ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Παραλληλοποίηση Αλγόριθμων Γραμμικής Άλγεβρας για Αρχιτεκτονικές Υψηλής Επίδοσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Α. Καλαβρή

Επιβλέπων : Κοζύρης Νεκτάριος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2010

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Παραλληλοποίηση Αλγόριθμων Γραμμικής Άλγεβρας για Αρχιτεκτονικές Υψηλής Επίδοσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Α. Καλαβρή

Επιβλέπων : Κοζύρης Νεκτάριος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την …….

…….....	…….....	…….....
Κοζύρης Νεκτάριος Αναπληρωτής Καθηγητής Ε.Μ.Π.	Κορρές Γεώργιος Αναπληρωτής Καθηγητής Ε.Μ.Π.	Φωτάκης Δημήτριος Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2010

.....

Βασιλική Α. Καλαβρή

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η μελέτη, υλοποίηση και παραλληλοποίηση αλγόριθμων γραμμικής άλγεβρας για αρχιτεκτονικές υψηλής επίδοσης. Η εργασία περιλαμβάνει θεωρητικό και προγραμματιστικό μέρος, ενώ ακολουθούν πειραματικά αποτελέσματα και μελέτη της επίδοσης των παράλληλων υλοποιήσεων.

Το θεωρητικό μέρος της εργασίας περιλαμβάνει κατ' αρχήν μελέτη των κυριότερων παράλληλων αρχιτεκτονικών. Περιγράφονται τα χαρακτηριστικά των αρχιτεκτονικών, τα διαθέσιμα προγραμματιστικά μοντέλα, τα πλεονεκτήματα και τα μειονεκτήματα κάθε αρχιτεκτονικής. Επίσης, πραγματοποιείται μία σύγκριση ανάμεσα στα δύο κύρια μοντέλα, κατανεμημένης και μοιραζόμενης μνήμης, ώστε να επεξηγηθεί η επιλογή του πρώτου, ως το μοντέλο που χρησιμοποιήθηκε στην παρούσα εργασία. Στη συνέχεια, παρέχεται το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση των μεθόδων επίλυσης συστημάτων γραμμικών εξισώσεων. Παρουσιάζονται οι σημαντικότερες άμεσες και επαναληπτικές μέθοδοι επίλυσης συστημάτων, καθώς και οι εφαρμογές στις οποίες χρησιμοποιούνται αυτές οι μέθοδοι.

Το προγραμματιστικό κομμάτι της εργασίας περιλαμβάνει την υλοποίηση δύο αλγόριθμων σειριακά, και στη συνέχεια την υλοποίηση παράλληλων εκδόσεων. Ο πρώτος αλγόριθμος αφορά στην εκτίμηση κατάστασης συστημάτων ηλεκτρικής ενέργειας με τη μέθοδο των ελαχίστων τετραγώνων και ο δεύτερος αποτελεί την υλοποίηση της μεθόδου Conjugate Gradient για την επίλυση γραμμικών συστημάτων. Η υλοποίηση των προγραμμάτων έγινε σε γλώσσα C, με τη χρήση των προγραμματιστικών εργαλείων MPI, PETSc και BLAS, τα οποία περιγράφονται συνοπτικά στην εργασία.

Τέλος, πραγματοποιήθηκαν μετρήσεις για διάφορα μεγέθη προβλημάτων σε συστοιχία υπολογιστικών κόμβων. Από τις μετρήσεις αυτές, κατασκευάστηκαν διαγράμματα, τα οποία βοηθούν στην εξαγωγή συμπερασμάτων για την απόδοση των παράλληλων υλοποιήσεων, αλλά και για την κατανομή του χρόνου εκτέλεσης των προγραμμάτων σε χρόνο υπολογισμών και χρόνο επικοινωνίας.

Λέξεις Κλειδιά

Παράλληλες αρχιτεκτονικές, αρχιτεκτονική κατανεμημένης μνήμης, αρχιτεκτονική μοιραζόμενης μνήμης, επεκτασιμότητα συστήματος, επίλυση γραμμικών συστημάτων, εκτίμηση κατάστασης, Conjugate Gradient, μέθοδοι Krylov, MPI, PETSc, BLAS.

Abstract

The scope of this diploma thesis is the study, implementation and parallelization of linear algebra algorithms for high scale architectures. This thesis consists of a theoretical and programming part, while experimental results are also available in order to study the efficiency of the parallel implementations.

The theoretical part contains study of the main parallel architectures. There is a description of the characteristics of the architectures, the available programming models, as well as the advantages and disadvantages of each architecture. Moreover, a comparison is conducted between the two fundamental models of distributed and shared memory, in order to explain the choice of using the distributed memory model for the implementations. Following, the necessary theory of the methods for solving systems of linear equations is provided, including the most important direct and iterative methods, as well as the corresponding applications.

The programming part contains the implementation of two serial algorithms and their parallel versions. The first algorithm refers to the state estimation of electrical systems using the least squares method and the second algorithm is an implementation of the Conjugate Gradient method for solving linear systems. The programs were implemented using the C programming language and the MPI, PETSc and BLAS programming tools, which are described briefly inside this document.

Finally, a set of measurements is provided for different problem sizes, conducted on a cluster of processing nodes. These measurements lead to the construction of diagrams used to extract conclusions on the performance of the parallel implementations and the distribution of the execution time to computation and communication among the processors.

Key Words

Parallel architectures, distributed memory architecture, shared memory architecture, system scalability, linear systems solving, state estimation, Conjugate Gradient, Krylov methods, MPI, PETSc, BLAS.

Ευχαριστίες
Ευχαριστώ θερμά τους επιβλέποντες της διπλωματικής εργασίας κ. Κοζύρη Νεκτάριο και κ. Γεώργιο Κορρέ, Αναπληρωτές Καθηγητές Ε.Μ.Π., καθώς και τον κ. Γκούμα Γεώργιο για τη βοήθεια και καθοδήγηση που μου προσέφεραν κατά τη διάρκεια της υλοποίηση της παρούσας διπλωματικής εργασίας. Θα ήθελα ακόμη να ευχαριστήσω τους συμφοιτητές και συνεργάτες Γαλήνα Ευάγγελο και Τζαβέλλα Αναστάσιο για την υποστήριξη και τη βοήθεια τους καθ' όλη τη διάρκεια της εκπόνησης της εργασίας, καθώς και την οικογένεια μου για την αμέριστη συμπαράσταση τους.

Περιεχόμενα

Κεφάλαιο 1ο: Παράλληλες Αρχιτεκτονικές

1.1 Εισαγωγή

Η ανάγκη για όλο και πιο αποδοτικές εφαρμογές γεννήθηκε σχεδόν ταυτόχρονα με την εμφάνιση των υπολογιστικών συστημάτων. Η πρόοδος που σημειώνεται στον τομέα του υλικού των υπολογιστών δίνει δυνατότητες για την ανάπτυξη νέων λειτουργιών στις εφαρμογές και κάνει ακόμη πιο σημαντική την ανάγκη για την υλοποίηση αρχιτεκτονικών, ικανών να ανταποκριθούν στις σύγχρονες απαιτήσεις. Το γεγονός αυτό έστρεψε τους ερευνητές στο σχεδιασμό παράλληλων αρχιτεκτονικών, οι οποίες μπορούν να ικανοποιήσουν τις πιο απαιτητικές από αυτές τις εφαρμογές. Πριν από την εμφάνιση των παράλληλων αρχιτεκτονικών, οι απαιτήσεις για επίδοση ικανοποιούνταν δίνοντας βάρος στο σχεδιασμό πολύπλοκων επεξεργαστικών κυκλωμάτων. Σήμερα, για την επίτευξη ακόμη μεγαλύτερης επίδοσης, η πρώτη επιλογή είναι η χρήση πολλών επεξεργαστών, και οι πιο απαιτητικές εφαρμογές γράφονται πλέον ως παράλληλα προγράμματα.

Τα μεγάλα επιτεύγματα που έχουν σημειωθεί τα τελευταία χρόνια στον τομέα της τεχνολογίας και της αρχιτεκτονικής υπολογιστών μαρτυρούν τη γενικότερη τάση που επικρατεί στον εμπορικό και επιστημονικό κόσμο. Η τάση αυτή βρίσκει δελεαστική την ιδέα των παράλληλων αρχιτεκτονικών θεωρώντας πιο δύσκολο «να περιμένουμε μέχρι ο ένας επεξεργαστής να γίνει αρκετά γρήγορος». Επιπλέον, η έως τώρα εμπειρία δείχνει ότι τα κρίσιμα ζητήματα που έχει να αντιμετωπίσει ο σχεδιαστής μιας παράλληλης αρχιτεκτονικής είναι θεμελιωδώς όμοια με αυτά που αντιμετωπίζει και κατά τη σχεδίαση μιας σειριακής. Συγκεκριμένα, ζητήματα όπως η κατανομή των πόρων στις λειτουργικές μονάδες που εκτελούν τους υπολογισμούς, η εκμετάλλευση της χωρικής τοπικότητας από τις κρυφές μνήμες και ο σχεδιασμός δικτύων που μεγιστοποιούν το εύρος της επικοινωνίας, είναι βασικά στο σχεδιασμό οποιουδήποτε τύπου αρχιτεκτονικής.

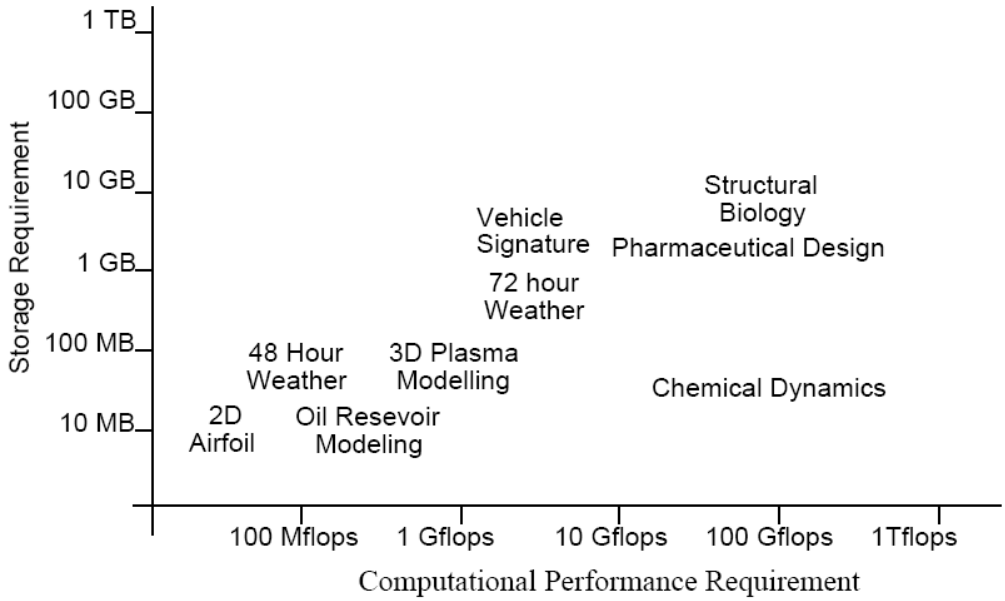
Από την άλλη πλευρά, ήταν φυσικό να προκύψουν και νέα ζητήματα σχεδιασμού. Όσο τα υπολογιστικά συστήματα στήριζαν όλη τους την απόδοση στην ύπαρξη μιας μοναδικής, ισχυρής κεντρικής μονάδας επεξεργασίας, η μόνη δυνατότητα παράλληλης επεξεργασίας ήταν μέσω της ταυτόχρονης λειτουργίας του επεξεργαστή και των συσκευών εισόδου/εξόδου σε επίπεδο εντολών (Instruction Level Parallelism). Η εισαγωγή, όμως, του παραλληλισμού στην εκτέλεση των εντολών με τη χρήση πολλών μονάδων επεξεργασίας συνοδεύτηκε από την ανάγκη για λήψη αποφάσεων πάνω σε θέματα, όπως:

- Τον τρόπο επικοινωνίας και συγχρονισμού των επεξεργαστικών μονάδων
- Τον έλεγχο των ταυτόχρονων αναφορών στην κοινή μνήμη και τις συσκευές εισόδου/εξόδου
- Την έκφραση της παραλληλίας μέσω ενός προγραμματιστικού μοντέλου.

1.1.1 Επιστημονικές Εφαρμογές

Η απαίτηση για αυξανόμενη επίδοση είναι ιδιαίτερα εμφανής στην περίπτωση επιστημονικών εφαρμογών. Κατ' αρχήν, στον τομέα της επιστήμης, τα υπολογιστικά συστήματα χρησιμοποιούνται για προσομοιώσεις φυσικών και άλλων φαινομένων, τα οποία είναι αδύνατο να παρατηρηθούν με εμπειρικά μέσα. Τυπικά παραδείγματα αποτελούν η μοντελοποίηση κλιματικών αλλαγών, η εξέλιξη ουράνιων σωμάτων, η ατομική δομή των υλικών, η απόδοση της καύσης σε μια μηχανή, η συμπεριφορά μικροσκοπικών ηλεκτρονικών συσκευών και άλλα. Η μοντελοποίηση με τη χρήση υπολογιστών επιτρέπει τη διεξαγωγή ανάλυσης σε βάθος, με χαμηλό κόστος, μέσω μεθόδων προσομοίωσης. Στο Σχήμα 1.1 παρουσιάζεται, με γραφικό τρόπο, η εξέλιξη

των υπολογιστικών απαιτήσεων σε διάφορους επιστημονικούς τομείς. Είναι φανερό ότι ακόμη και μια δραματική αύξηση στην απόδοση του ενός επεξεργαστή δεν θα μπορούσε να ανταποκριθεί στις ανάγκες των σύγχρονων και μελλοντικών επιστημονικών εφαρμογών. Η παρατήρηση αυτή καθιστά απαραίτητη την ανάπτυξη παράλληλων αρχιτεκτονικών για επιστημονικές εφαρμογές.



Σχήμα 1.1: Υπολογιστικές απαιτήσεις επιστημονικών εφαρμογών
(Πηγή: Committee of Physical, Mathematical and Engineering sciences, Federal Office of Science and Technology Policy, 1003)

Οι παράλληλες αρχιτεκτονικές είναι ο στυλοβάτης κάθε τομέα επιστημονικών υπολογισμών, συμπεριλαμβανομένων της φυσικής, της χημείας, της επιστήμης των υλικών, της βιολογίας, της αστρονομίας και άλλων. Η εφαρμογή των εργαλείων μοντελοποίησης που βασίζονται σε παράλληλες αρχιτεκτονικές, είναι πλέον απαραίτητη σε πολλές βιομηχανίες, όπως την πετρελαϊκή (μοντελοποίηση αποθεμάτων), την αυτοκινητοβιομηχανία (προσομοίωση συγκρούσεων, απόδοση καύσης), την αεροναυπηγική (απόδοση μηχανών, μοντελοποίηση μηχανικών δομών, ηλεκτρομαγνητισμός), τη φαρμακευτική (μοντελοποίηση ατομικής δομής) και άλλες. Σχεδόν σε όλες αυτές τις εφαρμογές, υπάρχει η ανάγκη για οπτικοποίηση των αποτελεσμάτων, η οποία από τη φύση της είναι μια ιδιαίτερα απαιτητική λειτουργία η οποία προσφέρεται για παράλληλο υπολογισμό.

1.1.2 Εμπορικές Εφαρμογές

Εκτός από τον επιστημονικό χώρο, και ο τομέας των εμπορικών εφαρμογών έχει στραφεί προς τις παράλληλες αρχιτεκτονικές τα τελευταία χρόνια. Αν και ο βαθμός του παραλληλισμού δεν είναι τόσο μεγάλος όπως στις επιστημονικές εφαρμογές, η χρήση του παραλληλισμού είναι αρκετά διαδεδομένη. Οι πολυεπεξεργαστές έχουν εισβάλλει στις εμπορικές εφαρμογές από τα μέσα της δεκαετίας του 1960. Στο χώρο αυτό, η ταχύτητα του υπολογιστικού συστήματος μεταφράζεται στην εταιρική έκταση που μπορεί να υποστηριχθεί από το σύστημα. Η σχέση ανάμεσα στην απόδοση του συστήματος και στην έκταση της εταιρικής υποστήριξης γίνεται

ξεκάθαρη αν θεωρήσουμε ένα σύστημα το οποίο εξυπηρετεί ένα φορτίο από συναλλαγές χρηστών, οι οποίοι εκτελούν ερωτήματα σε μια μεγάλη βάση δεδομένων. Στην περίπτωση αυτή, η απόδοση του συστήματος αναλογεί στον όγκο των χρηστών που μπορεί να εξυπηρετήσει ταυτόχρονα το σύστημα.

Η μετάβαση στην παράλληλη επεξεργασία είναι ακόμα σε εξέλιξη στον εμπορικό τομέα. Τυπικά, οι εμπορικές εφαρμογές στοχεύουν σε μεσαίου μεγέθους πολυεπεξεργαστικά συστήματα, τα οποία κυριαρχούν στην αγορά των εξυπηρετητών. Στον εμπορικό κόσμο, όλοι οι μεγάλοι κατασκευαστές βάσεων δεδομένων υποστηρίζουν παράλληλα μηχανήματα για τα προϊόντα τους. Αρκετοί από αυτούς παρέχουν, επίσης, εκδόσεις παράλληλων μηχανημάτων και συστημάτων σταθμών εργασίας πάνω από γρήγορα δίκτυα, τα επονομαζόμενα clusters.

Τέλος, ακόμη και στον προσωπικό επιτραπέζιο υπολογιστή συναντάμε την ανάγκη για πολυεπεξεργασία. Ένας μέσος χρήστης θα επιθυμεί συχνά πολλαπλές διεργασίες να εκτελούνται ταυτόχρονα στον προσωπικό του υπολογιστή, έχοντας αρκετά ενεργά παράθυρα ή νήματα να εκτελούνται στο παρασκήνιο. Όλες αυτές οι τάσεις και οι αυξανόμενες απαιτήσεις του μέσου χρήστη, έχουν οδηγήσει στο σχεδιασμό παράλληλων αρχιτεκτονικών μικρότερης κλίμακας για επιτραπέζιους υπολογιστές.

1.2 Μορφή και οργάνωση Παράλληλων Αρχιτεκτονικών

Ιστορικά, έχουν χρησιμοποιηθεί αρκετές διακριτές αρχιτεκτονικές αρχές για την κατασκευή παράλληλων συστημάτων. Ωστόσο, παρακολουθώντας την εξέλιξη των παράλληλων αρχιτεκτονικών, είναι ξεκάθαρο ότι ο σχεδιασμός έχει επηρεαστεί σημαντικά από παρόμοιες τεχνολογικές τάσεις και απαιτήσεις εφαρμογών. Έτσι, όλες οι παράλληλες αρχιτεκτονικές είναι στηριγμένες πάνω σε κοινές θεμελιώδεις αρχές.

Αν σκεφτούμε έναν παράλληλο υπολογιστή ως «μια συλλογή από επεξεργαστικά στοιχεία τα οποία επικοινωνούν μεταξύ τους και συνεργάζονται για να λύσουν προβλήματα με ταχύτητα» (Almansi ακαι Gottlieb 1989), μπορούμε εύκολα να συνειδητοποιήσουμε ότι μια παράλληλη αρχιτεκτονική είναι η επέκταση μιας συμβατικής αρχιτεκτονικής, στην οποία έχουν απλώς προστεθεί τα θέματα της επικοινωνίας και της συνεργασίας ανάμεσα στις υπολογιστικές μονάδες. Η αρχιτεκτονική ενός υπολογιστικού συστήματος έχει δύο πλευρές. Η μία αναφέρεται στο αφαιρετικό επίπεδο στο οποίο ορίζονται τα όρια μεταξύ λογισμού/υλικού και χρήστη/συστήματος. Η άλλη πλευρά αναφέρεται στην δομή η οποία πραγματοποιεί αυτή την αφαίρεση έτσι ώστε να επιτευχθεί υψηλή επίδοση με το χαμηλότερο δυνατό κόστος. Μια αρχιτεκτονική βασισμένη στην επικοινωνία (όπως οι περισσότερες παράλληλες αρχιτεκτονικές), έχει επίσης τις δύο αυτές πλευρές. Ορίζει τις βασικές διαδικασίες επικοινωνίας και συγχρονισμού και αναφέρεται στις δομές οργάνωσης που μπορούν να υλοποιήσουν τις διαδικασίες αυτές.

Τα μοντέλα παράλληλων αρχιτεκτονικών που βασίζονται στην επικοινωνία μπορούν να ταξινομηθούν στις εξής τρεις κατηγορίες:

- *Κατανεμημένης Μνήμης (distributed memory):* Μηνύματα που περιέχουν πληροφορία μεταφέρονται από έναν συγκεκριμένο αποστολές σε έναν συγκεκριμένο χρήστη. Το γεγονός της αποστολής ή λήψης της πληροφορίας ορίζεται ξεκάθαρα και καθορίζει το συντονισμό των ατομικών ενεργειών. Σε αυτό το μοντέλο, δεν υπάρχει κοινός χώρος μνήμης, όπου να έχουν πρόσβαση όλες οι επεξεργαστικές μονάδες.
- *Μοιραζόμενης Μνήμης(shared memory):* Στο μοντέλο αυτό υπάρχει ένας κοινός χώρος αποθήκευσης στον οποίο έχουν πρόσβαση όλες οι

επεξεργαστικές μονάδες. Κάθε μονάδα μπορεί να γράφει στο χώρο αυτό και να διαβάζει πληροφορίες που έγραψαν άλλες μονάδες.

- *Παράλληλων δεδομένων (data parallel):* Εδώ υπάρχει μια πιο πειθαρχημένη μορφή συνεργασίας, όπου διάφορες μονάδες εκτελούν ταυτόχρονες ενέργειες σε ξεχωριστά στοιχεία ενός συνόλου δεδομένων και στη συνέχεια, ανταλλάσσουν πληροφορίες με όλες τις άλλες μονάδες, πριν συνεχίσουν την εργασία τους. Η καθολική αναδιοργάνωση των δεδομένων επιτυγχάνεται είτε μέσω προσβάσεων σε έναν κοινό χώρο διευθύνσεων, είτε μέσω της ανταλλαγής μηνυμάτων.

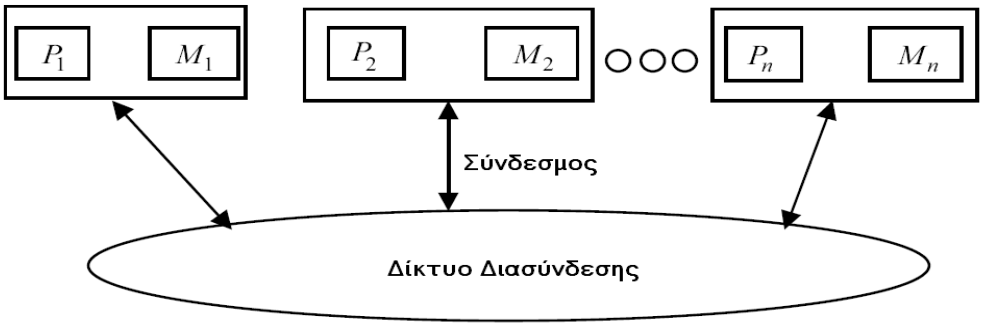
Στη συνέχεια, θα εξετάσουμε σε μεγαλύτερο βάθος τα δύο πρώτα μοντέλα, κατανεμημένης και μοιραζόμενης μνήμης, τα οποία είναι και έχουν επικρατήσει στη σχεδίαση των σύγχρονων παράλληλων αρχιτεκτονικών. Στο τέλος αυτού του κεφαλαίου, πραγματοποιούμε μια σύγκριση αυτών των δύο μοντέλων, παρουσιάζοντας συνοπτικά τα πλεονεκτήματα και τα μειονεκτήματα του καθενός.

1.2.1 Αρχιτεκτονική Κατανεμημένης Μνήμης

Ένα σύστημα κατανεμημένης μνήμης συνδυάζει την τοπική μνήμη και τον επεξεργαστή σε κάθε κόμβο του δικτύου διασύνδεσης. Δεν υπάρχει καθολική μνήμη, οπότε είναι αναγκαία η μεταφορά δεδομένων ανάμεσα σε τοπικές μνήμες μέσω της αποστολής μηνυμάτων. Αυτό υλοποιείται συνήθως από ζεύγη εντολών αποστολής/λήψης οι οποίες πρέπει να γραφούν από τον προγραμματιστή. Στο Σχήμα 1.2 φαίνεται μία υψηλού επιπέδου περιγραφή ενός τέτοιου συστήματος. Κάθε επεξεργαστής έχει πρόσβαση στη δική του τοπική μνήμη και μπορεί να επικοινωνεί με τους άλλους επεξεργαστές μέσω του δικτύου διασύνδεσης. Αυτά τα συστήματα διαδέχτηκαν τελικά διαδικτυακά συστήματα όπου οι επεξεργαστικοί κόμβοι μπορεί να είναι κόμβοι ενός cluster, εξυπηρετητές, πελάτες ή κόμβοι σε κάποιο μεγαλύτερο πλέγμα.

Η αρχιτεκτονική κατανεμημένης μνήμης χρησιμοποιείται για τη μεταφορά δεδομένων ανάμεσα σε ένα σύνολο επεξεργαστών χωρίς την ανάγκη για μία καθολική μνήμη. Η ιδέα είναι ότι κάθε επεξεργαστής έχει τη δική του τοπική μνήμη και επικοινωνεί με τους άλλους επεξεργαστές χρησιμοποιώντας μηνύματα. Η εξάλειψη της ανάγκης για μία μεγάλη και καθολική μνήμη καθώς και της απαίτησής της για συγχρονισμό προσδίδει στην αρχιτεκτονική κατανεμημένης μνήμης ένα μεγάλο πλεονέκτημα σε σχέση με τις αρχιτεκτονικές μοιραζόμενης μνήμης.

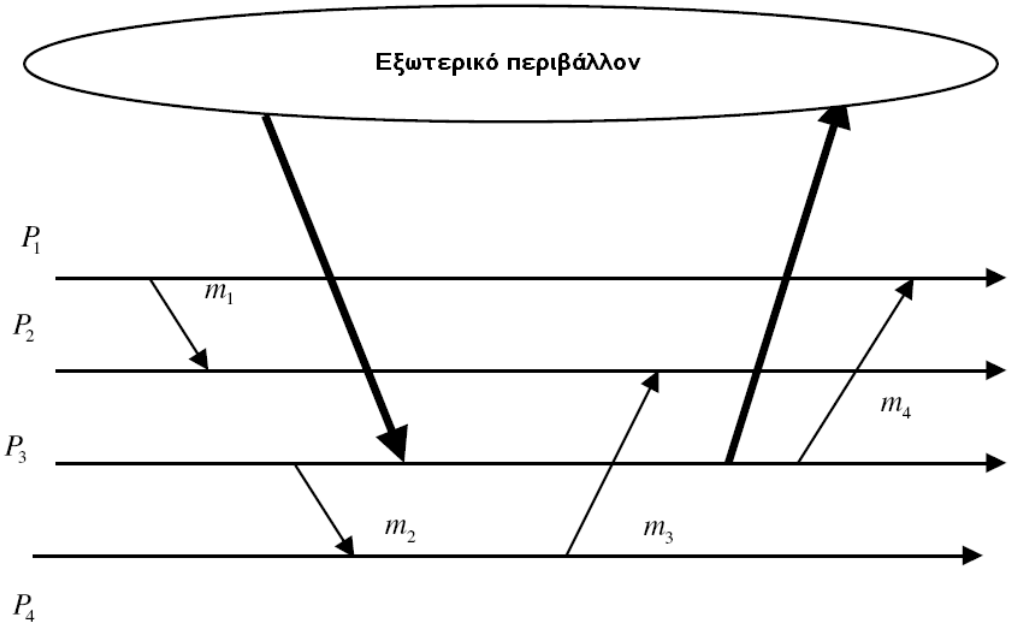
Στο Σχήμα 1.2 φαίνονται τα βασικά συστατικά μιας πολυεπεξεργαστικής αρχιτεκτονικής κατανεμημένης μνήμης. Υπάρχουν n κόμβοι αριθμημένοι από N_1 ως N_n όπου κάθε κόμβος N_i αποτελείται από έναν επεξεργαστή P_i και την τοπική μνήμη M_i. Κάθε επεξεργαστής έχει, λοιπόν, το δικό του χώρο διευθύνσεων. Οι κόμβοι επικοινωνούν μεταξύ τους μέσω συνδέσμων (ονομάζονται και εξωτερικά κανάλια) και μέσω του δικτύου διασύνδεσης που είναι συνήθως στατικό.



Σχήμα 1.2: Σύστημα Κατανεμημένης Μνήμης

Σε μια τέτοια αρχιτεκτονική, η εκτέλεση ενός προγράμματος διαχωρίζεται σε παράλληλες ταυτόχρονες διεργασίες όπου κάθε μία εκτελείται σε ξεχωριστό επεξεργαστή. Αν ο αριθμός των διεργασιών είναι μεγαλύτερος από τον αριθμό των επεξεργαστών, σε κάθε επεξεργαστή θα πρέπει να εκτελεστούν περισσότερες από μια διεργασίες με κάποια χρονοδρομολόγηση. Οι διεργασίες που τρέχουν στον ίδιο επεξεργαστή χρησιμοποιούν τα ονομαζόμενα εσωτερικά κανάλια για να ανταλλάξουν μηνύματα μεταξύ τους. Τα δεδομένα που ανταλλάσσονται μεταξύ επεξεργαστών δεν είναι μοιραζόμενα αλλά γίνεται αντιγραφή αυτών με αποστολή και παραλαβή μηνυμάτων. Ένα σημαντικό πλεονέκτημα αυτής της μορφής ανταλλαγής μηνυμάτων είναι ότι δεν απαιτεί μηχανισμούς συγχρονισμού, όπως σηματοφορείς, γεγονός που βελτιώνει αισθητά την επίδοση.

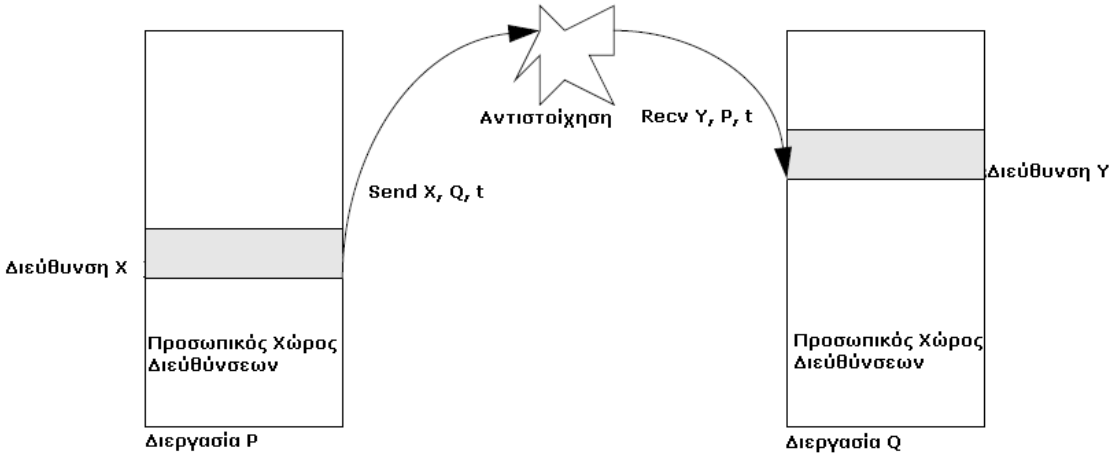
Στο Σχήμα 1.3 φαίνεται ένα παράδειγμα ενός συστήματος κατανεμημένης μνήμης που αποτελείται από τέσσερεις διεργασίες. Στο σχήμα αυτό οι οριζόντιες γραμμές παριστάνουν την εκτέλεση κάθε διεργασίας και οι γραμμές ανάμεσα στις διεργασίες παριστάνουν τα μηνύματα που ανταλλάσσουν μεταξύ τους. Το *μήνυμα* ορίζεται ως η λογική μονάδα για τη διακομβική επικοινωνία. Θεωρείται ως μια συλλογή συσχετιζόμενης πληροφορίας που μεταφέρεται σαν μια οντότητα. Ένα σύστημα κατανεμημένης μνήμης επικοινωνεί με το εξωτερικό περιβάλλον λαμβάνοντας μηνύματα εισόδου και στέλνοντας μηνύματα εξόδου.



Σχήμα 1.3: Παράδειγμα Συστήματος Κατανεμημένης Μνήμης

1.2.1.1 Προγραμματιστικά Μοντέλα Ανταλλαγής Μηνυμάτων

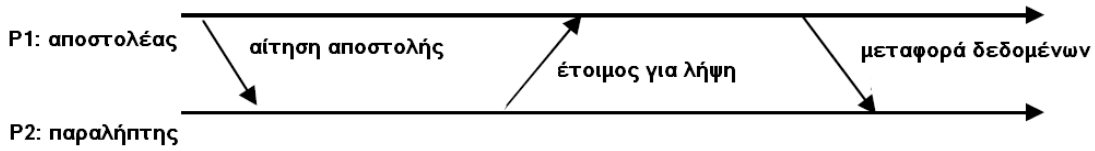
Μια αρχιτεκτονική ανταλλαγής μηνυμάτων χρησιμοποιεί ένα σύνολο από κύριες λειτουργίες που επιτρέπουν στις διεργασίες να επικοινωνούν μεταξύ τους. Σε αυτές περιλαμβάνονται οι λειτουργίες *send*, *receive*, *broadcast* και *barrier*. Η λειτουργία *send* παίρνει δεδομένα από τη μνήμη του κόμβου-αποστολέα και τα στέλνει στον κόμβο-προορισμό. Η λειτουργία *receive* δέχεται ένα μήνυμα από τον αποστολέα και το αποθηκεύει στη μνήμη του παραλήπτη. Το βασικό προγραμματιστικό μοντέλο που χρησιμοποιείται στις αρχιτεκτονικές ανταλλαγής μηνυμάτων βασίζεται στην ιδέα του συνδυασμού μιας αίτησης *send* από έναν επεξεργαστή με μια αίτηση *receive* από κάποιον άλλο. Η λογική αυτή παρουσιάζεται στο Σχήμα 1.4. Η μεταφορά των δεδομένων, από τον ένα χώρο διευθύνσεων στον άλλο, λαμβάνει χώρα όταν μια λειτουργία send προς κάποια διεργασία αντιστοιχιστεί με μια λειτουργία receive από αυτή τη διεργασία. Σε ένα τέτοιο σχήμα, ο επεξεργαστής που εκτελεί την αποστολή περιμένει μέχρι ο άλλος επεξεργαστής να εκτελέσει τη λειτουργία λήψης και στη συνέχεια ξεκινά η μεταφορά δεδομένων. Στην περίπτωση αυτή λέμε ότι συμβαίνει *blocking-send* και *blocking-receive.*



Σχήμα 1.4: Αντιστοίχιση λειτουργιών Send και Receive

Η υλοποίηση της αποστολής και λήψης ανάμεσα σε διεργασίες απαιτεί ένα πρωτόκολλο τριών φάσεων όπως φαίνεται στο Σχήμα 1.4. Στην περίπτωση αυτή, η διεργασία-αποστολέας στέλνει μια αίτηση αποστολής στη διεργασία-παραλήπτη. Η τελευταία αποθηκεύει την αίτηση και στέλνει ένα μήνυμα απάντησης. Όταν εκτελεστεί η λειτουργία λήψης, η διεργασία-αποστολέας λαμβάνει τη απάντηση και τελικά ξεκινά η μεταφορά δεδομένων. Οι λειτουργίες *blocking-send* και *blocking-receive* είναι πολύ απλές. Δεν απαιτείται καμία αποθήκευση ούτε στην αφετηρία ούτε στον προορισμό. Ωστόσο, η τριμερής χειραψία του πρωτοκόλλου υποχρεώνει τόσο τον πομπό όσο και τον δέκτη να περιμένουν για έναν ολόκληρο κύκλο τουλάχιστον, κατά τη διάρκεια του οποίου οι επεξεργαστές είναι αδρανείς. Επιπλέον, με λειτουργίες *blocking* είναι αδύνατη η επικάλυψη επικοινωνίας και υπολογισμών, γεγονός που αφήνει μέρος του εύρους ζώνης του δικτύου αχρησιμοποίητο.

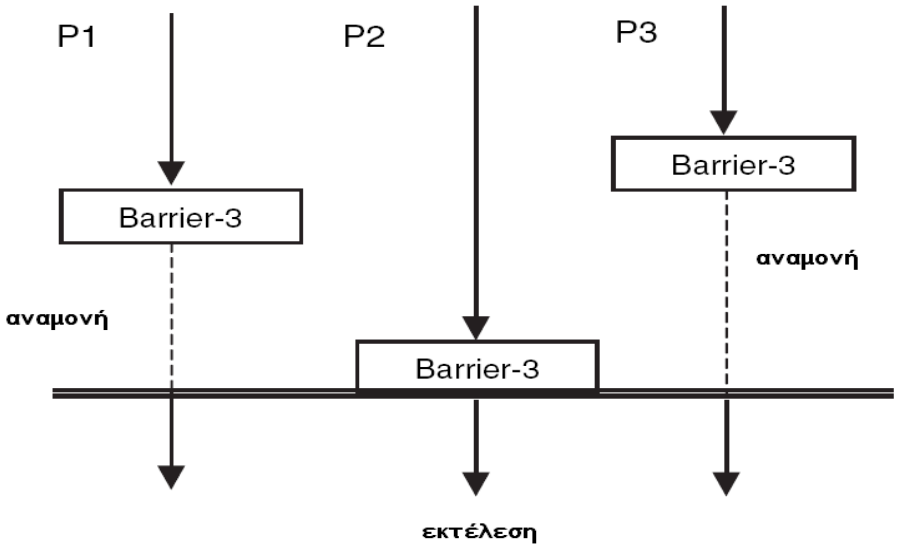
Για την αποφυγή των μειονεκτημάτων της τριμερούς χειραψίας, οι περισσότερες υλοποιήσεις ανταλλαγής μηνυμάτων χρησιμοποιούν *non-blocking* λειτουργίες. Στην περίπτωση αυτή, υπάρχει ένα στρώμα μηνυμάτων στο οποίο αποθηκεύονται τα μηνύματα μέχρι να είναι διαθέσιμη μία θύρα στο δίκτυο, ώστε ο αποστολέας να μη χρειάζεται να περιμένει μέχρι την εκτέλεση της λήψης. Όταν ο κόμβος προορισμού εκτελέσει τη λειτουργία λήψης τότε θα αρχίσει η μεταφορά από το στρώμα μηνυμάτων σε αυτόν.



Σχήμα 1.5: Πρωτόκολλο τριμερούς χειραψίας *blocking-send/receive*.

Η λειτουργία *broadcast* υλοποιεί την επικοινωνία ενός προς πολλούς. Ένας επεξεργαστής στέλνει το ίδιο μήνυμα σε διάφορους προορισμούς με μία εντολή.

Η λειτουργία *barrier* χρησιμοποιείται για το συγχρονισμό των διεργασιών όπως φαίνεται στο Σχήμα 1.6. Καμία ανταλλαγή δεδομένων δεν επιτρέπεται μέχρι όλοι οι επεξεργαστές που συμμετέχουν σε αυτή να καλέσουν τη ρουτίνα *barrier*.



Σχήμα 1.6: Λειτουργία *barrier*

1.2.2 Αρχιτεκτονική Μοιραζόμενης Μνήμης

Το βασικό χαρακτηριστικό του μοντέλου αυτού είναι το γεγονός ότι η επικοινωνία πραγματοποιείται έμμεσα, ως αποτέλεσμα μιας σειράς από συμβατικές εντολές πρόσβασης στη μνήμη (π.χ. load, store). Κάθε επεξεργαστής έχει τη δική του κρυφή μνήμη και όλοι οι επεξεργαστές και οι μανάδες μνήμες συνδέονται στο ίδιο δίκτυο διασύνδεσης, το οποίο είναι συχνά ένας κοινός διάδρομος. Η δυνατότητα για αποδοτική πρόσβαση σε όλα τα μοιραζόμενα δεδομένα από όλους τους επεξεργαστικούς κόμβους, σε συνδυασμό με την αυτόματη μεταφορά των δεδομένων στις τοπικές caches, διευκολύνει τον παράλληλο προγραμματισμό. Τα χαρακτηριστικά αυτά είναι επίσης σημαντικά για το λειτουργικό σύστημα, του οποίου οι λειτουργίες μοιράζονται δομές δεδομένων και μπορούν εύκολα να εκτελεστούν σε ξεχωριστούς επεξεργαστές. Το μοντέλο αυτό χρονολογείται από τις αρχές της δεκαετίας του 1960, και σήμερα συναντάται σχεδόν σε όλους τους τομείς της βιομηχανίας υπολογιστών. Κυριαρχεί στην αγορά των εξυπηρετητών, ενώ γίνεται όλο και πιο σύνηθες και στους προσωπικούς υπολογιστές.

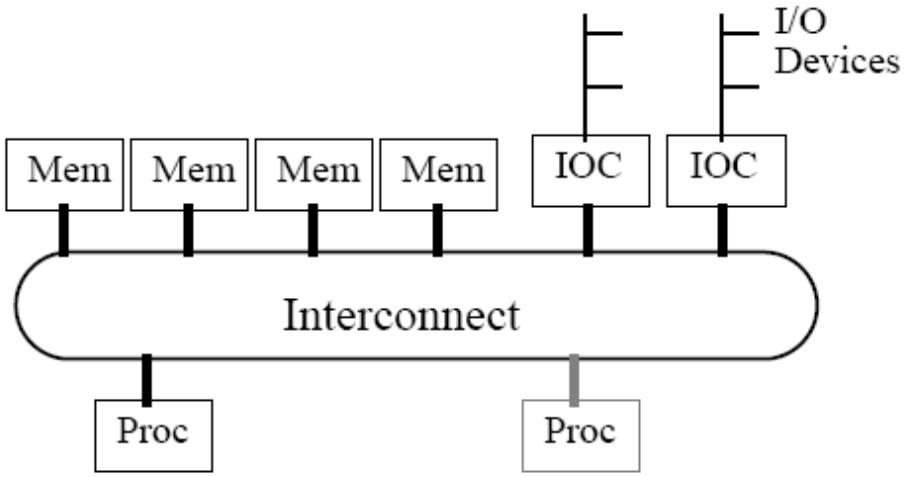
Η επικοινωνία και η συνεργασία των επεξεργαστών στο μοντέλο αυτό, αποτελείται από εγγραφές και διάβασμα μοιραζόμενων μεταβλητών. Οι λειτουργίες αυτές αντιστοιχίζονται απ' ευθείας σε ένα αφηρημένο μοντέλο επικοινωνίας, η οποία αποτελείται από εντολές load και store σε έναν κοινό χώρο διευθύνσεων, ο οποίος υποστηρίζεται από το υλικό μέσω ρητών προσβάσεων σε κοινές φυσικές

θέσεις μνήμης. Το προγραμματιστικό μοντέλο και το μοντέλο της επικοινωνίας βρίσκονται πολύ κοντά στο υλικό. Κάθε επεξεργαστής μπορεί να *ονοματίσει* οποιαδήποτε φυσική θέση στο μηχάνημα. Μια διεργασία μπορεί να ονοματίσει όλα τα δεδομένα που μοιράζεται με τις υπόλοιπες διεργασίες, μέσα στον εικονικό της χώρο διευθύνσεων. Τα δεδομένα μεταφέρονται είτε ως πρωταρχικοί τύποι μέσα στο σύνολο των εντολών είτε ως blocks της κρυφής μνήμης. Κάθε διεργασία εκτελεί λειτουργίες πρόσβασης στη μνήμη, σε διευθύνσεις του δικού της εικονικού χώρου. Η διαδικασία μετάφρασης των διευθύνσεων υποδεικνύει μια φυσική διεύθυνση, η οποία μπορεί να είναι είτε τοπική είτε απομακρυσμένη από τον επεξεργαστή και μπορεί να είναι μοιραζόμενη με άλλες διεργασίες. Σε οποιαδήποτε από τις δύο περιπτώσεις, η πρόσβαση από το υλικό είναι άμεση, χωρίς την επέμβαση του χρήστη ή του λειτουργικού συστήματος.

Η επίδοση του μοντέλου μοιραζόμενης μνήμης εξαρτάται σε μεγάλο βαθμό από το εύρος του χρονικού διαστήματος ανάμεσα στην πρόσβαση σε μια θέση μνήμης και στην ανάκτηση των επιθυμητών δεδομένων, όπως επίσης και από το εύρος μεταφοράς δεδομένων που μπορούν να υποστηριχθεί. Ακριβώς όπως μια ιεραρχία μνήμης επιτρέπει σε δεδομένα που είναι δεσμευμένα σε μια θέση μνήμης να μεταφερθούν προς τον επεξεργαστή, η έκφραση της επικοινωνίας με όρους του χώρου διευθύνσεων επιτρέπει σε μοιραζόμενα δεδομένα να μεταφερθούν προς τον επεξεργαστή που τα ζητάει.

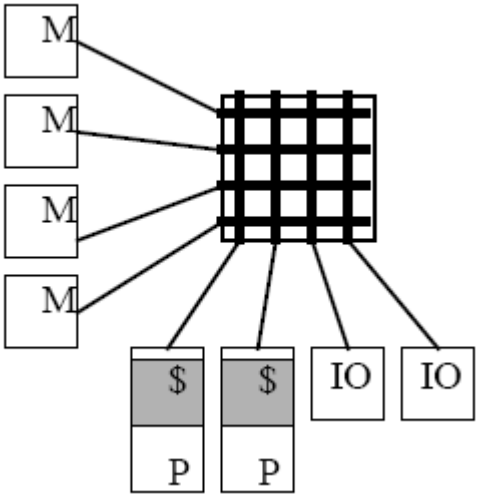
1.2.2.1 Σχήματα Διασύνδεσης Μοντέλων Μοιραζόμενης Μνήμης

Η οργάνωση του υλικού που χρησιμοποιείται στα πολυεπεξεργαστικά συστήματα μοιραζόμενης μνήμης είναι μια φυσική επέκταση των συστημάτων μνήμης που συναντάμε στους περισσότερους υπολογιστές. Όλα τα υπολογιστικά συστήματα περιέχουν απαραίτητα έναν επεξεργαστή και ένα σύνολο από ελεγκτές εισόδου/εξόδου για την πρόσβαση στα στοιχεία της μνήμης μέσω κάποιου τύπου διασύνδεσης, όπως παρουσιάζεται στο Σχήμα 1.7. Η χωρητικότητα της μνήμης αυξάνεται προσθέτοντας απλά επιπλέον στοιχεία μνήμης. Η επιπλέον χωρητικότητα μπορεί να αυξήσει το διαθέσιμο εύρος μνήμης ή όχι, ανάλογα με τη συγκεκριμένη οργάνωση του συστήματος. Η χωρητικότητα των λειτουργιών εισόδου/εξόδου αυξάνεται προσθέτοντας συσκευές ελέγχου. Αντίστοιχα, υπάρχουν δύο τρόποι αύξησης της υπολογιστικής επίδοσης: αναμονή μέχρι να γίνει διαθέσιμος ένας πιο γρήγορος επεξεργαστής ή προσθήκη επιπλέον επεξεργαστικών μονάδων. Θεωρητικά, όταν περισσότεροι επεξεργαστές είναι διαθέσιμοι, περισσότερες διεργασίες μπορούν να εκτελούν ταυτόχρονα, με αποτέλεσμα αύξηση στην επίδοση. Αν μια μοναδική εφαρμογή προγραμματιστεί ώστε να δημιουργεί πολλαπλά νήματα, η παρουσία πολλών επεξεργαστών θα επιταχύνει την εκτέλεσή της.

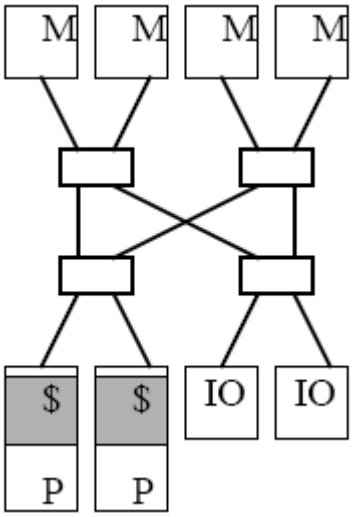


Σχήμα 1.7: Αφαιρετική παρουσίαση οργάνωσης συστήματος μοιραζόμενης μνήμης

Μεγάλη πρόοδος σημειώνεται στα συστήματα μοιραζόμενης μνήμης, επεκτείνοντας το γενικό σύστημα που παρουσιάζεται στο Σχήμα 1.7, καθώς το τεχνολογικό υπόβαθρο βελτιώνεται. Στα πρώτα υπολογιστικά συστήματα, η μνήμη ήταν αργή σε σχέση με τον επεξεργαστή, οπότε υπήρχε η ανάγκη για δρομολόγηση δεδομένων μέσω πολλαπλών καναλιών, ώστε να αυξηθεί το εύρος ζώνης. Η τεχνική αυτή έθετε την προϋπόθεση ύπαρξης ενός δικτύου διασύνδεσης ανάμεσα στον επεξεργαστή και στα κανάλια. Έτσι, για την εξυπηρέτηση των απαιτήσεων εισόδου/εξόδου ενός φορτίου, προσαρτούνταν στο σύστημα πολλά κανάλια εισόδου/εξόδου και συσκευές ελέγχου. Υπήρχε, επίσης, η ανάγκη για απ' ευθείας πρόσβαση στη μνήμη από τα κανάλια. Για το λόγο αυτόν, τα συστήματα τέτοιου τύπου ήταν συνήθως σχεδιασμένα ώστε ο επεξεργαστής να συνδέεται με τα κανάλια εισόδου/εξόδου μέσω *σταυρωτών διακοπτών*, όπως φαίνεται στο Σχήμα 1.8. Η προσθήκη επιπλέον επεξεργαστών ήταν απλώς θέμα επέκτασης του δικτύου. Η δομή του υλικού για την πρόσβαση μιας θέσης μνήμης από μια θύρα του επεξεργαστή και η πλευρά της εισόδου/εξόδου στον διακόπτη διατήρησαν την ίδια λογική. Αυτό που περιόριζε τον αριθμό των επεξεργαστών στα πρώιμα αυτά συστήματα, ήταν το μέγεθος και το κόστος του επεξεργαστή. Ωστόσο, όσο η πυκνότητα του υλικού μεγάλωνε και το κόστος μειωνόταν, όλο και μεγαλύτερα συστήματα κατασκευάζονταν. Αυτό που τελικά κατέληξε να αποτελεί περιορ9ιστικό παράγοντα, ήταν το κόστος της επέκτασης του σταυρωτού διακόπτη, ο οποίος αντικαταστάθηκε γρήγορα από τα *πολυεπίπεδα δίκτυα διασύνδεσης*, η δομή των οποίων φαίνεται στο Σχήμα 1.9.



Σχήμα 1.8: Δίκτυο Διασύνδεσης Σταυρωτού Διακόπτη

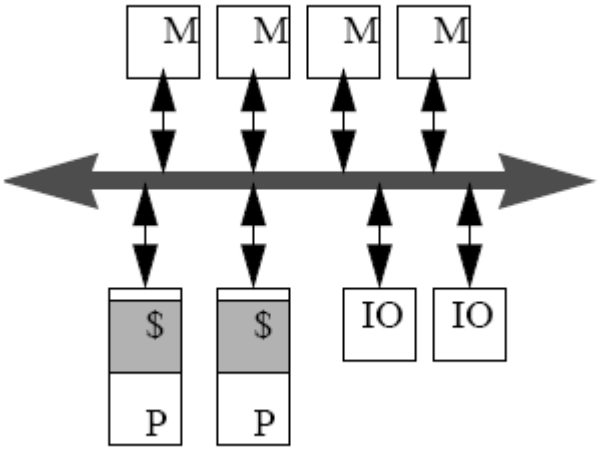


Σχήμα 1.9: Πολυεπίπεδο Δίκτυο Διασύνδεσης

Αν και το κόστος επέκτασης σε αυτό το δίκτυο είναι αρκετά χαμηλότερο, η ταχύτητα του δικτύου είναι μικρή όταν όλες οι θύρες χρησιμοποιούνται ταυτόχρονα.

Η έκρηξη στη χρήση υπολογιστικών συστημάτων μοιραζόμενης μνήμης συνέβη κατά την εμφάνιση των 32-bit επεξεργαστών, στα μέσα της δεκαετίας του 1980, διότι ο επεξεργαστής, η κρυφή μνήμη, η μονάδα διαχείρισης εντολών κινητής υποδιαστολής και η μονάδα ελέγχου της μνήμης χωρούσαν πλέον σε μία πλακέτα. Τα περισσότερα, μεσαίου μεγέθους υπολογιστικά συστήματα, συμπεριλαμβανομένων των μικροϋπολογιστών, των εξυπηρετητών και των προσωπικών υπολογιστών, είναι οργανωμένα γύρω από έναν κεντρικό *διάδρομο* μνήμης, όπως φαίνεται στο Σχήμα 1.10. Ο διάδρομος μπορεί να προσαρμοστεί ώστε να υποστηρίζει πολλαπλούς επεξεργαστές. Ο τυπικός μηχανισμός πρόσβασης στη μνήμη μέσω του διαδρόμου επιτρέπει σε κάθε επεξεργαστή να προσπελάσει οποιαδήποτε φυσική διεύθυνση στο σύστημα. Όπως και στον σχεδιασμό που βασίζεται σε διακόπτες, όλες οι θέσεις μνήμης απέχουν το ίδιο από τους

επεξεργαστές, με αποτέλεσμα όλες οι επεξεργαστικές μονάδες να απαιτούν τον ίδιο χρόνο για να ολοκληρώσουν μια προσπέλαση στη μνήμη.



Σχήμα 1.10: Σύστημα μοιραζόμενης μνήμης με σύνδεση διαδρόμου

Οι παράγοντες που περιορίζουν τον αριθμό των επεξεργαστών που μπορούν να υποστηριχθούν από ένα σύστημα οργανωμένο με διάδρομο, είναι αρκετά διαφορετικοί από αυτούς που αναφέραμε στην περίπτωση των σταυρωτών διακοπτών. Η προσθήκη ενός επιπλέον επεξεργαστή σε ένα δίκτυο διακόπτη είναι μεν ακριβή, αλλά η συνολική ταχύτητα του δικτύου αυξάνεται με την αύξηση των θυρών. Στην περίπτωση του διαδρόμου, το κόστος προσθήκης επιπλέον επεξεργαστών είναι πολύ μικρό, αλλά η συνολική ταχύτητα είναι δεδομένη. Διαιρώντας τη δεδομένη αυτή ταχύτητα με έναν μεγαλύτερο αριθμό επεξεργαστών, μειώνεται η επεκτασιμότητα της προσέγγισης αυτής. Το ευτυχές γεγονός είναι ότι η παρουσία των κρυφών μνημών μειώνει τις απαιτήσεις κάθε επεξεργαστή, καθώς πολλές από τις αιτήσεις εξυπηρετούνται από δεδομένα που υπάρχουν σε αυτές. Ωστόσο, η διατήρηση πολλαπλών αντιγράφων σε κάθε προσωπική κρυφή μνήμη, γεννά το πρόβλημα της συνάφειας της κρυφής μνήμης, το οποίο δε θα εξετάσουμε όμως εδώ.

Αν αναρωτηθούμε τι πρέπει να προσέξουμε αν θέλουμε να επεκτείνουμε ένα μικρού μεγέθους σύστημα μοιραζόμενης μνήμης , γίνεται αμέσως φανερό πως το πρόβλημα δεν βρίσκεται στις υπολογιστικές μονάδες, αλλά στο δίκτυο διασύνδεσης. Το δίκτυο σταυρωτού διακόπτη επεκτείνεται δύσκολα λόγω κόστους, ενώ η οργάνωση διαδρόμου πάσχει από την ύπαρξη ενός δεδομένου εύρους ζώνης που πρέπει να μοιραστεί στους επεξεργαστές. Ωστόσο, έχουν κατασκευαστεί αρκετά εναλλακτικά δίκτυα διασύνδεσης τα οποία ξεπερνούν σε μεγάλο βαθμό τα παραπάνω προβλήματα. Αυτό που θα πρέπει να προσέξουμε όταν σχεδιάζουμε ένα δίκτυο διασύνδεσης, είναι η επίδραση που θα έχει η σχεδίασή μας στο χρόνο που θα πρέπει να περιμένει ένας επεξεργαστεί μέχρι να εξυπηρετηθεί μια αίτησή του στη μνήμη. Αν ο χρόνος αυτός αυξηθεί σημαντικά, τότε οι επεξεργαστές θα ξοδεύουν τον περισσότερο χρόνο τους περιμένοντας για δεδομένα και δεν θα είναι εκμεταλλεύσιμα τα πλεονεκτήματα από την ύπαρξη πολλών επεξεργαστών.

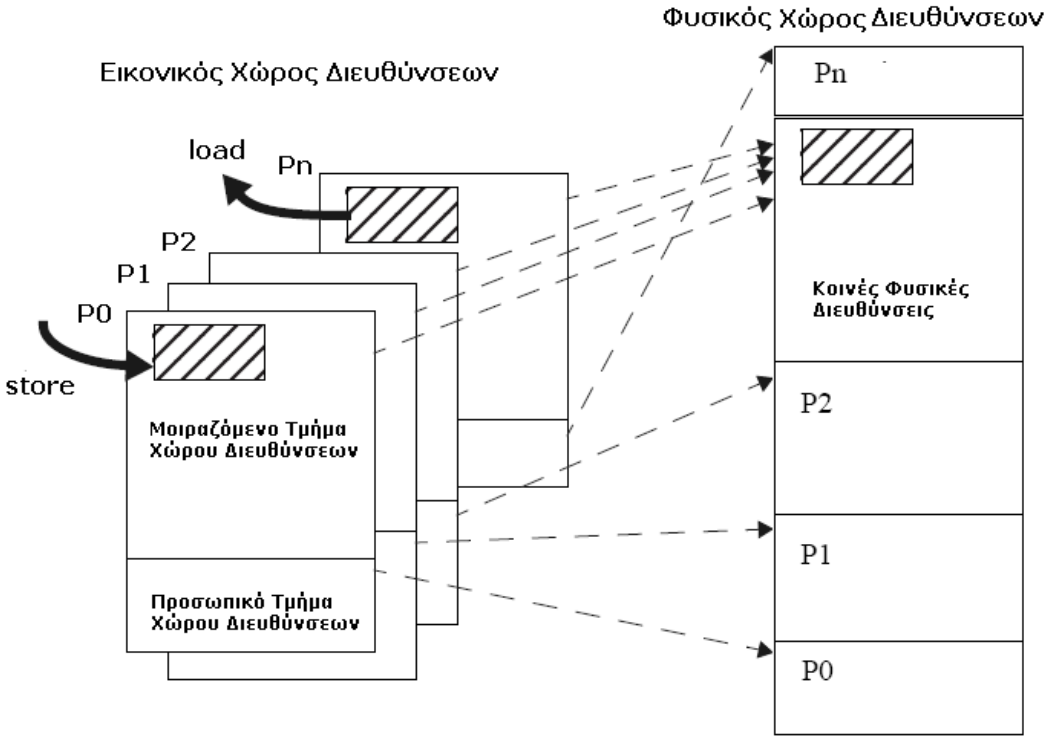
1.2.2.2 Προγραμματιστικά Μοντέλα Μοιραζόμενης Μνήμης

Το κυρίαρχο προγραμματιστικό μοντέλο σε μια αρχιτεκτονικής μοιραζόμενης μνήμης είναι αναγκαστικά παρόμοιο με αυτό της επικάλυψης χρόνου σε ένα

μονοεπεξεργαστικό σύστημα, μόνο που τη θέση του ψευδοπαραλληλισμού των εργασιών παίρνει πλέον ο πραγματικός παραλληλισμός. Τυπικά, στο μοντέλο αυτό, ορίζουμε ως διεργασία έναν εικονικό χώρο μνήμης και ένα ή περισσότερα νήματα ελέγχου. Οι διεργασίες μπορούν να διαμορφωθούν ώστε τμήματα του χώρου διευθύνσεών τους να είναι μοιραζόμενα, δηλαδή να αντιστοιχίζονται σε μια κοινή φυσική τοποθεσία. Το τυπικό μοντέλο μνήμης για μια αρχιτεκτονική μοιραζόμενης μνήμης παρουσιάζεται στο Σχήμα 1.11. Στο σχήμα αυτό, βλέπουμε ότι συλλογές από διεργασίες μοιράζονται μια κοινή περιοχή διευθύνσεων η οποία αντιστοιχίζεται στον εικονικό τους χώρο διευθύνσεων, αλλά έχουν, επίσης, και μια προσωπική περιοχή μνήμης, όπου διατηρείται η στοίβα εκτέλεσης και άλλα προσωπικά δεδομένα.

Η συνεργασία ανάμεσα στα νήματα και ο συντονισμός τους επιτυγχάνεται με το διάβασμα και την εγγραφή κοινών μεταβλητών και δεικτών οι οποίοι δείχνουν σε κοινές διευθύνσεις. Εγγραφές από ένα νήμα σε μια λογικά μοιραζόμενη διεύθυνση είναι ορατές προς ανάγνωση από τα υπόλοιπα νήματα. Η αρχιτεκτονική χρησιμοποιεί τις συνήθεις λειτουργίες πρόσβασης στη μνήμη για την παροχή επικοινωνίας ανάμεσα στις μοιραζόμενες διευθύνσεις, όπως επίσης και ειδικές ατομικές λειτουργίες για συγχρονισμό. Ακόμη και εντελώς ανεξάρτητες μεταξύ τους διεργασίες, μοιράζονται τυπικά, τον πυρήνα του χώρου διευθύνσεων, αν σε αυτόν έχουν πρόσβαση μόνο διεργασίες του λειτουργικού συστήματος.

Αν και η κοινή μνήμη μπορεί να χρησιμοποιηθεί για την επικοινωνία αυθαίρετων συλλογών από διεργασίες, τα περισσότερα παράλληλα προγράμματα είναι προσεκτικά δομημένα όσον αφορά στη χρήση των εικονικών διευθύνσεων. Τυπικά, υπάρχει μια κοινή απεικόνιση του κώδικα, προσωπικά τμήματα για τη στοίβα και άλλα προσωπικά δεδομένα και κοινά τμήματα τα οποία βρίσκονται στην ίδια περιοχή του εικονικού χώρου διευθύνσεων κάθε διεργασίας ή νήματος του προγράμματος. Αυτή η απλοϊκή δομή υποδηλώνει ότι οι προσωπικές μεταβλητές του προγράμματος βρίσκονται σε κάθε διεργασία, ενώ οι μοιραζόμενες μεταβλητές έχουν την ίδια διεύθυνση και νόημα σε κάθε νήμα. Πολλές φορές, εφαρμόζονται τεχνικές ευθύ παραλληλισμού. Για παράδειγμα, κάθε διεργασία μπορεί να εκτελεί ένα υποσύνολο των επαναλήψεων ενός loop ή, πιο γενικά, οι διεργασίες μπορούν να λειτουργούν ως μια ομάδα υπολογιστικών πόρων στους οποίους ανατίθενται εργασίες από μία ουρά.



Σχήμα 1.11: Τυπικό μοντέλο για παράλληλα προγράμματα μοιραζόμενης μνήμης

1.2.3 Αρχιτεκτονική Κατανεμημένης Μνήμης vs. Αρχιτεκτονική Μοιραζόμενης Μνήμης

Το κύριο χαρακτηριστικό των συστημάτων μοιραζόμενης μνήμης είναι το γεγονός ότι η επικοινωνία υλοποιείται με λειτουργίες load-store σε έναν κοινό χώρο διευθύνσεων. Ακόμη ένα θεμελιώδες χαρακτηριστικό της μοιραζόμενης μνήμης είναι ο ξεκάθαρος διαχωρισμός επικοινωνίας και συγχρονισμού. Απαιτείται να υπάρχουν ειδικοί μηχανισμοί συγχρονισμού, επιπλέον των λειτουργιών load-store, ώστε να διαπιστώνεται πότε παράγονται ή καταναλώνονται δεδομένα. Αντιθέτως, σε ένα σύστημα κατανεμημένης μνήμης χρησιμοποιείται ένα σαφές μοντέλο επικοινωνίας. Διακριτά μηνύματα ανταλλάσσονται ανάμεσα στους επεξεργαστές. Ο συγχρονισμός και η επικοινωνία ενοποιούνται μέσα από την ανταλλαγή μηνυμάτων. Η παραγωγή απομακρυσμένων, ασύγχρονων γεγονότων είναι αναπόσπαστο κομμάτι του μοντέλου ανταλλαγής μηνυμάτων. Είναι, ωστόσο, σημαντικό να σημειώσουμε ότι τα μοντέλα επικοινωνίας των δύο αρχιτεκτονικών είναι καθολικά, δηλαδή μπορούμε να χρησιμοποιήσουμε το ένα για να προσομοιώσουμε το άλλο. Ωστόσο, έχει παρατηρηθεί ότι είναι πιο εύκολο να προσομοιωθεί ένα σύστημα μοιραζόμενης μνήμης από ένα σύστημα ανταλλαγής μηνυμάτων απ' ότι το αντίστροφο.

Οι αρχιτεκτονικές μοιραζόμενης μνήμης χαρακτηρίζονται από πολλά επιθυμητά στοιχεία. Το μοντέλο επικοινωνίας επιτρέπει στον προγραμματιστή να επικεντρωθεί σε θέματα παραλληλισμού αποκρύπτοντάς του τις λεπτομέρειες της πολυεπεξεργαστικής επικοινωνίας. Από αυτή την άποψη, η επικοινωνία μοιραζόμενης μνήμης παριστάνει μια σαφή επέκταση του μονοεπεξεργαστικού μοντέλου προγραμματισμού. Επιπλέον, η σημασιολογία της μοιραζόμενης μνήμης είναι ανεξάρτητη της φυσικής θέσης με αποτέλεσμα να επιδέχεται δυναμικών βελτιστοποιήσεων από το εκάστοτε λειτουργικό σύστημα. Από την άλλη πλευρά το μοντέλο μοιραζόμενης μνήμης είναι κατ' ανάγκη ανταγωνιστικό μοντέλο. Αυτό είναι μειονέκτημα όσον αφορά στο συγχρονισμό. Το γεγονός αυτό έχει διαπιστωθεί από

πολλούς αρχιτέκτονες πολυεπεξεργαστικών συστημάτων και η λύση τους ήταν πάντα η επαύξηση του βασικού μοντέλου επικοινωνίας με πρόσθετους μηχανισμούς συγχρονισμού. Ακόμη ένα μειονέκτημα είναι η αδυναμία των αρχιτεκτονικών μοιραζόμενης μνήμης για μονόδρομη επικοινωνία.

Η ανταλλαγή μηνυμάτων μπορεί να χαρακτηριστεί ως ένα είδος επικοινωνιακού μοντέλου βασισμένο στις διακοπές. Η αρχιτεκτονική αυτή ενοποιεί τα δεδομένα και το συγχρονισμό σε μία μονάδα. Έτσι, το μοντέλο ανταλλαγής μηνυμάτων προσφέρεται για τις ενέργειες εκείνες ενός λειτουργικού συστήματος κατά τις οποίες τα πρότυπα επικοινωνίας είναι εκ των προτέρων γνωστά, όπως λειτουργίες εισόδου/εξόδου, διακοπές και μεταφορά διεργασιών και δεδομένων. Επιπλέον, τα μοντέλα ανταλλαγής μηνυμάτων είναι εξ' ορισμού μοντέλα πελάτη-εξυπηρετητή. Ωστόσο, το μοντέλο ανταλλαγής μηνυμάτων πάσχει από το κόστος της συναρμολόγησης και αποσυναρμολόγησης των μηνυμάτων.

Συμπερασματικά, το μοντέλο μοιραζόμενης μνήμης και το μοντέλο ανταλλαγής μηνυμάτων είναι κατάλληλα για συγκεκριμένους τομείς εφαρμογής το καθένα. Το μεγάλο πλεονέκτημα των εφαρμογών μοιραζόμενης μνήμης είναι η ευκολία υλοποίησης, σε αντίθεση με το μοντέλο ανταλλαγής μηνυμάτων. Από την πλευρά, όμως, ένα σύστημα μοιραζόμενης μνήμης δεν είναι τόσο εύκολα επεκτάσιμο όσο ένα σύστημα κατανεμημένης μνήμης. Το μοντέλο μοιραζόμενης μνήμης επικρατεί ανάμεσα στους συγγραφείς εφαρμογών ενώ το μοντέλο ανταλλαγής μηνυμάτων είναι ιδιαίτερα δημοφιλές στη σχεδίαση λειτουργικών συστημάτων και σε εφαρμογές που περιέχουν συστατικά συγχρονισμού, όπως επίλυση συστημάτων αραιών πινάκων και προσομοίωση γεγονότων

1.3 Συμπεράσματα

Το προγραμματιστικό μέρος της παρούσας εργασίας περιλαμβάνει υλοποιήσεις δύο αλγόριθμων, τόσο σε σειριακή έκδοση και στη συνέχεια, παραλληλοποίηση τους. Ο πρώτος αλγόριθμος αφορά στην εκτίμηση κατάστασης συστημάτων ηλεκτρικής ενέργειας με τη μέθοδο των ελαχίστων τετραγώνων, ενώ ο δεύτερος είναι η μέθοδος επίλυσης γραμμικών συστημάτων Conjugate Gradient. Για την υλοποίηση των εφαρμογών αυτών, επιλέχθηκε το μοντέλο της ανταλλαγής μηνυμάτων. Η επιλογή βασίστηκε σε διάφορους παράγοντες. Ο βασικός λόγος ήταν η ανάγκη για κλιμάκωση των εφαρμογών σε πολλούς επεξεργαστές. Η ανάγκη αυτή είναι επιτακτική ιδιαίτερα για τον αλγόριθμο εκτίμησης κατάστασης, ο οποίος αφορά σε ένα σύστημα ηλεκτρικής ενέργειας. Τα κέντρα ελέγχου του δικτύου είναι απομακρυσμένα το ένα από το άλλο, περιέχουν διαφορετικά δεδομένα και δεν μοιράζονται κάποιο κοινό χώρο δεδομένων. Επίσης, το μέγεθος του δικτύου μπορεί να μεταβάλλεται, γεγονός που, επίσης, επιβάλλει τη δυνατότητα κλιμάκωσης του μοντέλου αρχιτεκτονικής.

Επιπλέον, τόσο αλγόριθμος εκτίμησης κατάστασης, όσο και η μέθοδος επίλυσης γραμμικών συστημάτων Conjugate Gradient, είναι δύο μαθηματικά προβλήματα, τα οποία περιέχουν κυρίως πράξεις πινάκων και διανυσμάτων. Η παραλληλοποίηση αυτών των λειτουργιών, όπως π.χ. του πολλαπλασιασμού ενός πίνακα με διάνυσμα, απαιτούν συγχρονισμό, ο οποίος επιτυγχάνεται με μεγάλη ευκολία στο μοντέλο ανταλλαγής μηνυμάτων.

Ένας άλλος καθοριστικός παράγοντας για την επιλογή του μοντέλου ανταλλαγής μηνυμάτων, ήταν η διαθεσιμότητα, στο μοντέλο αυτό, προγραμματιστικών εργαλείων για επιστημονικούς υπολογισμούς. Η χρήση των εργαλείων αυτών, τα οποία παρουσιάζονται αναλυτικά σε επόμενο κεφάλαιο, διευκόλυνε σε μεγάλο βαθμό την υλοποίηση της εργασίας.

Πριν προχωρήσουμε στην περιγραφή των προγραμματιστικών εργαλείων και των ίδιων των υλοποιήσεων που πραγματοποιήθηκαν, θα παρουσιάσουμε, στο κεφάλαιο που ακολουθεί, κάποια θεωρητικά στοιχεία για τις μεθόδους επίλυσης γραμμικών συστημάτων που χρησιμοποιήθηκαν.

Κεφάλαιο 2ο: Μέθοδοι Επίλυσης Γραμμικών Συστημάτων

2.1 Γενικά

Ο μηχανικός καλείται συχνά να αντιμετωπίσει το πρόβλημα της επίλυσης γραμμικών συστημάτων. Τα γραμμικά συστήματα μπορούν να προκύψουν από φαινομενικά μη σχετικά προβλήματα. Χαρακτηριστικά παραδείγματα στα οποία εφαρμόζονται μέθοδοι επίλυσης γραμμικών συστημάτων είναι οι συνήθεις και οι μερικές διαφορικές εξισώσεις, τα προβλήματα ιδιοτιμών και ιδιοδιανυσμάτων, οι πολυωνυμικές ή άλλες προσεγγίσεις συναρτήσεων, ο υπολογισμός ελαχίστων τετραγώνων, ο γραμμικός προγραμματισμός, ο βέλτιστος έλεγχος κ.ά. Ακόμη και τα μη γραμμικά συστήματα, και τα άλλα μη γραμμικά προβλήματα, ανάγονται και αυτά στην επίλυση γραμμικών συστημάτων. Έτσι, είναι σαφές ότι σε κλάδους όπως οι εξομοιώσεις φυσικών διεργασιών, ο βέλτιστος έλεγχος, η θεωρία προσέγγισης, ο μαθηματικός προγραμματισμός και σε διάφορα προβλήματα επιχειρησιακής έρευνας, η ταχύτητα και η αξιοπιστία με την οποία μπορούν να λυθούν τα γραμμικά συστήματα, παίζουν σημαντικό ρόλο και, προφανώς, εξαρτώνται από τη μέθοδο επίλυσης που χρησιμοποιείται.

Διακρίνονται δύο γενικές μέθοδοι για την επίλυση γραμμικών συστημάτων της μορφής $\mathbf{Ax} = \mathbf{b}$, οι άμεσες και οι έμμεσες ή επαναληπτικές. Στην πρώτη κατηγορία ανήκει η μέθοδος απαλοιφής του Gauss, η μέθοδος του Cramer και η παραγοντοποίηση LU. Στην περίπτωση όπου ο πίνακας του συστήματος είναι συμμετρικός και θετικά οριζόμενος, η επίλυση μπορεί να γίνει με την ιδιαίτερα γρήγορη μέθοδο παραγοντοποίησης Cholesky. Στη δεύτερη κατηγορία ανήκουν οι μέθοδοι Jacobi και Gauss-Siedel, η μέθοδος Ελάχιστων Υπολοίπων (GMRES) και η μέθοδος Conjugate Gradient. Στη συνέχεια, θα μελετήσουμε αναλυτικότερο κάποιες από αυτές.

2.2 Άμεσες Μέθοδοι

Πρόκειται για μεθόδους όπου με έναν πεπερασμένο αριθμό στοιχειωδών πράξεων αριθμητικής επιτυγχάνεται η λύση του προβλήματος. Ευνόητο είναι ότι τέτοιες μέθοδοι δίνουν ακριβή αποτελέσματα, εφόσον δεν εμπλέκονται σφάλματα στρογγυλοποίησης. Σε περίπτωση που παρουσιάζονται τέτοιου είδους σφάλματα, τότε οι λύσεις που προκύπτουν απέχουν πολύ από τις πραγματικές λύσεις του συστήματος. Για αυτό το λόγο, ένα μεγάλο μέρος της έρευνας αφορά τον προσδιορισμό διαδικασιών που περιορίζουν την επίδραση σφαλμάτων και βελτιώνουν την ευστάθεια των άμεσων μεθόδων. Στη συνέχεια, θα παρουσιάσουμε δύο από τις άμεσες μεθόδους επίλυσης γραμμικών συστημάτων, την παραγοντοποίηση LU και την παραγοντοποίηση Cholesky.

2.2.1 Παραγοντοποίηση LU

Στη γραμμική άλγεβρα, η παραγοντοποίηση LU είναι η διάσπαση ενός πίνακα στο γινόμενο ενός άνω και ενός κάτω τριγωνικού πίνακα. Το γινόμενο περιέχει μερικές φόρες και έναν τρίτο πίνακα, τον πίνακα μετάθεσης. Η μέθοδος αυτή είναι κυρίως χρήσιμη όταν έχουμε να λύσουμε πολλά συστήματα με κοινό πίνακα **A** και διαφορετικά δεύτερα μέλη **b**, οπότε η παραγοντοποίηση γίνεται μία φορά μόνο. Όταν ο πίνακας **A** είναι συμμετρικός και θετικά ορισμένος, η μέθοδος απλοποιείται και ονομάζεται παραγοντοποίηση Cholesky, η οποία περιγράφεται στην επόμενη ενότητα.

Περιγραφή

Έστω **A** τετραγωνικός πίνακας. Η παραγοντοποίηση LU έχει τη μορφή

$$\boldsymbol{A = LU}$$

όπου $\boldsymbol{L}$ και $\boldsymbol{U}$ είναι αντίστοιχα ένας κάτω και ένας άνω τριγωνικός πίνακας (ίδιου μεγέθους).

Ένας αντιστρέψιμος πίνακας επιδέχεται παραγοντοποίησης LU αν και μόνο αν όλες οι πρωτεύουσες ελάσσονες ορίζουσές του είναι μη μηδενικές. Τότε, η παραγοντοποίηση είναι μοναδική αν απαιτήσουμε η διαγώνιος του ενός από τους δύο παράγοντες να αποτελείται από μονάδες.

Υπολογισμός της παραγοντοποίησης LU

Εξισώνοντας τους συντελεστές στην παρακάτω εξίσωση

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \dots & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} & \dots & u_{n1} \\ 0 & u_{22} & u_{32} & \dots & u_{n2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \dots & u_{nn} \end{bmatrix}$$

και απαιτώντας τα διαγώνια στοιχεία του πίνακα **L** να είναι ίσα με τη μονάδα, παίρνουμε τις ακόλουθες σχέσεις:

$$l_{ik} = \begin{cases} 0, & i < k \\ 1, & i = k \\ \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, & i > k \end{cases}$$

$$u_{kj} = \begin{cases} 0, & k > j \\ a_{kj}^{(k)}, & k \le j \end{cases}$$

Η μέθοδος μπορεί να περιγραφεί με ψευδοκώδικα ως εξής:

For $j = 1$ **to** n

    $u_{1j} = a_{1j}$

For $i = 1$ **to** n

    $l_{i1} = a_{i1} / u_{11}$

For $m = 2$ **to** n

    For $j = m$ **to** n

        $u_{mj} = a_{mj} - \sum_{k=1}^{m-1} l_{mk} u_{kj}$

    For $i = m + 1$ **to** n

        $l_{im} = (a_{im} - \sum_{k=1}^{m-1} l_{ik} u_{km}) / u_{mm}$

Στη συνέχεια, η επίλυση του συστήματος γίνεται σε δύο βήματα. Αρχικά, επιλύουμε την εξίσωση $Ly = b$ ως προς y και στη συνέχεια λύνουμε την εξίσωση $Ux = y$ ως προς x. Ας σημειώσουμε εδώ, ότι και στις δύο περιπτώσεις, έχουμε τριγωνικούς πίνακες, οπότε τα συστήματα μπορούν να λυθούν απ' ευθείας χρησιμοποιώντας προς-πίσω αντικαταστάσεις, χωρίς να χρησιμοποιήσουμε τη μέθοδο απαλοιφής του Gauss. Ωστόσο, η μέθοδος απαλοιφής του Gauss απαιτείται

για τον υπολογισμό της ίδιας της παραγοντοποίησης LU, γεγονός που καθιστά τη μέθοδο αποδοτική μόνο όταν έχουμε να λύσουμε πολλαπλά συστήματα με τον ίδιο πίνακα **A** αλλά διαφορετικό διάνυσμα **b**.

2.2.1.1 Ατελής Παραγοντοποίηση LU

Για λόγους επίδοσης και κυρίως όταν ο πίνακας του συστήματος είναι ιδιαίτερα αραιός, χρησιμοποιείται συχνά μια παραλλαγή της παραγοντοποίησης LU, η ατελής LU. Σύμφωνα με τη μέθοδο αυτή, η παραγοντοποίηση προσεγγίζεται χρησιμοποιώντας ένα υποσύνολο των στοιχείων του πίνακα. Έστω **A** ο nxn πίνακας του συστήματος και **S** το σύνολο των στοιχείων του που έχουμε επιλέξει για την προσέγγιση της παραγοντοποίησης. Ο αλγόριθμος παρουσιάζεται στη συνέχεια με τη μορφή ψευδοκώδικα:

$$\boldsymbol{For\ r = 1\ to\ n-1}$$
$$\boldsymbol{d = 1/a_{rr}}$$
$$\boldsymbol{for\ i = r+1\ to\ n}$$
$$\boldsymbol{if\ (i,r) \in S\ then}$$
$$\boldsymbol{e = da_{ir}}$$
$$\boldsymbol{a_{ir} = e}$$
$$\boldsymbol{for\ j = r+1\ to\ n}$$
$$\boldsymbol{if\ ((i,j) \in S)\ and\ ((r,j) \in S)\ then}$$
$$\boldsymbol{a_{ij} = a_{ij} - ea_{rj}}$$

2.2.2 Παραγοντοποίηση Cholesky

Όταν ο πίνακας του συστήματος είναι συμμετρικός και θετικά ορισμένος, η παραγοντοποίηση LU παίρνει μια ιδιαίτερη μορφή και ονομάζεται παραγοντοποίηση Cholesky. Η παραγοντοποίηση Cholesky αναφέρεται στη διάσπαση του πίνακα στο γινόμενο ενός κάτω τριγωνικού πίνακα και του συζυγή ανάστροφού του. Δεδομένου ότι ικανοποιούνται οι προϋποθέσεις για την εφαρμογής της, έχει βρεθεί πειραματικά ότι η παραγοντοποίηση Cholesky είναι ως και δύο φορές πιο γρήγορη στην επίλυση ενός γραμμικού συστήματος από την LU.

Περιγραφή

Αν ο πίνακας $\boldsymbol{A}$ είναι συμμετρικός και θετικά ορισμένος, τότε ο $\boldsymbol{A}$ μπορεί να γραφτεί ως εξής:

$$\boldsymbol{A = LL^T}$$

όπου $\boldsymbol{L}$ είναι ένας κάτω τριγωνικός πίνακας με αυστηρώς θετικά διαγώνια στοιχεία και $\boldsymbol{L^T}$ είναι ο ανάστροφός του πίνακας. Η παραγοντοποίηση Cholesky είναι μοναδική. Δεδομένου ενός συμμετρικού, θετικά ορισμένου πίνακα $\boldsymbol{A}$, υπάρχει μόνο ένας κάτω τριγωνικός πίνακας $\boldsymbol{L}$ με αυστηρώς θετικά διαγώνια στοιχεία, τέτοιος ώστε $\boldsymbol{LL^T}$.

Ισχύει, επίσης και το αντίστροφο, ότι δηλαδή, αν ο A μπορεί να γραφτεί ως $\boldsymbol{LL^T}$ για κάποιον αντιστρέψιμο κάτω τριγωνικό πίνακα $\boldsymbol{L}$, τότε ο A είναι συμμετρικός και θετικά ορισμένος.

Υπολογισμός της παραγοντοποίησης Cholesky

Για τον υπολογισμό των $\boldsymbol{L}$, $\boldsymbol{L^T}$ εξισώνουμε απλώς τους συντελεστές των δύο μελών της εξίσωσης:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \dots & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & \dots & l_{n1} \\ 0 & l_{22} & l_{32} & \dots & l_{n2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \dots & l_{nn} \end{bmatrix}$$

και λαμβάνουμε:

$$a_{11} = l_{11}^2 \rightarrow l_{11} = \sqrt{a_{11}}$$

$$a_{21} = l_{21} l_{11} \rightarrow l_{21} = \frac{a_{21}}{l_{11}}, \dots l_{n1} = \frac{a_{n1}}{l_{11}}$$

$$a_{22} = l_{21}^2 + l_{22}^2 \rightarrow l_{22} = \sqrt{(a_{22} - l_{21}^2)}$$

$$a_{32} = l_{31} l_{21} + l_{32} l_{22} \rightarrow l_{32} = \frac{(a_{32} - l_{31} l_{21})}{l_{22}}$$, κ.τ.λ.

Γενικά, για i=1, …, n και j = i+1, …, n παίρνουμε:

$$l_{ii} = \sqrt{(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2)}$$

$$l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}}$$

Η μέθοδος μπορεί να περιγραφεί με ψευδοκώδικα ως εξής:

For $k = 1$ **to** $n - 1$

    $l_{kk} = \sqrt{a_{kk}}$

    For $s = k + 1$ **to** n

        $l_{sk} = a_{sk} / l_{kk}$

    For $j = k + 1$ **to**

        For $i = j$ **to** n

            $a_{ij} = a_{ij} - l_{ik} l_{jk}$

$l_{nn} = \sqrt{a_{nn}}$

Αφού υπολογίσουμε την παραγοντοποίηση του πίνακα $\boldsymbol{A}$, μπορούμε στη συνέχεια να λύσουμε το σύστημα $\boldsymbol{Ax = b}$, λύνοντας πρώτα το σύστημα $\boldsymbol{Ly = b}$, ώστε να πάρουμε το διάνυσμα **y** και στη συνέχεια λύνοντας το σύστημα $\boldsymbol{L^T x = y}$, ώστε να πάρουμε τελικά τη λύση **x.**

2.3 Επαναληπτικές Μέθοδοι

Στις επαναληπτικές μεθόδους η λύση προσεγγίζεται, με τη βοήθεια μιας αρχικής εκτίμησης της λύσης και με ένα κατάλληλα επιλεγμένο αλγόριθμο. Ένα σημαντικό πρόβλημα που απασχολεί τους χρήστες των έμμεσων μεθόδων, είναι η σύγκλιση και μάλιστα η ταχύτητα σύγκλισης που τις χαρακτηρίζουν. Σημαντικό πλεονέκτημα των έμμεσων μεθόδων είναι η απλότητά τους και η ανεξαρτησία τους από σφάλματα στρογγυλοποίησης. Οι επαναληπτικές μέθοδοι χρησιμοποιούνται συχνά, ιδιαίτερα στη λύση μερικών διαφορικών εξισώσεων και προβλημάτων βελτιστοποίησης.

2.3.1 Μέθοδοι Krylov

Τα συστήματα γραμμικών εξισώσεων που προκύπτουν από τα περισσότερα πρακτικά προβλήματα επιστημονικών υπολογισμών, είναι συνήθως τόσο μεγάλα, ώστε η χρήση των άμεσων μεθόδων που περιγράφηκαν στην προηγούμενη ενότητα, είναι απαγορευτική. Το γεγονός αυτό, οφείλεται τόσο στους περιορισμούς στο χώρο αποθήκευσης των δεδομένων, όσο και σε περιορισμούς στο χρόνο υπολογισμού της λύσης.

Ευτυχώς, οι περισσότεροι πίνακες, σε «πραγματικές» συνθήκες, είναι συνήθως αραιοί, έχουν, δηλαδή, λίγες μόνο μη μηδενικές τιμές. Πρακτικά, αυτό σημαίνει ότι γινόμενα τέτοιων πινάκων με διανύσματα μπορούν να υπολογιστούν σχετικά γρήγορα. Οι επαναληπτικές μέθοδοι παράγουν μια ακολουθία από προσεγγιστικές λύσεις, όπου ο προσδιορισμός της επόμενης προσέγγισης προκύπτει από την προηγούμενη, εκτελώντας μια σειρά από πολλαπλασιασμούς πίνακα επί διάνυσμα.

Οι μέθοδοι Krylov αποτελούν τη σημαντικότερη κατηγορία επαναληπτικών μεθόδων επίλυσης γραμμικών συστημάτων. Ένας υποχώρος Krylov τάξης r παράγεται από έναν nxn πίνακα A και ένα διάνυσμα b μεγέθους n ως ένα γραμμικό υποσύνολο ανεπτυγμένο από τις προβολές του b πάνω στις πρώτες r δυνάμεις του A, ως εξής:

$$\boldsymbol{K_r(A,b) = span\{b, Ab, A^2b, \dots, A^{r-1}b\}}$$

Αν τα διανύσματα $\boldsymbol{b, Ab, A^2b, \dots, A^{r-1}b}$ είναι γραμμικά ανεξάρτητα, τότε λέμε ότι σχηματίζουν μια βάση για τον υποχώρο Krylov $\boldsymbol{K_r(A,b)}$. Συνήθως, όμως, η βάση αυτή δεν είναι η καταλληλότερη για τη μέθοδο. Αυτό συμβαίνει διότι τα διανύσματα $\boldsymbol{A^jb}$ τείνουν να πλησιάζουν την κατεύθυνση του κυρίαρχου ιδιοδιανύσματος, όσο το j αυξάνεται. Αυτό, σημαίνει, πως για k αρκετά μεγάλο, τα περισσότερα διανύσματα του υποχώρου Krylov θα δείχνουν προς την ίδια κατεύθυνση. Για το λόγο αυτό, απαιτείται η δημιουργία μιας άλλης βάσης για τον υποχώρο, με διανύσματα $\boldsymbol{u_i}$, τέτοια ώστε $\boldsymbol{span\{u_1, \dots, u_i\} = K_i(A,x), για\ i = 1, \dots k}$

Οι διάφορες διαδικασίες που οδηγούν στη δημιουργία μιας βάσης Krylov γεννούν τις μεθόδους Krylov για την επίλυση γραμμικών συστημάτων. Δύο από τις βασικότερες, η GMRES και η Conjugate Gradient περιγράφονται στη συνέχεια.

2.3.2 Μέθοδος GMRES (Generalized Minimal Residual Method)

Η μέθοδος GMRES είναι σχεδιασμένη για την επίλυση μη-συμμετρικών γραμμικών συστημάτων. Η πιο δημοφιλής μορφή της βασίζεται σε μια τροποποιημένη Gram-Schmidt διαδικασία και χρησιμοποιεί επανεκκινήσεις για τον έλεγχο των αποθηκευτικών απαιτήσεων. Στην περίπτωση όπου δεν χρησιμοποιούνται επανεκκινήσεις, η μέθοδος (όπως και κάθε άλλη ορθογωνική μέθοδος Krylov) θα συγκλίνει σε n το πολύ βήματα, όπου n είναι το μέγεθος του συστήματος. Φυσικά, για n αρκετά μεγάλο, η σύγκλιση σε n βήματα δεν είναι πρακτικά αποδεκτή. Επιπλέον, σε αυτή την περίπτωση, η χρήση της μεθόδου χωρίς επανεκκινήσεις είναι απαγορευτική, λόγω του φόρτου υπολογισμών αλλά και των αποθηκευτικών απαιτήσεων. Στην πραγματικότητα, η κρίσιμη απόφαση κατά τη χρήση της μεθόδου αυτής, είναι ο προσδιορισμός του σημείου επανεκκίνησης.

Περιγραφή της μεθόδου.

Έστω ότι ο πίνακας A είναι αντιστρέψιμος και το διάνυσμα b είναι κανονικοποιημένο (δηλαδή $\|\boldsymbol{b}\| = \boldsymbol{1}$). Ο υποχώρος Krylov τάξης r για το πρόβλημα θα είναι:

$K_r(A,b) = span\{b, Ab, A^2b, \dots, A^{r-1}b\}$. Η μέθοδος GMRES προσεγγίζει τη λύση αναζητώντας το διάνυσμα $x_r \in K_r$ το οποίο ελαχιστοποιεί τη νόρμα του υπολοίπου $Ax_r - b$. Όπως αναφέραμε και προηγουμένως, τα διανύσματα $b, Ab, A^2b, \dots, A^{r-1}b$ είναι σχεδόν γραμμικά εξαρτημένα. Για το λόγο αυτό, αντί της βάσης, χρησιμοποιείται η μέθοδος Arnoldi για την εύρεση ορθοκανονικών διανυσμάτων βάσης $q_1, q_2, \dots, q_n$. Τώρα, το διάνυσμα $x_r \in K_r$ μπορεί να γραφτεί ως $x_r = Q_r y_r$ όπου $y_r \in R^r$ και Q_r είναι ο nxr πίνακας που σχηματίζεται από τα $q_1, q_2, \dots, q_n$. Η μέθοδος Arnoldi παράγει, επίσης, έναν πίνακα Hessenberg $\tilde{H}_r$ τέτοιο ώστε $AQ_r = Q_{r+1}\tilde{H}_r$. Επειδή ο Q_r είναι ορθογωνικός, έχουμε $\|Ax_r - b\| = \|\tilde{H}_r y_r - \beta e_1\|$, όπου $e_1 = (1, 0, 0, \dots, 0)$ είναι το πρώτο διάνυσμα στη βάση του R^{r+1} και $\beta = \|b - Ax_0\|$, όπου x_0 είναι η πρώτη εκτίμηση της λύσης (συνήθως το μηδενικό διάνυσμα). Επομένως, το διάνυσμα x_r μπορεί να βρεθεί ελαχιστοποιώντας τη νόρμα του υπολοίπου $r_r = H_r y_r - \beta e_1$. Αυτό είναι ένα πρόβλημα ελαχίστων τετραγώνων μεγέθους n, από το οποίο προκύπτει και η μέθοδος GMRES. Η μέθοδος παρουσιάζεται στη συνέχεια σε ψευδοκώδικα:

$r_0 = b - Ax_0$, $\beta = \|r_0\|$, $v_1 = r_0 / \beta$
$\tilde{H}_r = 0$
For $j = q$ **to** r
    $w_j = Av_j$
    For $i = 1$ **to** j
        $h_{ij} = (w_j, v_i)$
        $w_j = w_j - h_{ij}v_i$
    $h_{j+1,j} = \|w_j\|_2$
    If $h_{j+1,j} = 0$ **then**
        $m = j$
        Goto End
    $v_{j+1} = w_j / h_{j+1,j}$
End:
    Compute y_m **which minimizes** $\|H_r y_r - \beta e_1\|_2$
    $x_m = x_0 + V_m y_m$

Όπως είναι φανερό από τον ψευδοκώδικα που παρατίθεται, το μεγαλύτερο μειονέκτημα της μεθόδου είναι ότι το πλήθος των υπολογισμών, αλλά και ο χώρος αυξάνονται σχεδόν γραμμικά με τον αριθμό των επαναλήψεων. Εκτός κι αν είμαστε ιδιαίτερα τυχεροί ώστε να επιτύχουμε πολύ γρήγορη σύγκλιση, το κόστος εκτέλεσης της μεθόδου θα γίνεται όλο και πιο απαγορευτικό. Ο συνήθης τρόπος αποφυγής του κινδύνου αυτού, είναι η επανεκκίνηση της μεθόδου έπειτα από έναν επιλεγμένο αριθμό m επαναλήψεων. Τα αποθηκευμένα δεδομένα καθαρίζονται και τα αποτελέσματα της m-οστής επανάληψης χρησιμοποιούνται ως αρχική τιμή για τις επόμενες m επαναλήψεις. Η διαδικασία αυτή επαναλαμβάνεται έως ότου επιτευχθεί σύγκλιση. Η επιλογή της κατάλληλης τιμής m είναι συνήθως μια πολύ δύσκολη απόφαση. Αν το m επιλεγεί πολύ μικρό, τότε η μέθοδος θα έχει πολύ αργή σύγκλιση ή μπορεί η σύγκλιση και να αποτύχει. Από την άλλη πλευρά, αν η τιμή του m είναι μεγαλύτερη απ' όσο απαιτείται, η μέθοδος θα εκτελέσει περιττούς υπολογισμούς και θα χρησιμοποιήσει περισσότερο χώρο απ' όσο απαιτείται. Δυστυχώς, δεν υπάρχει κάποια μέθοδος ακριβούς προσδιορισμού της τιμής m και συνήθως επιλέγεται με βάση εμπειρικά κριτήρια.

2.3.3 Μέθοδος Conjugate Gradient

Η μέθοδος conjugate gradient είναι ένας αλγόριθμος για την αριθμητική επίλυση εκείνων των συστημάτων γραμμικών εξισώσεων, των οποίων ο πίνακας είναι συμμετρικός και θετικά ορισμένος. Η conjugate gradient είναι μια επαναληπτική μέθοδος, γεγονός το οποίο μας επιτρέπει να τη χρησιμοποιήσουμε σε αραιά συστήματα τα οποία είναι πολύ μεγάλα για να επιλυθούν από άμεσες μεθόδους, όπως η παραγοντοποίηση Cholesky. Τέτοια συστήματα προκύπτουν κατά την αριθμητική επίλυση μερικών διαφορικών εξισώσεων.
Η μέθοδος conjugate gradient μπορεί, επίσης, να χρησιμοποιηθεί για την επίλυση προβλημάτων βελτιστοποίησης χωρίς περιορισμούς, όπως ελαχιστοποίηση της ενέργειας.

Περιγραφή της μεθόδου.

Ας υποθέσουμε ότι ο nxn πίνακας A είναι συμμετρικός (δηλαδή $\boldsymbol{A^T = A}$), θετικά ορισμένος (δηλαδή $\boldsymbol{x^T Ax > 0}$ *για κάθε μη μηδενικό διάνυσμα x στο* $\boldsymbol{R^n}$) και πραγματικός.
Συμβολίζουμε τη μοναδική λύση του συστήματος με **x***.

Άμεση Conjugate Gradient

Λέμε ότι δύο διανύσματα *u* και *v* είναι συζυγή (σε σχέση με τον A) αν $\boldsymbol{u^T Av = 0}$.
Εφ' όσον ο A είναι συμμετρικός και θετικά ορισμένος, το αριστερό μέλος ορίζει ένα εσωτερικό γινόμενο

$$\langle u, v\rangle_A := \langle A^T u, v\rangle = \langle Au, v\rangle = \langle u, Av\rangle = u^T Au.$$

Άρα, δύο διανύσματα είναι συζυγή αν είναι ορθογώνια σε σχέση με αυτό το εσωτερικό γινόμενο. Η σχέση αυτή είναι συμμετρική: Αν το u είναι συζυγές του v, τότε και το v είναι συζυγές του u.
Ας υποθέσουμε ότι $\{p_k\}$ είναι μια ακολουθία από n αμοιβαίως συζυγείς κατευθύνσεις. Τότε, η p_k σχηματίζει μια βάση του R^n, ώστε να μπορούμε να επεκτείνουμε τη λύση **x*** της **Ax=b** στη βάση:

$$x* = \sum_{i=1}^{n} a_i p_i$$

Οι συντελεστές δίνονται από τις εξισώσεις:

$$b = Ax* = \sum_{i=1}^{n} a_i A p_i$$

$$p_k^T b = p_k^T Ax* = \sum_{i=1}^{n} a_i p_k^T A p_i = a_k p_k^T A p_k$$

$$a_k = \frac{p_k^T b}{p_k^T A p_k} = \frac{\langle p_k, b\rangle}{\langle p_k, p_k\rangle A} = \frac{\langle p_k, b\rangle}{\|p_k\|_A^2}$$

Το αποτέλεσμα είναι πιθανόν πιο εμφανές λαμβάνοντας υπόψη το εσωτερικό γινόμενο που ορίζεται παραπάνω. Με αυτόν τον τρόπο, προκύπτει η ακόλουθη μέθοδος για την επίλυση της εξίσωσης **Ax = b.** Αρχικά, βρίσκουμε μια ακολουθία από n συζυγείς κατευθύνσεις και στη συνέχεια υπολογίζουμε τους συντελεστές a_k.

Η μέθοδος ως επαναληπτική μέθοδος.

Αν επιλέξουμε τα συζυγή διανύσματα p_k προσεκτικά, τότε ίσως να μην τα χρειαζόμαστε όλα για να επιτύχουμε μια καλή προσέγγιση της λύσης **x***. Έτσι, θέλουμε να θεωρήσουμε τη μέθοδο conjugate gradient ως μια επαναληπτική μέθοδο. Η θεώρηση αυτή θα μας επιτρέψει επιπλέον να επιλύσουμε συστήματα όπου το n είναι τόσο μεγάλο που η άμεση μέθοδος θα πάρει πάρα πολύ χρόνο.

Συμβολίζουμε την αρχική εκτίμηση της λύσης **x*** με x_0. Μπορούμε να υποθέσουμε χωρίς βλάβη της γενικότητας ότι $x_0 = 0$ (ισοδύναμα, μπορούμε να θεωρήσουμε το σύστημα $Az = b - Ax_0$). Ξεκινώντας με τη x_0 αναζητούμε τη λύση και σε κάθε επανάληψη χρειαζόμαστε μια μετρική με βάση την οποία να βλέπουμε αν κινούμαστε πιο κοντά στη λύση **x*** (η οποία μας είναι άγνωστη). Η μετρική αυτή προκύπτει από το γεγονός ότι η λύση **x*** είναι επίσης και το μοναδικό διάνυσμα που ελαχιστοποιεί την εξής συνάρτηση:

$$f(x) = \frac{1}{2}x^T Ax - x^T b, x \in R^n$$

Οπότε, αν η f(x) μειώνεται σε κάθε επανάληψη, αυτό σημαίνει ότι πλησιάζουμε τη λύση **x***.

Η παρατήρηση αυτή μας ωθεί ώστε να πάρουμε ως πρώτο βασικό διάνυσμα p_1 την αρνητική τιμή της κλίσης της f στο σημείο $x = x_0$. Η κλίση αυτή θα είναι ίση με $Ax_0 - b$. Αφού $x_0 = 0$, αυτό σημαίνει ότι παίρνουμε $p_1 = b$.

Έστω r_k το υπόλοιπο στο κ-οστό βήμα:

$r_k = b - Ax_k$.

Ας σημειώσουμε εδώ ότι το r_k είναι η αρνητική τιμή της κλίσης στο $x = x_k$, οπότε σύμφωνα με τη μέθοδο της επαναληπτικής καθόδου θα έπρεπε να κινηθούμε στην κατεύθυνση r_k. Εδώ επιμένουμε ότι οι κατευθύνσεις p_k είναι συζυγείς μεταξύ τους. Επίσης, απαιτούμε η επόμενη κατεύθυνση αναζήτησης να προκύψει από το τρέχον υπόλοιπο και όλες τις προηγούμενες κατευθύνσεις αναζήτησης, το οποίο είναι αρκετά λογικό στην πράξη.

Προκύπτει η ακόλουθη έκφραση:

$$p_{k+1} = r_k - \sum_{i \le k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$$

Ακολουθώντας την κατεύθυνση αυτή, η επόμενη βέλτιστη λύση δίνεται από τη σχέση:

$x_{k+1} = x_k + a_{k+1} p_{k+1}$

όπου

$$a_{k+1} = \frac{p_{k+1}^T r_k}{p_{k+1}^T A p_{k+1}}$$

Ο αλγόριθμος

Ο παραπάνω αλγόριθμος απαιτεί την αποθήκευση όλων των προηγούμενων κατευθύνσεων αναζήτησης και των διανυσμάτων υπολοίπων, όπως και τον υπολογισμό πολλών πολλαπλασιασμών πινάκων επί διάνυσμα, γεγονός που τον καθιστά υπολογιστικά ακριβό. Στην πράξη, τροποποιούμε ελαφρώς τη συνθήκη

υπολογισμού του τελευταίου διανύσματος υπολοίπου, έτσι ώστε όχι να ελαχιστοποιεί τη μετρική που ακολουθεί η κατεύθυνση αναζήτησης, αλλά να την κάνει απλώς ορθογώνια στο προηγούμενο διάνυσμα υπολοίπου. Η ελαχιστοποίηση της μετρικής κατά την κατεύθυνση αναζήτησης θα επιτευχθεί αυτόματα στην περίπτωση αυτή. Με αυτόν τον τρόπο, μπορούμε να καταλήξουμε σε έναν αλγόριθμο ο οποίος απαιτεί την αποθήκευση μόνο των δύο προηγούμενων διανυσμάτων υπολοίπων και της τελευταίας κατεύθυνσης αναζήτησης, και μόνο έναν πολλαπλασιασμό πίνακα επί διάνυσμα.

Στη συνέχεια, περιγράφεται λεπτομερώς ο αλγόριθμος για την επίλυση της εξίσωσης **Ax** = **b**, όπου **A** είναι πραγματικός, συμμετρικός και θετικά ορισμένος πίνακας. Το διάνυσμα εισόδου $\boldsymbol{x_0}$ μπορεί να είναι είτε μια πρώτη εκτίμηση της λύσης είτε το μηδενικό διάνυσμα.

$\boldsymbol{r_0 := b - Ax_0}$
$\boldsymbol{p_0 := r_0}$
$\boldsymbol{k := 0}$
Repeat
$\boldsymbol{a_k := \frac{r_k^T r_k}{p_k^T A p_k}}$
$\boldsymbol{x_{k+1} := x_k + a_k p_k}$
$\boldsymbol{r_{k+1} := r_k - a_k A p_k}$

Αν r_{k+1} είναι αρκετά μικρό τότε έξοδος από το loop

$\boldsymbol{\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}}$

$\boldsymbol{p_{k+1} := r_{k+1} + \beta_k p_k}$
$\boldsymbol{k := k + 1}$

End Repeat
Το αποτέλεσμα είναι το $\boldsymbol{x_{k+1}}$.

2.3.3.1 Preconditioned Conjugate Gradient

Μερικές φορές είναι απαραίτητη η χρήση preconditioner για την εξασφάλιση της γρήγορης σύγκλισης του αλγορίθμου. Η preconditioned conjugate gradient μέθοδος παίρνει την ακόλουθη μορφή:
$\boldsymbol{r_0 := b - Ax_0}$
$\boldsymbol{z_0 := M^{-1} r_0}$
$\boldsymbol{p_0 := z_0}$
$\boldsymbol{k := 0}$
Repeat
$\boldsymbol{a_k := \frac{r_k^T z_k}{p_k^T A p_k}}$
$\boldsymbol{x_{k+1} := x_k + a_k p_k}$
$\boldsymbol{r_{k+1} := r_k - a_k A p_k}$

Αν r_{k+1} είναι αρκετά μικρό τότε έξοδος από το loop

$z_{k+1} := M^{-1} r_{k+1}$

$\beta_k := \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$

$p_{k+1} := z_{k+1} + \beta_k p_k$

$k := k + 1$

End Repeat

Το αποτέλεσμα είναι το x_{k+1}.

Στον παραπάνω αλγόριθμο, **M** είναι ο preconditioner και πρέπει να είναι συμμετρικός και θετικά ορισμένος. Ισοδύναμα, μπορούμε να πούμε ότι εφαρμόζουμε τη μέθοδο conjugate gradient χωρίς χρήση preconditioner στο σύστημα

$E^{-1} A (E^{-1})^T \hat{x} = E^{-1} b$,

όπου

$E E^T = M$

$\hat{x} = E^T x$

Κεφάλαιο 3ο: Εργαλεία Επιστημονικών Υπολογισμών

3.1 Προγραμματιστικά εργαλεία για επιστημονικές εφαρμογές σε αρχιτεκτονικές κατανεμημένης μνήμης

Στο παρόν κεφάλαιο θα παρουσιάσουμε συνοπτικά τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του τεχνικού μέρους της παρούσας εργασίας. Τα εργαλεία που χρησιμοποιήθηκαν είναι το πρότυπο MPI, η βιβλιοθήκη BLAS και η βιβλιοθήκη PETSc. Τα εργαλεία αυτά είναι υλοποιημένα για χρήση με το μοντέλο ανταλλαγής μηνυμάτων και προορίζονται για εφαρμογές που περιέχουν επιστημονικούς υπολογισμούς. Το πρότυπο MPI παρέχει τη διεπαφή για την υλοποίηση του μοντέλου ανταλλαγής μηνυμάτων. Περιέχει έτοιμες ρουτίνες για τη δημιουργία διεργασιών, την κατανομή των δεδομένων και, φυσικά, την επικοινωνία με ανταλλαγή μηνυμάτων. Οι βιβλιοθήκες BLAS και PETSc παρέχουν υλοποιημένες τις ρουτίνες και τις δομές που χρησιμοποιούνται συχνά στις επιστημονικές εφαρμογές, όπως δομές αραιών πινάκων και επιλυτές γραμμικών συστημάτων, πράξεις μεταξύ πινάκων και διανυσμάτων κ.ά.. Η χρήση των εργαλείων αυτών διευκόλυνε ιδιαίτερα την υλοποίηση των αλγόριθμων της παρούσας εργασίας, γι' αυτό και κρίθηκε σκόπιμη η αφιέρωση ενός κεφαλαίου για την περιγραφή τους.

3.2 Το Πρότυπο MPI

Η ανταλλαγή μηνυμάτων είναι μια προγραμματιστική τεχνική που χρησιμοποιείται ευρέως στα παράλληλα συστήματα υπολογιστών, κυρίως στα συστήματα με κατανεμημένη μνήμη (Scalable Parallel Computers) και σε δίκτυα σταθμών εργασίας (Networks of Workstations). Αν και υπάρχουν διάφορες παραλλαγές, η βασική ιδέα της επικοινωνίας μεταξύ διεργασιών με τη χρήση μηνυμάτων είναι πλήρως κατανοητή. Τα τελευταία δέκα χρόνια έχει γίνει ουσιαστική πρόοδος όσον αφορά στην προσαρμογή σημαντικών εφαρμογών στο μοντέλο αυτό. Έτσι, λοιπόν, ήταν επιτακτική η ανάγκη για τον ορισμό της σύνταξης και της σημασιολογίας ενός πρότυπου πυρήνα ρουτινών βιβλιοθήκης ώστε να χρησιμεύσουν σε ένα μεγάλο πλήθος χρηστών και να είναι υλοποιήσιμες σε ένα μεγάλο εύρος υπολογιστικών συστημάτων. Την προσπάθεια αυτή έχει αναλάβει τα τελευταία χρόνια το Message Passing Interface (MPI) Forum, μία ομάδα περισσότερων από 80 ατόμων από περισσότερους από 40 οργανισμούς, οι οποίοι εκπροσωπούν εμπόρους παράλληλων συστημάτων, βιομηχανικούς χρήστες, ιδιωτικά και δημόσια ερευνητικά κέντρα και πανεπιστήμια.

Οι σχεδιαστές του MPI επέλεξαν να χρησιμοποιήσουν τα πιο ελκυστικά χαρακτηριστικά των προϋπάρχοντων συστημάτων ανταλλαγής μηνυμάτων αντί να επικεντρωθούν σε ένα από αυτά. Το MPI έχει, λοιπόν, επιρροές από εργασίες διάφορων ερευνητικών κέντρων και εταιρειών. Το πρότυπο MPI ορίζει τη διεπαφή χρήστη και τη λειτουργικότητα ενός μεγάλου συνόλου δυνατοτήτων για πέρασμα μηνυμάτων. Από την ολοκλήρωσή του το 1994 το MPI είναι κοινώς αποδεκτό και έχει χρησιμοποιηθεί ευρέως. Είναι διαθέσιμες διάφορες υλοποιήσεις τόσο για συστήματα κατανεμημένης μνήμης όσο και για δίκτυα σταθμών εργασίας. Έτσι, λοιπόν, το MPI πέτυχε έναν από τους στόχους του, την προσθήκη αξιοπιστίας στα παράλληλα συστήματα.

Ο κύριος στόχος του MPI, όπως και σχεδόν κάθε προτύπου, είναι η επίτευξη ενός βαθμού φορητότητας ανάμεσα σε διαφορετικές πλατφόρμες, συγκρίσιμη με αυτή που προσφέρεται από γλώσσες προγραμματισμού, όπως για παράδειγμα η Fortran. Αυτό σημαίνει ότι ο ίδιος πηγαίος κώδικας MPI να μπορεί να εκτελεστεί σε πληθώρα

μηχανημάτων με μόνη προϋπόθεση τη διαθεσιμότητα της βιβλιοθήκης MPI, ενώ παράλληλα να δίνεται η δυνατότητα για εκμετάλλευση των πλεονεκτημάτων κάθε ξεχωριστού συστήματος. Αν και τις περισσότερες φορές το πέρασμα μηνυμάτων είναι στενά συνδεδεμένο με τα παράλληλα συστήματα κατανεμημένης μνήμης, ο ίδιος κώδικας μπορεί να εκτελεστεί και σε έναν παράλληλο υπολογιστή μοιραζόμενης μνήμης επίσης. Μπορεί να εκτελεστεί από ένα δίκτυο σταθμών εργασίας ή ακόμη και από ένα σύνολο διεργασιών ενός μοναδικού σταθμού εργασίας. Γνωρίζοντας την ύπαρξη αποδοτικών υλοποιήσεων με MPI σε μεγάλη ποικιλία προβλημάτων παρέχεται ένας υψηλός βαθμός ευελιξίας στη συγγραφή και τον έλεγχο του κώδικα και στην επιλογή της κατάλληλης πλατφόρμας εκτέλεσης.

Η φορητότητα ήταν από τους κύριους στόχους του προτύπου, όπως και η επίδοση στην οποία τελικά επικεντρώθηκαν οι εμπνευστές του ώστε να το κάνουν ευρέως χρησιμοποιούμενο. Το κρίσιμο σημείο είναι ότι το MPI σχεδιάστηκε προσεχτικά ώστε να επιτρέπει τη δημιουργία αποδοτικών υλοποιήσεων και φαίνεται πως αυτές οι σχεδιαστικές επιλογές ήταν τελικά σωστές, όπως αποδεικνύουν οι αποδοτικές MPI εφαρμογές που έχουν υλοποιηθεί σε πληθώρα συστημάτων.

Το MPI σχεδιάστηκε με σκοπό να ενθαρρύνει την επικάλυψη της επικοινωνίας και των υπολογισμών ώστε να μπορεί να εκμεταλλευτεί ευφυείς πράκτορες επικοινωνίας και να αποκρύψεις το χρόνο που απαιτείται από την έναρξη της επικοινωνίας ως την ολοκλήρωσή της.

Η επεκτασιμότητα είναι πολύ σημαντικός στόχος της παράλληλης επεξεργασίας. Το MPI υποστηρίζει επεκτασιμότητα μέσω διάφορων χαρακτηριστικών του. Για παράδειγμα, είναι δυνατόν μια εφαρμογή να δημιουργήσει υποσύνολα διεργασιών οι οποίες με τη σειρά τους επιτρέπουν συλλογικές λειτουργίες επικοινωνίας περιορισμένες στην εμβέλεια των εμπλεκόμενων διεργασιών.

Συμπερασματικά, το MPI, όπως όλα τα καλά σχεδιασμένα πρότυπα, είναι χρήσιμο γιατί ορίζει την ελάχιστη συμπεριφορά υλοποιήσεων που χρησιμοποιούν αποστολή μηνυμάτων. Το γεγονός αυτό απαλλάσσει τον προγραμματιστή από την ευθύνη να αντιμετωπίσει διάφορα προβλήματα που μπορούν να προκύψουν. Ένα παράδειγμα είναι το γεγονός ότι το MPI φροντίζει για την αξιόπιστη μετάδοση των μηνυμάτων. Έτσι, ο χρήστης δε χρειάζεται να ελέγξει αν κάθε μήνυμα παραδόθηκε σωστά.

3.2.1 Η έννοια της διεργασίας

Μια εφαρμογή MPI μπορεί να θεωρηθεί ως μια συλλογή από ταυτόχρονες διεργασίες που επικοινωνούν. Ένα πρόγραμμα περιέχει κώδικα, γραμμένο από τον προγραμματιστή της εφαρμογής, ο οποίος συνδέεται με μια βιβλιοθήκη από ρουτίνες που παρέχει η υλοποίηση MPI. Σε κάθε διεργασία δίνεται ένας μοναδικός βαθμός *(rank),* δηλαδή ένας ακέραιος από 0 ως n-1, αν n είναι οι διεργασίες που αποτελούν την εφαρμογή. Τα αναγνωριστικά αυτά χρησιμοποιούνται από τις διεργασίες του MPI για να αναγνωρίζουν η μία την άλλη όταν στέλνουν ή λαμβάνουν μηνύματα, όταν εκτελούν συλλογικές λειτουργίες και γενικά για να συνεργάζονται. Οι διεργασίες του MPI μπορούν να εκτελούνται στον ίδιο ή σε διαφορετικούς επεξεργαστές ταυτόχρονα.

3.2.2 Communicator

Μια σημαντική απαίτηση σε όλα τα συστήματα ανταλλαγής μηνυμάτων είναι να εγγυώνται έναν ασφαλή χώρο επικοινωνίας όπου μη σχετιζόμενα μεταξύ τους μηνύματα να είναι διαχωρισμένα το ένα από το άλλο. Για παράδειγμα, τα μηνύματα

βιβλιοθήκης μπορούν να αποσταλούν και να ληφθούν χωρίς παρεμβολές από άλλα μηνύματα που παράγονται στο σύστημα.

Στο MPI, όπου δεν υπάρχει εικονική μηχανή, η χρήση μιας απλής ετικέτας σε κάθε μήνυμα δεν είναι αρκετή για τον ασφαλή διαχωρισμό των μηνυμάτων βιβλιοθήκης από τα μηνύματα του χρήστη. Για την επίτευξη της απαίτησης αυτής, εισάγεται στο MPI η έννοια του *communicator*. Ο *communicator* μπορεί να θεωρηθεί ως μια συσχέτιση ανάμεσα σε μια ομάδα διεργασιών στο γενικό πλαίσιο της επικοινωνίας. Ο *communicator* είναι ένα αντικείμενο που μπορεί να προσπελαστεί μέσω ενός χειριστή τύπου MPI_COMM.

Βαθμός Διεργασίας

Στις διεργασίες που ανήκουν σε κάποιον communicator ανατίθενται συνεχόμενοι ακέραιοι από μηδέν ως n, όπου n το μέγεθος της ομάδας του communicator πλην ένα, ως αναγνωριστικά. Τα αναγνωριστικά αυτά, που ονομάζονται βαθμοί (ranks), χρησιμοποιούνται για το διαχωρισμό των διεργασιών μέσα στην ίδια ομάδα. Για παράδειγμα, σε διεργασίες με διαφορετικό βαθμό μπορούν να ανατεθούν διαφορετικές εργασίες για εκτέλεση. Μια διεργασία μπορεί να βρει το βαθμό της μέσα σε έναν communicator καλώντας τη συνάρτηση MPI_Comm_Rank ως εξής:

```
MPI_Comm communicator; /* communicator handle */
int my_rank; /* ο βαθμός της καλούσας διεργασίας */
MPI_Comm_rank(communicator, &my_rank);
```

Μέγεθος μιας ομάδας

Το μέγεθος μιας ομάδας συσχετιζόμενης με έναν communicator μπορεί να βρεθεί καλώντας τη συνάρτηση MPI_Comm_size(). Η συνάρτηση αυτή δέχεται έναν communicator και επιστρέφει το μέγεθος της αντίστοιχης ομάδας ως εξής:

```
MPI_Comm communicator; /*communicator handle */
int number_of_tasks;
MPI_Comm_size(communicator, &number_of_tasks)
```

3.2.3 Επικοινωνία

Η επικοινωνία ανάμεσα στις διεργασίες του MPI βασίζεται στην τεχνική της ανταλλαγής μηνυμάτων. Το MPI χρησιμοποιεί ένα πλούσιο σύνολο συναρτήσεων για την αποστολή και τη λήψη μηνυμάτων. Η επικοινωνία μεταξύ δύο διεργασιών περιλαμβάνει τα εξής στοιχεία:

1. Αποστολέας, ο οποίος προσδιορίζεται συνήθως από το βαθμό του.
2. Παραλήπτης, ο οποίος προσδιορίζεται επίσης από το βαθμό του.
3. Τα δεδομένα του μηνύματος.
4. Η ετικέτα του μηνύματος, η οποία χρησιμοποιείται για την ταξινόμηση των μηνυμάτων που ανταλλάσσονται μεταξύ δύο διεργασιών.
5. Ο Communicator, ο οποίος παρέχει το γενικό πλαίσιο για την επικοινωνία.

3.2.3.1 Blocking Επικοινωνία

Οι βασικές συναρτήσεις για την αποστολή και τη λήψη μηνυμάτων στο MPI είναι η *blocking send* και η *blocking receive*. Υπάρχουν διάφορες παραλλαγές των

συναρτήσεων αυτών που εξυπηρετούν διαφορετικά είδη επικοινωνίας μεταξύ διεργασιών. Το MPI υποστηρίζει τις παρακάτω μεθόδους:

Standard Send

Με τη μέθοδο αυτή, ο αποστολέας θα «μπλοκάρει» μέχρι το μήνυμά του να αντιγραφεί με ασφάλεια είτε στον αντίστοιχο χώρο αποθήκευσης του παραλήπτη είτε σε έναν προσωρινό χώρο αποθήκευσης του συστήματος. Αν το μήνυμα πρέπει να αποθηκευτεί ή όχι εξαρτάται από την εκάστοτε υλοποίηση του MPI. Μόλις γίνει επιστροφή της κλήσης send, η προσωρινή μνήμη του αποστολέα μπορεί να επαναχρησιμοποιηθεί για άλλους σκοπούς από τον αποστολέα. Η συνάρτηση αυτή ορίζεται ως εξής:

```
MPI_Send(buf, count, data_type, to_whom, tag,
communicator)
```

Η συνάρτηση αυτή θα στείλει το μήνυμα που είναι αποθηκευμένο στην αρχή της διεύθυνσης buf στη διεργασία με rank to_whom. Το μήνυμα αποτελείται από count στοιχεία, καθένα από τα οποία είναι τύπου data_type. Η ετικέτα του μηνύματος είναι η παράμετρος tag. Τόσο ο αποστολέας όσο και ο παραλήπτης πρέπει να είναι μέλη του ίδιου communicator communicator. Αν χρησιμοποιείται προσωρινή μνήμη για την αποθήκευση του εξερχόμενου μηνύματος, ο αποστολέας θα συνεχίσει την εκτέλεσή του χωρίς να χρειάζεται να περιμένει για την αντίστοιχη λειτουργία receive.

Blocking Receive

Η πρότυπη συνάρτηση λήψης στο MPI είναι η *blocking receive*. Μια κλήση σε αυτή τη συνάρτηση δεν θα επιστρέψει μέχρι να παραλάβει το μήνυμα που αναμένει στην προσωρινή της μνήμη. Η συνάρτηση ορίζεται ως εξής:

```
MPI_Recv(buf, count, data_type, from_whom, tag,
communicator,&status)
```

Η συνάρτηση αυτή θα επιλέξει ένα μήνυμα από τον αποστολέα με rank from_whom με ετικέτα tag και θα το αποθηκεύσει στη διεύθυνση που ξεκινά από buf. Πρόσθετη πληροφορία για την κατάσταση της λειτουργίας θα επιστραφεί στη μεταβλητή status. Η μεταβλητή αυτή είναι συνήθως μια δομή που αποτελείται από δύο πεδία MPI_SOURCE και MPI_TAG που αντιστοιχούν στο βαθμό του αποστολέα και στην ετικέτα του μηνύματος. Και εδώ ο αποστολέας και ο παραλήπτης πρέπει να ανήκουν στον communicator.

3.2.3.2 Non-Blocking Επικοινωνία

Το MPI υποστηρίζει non-blocking επικοινωνία κατά την οποία μια διεργασία μπορεί να ξεκινήσει μια λειτουργία αποστολής ή παραλαβής, να εκτελέσει στη συνέχεια κάποια άλλη εργασία και έπειτα να επιστρέψει για να ελέγξει την κατάσταση της λειτουργίας ανταλλαγής μηνύματος. Αυτές οι δυνατότητες μπορούν να χρησιμοποιηθούν σε συνδυασμό με τις standard λειτουργίες και δεν είναι ανάγκη να απαντώνται σε ζεύγη. Η χρήση μιας non-blocking λειτουργίας αποστολής (λήψης) μπορεί να επιτευχθεί σε τρία βήματα:

1. Έναρξη λειτουργίας αποστολής (λήψης) καλώντας τη συνάρτηση MPI_Isend() (MPI_Irecv()).
2. Εκτέλεση κάποιων υπολογισμών κατά τη διάρκεια του χρόνου επικοινωνίας.

3. Ολοκλήρωση της επικοινωνίας καλώντας την MPI_Wait() ή την MPI_Test().

Έναρξη της Nonblocking Επικοινωνίας

Η συνάρτηση που χρησιμοποιείται για μια nonblocking αποστολή είναι η εξής:

```
MPI_Isend(buf, count, data_type, to_whom, tag,
communicator,&request)
```

Μια κλήση στη συνάρτηση αυτή θα επιστρέψει πριν αντιγραφεί το μήνυμα από την προσωρινή μνήμη του αποστολέα. Όλα τα ορίσματα της συνάρτησης αυτής, εκτός από ένα, έχουν το ίδιο νόημα όπως στις υπόλοιπες συναρτήσεις αποστολής. Το τελευταίο όρισμα, request, είναι ένα αντικείμενο του συστήματος που επιστρέφεται όταν καλείται η συνάρτηση και μπορεί να προσπελαστεί μέσω ενός χειριστή**.** Χρησιμοποιείται για την αναγνώριση και το συνδυασμό διάφορων λειτουργιών επικοινωνίας. Ομοίως, για nonblocking λήψη ενός μηνύματος χρησιμοποιείται η εξής συνάρτηση:

```
MPI_Irecv(buf, count, data_type, from_whom, tag,
communicator,&request)
```

Μια κλήση στη συνάρτηση αυτή θα ξεκινήσει τη λειτουργία λήψης. Θα επιστρέψει αμέσως χωρίς να χρειάζεται να περιμένει να αποθηκευτεί κάποιο μήνυμα στην προσωρινή μνήμη λήψης. Όπως και στην περίπτωση της αποστολής, το όρισμα request μπορεί να χρησιμοποιηθεί αργότερα για την εξακρίβωση της κατάστασης της λειτουργίας.

Ολοκλήρωση της Non-blocking Επικοινωνίας

Το MPI παρέχει ρουτίνες για τον έλεγχο της ολοκλήρωσης των λειτουργιών επικοινωνίας που έχουν ξεκινήσει. Η ολοκλήρωση μιας non-blocking λειτουργίας αποστολής υποδηλώνει ότι τα δεδομένα έχουν αντιγραφεί από τη μνήμη του αποστολέα και η μνήμη αυτή μπορεί να χρησιμοποιηθεί από άλλες λειτουργίες. Αντίστοιχα, η ολοκλήρωση μιας non-blocking λειτουργίας λήψης υποδηλώνει ότι τα δεδομένα έχουν ήδη τοποθετηθεί στον κατάλληλο χώρο αποθήκευσης ώστε ο παραλήπτης να είναι σε θέση να τα διαβάσει.
Η ακόλουθη συνάρτηση χρησιμοποιείται για τον έλεγχο της ολοκλήρωσης μιας non-blocking λειτουργίας:

```
MPI_Test(request, &flag, &status)
```

Μια κλήση στη συνάρτηση αυτή επιστρέφει την τρέχουσα κατάσταση της λειτουργίας που δηλώνεται μέσω της μεταβλητής request. Το όρισμα flag θα τεθεί true αν η επικοινωνία έχει ολοκληρωθεί, αλλιώς θα πάρει την τιμή false. Το πεδίο status θα επιστρέψει πρόσθετη πληροφορία για την κατάσταση της λειτουργίας.
Η ακόλουθη συνάρτηση θα περιμένει μέχρι η επικοινωνία να ολοκληρωθεί:

```
MPI_Wait(request, &status)
```

Μια κλήση στην MPI_Wait() θα επιστρέψει όταν ολοκληρωθεί η λειτουργία που ορίζεται από τη μεταβλητή request.

3.2.4 Συγχρονισμός

Οι δομές συγχρονισμού χρησιμοποιούνται για την επιβολή μιας συγκεκριμένης σειράς εκτέλεσης ανάμεσα στις λειτουργίες των παράλληλων διεργασιών. Σε μερικές περιπτώσεις, οι παράλληλες διεργασίες απαιτείται να συγχρονιστούν μεταξύ τους σε κάποιο σημείο κατά τη διάρκεια της εκτέλεσης. Τα μέλη μιας ομάδας ίσως χρειαστεί να περιμένουν σε κάποιο σημείο συγχρονισμού μέχρι όλες οι διεργασίες να φτάσουν στο ίδιο σημείο. Ο συγχρονισμός στο MPI μπορεί να επιτευχθεί είτε με την ανταλλαγή μηνυμάτων, χρησιμοποιώντας blocking μεθόδους, είτε με τη χρήση barriers.

Barriers

Οι διεργασίες που ανήκουν σε μία ομάδα μπορούν να συγχρονιστούν σε ένα σημείο χρησιμοποιώντας ένα barrier. Καμία διεργασία δεν μπορεί να προχωρήσει πέραν του σημείου μέχρι ότου όλες οι διεργασίες να φτάσουν στο barrier. Ανάλογα με τον communicator, μπορούμε να συγχρονίσουμε όλες τις διεργασίες μιας ομάδας ή ένα υποσύνολο αυτών. Η συνάρτηση που εκτελεί την παραπάνω λειτουργία ορίζεται ως εξής:

```
MPI_Barrier(communicator)
```

Ο συγχρονισμός επιτυγχάνεται αν όλες οι διεργασίες στην ομάδα του communicator καλέσουν τη συνάρτηση αυτή. Μία κλήση στη συνάρτηση αυτή επιστρέφει όταν όλα τα μέλη της ομάδας του communicator έχουν εκτελέσει την δική τους κλήση στην MPI_Barrier().

3.2.5 Collective Operations

Συλλογικές λειτουργίες (collective operations) στο MPI είναι εκείνες που εφαρμόζονται σε όλα τα μέλη μιας ομάδας ενός communicator. Μια συλλογική λειτουργία ορίζεται συνήθως στα πλαίσια μιας ομάδας διεργασιών. Η λειτουργία εκτελείται όταν όλες οι διεργασίες της ομάδας καλέσουν την αντίστοιχη ρουτίνα με τις κατάλληλες παραμέτρους. Υπάρχουν τρεις τύποι συλλογικών λειτουργιών: ελέγχου διεργασιών, καθολικού υπολογισμού και μεταφοράς δεδομένων. Η συνάρτηση MPI_Barrier()που περιγράφηκε προηγουμένως μπορεί να θεωρηθεί ως συλλογική λειτουργία ελέγχου διεργασιών.

Καθολικός Υπολογισμός (Global Computation)

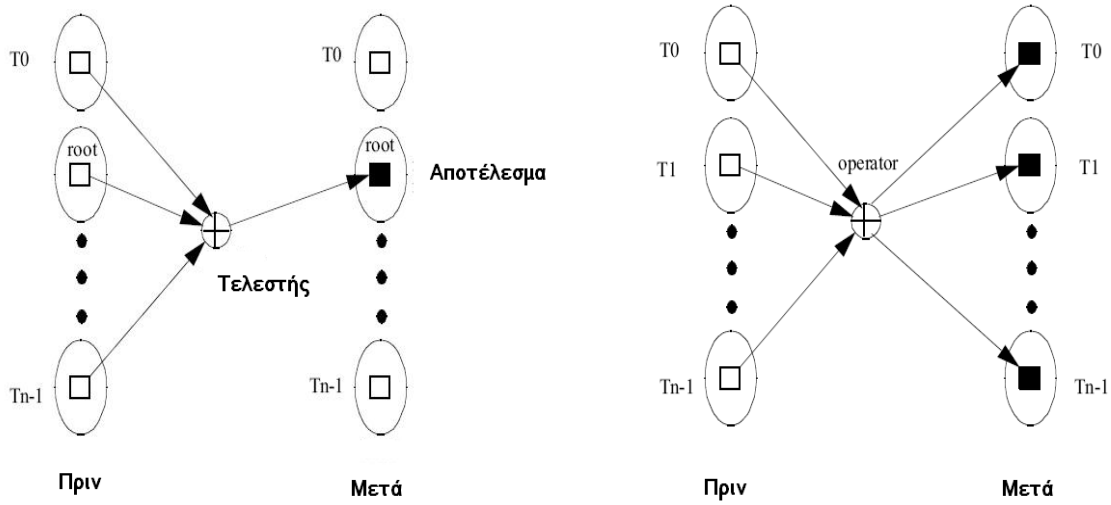
Στην κατηγορία αυτή ανήκουν λειτουργίες αναγωγής και σάρωσης. Μια λειτουργία αναγωγής είναι μια συσχετιστική, τροποποιητική λειτουργία που εφαρμόζεται πάνω σε δεδομένα που παρέχονται από διεργασίες μιας ομάδας. Μια λειτουργία αναγωγής μπορεί να είναι προκαθορισμένη από το MPI, όπως άθροισμα, εύρεση ελάχιστου-μέγιστου κ.ά. ή μια ορισμένη από το χρήστη συνάρτηση. Το αποτέλεσμα της αναγωγής μπορεί να σταλεί σε κάθε διεργασία της ομάδας ή σε μία μόνο η οποία ονομάζεται ρίζα (root).
Το MPI παρέχει την παρακάτω συνάρτηση αναγωγής με την οποία τα αποτέλεσμα επιστρέφεται μόνο στη διεργασία-ρίζα:

```
MPI_Reduce(sbuf, rbuf, n, data_type, op, rt,
communicator)
```

Μια παραλλαγή της παραπάνω συνάρτησης είναι η λειτουργία αναγωγής κατά την οποία το αποτέλεσμα αποστέλλεται σε όλες τις διεργασίες που είναι μέλη της ομάδας. Η συνάρτηση ορίζεται ως εξής:

```
MPI_Allreduce(sbuf, rbuf, n, data_type, op, communicator)
```

Τα ορίσματα έχουν την ίδια σημασία όπως στην MPI_Reduce(). Οι δύο παραπάνω λειτουργίες παρουσιάζονται σχηματικά στο Σχήμα 3.1.



Σχήμα 3.1: Αριστερά παρουσιάζεται η λειτουργία Reduce και δεξιά η λειτουργία AllReduce.

Μεταφορά Δεδομένων

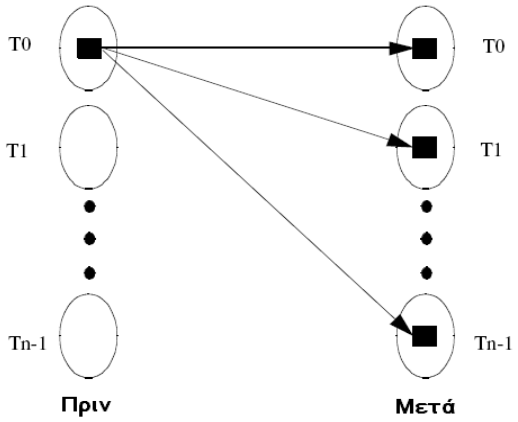
Το MPI υποστηρίζει μεγάλη ποικιλία συλλογικών συναρτήσεων για τη μεταφορά δεδομένων. Οι βασικές λειτουργίες που καλύπτονται είναι οι *broadcast, scatter* και *gather*. Με τη λειτουργία *broadcast*, μια διεργασία στέλνει το ίδιο μήνυμα σε κάθε μέλος μιας ομάδας. Μια λειτουργία *scatter* επιτρέπει σε μια διεργασία να στείλει διαφορετικό μήνυμα σε κάθε μέλος, ενώ η λειτουργία *gather* είναι η δυική της *scatter*, κατά την οποία μία διεργασία θα λάβει ένα μήνυμα από κάθε μέλος μιας ομάδας διεργασιών.

Broadcasting

Το MPI παρέχει την ακόλουθη συνάρτηση για την αποστολή ενός μηνύματος από τη root-διεργασία σε όλες τις υπόλοιπες στην ομάδα ενός communicator:

```
MPI_Bcast(buffer, n, data_type, root, communicator)
```

Η συνάρτηση αυτή πρέπει να κληθεί από όλες τις διεργασίες μιας ομάδας χρησιμοποιώντας τα ίδια ορίσματα για το βαθμό της root-διεργασίας και τον communicator. Τα περιεχόμενα της προσωρινής μνήμης αποστολής της root-διεργασίας θα αντιγραφούν στις μνήμες όλων των υπόλοιπων διεργασιών (βλέπε Σχήμα 3.2).



Σχήμα 3.2 : Λειτουργία Broadcast

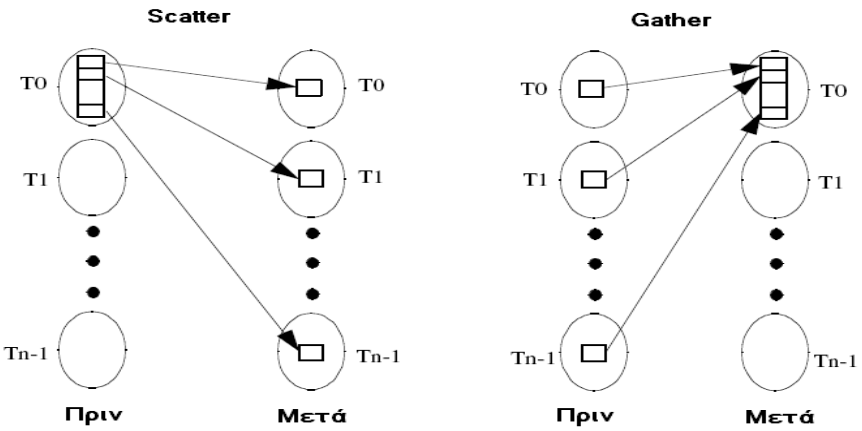
Scatter – Gather

Η συνάρτηση *scatter* επιτρέπει σε μια διεργασία να κατανείμει τα δεδομένα της μνήμης της σε κάθε μέλος μιας ομάδας διεργασιών ενώ η συνάρτηση *gather* επιτρέπει σε μια διεργασία να συλλέξει κομμάτια δεδομένων από τις άλλες διεργασίες για να χτίσει το περιεχόμενο της μνήμης της. Οι δύο συναρτήσεις ορίζονται ως εξής:

```
MPI_Scatter(sbuf, n, stype, rbuf, m, rtype, rt, communicator)
```

```
MPI_Gather(sbuf, n, stype, rbuf, m, rtype, rt, communicator)
```

Στη *scatter* η προσωρινή μνήμη αποστολής της root-διεργασίας χωρίζεται σε τμήματα μεγέθους n. Τα πρώτα n στοιχεία στη μνήμη της root-διεργασίας αντιγράφονται στο χώρο αποθήκευσης του πρώτου μέλους της ομάδας, τα επόμενα n στοιχεία στο χώρο του δεύτερου μέλους κ.ο.κ..
Στη *gather,* κάθε διεργασία (συμπεριλαμβανομένης της root) στέλνει τα περιεχόμενα της προσωρινής της μνήμης στη root-διεργασία. Η root-διεργασία λαμβάνει τα μηνύματα και τα αποθηκεύει στη σειρά με βάση το βαθμό της κάθε διεργασίας.
Οι λειτουργίες αυτές παρουσιάζονται στο Σχήμα 3.3.



Σχήμα 3.3: Λειτουργίες Scatter και Gather

3.3 Η βιβλιοθήκη BLAS

Η BLAS (Basic Linear Algebra Subprograms) είναι μια συλλογή από ρουτίνες οι οποίες προσφέρουν έναν προκαθορισμένο τρόπο εκτέλεσης βασικών πράξεων πινάκων και διανυσμάτων. Έχει χρησιμοποιηθεί ευρέως στη συγγραφή αλγεβρικού λογισμικού υψηλής επίδοσης, όπως για παράδειγμα στη δημιουργία του λογισμικού LAPACK και της PETSc. Η BLAS είναι διαθέσιμη τόσο ως διεπαφή σε ANSI C όσο και σε FORTRAN 77/90.

Η BLAS είναι λογισμικό ανοιχτού κώδικα, διαθέσιμο στο διαδίκτυο (http://www.netlib.org/blas). Μπορεί να χρησιμοποιηθεί ελεύθερα στη δημιουργία εμπορικών πακέτων λογισμικού.

Εκτός από τη standard έκδοση της BLAS, έχουν αναπτυχθεί βελτιστοποιημένες εκδόσεις της βιβλιοθήκης για μια πληθώρα αρχιτεκτονικών. Εταιρείες όπως η AMD, η Apple, η IBM, η Intel, η SUN και άλλες, παρέχουν βελτιστοποιημένες εκδόσεις της βιβλιοθήκης, ειδικά σχεδιασμένες για κάθε αρχιτεκτονική.

3.3.2 Λειτουργικότητα

Ανάλογα με τη λειτουργικότητά τους, οι ρουτίνες της BLAS κατηγοριοποιούνται στα εξής τρία επίπεδα:

- Επίπεδο 1: Ρουτίνες βαθμωτών και διανυσματικών πράξεων.
- Επίπεδο 2: Ρουτίνες για πράξεις διανυσμάτων με πίνακες.
- Επίπεδο 3: Ρουτίνες για πράξεις πινάκων με πίνακες.

Οι ρουτίνες της BLAS παρέχονται μέχρι και σε τέσσερεις διαφορετικές εκδόσεις, ανάλογα με τον τύπο δεδομένων των ορισμάτων – πραγματικοί απλής ακρίβειας, πραγματικοί διπλής ακρίβειας, μιγαδικοί απλής ακρίβειας και μιγαδικοί διπλής ακρίβειας. Για τον διαχωρισμό των ρουτινών, χρησιμοποιούνται τα γράμματα S, D, C και Z, αντίστοιχα πριν από το όνομα της ρουτίνας. Γενικά, μια ρουτίνα BLAS έχει όνομα BLAS_x<name>, όπου x είναι ένα από τα παραπάνω γράμματα και δηλώνει τον τύπο δεδομένων, ενώ το <name> αναφέρεται στη λειτουργία που επιτελεί η ρουτίνα. Στις ρουτίνες επιπέδου 2 και 3, συμπεριλαμβάνεται, επίσης, πληροφορία για τον τύπο του πίνακα στο <name>. Οι διάφοροι τύποι πινάκων που υποστηρίζονται και η ονοματολογία που χρησιμοποιείται παρουσιάζονται στον Πίνακα 3.1.

Πίνακας 3.1: Τύποι Πινάκων στις ρουτίνες της BLAS

Ονοματολογία	Τύπος Πίνακα
GB	General Band
GE	General (μη συμμετρικός)
HB	Hermitian Band
HE	Hermitian
HP	Hermitian, packed storage
SB	Symmetric Band
SP	Symmetric, packed storage
SY	Symmetric
TB	Triangular Band
TP	Triangular, packed storage
TR	Triangular
US	Unstructured Sparse

3.3.2.1 Ρουτίνες Επιπέδου 1

Οι ρουτίνες της BLAS που αφορούν βαθμωτές και διανυσματικές λειτουργίες παρουσιάζονται στον Πίνακα 3.2.

Πίνακας 3.2: Ρουτίνες BLAS Επιπέδου 1

Ρουτίνα ή Ομάδα Συναρτήσεων	Τύπος Δεδομένων	Λειτουργία
_asum	s, d, c, z	Άθροισμα μέτρων διανυσμάτων
_axpy	s, d, c, z	Γινόμενο διανύσματος με σταθερά
_copy	s, d, c, z	Αντιγραφή διανύσματος
_dot	s, d	Εσωτερικό γινόμενο διανυσμάτων
_sdot	s, d	Εσωτερικό γινόμενο διανυσμάτων εκτεταμένης ακρίβειας
_dotc	c, z	Εσωτερικό γινόμενο συζυγών διανυσμάτων
_dotu	c, z	Εσωτερικό γινόμενο μη συζυγών διανυσμάτων
_nrm2	s, d, c, z	Ευκλείδεια νόρμα
_rot	s, d, c, d	Περιστροφή
_scal	s, d, c, z	Ρουτίνες πράξεων διανύσματος με σταθερά
_swap	s, d, c, z	Ρουτίνες ανταλλαγής διανυσμάτων
i_amax	s, d, c, z	Μέγιστη απόλυτη τιμή στοιχείου διανύσματος
i_amin	s, d, c, z	Ελάχιστη απόλυτη τιμή στοιχείου διανύσματος
dcabs1	d	Απόλυτη τιμή μιγαδικού διπλής ακρίβειας

3.3.2.2 Ρουτίνες Επιπέδου 2

Οι ρουτίνες BLAS που αναφέρονται σε πράξεις διανυσμάτων με πίνακες παρουσιάζονται στον Πίνακα 3.3:

Πίνακας 3.3: Ρουτίνες BLAS Επιπέδου 2

Ρουτίνα ή Ομάδα Συναρτήσεων	Τύπος Δεδομένων	Λειτουργία
_gbmv, _gemv, _sbmv, _spmv, _symv	s, d, c, z	Γινόμενο Πίνακα επί διάνυσμα
_ger, _gerc, _geru, _spr, _spr2, _syr, _syr2	s, d	Ενημέρωση Πίνακα
_hbmv, _hemv, _hpmv	c, z	Γινόμενο Ερμιτιανού Πίνακα επί διάνυσμα
_her, _her2, _hpr, _hpr2	c, z	Ενημέρωση Ερμιτιανού Πίνακα
_tbmv, _tpmv, _trmv	s, d, c, z	Γινόμενο τριγωνικού Πίνακα επί διάνυσμα
_tbsv, _trsv, _tpsv	s, d, c, z	Επίλυση Γραμμικού Συστήματος Εξισώσεων με τριγωνικό Πίνακα

3.3.2.3 Ρουτίνες Επιπέδου 3

Στο επίπεδο αυτό ανήκουν οι ρουτίνες που εκτελούν πράξεις μεταξύ πινάκων. Οι βασικότερες από αυτές αναφέρονται στον Πίνακα 3.4 που ακολουθεί:

Πίνακας 3.4: Ρουτίνες BLAS Επιπέδου 3

Ρουτίνα ή Ομάδα Συναρτήσεων	Τύπος Δεδομένων	Λειτουργία
_gemm	s, d, c, z	Γινόμενο Πίνακα επί Πίνακα γενικού τύπου Πινάκων
_hemm	c, z	Γινόμενο Πίνακα επί Πίνακα Ερμιτιανών Πινάκων
_symm	s, d, c, z	Γινόμενο Πίνακα επί Πίνακα Συμμετρικών Πινάκων
_trmm	s, d, c, z	Γινόμενο Πίνακα επί Πίνακα Τριγωνικών Πινάκων
_trsm	s, d, c, z	Γραμμική Επίλυση Συστήματος Τριγωνικών Πινάκων

3.4 PETSc

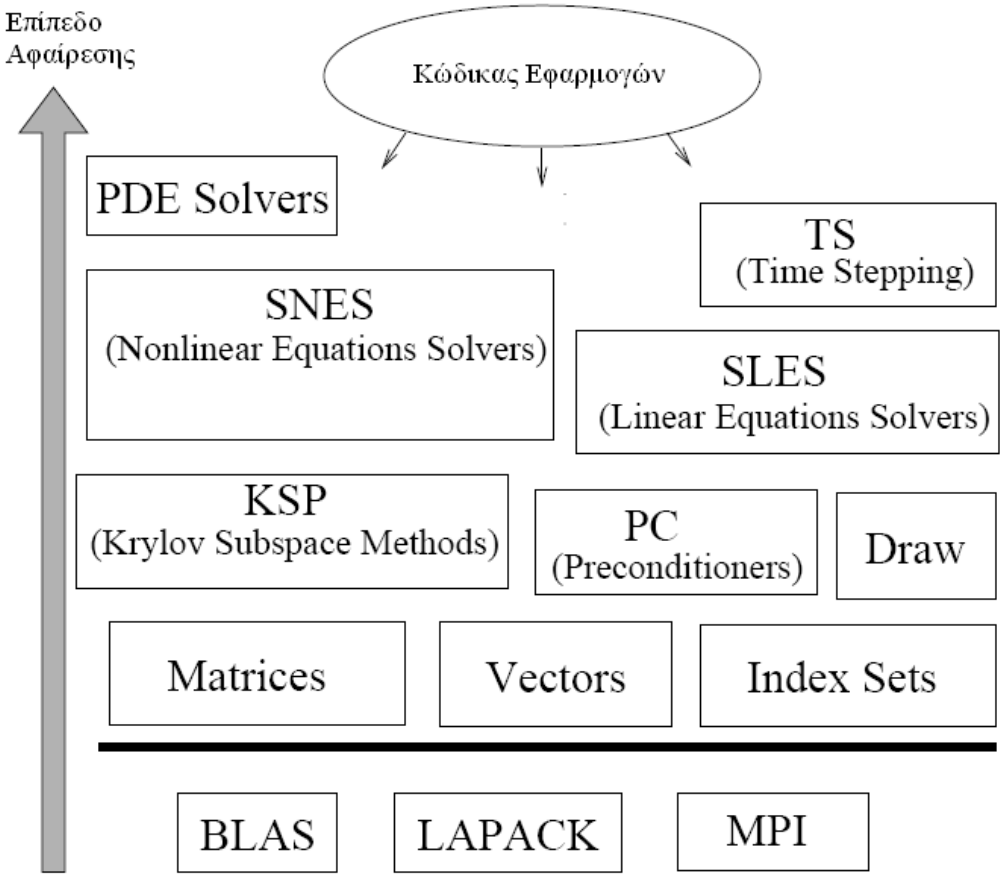
Η βιβλιοθήκη PETSC αναπτύχθηκε αρχικά ως μία πλατφόρμα για πειραματισμό πάνω σε μοντέλα, επιλυτές και αλγόριθμους. Το έργο ξεκίνησε το Σεπτέμβριο του 1991 και σήμερα, είναι ένα σύνολο ελεύθερα διαθέσιμου και ερευνητικά υποστηριζόμενου κώδικα που μπορεί να χρησιμοποιηθεί από τις γλώσσες C, C++, Fortran 77/90 και Python.

Μπορεί να εγκατασταθεί σε οποιοδήποτε παράλληλο σύστημα που υποστηρίζει MPI, όπως σε ισχυρά συνδεδεμένα συστήματα (Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise) ή σε χαλαρά συνδεδεμένα όπως δίκτυα και σταθμούς εργασίας (Compaq,HP, IBM, SGI, Sun, PCs με Linux ή Windows).

Η PETSc (Portable Extensive Toolkit for Scientific Computation) έχει αποδείξει με επιτυχία ότι η χρήση σύγχρονων προγραμματιστικών προτύπων μπορεί να διευκολύνει σημαντικά την ανάπτυξη εκτεταμένων επιστημονικών εφαρμογών σε Fortran , C και C++. Η ανάπτυξη του εργαλείου αυτού ξεκίνησε πριν αρκετά χρόνια και έχει σήμερα εξελιχθεί σε ένα ισχυρό σύνολο εργαλείων για την αριθμητική επίλυση μερικών διαφορικών εξισώσεων και συναφών προβλημάτων σε υπολογιστικά συστήματα υψηλής επίδοσης.

Η PETSc αποτελείται από μία ποικιλία βιβλιοθηκών (κάτι παρόμοιο με τις κλάσεις της C++) . Κάθε βιβλιοθήκη διαχειρίζεται μια συγκεκριμένη οικογένεια αντικειμένων (όπως διανύσματα, για παράδειγμα) και τις λειτουργίες που εφαρμόζονται στα αντικείμενα αυτά. Τα αντικείμενα και οι λειτουργίες που περιλαμβάνει η PETSc προέκυψαν μέσα από την μακρά εμπειρία στους επιστημονικούς υπολογισμούς. Μεταξύ των στοιχείων της PETSc περιλαμβάνονται και τα εξής:

- Σύνολα δεικτών, συμπεριλαμβανομένων μεταθέσεων για δεικτοδότηση διανυσμάτων, αναρίθμηση, κτλ.
- Διανύσματα.
- Πίνακες (κυρίως αραιοί).
- Μέθοδοι υποχώρων Krylov.
- Preconditioners.
- Μη γραμμικοί επιλυτές.

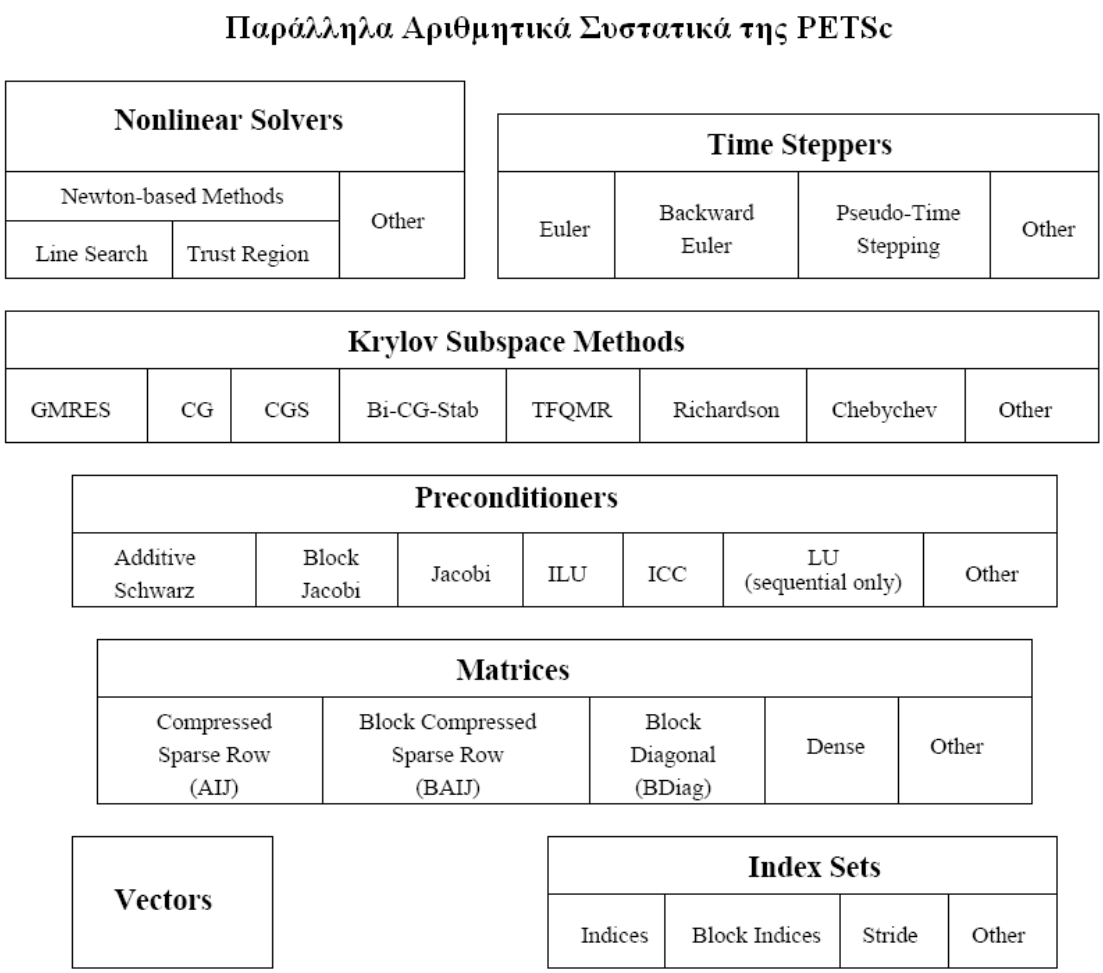


Σχήμα 3.4: Οργάνωση των βιβλιοθηκών της PETSc

Κάθε ένα από τα παραπάνω στοιχεία αποτελείται από μια αφηρημένη διεπαφή (ή απλούστερα ένα σύνολο ακολουθιακών κλήσεων) και από μία ή περισσότερες υλοποιήσεις χρησιμοποιώντας συγκεκριμένες δομές δεδομένων. Έτσι, η PETSc παρέχει ξεκάθαρο και αποτελεσματικό κώδικα για την επίλυση ΜΔΕ, με μία καθολική προσέγγιση για κάθε είδος προβλήματος. Ο σχεδιασμός αυτός διευκολύνει τη σύγκριση και τη χρήση διαφορετικών αλγόριθμων (για παράδειγμα, πειραματισμός με διάφορες μεθόδους Krylov, preconditioners, ή μεθόδους Newton). Έτσι, η PETSc παρέχει ένα πλούσιο περιβάλλον για τη μοντελοποίηση επιστημονικών εφαρμογών όπως επίσης και το γρήγορο σχεδιασμό και προτυποποίηση αλγόριθμων.

Οι βιβλιοθήκες διευκολύνουν την προσαρμογή και την επέκταση τόσο των αλγόριθμων όσο και των υλοποιήσεων. Η προσέγγιση αυτή προωθεί την επαναχρησιμοποίηση του κώδικα και την ευελιξία και διαχωρίζει τα θέματα του παραλληλισμού από την επιλογή των αλγόριθμων. Η υποδομή της PETSc δημιουργεί το θεμέλιο για την ανάπτυξη εφαρμογών μεγάλης κλίμακας.

Είναι χρήσιμο να λάβουμε υπόψη μας τις εσωτερικές σχέσεις ανάμεσα στα διαφορετικά τμήματα της PETSc. Στο Σχήμα 3.4 παρουσιάζεται ένα διάγραμμα των σημαντικότερων από αυτά τα τμήματα. Στο Σχήμα 3.5 μπορούμε να δούμε κάποια από τα αυτόνομα τμήματα με μεγαλύτερη λεπτομέρεια. Μέσα από τα σχήματα αυτά, γίνεται εμφανής η ιεραρχική οργάνωση της βιβλιοθήκης, η οποία δίνει τη δυνατότητα στους χρήστες να εφαρμόσουν το επίπεδο εκείνο της αφαίρεσης που είναι κατάλληλο για κάθε συγκεκριμένο πρόβλημα.



Σχήμα 3.5: Αριθμητικές Βιβλιοθήκες της PETSc.

3.4.1 Συγγραφή Προγραμμάτων με την PETSc

Τα περισσότερα προγράμματα PETSc ξεκινούν με μια κλήση στη συνάρτηση:

```
PetscInitialize(int *argc, char ***argv, char *file, char
*help);
```

η οποία αρχικοποιεί τόσο την PETSc όσο και το MPI. Τα ορίσματα argc και argv είναι τα ορίσματα γραμμής εντολών που δίνονται σε όλα τα προγράμματα σε C ή C++. Το όρισμα file καθορίζει προαιρετικά ένα εναλλακτικό όνομα για το αρχείο προτιμήσεων της PETSc (.petsrc), το οποίο βρίσκεται στον αρχικό φάκελο του χρήστη. Το τελευταίο όρισμα, help, είναι ένα προαιρετικό string που θα εκτυπωθεί αν το πρόγραμμα εκτελεστεί χρησιμοποιώντας την επιλογή –help.

Η κλήση στην PetscInitialize() αυτόματα καλεί την MPI_Init() αν το MPI δεν έχει προηγουμένως αρχικοποιηθεί. Κάτω από συγκεκριμένες συνθήκες όπου το MPI πρέπει να αρχικοποιηθεί ευθέως (ή αρχικοποιείται από κάποια άλλη βιβλιοθήκη), ο χρήστης μπορεί να καλέσει την MPI_Init() και έπειτα την PetscInitialize(). Η PetscInitialize() θέτει ως προεπιλογή στον καθολικό communicator, PETSC_COMM_WORLD, τον MPI_COMM_WORLD. Να σημειώσουμε εδώ, ότι οι χρήστες της PETSc δεν απαιτείται να χρησιμοποιούν συχνά απ' ευθείας κλήσεις

του MPI, ωστόσο είναι αναγκαίο να είναι εξοικειωμένοι με την έννοια της ανταλλαγής μηνυμάτων και του κατανεμημένου προγραμματισμού.

Όλες οι ρουτίνες της PETSc επιστρέφουν έναν ακέραιο ο οποίος δηλώνει αν προέκυψε κάποιο λάθος κατά την κλήση. Ο κωδικός λάθους τίθεται σε μη μηδενική τιμή αν εντοπιστεί κάποιο σφάλμα, αλλιώς παίρνει μηδενική τιμή. Στο τέλος κάθε PETSc προγράμματος πρέπει να υπάρχει μια κλήση στην PetscFinalize(). Η ρουτίνα αυτή διαχειρίζεται επιλογές τερματισμού του προγράμματος και καλεί την MPI_Finalize() αν το MPI αρχικοποιήθηκε από την PetscInitialize().

3.4.2 Διανύσματα και βασικές λειτουργίες διανυσμάτων

Το διάνυσμα (το οποίο συμβολίζεται με Vec) είναι ένα από τα απλούστερα αντικείμενα της PETSc. Τα διανύσματα χρησιμοποιούνται για την αποθήκευση λύσεων διακριτών μερικών διαφορικών εξισώσεων, δεξιών μελών γραμμικών συστημάτων, κτλ.

Δημιουργία και συναρμολόγηση διανυσμάτων

Η PETSc παρέχει δύο βασικούς τύπους διανυσμάτων: σειριακά και παράλληλα (βασισμένα στο MPI). Για να δημιουργήσουμε ένα σειριακό διάνυσμα με *m* στοιχεία μπορούμε να χρησιμοποιήσουμε τη ρουτίνα

```
VecCreateSeq(PETSC COMM SELF, int m, Vec *x);
```

Αν θέλουμε να δημιουργήσουμε ένα παράλληλο διάνυσμα μπορούμε είτε να καθορίσουμε τον αριθμό των στοιχείων του που θα αποθηκευτούν σε κάθε διεργασία, είτε να αφήσουμε την PETSc να το αποφασίσει. Η εντολή

```
VecCreateMPI(MPI Comm comm, int m, int M, Vec *x);
```

δημιουργεί ένα διάνυσμα το οποίο κατανέμεται σε όλες τις διεργασίες του communicator, *comm*, όπου το *m* προσδιορίζει τον αριθμό των στοιχείων που θα αποθηκευτούν στην τοπική διεργασία, και *M* είναι ο συνολικός αριθμός των στοιχείων του διανύσματος. Είτε η τοπική είτε η καθολική διάσταση, αλλά όχι και οι δύο, μπορεί να τεθεί σε PETSC_DECIDE
ώστε να δηλωθεί ότι θα καθοριστεί από την PETSc. Πιο γενικά, μπορεί κανείς να χρησιμοποιήσει τις ρουτίνες

```
VecCreate(MPI Comm comm,Vec *v);
VecSetSizes(Vec v, int m, int M);
VecSetFromOptions(Vec v);
```

οι οποίες αυτόματα παράγουν τον κατάλληλο τύπο διανύσματος (σειριακό ή παράλληλο) πάνω σε όλες τις διεργασίες του *comm.* Η επιλογή -vec_type mpi μπορεί να χρησιμοποιηθεί σε συνδυασμό με τις VecCreate() και VecSetFromOptions() ώστε να εξειδικεύσει τη χρήση των MPI διανυσμάτων, ακόμα στην περίπτωση ενός επεξεργαστή.

Πρέπει να σημειώσουμε ότι όλες οι διεργασίες στον *comm.*, πρέπει να καλέσουν τις ρουτίνες δημιουργίας του διανύσματος, εφ' όσον οι ρουτίνες αυτές είναι συλλογικές επάνω σε όλες τις διεργασίες του communicator.
Για να αναθέσουμε μια τιμή σε όλα τα στοιχεία ενός διανύσματος μπορούμε να χρησιμοποιήσουμε την εντολή

```
VecSet(Vec x,PetscScalar value);
```

Η ανάθεση διαφορετικών τιμών στα στοιχεία ενός διανύσματος είναι πιο περίπλοκη, ώστε να καταστεί δυνατή η συγγραφή αποδοτικού παράλληλου κώδικα. Σε δύο βήματα, μπορεί κανείς να χρησιμοποιήσει αρχικά τη ρουτίνα

```
VecSetValues(Vec x, int n, int *indices, PetscScalar
*values, INSERT VALUES);
```

όσες φορές απαιτείται, σε κάποιες ή όλες τις διεργασίες. Το όρισμα *n* δίνει τον αριθμό των στοιχείων τα οποία τίθενται κατά την εισαγωγή. Ο πίνακας ακεραίων *indices* περιέχει τους καθολικούς δείκτες των στοιχείων, ενώ ο πίνακας *values* περιέχει τις τιμές προς εισαγωγή. Οποιαδήποτε διεργασία μπορεί να δώσει τιμή σε οποιοδήποτε στοιχείο του διανύσματος. Η PETSc, έπειτα, φροντίζει ώστε η τιμή να αποθηκευτεί στη σωστή τοποθεσία. Όταν όλες οι τιμές έχουν εισαχθεί με την VecSetValues(), πρέπει να κληθούν οι ρουτίνες

```
VecAssemblyBegin(Vec x);
VecAssemblyEnd(Vec x);
```

ώστε να εκτελεστούν οι απαραίτητες ενέργειες περάσματος μηνυμάτων για τα μη τοπικά στοιχεία. Για να πετύχουμε επικάλυψη επικοινωνίας και υπολογισμών, ο κώδικας του χρήστη μπορεί να περιέχει ενδιάμεσα στις δύο αυτές κλήσεις, άλλες εντολές οι οποίες θα εκτελεστούν όσο μεταδίδονται τα μηνύματα.
Όταν ένα διάνυσμα δεν μας είναι πλέον χρήσιμο, πρέπει να καταστραφεί με την εντολή

```
VecDestroy(Vec x);
```

Βασικές λειτουργίες διανυσμάτων

Οι βασικότερες λειτουργίες διανυσμάτων που υποστηρίζονται από την PETSc παρουσιάζονται στον Πίνακα 3.5.
Για τον καθορισμό των τοπικών ορίων ενός κατανεμημένου διανύσματος σε διεργασίες υπάρχει η ρουτίνα `VecGetOwnershipRange(Vec vec,int *low,int *high)`.

Το όρισμα *low* υποδεικνύει το πρώτο στοιχείο που ανήκει στη διεργασία, ενώ το όρισμα *high* δηλώνει το επόμενο από το τελευταίο στοιχείο της διεργασίας.
Περιστασιακά, είναι πιθανό να θελήσει ο χρήστης να προσπελάσει τα πραγματικά στοιχεία του διανύσματος. Για το σκοπό αυτό, η ρουτίνα VecGetArray(Vec v, PetscScalar **array) επιστρέφει ένα δείκτη στα τοπικά στοιχεία της διεργασίας. Όταν δεν απαιτείται πλέον πρόσβαση στον πίνακα, πρέπει να κληθεί η `VecRestoreArray(Vec v, PetscScalar **array)`.

Ο αριθμός των στοιχείων που αποθηκεύονται τοπικά σε μία διεργασία και ο συνολικός αριθμός των στοιχείων του διανύσματος μπορούν να βρεθούν καλώντας τις ρουτίνες

```
VecGetLocalSize(Vec v,int *size);
```

και

```
VecGetSize(Vec v,int *size);
```

αντίστοιχα.

Πίνακας 3.5: Βασικές λειτουργίες διανυσμάτων στην PETSc.

Όνομα Συνάρτησης	Λειτουργία
VecAXPY(Vec y,PetscScalar a,Vec x);	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x);	$y = x + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y);	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a);	$x = a * x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}' * y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x' * y$
VecNorm(Vec x,NormType type, double *r);	$r = \|\|x\|\|type$
VecSum(Vec x, PetscScalar *r);	$r = \sum x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x \; while \; x = y$
VecPointwiseMult(Vec w,Vec x,Vec y);	$wi = xi * yi$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$wi = xi / yi$
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = \bar{x}' * y[i]$
VecMTDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = x' * y[i]$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i * x[i]$
VecMax(Vec x, int *idx, double *r);	$r = max \; x_i$
VecMin(Vec x, int *idx, double *r);	$r = min \; x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x,PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = a$

3.4.3 Πίνακες και βασικές λειτουργίες πινάκων

Η PETSc παρέχει μια μεγάλη ποικιλία υλοποιήσεων για πίνακες αφού δεν υπάρχει κάποια μορφή πίνακα η οποία να είναι κατάλληλη για όλα τα είδη των προβλημάτων. Στην τρέχουσα έκδοση, υποστηρίζεται πυκνή μορφή και διάφορες αραιές μορφές, τόσο σειριακές όσο και κατανεμημένες.

Η χρήση πινάκων της PETSc περιλαμβάνει τις εξής ενέργειες: δημιουργία ενός συγκεκριμένου τύπου πίνακα, εισαγωγή τιμών σε αυτόν, επεξεργασία του πίνακα, χρήση του πίνακα σε διάφορους υπολογισμούς και τέλος, αποδέσμευση της μνήμης που καταλαμβάνει. Ο κώδικας μια εφαρμογής δεν απαιτείται να γνωρίζει τίποτα για το συγκεκριμένο τύπο του πίνακα ή τη μορφή του.

Δημιουργία και συναρμολόγηση πινάκων

Οι απλούστερες ρουτίνες που παρέχει η PETSc για τη δημιουργία ενός πίνακα A είναι οι

```
MatCreate(MPI Comm comm,Mat *A)

MatSetSizes(Mat A,int m,int n,int M,int N)
```

Οι ρουτίνες αυτές παράγουν ένα σειριακό πίνακα όταν το πρόγραμμα εκτελείται με μία διεργασία ενώ δημιουργούν παράλληλο πίνακα όταν δύο ή περισσότερες διεργασίες είναι διαθέσιμες. Ο συγκεκριμένος τύπος για τον πίνακα

μπορεί να ορισθεί από το χρήστη χρησιμοποιώντας την κατάλληλη επιλογή κατά το χρόνο εκτέλεσης. Ο χρήστης προσδιορίζει είτε τις καθολικές διαστάσεις του πίνακα, οι οποίες δίνονται από τα ορίσματα M και N, είτε τις τοπικές διαστάσεις, οι οποίες δίνονται από τα m και n, ενώ η PETSc αναλαμβάνει εξ' ολοκλήρου τη δέσμευση της απαιτούμενης μνήμης. Οι ρουτίνες αυτές διευκολύνουν την εναλλαγή ανάμεσα σε διάφορους τύπους πινάκων, όταν, για παράδειγμα, είναι πιο αποδοτική η χρήση ενός τύπου για μια συγκεκριμένη εφαρμογή. Ο προκαθορισμένος τύπος που δημιουργείται με την MatCreate() είναι ο AIJ.

Για την εισαγωγή ή την πρόσθεση τιμών σε έναν πίνακα, διατίθεται η ρουτίνα

```
MatSetValues(Mat A,int m,const int idxm[],int n,const int
idxn[],const PetscScalar values[],InsertMode addv);
```

όπου το όρισμα *addv* μπορεί να είναι είτε INSERT_VALUES είτε ADD_VALUES ανάλογα με τη λειτουργία. Οι πίνακες ακεραίων *idxm* και *idxn* δίνουν τους καθολικούς δείκτες των γραμμών και στηλών των στοιχείων προς εισαγωγή αντίστοιχα. Η MatSetValues() θεωρεί ότι το πρώτο στοιχείο του πίνακα βρίσκεται στο (0, 0) ανεξάτητα του τύπου πίνακα που θα παραχθεί. Το όρισμα *values* είναι ένας πίνακας δύο διαστάσεων ο οποίος περιέχει τις τιμές που θα εισαχθούν.

Αφού εισαχθούν ή προστεθούν τα στοιχεία σε έναν πίνακα, πρέπει να επεξεργαστούν πριν χρησιμοποιηθεί ο πίνακας σε οποιονδήποτε υπολογισμό. Οι ρουτίνες που αναλαμβάνουν την επεξεργασία αυτή είναι οι

```
MatAssemblyBegin(Mat A,MAT FINAL ASSEMBLY);
MatAssemblyEnd(Mat A,MAT FINAL ASSEMBLY);
```

Προσθέτοντας κώδικα ανάμεσα σε αυτές τις δύο ρουτίνες, ο χρήστης μπορεί να επιτελέσει διάφορους υπολογισμούς όσο αποστέλλονται μηνύματα για την τελική διαμόρφωση του πίνακα. Κλήσεις στην MatSetValues() με επιλογή εισαγωγής και πρόσθεσης δεν μπορούν να αναμειγνύονται χωρίς προηγούμενη κλήση των ρουτινών τελικής διαμόρφωσης.

Αν και επιτρέπεται να εισάγουμε τιμές σε έναν πίνακα, χωρίς να ορίσουμε σε ποια διεργασία θα ανήκουν, είναι καλό, για λόγους επίδοσης, να παράγονται οι περισσότερες τιμές από τη διεργασία στην οποία τελικά θα αποθηκευτούν. Προς διευκόλυνση του προγραμματιστή προσφέρεται η ρουτίνα

```
MatGetOwnershipRange(Mat A,int *first row,int *last row);
```

η οποία πληροφορεί το χρήστη ότι όλες οι γραμμές μεταξύ των first_row και last_row-1 θα αποθηκευτούν στην τοπική διεργασία. Στις υλοποιήσεις αραιών πινάκων, μόλις κληθούν οι συναρτήσεις τελικής διαμόρφωσης, ο πίνακας είναι συμπτυγμένος και μπορεί να χρησιμοποιηθεί σε υπολογισμούς, όπως πολλαπλασιασμό πινάκων κτλ. Η εισαγωγή στοιχείων σε αυτό το σημείο είναι ιδιαίτερα «ακριβή», καθώς, κατά πάσα πιθανότητα, θα χρειαστεί αποθήκευση ενός αντίγραφου και εκ νέου δέσμευση μνήμης. Έτσι, όποτε είναι δυνατό, πρέπει όλες οι εισαγωγές τιμών να γίνονται πριν την κλήση των συναρτήσεων τελικής διαμόρφωσης.

Αραιοί πίνακες

Η προκαθορισμένη υλοποίηση για έναν αραιό πίνακα γενικού σκοπού στην PETSc είναι η AIJ (η οποία καλείται επίσης αραιή μορφή του Yale ή Compressed Sparse Row, CSR).

Σειριακοί AIJ Αραιοί Πίνακες

Στους AIJ PETSc πίνακες, τα μη μηδενικά στοιχεία αποθηκεύονται κατά γραμμές, συνοδευόμενα από έναν πίνακα με τους δείκτες των στηλών των στοιχείων και έναν πίνακα με στοιχεία που δείχνουν την αρχή κάθε γραμμής. Ας σημειωθεί ότι τα διαγώνια στοιχεία δεν αποθηκεύονται ξεχωριστά από τα υπόλοιπα μη μηδενικά στοιχεία.

Για να δημιουργήσουμε έναν σειριακό αραιό πίνακα τύπου AIJ, A, με m γραμμές και n στήλες, μπορούμε να χρησιμοποιήσουμε την εντολή

```
MatCreateSeqAIJ(PETSC COMM SELF,int m,int n,int
nz,int*nnz,Mat *A);
```

όπου τα ορίσματα *nz* ή *nnz* μπορούν να χρησιμοποιηθούν για πιο αποδοτική δέσμευση μνήμης. Ο χρήστης έχει την επιλογή να θέσει nz=0 και nnz=PETSC_NULL αν επιθυμεί η δέσμευση της μνήμης να ελεγχθεί αποκλειστικά από την PETSc.

Παράλληλοι AIJ Αραιοί Πίνακες

Παράλληλοι αραιοί πίνακες τύπου AIJ μπορούν να δημιουργηθούν με την εντολή

```
MatCreateMPIAIJ(MPI Comm comm,int m,int n,int M,int N,int
d nz,int *d nnz, int o nz,int *o nnz,Mat *A);
```

όπου A είναι ο πίνακας που δημιουργείται, ενώ τα ορίσματα m, M και N καθορίζουν τον αριθμό των τοπικών γραμμών και τους αριθμούς των καθολικών γραμμών και στηλών, αντίστοιχα. Είτε οι τοπικές είτε οι καθολικές παράμετροι μπορούν να έχουν την τιμή PETSC_DECIDE, ώστε να αναλάβει η PETSc τον καθορισμό τους. Ο αριθμός των γραμμών του πίνακα που αποθηκεύονται σε κάθε διεργασία δίνεται από το όρισμα m, είναι προκαθορισμένος και δεν μπορεί να αλλάξει, ενώ καθορίζεται από την PETSc αν έχει την τιμή PETSC_DECIDE.

Αν η τιμή PETSC_DECIDE δεν χρησιμοποιηθεί για τα ορίσματα m και n, θα πρέπει ο χρήστης να διασφαλίσει ότι τα ορίσματα αυτά θα συμφωνούν με τα διανύσματα. Για να το επιτύχουμε αυτό, θεωρούμε πρώτα το γινόμενο y=Ax. Η μεταβλητή m της ρουτίνας δημιουργίας MatCreateMPIAIJ() θα πρέπει τότε να συμφωνεί με το τοπικό όρισμα του μεγέθους στη ρουτίνα δημιουργίας του διανύσματος y, VecCreateMPI(). Ομοίως, το όρισμα n θα πρέπει να συμφωνεί με το μέγεθος του διανύσματος στη ρουτίνα δημιουργίας του διανύσματος x.

Εκ των προτέρων δέσμευση μνήμης για παράλληλους AIJ αραιούς πίνακες

Όπως παρατηρήθηκε και προηγουμένως, η εκ των προτέρων δέσμευση μνήμης είναι κρίσιμη για την καλύτερη επίδοση κατά τη διάρκεια της τελικής διαμόρφωσης του πίνακα. Παρουσιάζουμε εδώ ένα παράδειγμα για να δείξουμε πώς μπορούμε να το επιτύχουμε αυτό για τον τύπο MATMPIAIJ. Ας θεωρήσουμε τρεις διεργασίες και έναν πίνακα 8x8, οποίος διαχωρίζεται αναθέτοντας τρεις γραμμές στην πρώτη διεργασία, τις τρεις επόμενες στη δεύτερη και τις δύο τελευταίες στην τρίτη:

$$\left(\begin{array}{ccc|ccc|cc} 1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\ 0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\ 9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\ \hline 13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\ 0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\ 0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\ \hline 25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\ 30 & 0 & 0 & 31 & 32 & 33 & 0 & 34 \end{array}\right)$$

Ο διαγώνιος υποπίνακας, d, που ανατίθεται στην πρώτη διεργασία θα είναι ο

$$\left(\begin{array}{ccc} 1 & 2 & 0 \\ 0 & 5 & 6 \\ 9 & 0 & 10 \end{array}\right)$$

, ενώ ο μη-διαγώνιος, o, δίνεται ως εξής:

$$\left(\begin{array}{ccccc} 0 & 3 & 0 & 0 & 4 \\ 7 & 0 & 0 & 8 & 0 \\ 11 & 0 & 0 & 12 & 0 \end{array}\right)$$

.

Για την πρώτη διεργασία, λοιπόν, μπορούμε να θέσουμε το όρισμα d_nz=2 (αφού κάθε γραμμή του διαγώνιου υποπίνακα έχει δύο μη μηδενικά στοιχεία), ή εναλλακτικά, μπορούμε να θέσουμε d_nnz={2, 2, 2}. Το όρισμα o_nz μπορεί να τεθεί ίσο με 2, αντίστοιχα, αφού κάθε γραμμή του υποπίνακα o έχει δύο μη μηδενικά στοιχεία, ή ισοδύναμα μπορούμε να θέσουμε o_nnz={2, 2, 2}. Με την ίδια λογική, οι πίνακες d και o για τη δεύτερη διεργασία θα δίνονται ως

$$\left(\begin{array}{ccc} 15 & 16 & 17 \\ 19 & 20 & 21 \\ 22 & 23 & 0 \end{array}\right) \quad \text{και} \quad \left(\begin{array}{ccccc} 13 & 0 & 14 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 24 & 0 \end{array}\right)$$

με d_nz=3 ή d_nnz={3, 3, 2} και o_nz=2 ή o_nnz={2, 1, 1}.

Ας σημειώσουμε εδώ ότι ο χρήστης δεν εργάζεται ποτέ απ' ευθείας με τους πίνακες d και o, παρά μόνο κατά τη διαδικασία της δέσμευσης μνήμης. Επίσης, δεν είναι απαραίτητος ο ακριβής καθορισμός του απαιτούμενου χώρου, αλλά είναι αποδεκτό όσον αφορά στην επίδοση, να παρέχεται μία καλή εκτίμηση του χώρου αυτού.

Βασικές λειτουργίες Πινάκων

Στον Πίνακα 3.6 παρουσιάζονται οι βασικότερες λειτουργίες πινάκων που διατίθενται από την PETSc.

Ένας παράλληλος πίνακας μπορεί να πολλαπλασιασθεί με ένα διάνυσμα μεγέθους n, δίνοντας ως γινόμενο ένα διάνυσμα μεγέθους m με την παρακάτω ρουτίνα:

```
MatMult(Mat A,Vec x,Vec y);
```

ενώ τα διανύσματα x, y θα πρέπει να έχουν δημιουργηθεί με τις εξής κλήσεις:

```
VecCreateMPI(MPI Comm comm,n,N,&x);
VecCreateMPI(MPI Comm comm,m,M,&y);
```

Για να πολλαπλασιάσουμε με τον ανάστροφο ενός πίνακα υπάρχει η ρουτίνα

```
MatMultTranspose(Mat A,Vec x,Vec y);
```

Υπάρχουν, επίσης, εκδοχές του πολλαπλασιασμού όπου το αποτέλεσμα προστίθεται σε ένα άλλο διάνυσμα:

```
MatMultAdd(Mat A,Vec x,Vec y,Vec w);
MatMultTransposeAdd(Mat A,Vec x,Vec y,Vec w);
```

Οι ρουτίνες αυτές παράγουν τα αποτελέσματα $w=A*x+y$ και $w=A^T*x+y$ αντίστοιχα. Για την εκτύπωση ενός πίνακα στην οθόνη παρέχεται η ρουτίνα

```
MatView(Mat mat,PETSC VIEWER DRAW WORLD).
```

Πίνακας 3.6: Βασικές λειτουργίες πινάκων στην PETSc.

Όνομα Συνάρτησης	Λειτουργία
MatMult(Mat A,Vec x, Vec y);	$y = A*x$
MatMultAdd(Mat A,Vec x, Vec y,Vec z	$z = y + A*x$
MatMultTranspose(Mat A,Vec x, Vec y);	$y = A^T*x$
MatMultTransposeAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A^T*x$
MatNorm(Mat A,NormType type, double *r);	$r = \|\|A\|\|type$
MatDiagonalScale(Mat A,Vec l,Vec r);	$A = diag(l) * A * diag(r)$
MatScale(Mat A,PetscScalar a);	$A = a * A$
MatConvert(Mat A,MatType type,Mat *B);	$B = A$
MatCopy(Mat A,Mat B,MatStructure);	$B = A$
MatGetDiagonal(Mat A,Vec x);	$x = diag(A)$
MatTranspose(Mat A,MatReuse,Mat* B);	$B = A^T$
MatZeroEntries(Mat A);	$A = 0$
MatShift(Mat Y,PetscScalar a);	$Y = Y + a * I$

3.4.4 KSP: Επίλυση γραμμικών συστημάτων με την PETSc

Το αντικείμενο KSP αποτελεί την καρδιά της PETSc, καθώς παρέχει μια ομοιόμορφη και αποδοτική πρόσβαση σε όλους τους γραμμικούς επιλυτές του πακέτου,

σειριακούς και παράλληλους, ευθείς και επαναληπτικούς. Το KSP προορίζεται για την επίλυση συστημάτων της μορφής

$$Ax=b,$$

όπου A ο πίνακας του γραμμικού συστήματος, b το διάνυσμα του δεξιού μέλους της εξίσωσης και x το διάνυσμα της λύσης. Το KSP χρησιμοποιεί την ίδια ακολουθία κλήσεων τόσο για την άμεση όσο και για την επαναληπτική διαδικασία επίλυσης ενός συστήματος. Σε ορισμένες τεχνικές επίλυσης, είναι ακόμη εφικτό να οριστούν κάποιες επιλογές κατά το χρόνο εκτέλεσης.

Ο συνδυασμός μιας μεθόδου υποχώρου Krylov και ενός preconditioner βρίσκεται στο κέντρο των πιο σύγχρονων αριθμητικών τεχνικών για την επαναληπτική επίλυση γραμμικών συστημάτων.

Χρήση της διεπαφής KSP

Για την επίλυση ενός γραμμικού συστήματος με KSP, πρέπει αρχικά να δημιουργήσουμε το υπόβαθρο του επιλυτή με την εντολή:

```
KSPCreate(MPI Comm comm,KSP *ksp);
```

Εδώ, comm είναι ο MPI communicator και ksp το υπόβαθρο του επιλυτή που δημιουργείται. Πριν την επίλυση του γραμμικού συστήματος, ο χρήστης θα πρέπει να καλέσει την ακόλουθη ρουτίνα ώστε να ορίσει τους πίνακες που σχετίζονται με το γραμμικό σύστημα:

```
KSPSetOperators(KSP ksp,Mat Amat,Mat Pmat,MatStructure
flag);
```

Το όρισμα *Amat* αποτελεί τον πίνακα που ορίζει το σύστημα και το όρισμα Pmat αποτελεί τον πίνακα από τον οποίο θα δημιουργηθεί ο preconditioner. Τυπικά, ο πίνακας αυτός είναι ο ίδιος με τον *Amat*. Μπορεί, ωστόσο, να διαφέρει, όταν για παράδειγμα ο πίνακας αυτός παρέχεται από κάποια μέθοδο χαμηλότερου επιπέδου που χρησιμοποιήθηκε για τη μορφοποίηση του πίνακα του γραμμικού συστήματος. Το όρισμα *flag* μπορεί να χρησιμοποιηθεί για να αποφευχθούν περιττοί υπολογισμοί όταν λύνονται επαναληπτικά συστήματα του ίδιου μεγέθους με την ίδια preconditioning μέθοδο. Κατά την επίλυση ενός μόνο συστήματος, το όρισμα αυτό αγνοείται. Ο χρήστης μπορεί να θέσει το όρισμα αυτό ως εξής:

- SAME_NONZERO_PATTERN: ο preconditioner διατηρεί τη μορφή των μη μηδενικών του στοιχείων ανάμεσα σε διαδοχικές γραμμικές επιλύσεις.
- DIFFERENT_NONZERO_PATTERN: ο preconditioner πίνακας δεν διατηρεί τη μορφή των μη μηδενικών του στοιχείων ανάμεσα σε διαδοχικές γραμμικές επιλύσεις.
- SAME_PRECONDITIONER: Ο preconditioner πίνακας είναι ακριβώς ο ίδιος με αυτόν που χρησιμοποιήθηκε στην αμέσως προηγούμενη επίλυση.

Αν η μορφή του πίνακα δεν είναι γνωστή εκ των προτέρων, θα πρέπει να τεθεί η τιμή DIFFERENT_NONZERO_PATTERN.

Οι περισσότερες από τις λειτουργίες του KSP μπορούν να οριστούν από την ρουτίνα

```
KSPSetFromOptions(KSP ksp);
```

Η ρουτίνα αυτή δέχεται τις επιλογές –h και –help όπως και όλες τις επιλογές των αντικειμένων KSP και PC που παρουσιάζονται στη συνέχεια. Για την επίλυση ενός γραμμικού συστήματος, αφού δημιουργηθούν τα διανύσματα του συστήματος, πρέπει να εκτελεστεί η εντολή:

```
KSPSolve(KSP ksp,Vec b,Vec x);
```

Όταν η επίλυση είναι επιτυχής, μπορούμε να ανακτήσουμε τον αριθμό των επαναλήψεων που χρειάστηκαν για να επιτευχθεί σύγκλιση καλώντας τη ρουτίνα:

```
KSPGetIterationNumber(KSP ksp, int *its);
```

Πρέπει να σημειώσουμε ότι το ίδιο υπόβαθρο KSP μπορεί να χρησιμοποιηθεί για πολλαπλές επιλύσεις συστημάτων. Όταν δεν χρειαζόμαστε πλέον το αντικείμενο KSP πρέπει να το καταστρέψουμε με την εντολή:

```
KSPDestroy(KSP ksp);
```

Η παραπάνω διαδικασία είναι επαρκής για μια γενική χρήση του πακέτου KSP. Ένα επιπλέον βήμα απαιτείται από τους χρήστες που επιθυμούν να ορίσουν κάποιον συγκεκριμένο preconditioner ή να καταγράψουν δεδομένα επίδοσης. Στη περίπτωση αυτή, ο χρήστης μπορεί να καλέσει προαιρετικά τη ρουτίνα:

```
KSPSetUp(KSP ksp)
```

πριν την κλήση της KSPSolve() ώστε να καθορίσει τις επιλογές του. Η ρητή κλήση της ρουτίνας αυτής επιτρέπει την παρακολούθηση και καταγραφή των υπολογισμών που εκτελούνται κατά τη φάση της προετοιμασίας, όπως π.χ. της παραγοντοποίησης για τον ILU(Incomplete LU) preconditioner.

Ο προκαθορισμένος επιλυτής του KSP είναι ο restarted GMRES (Generalized Minimal Residual), και χρησιμοποιεί ILU preconditioner για μία διεργασία, ενώ στην περίπτωση πολλών διεργασιών χρησιμοποιείται η μέθοδος Jacobi με ένα block ανά διεργασία, καθένα από τα οποία λύνεται με ILU. Παρέχεται, ωστόσο, ποικιλία άλλων επιλυτών και επιλογών επίλυσης. Για τον ορισμό ενός άλλου preconditioner, εκτός του προκαθορισμένου από τον χρήστη, παρέχεται η ρουτίνα

```
KSPGetPC(KSP ksp,PC *pc);
```

με την οποία μπορούμε να έχουμε πρόσβαση στα αντικείμενα KSP και PC, ώστε να αλλάξουμε τις προκαθορισμένες τιμές και επιλογές της επίλυσης.

3.4.4.1 Μέθοδοι Krylov

Όταν απαιτείται η επίλυση πολλαπλών γραμμικών συστημάτων, ίδιου μεγέθους με την ίδια μέθοδο, η PETSc προσφέρει πολλές επιλογές. Για την επίλυση διαδοχικών γραμμικών συστημάτων τα οποία έχουν τον ίδιο preconditioner (δηλαδή, έχουν ουσιαστικά τη ίδια δομή με τα ίδια στοιχεία) αλλά διαφορετικά διανύσματα δεξιού μέλους, ο χρήστης μπορεί απλώς να καλέσει τη ρουτίνα KSPSolve() πολλές φορές. Οι λειτουργίες αρχικοποίησης του preconditioner θα εκτελεστούν μόνο κατά την πρώτη κλήση της KSPSolve() και θα επαναληφθούν κατά τις διαδοχικές κλήσεις σε αυτή. Για την επίλυση διαδοχικών συστημάτων με διαφορετικούς preconditioners

(όπου δηλαδή η δομή του πίνακα μεταβάλλεται ανάμεσα στις επαναλήψεις) ο χρήστης θα πρέπει να χρησιμοποιήσει τις συναρτήσεις KSPSetOperators() και KSPSolve() για κάθε επίλυση. Ωστόσο, και πάλι υπάρχουν ορίσματα, τα οποία λάβουν τη σωστή τιμή, θα αποφευχθούν αρκετοί υπολογισμοί από τη μία επίλυση στην άλλη.

Οι μέθοδοι Krylov δέχονται μια πληθώρα επιλογών, μερικές από τις οποίες παρουσιάζονται στη συνέχεια. Αρχικά, για να ορίσουμε ποια Krylov μέθοδος θα χρησιμοποιηθεί, πρέπει να καλέσουμε τη ρουτίνα

```
KSPSetType(KSP ksp,KSPType method)
```

Το όρισμα type μπορεί να πάρει μία από τις τιμές που φαίνονται στον Πίνακα 3.7. Ο τύπος μπορεί, επίσης, να καθοριστεί κατά το χρόνο εκτέλεσης, χρησιμοποιώντας την επιλογή –ksp_type.

Πίνακας 3.7: Krylov μέθοδοι της PETSc

Μέθοδος	**KSPType**
Richardson	KSPRICHARDSON
Chebychev	KSPCHEBYCHEV
Conjugate Gradient	KSPCG
BiConjugate Gradient	KSPBICG
Generalized Minimal Residual	KSPGMRES
BiCGSTAB	KSPBCGS
Conjugate Gradient Squared	KSPCGS
Transpose-Free Quasi-Minimal Residual	KSPTFQMR
Transpose-Free Quasi-Minimal Residual	KSPTCQMR
Conjugate Residual	KSPCR
Least Squares Method	KSPLSQR
Shell for no KSP method	KSPPREONLY

3.4.4.2 Preconditioners

Όπως αναφέραμε στην προηγούμενη ενότητα, οι μέθοδοι Krylov χρησιμοποιούνται συνήθως σε συνδυασμό με κάποιον preconditioner. Για να υλοποιήσουμε μια συγκεκριμένη τεχνική preconditioning μπορούμε να καλέσουμε τη ρουτίνα

```
PCSetType(PC pc,PCType method).
```

Στον Πίνακα 3.8 παρουσιάζονται οι βασικές μέθοδοι preconditioning που παρέχονται από την PETSc. Ο preconditioner PSCSHELL χρησιμοποιεί έναν συγκεκριμένο preconditioner που παρέχεται από την εφαρμογή. Ο άμεσος preconditioner, PCLU, είναι ουσιαστικά ένας άμεσος επιλυτής για γραμμικά συστήματα ο οποίος χρησιμοποιεί παραγοντοποίηση LU.

Κάθε preconditioner μπορεί να συνδεθεί με ένα σύνολο από επιλογές, οι οποίες μπορούν να οριστούν μέσα από ρουτίνες ή εντολές γραμμής εργασιών κατά την εκτέλεση της εφαρμογής. Όλες οι ρουτίνες αυτού του τύπου έχουν ονόματα της μορφής PC<TYPE>Option, ενώ οι εντολές είναι της μορφής –pc_<type>_option [value].

Πίνακας 3.8: Preconditioners της PETSc

Μέθοδος	PCType	Όνομα Επιλογής Εντολής
Jacobi	PCJACOBI	jacobi
Block Jacobi	PCBJACOBI	bjacobi
SOR	PCSOR	sor
Incomplete Cholesky	PCICC	icc
Incomplete LU	PCILU	ilu
Additive Swartz	PCASM	asm
Linear Solver	PCKSP	ksp
LU	PCLU	lu
Cholesky	PCCholesky	cholesky
Χωρίς Preconditioner	PCNONE	none
Preconditioner ορισμένος από το χρήστη	PCSHELL	shell

Κεφάλαιο 4ο: Εφαρμογές

4.1 Εισαγωγή

Το προγραμματιστικό κομμάτι της παρούσας διπλωματικής εργασίας συμπεριλαμβάνει την σειριακή υλοποίηση και στη συνέχεια, την παραλληλοποίηση δύο αλγόριθμων, χρησιμοποιώντας τα προγραμματιστικά εργαλεία που αναλύθηκαν στο προηγούμενο κεφάλαιο.

Η πρώτη υλοποίηση, προέρχεται από τον τομέα των Συστημάτων Ηλεκτρικής Ενέργειας και αφορά στον αλγόριθμο WLS (Weighted Least Squares) για την εκτίμηση της κατάστασης ενός Σ.Η.Ε. Είναι μια ρεαλιστική εφαρμογή των τεχνολογικών θεμάτων που εξετάζονται στην παρούσα εργασία. Ο αλγόριθμος αυτός αποτελεί ένα τυπικό παράδειγμα επιστημονικών υπολογισμών, ο οποίος περιέχει πράξεις γραμμικής άλγεβρας με πίνακες και διανύσματα, όπως και επίλυση γραμμικών συστημάτων. Επιπλέον, προσφέρεται για παραλληλοποίηση και υλοποίηση σε μια παράλληλη αρχιτεκτονική, έτσι ώστε να επιβεβαιώσουμε πειραματικά τα όσα συζητήθηκαν στα προηγούμενα κεφάλαια σε θεωρητικό επίπεδο. Η υλοποίηση έγινε με τη χρήση της βιβλιοθήκης PETSc και του MPI.

Σε συνέχεια της υλοποίησης μιας επιστημονικής εφαρμογής χρησιμοποιώντας έτοιμες ρουτίνες γραμμικής άλγεβρας, προχωρήσαμε σε μία εξαρχής υλοποίηση μιας μεθόδου επίλυσης γραμμικών συστημάτων. Η μέθοδος αυτή αφορά στον αλγόριθμο Conjugate Gradient, ο οποίος υλοποιήθηκε σε δύο σειριακές εκδόσεις (με τη χρήση της βιβλιοθήκης BLAS και χωρίς) και στη συνέχεια παραλληλοποιήθηκε με MPI.
Αρχικά, παρουσιάζεται το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση του αλγόριθμου εκτίμησης κατάστασης WLS και ακολουθεί αναλυτική περιγραφή των υλοποιήσεων, παρουσίαση και ανάλυση των μετρήσεων, καθώς και συμπεράσματα και γνώση που αποκομίσαμε από τις εφαρμογές αυτές.

4.2 Εκτίμηση Κατάστασης ενός Σ.Η.Ε.

Τα Συστήματα Ηλεκτρικής Ενέργειας παρακολουθούνται συνεχώς με στόχο να διατηρηθεί η λειτουργία τους σε ασφαλή κατάσταση. Η λειτουργία της Εκτίμησης Κατάστασης εφαρμόζεται για τον σκοπό αυτό. Επεξεργάζεται μετρήσεις, με στόχο να παρέχει μια βέλτιστη εκτίμηση της παρούσας λειτουργικής κατάστασης.

Το πρόβλημα της εκτίμησης κατάστασης ερευνάται από τα τέλη της δεκαετίας του 1960, όταν και πρωτοεισήχθη ο όρος. Όντας μια on-line λειτουργία, εμπεριέχει υπολογιστικά θέματα σχετικά με την ταχύτητα, την αποθήκευση και την αριθμητική ευρωστία των αλγόριθμων εκτίμησης. Οι εκτιμητές κατάστασης, λειτουργούν και ως φίλτρα λανθασμένων μετρήσεων , δεδομένων και άλλων πληροφοριών που λαμβάνονται μέσω του συστήματος μετρήσεων. Ακόμη, έχουν αναπτυχθεί αλγόριθμοι αναγνώρισης λαθών σε αναλογικές μετρήσεις αλλά και ειδικές μέθοδοι για την ανίχνευση σφαλμάτων στις πληροφορίες τοπολογίας και παραμέτρων του δικτύου.

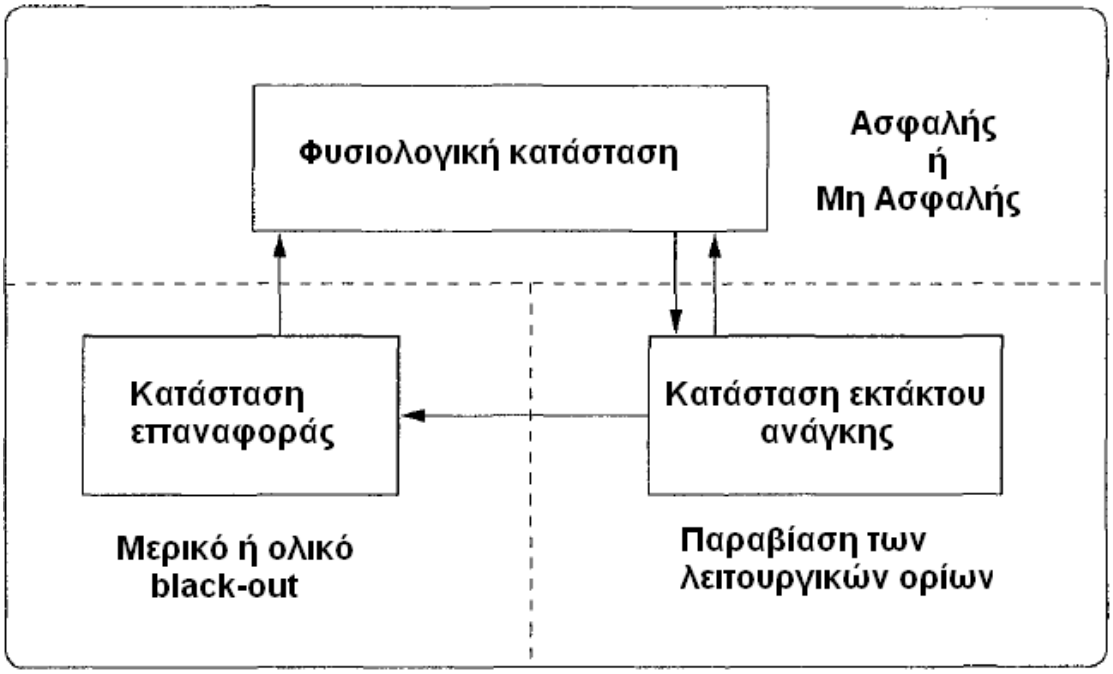
Στη συνέχεια, παρουσιάζουμε αναλυτικά τις λειτουργικές καταστάσεις στις οποίες μπορεί να βρεθεί ένα Σ.Η.Ε. και έπειτα παρουσιάζουμε τις μεθόδους που έχουν αναπτυχθεί για την εκτίμηση των καταστάσεων αυτών.

4.2.1 Περιγραφή των Καταστάσεων Ενός Σ.Η.Ε.

Τα συστήματα ηλεκτρικής ενέργειας, γενικά, αποτελούνται από υποσυστήματα μεταφοράς, υπομεταφοράς, διανομής καθώς και παραγωγής. Τα δίκτυα μεταφοράς δύνανται να περιέχουν μεγάλο αριθμό υποσταθμών οι οποίοι είναι

συνδεδεμένοι με γραμμές μεταφοράς, μετασχηματιστές και άλλα μέσα ελέγχου και προστασίας του δικτύου. Η ισχύς μπορεί να εγχέεται στο δίκτυο μέσω γεννητριών η να απορροφάται από το σύστημα από τα φορτία αυτών των υποσταθμών.

Οι λειτουργικές συνθήκες ενός συστήματος ηλεκτρικής ενέργειας σε ένα συγκεκριμένο σημείο μια καθορισμένη χρονική στιγμή, μπορούν να προσδιοριστούν εάν το μοντέλο του δικτύου και οι σύνθετες τάσεις (μέτρο-γωνία) κάθε ζυγού του δικτύου είναι γνωστές. Καθώς το σύνολο των σύνθετων τάσεων προσδιορίζει πλήρως το σύστημα, αναφέρεται ως η στατική συνθήκη του συστήματος. Σύμφωνα με το Σχήμα 4.1 που ακολουθεί, το σύστημα, καθώς οι συνθήκες λειτουργίας μεταβάλλονται, είναι πιθανό να βρεθεί σε μία από τις εξής τρεις δυνατές καταστάσεις: *Φυσιολογική*, *Εκτάκτου Ανάγκης* και *Αποκατάστασης*.



Σχήμα 4.1: Διάγραμμα καταστάσεων λειτουργιάς Συστήματος Ηλ. Ενέργειας

Ένα Σ.Η.Ε. θεωρείται ότι λειτουργεί σε *Φυσιολογική Κατάσταση* αν όλα τα φορτία μπορούν να τροφοδοτηθούν από τις υπάρχουσες γεννήτριες χωρίς να παραβιάζεται κανένας λειτουργικός περιορισμός. Οι περιορισμοί αυτοί περιλαμβάνουν τα όρια φόρτισης των γραμμών μεταφοράς καθώς και τα άνω και κάτω όρια των μέτρων των τάσεων. Η φυσιολογική κατάσταση του δικτύου θεωρείται ασφαλής εάν το σύστημα μπορεί να παραμείνει σε ομαλή κατάσταση ακόμη και μετά την παρουσίαση συγκεκριμένων - κρίσιμων ενδεχομένων. Άξια ενδιαφέροντος απρόοπτα αποτελούν η διακοπή λειτουργιάς μιας γραμμής μεταφοράς ή μιας γεννήτριας εξαιτίας απροσδόκητων σφαλμάτων στον εξοπλισμό ή λόγω φυσικών αιτιών όπως π.χ. οι καταιγίδες. Στην αντίθετη περίπτωση η φυσιολογική κατάσταση κατατάσσεται ως μη ασφαλής και ενώ το ισοζύγιο ισχύος σε κάθε ζυγό καθώς και όλοι οι λειτουργικοί ανισοτικοί περιορισμοί συνεχίζουν να ικανοποιούνται, το σύστημα παραμένει ευάλωτο όσον αφορά κάποια συγκεκριμένα απρόοπτα. Αν το σύστημα βρεθεί να είναι σε ομαλή - μη ασφαλή λειτουργία, τότε συγκεκριμένοι τρόποι δράσης πρέπει να εφαρμοστούν ουτοσώστε να παρεμποδιστεί η μεταπήδηση του σε συνθήκη *Εκτάκτου Ανάγκης*. Τέτοιοι περιοριστικοί έλεγχοι μπορούν να προσδιοριστούν με τη βοήθεια ενός προγράμματος ασφάλειας,

εξαναγκασμένης και βέλτιστης ροής φορτίου το οποίο θα λαμβάνει υπόψη του συγκεκριμένες καταστάσεις κινδύνου.

Οι συνθήκες λειτουργίας δύνανται να μεταβληθούν σημαντικά εξαιτίας ενός συμβάντος που μπορεί να οδηγήσει στην παραβίαση κάποιων λειτουργικών περιορισμών, ενώ το Σ.Η.Ε. συνεχίζει να παρέχει ισχύ σε όλα τα φορτία που είναι συνδεδεμένα στο σύστημα. Σε μια τέτοια περίπτωση το δίκτυο θεωρείται πως λειτουργεί σε *Κατάσταση Εκτάκτου Ανάγκης*. Σε ένα τέτοιο ενδεχόμενο, επιβάλλεται η εφαρμογή άμεσων διορθωτικών ρυθμίσεων από τον διαχειριστή του δικτύου ώστε το σύστημα να επιστρέψει σε φυσιολογική κατάσταση.

Όσο το δίκτυο παραμένει σε κατάσταση εκτάκτου ανάγκης, η κατάρρευση ολόκληρου του συστήματος μπορεί να αποφευχθεί με διορθωτικές κινήσεις ελέγχου όπως η αποσύνδεση φορτίων, μετασχηματιστών, γραμμών μεταφοράς και άλλου εξοπλισμού. Με απόρριψη φορτίου και αναδιαρθρωμένη τοπολογία, λοιπόν, η παραβίαση των λειτουργικών ορίων μπορεί να επαλειφθεί και το σύστημα να επανέλθει σε σταθερότητα. Τότε η ισορροπία μεταξύ φορτίου και παραγωγής χρειάζεται να αποκατασταθεί για να αρχίσει η τροφοδότηση όλων των φορτίων. Εδώ, όπως είναι προφανές, έγινε περιγραφή της *Καταστάσεως Επαναφοράς*, ενώ οι ενέργειες που πρέπει να πραγματοποιηθούν για την μεταπήδηση στην ομαλή κατάσταση λειτουργίας αναφέρονται ως έλεγχοι επαναφοράς. Το διάγραμμα καταστάσεων που προηγείται παρουσιάζει τις πιθανές μεταβάσεις ανάμεσα στις λειτουργικές καταστάσεις.

Τα συστήματα Ηλεκτρικής Ενέργειας λειτουργούν υπό την εποπτεία των διαχειριστών του δικτύου από τα κέντρα ελέγχου της περιοχής. Ο κύριος στόχος του διαχειριστή είναι να διατηρεί το σύστημα σε ομαλή – ασφαλή κατάσταση καθώς οι συνθήκες μεταβάλλονται κατά την καθημερινή λειτουργία. Η πραγματοποίηση αυτού του στόχου απαιτεί συνεχή παρακολούθηση των συνθηκών του συστήματος, αναγνώριση της κατάστασης λειτουργίας αλλά και προσδιορισμό των απαραίτητων περιοριστικών ενεργειών σε περίπτωση που το σύστημα ευρεθεί να είναι Μη Ασφαλές. Αυτή η ακολουθία ενεργειών αναφέρεται ως ανάλυση ασφαλείας του συστήματος.

4.2.2 Διαδικασία Εκτίμησης Κατάστασης

Για την αναγνώριση της παρούσας λειτουργικής κατάστασης, οι εκτιμητές κατάστασης διευκολύνουν την ακριβή και αποτελεσματική παρακολούθηση των λειτουργικών περιορισμών σε ποσότητες όπως η φόρτιση των γραμμών μεταφοράς ή τα μέτρα των τάσεων των ζυγών. Παρέχουν μια αξιόπιστη, πραγματικού χρόνου βάση δεδομένων του συστήματος, και βασισμένοι στην υπάρχουσα κατάσταση, καθιστούν δυνατή την εφαρμογή λειτουργιών αποτίμησης της ασφάλειας του συστήματος με στόχο την ανάλυση δυσμενών ενδεχομένων, καθώς και τον καθορισμό των απαραίτητων διορθωτικών κινήσεων.

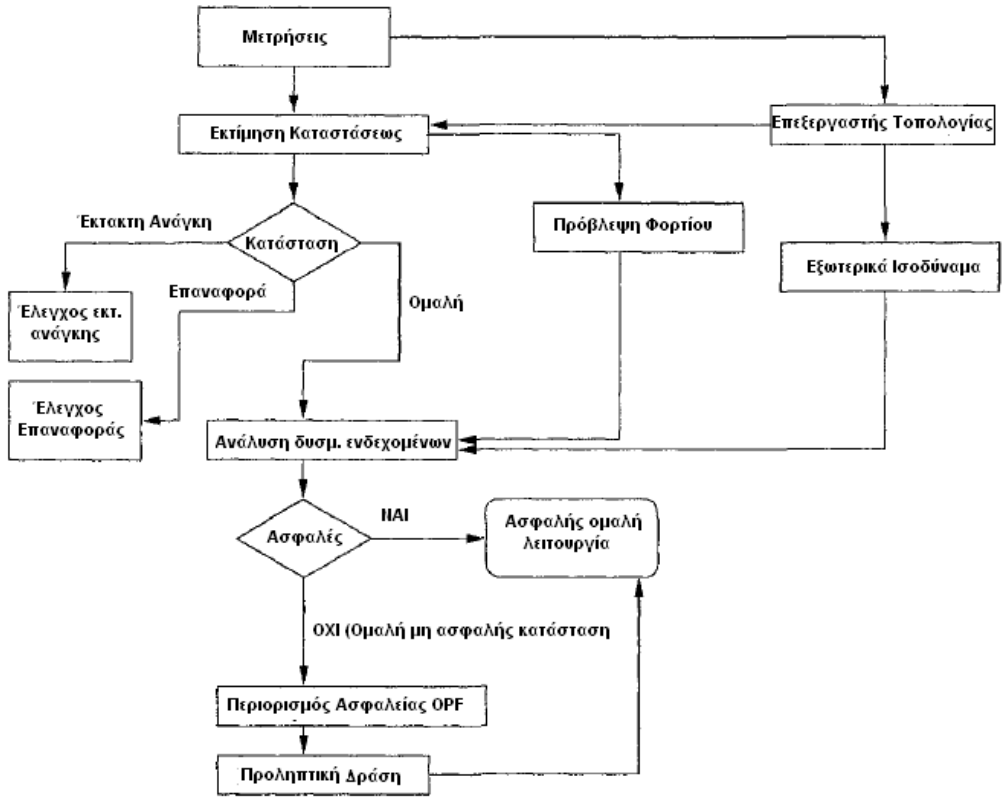
Οι τυπικοί εκτιμητές καταστάσεως περιλαμβάνουν τις ακόλουθες λειτουργίες:

- Επεξεργασία τοπολογίας: Συγκέντρωση δεδομένων καταστάσεως αποζευκτών και διακοπτών ισχύος αλλά και διαμόρφωση του μονογραμμικού διαγράμματος του συστήματος.
- Ανάλυση παρατηρησιμότητας: Καθορίζεται εάν με τις διαθέσιμες μετρήσεις μπορεί να προσδιοριστεί μια λύση της εκτίμησης καταστάσεως για ολόκληρο το σύστημα. Αναγνωρίζονται, ακόμη, οι μη παρατηρήσιμοι κλάδοι του συστήματος καθώς και (αν υπάρχουν) οι παρατηρήσιμες νησίδες.

- Λύση της εκτίμησης καταστάσεως: Καθορίζεται η βέλτιστη εκτίμηση για την κατάσταση του συστήματος, η οποία αποτελείται από σύνθετες τάσεις ζυγών σε ολόκληρο το Σ.Η.Ε, βασισμένη στο μοντέλο του δικτύου και στις μετρήσεις που έχουν παρθεί από το σύστημα. Ακόμη, παρέχεται η καλύτερη εκτίμηση για όλες τις ροές , τα φορτία, τις λήψεις των μετασχηματιστών και τις εξόδους των γεννητριών
- Επεξεργασία των λανθασμένων δεδομένων: Ανίχνευση της ύπαρξης μεγάλων σφαλμάτων στο σύνολο των μετρήσεων. Αναγνωρίζει και εξαλείφει λανθασμένες μετρήσεις δεδομένου ότι υπάρχει πλεόνασμα στη διαμόρφωση των μετρήσεων
- Επεξεργασία των παραμετρικών και δομικών σφαλμάτων: Εκτίμηση διάφορων παραμέτρων του δικτύου, όπως αυτές των μοντέλων των γραμμών μεταφοράς, των αλλαγών των λήψεων των μετασχηματιστών, των εγκάρσιων χωρητικοτήτων και αντιδράσεων. Ανίχνευση, ακόμη, κατασκευαστικών σφαλμάτων στη διαμόρφωση του δικτύου αλλά και αναγνώριση εσφαλμένων καταστάσεων των διακοπτών, δεδομένου και εδώ υπάρξεως αφθονίας μετρήσεων.

Κατ' αυτόν τον τρόπο, η εκτίμηση κατάστασης ενός Σ.Η.Ε. αποτελεί τον πυρήνα της on-line ανάλυσης ασφαλείας. Δρα σαν φίλτρο μεταξύ των ανεπεξέργαστων μετρήσεων που λαμβάνονται από το σύστημα και όλων των εφαρμογών που απαιτούν την πιο αξιόπιστη βάση δεδομένων για την παρούσα κατάσταση του συστήματος. Το Σχήμα 4.2 περιγράφει τα δεδομένα και τις λειτουργικές διασυνδέσεις μεταξύ των διαφόρων εφαρμογών που εμπλέκονται στην on-line διαδικασία της στατικής αποτίμησης ασφαλείας. Οι μετρήσεις, που περιλαμβάνουν τις θέσεις των αποζευκτών και των διακοπτών ισχύος στους υποσταθμούς, επεξεργάζονται από τον επεξεργαστή τοπολογίας, ο οποίος με τη σειρά του παράγει είναι μοντέλο γεννητριών – κλάδων του συστήματος. Το μοντέλο αυτό, όχι μόνο περιλαμβάνει όλους τους ζυγούς του κέντρου ελέγχου αλλά και ζυγούς γειτονικών συστημάτων. Οι πληροφορίες και οι μετρήσεις που αποκτώνται από τα γειτονικά συστήματα χρησιμοποιούνται για τη δημιουργία και την ενημέρωση του μοντέλου του εξωτερικού συστήματος. Επιπλέον, δύναται η ύπαρξη μη παρατηρήσιμων νησίδων μέσα στην ίδια περιοχή του συστήματος εξαιτίας παροδικής απώλειας των τηλεμετρήσεων, απόρριψης λανθασμένων δεδομένων ή άλλα μη αναμενόμενα σφάλματα. Τέτοιες περιοχές, είτε φυσικά τοποθετημένες στην περιοχή ελέγχου του συστήματος είτε ως μέρος του εξωτερικού συστήματος, θα εκτιμηθούν με τη χρήση ψευδομετρήσεων. Ψευδομετρήσεις μπορούν να παραχθούν με βάση τη βραχυχρόνια πρόβλεψη φορτίου, τα ιστορικά στοιχεία και άλλες παρόμοιες προσεγγιστικές μεθόδους. Βέβαια, στις ψευδομετρήσεις προσδίδονται υψηλές μεταβλητότητες (χαμηλά βάρη) ή επιβάλλονται ως κρίσιμες μετρήσεις κατά το σχεδιασμό. Ακόμη, μπορεί να υπάρχουν παθητικοί ζυγοί χωρίς καθόλου παραγωγή ή φορτίο, με μηδενική έγχυση ενεργού και άεργου ισχύος. Τέτοιες εγχύσεις ζυγών, αν και δεν μετρώνται, μπορούν να θεωρηθούν ως μετρήσεις δίχως σφάλμα στη διαμόρφωση της εκτίμησης καταστάσεως και να αναφέρονται ως "εικονικές" μετρήσεις. Τα αποτελέσματα στα οποία οδηγούμαστε από τη εκτίμηση καταστάσεως, ελέγχονται έτσι ώστε να κατατάξουμε την κατάσταση του συστήματος σε μία από τις τρεις κατηγορίες που εμφανίζονται στο Σχήμα 4.1. Σε περίπτωση που το δίκτυο ευρεθεί να είναι σε ομαλή κατάσταση, τότε απαιτείται η διεξαγωγή ανάλυσης δυσμενών ενδεχομένων για να καθοριστεί η ασφάλεια του συστήματος απέναντι σε ένα σύνολο συγκεκριμένων απροόπτων. Εάν το σύστημα αποδειχθεί μη ασφαλές , διορθωτικές-προληπτικές κινήσεις ελέγχου επιβάλλεται να υπολογιστούν μέσω της χρήσης κατάλληλου λογισμικού, όπως η βέλτιστη ροή φορτίου υπό περιορισμούς ασφαλείας. Υλοποιώντας αυτά τα μέτρα, θα οδηγήσει το σύστημα στην επιθυμητή ομαλή – ασφαλή κατάσταση λειτουργίας. Το Σχήμα 4.2, επιπλέον, παρουσιάζει και

τις δράσεις ελέγχου επαναφοράς οι οποίες θα πραγματοποιηθούν κάτω από μη φυσιολογικές συνθήκες λειτουργίας.



Σχήμα 4.2: Λειτουργικό Διάγραμμα: On-line στατική αποτίμηση ασφαλείας

4.2.3 Μέθοδοι Εκτίμησης Κατάστασης

Η στατική εκτίμηση καταστάσεως αναφέρεται στη διαδικασία λήψης των διανυσμάτων της τάσης σε όλους τους ζυγούς τους συστήματος σε δοσμένο σημείο συγκεκριμένη χρονική στιγμή. Τούτο μπορεί να επιτευχθεί με άμεσα μέσα που περιλαμβάνουν πολύ ακριβείς συγχρονισμένες μετρήσεις διανυσμάτων τάσεων σε όλους τους ζυγούς του συστήματος. Ωστόσο, μια τέτοια προσέγγιση θα ήταν πολύ ευάλωτη σε λανθασμένες μετρήσεις και σφάλματα τηλεμετρήσεων. Εν αντιθέσει, η διαδικασία της εκτίμησης κατάστασης χρησιμοποιεί ένα σύνολο περιορισμένων μετρήσεων με στόχο το φιλτράρισμα τέτοιων λαθών και την εύρεση μιας βέλτιστης εκτίμησης. Οι μετρήσεις μπορεί να περιλαμβάνουν όχι μόνο τις συμβατικές μετρήσεις τάσεων και ισχύος, αλλά και άλλες, όπως τα πλάτη των ρευμάτων ή και τα συγχρονισμένα διανύσματα των τάσεων. Η ταυτόχρονη μέτρηση ποσοτήτων σε διάφορα τμήματα του δικτύου είναι πρακτικά αδύνατη και γι' αυτό μια μικρή ανοχή είναι γενικά αποδεκτή. Αυτή η ανοχή δικαιολογείται κυρίως λόγω της αργής μεταβολής των λειτουργικών συνθηκών του συστήματός στην ομαλή – φυσιολογική κατάσταση λειτουργίας.

Ο ορισμός της κατάστασης του συστήματος συνήθως περιλαμβάνει μόνο τα διανύσματα των τάσεων των ζυγών σε μόνιμη κατάσταση. Αυτό, βέβαια, συνεπάγεται ότι η τοπολογία και οι παράμετροι του δικτύου είναι πλήρως γνωστά. Εντούτοις, σφάλματα στην τοπολογία και στις παραμέτρους εμφανίζονται περιστασιακά εξαιτίας διαφόρων λόγων όπως μη προγραμματισμένες διακοπές της λειτουργίας τμήματος του δικτύου, προβλήματα στις γραμμές μεταφοράς τις ζεστές ημέρες κλπ.

Εκτίμηση μέγιστης πιθανοφάνειας

Ο σκοπός της εκτίμησης καταστάσεως είναι να προσδιορίσει την πιο πιθανή κατάσταση του συστήματος με βάση τις μετρούμενες ποσότητες κατά μήκος του δικτύου. Ένας τρόπος να επιτευχθεί τούτο είναι μέσω της εκτίμησης μεγίστης πιθανοφάνειας, μιας μεθόδου ευρέως χρησιμοποιούμενης στη στατιστική. Τα λάθη των μετρήσεων θεωρείται πως έχουν μια γνωστή κατανομή πιθανότητας με άγνωστες παραμέτρους. Η Αθροιστική Συνάρτηση Πυκνότητας Πιθανότητας μπορεί να γραφτεί για όλες τις μετρήσεις βάσει αυτών των άγνωστων παραμέτρων. Η συνάρτηση αυτή αναφέρεται ως η συνάρτηση πιθανοφάνειας που αποκτά τη μέγιστη τιμή της όταν οι άγνωστες παράμετροι επιλέγονται έτσι ώστε να είναι όσο το δυνατόν πιο κοντά στις πραγματικές τους τιμές. Ως εκ τούτου, προσδιορίζεται ένα πρόβλημα βελτίωσης με στόχο τη μεγιστοποίηση της συνάρτησης πιθανοφάνειας συναρτήσει αυτών των αγνώστων παραμέτρων. Το αποτέλεσμα θα δώσει τις εκτιμήσεις μέγιστης πιθανοφάνειας για τις παραμέτρους που μας ενδιαφέρουν.

Τα λάθη των μετρήσεων συνήθως θεωρούνται πως έχουν κανονική (Gauss) κατανομή. Οι παράμετροι για μια τέτοια κατανομή είναι ο μέσος όρος μ καθώς και η διακύμανση σ^2. Το πρόβλημα της εκτίμησης της μέγιστης πιθανοφάνειας λύνεται τότε γι' αυτές τις δύο παραμέτρους . Η γκαουσιανή Συνάρτηση Πυκνότητας Πιθανότητας (ΣΠΠ) και η αντίστοιχη συνάρτηση κατανομής πιθανότητας παρουσιάζονται σύντομα παρακάτω.

Συνάρτηση Πιθανοφάνειας

Θεωρούμε την Αθροιστική Συνάρτηση Πυκνότητας Πιθανότητας ή οποία αντιπροσωπεύει την πιθανότητα μέτρησης m ανεξάρτητων μετρήσεων η κάθε μία εκ των οποίων έχει την ίδια Γκαουσιανή Συνάρτησης Πυκνότητας Πιθανότητας. Η Αθροιστική ΣΠΠ μπορεί απλά να εκφραστεί ως το γινόμενο των ανεξάρτητων ΣΠΠ εάν κάθε μέτρηση υποτίθεται ανεξάρτητη των υπόλοιπων. Έτσι,

$$f_m(z) = f(z_1)f(z_2)\dots f(z_m)$$
$$z^T : [z_1, z_2, \dots, z_m]$$

όπου z_i η i-ιοστή μέτρηση

Η συνάρτηση $f_m(z)$ είναι η συνάρτηση πιθανοφάνειας για τη μεταβλητή z. Βασικά, είναι ένα μέτρο της πιθανότητας να παρατηρηθεί ένα συγκεκριμένο σύνολο μετρήσεων στο διάνυσμα z.

Ο αντικειμενικός σκοπός της εκτίμησης της μεγίστης πιθανοφάνειας είναι να μεγιστοποιήσει τη συνάρτηση πιθανοφάνειας μεταβάλλοντας τις παραμέτρους της συνάρτησης πυκνότητας πιθανότητας μ και σ. Κατά τον προσδιορισμό των βέλτιστων τιμών τους, για την απλοποίηση της διαδικασίας βελτιστοποίησης, η συνάρτηση συνήθως αντικαθίσταται από τον λογάριθμο της. Η τροποποιημένη συνάρτηση ονομάζεται Λογαριθμική Συνάρτηση Πιθανοφάνειας και δίνεται από:

$$L = logf_m(z) = \sum_{i=1}^{m} logf(z_i) = -\frac{1}{2}\sum_{i=1}^{m}(\frac{z_i-\mu_i}{\sigma_i})^2 - \frac{m}{2}log2\pi - \sum_{i=1}^{m} log\sigma_i$$

Η εκτίμηση μέγιστης πιθανοφάνειας (ΕΜΠ) μεγιστοποιεί τη συνάρτηση (λογαριθμικής) πιθανοφάνειας για ένα δεδομένο σύνολο παρατηρήσεων z_1, z_2, z_3,…, z_m. Έτσι, επιβάλλεται η επίλυση του ακόλουθου προβλήματος:

$$\max log f_m(z)$$

Ή

$$\min \sum_{i=1}^{m} (\frac{z_i - \mu_i}{\sigma_i})^2$$

Το πρόβλημα ελαχιστοποίησης αυτό μπορεί να γραφτεί και με βάση τα υπόλοιπα r_i των μετρήσεων i, τα οποία ορίζονται ως εξής:

$$r_i = z_i - \mu_i = z_i - E(z_i)$$

όπου η μέση τιμή μ_i ή η αναμενόμενη τιμή $E(z_i)$ της μέτρησης z_i εκφράζεται ως $h_i(x)$, μια μη γραμμική συνάρτηση που συσχετίζει το διάνυσμα κατάστασης του συστήματος x με την ιοστή μέτρηση. Το τετράγωνο του κάθε υπολοίπου r_i^2 σταθμίζεται με βάρος $W_{ii}=\sigma_i^{-2}$ που είναι αντιστρόφως συνδεδεμένο με τη διακύμανση του λάθους για τη συγκεκριμένη μέτρηση. Συνεπώς, το πρόβλημα ελαχιστοποίησης της προηγούμενης εξίσωσης είναι ισοδύναμο με την ελαχιστοποίηση του σταθμισμένου αθροίσματος των τετραγώνων των υπολοίπων, ή την επίλυση του ακολούθου προβλήματος βελτιστοποίησης για το διάνυσμα κατάστασης x:

$$\min \sum_{i=1}^{m} W_{ii} r_i^2$$

$$z_i = h_i(x) + r_i, 1 = 1, \dots, m.$$

Η λύση του ανωτέρου προβλήματος βελτιστοποίησης ονομάζεται εκτίμηση σταθμισμένων ελαχίστων τετραγώνων (WLS estimator) για το x.

Μοντέλο μετρήσεων

Θεωρούμε το σύνολο των μετρήσεων που δίνεται από το διάνυσμα z:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} h_1(x_1, x_2, \dots, x_n) \\ h_2(x_1, x_2, \dots, x_n) \\ \vdots \\ h_m(x_1, x_2, \dots, x_n) \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix} = h(x) + e$$

Όπου $h^T= [h_1(x),h_2(x),\dots,h_m(x)]$

$h_i(x)$ ή μη γραμμική συνάρτηση που συσχετίζει την ιοστή μέτρηση με το διάνυσμα κατάστασης του συστήματος x

$x^T=[x_1,x_2,\dots,x_m]$ το διάνυσμα κατάστασης του συστήματος

$e^T=[e_1,e_2,\dots,e_m]$ το διάνυσμα των σφαλμάτων μετρήσεων

Όσον αφορά τα σφάλματα μετρήσεων έχουν γίνει οι παρακάτω συνηθισμένες συμβάσεις – υποθέσεις:

- $E(e_i) = 0$, i = 1,…,m.
- Τα σφάλματα των μετρήσεων είναι ανεξάρτητα. Έτσι, $Cov(e)= E[e\ e^T]=R=diag\{\sigma_1^2, \sigma_2^2,..., \sigma_m^2\}$

Η τυπική απόκλιση σ_i κάθε μιας από τις i μετρήσεις υπολογίζεται έτσι ώστε να αντιπροσωπεύει την αναμενόμενη ακρίβεια του αντίστοιχου οργάνου που χρησιμοποιήθηκε.

Ο εκτιμητής WLS θα ελαχιστοποιήσει την ακόλουθη αντικειμενική συνάρτηση:

$$J(x) = \sum_{i=1}^{m} \frac{(z_i - h_i(x))^2}{R_{ii}}$$

$$= [z - h(x)]^T R^{-1} [z - h(x)]$$

Στο ελάχιστο, οι πρώτης τάξεως συνθήκες ακρότατου θα πρέπει να ικανοποιούνται. Τούτο εκφράζεται σε συμπτυγμένη μορφή ως εξής:

$$g(x) = \frac{\theta J(x)}{\theta x} = -H^T(x)R^{-1}[z - h(x)] = 0$$

όπου

$$H(x) = [\frac{\theta h(x)}{\theta x}]$$

Αναπτύσσοντας τη μη γραμμική συνάρτηση g(x) σε σειρά Taylor γύρω από το διάνυσμα κατάστασης x^k προκύπτει:

$$g(x) = g(x^k) + G(x^k)(x - x^k) + \ldots = 0$$

Αμελώντας τους μεγιστοτάξιους όρους οδηγούμαστε σε μια επαναληπτική μεθοδολογία γνωστή ως Gauss-Newton όπως χαρακτηριστικά δείχνεται παρακάτω:

$$x^{k+1} = x^k - [G(x^k)]^{-1} g(x^k)$$

όπου k ο δείκτης επανάληψης και x^k το διάνυσμα της λύσης στην k-στη επανάληψη.

$$G(x^k) = \frac{\theta g(x^k)}{\theta x} = H^T(x^k)R^{-1}H(x^k)$$

$$g(x^k) = -H^T(x^k)R^{-1}[z - h(x^k)]$$

Ο πίνακας G(x) ονομάζεται μήτρα κέρδους. Είναι αραιός, θετικά ορισμένος και συμμετρικός δεδομένου πως το σύστημα είναι πλήρως παρατηρήσιμο. Ο G(x) είναι γενικά μη αντιστρέψιμος, τουναντίον ωστόσο, δύναται να αναλυθεί στους τριγωνικούς του παράγοντες και έτσι το ακόλουθο σύνολο αραιών και γραμμικών εξισώσεων επιλύεται χρησιμοποιώντας μπρος - πίσω αντικαταστάσεις σε κάθε επανάληψη k:

$$[G(x^k)]\Delta x^{k+1} = H^T(x^k)R^{-1}[z - h(x^k)]$$

όπου $\Delta x^{k+1} = x^{k+1} - x^k$. Το σύνολο αυτό των εξισώσεων ονομάζεται συχνά και «κανονικές εξισώσεις».

4.2.4 Αλγόριθμος WLS Εκτίμησης Κατάστασης

Η WLS(σταθμισμένα ελάχιστα τετράγωνα) εκτίμηση καταστάσεως περιλαμβάνει την επαναληπτική επίλυση των κανονικών εξισώσεων. Μια αρχική υπόθεση επιβάλλεται για το διάνυσμα κατάστασης x^0. Όπως και στην περίπτωση της ροής φορτίου, η εικασία αυτή αντιστοιχεί σε αρχικές συνθήκες, όπου οι τάσεις όλων των ζυγών υποτίθενται ίσες με 1,0 ανά μονάδα και σε φάση μεταξύ τους (flat conditions). Ο αλγόριθμος της επαναληπτικής επίλυσης για το πρόβλημα της WLS εκτίμησης κατάστασης μπορεί να σκιαγραφηθεί ως εξής:

1. Αρχή επαναλήψεων. Θέτουμε τον δείκτη επανάληψης k=0
2. Αρχικοποίηση του διανύσματος κατάστασης x^k, συνήθως με flat αρχικές συνθήκες (τάσεις 1 α.μ. και μηδενικές γωνίες)
3. Υπολογισμός της μήτρας κέρδους $G(x^k)$
4. Υπολογισμός του δεξιού μέλους της εξίσωσης 4.1, $t^k = H(x^k)^T R^{-1}(z - h(x^k))$
5. Παραγοντοποίηση της $G(x^k)$ και επίλυση για Δx^k
6. Έλεγχος σύγκλισης.
7. Αν δεν επιτεύχθηκε σύγκλιση, ενημέρωση των $x^{k+1} = x^\kappa + \Delta x^k$, k=k+1 και επιστροφή στο βήμα 3.

Ο ανωτέρω αλγόριθμος, βασικά, περιλαμβάνει τους ακόλουθους υπολογισμούς σε κάθε επανάληψη k:

1. Υπολογισμός του δεξιού μέλους της εξίσωσης 4.1, t^k (βήμα 4)
   a. Υπολογισμός της συνάρτησης μετρήσεων $h(x^k)$
   b. Κατασκευή της Ιακωβιανής μήτρας $H(x^k)$
2. Υπολογισμός της $G(x^k)$ και επίλυση της εξίσωσης 4.1
   a. Κατασκευή της μήτρας κέρδους $G(x^k)$

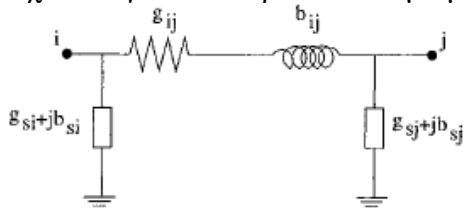
b. Ανάλυση της $G(x^k)$ στους Cholesky παράγοντές της
c. Εφαρμογή της μπρος/πίσω αντικατάστασης για την επίλυση του Δx^{k+1}

Συνάρτηση μετρήσεων

Οι μετρήσεις μπορεί να είναι διαφόρων ειδών. Οι πιο συνηθισμένες είναι: μετρήσεις ροών ισχύος στις γραμμές, εγχύσεων ισχύος στους ζυγούς, πλάτη τάσεων στους ζυγούς καθώς και πλάτη ροών ρεύματος στις γραμμές. Οι μετρήσεις αυτές δύνανται να εκφρασθούν ως συνάρτηση των μεταβλητών κατάστασης σε καρτεσιανές ή σε πολικές συντεταγμένες. Σε περίπτωση που χρησιμοποιηθούν πολικές συντεταγμένες για ένα σύστημα που περιλαμβάνει N ζυγούς, το διάνυσμα κατάστασης θα περιλαμβάνει 2N-1 στοιχεία, εκ των οποίων N πλάτη τάσεων ζυγών και (N-1) φάσεις γωνιών, όπου η φάση της γωνίας του ζυγού αναφοράς τίθεται ίση με μια αυθαίρετη τιμή ίση με το μηδέν. Το διάνυσμα κατάστασης x θα έχει την ακόλουθη μορφή θεωρώντας πως ο ζυγός 1 επιλέγεται ως αναφορά:

$X^T=[\theta_2\theta_3...\theta_N V_1 V_2 \dots V_N]$

Οι εκφράσεις του κάθε τύπου μέτρησης δίνονται παρακάτω, υποθέτοντας ότι ισχύει το γενικό 2θυρο ισοδύναμο μοντέλο π του ακόλουθου σχήματος



Οι πραγματικές και οι αντιδραστικές εγχύσεις ισχύος στο ζυγό i δίνονται από:

$$P_i = V_i \sum_{j \in N_i} V_j(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij})$$

$$Q_i = V_i \sum_{j \in N_i} V_j(G_{ij}\sin\theta_{ij} + B_{ij}\cos\theta_{ij})$$

ενώ οι πραγματικές και αντιδραστικές ροές ισχύος από τον ζυγό i στο ζυγό j από:

$$P_{ij} = V_i^2(g_{si} + g_{ij}) - V_iV_j(g_{ij}\cos\theta_{ij} + b_{ij}\sin\theta_{ij})$$

$$Q_{ij} = -V_i^2(b_{si} + b_{ij}) - V_iV_j(g_{ij}\sin\theta_{ij} - b_{ij}\cos\theta_{ij})$$

Ακόμη τα πλάτη των ροών ρεύματος στις γραμμές από τον ζυγό i στο ζυγό j από:

$$I_{ij} = \frac{\sqrt{P_{ij}^2 + Q_{ij}^2}}{V_i}$$

Ή αγνοώντας την εγκάρσια αγωγιμότητα ($g_{si}+jb_{si}$):

$$I_{ij} = \sqrt{(g_{ij}^2 + b_{ij}^2)(V_i^2 + V_j^2 - 2V_iV_j\cos\theta_{ij})}$$

Όπου V_i,θ_i το πλάτος της τάσης και η φάση της γωνίας στον ζυγό i
$\theta_{ij}= \theta_i- \theta_j$
$G_{ij}+jB_{ij}$ το ij-οστό στοιχείο της μήτρας αγωγιμοτήτων
$g_{ij}+jb_{ij}$ η αγωγιμότητα σειράς του κλάδου που συνδέει τους ζυγούς i και j
$g_{sj}+jb_{sj}$ η εγκάρσια αγωγιμότητα του κλάδου i
N_i το σύνολο των ζυγών που είναι απευθείας συνδεδεμένοι με το ζυγό i

Jacobian Matrix

Η κατασκευή της Ιακωβιανής μήτρας πραγματοποιείται όπως δείχνεται παρακάτω:

$$H = \begin{bmatrix} \frac{\partial P_{inj}}{\partial \theta} & \frac{\partial P_{inj}}{\partial V} \\ \frac{\partial P_{flow}}{\partial \theta} & \frac{\partial P_{flow}}{\partial V} \\ \frac{\partial Q_{inj}}{\partial \theta} & \frac{\partial Q_{inj}}{\partial V} \\ \frac{\partial Q_{flow}}{\partial \theta} & \frac{\partial Q_{flow}}{\partial V} \\ 0 & \frac{\partial V_{mag}}{\partial V} \end{bmatrix}$$

Οι εκφράσεις για κάθε μέλος, τότε είναι:

- Στοιχεία που αντιστοιχούν σε μετρήσεις ενεργού εγχύσεως ισχύος

$$\frac{\partial P_i}{\partial \theta_i} = \sum_{j=1}^{N} V_i V_j(-G_{ij}sin\theta_{ij} + B_{ij}cos\theta_{ij}) - V_i^2 B_{ii}$$

$$\frac{\partial P_i}{\partial \theta_j} = V_i V_j(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij})$$

$$\frac{\partial P_i}{\partial V_i} = \sum_{j=1}^{N} V_i\,(G_{ij}cos\theta_{ij} + B_{ij}sin\theta_{ij}) + V_i G_{ii}$$

$$\frac{\partial P_i}{\partial V_j} = V_i(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij})$$

- Στοιχεία που αντιστοιχούν σε μετρήσεις άεργου εγχύσεως ισχύος

$$\frac{\partial Q_i}{\partial \theta_i} = \sum_{j=1}^{N} V_i V_j(G_{ij}cos\theta_{ij} + B_{ij}sin\theta_{ij}) - V_i^2 G_{ii}$$

$$\frac{\partial P_i}{\partial \theta_j} = V_i V_j(-G_{ij}\cos\theta_{ij} - B_{ij}\sin\theta_{ij})$$

$$\frac{\partial Q_i}{\partial V_i} = \sum_{j=1}^{N} V_i\,(G_{ij}sin\theta_{ij} - B_{ij}cos\theta_{ij}) - V_i B_{ii}$$

$$\frac{\partial Q_i}{\partial V_j} = V_i(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij})$$

- Στοιχεία που αντιστοιχούν σε μετρήσεις πραγματικής ροής ισχύος

$$\frac{\partial P_{ij}}{\partial \theta_i} = V_i V_j(g_{ij}\sin\theta_{ij} - b_{ij}\cos\theta_{ij})$$

$$\frac{\partial P_{ij}}{\partial \theta_j} = -V_i V_j(g_{ij}\sin\theta_{ij} - b_{ij}\cos\theta_{ij})$$

$$\frac{\partial P_{ij}}{\partial V_i} = -V_j(g_{ij}\cos\theta_{ij} + b_{ij}\sin\theta_{ij}) + 2(g_{ij} + g_{si})V_i$$

$$\frac{\partial P_{ij}}{\partial V_j} = -V_i(g_{ij}\cos\theta_{ij} + b_{ij}\sin\theta_{ij})$$

- Στοιχεία που αντιστοιχούν σε μετρήσεις άεργου ροής ισχύος

$$\frac{\partial Q_{ij}}{\partial \theta_i} = -V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij})$$

$$\frac{\partial Q_{ij}}{\partial \theta_j} = V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij})$$

$$\frac{\partial Q_{ij}}{\partial V_i} = -V_j (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij}) - 2V_i (b_{ij} + b_{si})$$

$$\frac{\partial Q_{ij}}{\partial V_j} = -V_i (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij})$$

- Στοιχεία που αντιστοιχούν σε μετρήσεις πλάτους τάσεως

$$\frac{\partial V_i}{\partial V_i} = 1, \frac{\partial V_i}{\partial V_j} = 0, \frac{\partial V_i}{\partial \theta_i} = 1, \frac{\partial V_i}{\partial \theta_j} = 0$$

- Στοιχεία που αντιστοιχούν σε μετρήσεις πλάτους ρεύματος (αγνοώντας την εγκάρσια αγωγιμότητα του κλάδου)

$$\frac{\partial I_{ij}}{\partial \theta_i} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} V_i V_j sin\theta_{ij}$$

$$\frac{\partial I_{ij}}{\partial \theta_j} = -\frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} V_i V_j sin\theta_{ij}$$

$$\frac{\partial I_{ij}}{\partial V_i} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} (V_i - V_j cos\theta_{ij})$$

$$\frac{\partial I_{ij}}{\partial V_j} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} (V_j - V_i cos\theta_{ij})$$

Σημειώνεται εδώ ότι η Η είναι μια αραιή μήτρα, γεγονός που είναι πιο ευδιάκριτο σε μεγάλης έκτασης συστήματα όπου ο αριθμός των μη μηδενικών στοιχείων παραμένει γενικά σταθερός, ανεξαρτήτως του μεγέθους του συστήματος.

Η μήτρα κέρδους G

Η μήτρα κέρδους σχηματίζεται χρησιμοποιώντας την Ιακωβιανή Η αλλά και τη μήτρα συνδιακύμανσης των λαθών μετρήσεων, R. Η μήτρα συνδιακύμανσης υποτίθεται διαγώνια με διαγώνια στοιχεία τις διακυμάνσεις των μετρήσεων. Τελικά η G σχηματίζεται ως $G(x^k)= H^T R^{-1} H$ και έχει τις ακόλουθες ιδιότητες:

- Είναι δομικά και αριθμητικά συμμετρική.
- Είναι αραιή, αν και λιγότερη αραιή σε σύγκριση με την Η
- Γενικά είναι μια μη αρνητικά ορισμένη μήτρα με όλες τις ιδιοτιμές της μη αρνητικές. Για πλήρως παρατηρήσιμα δίκτυα είναι θετικά ορισμένη

Η μήτρα G κατασκευάζεται και αποθηκεύεται ως αραιή μήτρα για υπολογιστική αποδοτικότητα αλλά και λόγους μνήμης. Δομείται με την επεξεργασία μιας μέτρησης κάθε φορά. Έστω, Ιακωβιανή Η και η μήτρα συνδιακυμάνσεων R για ένα σύνολο μετρήσεων m, κάθε μία εκ των οποίων ανταποκρίνονται σε μια γραμμή όπως χαρακτηριστικά δείχνεται πιο κάτω.

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_m \end{bmatrix}, R = \begin{bmatrix} R_{11} & 0 & \ldots & 0 \\ 0 & R_{22} & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & . & R_{mm} \end{bmatrix}$$

Έπειτα, η μήτρα κέρδους μπορεί να ξαναγραφεί ως εξής:

$$G = \sum_{i=1}^{m} H_i^T R_{ii}^{-1} H_i$$

Καθώς οι πίνακες H_i είναι πολύ αραιά διανύσματα στήλες, το γινόμενο τους θα προκύψει κ αυτό ένας αραιός πίνακας. Μη μηδενικοί όροι στη G μπορούν, έτσι, να υπολογιστούν και να αποθηκευτούν σε αραιά μορφή.

Παραγοντοποίηση Cholesky της G

Η μήτρα κέρδους G δύναται να γραφεί σαν γινόμενο ενός κάτω τριγωνικού αραιού πίνακα και του αντίστροφου του, δηλαδή ως $G=LL^T$.

Σημειώνουμε εδώ πως η παραγοντοποίηση αυτή μπορεί να μην είναι δυνατό να πραγματοποιηθεί σε συστήματα που δεν είναι πλήρως παρατηρήσιμα. Σαν αποτέλεσμα, δεν μπορεί να προσδιοριστεί μια επίλυση της εκτίμησης καταστάσεως για τέτοια μη παρατηρήσιμα συστήματα.

Οι τριγωνικοί συντελεστές της G δεν είναι μοναδικοί και το πόσο αραιοί είναι εξαρτάται σημαντικά από τον τρόπο με τον οποίο πραγματοποιείται η τριγωνοποίηση και γι' αυτό υπάρχουν διάφοροι τρόποι βελτίωσης της αραιότητας των L συντελεστών που προκύπτουν.

Πραγματοποιώντας τις Μπρος/Πίσω Αντικαταστάσεις

Υποθέτοντας πως η μήτρα κέρδους έχει αναλυθεί σωστά στους παράγοντες Cholesky L και L^T, το επόμενο βήμα είναι η επίλυση της κανονικής εξίσωσης ως προς Δx^k, δηλ.

$$LL^T\Delta x^k = t^k$$

Όπου το t^k υποδηλώνει το δεξί μέλος της 666. Η λύση αυτή αποκτάται σε 2 στάδια:

1. Εμπρός αντικατάσταση: έστω $L^T\Delta x^k = u$. Τότε παίρνουμε τα στοιχεία του u ξεκινώντας από το u_1 και χρησιμοποιώντας αντικαταστάσεις στη μετασχηματισμένη εξίσωση $Lu = t^k$. Από την πάνω σειρά θα προκύψει η λύση για το u_1 ως t_1/L_{11}. Αντικαθιστώντας το u_1 στις εναπομείναντες σειρές θα μειώσει το σύνολο των εξισώσεων κατά 1. Επαναλαμβάνοντας την ίδια διαδικασία για το u_2 και τα υπόλοιπα στοιχεία, θα προκύψει η επίλυση για ολόκληρο το u.
2. Πίσω αντικατάσταση: Τώρα που πλέον το u είναι διαθέσιμο, χρησιμοποιούμε την $L^T\Delta x^k = u$, για να αντικαταστήσουμε το u και να επιλύσουμε για τα στοιχεία του Δx^k. Ετούτη τη φορά, οι αντικαταστάσεις επιβάλλεται να αρχίσουν από την τελευταία σειρά, όπου το τελευταίο στοιχείο του διανύσματος επίλυσης δίνεται ως $\Delta x^k(n)=u_n/L_{nn}$. Αντικαθιστώντας στις υπόλοιπες σειρές, η διαδικασία της πίσω αντικατάστασης συνεχίζεται μέχρι όλα τα στοιχεία να έχουν υπολογιστεί.

Σημειώνουμε πως τόσο η εμπρός όσο και η πίσω αντικατάσταση λειτουργούν πολύ αποδοτικά λόγω της αραιής δομής των τριγωνικών συντελεστών L.

4.2.4.1 Σειριακή Υλοποίηση με τη χρήση της PETSc

Η σειριακή υλοποίηση του αλγόριθμου WLS για την εκτίμηση κατάστασης έγινε σε γλώσσα C, με χρήση της βιβλιοθήκης PETSc. Ο σχεδιασμός βασίστηκε στο διαχωρισμό των βημάτων του αλγόριθμου και την υλοποίηση κάθε βήματος ως ξεχωριστή ρουτίνα. Έτσι, προέκυψαν οι εξής ρουτίνες:

- calculateH: Υπολογισμός της Ιακωβιανής Μήτρας H
- calculateG: Υπολογισμός της Μήτρας Κέρδους G
- calculatet: Υπολογισμός του δεξιού μέλους της εξίσωσης 4.1

Για την παραγοντοποίηση της μήτρας κέρδους και την επίλυση του συστήματος χρησιμοποιήθηκαν ρουτίνες της PETSc.

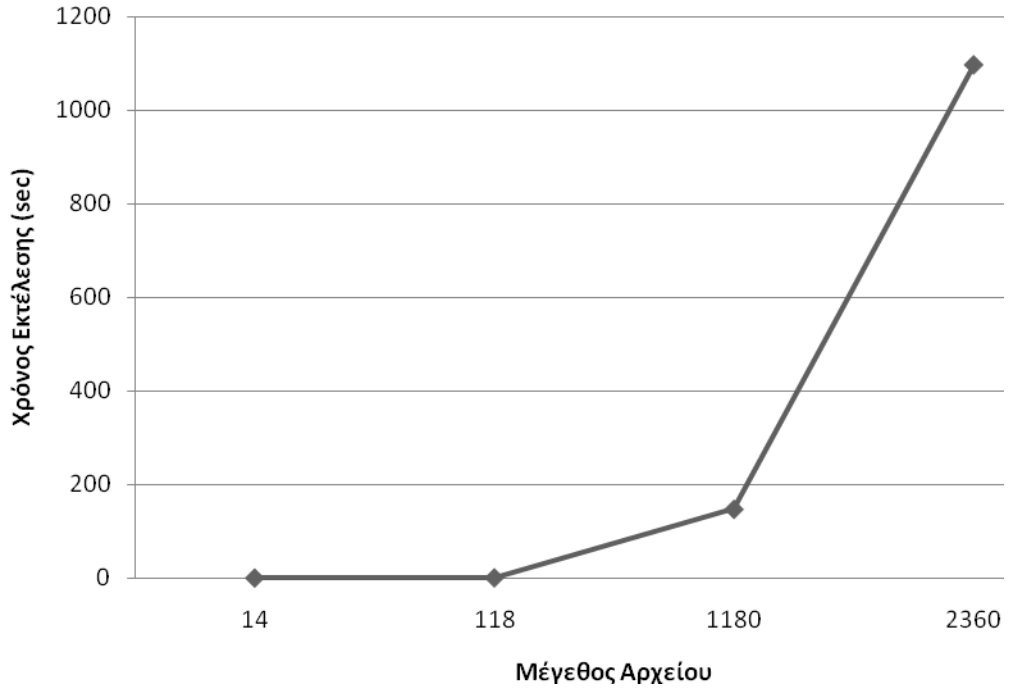
Η είσοδος στο πρόγραμμα γίνεται διαβάζοντας δύο αρχεία, τα οποία έχουν συγκεκριμένη δομή, η οποία περιγράφεται λεπτομερώς στο παράρτημα. Για την ανάγνωση των αρχείων αυτών και την αποθήκευση των απαραίτητων δεδομένων, υλοποιήθηκαν ξεχωριστές συναρτήσεις. Το πρώτο αρχείο, το οποίο έχει κατάληξη .sys, περιέχει δεδομένα για την τοπολογία του δικτύου, όπως τον αριθμό των ζυγών, των γεννητριών και των κλάδων του δικτύου, τις συνδέσεις μεταξύ των ζυγών, τις αγωγιμότητες στους κλάδους κ.ά. Τα δεδομένα αυτά, παρέχουν στο πρόγραμμα πληροφορίες για το μέγεθος και τη δομή των μητρών H και G, όπως και για την κατασκευή των μητρών αγωγιμοτήτων. Το δεύτερο αρχείο, με κατάληξη .ses, περιέχει μετρήσεις τάσης, ενεργού και άεργου ροής ισχύος και ενεργού και άεργου εγχύσεως. Τα δεδομένα αυτά χρησιμοποιούνται για την κατασκευή του διανύσματος μετρήσεων και κατ' επέκταση του δεξιού μέλους της εξίσωσης 4.1.

Λόγω της αραιότητας των πινάκων του προβλήματος, και κυρίως της μήτρας H, ο τύπος δεδομένων της PETSc που χρησιμοποιήθηκε για την αποθήκευσή τους είναι ο σειριακός AIJ (SeqAIJ). Αντίστοιχα, για την αποθήκευση των διανυσμάτων του προβλήματος χρησιμοποιήθηκε ο απλός σειριακός τύπος της PETSc. Για την επίλυση του συστήματος εξισώσεων σε κάθε επανάληψη, χρησιμοποιήθηκε η παραγοντοποίηση Cholesky, η οποία επιτυγχάνεται στην PETSc σε τέσσερα βήματα (MatGetFactor(), MatCholeskyFactorSymbolic(), MatCholeskyFactorNumeric() και MatSolve()). Για τον έλεγχο σύγκλισης, ως σφάλμα θεωρήθηκε η μέγιστη απόλυτη διαφορά μεταξύ του διανύσματος λύσης της τρέχουσας επανάληψης και του διανύσματος λύσης της προηγούμενης επανάληψης.

Το πρόγραμμα εκτελέστηκε στη συνέχεια για τέσσερα αρχεία, τα οποία περιείχαν δεδομένα για δίκτυα 14, 118, 1180 και 2360 ζυγών. Στον Πίνακα 4.1 παρουσιάζονται οι χρόνοι εκτέλεσης του προγράμματος, ενώ στο Σχήμα 4.3 παρουσιάζεται ένα διάγραμμα του συνολικού χρόνου εκτέλεσης σε σχέση με το μέγεθος του δικτύου.

	14 ζυγοί	**118 ζυγοί**	**1180 ζυγοί**	**2360 ζυγοί**
Συνολικός Χρόνος (sec)	0,007260009	0,462895996	146,8647576	1097,734655
Χρόνος CalculateH (sec)	0,001249023	0,173136475	16,75624023	43,29203125
Χρόνος CalculateG (sec)	0,002703125	0,02502832	0,233516113	0,3190896
Χρόνος Calculatet (sec)	0,002337646	0,077899902	6,732709229	17,13288916
Χρόνος Επίλυσης (sec)	0,000970215	0,186831299	123,142292	1036,990645

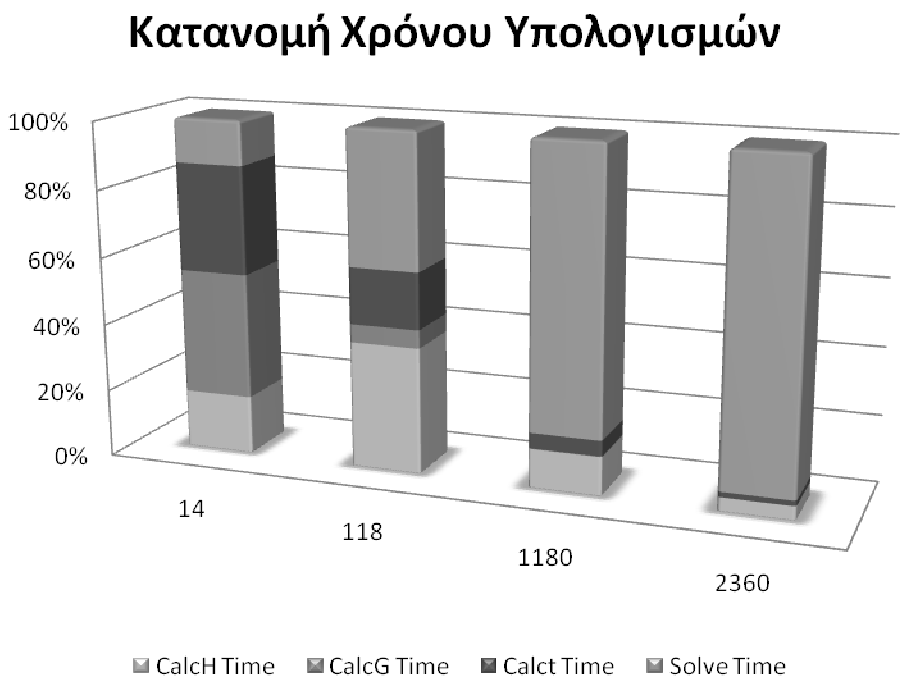
Πίνακας 4.1: Μετρήσεις χρόνου για τη σειριακή υλοποίηση του WLS αλγόριθμου



Σχήμα 4.3: Συνολικός χρόνος εκτέλεσης σειριακού WLS σε σχέση με το μέγεθος του δικτύου.

Από το διάγραμμα του Σχήματος 4.3, είναι φανερό ότι ο χρόνος εκτέλεσης του αλγόριθμου αυξάνεται εκθετικά με το μέγεθος του δικτύου. Όπως αναφέραμε και προηγουμένως, η εκτίμηση κατάστασης ενός Σ.Η.Ε. είναι μια διαδικασία ύψιστης σημασίας για την ομαλή λειτουργία και την ασφάλεια του δικτύου. Επίσης, καθώς οι συνθήκες στο δίκτυο μεταβάλλονται με το χρόνο, η ακρίβεια της εκτίμησης κατάστασης εξαρτάται από την ταχύτητα υπολογισμού της. Επιπλέον, η ταχύτητα υπολογισμού της εκτίμησης, είναι υψίστης σημασίας και για τον διαχειριστή του δικτύου, ώστε να μπορέσει εγκαίρως να λάβει τα κατάλληλα μέτρα για την αποφυγή μετάπτωσης σε ανεπιθύμητη κατάσταση. Οι παραπάνω διαπιστώσεις κάνουν φανερούς τους λόγους για τους οποίους η παραλληλοποίηση του αλγόριθμου WLS εκτίμησης κατάστασης είναι αναγκαία, ώστε να επιτύχουμε καλύτερη επίδοση του αλγόριθμου. Πριν προχωρήσουμε, όμως στην περιγραφή της διαδικασίας παραλληλοποίησης, αξίζει να εξετάσουμε την κατανομή του χρόνου εκτέλεσης στα διάφορα βήματα του αλγόριθμου. Η μελέτη της κατανομής αυτής θα μας πληροφορήσει για το ποιο ή ποια βήματα του αλγόριθμου περιέχουν υπολογισμούς οι οποίοι βλάπτουν την επίδοση και επιβραδύνουν την εκτέλεση. Έτσι, θα έχουμε έναν οδηγό για τη διαδικασία της παραλληλοποίησης, ώστε να γνωρίζουμε ποια βήματα αξίζει να εκτελεστούν παράλληλα, ώστε να πετύχουμε επιτάχυνση.

Η κατανομή του χρόνου επεξεργασίας στα διάφορα βήματα του αλγόριθμου παρουσιάζεται σχηματικά στο διάγραμμα του Σχήματος 4.4 που ακολουθεί:



Σχήμα 4.4: Κατανομή του χρόνου εκτέλεσης στα διάφορα βήματα του WLS αλγόριθμου

Είναι φανερό πως η πιο «βαριά» εργασία του αλγόριθμου είναι αυτή της επίλυσης του συστήματος των γραμμικών εξισώσεων σε κάθε επανάληψη. Από τους υπόλοιπους υπολογισμούς που γίνονται σε κάθε επανάληψη, τον περισσότερο χρόνο καταλαμβάνει η κατασκευή της μήτρας H και η κατασκευή της συνάρτησης μετρήσεων. Αυτές είναι και οι δύο λειτουργίες με βάση τις οποίες σχεδιάστηκε το παράλληλο πρόγραμμα, καθώς για τη λειτουργία της επίλυσης, η PETSc παρέχει ήδη υλοποιημένους παράλληλους επιλυτές.

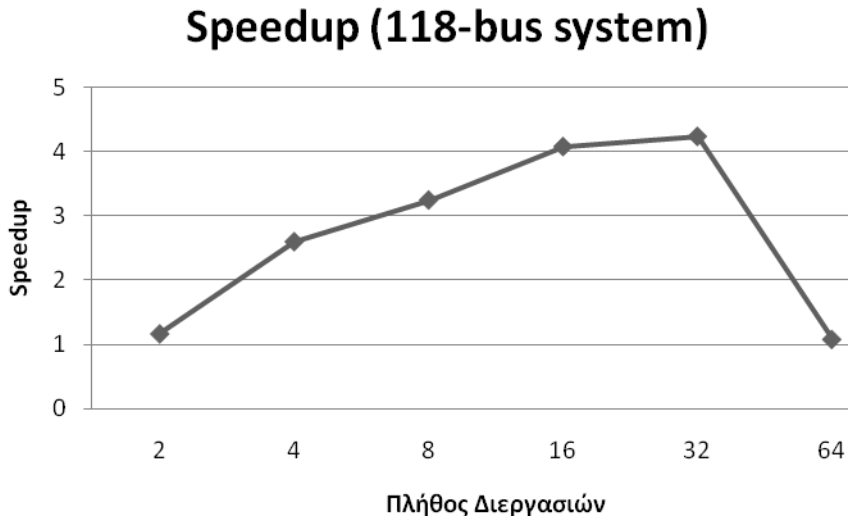
4.2.4.2 Παραλληλοποίηση με τη χρήση της PETSc

Οι σχεδιαστικές αποφάσεις της παράλληλης υλοποίησης βασίστηκαν στη μελέτη της κατανομής του χρόνου εκτέλεσης της σειριακής έκδοσης του αλγόριθμου. Η υλοποίηση βασίστηκε στην ιδέα της παραλληλοποίησης κάθε βήματος, διατηρώντας, όσο το δυνατόν, ανέπαφη τη δομή του προγράμματος. Έτσι, οι ρουτίνες που αναφέραμε προηγουμένως, παραλληλοποιήθηκαν η καθεμία ξεχωριστά. Οι εργασίες αρχικοποίησης και προετοιμασίας για την εκτέλεση του κυρίως πυρήνα του προγράμματος ανατέθηκαν στη διεργασία 0. Η διεργασία αυτή, αναλαμβάνει να διαβάσει τα αρχεία .sys και .ses, να αποθηκεύσει τα απαραίτητα δεδομένα σε δομές και να υπολογίσει τις μήτρες αγωγιμοτήτων. Στη συνέχεια, αναλαμβάνει να υπολογίσει πόσα και ποια δεδομένα πρέπει να σταλούν σε κάθε άλλη διεργασία. Καθολικά δεδομένα για το δίκτυο, όπως π.χ. ο αριθμός των ζυγών και των κλάδων, ο αριθμός των μετρήσεων, οι μήτρες αγωγιμοτήτων και το αρχικό διάνυσμα εισόδου γίνονται broadcast στις άλλες διεργασίες. Η υλοποίηση των βημάτων αυτών της αρχικοποίησης και της αποστολής των αρχικών δεδομένων έγιναν με τη χρήση των κλασικών ρουτινών του MPI και όχι με χρήση κατανεμημένων δομών της PETSc. Οι κύριες δομές του προγράμματος, όμως, οι οποίες συμμετέχουν στο βρόχο επίλυσης του συστήματος, υλοποιήθηκαν όλες ως κατανεμημένες PETSc δομές πινάκων και διανυσμάτων, ώστε να επωφεληθούμε από τη χρήση των έτοιμων ρουτινών πράξεων

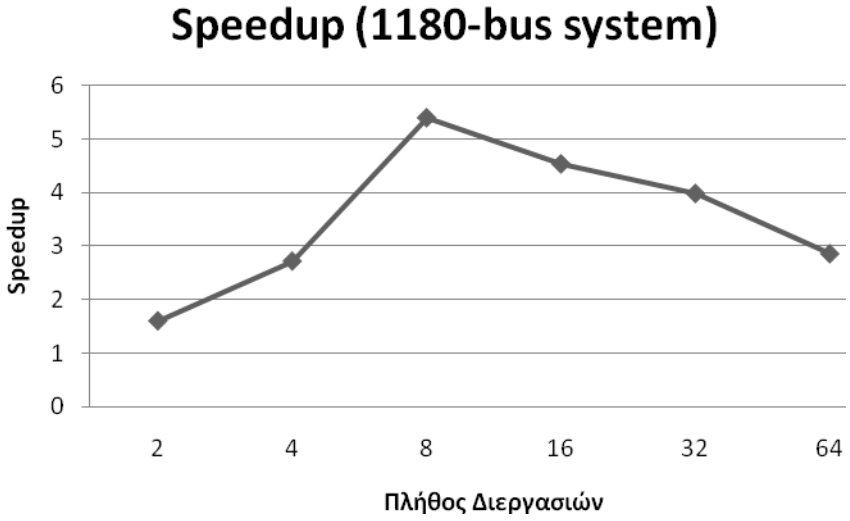
και επίλυσης. Έτσι, αφού μοιραστούν τα δεδομένα, δημιουργούνται οι κατάλληλες κατανεμημένες δομές και οι υπόλοιπες εργασίες εκτελούνται παράλληλα από όλες τις διεργασίες. Έτσι, όποια επικοινωνία απαιτείται κατά τους υπολογισμούς, ουσιαστικά «κρύβεται» μέσα στις ρουτίνες της PETSc. Η μόνη ανάγκη για επικοινωνία και ρητή χρήση MPI ρουτίνας, προκύπτει στο τέλος κάθε επανάληψης, όπου και πρέπει να ενημερωθεί το διάνυσμα κατάστασης και να υπολογιστεί το σφάλμα της επανάληψης.

4.2.5 Μετρήσεις

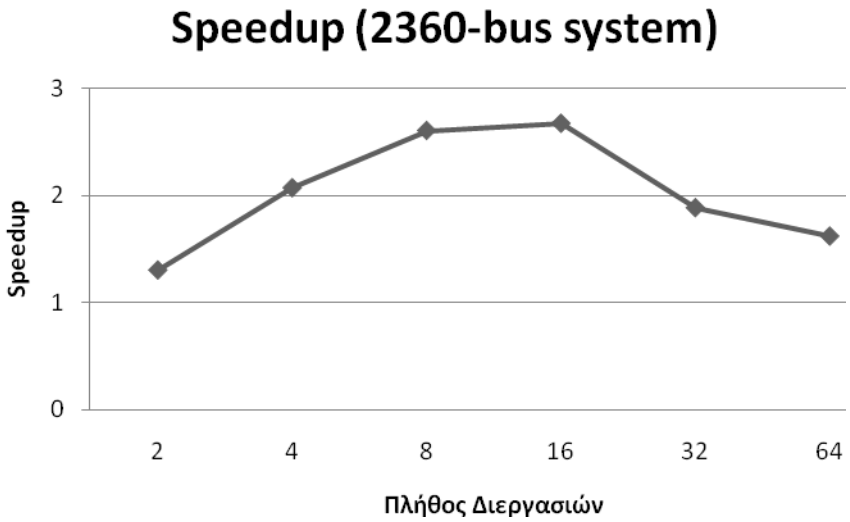
Μετά την ολοκλήρωση της υλοποίησης, πραγματοποιήθηκαν μετρήσεις στα μηχανήματα του εργαστηρίου υπολογιστικών συστημάτων, σε συστοιχία οκτώ επεξεργαστικών κόμβων. Κάθε κόμβος αποτελείται από δύο τετραπύρηνους επεξεργαστές Intel(R) Xeon(R) CPU E5335, 2.00GHz, με 4 MB L1 cache. Πραγματοποιήθηκαν μετρήσεις για 2, 4, 8, 16, 32 και 64 διεργασίες για τα δίκτυα των 118, 1180 και 2360 ζυγών. Στα επόμενα τρία σχήματα παρουσιάζονται τα διαγράμματα επιτάχυνσης σε σχέση με την σειριακή υλοποίηση, για κάθε ένα από τα τρία αρχεία αντίστοιχα. Στο χρόνο υπολογισμού συμπεριλαμβάνεται μόνο ο χρόνος εκτέλεσης των επαναλήψεων μέχρι τη σύγκλιση του αλγόριθμου, και όχι οι χρόνοι αρχικοποιήσεων, διαβάσματος των αρχείων και αρχικής επικοινωνίας-αποστολής δεδομένων.



Σχήμα 4.5: Επιτάχυνση παράλληλου WLS για το αρχείο των 118 ζυγών.



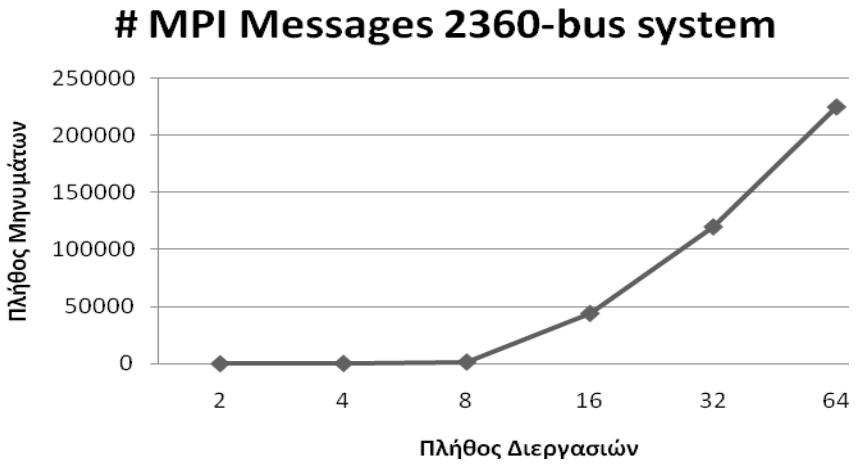
Σχήμα 4.6: Επιτάχυνση παράλληλου WLS για το αρχείο των 1180 ζυγών.



Σχήμα 4.7: Επιτάχυνση παράλληλου WLS για το αρχείο των 2360 ζυγών.

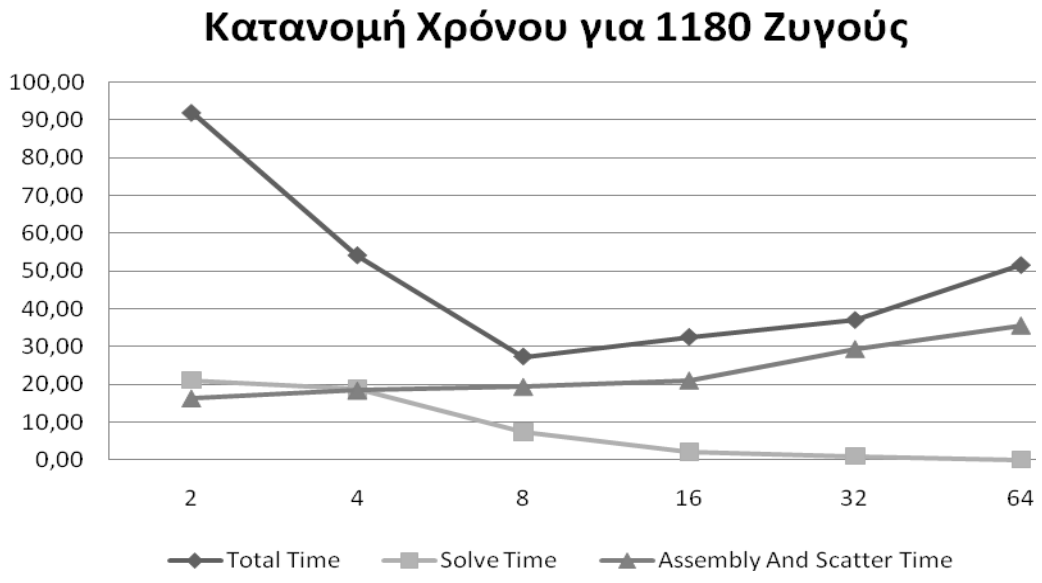
Να σημειώσουμε αρχικά, ότι όλες οι μετρήσεις πραγματοποιήθηκαν με την επιλογή –bynode, δηλαδή αναθέτοντας τις διεργασίες ανά κόμβο. Παρατηρώντας τα τρία διαγράμματα της επιτάχυνσης του παράλληλου προγράμματος σε σχέση με το σειριακό, βλέπουμε αρχικά βελτίωση στην επίδοση σε σχέση με το σειριακό πρόγραμμα. Το παράλληλο πρόγραμμα εκτελείται έως και 6 φορές πιο γρήγορα, στην περίπτωση των 1180 ζυγών για οκτώ διεργασίες. Συνολικά, φαίνεται να έχουμε επιτύχει επιτάχυνση, η οποία ωστόσο, δεν βελτιώνεται γραμμικά με την αύξηση των διεργασιών, όπως θα θέλαμε, αλλά αντιθέτως παρατηρούμε πτώση της επίδοσης όταν οι διεργασίες ξεπεράσουν τις 16. Παρατηρούμε, επίσης, ότι η μεγαλύτερη επιτάχυνση για τα δύο μεγάλα αρχεία, σημειώνεται κατά την εκτέλεση με οκτώ διεργασίες, δηλαδή όταν έχει ανατεθεί μία διεργασία σε κάθε επεξεργαστικό κόμβο του συστήματος. Το φαινόμενο αυτό δικαιολογείται αν σκεφτούμε πως όταν οι διεργασίες ξεπεράσουν τον αριθμό των κόμβων, θα πρέπει να ανατεθούν περισσότερες σε κάθε έναν από αυτούς, με αποτέλεσμα την εμφάνιση ανταγωνισμού μεταξύ τους, για τους κοινούς προς εκμετάλλευση πόρους του κόμβου. Ακόμη ένα

γεγονός που μπορεί να δικαιολογήσει την πτώση της επιτάχυνσης, καθώς αυξάνεται ο αριθμός των διεργασιών, γίνεται εμφανές παρατηρώντας το ακόλουθο διάγραμμα:

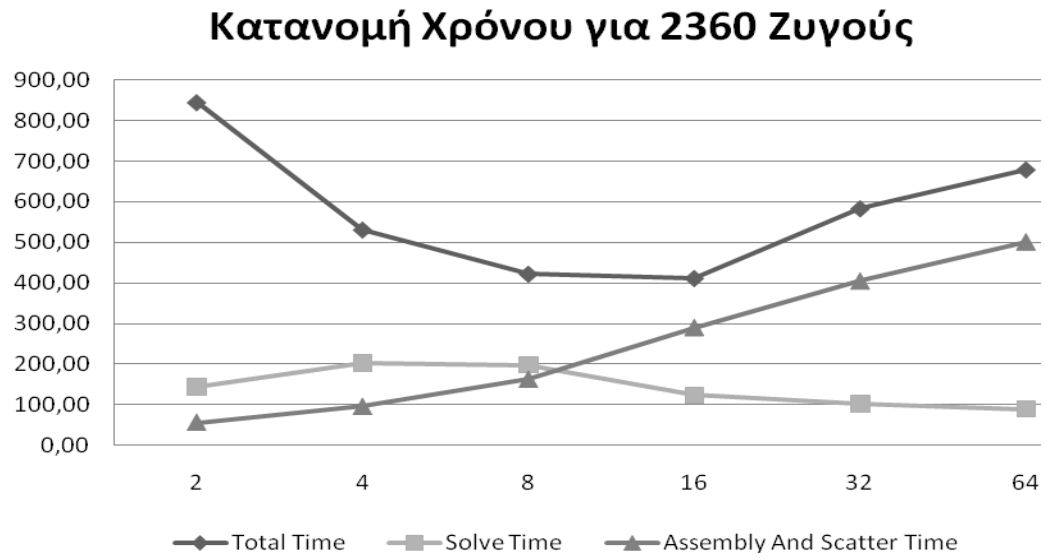


Σχήμα 4.8: Πλήθος MPI μηνυμάτων σε σχέση με τον αριθμό των διεργασιών για το αρχείο των 2360 ζυγών.

Το παραπάνω σχήμα απεικονίζει το πλήθος των μηνυμάτων MPI που ανταλλάσσονται μεταξύ των διεργασιών σε σχέση με το πλήθος των διεργασιών, κατά την εκτέλεση του παράλληλου WLS αλγόριθμου, για το δίκτυο των 2360 ζυγών. Τα διαγράμματα που αφορούν στα άλλα δύο δίκτυα έχουν πανομοιότυπη μορφή, οπότε και δεν κρίθηκε σκόπιμη η παράθεσή τους. Παρατηρούμε πως το πλήθος των μηνυμάτων αυξάνεται σχεδόν εκθετικά με το πλήθος των διεργασιών, γεγονός που σίγουρα επηρεάζει την επίδοση. Τα επόμενα δύο διαγράμματα παρουσιάζουν την κατανομή του χρόνου εκτέλεσης του αλγόριθμου, σε χρόνο επίλυσης και χρόνο επικοινωνίας, και υποστηρίζουν απόλυτα την υπόθεσή μας αυτή:



Σχήμα 4.9: Κατανομή Χρόνου Εκτέλεσης του παράλληλου WLS για το αρχείο των 1180 ζυγών.



Σχήμα 4.10: Κατανομή Χρόνου Εκτέλεσης του παράλληλου WLS για το αρχείο των 2360 ζυγών.

Ο χρόνος επικοινωνίας που αναφέραμε πιο πάνω, αποτελείται από το χρόνο που χρειάζεται η PETSc για τη συναρμολόγηση των κατανεμημένων δομών της και ουσιαστικά, είναι «κρυμμένος» μέσα στις συλλογικές ρουτίνες της. Ωστόσο, η PETSc παρέχει έναν εύκολο τρόπο στον προγραμματιστή, για την εξαγωγή ενός λεπτομερούς αρχείου με πληροφορίες όπως ο χρόνος αυτός. Από τα διαγράμματα γίνεται φανερό ότι ο χρόνος για την κατανομή και την επανασυναρμολόγηση των δομών σε κάθε επανάληψη, γίνεται σημαντικά μεγαλύτερος από το χρόνο επίλυσης, για 16 και ή περισσότερες διεργασίες. Αντίθετα, αξίζει, να παρατηρήσουμε ότι ο χρόνος επίλυσης μειώνεται με την αύξηση των διεργασιών, γεγονός που φανερώνει την αποδοτική υλοποίηση των ρουτινών επίλυσης της βιβλιοθήκης.

4.3 Υλοποίηση του αλγόριθμου Conjugate Gradient με Jacobi Preconditioner

Η πρώτη εφαρμογή που εξετάσαμε, αφορούσε υλοποίηση χρησιμοποιώντας προγραμματιστικά εργαλεία, τα οποία μας παρείχαν υλοποιημένες ρουτίνες επίλυσης. Για τη δεύτερη εφαρμογή, θεωρήθηκε σκόπιμο να μελετηθεί μεμονωμένα κάποιος αλγόριθμος επίλυσης γραμμικών συστημάτων, να υλοποιηθεί εξ' αρχής και στη συνέχεια να παραλληλοποιηθεί. Ο αλγόριθμος που επιλέχθηκε είναι η μέθοδος Conjugate Gradient. Καθώς η σύγκλιση της απλής μεθόδου είναι ιδιαίτερα αργή στη μέση περίπτωση (απαιτούνται τόσα βήματα όσο είναι το μέγεθος του συστήματος), υλοποιήθηκε η μέθοδος με Jacobi preconditioner. Ο Jacobi preconditioner, έστω C, δημιουργείται από τα διαγώνια στοιχεία του πίνακα A του συστήματος, δηλαδή:

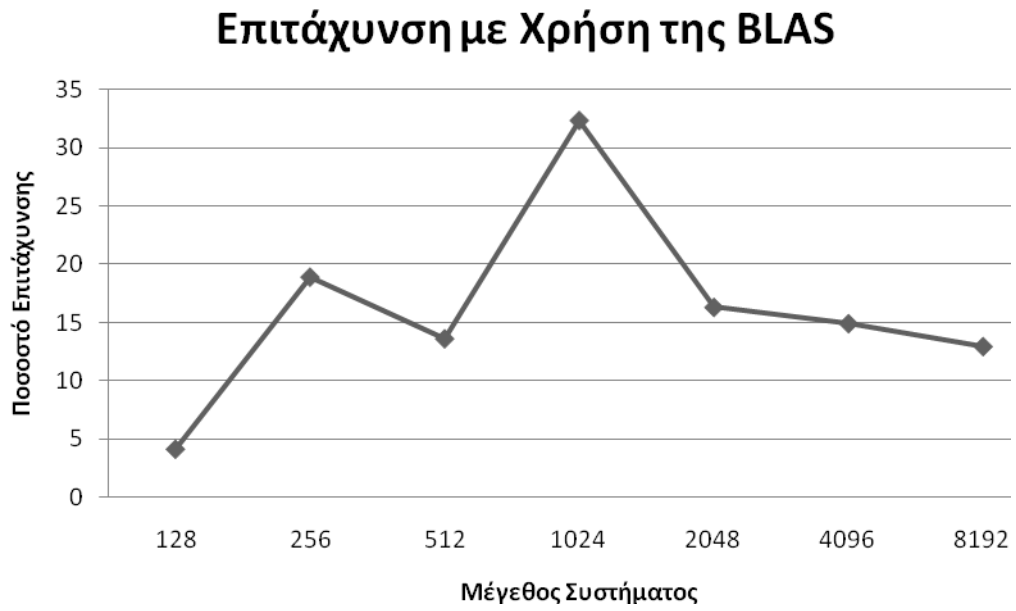
$$C_{ij} = \begin{cases} A_{ii}, & \alpha\nu\ i = j \\ 0, & \alpha\nu\ i \neq j \end{cases}$$

Ο Jacobi preconditioner είναι πολύ εύκολο και γρήγορο να υλοποιηθεί και έχει ιδιαίτερα χαμηλές απαιτήσεις σε μνήμη.

4.3.1 Σειριακή Υλοποίηση

Πραγματοποιήθηκαν δύο σειριακές εκδόσεις του αλγόριθμου, μία χρησιμοποιώντας «καθαρή» γλώσσα C και τις δικές μας ρουτίνες γραμμικής άλγεβρας και μία χρησιμοποιώντας ρουτίνες από τη βιβλιοθήκη BLAS. Σκοπός μας ήταν να γίνει μία σύγκριση στην επίδοση των δύο προγραμμάτων, ώστε να διαπιστώσουμε αν υπάρχουν οφέλη από τη χρήση της BLAS.
Από την περιγραφή του αλγόριθμου που παρατίθεται στο $2^{ο}$ κεφάλαιο της παρούσας εργασίας, μπορούμε να προσδιορίσουμε τις πράξεις - λειτουργίες που απαιτούνται για την υλοποίηση: πολλαπλασιασμός πίνακα με διάνυσμα, εσωτερικό γινόμενο, γραμμικό άθροισμα διανυσμάτων. Καθεμία από τις παραπάνω πράξεις υλοποιήθηκε ως ξεχωριστή ρουτίνα. Ακόμη, δημιουργήθηκε μία ρουτίνα για τη δημιουργία του preconditioner και μία για την εφαρμογή του preconditioner στο σύστημα. Τέλος, υλοποιήθηκαν και ρουτίνες δημιουργίας τυχαίων γραμμικών συστημάτων προς επίλυση.

Στην υλοποίηση που έγινε με τη χρήση της BLAS, οι δικές μας ρουτίνες πράξεων γραμμικής άλγεβρας (εσωτερικό γινόμενο, πολλαπλασιασμός πίνακα με διάνυσμα και γραμμικό άθροισμα διανυσμάτων) αντικαταστάθηκαν από ρουτίνες της βιβλιοθήκης. Στο Σχήμα 4.11 που ακολουθεί, παρουσιάζεται το διάγραμμα της επιτάχυνσης που επιτεύχθηκε με τον τρόπο αυτό:



Σχήμα 4.11: Επιτάχυνση σειριακής υλοποίησης Conjugate Gradient σε σχέση με την υλοποίηση BLAS. Ο οριζόντιος άξονας αντιστοιχεί στο μέγεθος του προς επίλυση συστήματος.

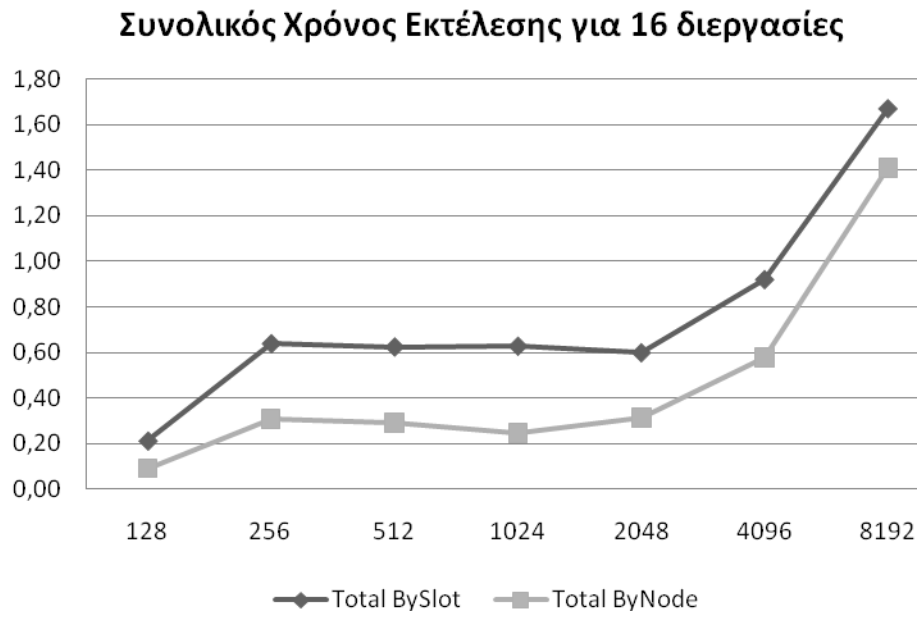
Δημιουργήθηκαν επτά διαφορετικά τυχαία συστήματα προς επίλυση, αυξανόμενου μεγέθους, από 128 έως 8192 αγνώστων. Βλέπουμε πως η χρήση συναρτήσεων της βιβλιοθήκης BLAS επιτάχυνε το πρόγραμμά μας για όλα τα μεγέθη συστημάτων, ενώ η μεγαλύτερη επιτάχυνση σημειώνεται για το σύστημα των 1024 αγνώστων και είναι της τάξης του 30%. Για τα υπόλοιπα μεγέθη συστημάτων, η επιτάχυνση είναι της τάξης του 15-20%.

4.3.2 Παράλληλη Υλοποίηση

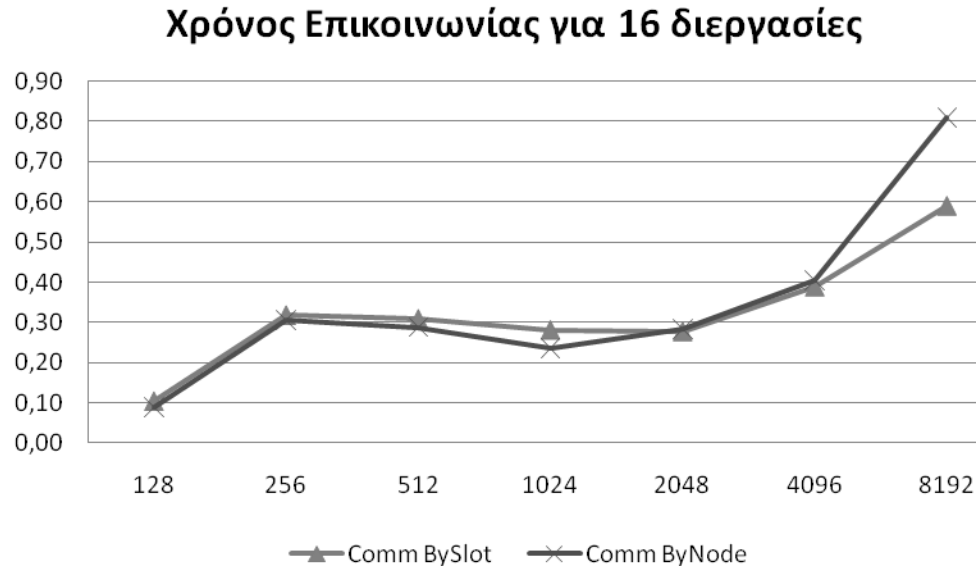
Για την υλοποίηση της παράλληλης μεθόδου Conjugate Gradient, χρειάστηκε να παραλληλοποιήσουμε τις τρεις βασικές πράξεις, εσωτερικό γινόμενο, γραμμικό άθροισμα και πολλαπλασιασμό πίνακα επί διάνυσμα. Ο πίνακας του συστήματος μοιράστηκε στις διεργασίες κατά γραμμές, αντίστοιχα και το διάνυσμα του δεξιού μέλους. Με την κατανομή αυτή, το γραμμικό άθροισμα υλοποιήθηκε χωρίς να χρειαστεί κάποια επικοινωνία – κάθε διεργασία υπολογίζει και χρησιμοποιεί το μερικό της αποτέλεσμα. Όσον αφορά στο εσωτερικό γινόμενο, η ρουτίνα τροποποιήθηκε ώστε κάθε διεργασία να υπολογίζει ένα μέρος του αποτελέσματος με τα στοιχεία που της ανήκουν. Στη συνέχεια, χρησιμοποιείται η κατάλληλη ρουτίνα επικοινωνίας MPI ώστε να αθροιστούν τα μερικά γινόμενα και πάρει κάθε διεργασία το συνολικό αποτέλεσμα. Αντίστοιχα, για τον πολλαπλασιασμό πίνακα επί διάνυσμα, χρησιμοποιήθηκε η ρουτίνα broadcast του MPI ώστε να διανεμηθεί ολόκληρο το διάνυσμα του υπολογισμού στις διεργασίες. Οπότε, σε κάθε επανάληψη της μεθόδου, απαιτείται μία λειτουργία επικοινωνίας για κάθε ένα από τα τρία εσωτερικά γινόμενα που υπολογίζονται και μία λειτουργία επικοινωνίας για τη λήψη του αποτελέσματος του πολλαπλασιασμού.

4.3.3 Μετρήσεις

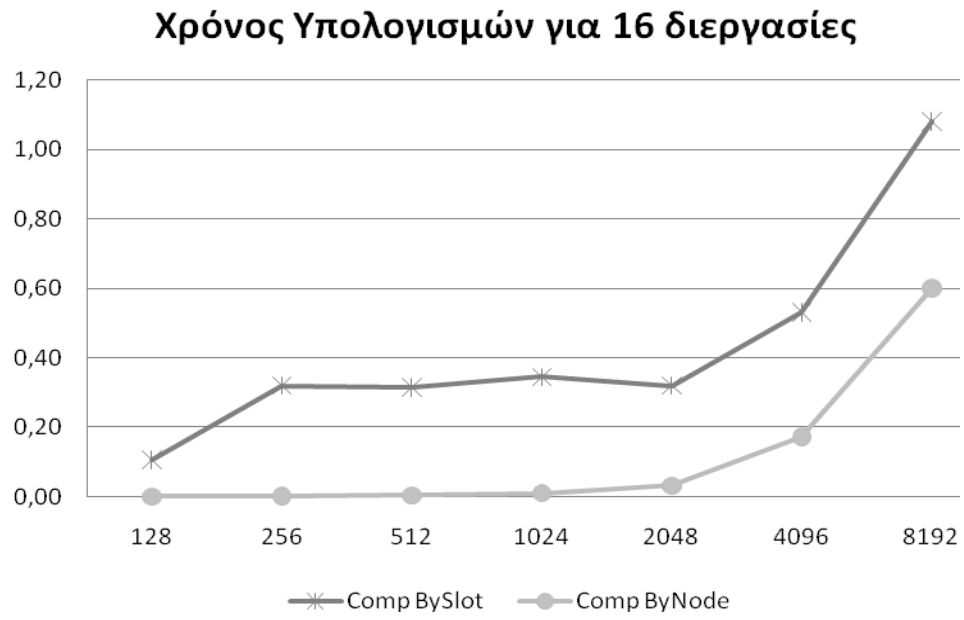
Όπως και στην περίπτωση του αλγόριθμου εκτίμησης κατάστασης, οι μετρήσεις πραγματοποιήθηκαν σε οκτώ επεξεργαστικούς κόμβους με οκτώ πυρήνες στον καθένα. Η μελέτη μας επικεντρώθηκε κυρίως στην κατανομή του χρόνου εκτέλεσης σε χρόνο για υπολογισμούς και χρόνο για επικοινωνία. Πραγματοποιήθηκαν δύο σύνολα μετρήσεων, θέτοντας διαφορετικές επιλογές στον τρόπο κατανομής των διεργασιών στους πυρήνες. Με την επιλογή –bynode ορίζουμε την κατανομή των διεργασιών ανά κόμβο, ενώ με την επιλογή –byslot ορίζουμε την κατανομή ανά «slot». Για παράδειγμα, αν εκτελεστεί το πρόγραμμα με 16 διεργασίες –bynode, τότε σε κάθε επεξεργαστικό κόμβο θα ανατεθούν 2 διεργασίες. Αντίστοιχα, με 16 διεργασίες –byslot, θα ανατεθούν από 8 διεργασίες σε 2 επεξεργαστικούς κόμβους. Στα τρία σχήματα που ακολουθούν, παρουσιάζονται διαγράμματα για τα δύο αυτά σύνολα μετρήσεων, με 16 διεργασίες. Στον οριζόντιο άξονα αντιστοιχεί το μέγεθος του συστήματος και στον κάθετο άξονα απεικονίζεται ο χρόνος εκτέλεσης σε sec. Τα διαγράμματα αφορούν στο συνολικό χρόνο εκτέλεσης, στον χρόνο επικοινωνίας και στο χρόνο υπολογισμών αντίστοιχα:



Σχήμα 4.12: Σύγκριση Συνολικού Χρόνου Εκτέλεσης --bynode και –byslot.



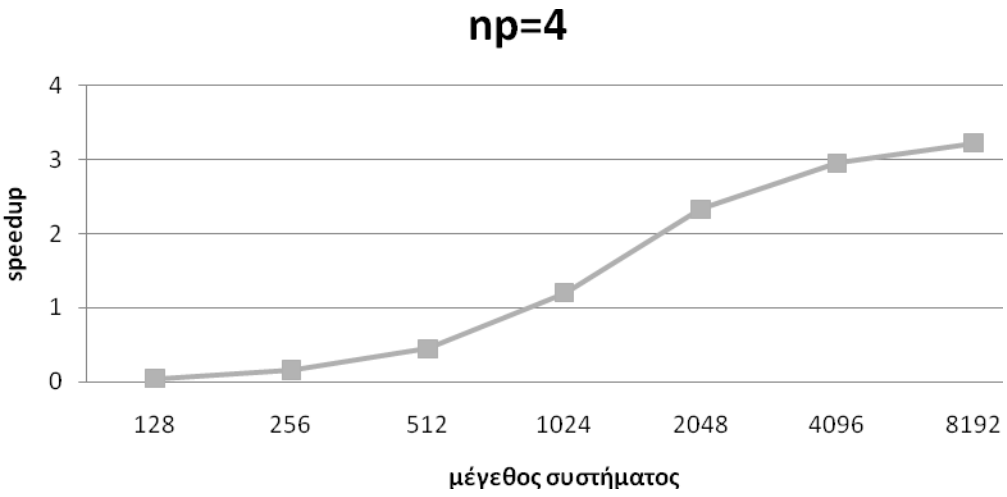
Σχήμα 4.13: Σύγκριση Χρόνου Επικοινωνίας --bynode και –byslot.



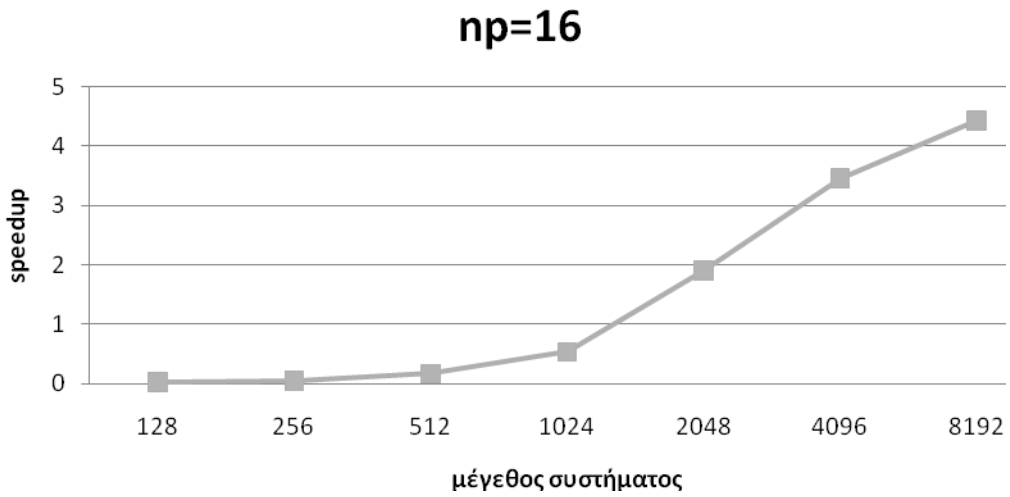
Σχήμα 4.14: Σύγκριση Χρόνου Υπολογισμών --bynode και –byslot.

Παρατηρώντας κατ' αρχήν το διάγραμμα του συνολικού χρόνου εκτέλεσης, βλέπουμε ότι υπάρχει σημαντική διαφορά, με την επιλογή ανά κόμβο να είναι εμφανώς πιο αποδοτική. Τα επόμενα δύο διαγράμματα κάνουν φανερό το λόγο που συμβαίνει αυτό, καθώς ο χρόνος υπολογισμών στην περίπτωση –byslot είναι σημαντικά υψηλότερος. Το γεγονός αυτό οφείλεται προφανώς στην κατανομή περισσότερων διεργασιών σε έναν κόμβο στην περίπτωση αυτή. Οκτώ διεργασίες εκτελούνται ταυτόχρονα στον ίδιο επεξεργαστικό κόμβο και «ανταγωνίζονται» μεταξύ τους για την εκμετάλλευση των ίδιων πόρων, ενώ στην περίπτωση –bynode οι ίδιοι πόροι είναι διαθέσιμοι σε δύο μόνο διεργασίες.

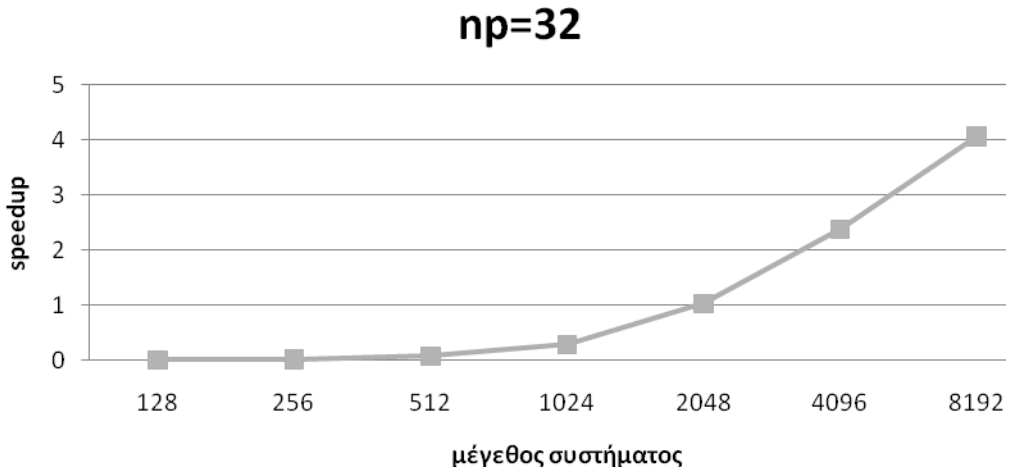
Με βάση τις παραπάνω διαπιστώσεις, τα διαγράμματα μετρήσεων που ακολουθούν αφορούν σε μετρήσεις που πραγματοποιήθηκαν με την επιλογή –bynode. Αρχικά, παρουσιάζουμε διαγράμματα επιτάχυνσης του παράλληλου κώδικα σε σχέση με το σειριακό (BLAS υλοποίηση) για 4, 16, 32 και 64 διεργασίες:



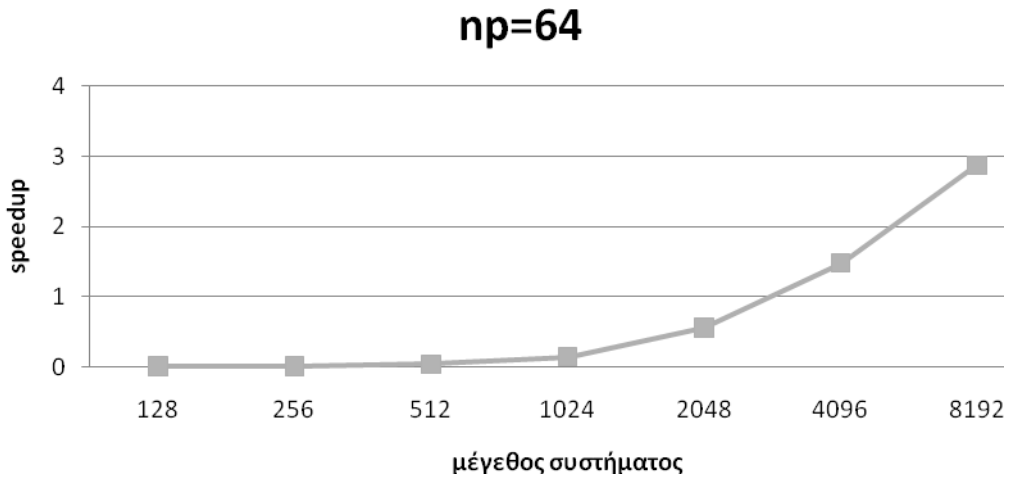
Σχήμα 4.15: Επιτάχυνση για 4 διεργασίες



Σχήμα 4.16: Επιτάχυνση για 16 διεργασίες

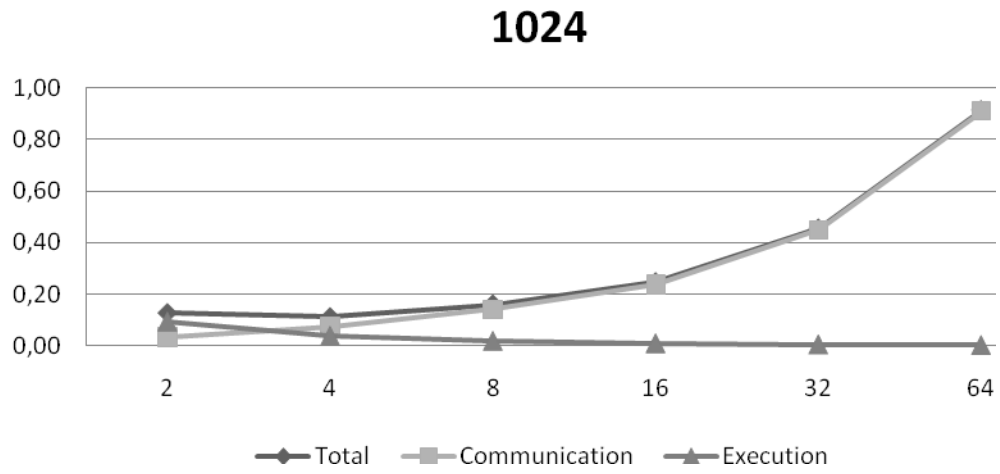


Σχήμα 4.17: Επιτάχυνση για 32 διεργασίες

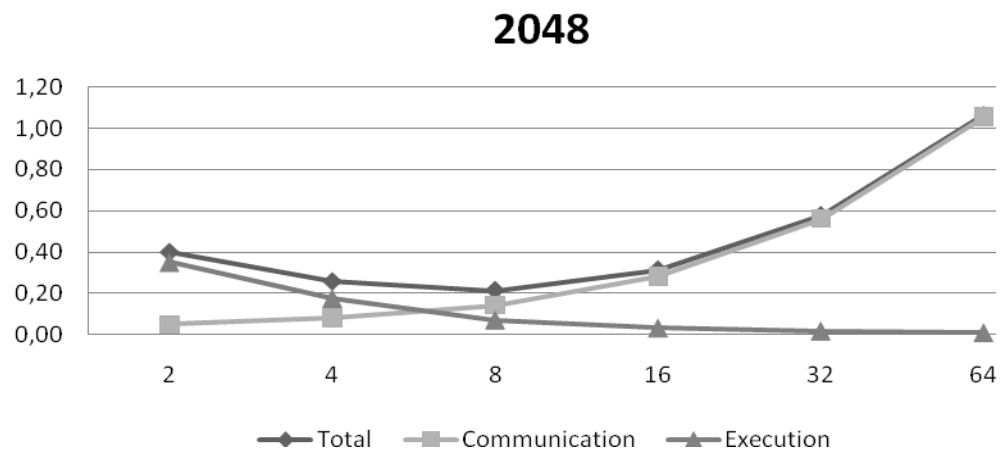


Σχήμα 4.18: Επιτάχυνση για 64 διεργασίες

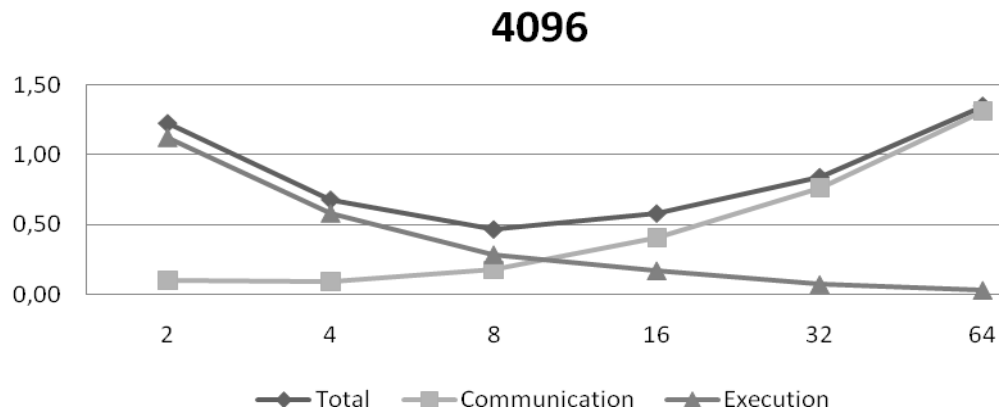
Παρατηρούμε ότι για 4 και 16 διεργασίες, καλύτερη επίδοση από το σειριακό παρουσιάζεται για το σύστημα των 2048 αγνώστων και πάνω, για 32 διεργασίες παρατηρείται επιτάχυνση για τα αρχεία 4096 και 8192 αγνώστων, ενώ για 64 διεργασίες, μόνο για το μεγαλύτερο σύστημα. Για τα μικρά συστήματα, παρατηρούμε ακόμη και χειρότερη επίδοση από το σειριακό. Αυτό οφείλεται στο γεγονός ότι το μέγεθος του προβλήματος είναι σημαντικά μικρό ώστε να φανεί κάποια βελτίωση στην επίδοση από την παραλληλοποίηση, ενώ ταυτόχρονα η επικοινωνία που περιέχεται στο παράλληλο πρόγραμμα επιβαρύνει το χρόνο εκτέλεσης. Η μεγαλύτερη βελτίωση επιτυγχάνεται για 16 διεργασίες και 8192 αγνώστους, ενώ αυξάνοντας τις διεργασίες δεν παίρνουμε την επιτάχυνση που θα επιθυμούσαμε. Για παράδειγμα, με 16 διεργασίες, στο σύστημα των 4096 αγνώστων, επιτύχαμε 3,5 φορές ταχύτερη εκτέλεση. Διπλασιάζοντας τις διεργασίες, το παράλληλο πρόγραμμα ήταν 2,5 φορές ταχύτερο, ενώ για 64 διεργασίες, μόλις 1,5 φορά. Συμπεραίνουμε, λοιπόν, πως αυξάνοντας τον αριθμό των διεργασιών, υπάρχει κάποιος παράγοντας ο οποίος επηρεάζει αρνητικά την επίδοση, ο οποίος, όπως μαρτυρούν τα διαγράμματα που ακολουθούν, είναι η επικοινωνία. Στα σχήματα 4.19 έως 4.22, βλέπουμε την κατανομή του χρόνου εκτέλεσης σε χρόνο επικοινωνίας και χρόνο υπολογισμών για τα συστήματα των 1024, 2048, 4096 και 8192 αγνώστων. Τα διαγράμματα που αφορούν στα μικρότερα συστήματα παραλείπονται, καθώς το μέγεθος τους δεν θεωρείται αντιπροσωπευτικό για την εξαγωγή ασφαλών συμπερασμάτων.



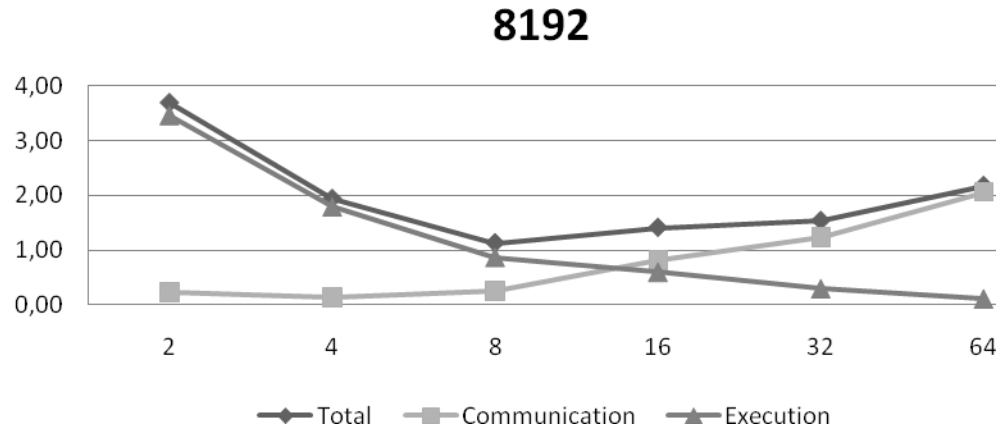
Σχήμα 4.19: Κατανομή χρόνου επεξεργασίας για 1024 αγνώστους.



Σχήμα 4.20: Κατανομή χρόνου επεξεργασίας για 2048 αγνώστους.



Σχήμα 4.21: Κατανομή χρόνου επεξεργασίας για 4096 αγνώστους.

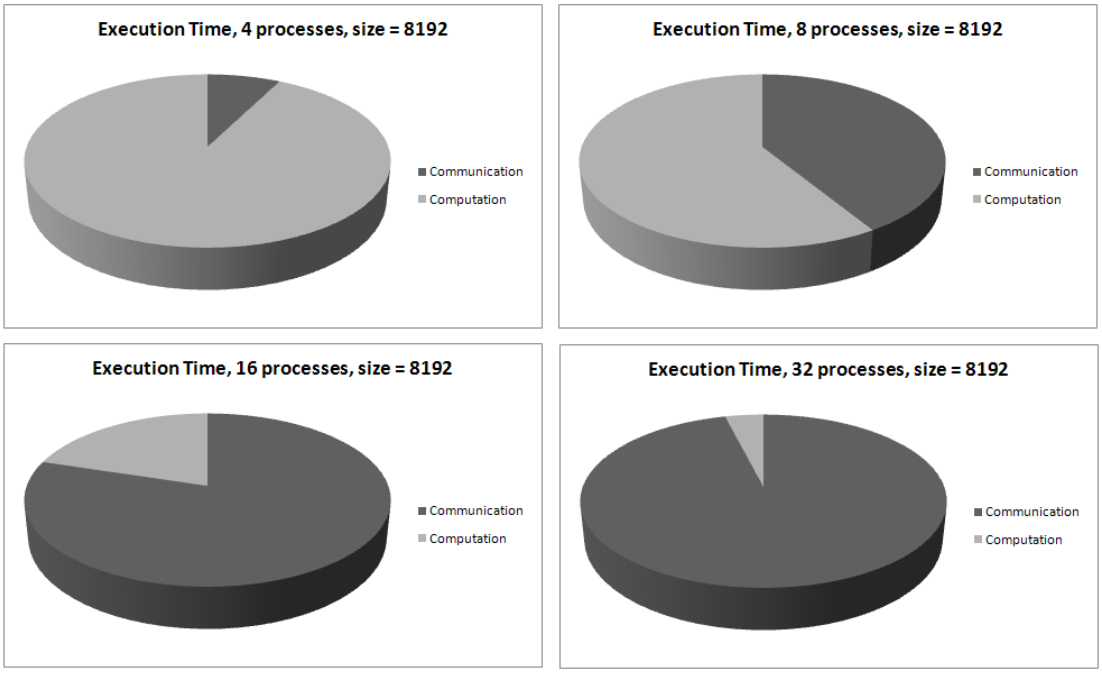


Σχήμα 4.22: Κατανομή χρόνου επεξεργασίας για 8192 αγνώστους.

Από τα παραπάνω σχήματα βλέπουμε καθαρά πως ο περιοριστικός παράγοντας της επιτάχυνσης είναι η επικοινωνία. Μάλιστα, για μικρότερο μέγεθος συστήματος, η καμπύλη του χρόνου εκτέλεσης ακολουθεί αυτή του χρόνου επικοινωνίας, ενώ ο χρόνος υπολογισμών είναι μηδαμινός. Όσο το μέγεθος του

συστήματος αυξάνεται, οι υπολογισμοί καταλαμβάνουν περισσότερο από το συνολικό χρόνο. Βλέπουμε, επίσης, ότι η βέλτιστη επίδοση επιτυγχάνεται για 8 διεργασίες, όταν δηλαδή, έχουμε αναθέσει μία διεργασία σε κάθε κόμβο. Για τα μεγάλα συστήματα, ο χρόνος επικοινωνίας επικρατεί όταν αυξηθεί αρκετά ο αριθμός των διεργασιών.

Στο σχήμα που ακολουθεί παρουσιάζεται η κατανομή του χρόνου εκτέλεσης για το σύστημα των 8192 αγνώστων και 4, 8, 16, 32 διεργασίες. Με κόκκινο χρώμα παρουσιάζεται ο χρόνος υπολογισμών και με μπλε ο χρόνος επικοινωνίας. Βλέπουμε πως για μικρό αριθμό διεργασιών, το μεγαλύτερο ποσοστό του χρόνου αναλώνεται σε υπολογισμούς, ενώ ο χρόνος που απαιτείται για επικοινωνία είναι αμελητέος. Όσο ο αριθμός των διεργασιών αυξάνεται, η εικόνα αντιστρέφεται και ο χρόνος επικοινωνίας αυξάνεται δραματικά σε σχέση με το χρόνο υπολογισμών.



Σχήμα 4.23: Κατανομή του χρόνου εκτέλεσης σε επικοινωνία (σκούρο γκρι) και υπολογισμους (ανοιχτό γκρι), για 8192 αγνώστους και 4, 8, 16 και 32 διεργασίες.

4.3.4 Συμπεράσματα

Έχοντας εξετάσει και τις δύο εφαρμογές, τον αλγόριθμο εκτίμησης κατάστασης και τη μέθοδο Conjugate Gradient, καταλήγουμε στο συμπέρασμα ότι σε μια παράλληλη εφαρμογή, η επικοινωνία παίζει σημαντικότατο ρόλο και η επίδρασή της στην επίδοση θα πρέπει να μελετάται με προσοχή. Και στις δύο υλοποιήσεις που πραγματοποιήσαμε και εξετάσαμε, η επικοινωνία αποτέλεσε τον περιοριστικό παράγοντα της επίδοσης. Ωστόσο, έγινε φανερό, πως η επίδραση της επικοινωνίας εξαρτάται σε μεγάλο βαθμό από το μέγεθος του προβλήματος και από τη σχέση της με το φορτίο των υπολογισμών. Όσο μεγαλύτερο είναι το μέγεθος ενός προβλήματος, τόσο αυξάνεται το φορτίο των υπολογισμών, με αποτέλεσμα να μειώνεται η διαφορά στο χρόνο επικοινωνίας και στο χρόνο υπολογισμών. Θα πρέπει, επίσης, να λάβουμε υπόψη το γεγονός ότι προγραμματιστικά εργαλεία όπως η PETSc, είναι σχεδιασμένα για εφαρμογές μεγάλης κλίμακας, όπως επίλυση συστημάτων με εκατομμύρια αγνώστους.

Γενικότερα, θα μπορούσαμε να σημειώσουμε ότι από τη μελέτη μας, προκύπτει ότι το όφελος που λαμβάνουμε από την παραλληλοποίηση ενός

αλγόριθμου, εξαρτάται τόσο από τη φύση του ίδιου του αλγόριθμου όσο και από το μέγεθος του προβλήματος που εξετάζεται. Σκοπός μας πρέπει να είναι η παραλληλοποίηση εφαρμογών οι οποίες απαιτούν πολύ χρόνο εκτέλεσης, ώστε η βελτίωση στην επίδοση που θα πετύχουμε με την παραλληλοποίηση, να αντισταθμίζει το κόστος υλοποίησης.

Κεφάλαιο 5ο: Ζητήματα υλοποίησης και συμπεράσματα

5.1 Εισαγωγή

Στο παρόν κεφάλαιο θα συζητήσουμε θέματα που αφορούν στην υλοποίηση των εφαρμογών, όπως δυσκολίες και διλλήματα που αντιμετωπίσαμε κατά την παραλληλοποίηση των αλγόριθμων. Θα παρουσιάσουμε εμπειρίες που αποκομίσαμε από τη χρήση των προγραμματιστικών εργαλείων που παρουσιάστηκαν στο κεφάλαιο 3, ενώ θα σχολιάσουμε, επίσης, την επίδραση της τοπολογίας του συστήματος όπου πραγματοποιήθηκαν οι μετρήσεις, στην επίδοση των υλοποιήσεων

5.2 Η χρήση της BLAS ως προγραμματιστικό εργαλείο

Όπως φάνηκε και από την περιγραφή της στο τρίτο κεφάλαιο, η βιβλιοθήκη BLAS, αν και ιδιαίτερα λειτουργική και αποδοτική, είναι, επίσης, απλή στη χρήση. Η εκμάθηση της βιβλιοθήκης γίνεται γρήγορα, ενώ είναι αρκετά βοηθητικό για τον προγραμματιστή, το γεγονός ότι τα ονόματα των συναρτήσεων δηλώνουν τη λειτουργία που επιτελούν. Η εγκατάστασή της είναι εύκολη και γίνεται σε λίγα και απλά βήματα. Αν και υλοποιημένη σε FORTRAN, παρέχεται ξεκάθαρο C interface, απλώς κάνοντας «include» το κατάλληλο header αρχείο. Έτσι, οι συναρτήσεις της μπορούν να χρησιμοποιηθούν όπως οποιεσδήποτε άλλες C συναρτήσεις, χωρίς κάποια ιδιαίτερη προγραμματιστική απαίτηση.

5.3 Η χρήση της PETSc ως προγραμματιστικό εργαλείο

Σε αντίθεση με τη BLAS, η PETSc είναι μια «αχανής» βιβλιοθήκη, αλλά προσφέρει, επίσης, πολλές περισσότερες δυνατότητες. Η διαδικασία εγκατάστασής της είναι αρκετά πολύπλοκη, και μπορεί να φανεί ιδιαίτερα δύσκολη σε κάποιον αρχάριο σε σύστημα linux. Κατά την εγκατάσταση μπορούν να τεθούν αρκετές επιλογές παραμετροποίησης, οι περισσότερες από τις οποίες εξαρτώνται από το σύστημα στο οποίο γίνεται η εγκατάσταση, και πρέπει να μελετηθούν με προσοχή. Λανθασμένη τιμή στις επιλογές αυτές απαιτεί επαναδιαμόρφωση της βιβλιοθήκης, μια λειτουργία αρκετά χρονοβόρα.

Για την εκμάθησή της απαιτείται αρκετός χρόνος και συμμόρφωση με αυστηρό προγραμματιστικό στυλ που ακολουθεί συγκεκριμένους κανόνες. Για τη χρήση της, ο προγραμματιστής, χρειάζεται να ανατρέχει συχνά στο εγχειρίδιο των συναρτήσεων, το οποίο, όμως, είναι πολύ καλά οργανωμένο με συνδέσμους σε κώδικα και αρκετά κατατοπιστικά παραδείγματα για τις περισσότερες από τις ρουτίνες. Οι ρουτίνες στο εγχειρίδιο είναι ομαδοποιημένες ανά τύπο αντικειμένου (διάνυσμα, πίνακα, KSP, κτλ) ώστε να διευκολύνεται η αναζήτηση. Παρέχονται, επίσης, πληροφορίες επίδοσης των συναρτήσεων και συμβουλές χρήσης.

5.3.1 Η χρήση PETSc στην παραλληλοποίηση του WLS

Συγκεκριμένα για τον αλγόριθμο εκτίμησης κατάστασης, η χρήση της PETSc ήταν βοηθητική σε πολλές περιπτώσεις, αλλά μας έθεσε και αρκετούς περιορισμούς στην ελευθερία υλοποίησης.

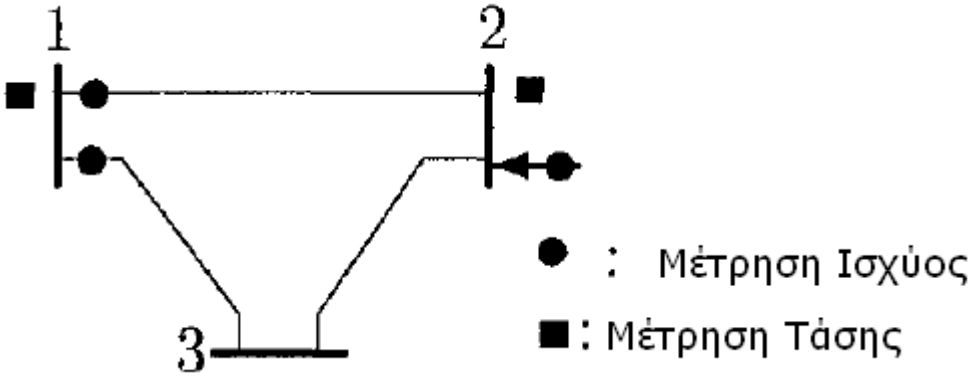
Ως πρώτο και σημαντικότατο πλεονέκτημα της χρήσης της PETSc, είναι η παροχή έτοιμων δομών αραιών πινάκων και ρουτινών πράξεων με αραιούς πίνακες. Η εξ' αρχής υλοποίηση δομών αποθήκευσης αραιών πινάκων είναι μια επίπονη διαδικασία και οι πράξεις μεταξύ πινάκων αποθηκευμένων με κάποια αραιή μορφή απαιτούν ιδιαίτερη διαχείριση. Η PETSc «κρύβει» τη μορφή αποθήκευσης από τον προγραμματιστή, ο οποίος εκτελεί τις πράξεις σε υψηλότερο επίπεδο.

Άλλο ένα πλεονέκτημα είναι ότι η επικοινωνία «κρύβεται» μέσα στις συλλογικές ρουτίνες και τις κατανεμημένες δομές. Χρησιμοποιώντας C και ρουτίνες MPI, για να υλοποιήσουμε π.χ. έναν πολλαπλασιασμό πίνακα επί διάνυσμα, θα πρέπει να μοιράσουμε τον πίνακα με MPI_Send() – MPI_Recv(), να κάνουμε broadcast το διάνυσμα, να κάνει η κάθε διεργασία τις πράξεις για το κομμάτι του πίνακα που της αντιστοιχεί, και στη συνέχεια, να «στείλει» πίσω το μερικό αποτέλεσμά της ώστε να πάρουμε το τελικό αποτέλεσμα. Με την PETSc, αρκεί να ορίσουμε τον πίνακα και το διάνυσμα ως κατανεμημένες δομές, δίνοντας ως παράμετρο μόνο τον communicator, και στη συνέχεια να καλέσουμε τη ρουτίνα πολλαπλασιασμού, και τις ρουτίνες Assembly Begin-End.

Ιδιαίτερα βοηθητικές είναι, επίσης, οι επιλογές παρακολούθησης εκτέλεσης που παρέχει η PETSc, όπως η επιλογή «-log_summary». Με την επιλογή αυτή, μπορούμε να δούμε τη δημιουργία και την αποδέσμευση αντικειμένων, να παρακολουθήσουμε τη δέσμευση και απελευθέρωση μνήμης, να καταγράψουμε ποιες ρουτίνες κλήθηκαν, πόσες φορές και πώς κατανέμεται ο χρόνος σε αυτές, να μετρήσουμε πόσα MPI μηνύματα στάλθηκαν και το μέγεθός αυτών.

Οι περιορισμοί που συναντήσαμε κατά την υλοποίηση του παράλληλου προγράμματος WLS, αφορούν περισσότερο στην κατανομή των δεδομένων στις διεργασίες. Κατ' αρχήν, όλες οι συλλογικές ρουτίνες τις PETSc προϋποθέτουν την ύπαρξη κατανεμημένων δομών δεδομένων (πινάκων και διανυσμάτων). Όμως, ένα κατανεμημένο αντικείμενο της PETSc μοιράζεται με συγκεκριμένο τρόπο σε διεργασίες. Για παράδειγμα, ένας πίνακας τύπου MatMPIAIJ, κατά τη δημιουργία του, μοιράζεται στις διαθέσιμες διεργασίες κατά γραμμές. Ο προγραμματιστής δεν έχει καμία ελευθερία στην κατανομή του πίνακα και μπορεί μόνο να μάθει ποιες γραμμές έχουν ανατεθεί σε ποιες διεργασίες, ώστε να χρησιμοποιήσει την πληροφορία αυτή σε άλλους υπολογισμούς. Κάθε διεργασία μπορεί να έχει πρόσβαση στα δεδομένα των άλλων, ωστόσο η προσπέλαση σε στοιχείο άλλης διεργασίας προσθέτει overhead κατά τη λειτουργία της τελικής συναρμολόγησης της δομής.

Ο πρώτος πίνακας που έπρεπε να κατανεμηθεί σε διεργασίες, ήταν η μήτρα H, η οποία υπολογίζεται στην αρχή κάθε επανάληψης από την αντίστοιχη ρουτίνα. Παρατηρώντας τη δομή του πίνακα αυτού, γίνεται αμέσως φανερό ότι η κατανομή κατά γραμμές δεν ισοκατανέμει και το φορτίο υπολογισμού των στοιχείων της μήτρας. Για να γίνει εμφανές το γεγονός αυτό, παραθέτουμε στη συνέχεια ένα παράδειγμα. Έστω το δίκτυο τριών ζυγών του παρακάτω σχήματος:



Σχήμα 5.1: Παράδειγμα δικτύου τριών ζυγών

Το δίκτυο αποτελείται από τρεις ζυγούς, άρα η μήτρα H θα έχει έξι στήλες (οι τρεις πρώτες αντιστοιχούν σε γωνίες και οι επόμενες τρεις σε τάσεις). Επίσης, στο

δίκτυο έχουμε δύο μετρήσεις ενεργού και άεργου ροής ισχύος (στους κλάδους 1-2 και 1-3), μία μέτρηση ενεργού και άεργου έγχυσης (στο ζυγό 2)και δύο μετρήσεις τάσεων (στους ζυγούς 1 και 2). Οι μετρήσεις καθορίζουν το πλήθος των γραμμών της μήτρας H, οι οποίες στην περίπτωσή μας θα είναι οκτώ. Με βάση τις εξισώσεις υπολογισμού των στοιχείων της μήτρας H που δόθηκαν στην προηγούμενη ενότητα, και θεωρώντας ως αρχικό διάνυσμα εισόδου το x_0=[0, 0, 0, 1.0, 1.0, 1.0], μπορούμε εύκολα να κατασκευάσουμε τη δομή της μήτρας για την πρώτη επανάληψη:

$$H(x_0) = \begin{array}{c} \\ \theta p_{12} \\ \theta p_{13} \\ \theta p_2 \\ \theta q_{12} \\ \theta q_{13} \\ \theta q_2 \\ \theta V_1 \\ \theta V_2 \end{array} \begin{array}{c} \begin{array}{cccccc} \theta_1 & \theta_2 & \theta_3 & V_1 & V_2 & V_3 \end{array} \\ \begin{bmatrix} h_{11} & h_{12} & 0 & h_{14} & h_{15} & 0 \\ h_{21} & 0 & h_{23} & h_{24} & 0 & h_{26} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} & h_{36} \\ h_{41} & h_{42} & 0 & h_{44} & h_{45} & 0 \\ h_{51} & 0 & h_{53} & h_{54} & 0 & h_{56} \\ h_{61} & h_{62} & h_{63} & h_{64} & h_{65} & h_{66} \\ 0 & 0 & 0 & h_{74} & 0 & 0 \\ 0 & 0 & 0 & 0 & h_{85} & 0 \end{bmatrix} \end{array}$$

Παρατηρώντας τη δομή αυτή, γίνεται αμέσως αντιληπτή η διαπίστωση που κάναμε προηγουμένως: Αν η παραπάνω μήτρα μοιραστεί κατά γραμμές, τότε η διεργασία η οποία θα πάρει μια γραμμή που αντιστοιχεί σε μέτρηση έγχυσης ισχύος, θα χρειαστεί να υπολογίσει έξι στοιχεία της μήτρας. Αντίθετα, η διεργασία στην οποία θα ανατεθεί μία γραμμή μέτρησης τάσης, θα χρειαστεί να υπολογίσει μόνο ένα στοιχείο. Στην υλοποίησή μας, λοιπόν, θεωρήσαμε καταλληλότερη τη διανομή του πίνακα κατά στήλες, δηλαδή το χωρισμό του δικτύου κατά ζυγούς. Για να το πετύχουμε αυτό, χρησιμοποιώντας ταυτόχρονα και τους τύπους δεδομένων της PETSc, τροποποιήσαμε τη ρουτίνα υπολογισμού της μήτρας H, ώστε να υπολογίζει την ανάστροφη μήτρα. Ο υπολογισμός του ανάστροφου πίνακα, στη συνέχεια, για την ανάκτηση της μήτρας H, δεν προσθέτει κάποιο overhead, αφού απαιτείται έτσι κι αλλιώς για τον υπολογισμό της μήτρας κέρδους G.

Άλλο ένα γεγονός που μπορεί να «ξενίσει» τον προγραμματιστή στη χρήση της PETSc, είναι πως λόγω της ιδιαίτερης μορφής αποθήκευσης, δεν μπορούμε να διαβάσουμε ένα στοιχείο του πίνακα με τον κλασικό τρόπο δεικτών της C. Αντ' αυτού, πρέπει να χρησιμοποιηθεί συγκεκριμένη ρουτίνα για το διάβασμα ενός στοιχείου, η οποία απαιτεί ενδιάμεση αποθήκευση σε άλλη μεταβλητή ή πίνακα. Το ίδιο πρόβλημα παρουσιάζεται και στην περίπτωση που θέλουμε να θέσουμε μια τιμή στον πίνακα. Και πάλι, πρέπει να χρησιμοποιήσουμε ενδιάμεση μεταβλητή για την αποθήκευση της τιμής και έπειτα να καλέσουμε την κατάλληλη ρουτίνα. Η διαδικασία αυτή, επιβάρυνε τις λειτουργίες υπολογισμού της μήτρας H και της συνάρτησης μετρήσεων, κάθε στοιχείο των οποίων προκύπτει από ξεχωριστό μαθηματικό τύπο.

Αν και «stable» η έκδοση της βιβλιοθήκης που χρησιμοποιήσαμε, είναι ακόμη υπό ανάπτυξη, πράγμα που σημαίνει ότι bugs υπάρχουν και διορθώνονται συνεχώς από την κοινότητα υποστήριξης. Ωστόσο, το debugging ρουτινών της βιβλιοθήκης είναι αρκετά δύσκολο, λόγω του δυσνόητου και περιεκτικού κώδικα και του συγκεκριμένου στυλ προγραμματισμού με το οποίο χρειάζεται εξοικείωση. Αλλαγή σε κάποια ρουτίνα απαιτεί μεταγλώττιση από την αρχή ολόκληρης της βιβλιοθήκης, το οποίο είναι χρονοβόρο.

Τέλος, ένα χαρακτηριστικό το οποίο δυσκόλεψε την υλοποίηση του παράλληλου WLS αλγόριθμου, είναι ότι το πακέτο της PETSc δεν περιέχει

παράλληλα υλοποιημένους άμεσους επιλυτές (LU, Cholesky). Οι προγραμματιστές της PETSc αποθαρρύνουν τη χρήση ακόμη και των σειριακά υλοποιημένων άμεσων επιλυτών. Αντίθετα, προτείνουν και υποστηρίζουν τη λύση με μεθόδους Krylov, ως περισσότερο φιλικές προς το χρήστη και ελεγμένες μεθόδους. Ωστόσο, υπάρχει δυνατότητα για χρήση «εξωτερικών πακέτων» που παρέχουν κάποιο interface για παράλληλους άμεσους επιλυτές, όπως το PASTIX. Επειδή, όμως, είναι πακέτα προσφερόμενα από τρίτους, δεν υπάρχει πλήρης υποστήριξη και διόρθωση λαθών-προβλημάτων από την κοινότητα της PETSc.

5.4 Επίδραση του δικτύου στην επίδοση

Από τα πειραματικά αποτελέσματα που εξετάσαμε στο προηγούμενο κεφάλαιο, είδαμε ότι ο χρόνος επικοινωνίας αποτελεί τον κύριο ανασταλτικό παράγοντα για την επίτευξη υψηλής επίδοσης. Ιδιαίτερα για τους αλγόριθμους επαναληπτικής επίλυσης γραμμικών συστημάτων, η ανάγκη για επικοινωνία είναι πολύ μεγάλη, καθώς οι διεργασίες απαιτείται να ανταλλάσσουν μεταξύ τους δεδομένα, συνήθως μικρού μεγέθους, τόσο κατά την διάρκεια μιας επανάληψης όσο και ανάμεσα σε διαδοχικές επαναλήψεις. Για το λόγο αυτό, για να επιτύχουμε υψηλή επίδοση σε υλοποιήσεις τέτοιων αλγόριθμων, είναι καλό να έχουμε στη διάθεσή μας ένα δίκτυο διασύνδεσης με πολύ μικρό χρόνο απόκρισης (latency). Το δίκτυο του εργαστηρίου υπολογιστικών συστημάτων, στο οποίο πραγματοποιήθηκαν οι πειραματικές μετρήσεις, είναι Gigabit Ethernet, το έχει σημαντικό latency, γεγονός που εμποδίζει τις εφαρμογές αυτές να κλιμακώσουν.

5.5 Θέματα για περαιτέρω διερεύνηση

Με την ολοκλήρωση της εργασίας αυτής, αναδείχθηκαν διάφορα θέματα τα οποία χρήζουν περαιτέρω διερεύνησης και θα μπορούσαν να αποτελέσουν αντικείμενο επόμενων διπλωματικών εργασιών. Αναφέρουμε μερικά από αυτά:

Α. Αλγόριθμος Εκτίμησης Κατάστασης

- Υλοποίηση του αλγόριθμου με την προσθήκη τεχνητών καθυστερήσεων στις συνδέσεις μεταξύ των κόμβων ώστε να προσομοιωθεί ακριβέστερα η τοπολογία ενός πραγματικού Σ.Η.Ε.
- Δημιουργία αρχείων και εκτέλεση του αλγόριθμου για μεγαλύτερα δίκτυα ώστε να μελετηθεί η συμπεριφορά κλιμάκωσής του.
- Δημιουργία και εκτέλεση σεναρίων διακοπής λειτουργίας κάποιου κόμβου του δικτύου και ανάπτυξη μηχανισμών ανάκαμψης.
- Δημιουργία και εκτέλεση σεναρίων εντοπισμού εσφαλμένων μετρήσεων από κάποιο κόμβο και ανάπτυξη μηχανισμών ανάκαμψης ή προσέγγισης της λύσης με ελλειπή δεδομένα.

Β. Αλγόριθμος Conjugate Gradient

- Μελέτη της υλοποιημένης έκδοσης του αλγόριθμου της PETSc και δημιουργία σεναρίων για σύγκριση.
- Υλοποίηση και σύγκριση συμπεριφορών του αλγόριθμου χρησιμοποιώντας διαφορετικούς Preconditioners.
- Περαιτέρω μελέτη κατανομής χρόνου υπολογισμών και επικοινωνίας. Εύρεση μεθόδων για δυνατότητα επικάλυψης υπολογισμών και επικοινωνίας.

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Βιβλιογραφία

[1] David Culler and Jaswinder Pal Singh, «Parallel Computer Architecture, A hardware/software Approach», Kaufmann 1997.

[2] Hennessy and Patterson, «Computer Architecture A Quantitative Approach», Morgan Kaufmann, 1990.

[3] Stone, «High-Performance Computer Architecture», Addison-Wesley, Third Edition, 1993

[4] El-Rewini and Lewis, «Distributed and Parallel Computing», Manning & Prentice Hall, 1998.

[5] Hesham El–Rewini and Mostafa Abd-El-Barr «Advanced Computer Architecture and parallel processing», John Wiley & Sons, New Jersey 2005

[6] Γ.Κ. Παπακωνσταντίνου, Θ.Α. Θεοχάρης, Π.Δ. Τσανάκας, «Συστήματα Παράλληλης Επεξεργασίας», Εκδόσεις Συμμετρία, Αθήνα 1994.

[7] Peter S. Pacheco «A User's Guide to MPI», University of San Francisco, San Francisco March 1998

[8] Marc Snir and Steve Otto «MPI The complete Reference», MIT Boston 1996

[9] Neil MacDonald, Elspeth Minty, Tim Haridng «Writing Message-Passing Parallel Programs With MPI», University of Edinburgh

[10] George Em Karniadakis and Robert M. Kirby II, «Parallel Scientific Computing in C++ and MPI», Cambridge University Press.

[11] Ν.Καδιανάκης, Σ.Καρανάσιος, «Γραμμική Άλγεβρα Αναλυτική Γεωμετρία και Εφαρμογές», 3η έκδοση, 2003.

[12] Α.Μπακόπουλος, Ι.Χρυσοβέργης, «Εισαγωγή στην Αριθμητική Ανάλυση», Εκδόσεις Συμεών, 1999.

[13] Torsten Hoefler, Peter Gottschling, Andrew Lumsdaine, Wolfgang Rehm, «Optimizing a Conjugate Gradient Solver with Non-Blocking Collective Operations», Indiana University, 2004.

[14] Jack Dongarra and Victor Eijkhout, «Finite-choice algorithm optimization in Conjugate Gradients», University of Tennessee 2003.

[15] George Karypis and Vipin Kumar, «Parallel Threshold-based ILU Factorization», University of Minnesota 1998.

[16] George N. Korres «A Distrubuted Multi – Area State Estimation», under review

[17] Γ. Κονταξής και Ν. Χατζηαργυρίου «Κέντρα Ελέγχου Ενέργειας». Εκδόσεις Ε.Μ.Π. Αθήνα 2003

[18] Mohammad Shahidehpour and Yaoyu Wang, «Communication and control in Electric Power Systems, Applications of Parallel and Distributed Processing», IEEE PRESS 2003

[19] Ali Abur and Antonio Gomez Exposito, «Power System Estimation, Theory and implementation», Marcel Dekker New York 2004

[20] T. Van Cutsem, J. L. Horward, and M. Ribbens-Pavella, «A two-level static state estimator for electric power systems», *IEEE Trans. Power App. Syst.*, vol. PAS-100, no. 8, pp. 3722–3732, Aug. 1981.

[21] T. Van Cutsem and M. Ribbens-Pavella, «Critical survey of hierarchical methods for state estimation of electric power systems», *IEEE Trans. Power App. Syst.*, vol. PAS-102, no. 10, pp. 247–256, Oct. 1983.

[22] H. Sasaki, K. Aoki, and R. Yokoyama, «A parallel computation algorithm for static state estimation by means of matrix inversion lemma», *IEEE Trans. Power Syst.*, vol. 2, no. 3, pp. 624-632, Aug. 1987.

[23] D. M. Falcao, F. F. Wu, and L. Murphy, «Parallel and Distributed State Estimation», *IEEE Trans. Power Syst.*, vol. 10, no. 2, pp. 724–730, May 1995.

[24] G. N. Korres and G. C. Contaxis, «Application of a reduced model to a distributed state estimator», in IEEE PES Winter Meeting, Singapore, Jan. 2000.

[25] R. Ebrahimian and R. Baldick, «State Estimation Distributed Processing», *IEEE Trans. Power Syst.*, vol. 15, no. 4, pp. 1240–1246, Nov. 2000.

[26] L. Zhao and A. Abur, «Multiarea State Estimation Using Synchronized Phasor Measurements», *IEEE Trans. Power Syst.*, vol. 20, no. 2, pp. 611–617, May 2005.

[27]http://www.mcs.anl.gov/petsc/petsc-as/

[28]http://www.mcs.anl.gov/petsc/petsc/as/documentation/tutorials/TACCTutorial2009.pdf

[29]http://www.mcs.anl.gov/petsc/petscas/documentation/tutorials/GUCASTutorial.pdf

[30] http://en.wikipedia.org/wiki/Krylov_subspace

[31] http://en.wikipedia.org/wiki/Conjugate_gradient

[32] http://en.wikipedia.org/wiki/GMRES

[33] http://en.wikipedia.org/wiki/LU_decomposition

[34] http://en.wikipedia.org/wiki/Cholesky_decomposition

[35] http://en.wikipedia.org/wiki/System_of_linear_equations