δεν θα μπορούσε να διεκπεραιώσει. Επίσης η σημαντικότητα του παράλληλου υπολογισμού φαίνεται και στο ότι μεγάλες εταιρίες αρχιτεκτονικής (όπως η Intel και η AMD) έχουν ήδη δώσει σε μαζική παραγωγή, συστήματα που χρησιμοποιούν την παραλληλοποίηση των δεδομένων προς επεξεργασία. Παράδειγμα αυτού είναι τα διάφορα supercomputers που έχουν δημιουργήσει πολλές εταιρίες και ινστιτούτα για εργασίες όπως πρόβλεψη καιρικών συνθηκών ή επεξεργασία εικόνας. Επίσης, το χαμηλό πλέον κόστος των διάφορων παράλληλων συστημάτων κάνει και οικονομικά συμφέρουσα τη χρήση παράλληλου υπολογισμού.

Για να μπορέσουμε να εισάγουμε μια εφαρμογή σε τέτοια παράλληλα συστήματα θα πρέπει πρώτα να σχεδιάσουμε παράλληλους αλγόριθμους. Οι παράλληλοι αλγόριθμοι ορίζονται ως μια ακολουθία εντολών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που οδηγούν στην επιτυχή επίλυση ενός προβλήματος και που είναι ειδικά σχεδιασμένος για παράλληλους υπολογιστές. Για να εκμεταλλευτούμε την υπολογιστική δύναμη του παράλληλου υπολογισμού, πρέπει έχουμε σχεδιάσει αποδοτικούς παράλληλους αλγορίθμους και απαιτείται:

- αποδοτική συνεργασία μεταξύ επεξεργαστών και
- σωστή και προσεκτική διαχείριση των διαθέσιμων πόρων

Η εφαρμογή για να μπορέσει να τρέξει σε ένα παράλληλο σύστημα πρέπει ο προγραμματιστής να χρησιμοποιήσει παράλληλο προγραμματισμό (parallel programming) που ορίζεται ως η χρήση παράλληλης γλώσσας προγραμματισμού προκειμένου το πρόγραμμα να εκτελεστεί σε μία παράλληλη μηχανή. Τα τρία βασικότερα μοντέλα/πρωτόκολλα παράλληλου προγραμματισμού είναι ο παραλληλισμός δεδομένων, ο κοινός χώρος διευθύνσεων και η μεταβίβαση μηνυμάτων. 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.

3.3 Παράλληλες αρχιτεκτονικές και Μοντέλα

Η ύπαρξη περισσοτέρων από ενός επεξεργαστή περιπλέκει την μοντελοποίηση και το σχεδιασμό αλγόριθμων. Υπάρχουν πολλά διαφορετικά μοντέλα παράλληλου υπολογισμού και κάθε μοντέλο προγραμματίζεται με διαφορετικό τρόπο.

Ένας σχεδιαστής παράλληλων αλγορίθμων θα πρέπει να έχει ένα συγκεκριμένο μοντέλο υπόψη του όταν σχεδιάζει ένα αλγόριθμο. Κάθε υπολογιστής λειτουργεί εκτελώντας οδηγίες (instructions) πάνω σε δεδομένα (data).

Μοντέλα Υπολογισμού:

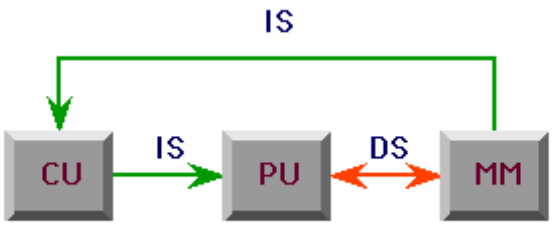
1. SISD: Single Instruction stream, Single Data stream (σειριακό)

2. MISD: Multiple Instruction stream, Single Data stream

3. SIMD: Single Instruction stream, Multiple Data stream

4. MIMD: Multiple Instruction stream, Multiple Data stream

Στο μοντέλο SISD υπάρχει μόνο ένας επεξεργαστής και είναι μόνο για σειριακούς υπολογισμούς όπως φαίνεται και στο σχήμα 3.1 υπάρχει μια μονάδα ελέγχου, ένας επεξεργαστής και μια τοπική μνήμη.

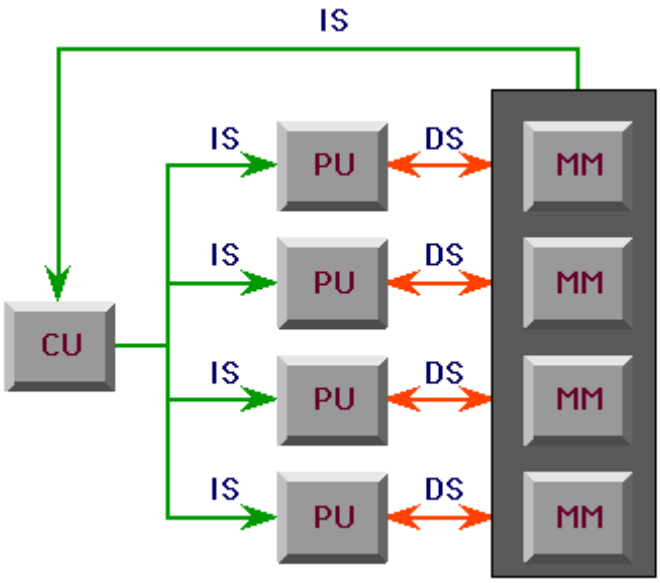


Σχήμα 3.1: SISD

MISD: Έχουμε N επεξεργαστές. Κάθε ένας έχει την δική του μονάδα ελέγχου (CU) και μοιράζονται μια κοινόχρηστη μνήμη όπου βρίσκονται τα δεδομένα. Σε κάθε βήμα οι επεξεργαστές μπορούν να εκτελέσουν διαφορετική εντολή στο ίδιο στοιχείο.

MIMD: Έχουμε N επεξεργαστές N ρεύματα εντολών και N ρεύματα δεδομένων. Οι επεξεργαστές μπορούν να εκτελέσουν διαφορετικά προγράμματα πάνω σε διαφορετικά δεδομένα για να επιλύουν διαφορετικά υποπροβλήματα.

SIMD: Έχουμε N επεξεργαστές. Ο κάθε ένας έχει την δική του τοπική μνήμη και διαφορετικό ρεύμα δεδομένων (Σχήμα 3.2).



Σχήμα 3.2: SIMD

Οι επεξεργαστές δουλεύουν συγχρονισμένα και σε κάθε βήμα (clock - cycle) όλοι οι επεξεργαστές εκτελούν την ίδια εντολή πάνω σε διαφορετικό δεδομένο.

Υπάρχουν δυο τρόποι επικοινωνίας:

1. Μέσω κοινόχρηστης μνήμης (PRAM) ή

2. Μέσω Δικτύου Αλληλοσύνδεσης

3.4 Αξιολόγηση της απόδοσης

Είναι ύψιστης σημασίας στον παράλληλο υπολογισμό να αξιολογηθεί το κέρδος που θα επιτευχθεί στην ταχύτητα από τη παράλληλη λειτουργία N επεξεργαστών. Για το λόγο αυτό εισάγεται μια παράμετρος αποκαλούμενη λόγος επιτάχυνσης. Ας υποθέσουμε ότι τρέχουμε ένα πρόγραμμα χρησιμοποιώντας έναν επεξεργαστή και παίρνει το χρόνο T(1) για να τρέξει. Ας υποθέσουμε ότι το πρόγραμμα είναι γραμμένο για να εκμεταλλεύεται το διαθέσιμο αριθμό των επεξεργαστών. Τρέχουμε έπειτα το πρόγραμμα χρησιμοποιώντας τους N επεξεργαστές και αυτό παίρνει το χρόνο T(N) για να τρέξει. Κατόπιν, καλούμε επιτάχυνση υπολογισμών το λόγο:

$$S = \frac{T(1)}{T(n)}$$

και απόδοση το λόγο:

$$E = \frac{S}{N}$$

Τώρα, ας υποθέσουμε ότι απομονώνουμε στο πρόγραμμα το αυστηρά σειριακό μέρος (δηλαδή το μέρος που δεν μπορεί να παραλληλιστεί) από το παράλληλο μέρος. Καλούμε B το ποσοστό του αυστηρά σειριακού μέρους του προγράμματος ($B \leq 1$). Κατόπιν, το αυστηρά σειριακά μέρος του προγράμματος εκτελείται σε χρόνο B*T(1). Το υπόλοιπο μέρος, (1-B), ανατίθεται σε ένα σύνολο από N επεξεργαστές. Εάν υποθέσουμε ότι κάθε επεξεργαστής απαιτεί το 1/N του χρόνου ενός επεξεργαστή που δουλεύει μόνος του, τότε το αυστηρά παράλληλο μέρος εκτελείται σε χρόνο ((1−B)*T(1))/N. Κατόπιν υπολογισμών, παίρνουμε τον τύπο:

$$S = \frac{N}{(B * N) + (1 - B)}$$

Αυτός είναι ο νόμος του Amdahl, που θέτει όρια στην επιτάχυνση που επιτυγχάνεται από την αύξηση του αριθμού των επεξεργαστών. Για να καταλάβουμε το νόμο του Amdahl, καταφεύγουμε σε μια καμπύλη επιτάχυνσης. Μία καμπύλη επιτάχυνσης είναι ένα γράφημα με τον αριθμό των επεξεργαστών στον άξονα των X, και την επιτάχυνση S στον άξονα των Y. Η καλύτερη ταχύτητα είναι όταν B = 0 (δηλαδή ολόκληρο το πρόγραμμα μπορεί να παραλληλιστεί). Αυτό θα έδινε μια ευθεία με κλίση 45 μοιρών αφού S = N. Όταν το B είναι σταθερό, ο νόμος του Amdahl δημιουργεί μια καμπύλη επιτάχυνσης που είναι λογαριθμική και παραμένει κάτω από τη γραμμή S = N. Αυτός ο νόμος δείχνει ότι είναι πράγματι ο αλγόριθμος

και όχι ο αριθμός επεξεργαστών που περιορίζει την επιτάχυνση. Επίσης καθώς καμπύλη αρχίζει για να τείνει προς ευθεία, η απόδοση μειώνεται δραστικά.

Ο νόμος του Amdahl είναι χρήσιμος για να αξιολογηθεί η δυνατότητα ενός αλγορίθμου να παραλληλιστεί, αλλά δεν είναι ένα πρακτικό εργαλείο. Σε πραγματικές εφαρμογές, η απόδοση εξαρτάται από έναν αριθμό βασικών παραγόντων, όπως το εύρος ζώνης του δικτύου στην περίπτωση των αρχιτεκτονικών κατανεμημένης μνήμης και το φορτίο των επεξεργαστών σε περιβάλλοντα πολλών χρηστών. Αυτό επιβεβαιώνεται όταν μετακινούμαστε από εξειδικευμένες παράλληλες μηχανές σε ευρεία περιβάλλοντα, όπου οι κατανεμημένες αρχιτεκτονικές αποτελούνται από ετερογενή συστήματα. Για να εκμεταλλευτούμε στο έπακρον τη διαθέσιμη υπολογιστική ισχύ, οι διεργασίες πρέπει να αποσταλούν στους επεξεργαστές λαμβάνοντας υπόψη την τρέχουσα κατάσταση των πόρων τους και την αντιστοιχία τους με τις απαιτήσεις της εργασίας. Υπάρχει ένας μεγάλος αριθμός εργαλείων που σχετίζονται με τον προγραμματισμό των εργασιών, όπως τα Platform’s Load Sharing Facility (LSF), Altair’s Portable Batch System (PBS) και το Condor. Τα εργαλεία προγραμματισμού των εργασιών δέχονται, σαν είσοδο, αρχεία που απαριθμούν τις εργασίες που υποβάλλονται και τις απαιτήσεις τους. Μόλις είναι γνωστές οι απαιτήσεις της εργασίας, επιθεωρούν την κατάσταση των συνδεδεμένων μηχανών και προγραμματίζουν τις εργασίες χρησιμοποιώντας κριτήρια με βάση την εξισορρόπηση του φορτίου. Κατά αυτόν τον τρόπο οι χρήστες, μπορούν να υποβάλουν τις εργασίες τους και στην συνέχεια να επικοινωνήσουν με το εργαλείο προγραμματισμού των εργασιών ώστε να ρωτήσουν την κατάστασή της.

ΚΕΦΑΛΑΙΟ 4

ΠΑΡΑΛΛΗΛΕΣ ΤΕΧΝΙΚΕΣ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ ΠΙΝΑΚΩΝ

Ο πολλαπλασιασμός δύο πινάκων, A και B, παράγει τον πίνακα C, του οποίου τα στοιχεία, $c_{i,j}$ ($0 \le i < n$, $0 \le j < m$), μπορούν να υπολογιστούν από τον εξής τύπο:

$$c_{i,\,j} = \sum_{k=0}^{l-1} a_{i,\,k}\ b_{k,\,j}$$

Όπου ο A είναι ένας n x l πίνακας και ο B είναι ένας l x m πίνακας. Κάθε στοιχείο της i-οστής γραμμής του A πολλαπλασιάζεται με ένα στοιχείο της j-οστής στήλης του B, και τα γινόμενα προστίθενται μεταξύ τους για να υπολογιστεί το στοιχείο του πίνακα C που βρίσκεται στη γραμμή i και τη στήλη j. Ο αριθμός των στηλών του A πρέπει να είναι ίσος με τον αριθμό των γραμμών του B.

4.1 Ακολουθιακός κώδικας

Ας υποθέσουμε ότι οι πίνακες A και B είναι τετράγωνοι διαστάσεων n x n. Από τον ορισμό που δόθηκε παραπάνω για τον πολλαπλασιασμό πινάκων, ο ακολουθιακός κώδικας για τον υπολογισμό του πολλαπλασιασμού πινάκων είναι ο εξής:

```
for (i = 0; i < n; i++)
        for (j = 0; j <n; j++) {
                C[i][j] = 0;
                for (k = 0; k < n; k++)
                        C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
```

Ο αλγόριθμος απαιτεί n^3 πολλαπλασιασμούς και n^3 προσθέσεις, που οδηγεί σε μια πολυπλοκότητα $O(n^3)$.

4.2 Παράλληλος κώδικας

Με μία ματιά στον ακολουθιακό κώδικα μπορούμε να διακρίνουμε ότι ο υπολογισμός σε κάθε επανάληψη των δύο εξωτερικών βρόγχων δεν εξαρτάται από καμία άλλη επανάληψη και κάθε στιγμιότυπο του εσωτερικού βρόγχου μπορεί να εκτελεστεί παράλληλα. Οπότε διαθέτοντας p = n επεξεργαστές (και n x n πίνακες), μπορούμε να περιμένουμε πολυπλοκότητα χρόνου $O(n^2)$. Επίσης, είναι εξίσου εύκολο να έχουμε μια πολυπλοκότητα O(n) αν διαθέσουμε $p = n^2$ επεξεργαστές, όπου ένα στοιχείο του A και ένα του B αναθέτονται σε ένα επεξεργαστή. Είναι, επίσης, εφικτό σύμφωνα με την [34] ότι διαθέτοντας $p = n^3$ επεξεργαστές μπορούμε να επιτύχουμε το κατώτερο όριο χρονικής πολυπλοκότητας για την παράλληλη εκδοχή του πολλαπλασιασμού πινάκων, που είναι O(log n).

Για τον παράλληλο πολλαπλασιασμό πινάκων έχουν προταθεί διάφορες μέθοδοι, οι οποίες μπορούν να βρεθούν αναλυτικότερα στο []. Σε αυτό το σημείο θα παρουσιαστούν τα κυριότερα σημεία τους, ενώ στο επόμενο κεφάλαιο θα αναλυθεί αναλυτικότερα η μέθοδος που υιοθετήθηκε στην εργασία αυτή.

4.2.1 Απευθείας υλοποίηση χωρίς υποπίνακες

Ένας τρόπος για να υλοποιηθεί ο πολλαπλασιασμός πινάκων είναι να αναθέσουμε σε κάθε επεξεργαστή να υπολογίσει κάθε στοιχείο του C. Τότε θα χρειαζόταν n^2 επεξεργαστές. Μία γραμμή στοιχείων από τον πίνακα A και μία στήλη από τον B χρειάζονται σε κάθε επεξεργαστή. Χρησιμοποιώντας μια master – slave προσέγγιση, αυτά τα στοιχεία μπορούν να σταλούν από τον επεξεργαστή που έχει το ρόλο του master στους επιλεγμένους slaves επεξεργαστές. Μερικά στοιχεία των πινάκων θα χρειαστεί να σταλούν περισσότερο από μία φορές.

Χρόνος Επικοινωνίας

Έστω ότι οι πίνακες A και B είναι διαστάσεων n x n. Στέλνοντας ξεχωριστά μηνύματα σε καθέναν από τους n^2 slaves επεξεργαστές, κάθε επεξεργαστής θα πάρει μία γραμμή και μία στήλη στοιχείων, δηλαδή 2n στοιχεία. Επίσης, κάθε επεξεργαστής θα επιστρέψει στον master επεξεργαστή ένα στοιχείο από τον πίνακα C. Συνολικά ο χρόνος επικοινωνίας είναι:

$$t_{comm} = n^2(t_{startup} + 2nt_{data}) + n^2(t_{startup} + t_{data}) = n^2(2t_{startup} + (2n + 1)t_{data})$$

Όπως προαναφέρθηκε μερικά στοιχεία των πινάκων A και B στέλνονται σε παραπάνω από έναν slave επεξεργαστή. Οπότε μια καλύτερη προσέγγιση είναι να μεταδώσουμε σε όλους τους slaves επεξεργαστές (broadcast) τους πίνακες A και B στην αρχή. Με αυτήν την προσέγγιση προκύπτει μείωση στο χρόνο επικοινωνίας:

$$t_{comm} = (t_{startup} + n^2 t_{data}) + n^2(t_{startup} + t_{data})$$

Χρόνος Υπολογισμού

Ο υπολογισμός λαμβάνει χώρο μόνο στους slaves επεξεργαστές. Καθένας από αυτούς διενεργεί παράλληλα n πολλαπλασιασμούς και n προσθέσεις, δηλαδή:

$$t_{comp} = 2n$$

4.2.2 Απευθείας υλοποίηση με υποπίνακες

Σε κάθε μέθοδο μπορούμε να χρησιμοποιήσουμε υποπίνακες για τη μείωση του αριθμού των επεξεργαστών. Ας χρησιμοποιήσουμε m x m υποπίνακες και s = n/m, δηλαδή έχουμε s^2 υποπίνακες και s^2 επεξεργαστές.

Χρόνος Επικοινωνίας

Καθένας από τους s^2 επεξεργαστές πρέπει να πάρει μία γραμμή και στήλη από τους υποπίνακες, αποτελούμενη από m^2 στοιχεία η καθεμιά. Επιπλέον, κάθε slave επεξεργαστής πρέπει να επιστρέψει στο master επεξεργαστή έναν υποπίνακα του C αποτελούμενο από m^2 στοιχεία. Οπότε, ο συνολικός χρόνος επικοινωνίας θα είναι:

$$t_{comm} = s^2(2(t_{startup} + n * m * t_{data}) + (t_{startup} + m^2 t_{data}))$$

$$= (n/m)^2(3t_{startup} + (m^2 + 2n * m)t_{data})$$

Χρόνος Υπολογισμού

Κάθε slave επεξεργαστής υπολογίζει παράλληλα s πολλαπλασιασμούς υποπινάκων και s προσθέσεις υποπινάκων. Ένας ακολουθιακός πολλαπλασιασμός υποπινάκων απαιτεί m^3 πολλαπλασιασμούς και m^3 προσθέσεις. Μια ακολουθιακή πρόσθεση υποπινάκων απαιτεί m^2 προσθέσεις. Οπότε:

$$t_{comp} = s(2m^3 + m^2) = O(sm^3) = O(nm^2)$$

4.2.3 Αναδρομική υλοποίηση

Ο πολλαπλασιασμός χρησιμοποιώντας υποπίνακες προτείνει μια αναδρομική διαίρει-και-βασίλευε υλοποίηση, όπως περιγράφεται από τους Horowitz και Zorat [25] και τον Hake [23].

Έστω οι πίνακες A και B διαστάσεων n x n, όπου τον n είναι δύναμη του 2 (αντίστοιχη είναι και η υλοποίηση όταν το n δεν είναι δύναμη του 2). Κάθε πίνακας υποδιαιρείται σε τέσσερις υποπίνακες. Έστω ότι οι υποπίνακες του A: A_{pp}, A_{pq}, A_{qp} και A_{qq}, και οι υποπίνακες του B: B_{pp}, B_{pq}, B_{qp} και B_{qq} (όπου p και q είναι η θέση στη γραμμή και στη στήλη αντίστοιχα). Για να προκύψει ο τελικός πίνακας C είναι απαραίτητο να πολλαπλασιαστούν μεταξύ τους οι εξής υποπίνακες: A_{pp} x B_{pp}, A_{pq} x B_{qp}, A_{pp} x B_{pq}, A_{pq} x B_{qp}, A_{qp} x B_{pp}, A_{qq} x B_{qp}, A_{qp} x B_{pq}, A_{qq} x B_{qq}. Ο ίδιος αλγόριθμος μπορεί να υποδιαιρέσει τον καθένα υποπίνακα σε 4 άλλους υποπίνακες και ούτω καθεξής. Οπότε, ο αλγόριθμος μπορεί να γραφεί αναδρομικά ως εξής:

```
mat_mult(A, B, s)
{
    if (s==1)                    //αν ο πίνακας έχει ένα στοιχείο
        C = A * B;               //πολλαπλασίασε τα στοιχεία
    else{                        //αλλιώς συνέχισε να κάνεις αναδρομικές κλήσεις
        s = s / 2;               //ο αριθμός των στοιχείων σε κάθε γραμμή/στήλη
        P0 = mat_mult(A_pp, B_pp, s);
        P1 = mat_mult(A_pq, B_qp, s);
        P2 = mat_mult(A_pp, B_pq, s);
        P3 = mat_mult(A_pq, B_qq, s);
        P4 = mat_mult(A_qp, B_pp, s);
        P5 = mat_mult(A_qq, B_qp, s);
        P6 = mat_mult(A_qp, B_pq, s);
        P7 = mat_mult(A_qq, B_qq, s);
        C_pp = P0 + P1;          //πρόσθεσε τα γινόμενα
```

```
            C_pq = P2 + P3;                    //για να σχηματιστεί ο τελικός πίνακας
            C_qp = P4 + P5;
            C_qq = P6 + P7;
        }
        return C;                              //επέστρεψε τον τελικό πίνακα
}
```

Καθεμιά από τις 8 αναδρομικές κλήσεις μπορεί να πραγματοποιηθεί ταυτόχρονα και από διαφορετικούς επεξεργαστές. Περισσότεροι επεξεργαστές μπορούν να χρησιμοποιηθούν μετά από επαναλαμβανόμενες αναδρομικές κλήσεις. Στη γενική περίπτωση ο αριθμός των επεξεργαστών πρέπει να είναι δύναμη του 8.

4.2.4 Υλοποίηση πλέγματος

Η πιο διαδεδομένη μέθοδος πολλαπλασιασμού πίνακα είναι η χρησιμοποίηση ενός δισδιάστατου πλέγματος, όπου κάθε κόμβος υπολογίζει ένα στοιχείο (ή υποπίνακα) του αποτελέσματος. Οι συνδέσεις στο πλέγμα διευκολύνουν την αποστολή ταυτόχρονων μηνυμάτων μεταξύ των γειτονικών κόμβων.

Υπάρχουν αρκετοί τρόποι για να αναπτυχθεί ένας αλγόριθμος πολλαπλασιασμού πινάκων σε μια τοπολογία πλέγματος. Οι πιο διαδεδομένοι από αυτούς είναι ο αλγόριθμος του Cannon [8] και η συστολική προσέγγιση. Άλλοι αλγόριθμοι πολλαπλασιασμού πινάκων έχουν προταθεί από τον Fox και μπορούν να βρεθούν στο Fox et al [16].

ΚΕΦΑΛΑΙΟ 5

ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΑΝΑΝΕΩΣΗΣ ΤΗΣ ΒΑΣΗΣ

5.1 Ανάλυση

Σε αυτό το κεφάλαιο θα περιγραφεί η διαδικασία παραλληλοποίησης του αλγόριθμου simplex εξωτερικών σημείων. Το μεγαλύτερο υπολογιστικό κόστος στον αλγόριθμο αυτόν είναι ο υπολογισμός της ανανεωμένης βάσης. Οι μετρήσεις από την υπολογιστική μελέτη που θα παρουσιαστεί στο επόμενο κεφάλαιο έδειξαν ότι η διαδικασία ανανέωσης της βάσης καταλαμβάνει σημαντικό μερίδιο από το συνολικό χρόνο εκτέλεσης. Γι' αυτό το λόγο η παραλληλοποίηση του αλγορίθμου βασίστηκε στην παραλληλοποίηση της ανανέωσης της βάσης.

Για την ανανέωση της βάσης δε χρησιμοποιείται σχήμα πλήρης αντιστροφής, αλλά γίνεται χρήση eta-μητρών η οποία μειώνει τον υπολογιστικό χρόνο. Βέβαια, μετά από κάποιο συγκεκριμένο αριθμό επαναλήψεων κάθε φορά γίνεται πλήρης αντιστροφή της βάσης.

Από τους παράλληλους αλγορίθμους πολλαπλασιασμού πινάκων που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, σε αυτήν την εργασία χρησιμοποιήσαμε τον αλγόριθμο απευθείας υλοποίησης με υποπίνακες (block matrices).

Για την ανανέωση της βάσης είναι απαραίτητοι δύο πίνακες:

- Ενός eta-πίνακα (έστω E_inv) και
- ο πίνακας που περιέχει την αντίστροφη της βάσης (έστω Base_inv).

Και οι δύο πίνακες έχουν διαστάσεις n x n. Στην παραλληλοποιημένη εκδοχή του αλγόριθμου EPSA ο πίνακας E_inv διαιρείται σε τόσους υποπίνακες όσους και οι επεξεργαστές που διαθέτουμε. Για την ακρίβεια λόγω της ειδικής δομής που έχει ο πίνακας E_inv, δεν είναι απαραίτητο να σταλεί όλος ο πίνακας, αλλά μόνο κάποια

στοιχεία μιας συγκεκριμένης στήλης. Ο πίνακας E_inv είναι ένας μοναδιαίος πίνακας εκτός από μια στήλη του που περιέχει στοιχεία. Στο παρακάτω σχήμα φαίνεται ένα παράδειγμα ενός eta-πίνακα διαστάσεων 4 x 4.

$$\begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix}\begin{pmatrix} 4 \\ 5 \\ -2 \\ 3 \end{pmatrix}\begin{matrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{matrix}$$

Δεν είναι απαραίτητη η αποστολή όλου του E_inv στους slaves επεξεργαστές, αλλά μόνο η αποστολή των στοιχείων της $2^{ης}$ γραμμής που τους αναλογούν. Για παράδειγμα, για τον πιο πάνω πίνακα αν χρησιμοποιήσουμε s = 4 επεξεργαστές, ο κάθε επεξεργαστής θα πρέπει να γνωρίζει μόνο ένα στοιχείο της $2^{ης}$ στήλης, αλλά και τον αριθμό της στήλης (δηλαδή το 2) που περιέχει τα επιπλέον στοιχεία.

Πιο συγκεκριμένα, ο $1^{ος}$ επεξεργαστής θα πάρει το στοιχείο E_inv[1, 2] θα δημιουργήσει τον υποπίνακα του eta που του αναλογεί και θα τον πολλαπλασιάσει με τον πίνακα B_inv για την εξαγωγή ενός υποπίνακα από το τελικό αποτέλεσμα. Σχηματικά:

$$\begin{bmatrix} 1 & 4 & 0 & 0 \end{bmatrix} x B\_inv = B\_inv_{new\ 1.}$$

Δηλαδή στο απλοποιημένο παράδειγμα που παρουσιάζουμε ο κάθε επεξεργαστής θα υπολογίσει μία γραμμή από το αποτέλεσμα. Στη γενική περίπτωση, όπου n x n οι διαστάσεις των πινάκων και s οι επεξεργαστές, ο κάθε επεξεργαστής θα λαμβάνει από το master n/s στοιχεία (συν ένα η στήλη του πίνακα E_inv που περιλαμβάνει το διάνυσμα στήλη) και θα υπολογίζει τις n/s του συνολικού αποτελέσματος. Αυτές τις γραμμές πρέπει να τις ξαναστείλει πίσω στον master επεξεργαστή.

Σε αντίθεση ο πίνακας B_inv στέλνεται σε όλους τους πίνακες. Δηλαδή, ο συνολικός χρόνος επικοινωνίας σε κάθε επανάληψη είναι:

$$t_{comm} = (s * t_{startup} + (n + 1) * t_{data}) + s * (t_{startup} + n^2/s * t_{data})$$

Ο κάθε υπολογιστής έχει να πολλαπλασιάσει έναν πίνακα διαστάσεων n/s x n με ένα πίνακα n x n. Άρα πρέπει να κάνει n^3/s πολλαπλασιασμούς και n^3/s προσθέσεις. Επίσης, ο master επεξεργαστής πρέπει να συνενώσει τους s υποπίνακες

μεταξύ τους. Ο συνολικός χρόνος υπολογισμού ανά επανάληψη του αλγορίθμου είναι:

$$t_{comp} = (s * t_{startup} + 2 * (n^3 + s) * t_{data})$$

5.2 Ψευδοκώδικας υλοποίησης

Σε αυτήν την ενότητα θα παρουσιαστεί υπό μορφή ψευδοκώδικα η παράλληλη υλοποίηση. Αρχικά, όλοι οι επεξεργαστές γνωρίζουν τον αριθμό των επεξεργαστών (έστω Numoflabs) και τη διάσταση του προβλήματος (έστω diastasi). Στη συνέχεια, αφού ο master επεξεργαστής πρέπει να στείλει τα απαραίτητα στοιχεία στους slave επεξεργαστές. Ο κώδικας του master επεξεργαστή υπό μορφή ψευδοκώδικα φαίνεται παρακάτω:

```
1. Base_inv = labBroadcast(1, Base_inv);
2. for i=2:Numoflabs
3.     labSend(E_inv((i-1)*diastasi/Numoflabs+1:
          i*diastasi/Numoflabs,:),i,2);
4. end
5. Base_inv=E_inv(1:diastasi/Numoflabs,:)*Base_inv;
6. for i=2:Numoflabs
7.          d2=labReceive(i,3);
8.     Base_inv=[Base_inv; d2];
9. end
```

Κώδικας master επεξεργαστή

Αρχικά, ο master στέλνει τον πίνακα Base_inv σε όλους τους slaves επεξεργαστές (γραμμή 1). Στη συνέχεια στέλνει στους slaves επεξεργαστές ένα κομμάτι από τον πίνακα E_inv (το κομμάτι έχει διαστάσεις $\frac{n}{Numoflabs}$ x n (γραμμές 2 – 4). Σημειώνεται ότι ο master επεξεργαστής κρατάει έναν υποπίνακα από τον πίνακα E_inv, καθώς είναι και slave επεξεργαστής αφού συμμετέχει στους υπολογισμούς. Στη γραμμή 5 ο master επεξεργαστής υπολογίζει το δικό του υποπίνακα. Τέλος, στις γραμμές 6 – 9, ο master επεξεργαστής λαμβάνει τους

υποπίνακες του τελικού αποτελέσματος από τους άλλους επεξεργαστές και τους συνενώνει σε έναν πίνακα τον Base_inv.

Στον παρακάτω πίνακα φαίνεται ο ψευδοκώδικας που εκετελούν οι slaves επεξεργαστές:

```
1. Base_inv=labBroadcast(1);
2. E_inv=labReceive(1,2);
3. Base_inv=E_inv*Base_inv;
4. labSend(Base_inv,1,3);
```

Κώδικας slave επεξεργαστή

Οι slaves επεξεργαστές αρχικά λαμβάνουν από τον master ολόκληρο τον πίνακα Base_inv (γραμμή 1) και ένα μέρος του πίνακα E_inv (γραμμή 2). Στη συνέχεια κάνουν τον υπολογισμό τους (γραμμή 3) και επιστρέφουν το αποτέλεσμα τους στο master (γραμμή 4).

Ο master επεξεργαστής εκτός από ένα μέρος του υπολογισμού για την ανανέωση της βάσης εκτελεί και τον ακολουθιακό αλγόριθμο. Οπότε, μόλις βρει βέλτιστη λύση στέλνει σήμα στους slaves επεξεργαστές να τερματίσουν.

ΚΕΦΑΛΑΙΟ 6

ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

6.1 Εισαγωγή

Πολλές εργασίες έχουν γραφεί για το πώς πρέπει να σχεδιάζονται τα πειράματα και πως πρέπει να διενεργούνται οι υπολογιστικές μελέτες [24, 43]. Για την υπολογιστική σύγκριση διαφόρων αλγορίθμων, απαραίτητη είναι η πολλαπλή εκτέλεση όλων των αλγορίθμων σε κοινές συλλογές από δεδομένα. Για τη διενέργεια μιας σωστής υπολογιστικής μελέτης, τα δεδομένα αυτά πρέπει να πληρούν ορισμένες προϋποθέσεις. Συγκεκριμένα, πρέπει να είναι μεγάλης διάστασης, να δημιουργούνται από γεννήτριες τυχαίων αριθμών και για κάθε διάσταση να λύνονται από τους αλγόριθμους 10 ή και περισσότερα στιγμιότυπα. Περισσότερες πληροφορίες σχετικά με τη δημιουργία τυχαίων συλλογών από δεδομένα μπορούν να βρεθούν στην αναφορά [22].

Στην ενότητα αυτή θα παρουσιαστεί η υπολογιστική μελέτη που διενεργήθηκε στα πλαίσια της εργασίας αυτής. Η υπολογιστική μελέτη διενεργήθηκε σε έναν Intel Core 2 Duo επεξεργαστή με 1.83 GHz και 3 GB μνήμης RAM με λειτουργικό σύστημα Windows Vista Home Premium. Όλοι οι χρόνοι επεξεργασίας υπολογίστηκαν σε δευτερόλεπτα (sec). Επίσης, τα προβλήματα που δημιουργήθηκαν ήταν όλα βέλτιστα και πυκνά (πάνω από 99% πυκνότητα).

Το λογισμικό που χρησιμοποιήθηκε για την παραλληλοποίηση του κώδικα ήταν το Matlab R2009a και έγινε χρήση της παράλληλης εργαλειοθήκης που προσφέρει το Matlab, της Parallel Computing Toolbox. Στο παράρτημα Α υπάρχει μία εκτενής παρουσίαση των κυριότερων χαρακτηριστικών της εργαλειοθήκης αυτής. Στην εργαλειοθήκη αυτή προσφέρεται η δυνατότητα εκτέλεσης παράλληλων αλγορίθμων σε πολυπύρηνους υπολογιστές. Στην υπολογιστική μελέτη αυτή τα πειράματα εκτελέστηκαν χρησιμοποιώντας 2 και 4 slaves επεξεργαστές.

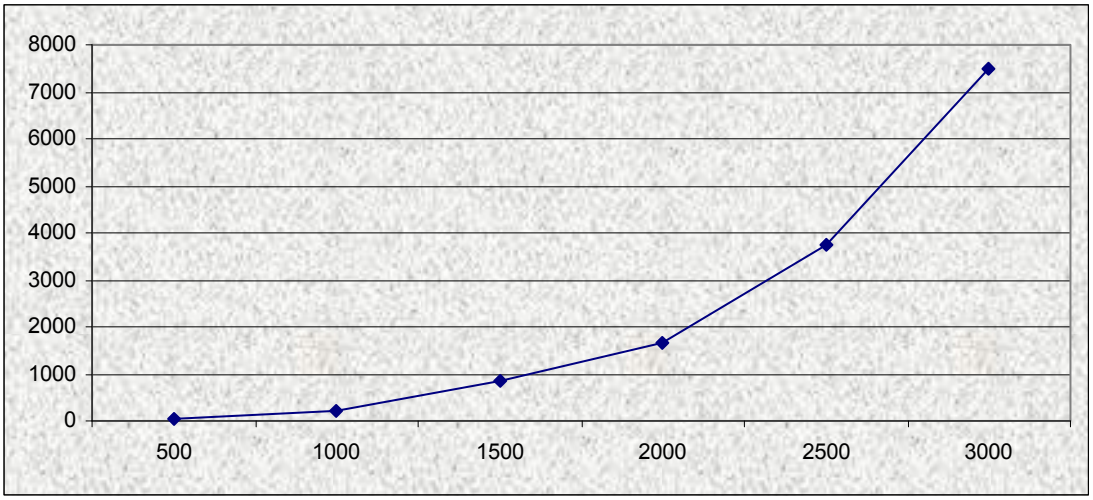
6.2 Εκτέλεση πειραμάτων

Για την διενέργεια της υπολογιστικής μελέτης δημιουργήθηκαν δέκα ΓΠ για κάθε διάσταση (από 500 μέχρι 3000 με βήμα 500). Τα γραμμικά προβλήματα που δημιουργήθηκαν είχαν όλα πυκνότητα πάνω από 99%. Στον παρακάτω πίνακα φαίνεται ο χρόνος για την εκτέλεση του ακολουθιακού αλγορίθμου σε κάθε διάσταση (ως χρόνος λαμβάνεται ο μέσος χρόνος από την εκτέλεση και των δέκα στιγμιότυπων):

Διάσταση	***Συνολικός χρόνος (σε sec)***
500	21,8105
1000	228,1047
1500	861,3751
2000	1659,1
2500	3731,8
3000	7474,4

Πίνακας 1: Συνολικός χρόνος ακολουθιακού αλγορίθμου

Παρατηρούμε, όπως ήταν αναμενόμενα, μια πολυωνυμική αύξηση του χρόνου. Γραφικά αποτυπώνεται στο παρακάτω σχήμα:



Διάγραμμα 1: Γράφημα συνολικού χρόνου ακολουθιακού αλγορίθμου

Ιδιαίτερη έμφαση στην παραλληλοποίηση δόθηκε στην φάση που ανανεώνεται η βάση. Όπως, περιγράφηκε στο προηγούμενο κεφάλαιο γίνεται χρήση eta-πινάκων για την αντιστροφή της βάσης. Για την καλύτερη κατανόηση της επιλογής μας να δώσουμε ιδιαίτερη έμφαση σε αυτήν τη διαδικασία του αλγόριθμου, παρατίθεται ο συνολικός χρόνος εκτέλεσης του αλγορίθμου αλλά και ο χρόνος περάτωσης της φάσης της αντιστροφής της βάσης:

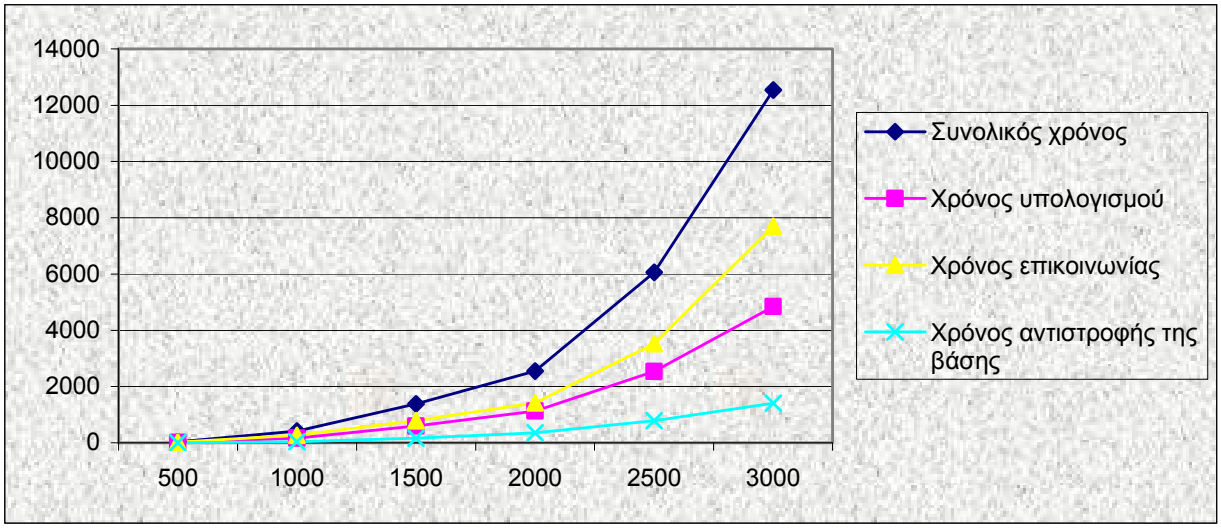
Διάσταση	***Συνολικός χρόνος (σε sec)***	***Χρόνος αντιστροφής της βάσης (σε sec)***
500	21,8105	8,0029
1000	228,1047	89,8416
1500	861,3751	340,59
2000	1659,1	640,15
2500	3731,8	1231,6
3000	7474,4	2567,89

Πίνακας 2: Σύγκριση ακολουθιακού χρόνου και χρόνου αντιστροφής της βάσης του ακολουθιακού αλγορίθμου

Διαπιστώνουμε ότι ένα σημαντικό μέρος του υπολογιστικού χρόνου καταλαμβάνει η αντιστροφή της βάσης. Όπως προαναφέρθηκε και στο προηγούμενο κεφάλαιο, ήταν φυσικό επακόλουθο η παραλληλοποίηση αυτού του σημείου του αλγόριθμου. Στο παρακάτω σχήμα φαίνεται ο χρόνος εκτέλεσης των πειραμάτων με την παράλληλη υλοποίηση. Έχουν μετρηθεί τέσσερις χρόνοι που ενδιαφέρουν την εργασία αυτή: i) ο συνολικός χρόνος, ii) ο χρόνος επικοινωνίας, iii) ο χρόνος υπολογισμού και iv) ο χρόνος για την ανανέωση της βάσης. Τα πειράματα διεξήχθησαν για 2 και 4 επεξεργαστές αντίστοιχα. Στους πίνακες 3 και 4 φαίνονται οι χρόνοι για την εκτέλεση του παράλληλου αλγορίθμου σε 2 και 4 επεξεργαστές αντίστοιχα. Επίσης, τα ίδια αποτελέσματα παρουσιάζονται γραφικά στα διαγράμματα 2 και 3.

Διάσταση	***Συνολικός χρόνος***	***Χρόνος υπολογισμού***	***Χρόνος επικοινωνίας***	***Χρόνος αντιστροφής της βάσης***
500	37,72	14,24	23,48	3,88
1000	424,59	160,96	263,63	42,20
1500	1384,70	595,92	788,80	167,51
2000	2548,40	1123,80	1424,60	355,07
2500	6061,60	2540,50	3521,10	786,71
3000	12536,70	4839,10	7697,10	1403,90

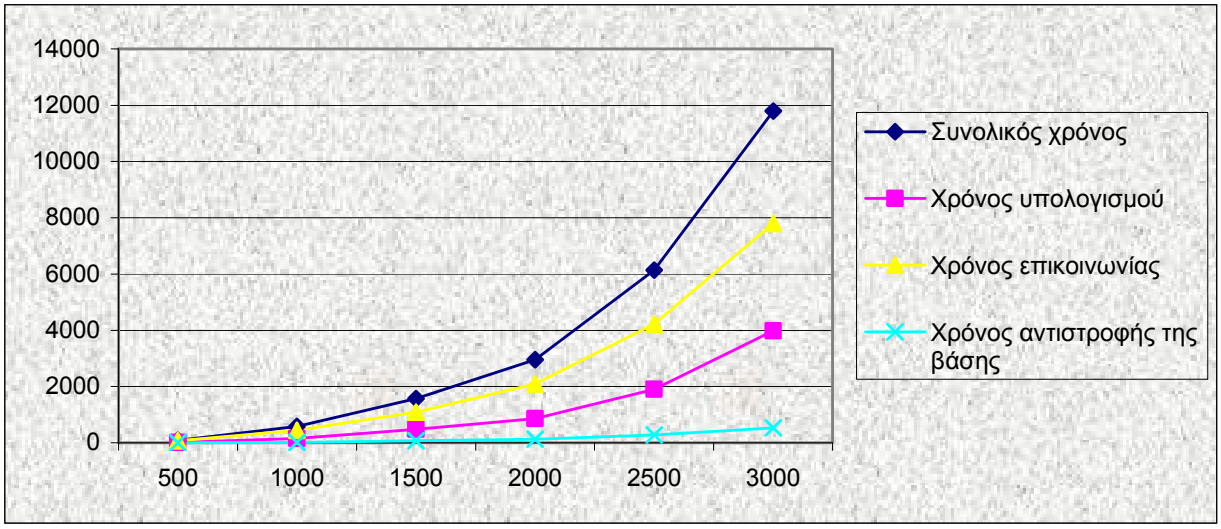
Πίνακας 3: Εκτέλεση παράλληλου αλγορίθμου για 2 επεξεργαστές



Διάγραμμα 2: Γράφημα εκτέλεσης παράλληλου αλγορίθμου για 2 επεξεργαστές

Διάσταση	***Συνολικός χρόνος***	***Χρόνος υπολογισμού***	***Χρόνος επικοινωνίας***	***Χρόνος αντιστροφής της βάσης***
500	85,02	14,20	70,82	0,37
1000	586,32	148,78	437,52	17,96
1500	1577,10	484,62	1092,50	76,80
2000	2957,50	859,38	2098,20	126,58
2500	6131,90	1905,10	4226,80	273,84
3000	11789,45	3985,54	7803,91	526,54

Πίνακας 4: Εκτέλεση παράλληλου αλγορίθμου για 4 επεξεργαστές



Διάγραμμα 3: Γράφημα εκτέλεσης παράλληλου αλγορίθμου για 4 επεξεργαστές

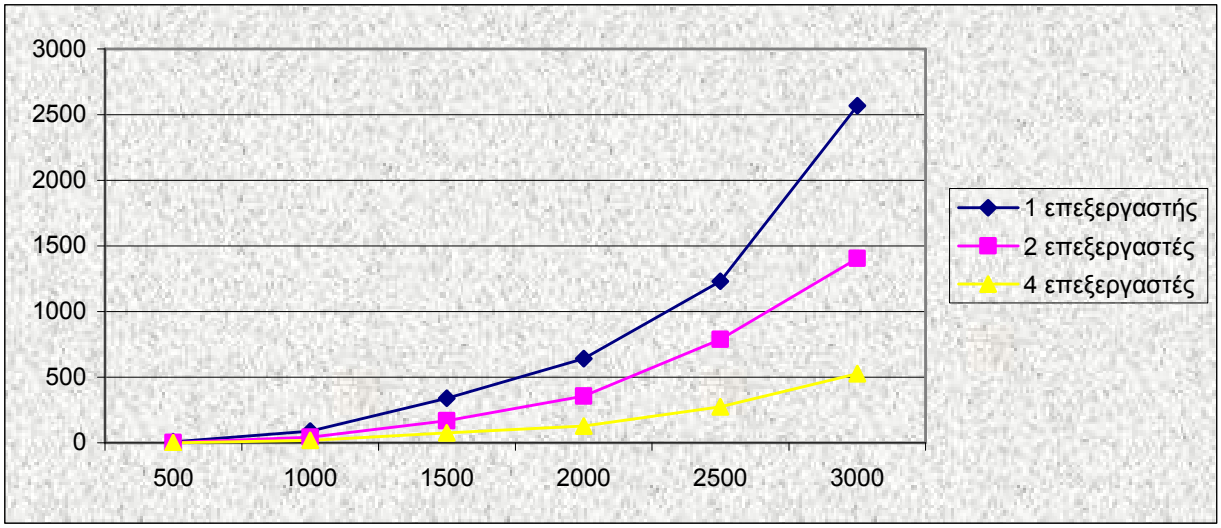
Από τα παραπάνω γραφήματα παρατηρούμε ότι ο λόγος του χρόνου υπολογισμού προς το χρόνο επικοινωνίας είναι μικρός. Παράλληλοι αλγόριθμοι που έχουν μεγάλες ανάγκες σε επικοινωνία μεταξύ των υπολογιστών είναι δύσκολοι στην παραλληλοποίηση. Γι΄ αυτό παρατηρούμε ότι η παράλληλη εκδοχή δεν είναι καλύτερη από την ακολουθιακή. Ωστόσο, σε έναν ιδανικό υπολογιστή όπου οι επικοινωνία είναι αμελητέα, η επιτάχυνση που αποκτούμε είναι 1.5 έως 2. Όσο αυξάνει η διάσταση του προβλήματος, τόσο αυξάνει και η επιτάχυνση.

Η μεγάλη ανάγκη για επικοινωνία μεταξύ των υπολογιστών πηγάζει από το γεγονός ότι ο master επεξεργαστής πρέπει σε κάθε επανάληψη να στέλνει ολόκληρο τον πίνακα που περιέχει τη βάση και στη συνέχεια να λαμβάνει τους υποπίνακες του τελικού αποτελέσματος από τους υπολογιστές. Οπότε, αν και η αντιστροφή της βάσης παραλληλοποιείται με μία σημαντική επιτάχυνση, ο χρόνος επικοινωνίας είναι πολύ μεγάλος για να υπάρχει βελτίωση του παράλληλου αλγορίθμου.

Στον παρακάτω πίνακα παρουσιάζονται οι χρόνοι αντιστροφής της βάσης για 1, 2 και 4 επεξεργαστές. Όπως παρατηρούμε με την αύξηση των υπολογιστών αποκτούμε μια πολύ καλή επιτάχυνση από αυτή τη φάση. Η επιτάχυνση είναι 1.9 όταν χρησιμοποιούμε 2 επεξεργαστές και 7.58 στην περίπτωση των 4 επεξεργαστών.

Διάσταση	*1 επεξεργαστής*	*2 επεξεργαστές*	*4 επεξεργαστές*
500	8,00	3,88	0,37
1000	89,84	42,20	17,96
1500	340,59	167,51	76,80
2000	640,15	355,07	126,58
2500	1231,60	786,71	273,84
3000	2567,89	1403,90	526,54

Πίνακας 5: Σύγκριση του χρόνου αντιστροφής της βάσης για 1, 2 και 4 επεξεργαστές



Διάγραμμα 4: Γράφημα σύγκρισης του χρόνου αντιστροφής της βάσης για 1, 2 και 4 επεξεργαστές

ΚΕΦΑΛΑΙΟ 7

ΣΥΜΠΕΡΑΣΜΑΤΑ – ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Ο αλγόριθμος simplex έχει χαρακτηριστεί ως ένας από τους σημαντικότερους αλγορίθμους του 20^{ου} αιώνα. Στις περισσότερες των περιπτώσεων είναι η καλύτερη επιλογή για την επίλυση γραμμικών προβλημάτων (ΓΠ). Οι υπολογιστικές του ανάγκες σε πόρους έχουν οδηγήσει τους ερευνητές να εστιάσουν στην εύρεση μεθόδων παραλληλοποίησης του. Η παραλληλοποίηση του είναι μια πρόκληση, γιατί οι ανάγκες του σε υπολογιστικούς πόρους είναι μεγάλες.

Σε αυτήν την εργασία παρουσιάστηκε ένας παράλληλος αλγόριθμος simplex εξωτερικών σημείων που υλοποιήθηκε στο παράλληλο περιβάλλον του Matlab R2009a. Ιδιαίτερη έμφαση δόθηκε στην παραλληλοποίηση της διαδικασίας όπου ανανεώνεται η βάση. Μελετήθηκαν διάφορα σχήματα πολλαπλασιασμού πινάκων από τα οποία επιλέχθηκε για υλοποίηση εκείνο όπου διαιρεί τους αρχικούς πίνακες σε υποπίνακες.

Η επιτάχυνση που αποκομίσαμε στην φάση της ανανέωσης της βάσης ήταν γραμμική. Ωστόσο, παρατηρήσαμε ότι ο αλγόριθμος έχει μεγάλες ανάγκες σε επικοινωνία μεταξύ των διαφορετικών επεξεργαστών και συνολικά ο λόγος υπολογισμού προς επικοινωνία είναι μικρός. Αυτό υπονοεί από μόνο του ότι πρέπει να μελετηθούν στο μέλλον και άλλες τεχνικές παραλληλοποίησης του αλγορίθμου, όπως ο υπολογισμός της ανανέωσης της βάσης με LU παραγοντοποίηση.

Επίσης, βελτιώσεις μπορούν να γίνουν στον παράλληλο αλγόριθμο με τη χρησιμοποίηση άλλων κανόνων περιστροφής, όπως:

- Κανόνας Bland’s,
- Κανόνας στοίβας,
- Κανόνας ουράς,
- Η μέθοδος του πιο πρόσφατα θεωρηθέντος στοιχείου,
- Κανόνας μέγιστης αύξησης,
- Κανόνας μερικής τιμολόγησης,
- Κανόνας της πιο κατηφορικής ακμής και

- Κανόνας περιστροφής EPSA.

Τέλος, κρίνεται σκόπιμο να εκτελεστούν οι αλγόριθμοι και σε άλλες δύο βιβλιοθήκες παράλληλου προγραμματισμού: τις OpenMP και MPI.

Βιβλιογραφία

[1] Alexouda, G., Paparrizos, K. (1997). “A comparative computational study with an exterior point simplex algorithm”, Proceedings of 4th Balkan Conference on Operational Research, Vol. 1, pp. 12-22.

[2] Anstreicher, K., Terlaky, T. (1994). “A monotonic build-up simplex algorithm for linear programming”, Operations Research, vol. 42, pp. 556-561.

[3] Badr, E. S., M. Moussa, K. Papparrizos, N. Samaras, and A. Sifaleras. (2006). “Some computational results on MPI parallel implementations of dense simplex method”, Transactions on Engineering, Computing and Technology, 17:228-231, December 2006.

[4] Bixby, E.R. (1992). “Implementing the simplex method: The initial basis”, ORSA Journal on Computing, Vol. 4, pp. 267-284.

[5] Blackfor, L. S., Choi, J., Cleary, A., Dazevedo, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, J., Petitet. A., Stanley. K., Walker, D., and Whaley, R.C. (1997). “ScaLAPACK User’s Guide”, Society for Industrial and Applied Mathematics.

[6] Borgwardt, H.K. (1982). “Some distribution independent results about the asymptotic order of the average number of pivot steps in the simplex method”, Mathematics of Operations Research, Vol. 7(3), pp. 441-462.

[7] Borgwardt, H.K. (1982). “The average number of the pivot steps required by the simplex method is polynomial”, Zeitschrift fur Operational Research, Series A: Theory, Vol. 26(5), pp. 157-177.

[8] Cannon, L. E., (1969). “A Cellular Computer to Implement the Kalman Filter Algorithm”, Ph.D Thesis, Montana State University, Bozman, MT

[9] Chapman, B., Bodin, F., Hill L., Merlin, J., Viland. J., and Wollenweber, F. (1999). “FITS – A light-weight integrated programming environment”, Lecture Notes in Computer Science 1685, 125-134.

[10] Cvetanovic, Z, Freedman E. G., and Nofsinger C. (1991). “Efficient decomposition and performance of parallel PDE, FFT, Monte-Carlo simulations, simplex, and sparse solvers”. Journal of Supercomputing, Vol. 5, pp. 19-38..

[11] Dantzig, G. B. (1949). “Programming in a Linear Structure”. Report of the September 9, 1948 meeting in Madison, Econometrica, Vol. 17, pp. 73-74.

[12] Dantzig, G. B. (1982). “Requirements about the Origins of Linear Programming”. Operations Research Letters, Vol. 1, pp. 43-48.

[13] Dikin, I. I. (1967). “Iterative Solution of Problems of Linear and Quadratic Programming”. Soviet Mathematics Doklady, Vol. 8, pp. 674-675

[14] Dosios, K., Paparrizos, K. (1997). “Resolution of the problem of degeneracy in a primal and dual simplex algorithm”, Operations Research Letters, Vol. 20, pp. 45-50.

[15] Eckstein, J., Boduroglu, I., Polymenakos, L. and Goldfarb D. (1995). "Data-Parallel Implementations of Dense Simplex Methods on the Connection Machine CM-2," ORSA Journal on Computing, Vol. 7, n. 4, pp. 402-416.

[16] Fox, G., Johnson, M., Lyzenga, G, Otto S., Salmon K., and Walker (1988). “Solving Problems on Concurrent Processors, Volume 1, Prentice Hall, Englewood Cliffs, NJ”

[17] Foster, I. (1995). “Designing and building parallel programs: Concepts and tools for parallel software engineering”, Addison-Wesley.

[18] Gay D. M. (1985). “Electronic mail distribution of linear programming test pro

blems". Mathematical Programming Society COAL Newsletter, Vol. 13, pp: 10-12

[19] Gondzio, J. (1992). "Splitting dense columns of constraint matrix in interior point methods for large-scale linear programming", Optimization, Vol. 24, pp. 285-297.

[20] Gondzio, J. (1996). "Multiple centrality corrections in a primal-dual method for linear programming", Computational Optimization and Applications, Vol. 6, pp. 137-156.

[21] Hall, J.A.J. and K.I.M. McKinnon , "ASYNPLEX an asynchronous parallel revised simplex algorithm," Technical Report MS95-050a Department of Mathematics University of Edinburgh.

[22] Hall, N.G., Posner M.E. (2001), "Generating Experimental Data for Computational Testing with Machine Scheduling Application", Operations Research, vol. 49, pp. 854-865.

[23] Hake, J. F. (1993), "Parallel Algorithms for Matrix Operations and Their Performance on Multiprocessor Systems", in Advances in Parallel Algorithms, L. Kronsjo and D. Shumsheruddin, eds., Halsted Press, New York..

[24] Hooker, J.N. (1994), "Needed: an empirical science of algorithms", Operations Research, vol. 42(2), pp. 201-212.

[25] Horowitz, E. and Zorat, A. (1983), "Divide and Conquer for Parallel Processing", IEEE Trans. Comput., Vol. C-32, No. 6, pp. 582-585.

[26] Karmarkar, N. (1984). "A New Polynomial – Time Algorithm for Linear Programming". Combinatorica, Vol. 4, pp. 373-395

[27] Khachian, L. G. (1979). “A polynomial Algorithm for Linear Programming”. Doklady Acadeniia Nauk USSR, Vol. 224, pp. 1093-1096. [English translation: Soviet Mathematics Doklady, Vol. 20, pp. 191-194].

[28] Klee, V. L. and Minty G. J. (1972). “How good is the Simplex Algorithm”. Inequalities III, Academic Press, London and New York, pp. 159-175.

[29] Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). “Introduction to parallel computing design and analysis of parallel algorithms”, Benjaming-Cummings.

[30] Lau, L. (1996). “Implementation of scientific applications in a heterogeneous distributed network”, PhD Thesis, University of Queensland, Department of Mathematics.

[31] Lentini, M., Reinoza A., Teruel A., and Guillen, A. (1995). “SIMPAR: a parallel sparse simplex”. Computational and Applied Mathematics, Vol. 14, pp. 49-58.

[32] Lustig, J. I., Marstenand, E.R., Shanno, F.D. (1992). “On implementing Mehrotra’s predictor-corrector interior-point method for linear programming”, SIAM Journal on Optimization, Vol. 2, pp. 435-449.

[33] Merlin, J., Baden, S., Fink, S., and Chapman, B., (1999). “Multiple data parallelism with HPF and KeLP”, Journal of Future Generation Computer Systems 15, 393-405.

[34] Moldovan, D. I. (1993), “Parallel Processing from Applications to Systems”, Morgan Kaufmann, San Mateo, CA

[35] Murty, G. K., Fathi, Y. (1984), “A feasible direction method for linear programming”, Operations Research Letters, Vol. 3, pp. 121-127.

[36] Nieplocha, J., Harisson R. J., and Littlefield R.J. (1994), “Global arrays: A portable shared-memory programming model for distributed memory computers”, Proceedings of Supercomputing 94, 340-349.

[37] Paparrizos, K. (1991). “An infeasible (Exterior Point) simplex algorithm for assignment problems”, Mathematical Programming, Vol. 51, pp. 45-54.

[38] Paparrizos, K. (1993). “An exterior point simplex algorithm for (general) linear programming problems”, Annals of Operations Research, Vol. 47, pp. 497-508.

[39] Paparrizos, K. (1996). “Exterior point simplex algorithms, simple and short proof of correctness”. Proceedings of SYMOPIS ’96, pp. 13-18.

[40] Paparrizos, K., Samaras, N., Stephanides, G. (2000). “An efficient simplex type algorithm for sparse and dense linear programs”, accepted for publication in European Journal of Operational Research.

[41] Paparrizos, K., Samaras, N., Tsiplidis, K. (2001). “Pivoting algorithms for (LP) generating two paths”, Encyclopedia of Optimization, Pardalos, P., Floudas, C. (Eds.), Kluwer Academic Publishers, Vol. 4, pp. 302-306.

[42] Plastino, A., Ribeiro, C. C., and Rodriguez N., (1999). “A tool for SPMD application development with support for load balancing”, Proceedings of the ParCo Parallel Computing Conference.

[43] Rardin R.L., Uzsou R. (2001), Experimental evaluation of heuristic optimization algorithms: A tutorial, Journal of heuristics, Vol. 7, pp. 261-304.

[44] Samaras, N. (2001). “Computational improvements and efficient implementation of two path pivoting algorithms”, PhD Dissertation, University of Macedonia.

[45] Sherali, H.D., Soyster, A.L., Baines, S.G. (1983). “Non adjacent extreme point methods for solving linear programs”, Naval Research Logistics Quarterly, Vol. 30, pp. 145-161.

[46] Shor, N. Z. (1977). “Cut-off Method with Space Extension in Convex Programming Problems”. Kibernetika, Vol. 13, pp. 94-95 [translated in Cybernetics, Vol. 13 pp. 94-96]

[47] W. Shu. (1995). “Parallel implementation of a sparse simplex algorithm on MIMD distributed memory computers”. Journal of Parallel and Distributed Computing, Vol. 3, pp. 25-40.

[48] Wei Shu and Min-You Wu. Sparse Implementation of Revised Simplex Algorithms. in the proceeding of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, March 22-24, 1993, Norfolk

[49] C. B. Stunkel. (1988). “Linear optimization via message-based parallel processing. In International Conference on Parallel Processing, volume 3, pages 264-271, August 1988.

[50] Terlaky, T., Zhang, S. (1992). “A survey of pivot rules for linear programming”, Research Memorandum No 480, faculty of Economics, University of Croningen, Netherlands.

[51] Thomadakis, Michael E., and Jyh-Charn Liu, (1996). “An Efficient Steepest-Edge Simplex Algorithm for SIMD Computers,” Proc. Of the International conference on Super-Computing, ICS ’96, pp. 286-293.

[52] Wilkinson, B. and M. Allen, “Parallel Programming: Techniques and applications using networked workstations and parallel computers”, Addison-Wesley, North Carolina, 2005

[53] Wolf, K. and Kraemer-Furhmann, O. (1996). “An integrated environment to design parallel object-oriented programming”, Lecture Notes in Computer Science 1123, 120-127.

Παράρτημα

Εισαγωγή

Σκοπός του παραρτήματος αυτού είναι η παρουσίαση των κυριότερων δυνατοτήτων του Parallel Computing Toolbox του λογισμικού Matlab R2009a. Τα παράλληλα υπολογιστικά συστήματα επιτρέπουν στον χρήστη του περιβάλλοντος Matlab να μεταφέρει το φόρτο εργασίας από μία σύνοδο του Matlab (τον πελάτη) σε άλλες συνόδους του Matlab, οι οποίες ονομάζονται εργάτες (workers) ή εργαστήρια (labs). Ο χρήστης έχει τη δυνατότητα να χρησιμοποιήσει μεγάλο πλήθος εργατών ή εργαστηρίων ώστε να επωφεληθεί από την κατανεμημένη ή την παράλληλη επεξεργασία, αντίστοιχα. Έχει τη δυνατότητα να χρησιμοποιήσει έναν μόνο εργάτη ή εργαστήριο για να επωφεληθεί από την υπολογιστική ισχύ ενός επιπλέον υπολογιστή, είτε απλά για να διατηρήσει την αρχική σύνοδο πελάτη του Matlab ελεύθερη.

Η εργαλειοθήκη Parallel Computing Toolbox επιτρέπει στον χρήστη να χρησιμοποιήσει στο τοπικό του μηχάνημα, εκτός από την αρχική σύνοδο πελάτη του Matlab, τέσσερις επιπλέον εργάτες ή εργαστήρια. Το λογισμικό Matlab Distributed Computing Server επιτρέπει στο χρήστη να χρησιμοποιήσει όσους περισσότερους εργάτες ή εργαστήρια, από μία απομακρυσμένη συστοιχία υπολογιστών, του επιτρέπει η άδεια του.

Ένα χαρακτηριστικό της εργαλειοθήκης Parallel Computing Toolbox είναι η δυνατότητα να τρέχει έναν τοπικό δρομολογητή και έως τέσσερις εργάτες ή εργαστήρια στον υπολογιστή πελάτη. Το χαρακτηριστικό αυτό δίνει τη δυνατότητα να εκτελούνται κατανεμημένες και παράλληλες εργασίες χωρίς να απαιτείται μία απομακρυσμένη συστοιχία ή το λογισμικό Matlab Distributed Computing Server. Στην περίπτωση αυτή, όλες οι ενέργειες που απαιτούνται για τον πελάτη, τον δρομολογητή και τον υπολογισμό των διεργασιών εκτελούνται στον ίδιο υπολογιστή.

Παράλληλη Επισκόπηση μέσω του MATLAB R2009a

Το Matlab R009a παρέχει τη δυνατότητα της παράλληλης επισκόπησης ενός προγράμματος μέσω του parallel profiler. Η παράλληλη επισκόπηση είναι μία επέκταση της εντολής profile ειδικά για παράλληλες εργασίες. Δίνει τη δυνατότητα σε έναν χρήστη να δει πόσο χρόνο χρειάστηκε κάθε εργαστήριο για να υπολογίσει κάθε συνάρτηση και πόσο χρόνο για να επικοινωνήσει ή για να περιμένει για επικοινωνία με άλλα εργαστήρια.

Για την έναρξη της παράλληλης επισκόπησης, ο χρήστης θα πρέπει να χρησιμοποιήσει την εντολή mpiprofile κατά τη διάρκεια μίας παράλληλης εργασίας. Για να εκκινήσει ο χρήστης τον parallel profiler και να αρχίσει να συλλέγει δεδομένα θα πρέπει να εισάγει είτε στο αρχείο .m της παράλληλης εργασίας, είτε στο Παράθυρο Παράλληλων Εντολών (Parallel Command Window) την ακόλουθη εντολή:

P>> mpiprofile on

Από την εντολή αυτή και μετά ο profiler συλλέγει πληροφορίες σχετικά με την εκτέλεση του κώδικα σε κάθε εργαστήριο καθώς και για την επικοινωνία μεταξύ των εργαστηρίων. Οι πληροφορίες αυτές περιλαμβάνουν:

- το χρόνο εκτέλεσης κάθε συνάρτησης σε κάθε εργαστήριο,
- το χρόνο εκτέλεσης κάθε γραμμής κώδικα για κάθε συνάρτηση,
- το πλήθος των δεδομένων τα οποία μεταφέρονται μεταξύ κάθε εργαστηρίου και
- το χρόνο που σπαταλάει κάθε εργαστήριο περιμένοντας για ανάκτηση επικοινωνίας

Για να ανοίξει ο parallel profile viewer κατά τη διάρκεια της παράλληλης λειτουργίας (pmode), ο χρήστης θα πρέπει να εισάγει στο Παράθυρο Παράλληλων Εντολών την εντολή:

P>> mpiprofile viewer

Παράλληλοι βρόχοι-for

Η βασική ιδέα για τον τρόπο λειτουργίας ενός παράλληλου βρόχου-for (parfor) στο λογισμικό του Matlab είναι ίδια με αυτή του βρόχου-for του Matlab. Δηλαδή, το λογισμικό Matlab εκτελεί μία σειρά από δηλώσεις (το σώμα του βρόχου) σε ένα πεδίο τιμών. Τμήμα του βρόχου parfor εκτελείται στη σύνοδο του πελάτη, όπου ο βρόχος ορίστηκε, ενώ άλλο τμήμα του βρόχου εκτελείται στις συνόδους εργάτες του Matlab. Τα απαραίτητα δεδομένα τα οποία αφορούν τον parfor βρόχο στέλνονται από τον πελάτη στους εργάτες, όπου γίνεται και το μεγαλύτερο μέρος του υπολογισμού, και τα αποτελέσματα επιστρέφονται στον πελάτη, όπου πραγματοποιείται και η συνένωση όλων των αποτελεσμάτων.

Με τη μέθοδο αυτή, οι εργάτες του Matlab έχουν τη δυνατότητα να υπολογίζουν κατά την ίδια χρονική στιγμή τον ίδιο βρόχο. Για το λόγο αυτό, ο

παράλληλος βρόχος μπορεί να προσφέρει πολύ καλύτερη απόδοση από τον απλό βρόχο-for.

Κάθε εκτέλεση του σώματος του parfor βρόχου είναι μία επανάληψη του βρόχου. Οι εργάτες του Matlab δεν έχουν συγκεκριμένη σειρά με την οποία υπολογίζουν τις επαναλήψεις, κάθε επανάληψη εκτελείται ανεξάρτητα από τις υπόλοιπες.

Ένας παράλληλος βρόχος είναι χρήσιμος σε περιπτώσεις όπου είναι απαραίτητες πολλές επαναλήψεις. Ο παράλληλος βρόχος χωρίζει τις συνολικές επαναλήψεις του βρόχου σε ομάδες, έτσι ώστε κάθε εργάτης να εκτελέσει ένα τμήμα του συνολικού αριθμού επαναλήψεων. Οι παράλληλοι βρόχοι, επίσης, είναι χρήσιμοι όταν υπάρχουν επαναλήψεις που χρειάζονται αρκετό χρόνο για να εκτελεστούν.

Δεν είναι δυνατόν να χρησιμοποιηθεί ένας παράλληλος βρόχος όταν μία επανάληψη εξαρτάται από το αποτέλεσμα άλλων επαναλήψεων. Κάθε επανάληψη θα πρέπει να είναι ανεξάρτητη από τις υπόλοιπες. Επίσης, εξαιτίας του χρόνου που δαπανάται κατά την επικοινωνία του πελάτη με τους εργάτες, ο παράλληλος βρόχος δεν προσφέρει κανένα πλεονέκτημα όταν χρησιμοποιείται για μικρό πλήθος επαναλήψεων.

Διαχείριση των πόρων του Matlab: Matlabpool

Πριν την εκτέλεση ενός παράλληλου βρόχου θα πρέπει να οριστεί ένας αριθμός εργατών, που θα εκτελέσουν το βρόχο. Η δέσμευση των τοπικών εργατών του Matlab που θα χρησιμοποιηθούν για τον υπολογισμό των επαναλήψεων του βρόχου πραγματοποιείται με την εντολή:

matlabpool

ή

matlabpool open

Με τη χρήση μίας εκ των παραπάνω εντολών ξεκινούν τέσσερις σύνοδοι εργατών του Matlab στον τοπικό υπολογιστή. Στις συνόδους αυτές θα κατανεμηθούν οι επαναλήψεις του παράλληλου βρόχου για να εκτελεστούν.

Όταν εκτελεστεί ο παράλληλος βρόχος τότε θα πρέπει να αποδεσμευτούν οι σύνοδοι εργατών του MATLAB. Αυτό πραγματοποιείται με την παρακάτω εντολή:

matlabpool close

Τα παραδείγματα που ακολουθούν παράγουν το ίδιο αποτέλεσμα. Η εκτέλεση, όμως, του αριστερού κώδικα κάνει πολύ περισσότερο χρόνο διότι δεν έχουν δεσμευτεί οι σύνοδοι εργατών του Matlab στον τοπικό υπολογιστή.

tic
x = 0;
parfor (i=1:40000000)
x = x + i;
end
toc
 	
(Συνολικός χρόνος 100.69 δευτερόλεπτα)
	matlabpool open
	tic
	x = 0;
	parfor (i=1:40000000)
	x = x+ i;
	end
	toc
	matlabpool close
	
	(Συνολικός χρόνος 50.33 δευτερόλεπτα)

Διαδραστική Παράλληλη Λειτουργία (pmode)

Στην παράγραφο αυτή χρησιμοποιείται ο τοπικός δρομολογητής και τρέχει τέσσερα εργαστήρια στο τοπικό μηχάνημα μέσω του Matlab. Για την παράλληλη λειτουργία δεν απαιτείται εξωτερική συστοιχία ή δρομολογητής, μπορούν να χρησιμοποιηθούν οι πυρήνες του τοπικού υπολογιστή.

Για να ξεκινήσει η παράλληλη λειτουργία (pmode) πληκτρολογούμε την εξής εντολή:

pmode start local 4

Η εντολή αυτή ξεκινά τέσσερα τοπικά εργαστήρια, δημιουργεί μία παράλληλη εργασία για να εκτελεστεί σε αυτά τα εργαστήρια και ανοίγει το Παράθυρο Παράλληλων Εντολών.

Εάν τεθεί μία μεταβλητή στη παράλληλη λειτουργία, τότε τίθεται σε όλα τα εργαστήρια.

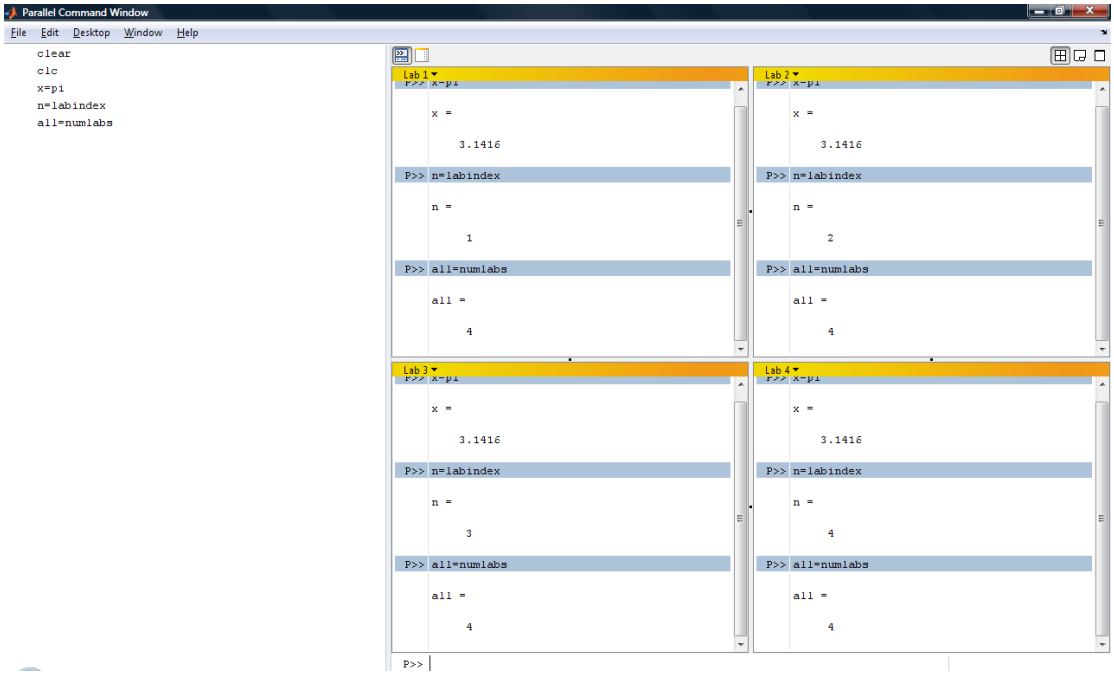
P>> x = pi

Μία μεταβλητή, όμως, δεν είναι απαραίτητο να έχει την ίδια τιμή σε όλα τα εργαστήρια. Η συνάρτηση labindex επιστρέφει την ταυτότητα κάθε εργαστηρίου το οποίο ανήκει στην παράλληλη εργασία. με την παρακάτω εντολή η μεταβλητή x λαμβάνει διαφορετική τιμή σε κάθε εργαστήριο. Συγκεκριμένα, λαμβάνει ως τιμή τον αριθμό του εργαστηρίου.

P>> x = labindex

Η συνάρτηση numlabs επιστρέφει το συνολικό αριθμό των εργαστηρίων που χρησιμοποιούνται στην παράλληλη εργασία.

P>> all = numlabs



Εικόνα 1: Παράθυρο Παράλληλων Εντολών

Με τις παρακάτω εντολές, αρχικά, δημιουργείται ο ίδιος πίνακας σε όλα τα εργαστήρια. Έπειτα αποκτά διαφορετική τιμή σε κάθε εργαστήριο.

P>> segment = [1 2; 3 4; 5 6]

*P>> segment = segment + 10 * labindex*

Ο πίνακας που δημιουργήθηκε στο προηγούμενο βήμα είναι ανεξάρτητος σε κάθε εργαστήριο, παρόλο που έχει το ίδιο όνομα σε όλα τα εργαστήρια. Υπάρχει, όμως, η δυνατότητα να δημιουργηθεί ένας πίνακας και να κατανεμηθεί στα

εργαστήρια. Η συνάρτηση, η οποία κατανέμει έναν πίνακα στα εργαστήρια είναι η εξής:

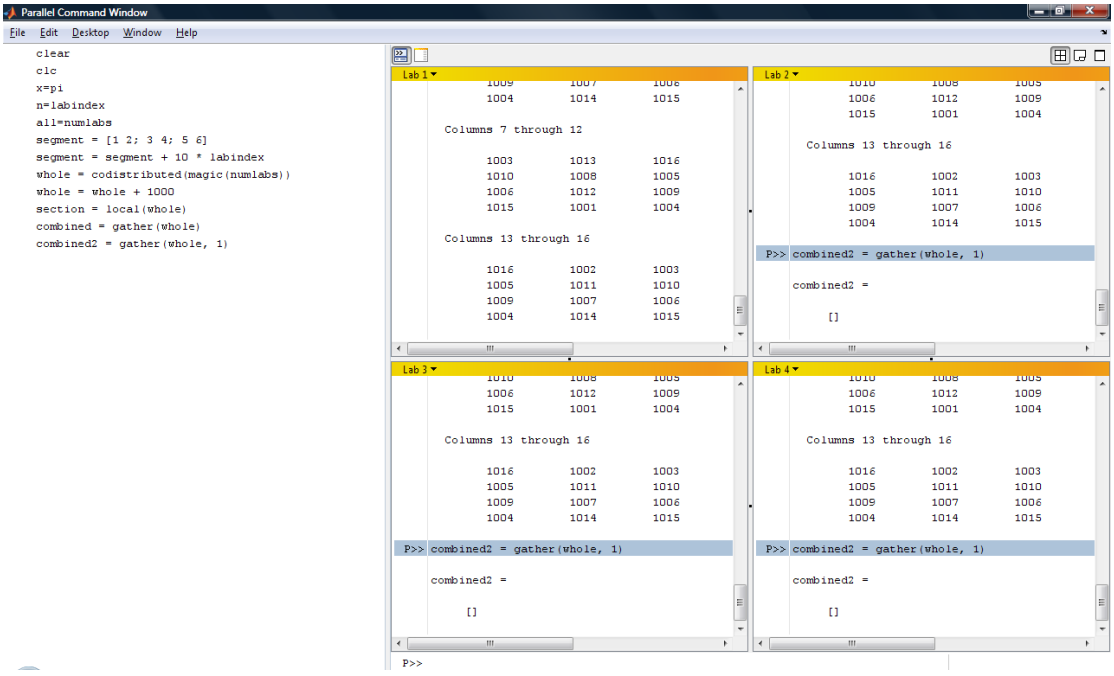
P>> whole = distribute(magic(numlabs))

Μετά τη χρήση της συνάρτησης distribute κάθε υπολογισμός που θα αφορά τον πίνακα whole θα πραγματοποιείται για κάθε τμήμα του πίνακα στο αντίστοιχο εργαστήριο. Όπως φαίνεται και με την εντολή που ακολουθεί.

P>> whole = whole + 1000

Παρόλο που ο κατανεμημένος πίνακας επιτρέπει στον χρήστη να τον διαχειρίζεται εξ ολοκλήρου, ο χρήστης έχει, επίσης τη δυνατότητα να χρησιμοποιήσει τη συνάρτηση local για να έχει πρόσβαση μόνο σε ένα τμήμα του κατανεμημένου πίνακα σε ένα συγκεκριμένο εργαστήριο.

P>> section = local(whole)



Εικόνα 2: Παραθύρο Παράλληλων Εντολών (2)

Ο χρήστης έχει, ακόμα, τη δυνατότητα να συλλέξει ολόκληρο τον κατανεμημένο πίνακα σε μία περιοχή εργασίας. αυτό επιτυγχάνεται με τη συνάρτηση gather.

P>> combined = gather(whole)

Πρέπει να σημειωθεί, όμως, το γεγονός ότι η χρήση της συνάρτησης gather συλλέγει τον κατανεμημένο πίνακα στις περιοχές εργασίας όλων των εργαστηρίων. Εάν, ο χρήστης επιθυμεί ο κατανεμημένος πίνακας να συλλεχθεί στο χώρο εργασίας ενός μόνο εργαστηρίου θα πρέπει να πληκτρολογήσει την αντίστοιχη εντολή, όπως παρουσιάζεται παρακάτω.

P>> combined2 = gather(whole, 1)

Ο χρήστης, έχει τη δυνατότητα να αντιγράψει οποιονδήποτε πίνακα ή μεταβλητή από ένα συγκεκριμένο εργαστήριο στο χώρο εργασίας του Matlab. Αυτή η δυνατότητα χρησιμοποιείται, συνήθως, σε περιπτώσεις όπου ο χρήστης επιθυμεί να απεικονίσει κάποια δεδομένα (plot). Η απεικόνιση δεδομένων μπορεί να πραγματοποιηθεί στον υπολογιστή πελάτη, δεν μπορεί, όμως, να πραγματοποιηθεί στα εργαστήρια. Η παρακάτω εντολή αντιγράφει τον πίνακα combined2 του εργαστηρίου 1 στην περιοχή εργασίας του Matlab.

pmode lab2client combined2 1

Πρέπει να σημειωθεί ότι η παραπάνω εντολή εκτελείται στο Παράθυρο Εντολών του Matlab.

Πολλές συναρτήσεις, οι οποίες χρησιμοποιούνται για τους πίνακες του Matlab μπορούν να χρησιμοποιηθούν και για τους κατανεμημένους πίνακες. Για παράδειγμα, η συνάρτηση eye δημιουργεί ένα μοναδιαίο πίνακα. με τις εντολές που ακολουθούν μπορεί ο χρήστης να δημιουργήσει και να κατανέμει έναν μοναδιαίο πίνακα.

>>P distobj = darray()

>>P I = eye(7, distobj)

Εάν απαιτείται η κατανομή του πίνακα σε διαφορετική διάσταση, τότε ο χρήστης μπορεί να χρησιμοποιήσει τη συνάρτηση redistribute.

P>> I = redistribute(I, 1)

Για να τερματιστεί η παράλληλη λειτουργία και να επιστρέψει ο χρήστης στο Matlab αρκεί να εκτελέσει την παρακάτω εντολή.

P>> pmode exit

Οι κυριότερες εντολές για τη δημιουργία του κώδικα της διεργασίας

Η διαβίβαση δεδομένων μεταξύ των εργαστηρίων μίας παράλληλης εργασίας πραγματοποιείται με τη χρήση των κατάλληλων συναρτήσεων. Οι κυριότερες από αυτές τις συναρτήσεις είναι οι εξής:

- **numlabs**
  *Ε*πιστρέφει το συνολικό αριθμό των εργαστηρίων που λειτουργούν παράλληλα στην τρέχουσα εργασία

- **labindex:**
  Χρησιμοποιείται για τη διευκρίνηση του αριθμού που χαρακτηρίζει κάθε εργαστήριο.

- **labSend(data, destination, tag)**
  Χρησιμοποιείται για την αποστολή δεδομένων σε προκαθορισμένο παραλήπτη – εργαστήριο. Η μεταβλητή data είναι τα δεδομένα ου στέλνονται σε συγκεκριμένο εργαστήριο. Η μεταβλητή destination είναι ο αριθμός (labindex) του εργαστηρίου που θα λάβει τα δεδομένα. Η μεταβλητή tag είναι μια ετικέτα που μπορεί να διευκρινίσει ποιο δεδομένο αποστέλλεται.

- **data = labReceive(source,tag)**
  Χρησιμοποιείται για τη λήψη δεδομένων από έναν προκαθορισμένο αποτολέα – εργαστήριο. Η μεταβλητή data είναι τα δεδομένα που λαμβάνονται από ένα συγκεκριμένο εργαστήριο. Η μεταβλητή source είναι ο αριθμός (labindex) του εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή tag είναι ένα επίθεμα το οποίο μπορεί να διευκρινίσει ποιο δεδομένο παραλαμβάνεται.

- **labBroadcast(senderlab, data)**
  Χρησιμοποιείται για την αποστολή δεδομένων από το κεντρικό εργαστήριο προς τα υπόλοιπα. Η μεταβλητή senderlab είναι ο αριθμός (labindex) του κεντρικού εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή data είναι τα δεδομένα που στέλνονται από το κεντρικό εργαστήριο. Για να λάβει ένα εργαστήριο δεδομένα που έχουν σταλεί με τη χρήση της εντολή labBroadcast πρέπει να χρησιμοποιήσει την ίδια την εντολή labBroadcast (senderlab) και όχι την εντολή labReceive.

- **is_data_availiable = labProbe(source, tag)**
  Η εντολή αυτή χρησιμεύει στον έλεγχο των εισερχόμενων μηνυμάτων χωρίς να προηγηθεί η λήψη τους. Η μεταβλητή is_data_availiable είναι ένας αριθμός (0 ή 1), ο οποίος δείχνει εάν κάποιο μήνυμα είναι έτοιμο για να παραληφθεί. Η μεταβλητή source είναι ο αριθμός (labindex) του εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή tag είναι ένα επίθεμα το οποίο μπορεί να διευκρινίσει ποιο δεδομένο παραλαμβάνεται.

- **labBarrier**
  Η εντολή αυτή αποτελεί την πιο απλή μορφή συλλογικής επικοινωνίας. Δεν σχετίζεται με την ανταλλαγή δεδομένων, αλλά αντίθετα παρέχει έναν μηχανισμό για το συγχρονισμό όλων των εργαστηρίων που την καλούν. Κάθε εργαστήριο σταματάει προσωρινά έως ότου όλες οι διεργασίες να καλέσουν τη συνάρτηση αυτή.

Παραδείγματα

Παράδειγμα 1

Στο παράδειγμα αυτό το εργαστήριο 1 μεταδίδει (broadcast) στα υπόλοιπα εργαστήρια έναν πίνακα. Έπειτα, κάθε εργαστήριο, συμπεριλαμβανομένου και του εργαστηρίου 1, υπολογίζει το άθροισμα της στήλης του πίνακα αυτού που ο αριθμός της ισούται με τον αριθμό του εργαστηρίου. Τα εργαστήρια, εκτός του εργαστηρίου 1, στέλνουν (labSend) στο εργαστήριο 1 το αποτέλεσμα του αθροίσματος. Τέλος, το εργαστήριο 1 λαμβάνει (labReceive) τα αθροίσματα από τα υπόλοιπα εργαστήρια και υπολογίζει το άθροισμα των παραπάνω αποτελεσμάτων. Επομένως, μόνο στο εργαστήριο 1 υπάρχει η τιμή του τελικού αποτελέσματος.

```
if labindex == 1
        % Broadcast magic square to other labs
        A = labBroadcast(1,magic(numlabs));
        total_sum = sum(A(:,labindex));
        % Calculate total sum by combining column sum from all labs
        for otherLab = 2:numlabs
                total_sum = total_sum + labReceive(otherLab);
        end
else
        % Receive Broadcast on other labs
        A = labBroadcast(1);
        % Calculate sum of column identified by labindex for this lab
```

```
    column_sum = sum(A(:,labindex));
    labSend(column_sum, 1);
end
```

Παράδειγμα 2

Στο παράδειγμα αυτό δημιουργούνται δύο πίνακες (A και B). Το εργαστήριο 1 μεταδίδει (broadcast) στα υπόλοιπα εργαστήρια τον πίνακα A και στέλνει (labSend) σε κάθε εργαστήριο ξεχωριστά τη στήλη του πίνακα B που ο αριθμός της ισούται με τον αριθμό του εργαστηρίου. Έπειτα, κάθε εργαστήριο, συμπεριλαμβανομένου και του εργαστηρίου 1, υπολογίζει το γινόμενο του πίνακα A με τη στήλη του πίνακα B που έχει διαθέσιμη. Τα εργαστήρια, εκτός του εργαστηρίου 1, στέλνουν (labSend) στο εργαστήριο 1 το αποτέλεσμα του γινομένου, που είναι ένας πίνακας στήλη. Τέλος, το εργαστήριο 1 λαμβάνει (labReceive) τα γινόμενα από τα υπόλοιπα εργαστήρια και συνθέτει τον τελικό πίνακα C που είναι και το τελικό αποτέλεσμα του πολλαπλασιασμού των πινάκων A και B.

```
if labindex == 1
    % Broadcast magic square to other labs
    A = labBroadcast(1,magic(numlabs));
    % Create B
    B = magic(numlabs);
    % Send appropriate column of B to other labs
    for otherLab = 2:numlabs
        labSend(B(:, otherLab), otherLab);
    end
    % Calculate appropriate column of C
    C(:,1) = A(:,:) * B(:, labindex);
    % Receive appropriate column of C from other labs
    for otherLab = 2:numlabs
        C(:,otherLab) = labReceive(otherLab);
```

```
        end
else
        % Receive Broadcast on other labs
        A = labBroadcast(1);
        % Calculate appropriate column of C
        C(:,:) = A(:,:) * labReceive(1);
        % Send appropriate column of C to master
        labSend(C, 1);
end
```