ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΧΗΜΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΙΙ: ΑΝΑΛΥΣΗΣ, ΣΧΕΔΙΑΣΜΟΥ &
ΑΝΑΠΤΥΞΗΣ ΔΙΕΡΓΑΣΙΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του ΜΙΧΑΗΛ ΑΛΙΦΙΕΡΑΚΗ

«ΥΠΟΛΟΓΙΣΜΟΙ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ ΣΕ ΠΑΡΑΛΛΗΛΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΜΕ ΧΡΗΣΗ ΜΕΘΟΔΩΝ ΔΙΑΧΩΡΙΣΜΟΥ ΧΩΡΙΟΥ»

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΑΝΔΡΕΑΣ Γ. ΜΠΟΥΝΤΟΥΒΗΣ

ΑΘΗΝΑ 2011

Περίληψη

Η μεγάλη διαθεσιμότητα των παράλληλων υπολογιστών και οι προοπτικές χρήσης τους για αριθμητική επίλυση μεγάλης κλίμακας συστημάτων διαφορικών εξισώσεων με μερικές παραγώγους έχει οδηγήσει σε εκτεταμένη έρευνα στον τομέα των μεθόδων διαχωρισμού χωρίου.

Στην εργασία αυτή επιχειρήθηκε η χρήση της επαναληπτικής μεθόδου Alternating Schwarz σε συνδυασμό με τη μέθοδο πεπερασμένων στοιχείων για την επίλυση ενός απλού προβλήματος αγωγής θερμότητας σε πτερύγιο. Στα πλαίσια της εργασίας αναπτύχθηκε κώδικας σε γλώσσα Fortran 90 και με χρήση της συστοιχίας Pegasus της σχολής Χημικών Μηχανικών ΕΜΠ εξετάζονται οι χρόνοι επίλυσης και η παράλληλη επιτάχυνση του κώδικα. Διαπιστώνεται μεγάλο πλήθος επαναλήψεων της μεθόδου και προτείνεται η στροφή σε άλλους μεθόδους διαχωρισμού χωρίου.

Παράλληλα αναπτύχθηκε κώδικας που να υλοποιεί σειριακά τον αλγόριθμο της Alternating Schwarz, με τον οποίο επιτεύχθηκε σημαντική μείωση της απαίτησης σε μνήμη σε σχέση με τον απλό κώδικα πεπερασμένων στοιχείων. Ο κώδικας αυτός είναι ένα χρήσιμο εργαλείο για οικονομικότερη επίλυση προβλημάτων πεπερασμένων στοιχείων μεγάλου μεγέθους σε έναν επεξεργαστή.

Abstract

Large-scale computational problems arising from the discretization of partial differential equations are solved most efficiently on high-performance computer clusters by means of numerical methods which exploit today's parallel computer architectures. Among the most promising ones are the Domain Decomposition Methods, which distribute, on a spatial basis, the computational load to the computer processors.

In this thesis the iterative method Alternating Schwarz was used in conjunction with the Finite Element Method for the solution of a simple, two-dimensional heat conduction problem. The calculations were programmed in Fortran 90 and performed on the cluster "Pegasus" of the Chemical Engineering School of NTUA. The reported results include the computer time required for the solution, the parallel speedup achieved along with the dependence of the number of iterations required for convergence in connection with the size of the computational problem. The results reveal advantages and disadvantages of the implemented method and are suggestive of modifications that need to be done.

In addition, a sequential code was developed which implements the Alternating Schwarz. Significant reduction in memory requirements was achieved compared to the simple Finite Element Method code. This code is a useful tool for economical (in terms of memory usage) solution of large finite element problems on a single processor.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή Α. Μπουντουβή για την καθοδήγησή του και την υποστήριξη που μου παρείχε.

Επίσης, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον Δρα Α. Σπυρόπουλο για όσα με δίδαξε σε θέματα παράλληλης επεξεργασίας και γενικότερα για τη βοήθεια του στο υπολογιστικό κομμάτι της εργασίας.

Τέλος, ευχαριστώ θερμά το Νίκο Χειμαριό, το Σωκράτη Γαρνέλη, το Δημήτρη Παπαγεωργίου και το Δρα. Μιχάλη Καβουσανάκη για τις πολύτιμες συζητήσεις μας σε υπολογιστικά ζητήματα και για το άριστο κλίμα συνεργασίας.

Περιεχόμενα

1. Παράλληλη Επεξεργασία

Ως παράλληλη επεξεργασία (parallel processing) ορίζεται η ταυτόχρονη χρήση περισσότερων της μίας επεξεργαστικής μονάδας για την εκτέλεση ενός προγράμματος ή μίας σειράς υπολογισμών. Αυτό που προσπαθούμε να επιτύχουμε μέσα από αυτή είναι η αύξηση των υπολογιστικών επιδόσεων και μείωση του απαιτούμενου χρόνου εκτέλεσης των υπολογισμών [I6].

Η συνεχώς αυξανόμενη ζήτηση μεγαλύτερης υπολογιστικής ισχύς για την επίλυση προβλημάτων αιχμής της έρευνας και της τεχνολογίας έχει οδηγήσει στη χρήση υπολογιστών με πολλούς επεξεργαστές σε συνδυασμό με μεθόδους παράλληλης επεξεργασίας. Οι μέθοδοι αυτές στοχεύουν στην ταυτόχρονη εκμετάλλευση πολλών επεξεργαστών για την αντιμετώπιση μιας μεγάλης υπολογιστικής διεργασίας (task), χωρίζοντας την σε μικρότερες ανεξάρτητες υποδιεργασίες (subtasks). Η χρήση λοιπόν του όρου παράλληλη επεξεργασία υπονοεί υπολογιστικές διατάξεις με περισσότερους του ενός επεξεργαστές. Οι διατάξεις αυτές είναι γνωστές ως υπερυπολογιστές (supercomputers) ή παράλληλοι υπολογιστές (parallel computers) ή υπολογιστές υψηλής απόδοσης (high performance computers) [Σπυρόπουλος (2003)].

1.1 Ταξινόμηση κατά Flynn

Ανάλογα με τον τρόπο παροχής των εντολών και των δεδομένων στις μονάδες επεξεργασίας διακρίνονται τέσσερις κατηγορίες (Flynn's taxonomy), αρχιτεκτονικής υπολογιστών, από τις οποίες η μία είναι σειριακή και οι υπόλοιπες τρεις έχουν δυνατότητες παράλληλης επεξεργασίας [I6, Σπυρόπουλος (2003), I5]:

1. Αρχιτεκτονική Μοναδικής Εντολής Μοναδικού Δεδομένου (Single Instruction Single Data, SISD): μια μονάδα επεξεργασίας εκτελεί ακολουθιακά τις εντολές (μια προς μια) πάνω σε μια σειρά δεδομένων. Πρόκειται για το κλασικό μοντέλο σειριακής αρχιτεκτονικής Von Neumann. Παράδειγμα τέτοιων υπολογιστών είναι οι προσωπικοί υπολογιστές (PC) μονού πυρήνα.
2. Αρχιτεκτονική Μοναδικής Εντολής Πολλαπλών Δεδομένων (Single Instruction Multiple Data, SIMD): οι μονάδες επεξεργασίας εκτελούν συγχρονισμένα μια κοινή ακολουθία εντολών πάνω σε διαφορετικά δεδομένα.

Οι υπολογιστές της κατηγορίας, ανάλογα με τον τρόπο που διαχειρίζονται τα δεδομένα, αναφέρονται με τον όρο διανυσματικοί υπολογιστές (vector computers) ή υπολογιστές πινάκων (array computers).

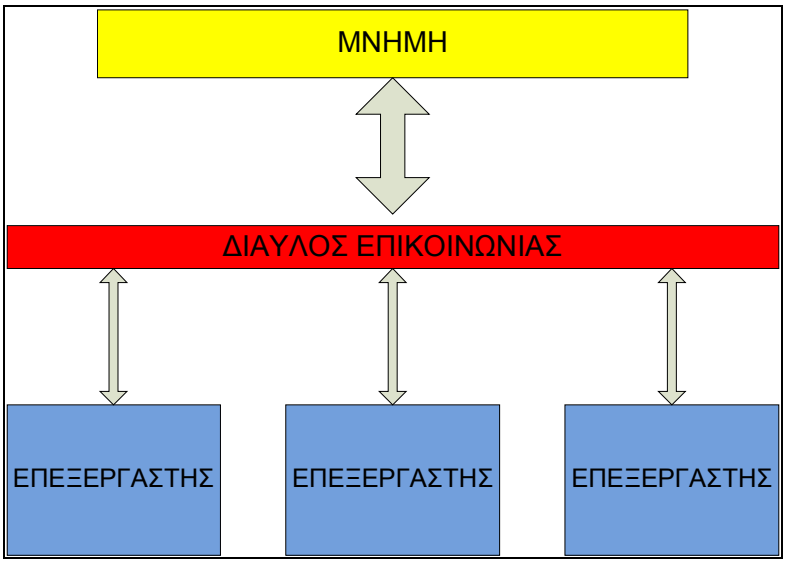
3. Αρχιτεκτονική Πολλαπλών Εντολών Μοναδικού Δεδομένου (Multiple Instruction Single Data, MISD). Λίγοι υπολογιστές μπορούν να ενταχθούν στην αρχιτεκτονική αυτή. Η αρχιτεκτονική αυτή δεν χρησιμοποιήθηκε για επιστημονικούς υπολογισμούς και σήμερα έχει εκλείψει.
4. Αρχιτεκτονική Πολλαπλών Εντολών Πολλαπλών Δεδομένων (Multiple Instruction Multiple Data, MIMD): οι μονάδες επεξεργασίας είναι ελεύθερες να εκτελούν, ανεξάρτητα η μια από την άλλη οποιαδήποτε εντολή με οποιαδήποτε δεδομένα. Οι υπολογιστές αυτής της αρχιτεκτονικής αναφέρονται ως υπολογιστές πολυεπεξεργαστών (multiprocessor computers).

1.2 Υπολογιστές Αρχιτεκτονικής MIMD

Οι υπολογιστές πού στηρίζονται στην αρχιτεκτονική MIMD παίζουν σήμερα τον σημαντικότερο ρόλο στην τομέα της παράλληλης επεξεργασίας. Ανάλογα με τον τρόπο προσπέλασης των επεξεργαστών στην μνήμη διακρίνονται σε δυο βασικές κατηγορίες: (α) στους υπολογιστές κοινής μνήμης (shared memory) και (β) στους υπολογιστές κατανεμημένης μνήμης (distributed memory) [Σπυρόπουλος (2003), Moldovan (1993)].

1.2.1 Κοινή Μνήμη

Σύμφωνα με το μοντέλο της κοινής μνήμης (Εικόνα 1.1) όλοι οι επεξεργαστές έχουν πρόσβαση σε μια μνήμη μέσω ενός διαύλου επικοινωνίας (bus). Οι επεξεργαστές επικοινωνούν μεταξύ τους μέσω της μνήμης, δηλαδή ένας επεξεργαστής γράφει τα δεδομένα σε μια θέση της μνήμης και οι υπόλοιποι μπορούν να τα διαβάσουν από την θέση αυτή.



Εικόνα 1.1. Διάταξη επεξεργαστών και μνήμης σε MIMD αρχιτεκτονικές κοινής μνήμης [Χειμαριός (2008)].

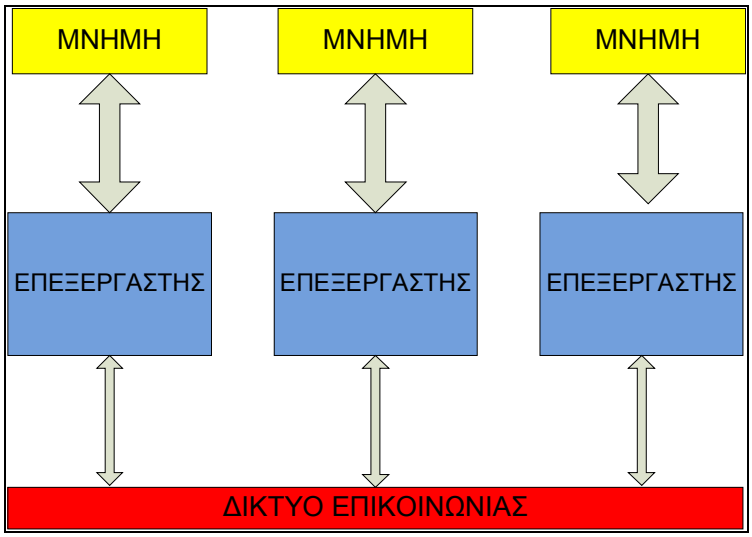
Το πλεονέκτημα της αρχιτεκτονικής αυτής είναι ο εύκολος προγραμματισμός αφού δεν υπάρχει η ανάγκη ρητής επικοινωνίας μεταξύ των επεξεργαστών. Το πλεονέκτημα αυτό είναι και ο βασικός λόγος που οι υπολογιστές αυτοί γνώρισαν ευρεία αποδοχή από την επιστημονική κοινότητα αφού οι αρχικοί σειριακοί κώδικες μπορούσαν να εφαρμοστούν στα συστήματα αυτά χωρίς βασικές αλλαγές στην δομή τους.

Ωστόσο, η αρχιτεκτονική αυτή περιορίζει το μέγιστο πλήθος των επεξεργαστών που μπορούν να χρησιμοποιηθούν, αφού δεν είναι δυνατή η ταυτόχρονη πρόσβαση πολλών επεξεργαστών στην κοινή μνήμη, λόγω τεχνολογικών ορίων στην κατασκευή του δίαυλου επικοινωνίας. Πρακτικά είναι πολύ δύσκολο να κατασκευαστούν υψηλής ταχύτητας δίαυλοι επικοινωνίας που να υποστηρίζουν τους σύγχρονους επεξεργαστές καθώς η ταχύτητα τους αυξάνεται πολύ γρήγορα. Έτσι τίθεται ένα άνω όριο στον αριθμό των επεξεργαστών που μπορούν να συνδεθούν με μονό δίαυλο επικοινωνίας και πρακτικά δεν υπερβαίνει τους 20 επεξεργαστές. Για την σύνδεση περισσότερων επεξεργαστών χρησιμοποιούνται πολλαπλοί δίαυλοι επικοινωνίας (crossbars), αυξάνοντας όμως κατά πολύ το κόστος κατασκευής. Παράδειγμα αυτής της αρχιτεκτονικής είναι οι προσωπικοί υπολογιστές με επεξεργαστές πολλαπλών πυρήνων.

1.2.2 Κατανεμημένη Μνήμη

Σύμφωνα με το μοντέλο της κατανεμημένης μνήμης (βλέπε εικόνα 1.2) κάθε επεξεργαστής έχει την δικιά του μνήμη χωρίς να έχει πρόσβαση στην μνήμη των υπολοίπων. Όταν ένας επεξεργαστής χρειάζεται δεδομένα που υπάρχουν στην μνήμη ενός άλλου επεξεργαστή τότε απαιτείται επικοινωνία μεταξύ των επεξεργαστών. Η επικοινωνία αυτή γίνεται μέσω ενός δικτύου επικοινωνίας.

Η κατηγορία αυτή αποτελεί σήμερα το πιο γρήγορα αναπτυσσόμενο κομμάτι των υπερυπολογιστών. Στον δικτυακό τόπο Top 500 [I1] κάθε εξάμηνο ανακοινώνονται οι 500 πιο γρήγοροι παράλληλοι υπολογιστές, σύμφωνα με το πρόγραμμα μέτρησης επίδοσης Linpack [I7]. Σύμφωνα με την λίστα αυτή τα ισχυρότερα υπολογιστικά συστήματα στηρίζονται στην αρχιτεκτονική της κατανεμημένης μνήμης. Οι υπολογιστές αυτοί υπερτερούν σε υπολογιστική ισχύ των υπολογιστών κοινής μνήμης αφού μπορούν να έχουν θεωρητικά άπειρο πλήθος επεξεργαστών χωρίς να υπάρχει ο περιορισμός της πρόσβασης στην μνήμη.

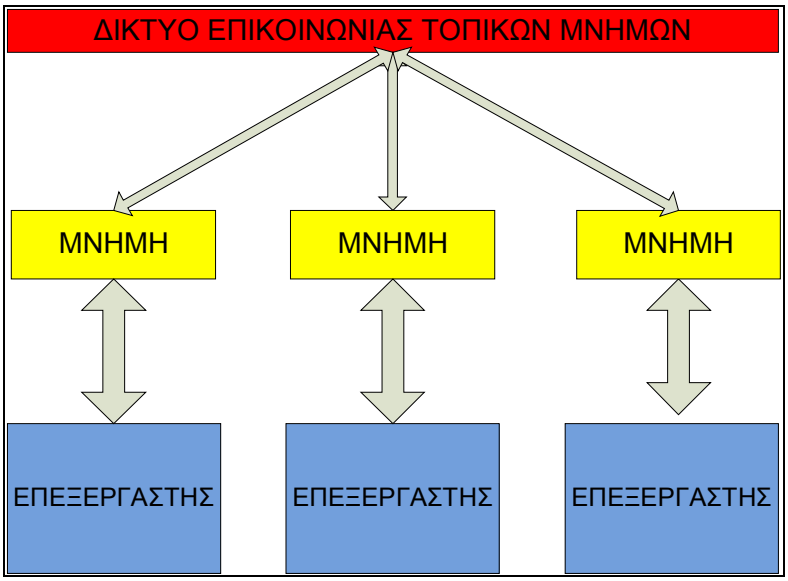


Εικόνα 1.2. Διάταξη επεξεργαστών και μνήμης σε MIMD αρχιτεκτονικές κατανεμημένης μνήμης [Χειμαριός (2008)].

Ο προγραμματισμός των υπολογιστών αυτών είναι πιο δύσκολος σε σχέση με τους υπολογιστές κοινής μνήμης αφού ο χρήστης πρέπει να κατανείμει τα δεδομένα στους επεξεργαστές και να προγραμματίσει ρητά την επικοινωνία μεταξύ τους.

1.2.3 Εικονική Κοινή Μνήμη

Οι δυο προηγούμενες κατηγορίες είναι οι βασικές αρχιτεκτονικές παράλληλων υπολογιστών. Οι κατασκευάστριες εταιρείες, στηριζόμενες στις αρχιτεκτονικές αυτές, κατασκευάζουν σήμερα παράλληλους υπολογιστές που συνδυάζουν τις δυο αυτές βασικές κατηγορίες. Στα Σχήματα 1.3 και 1.4 φαίνονται δυο σύγχρονες αρχιτεκτονικές υπερυπολογιστών.



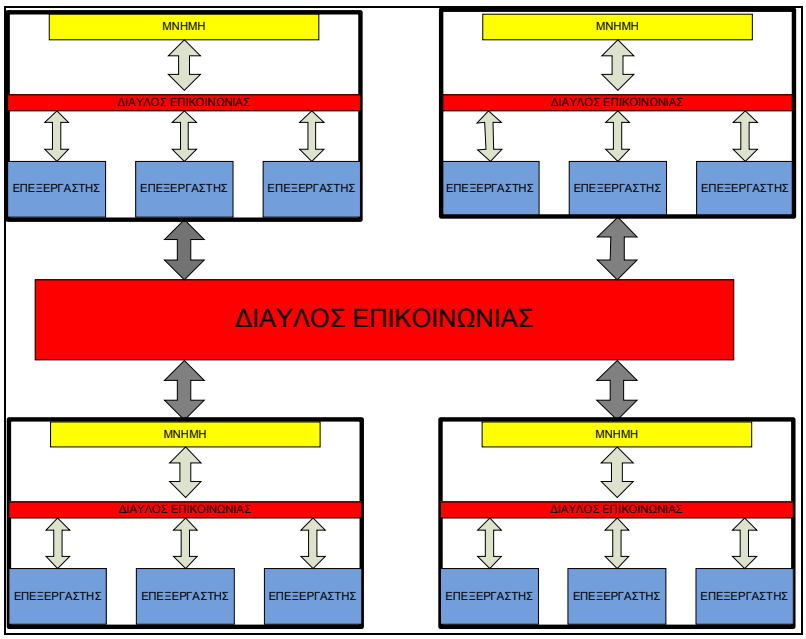
Εικόνα 1.3. Διάταξη επεξεργαστών και μνήμης σε MIMD αρχιτεκτονικές εικονικής κοινής μνήμης [Χειμαριός (2008)].

Η αρχιτεκτονική του σχήματος 1.3, γνωστή ως αρχιτεκτονική εικονικής κοινής μνήμης (virtual shared memory), στηρίζεται στο μοντέλο των υπολογιστών κατανεμημένης μνήμης, όπου κάθε επεξεργαστής έχει την δικιά του μνήμη, ταυτόχρονα όμως κάθε επεξεργαστής έχει και απ' ευθείας πρόσβαση στην μνήμη των υπολοίπων μέσω ενός κυκλώματος επικοινωνίας ανεξάρτητου των επεξεργαστών.

1.2.4 Μεταβλητή Ταχύτητα Πρόσβασης στην Μνήμη

Το πρόβλημα του περιορισμού στο μέγιστο αριθμό των επεξεργαστών που μπορούν να χρησιμοποιηθούν στις αρχιτεκτονικές κοινής μνήμης επιλύεται με τεχνικές μεταβλητής ταχύτητας πρόσβασης στη μνήμη (Non Uniform Memory Access, NUMA). Το βασικό χαρακτηριστικό των αρχιτεκτονικών κοινής μνήμης είναι ότι όλοι οι επεξεργαστές έχουν την ίδια ταχύτητα πρόσβασης στη μνήμη μέσω του διαύλου ή των πολλαπλών διαύλων επικοινωνίας. Μπορούν όμως να

κατασκευαστούν υπολογιστές κοινής μνήμης με διαφορετική ταχύτητα πρόσβασης των επεξεργαστών στη μνήμη.



Εικόνα 1.4. Διάταξη επεξεργαστών και μνήμης σε MIMD αρχιτεκτονικές NUMA [Χειμαριός 2008)].

Στην εικόνα 1.4 φαίνεται μια τέτοια αρχιτεκτονική. Αποτελείται από τέσσερις μονάδες επεξεργασίας, κάθε μια από τις οποίες βασίζεται στην αρχιτεκτονική της κοινής μνήμης (συνήθως κάθε τέτοια μονάδα επεξεργασίας περιέχει μέχρι 16 επεξεργαστές). Κάθε επεξεργαστής έχει πρόσβαση στην τοπική μνήμη της μονάδας επεξεργασίας που ανήκει μέσω ενός διαύλου επικοινωνίας αλλά ταυτόχρονα έχει πρόσβαση και στις μνήμες των υπολοίπων μονάδων μέσω ενός δεύτερου διαύλου επικοινωνίας μικρότερης ταχύτητας από τον τοπικό δίαυλο. Με την τεχνική αυτή μπορούν να κατασκευαστούν υπολογιστές κοινής μνήμης με μεγάλο πλήθος επεξεργαστών. Οι υπολογιστές της κατηγορίας αυτής συχνά αναφέρονται με τον όρο υπολογιστές κατανεμημένης κοινής μνήμης (DSM, Distributed Shared Memory).

1.3 Συστοιχίες Υπολογιστών

Τα τελευταία χρόνια η ραγδαία εξέλιξη στο υλικό (hardware) των προσωπικών υπολογιστών, κυρίως στους επεξεργαστές, από τις δύο κυρίαρχες στον χώρο εταιρίες Intel και AMD, αλλά και στο υπόλοιπο υλικό (π.χ. μητρικές πλακέτες, μνήμες), οδήγησε στην κατασκευή προσωπικών υπολογιστών με εξαιρετικά υψηλές

υπολογιστικές επιδόσεις και με χαμηλό οικονομικό κόστος. Από την άλλη πλευρά, η εξέλιξη της τεχνολογίας των δικτύων υπολογιστών, οδήγησε σε δίκτυα με μεγάλη χωρητικότητα (bandwidth) και με μικρή καθυστέρηση (latency), όπως το Fast Ethernet, το Gigabit Ethernet, το Myrinet και το SCI. Οι εξελίξεις αυτές σε συνδυασμό με την ανάπτυξη του σταθερού, αξιόπιστου και ελεύθερα διαθέσιμου λειτουργικού συστήματος Linux, έδωσαν την ώθηση για την πραγματοποίηση επιστημονικών υπολογισμών μεγάλης κλίμακας σε συστοιχίες Beowulf (Beowulf clusters) [Σπυρόπουλος (2003)].

Μια συστοιχία Beowulf είναι ένας υψηλής απόδοσης παράλληλος υπολογιστής, κατασκευασμένος από προσωπικούς υπολογιστές – κόμβους της συστοιχίας. Οι κόμβοι φέρουν ελεύθερα διαθέσιμο λειτουργικό σύστημα, συνήθως Linux, και είναι κατάλληλα διασυνδεδεμένοι με αποκλειστικό (private) δίκτυο επικοινωνίας υψηλής ταχύτητας.

Μια συστοιχία μπορεί να είναι είτε ομογενής δηλαδή όλοι οι κόμβοι να έχουν το ίδιο υλικό και το ίδιο λειτουργικό σύστημα είτε ετερογενής, δηλαδή να έχει διαφορετικούς κόμβους ως προς το υλικό ή/και ως προς το λειτουργικό σύστημα.

Το 1994 ο Thomas Sterling, στα πλαίσια του ερευνητικού έργου Beowulf, κατασκεύασε την πρώτη συστοιχία με 16 κόμβους διασυνδεδεμένους με δίκτυο Ethernet, στο ερευνητικό κέντρο CESDIS (Center of Excellence in Space Data and Information Sciences) της NASA. Ήταν η πρώτη προσπάθεια για την ανάπτυξη παράλληλου υπολογιστικού συστήματος βασισμένου σε προσωπικούς υπολογιστές με στόχο την δημιουργία ενός συστήματος υψηλών επιδόσεων με πολύ μικρότερο κόστος από αυτό των υπερυπολογιστών [I6].

Από τότε οι συστοιχίες άρχισαν να κερδίζουν συνεχώς έδαφος έναντι των υπερυπολογιστών γιατί εκτός απ’ την χαμηλή τους τιμή έφτασαν οι επιδόσεις τους να είναι αντίστοιχες. Χάρη σε αυτά τα πλεονεκτήματα φτάσαμε σήμερα σε σημείο στη λίστα υπερυπολογιστών TOP 500 οι 411 από τους 500 υπολογιστές της λίστας να είναι συστοιχίες (δεδομένα Ιουνίου 2011 [I1]).

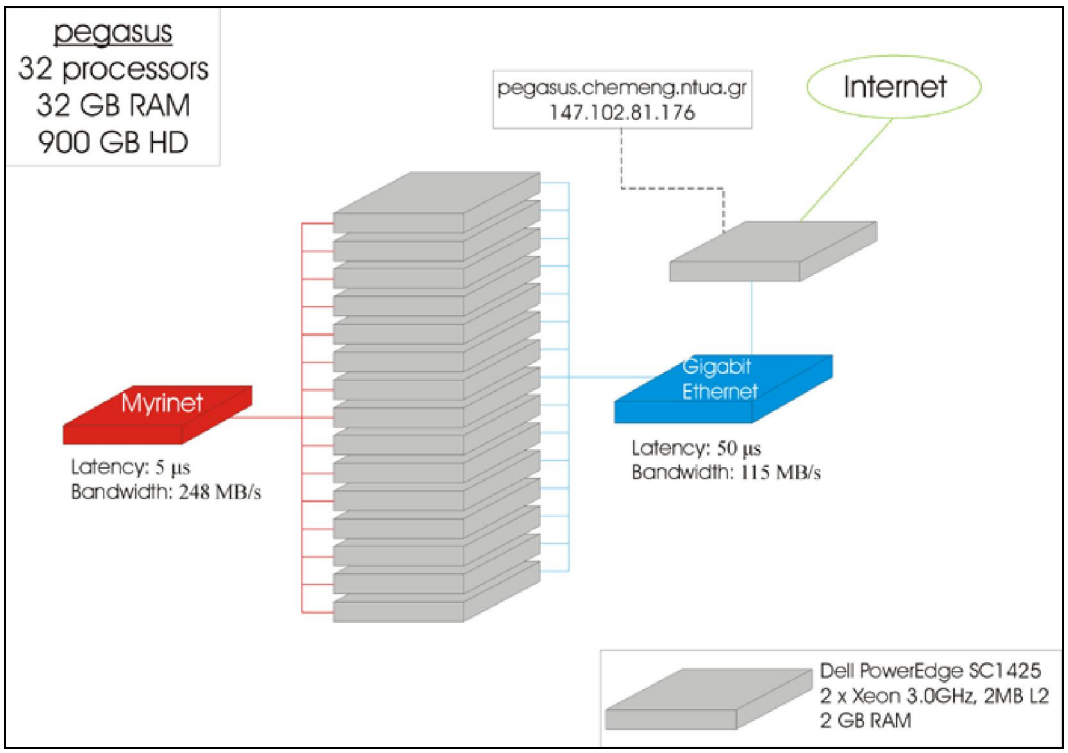
Στη συνέχεια παρουσιάζουμε τα δύο συστήματα που κατασκευάστηκαν στη σχολή Χημικών Μηχανικών ΕΜΠ και τα οποία χρησιμοποιήθηκαν για τους υπολογισμούς μας.

1.3.1 Η συστοιχία Pegasus

Ο Pegasus αποτελείται από 16 κόμβους διπλοεπεξεργασίας (32 επεξεργαστές συνολικά) και συναρμολογήθηκε στο Υπολογιστικό Κέντρο της Σχολής Χημικών Μηχανικών, Ε.Μ.Π. [I3]. Κάθε κόμβος αποτελείται από δυο επεξεργαστές Xeon στα 3 GHz και 2GB RAM. Η διασύνδεση των κόμβων υλοποιείται με δύο δίκτυα επικοινωνίας Myrinet και Gigabit Ethernet. Μέσω του δικτύου Gigabit Ethernet γίνεται η διαχείριση (administration) των κόμβων της συστοιχίας (NFS, monitoring, file transfer κ.τ.λ), ενώ μέσω του πιο γρήγορου δικτύου Myrinet πραγματοποιείται η ανταλλαγή μηνυμάτων με την MPI (βλέπε παράγραφο 1.6). Το λειτουργικό σύστημα της συστοιχίας είναι η ελεύθερα διαθέσιμη διανομή Linux για συστοιχίες, Rocks 4.1 (http://www.rocksclusters.org).

Εικόνα 1.5. Ο Pegasus [I3].

Στην εικόνα 1.6 φαίνεται σχηματικά η διάταξη του Pegasus με τα βασικότερα τμήματα που την αποτελούν.



Εικόνα 1.6. Σχηματική αναπαράσταση του Pegasus [I3].

1.3.2 Ο Arion

Ο Arion είναι ένας υπολογιστής κοινής μνήμης ο οποίος αποτελείται από 2 διπλοπύρηνους επεξεργαστές Opteron της εταιρίας AMD και διαθέτει μνήμη 8 Gb. Το λειτουργικό σύστημα του Arion είναι η ελεύθερα διαθέσιμη διανομή Linux, Centos 4.5.

1.4 Παράλληλοι Αλγόριθμοι

Για την επίτευξη υψηλής παράλληλης επιτάχυνσης απαιτείται η μετατροπή του ήδη υπάρχοντος σειριακού αλγόριθμου σε παράλληλο. Ως παράλληλος αλγόριθμος ορίζεται ο αλγόριθμος του οποίου οι λειτουργίες μπορούν να διαμεριστούν σε ανεξάρτητες υπολειτουργίες και να εκτελεστούν ταυτόχρονα σε πολλούς επεξεργαστές. Η μετατροπή αυτή μπορεί να έχει ως αποτέλεσμα την αλλαγή του σειριακού κώδικα σε ποσοστό από 1% μέχρι την αντικατάστασή του από καινούργιο αλγόριθμο. Η πρώτη περίπτωση αναφέρεται στην ταυτόχρονη εκτέλεση του σειριακού κώδικα σε πολλούς επεξεργαστές με διαφορετικές παραμέτρους εισαγωγής (input parameters) και στη συνέχεια επεξεργασία των επιμέρους αποτελεσμάτων. Η

μορφή αυτή της παράλληλης επεξεργασίας είναι δυνατή όταν η εκτέλεση του σειριακού κώδικα πάνω σε μια ομάδα παραμέτρων δεν εξαρτάται από την προηγούμενη. Ακόμη όμως και σε τέτοιες περιπτώσεις η ανάγκη της αύξησης του μεγέθους του προβλήματος για την καλύτερη εκτίμηση της λύσης ή την διερεύνηση νέων φαινομένων οδηγεί στην παράλληλη επεξεργασία [Σπυρόπουλος (2003), Jordan & Alaghband (2003)].

1.5 Παράμετροι Εκτίμησης Παράλληλης Απόδοσης

Η ποιότητα ενός παράλληλου αλγορίθμου μετριέται με την παράλληλη επιτάχυνση (parallel speedup) και με την παράλληλη απόδοση (parallel efficiency). Ως παράλληλη επιτάχυνση, S_p, ορίζεται ο λόγος του χρόνου εκτέλεσης (execution time) του παράλληλου αλγόριθμου σε έναν επεξεργαστή προς τον χρόνο εκτέλεσης σε p επεξεργαστές [Σπυρόπουλος (2003), I6]:

$$S_p = \frac{T_1}{T_p} \tag{1.1}$$

όπου T_1 και T_p ο χρόνος εκτέλεσης σε έναν και σε p επεξεργαστές αντίστοιχα. Η ιδανική παράλληλη επιτάχυνση είναι ίση με p.

Ως παράλληλη απόδοση, E_p, ορίζεται ο λόγος της παράλληλης επιτάχυνσης προς τον αριθμό των επεξεργαστών [Σπυρόπουλος (2003), I6]:

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p} \tag{1.2}$$

Η ιδανική παράλληλη απόδοση είναι ίση με 1.

1.5.1 Ο Νόμος του Amdahl

Η ιδανική περίπτωση ενός παράλληλου αλγορίθμου είναι όταν εκτελείται σε p επεξεργαστές να "τρέχει" p φορές ταχύτερα. Τυχόν σειριακά κομμάτια του παράλληλου αλγόριθμου μειώνουν την παράλληλη επιτάχυνση. Ο νόμος του Amdahl [Σπυρόπουλος (2003), Dongarra et al (2003)] θέτει ένα άνω όριο στη μέγιστη παράλληλη επιτάχυνση που μπορεί να επιτευχθεί σε σχέση με το ποσοστό, a, του

χρόνου εκτέλεσης των λειτουργιών (operations) που γίνονται σειριακά προς τον χρόνο εκτέλεσης των λειτουργιών που γίνονται παράλληλα:

$$S_p \leq \frac{1}{a + (1\text{-}a)/p} \leq \frac{1}{a} \tag{1.3}$$

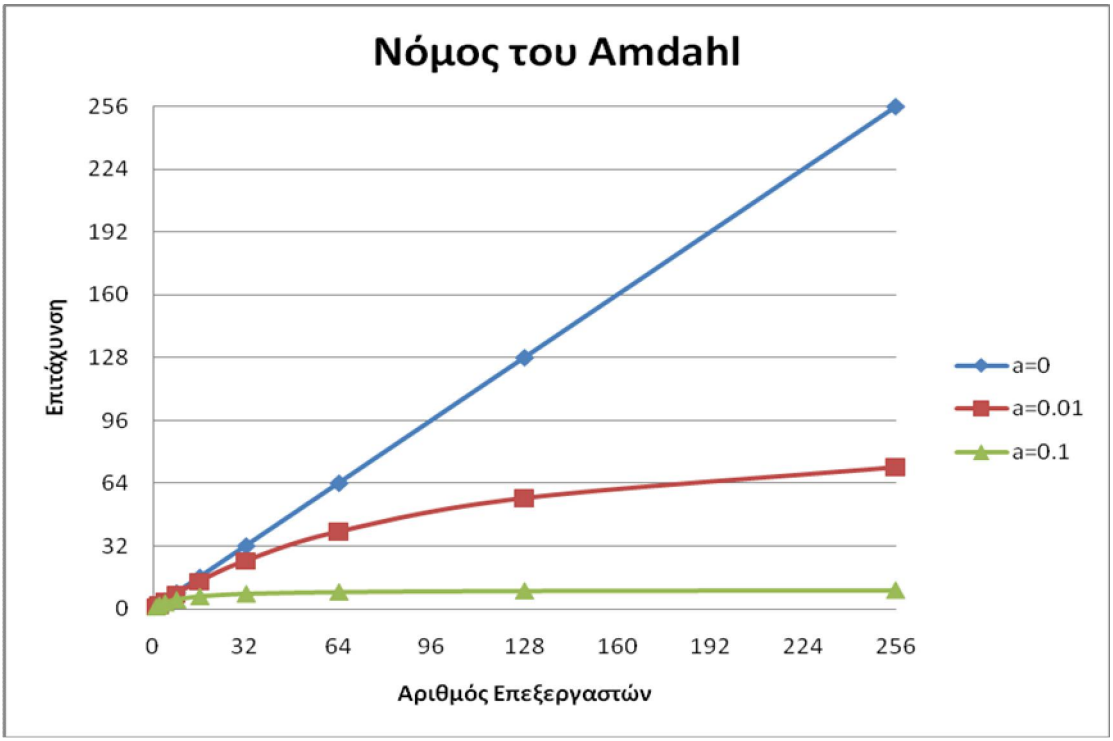
Στην εικόνα 1.7 φαίνεται το διάγραμμα της μέγιστης αναμενόμενης παράλληλης επιτάχυνσης σε σχέση με τον αριθμό των επεξεργαστών για τρεις τιμές του a.

Όπως φαίνεται στο διάγραμμα ακόμη και στην περίπτωση που το ποσοστό του χρόνου εκτέλεσης των σειριακών λειτουργιών είναι μόλις το 1% του χρόνου εκτέλεσης των παράλληλων λειτουργιών η αύξηση των επεξεργαστών πέρα από τους 128 δεν έχει ουσιαστικά επίδραση στον χρόνο εκτέλεσης. Τα πράγματα γίνονται χειρότερα στην περίπτωση που το a αυξηθεί περισσότερο.

Ακόμη και στην ιδανική περίπτωση όπου το ποσοστό a, είναι ίσο με το μηδέν πάλι παρουσιάζονται αποκλίσεις από την ιδανική παράλληλη επιτάχυνση. Οι παράγοντες που αυξάνουν το ποσοστό a εκτός από τα σειριακά κομμάτια ενός παράλληλου αλγόριθμου είναι [Σπυρόπουλος (2003)]:

(α) ο απαιτούμενος χρόνος επικοινωνίας μεταξύ των επεξεργαστών στην περίπτωση που ο παράλληλος αλγόριθμος εκτελείται σε αρχιτεκτονικές κατανεμημένης μνήμης. Ο αντίστοιχος χρόνος σε αρχιτεκτονικές κοινής μνήμης είναι η καθυστέρηση που προκύπτει στην προσπέλαση τον επεξεργαστών στην μνήμη μέσω του διαύλου επικοινωνίας,

(β) ανισομερής κατανομή του υπολογιστικού φόρτου (load imbalance) στους επεξεργαστές. Οι επεξεργαστές είναι αναγκασμένοι να περιμένουν τον επεξεργαστή που έχει αναλάβει το μεγαλύτερο υπολογιστικό φορτίο να τελειώσει προκειμένου να συνεχίσουν.

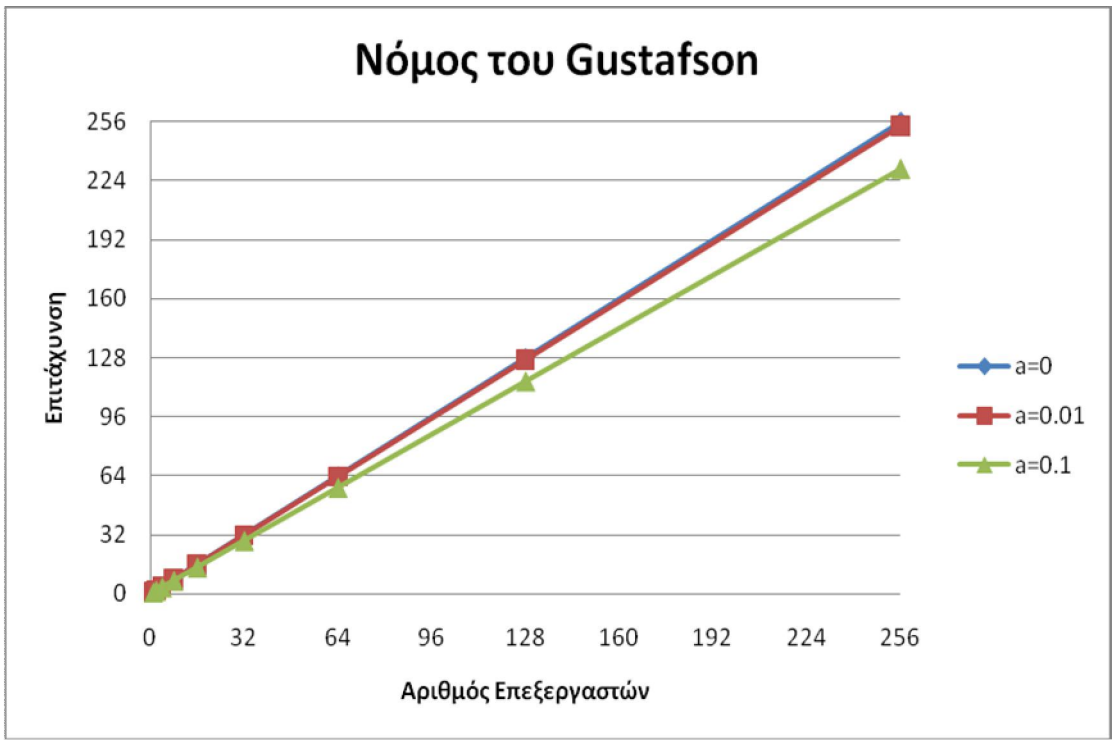


Εικόνα 1.7. Διαγραμματική παράσταση του Νόμου του Amdahl [Σπυρόπουλος (2003)].

1.5.2 Ο Νόμος του Gustafson

Όταν διατυπώθηκε για πρώτη φορά του νόμου του Amdahl δημιουργήθηκε η εντύπωση ότι υπολογιστικά συστήματα με μεγάλο πλήθος επεξεργαστών είναι στην ουσία άχρηστα. Οι ερωτήσεις που τίθενται αμέσως είναι σε τι εξυπηρετούν οι ήδη υπάρχοντες παράλληλοι υπολογιστές με επεξεργαστές της τάξης των 5000, γιατί να επενδυθεί, χρόνος για την μετατροπή ενός σειριακού αλγόριθμου σε παράλληλο, και χρήμα για την αγορά παράλληλου υπολογιστή, από την στιγμή που η απόδοση είναι πολύ μικρή [Σπυρόπουλος (2003)].

Παρόλο που ο νόμος του Amdahl δεν είναι λανθασμένος, οδηγεί σε λανθασμένα συμπεράσματα για την αστοχία της παράλληλης επεξεργασίας και δεν μπορεί να χρησιμοποιηθεί για να απαντηθούν τα παραπάνω ερωτήματα. Ο λόγος είναι ότι δεν λαμβάνει υπόψη του, τον σημαντικότερο ίσως παράγοντα της παράλληλης επεξεργασίας που είναι το μέγεθος του υπολογιστικού προβλήματος. Στις περισσότερες περιπτώσεις ο διπλασιασμός του μεγέθους του προβλήματος έχει ως αποτέλεσμα τον τετραπλασιασμό του χρόνου που απαιτείται για τις παράλληλες λειτουργίες σε σχέση με τον χρόνο που απαιτείται για τις σειριακές.



Εικόνα 1.8. Διαγραμματική παράσταση του Νόμου του Gustafson [Σπυρόπουλος (2003)].

Ο νόμος του Gustafson λαμβάνοντας υπόψη την εξάρτηση του a από το μέγεθος του προβλήματος ορίζει το ανώτατο όριο της παράλληλης επιτάχυνσης ως:

$$S_p \le p(1-a)+a \qquad (1.4)$$

Στην εικόνα 1.8 φαίνεται το διάγραμμα της μέγιστης αναμενόμενης παράλληλης επιτάχυνσης, σύμφωνα με τον νόμο του Gustafson, σε σχέση με τον αριθμό των επεξεργαστών για τρεις τιμές του a.

Όπως φαίνεται από το διάγραμμα ακόμα και με a=10% η παράλληλη επιτάχυνση είναι κοντά στην ιδανική. Ο νόμος του Gustafson δείχνει ότι η αποδοτική παράλληλη επεξεργασία συμβαδίζει με τα μεγάλου μεγέθους υπολογιστικά προβλήματα, όπου ο χρόνος που απαιτείται για τις σειριακές λειτουργίες και ο χρόνος της επικοινωνίας μεταξύ των επεξεργαστών είναι πολύ μικρός σε σχέση με τον χρόνο εκτέλεσης των παράλληλων λειτουργιών.

1.6 Η βιβλιοθήκη MPI

Η MPI *(Message Passing Interface)* είναι η καθιερωμένη βιβλιοθήκη για τον προγραμματισμό παράλληλων υπολογιστών [Χειμαριός (2008)] σύμφωνα με το

μοντέλο ανταλλαγής μηνυμάτων. Σύμφωνα με το μοντέλο αυτό ο χρήστης μπορεί να κατανέμει τα δεδομένα στους επεξεργαστές και να προγραμματίσει ρητά την μεταξύ τους επικοινωνία. Ο κώδικας μπορεί να γραφεί σε Fortran και C/C ++.

Ένα από τα σημαντικότερα πλεονεκτήματα της MPI είναι η εφαρμογή της ανεξάρτητα από την εταιρία κατασκευής του παράλληλου υπολογιστή. Κατά την εξέλιξη των πρώτων υπερυπολογιστών κάθε κατασκευάστρια εταιρία εφοδίαζε και προωθούσε το σύστημά της με τα κατάλληλα εργαλεία παράλληλου προγραμματισμού. Αυτό είχε σαν αποτέλεσμα αλλαγή του υπερυπολογιστή να απαιτεί αλλαγή του παράλληλου κώδικα. Η MPI μπορεί να χρησιμοποιηθεί σε οποιονδήποτε παράλληλο υπολογιστή, αν και μικρές ρυθμίσεις στον κάθε κώδικα μπορεί να γίνουν προκειμένου να εκμεταλλεύεται πλήρως τα αρχιτεκτονικά χαρακτηριστικά του κάθε παράλληλου υπολογιστή.

Υπάρχουν και άλλες βιβλιοθήκες καθώς και γλώσσες παράλληλου προγραμματισμού. Ενδεικτικά αναφέρουμε την OpenMP και την HPF *(High Performance Fortran)*.

2. Μέθοδοι διαχωρισμού χωρίου

Οι μέθοδοι διαχωρισμού χωρίου (domain decomposition methods) είναι μέθοδοι επίλυσης προβλημάτων συνοριακών τιμών χωρίζοντάς τα σε μικρότερα προβλήματα συνοριακών τιμών πάνω σε επιμέρους υποχωρία και λύνοντας επαναληπτικά για να "συντονιστούν" οι λύσεις μεταξύ των παρακείμενων υποχωρίων. Το υποπρόβλημα που δημιουργείται σε κάθε υποχωρίο είναι ανεξάρτητο από τα υπόλοιπα και αυτός είναι και ο λόγος που οι μέθοδοι διαχωρισμού χωρίου είναι ιδανικές για παράλληλους υπολογισμούς [I6].

Η υψηλή διαθεσιμότητα των παράλληλων υπολογιστών και οι προοπτικές τους για αριθμητική επίλυση υπολογιστικά απαιτητικών μερικών διαφορικών εξισώσεων (ΜΔΕ) έχει οδηγήσει σε εκτεταμένη έρευνα στον τομέα των μεθόδων διαχωρισμού χωρίου [I2]. Οι μέθοδοι αυτοί είναι γενικά ευέλικτες στην επίλυση είτε γραμμικών είτε μη γραμμικών συστημάτων εξισώσεων που προκύπτουν από τη διακριτοποίηση ΜΔΕ και χρησιμοποιούν θεμελιώδεις ιδιότητες των ΜΔΕ για επίτευξη γρήγορης επίλυσης. Για γραμμικά προβλήματα, οι μέθοδοι διαχωρισμού χωρίου μπορούν να χρησιμοποιηθούν ως προσταθεροποιητές (preconditioners) σε μεθόδους προβολής τύπου Krylov όπως είναι η CG (Conjugate Gradient) και η GMRES (Generalized Minimum RESidual) [Flemisch (2001)]. Για μη γραμμικά συστήματα, μπορούν να χρησιμοποιηθούν ως προσταθεροποιητές για την επίλυση των γραμμικών συστημάτων που προκύπτουν από τη χρήση της μεθόδου Newton ή ως προσταθεροποιητές για επιλύτες όπως την μέθοδο NCG (Nonlinear Conjugate Gradient) [Smith et al (1996)].

Ο όρος διαχωρισμός χωρίου έχει ελαφρώς διαφορετική έννοια για ειδικούς στο χώρο της παράλληλης επεξεργασίας [Smith et al (1996)]. Μπορεί να σημαίνει είτε τη διανομή δεδομένων στους επεξεργαστές (partitioning), είτε το διαχωρισμό του φυσικού χωρίου σε υποπεριοχές που μπορούν να μοντελοποιηθούν με διαφορετικές εξισώσεις, είτε τέλος την υποδιαίρεση της λύσης ενός μεγάλου γραμμικού συστήματος σε μικρότερα προβλήματα των οποίων οι λύσεις μπορούν να χρησιμοποιηθούν για να παραχθεί ένας προσταθεροποιητής. Σε αυτή την εργασία ο όρος διαχωρισμός χωρίου χρησιμοποιήθηκε με τη δεύτερη έννοια.

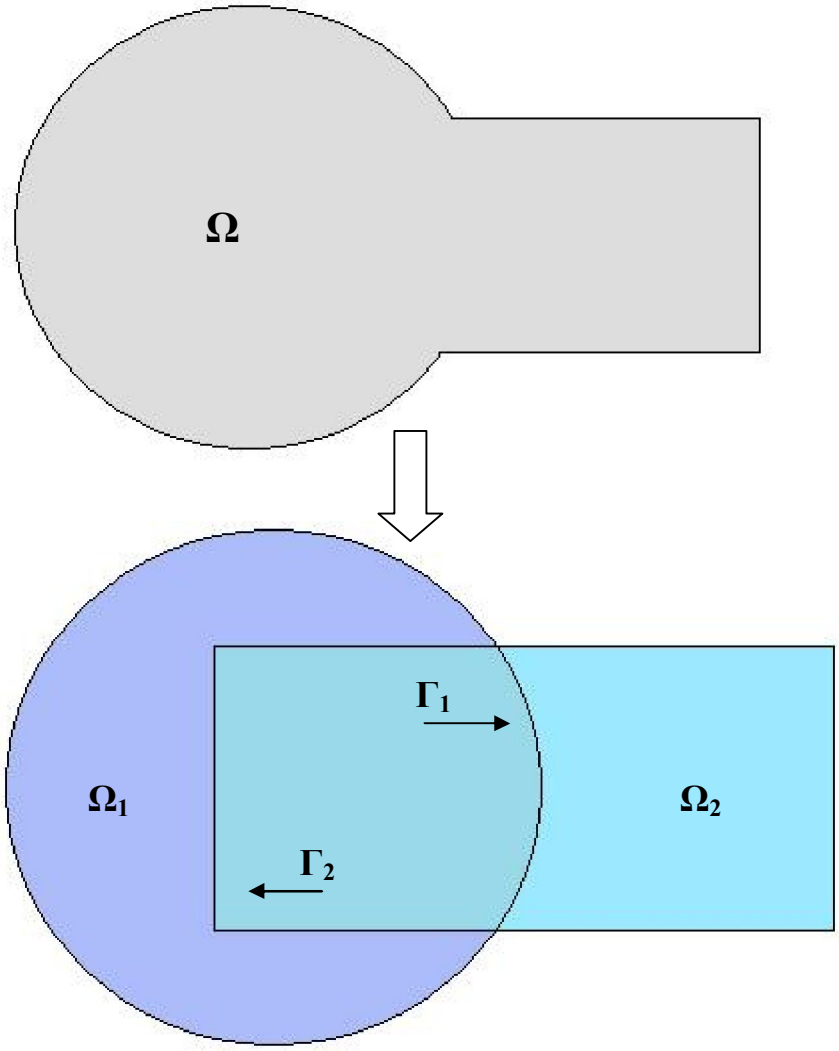
Τις μεθόδους διαχωρισμού χωρίου μπορούμε να τις χωρίσουμε σε δύο μεγάλες κατηγορίες: Σε αυτές οι οποίες απαιτούν να υπάρχει επικάλυψη μεταξύ των υποχωρίων (overlapping) και σε αυτές που δεν απαιτούν επικάλυψη υποχωρίων (non-overlapping).

2.1 Μέθοδοι με επικαλυπτόμενα υποχωρία

Στις μεθόδους με επικάλυψη τα γειτονικά υποχωρία επικαλύπτονται σε ποσοστό που ξεπερνάει τη διεπιφάνειά τους. Οι μέθοδοι αυτοί αποκαλούνται Schwarz[1] καθώς ήταν ο πρώτος που μίλησε για αυτούς σε άρθρο του που εκδόθηκε το 1870 [Schwartz (1870)]. Σε αυτό το άρθρο ο Schwarz προτείνει μία επαναληπτική μέθοδο επίλυσης ενός προβλήματος συνοριακών τιμών διαχωρίζοντας ένα σύνθετο χωρίο σε δύο υποχωρία απλούστερης γεωμετρίας (εικόνα 2.1).

Οι βασικές μέθοδοι που προέκυψαν αργότερα από την αρχική πρόταση του Schwarz ήταν οι: Alternating Schwarz, Multiplicative Schwarz και Additive Schwarz αλλά και πολλές παραλλαγές τους και συνδυασμοί τους με άλλες μεθόδους.

[1] Αξίζει να αναφέρουμε ότι τα τελευταία χρόνια έχουν αναπτυχθεί και μέθοδοι Schwarz που ανήκουν στην ομάδα των μεθόδων μη-επικαλυπτόμενων υποχωρίων [Bjorstad et al (1996)].



Εικόνα 2.1. Το πρωτότυπο χωρίο με το οποίο καταπιάστηκε ο Schwarz στο πρώτο άρθρο που εκδόθηκε για το διαχωρισμό χωρίου.

2.1.1 Μέθοδοι Alternating Schwarz

Αποτελεί ουσιαστικά την πρόταση του Schwarz για την επίλυση ενός σύνθετου χωρίου χωρίζοντάς το σε δύο υποχωρία των οποίων τα πλέγματα δεν ταυτίζονται στην περιοχή της επικάλυψης.

Ας πούμε ότι θέλουμε να επιλύσουμε ένα πρόβλημα συνοριακών τιμών της μορφής :

$$Lu = f \text{ στο } \Omega \qquad (2.1\alpha)$$

$$Bu = g \text{ στο } \partial\Omega \qquad (2.1\beta)$$

Ο αλγόριθμος αυτός λέει το εξής:

Αρχικά χωρίζω το χωρίο μου σε δύο επιμέρους υποχωρία ευκολότερης γεωμετρίας. Μετά ξεκινάω τους υπολογισμούς μου κάνοντας μία αρχική εκτίμηση για το u_2 (u_2^0) στα σημεία του Ω_2 που συμπίπτουν με το εσωτερικό σύνορο Γ_1 του Ω_1 και λύνω επαναληπτικά το πρόβλημα συνοριακών τιμών [Smith et al (1996), Wohlmuth (2000)]:

$$Lu_1^n = f \quad \text{στο } \Omega_1 \tag{2.2α}$$

$$Bu_1^n = g \quad \text{στο } \partial\Omega_1 \setminus \Gamma_1 \tag{2.2β}$$

$$u_1^n = u_2^{n-1}\Big|_{\Gamma_1} \quad \text{στο } \Gamma_1 \tag{2.2γ}$$

για να βρω το u_1^n (όπου n ο αριθμός της επανάληψης). Μετά ακολουθεί η επίλυση του παρακάτω συστήματος συνοριακών τιμών:

$$Lu_2^n = f \quad \text{στο } \Omega_2 \tag{2.3α}$$

$$Bu_2^n = g \quad \text{στο } \partial\Omega_2 \setminus \Gamma_2 \tag{2.3β}$$

$$u_2^n = u_1^n\Big|_{\Gamma_2} \quad \text{στο } \Gamma_2. \tag{2.3γ}$$

Άρα, σε κάθε υποβήμα της μεθόδου επιλύεται ένα ελλειπτικό πρόβλημα συνοριακών τιμών στο υποχωρίο Ω_i με τις δεδομένες συνοριακές τιμές, g, στο πραγματικό σύνορο $\partial\Omega_i \setminus \Gamma_i$ και για το εσωτερικό σύνορο Γ_i χρησιμοποιείται η προσεγγιστική λύση της προηγούμενης επανάληψης.

2.1.2 Μέθοδοι Multiplicative Schwarz

Σε πολλές εφαρμογές όμως είναι δυνατόν τα χρησιμοποιούμενα πλέγματα των υποχωρίων να ταυτίζονται στην περιοχή επικάλυψης. Η απλοποιημένη εκδοχή της μεθόδου Alternating Schwarz στην οποία θεωρούμε την παραπάνω προϋπόθεση ονομάζεται Multiplicative Schwarz.

Γράφοντας το γραμμικό σύστημα που προκύπτει για το διακριτό πρόβλημα ως $Au = f$, κάθε επανάληψη μπορεί να γραφεί σε δύο βήματα (ή σε n βήματα αν έχουμε n υποχωρία) [Smith et al (1996), I4]:

$$u^{n+1/2} = u^n + \begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix}(f - Au^n) \tag{2.4α}$$

και

$$u^{n+1} = u^{n+1/2} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix} (f - Au^{n+1/2}) . \qquad (2.4\beta)$$

Όπου το A_{Ω_i} είναι η διακριτή μορφή του τελεστή L περιορισμένου στο χωρίο Ωi. Απ' τη στιγμή που κάθε επανάληψη περιλαμβάνει σειριακά επιμέρους βήματα είναι εμφανές ότι αυτή η μέθοδος δεν είναι κατάλληλη για μεθόδους παράλληλης επεξεργασίας.

2.1.3 Μέθοδοι Additive Schwarz

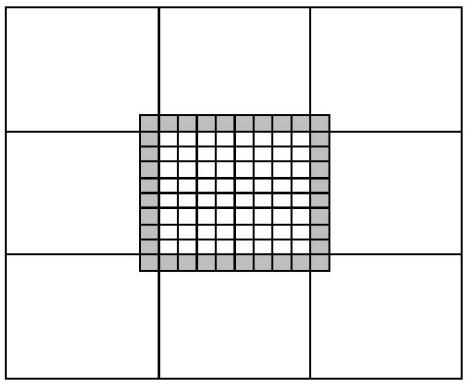
Η μέθοδος Additive Schwarz μπορεί να θεωρηθεί ως η παράλληλη εκδοχή της Multiplicative Schwarz και μπορεί να γραφτεί ως εξής [Smith et al (1996), I4]:

$$u^{n+1} = u^n + \left[\begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix} \right] (f - Au^n) . \qquad (2.5)$$

Η παραπάνω εξίσωση γράφεται και έτσι:

$$u^{n+1} = u^n + (B_1 + B_2)(f - Au^n) \text{, όπου } B_i = R_i^T A_{\Omega_i}^{-1} R_i , \qquad (2.6)$$

όπου Ri είναι ο ορθογώνιος περιοριστικός πίνακας (restriction matrix) ο οποίος επιστρέφει το διάνυσμα των συνιστωσών που ορίστηκαν στο εσωτερικό του Ωi (εικόνα 2.3).

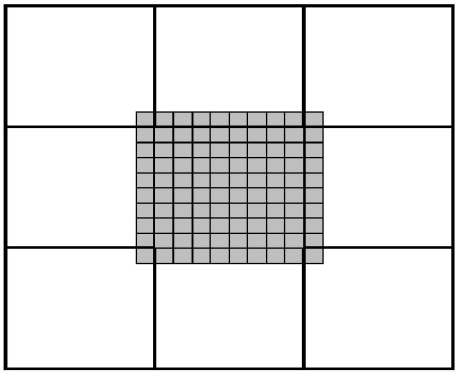


Εικόνα 2.2. Παράδειγμα υποχωρίου που επικαλύπτει τα έξι γειτονικά του.

Η μέθοδος αυτή εύκολα μπορεί να επεκταθεί ώστε να χρησιμοποιηθεί για περισσότερα των δύο υποχωρίων. Για ένα χωρίο $\Omega = \bigcup_i \Omega_i$ η Additive Schwarz μετατρέπεται ως εξής:

$$u^{n+1} = u^n + \sum_i B_i (f - Au^n) \text{, όπου } B_i = R_i^T A_{\Omega_i}^{-1} R_i . \qquad (2.7)$$

Οι μέθοδοι Additive Schwarz μπορούν να θεωρηθούν ως γενικεύσεις των μεθόδων block Jacobi.



Εικόνα 2.3. Η λειτουργία του περιοριστικού πίνακα R_i της Additive Schwarz.

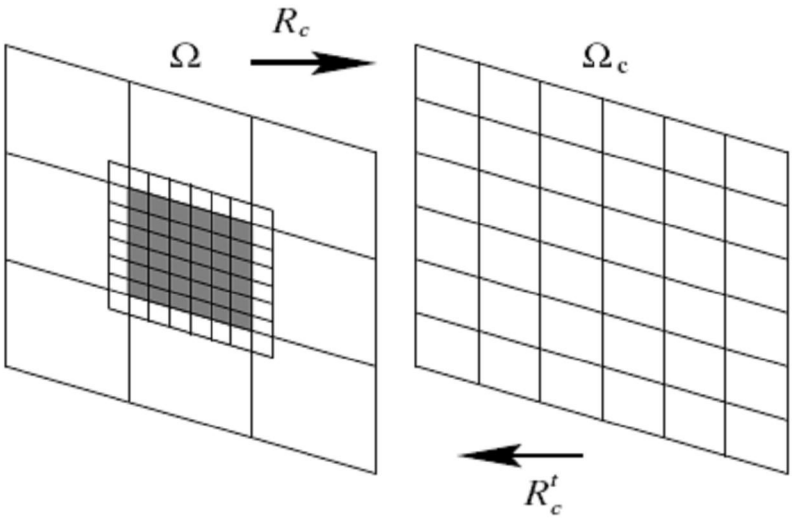
2.1.4 Μέθοδοι πολλαπλών επιπέδων

Οι μέθοδοι που περιγράψαμε παραπάνω ονομάζονται μονού επιπέδου (single level). Έχει αποδειχτεί όμως ότι είναι αποτελεσματικές μόνο για μικρό αριθμό υποχωρίων. Αυτό οφείλεται στο ότι η πληροφορία για τα δεδομένα f και g των εξισώσεων (2.1) σε ένα υποχωρίο μεταφέρεται στα υπόλοιπα υποχωρία μόνο αν περάσει από τα υποχωρία που βρίσκονται ενδιάμεσα. Ο επαναληπτικός επιλύτης του γραμμικού συστήματος χρειάζεται, λοιπόν, έναν μηχανισμό για την συνολική ανταλλαγή πληροφοριών μεταξύ όλων των υποχωρίων σε κάθε επανάληψη.

Ο πιο κοινός τρόπος μετάδοσης πληροφοριών που αφορούν πολλά υποχωρία είναι η χρήση μεθόδων δύο ή ακόμα καλύτερα πολλαπλών επιπέδων (multilevel methods). Μία μέθοδος πολλαπλών επιπέδων χρησιμοποιεί ένα ή περισσότερα αραιά χωρία (coarse spaces) Ω_c και λύνει το κατάλληλο πρόβλημα σε κάθε αραιό πλέγμα.

Όλοι οι "καλοί" (scalable) παράλληλοι γραμμικοί επιλύτες για πολύ μεγάλης κλίμακας γραμμικά συστήματα θα πρέπει να μπορούν να χρησιμοποιηθούν για διαφορετικούς αριθμούς υποχωρίων χωρίς να επηρεάζεται η απόδοσή τους. Ο ρυθμός σύγκλισης της Additive Schwarz μονού επιπέδου χειροτερεύει όταν ο αριθμός των υποχωρίων γίνεται μεγάλος. Αυτή η αδυναμία ξεπερνιέται με την εισαγωγή ενός αραιού, ενιαίου για όλο το χωρίο, προβλήματος και με βάση αυτό δημιουργείται ένας συνολικός μηχανισμός επικοινωνίας μεταξύ των υποχωρίων. Η επαναληπτική διαδικασία μίας μεθόδου Additive Schwarz δύο επιπέδων έχει ως εξής [I4]:

$r^n = b - Au^n$	Υπολόγισε το υπόλοιπο στο Ω
$r_c = R_c r^n$	Περιόρισε το υπόλοιπο στο Ω_c
$A_c c_c = r_c$	Λύσε στο Ω_c ($A_c = R_c A R_c^t$)
$c^n = R_c^t c_c$	Παρέμβαλε τη διόρθωση
$u^n = u^n + c^n$	Υπολόγισε τη νέα προσέγγιση
$u^{n+1} = u^n + \sum_i B_i r^n$	Additive Schwarz στο Ω.



Εικόνα 2.4. Η σχέση μεταξύ του Ω και του Ω_c.

2.2 Μέθοδοι με μη-επικαλυπτόμενα υποχωρία

Στις μεθόδους με μη-επικαλυπτόμενα υποχωρία, τα υποχωρία επικαλύπτονται μόνο στη διεπιφάνεια τους. Οι μέθοδοι αυτοί ονομάζονται και επαναληπτικές δομικές μέθοδοι (iterative substructuring methods) ή μέθοδοι συμπληρώματος Schur (Schur complement methods). Στις πρωτογενείς μεθόδους (primal) αυτού του είδους, όπως είναι η Balancing Domain Decomposition (BDD) και η BDDC (Balancing Domain Decomposition by Constraints), η συνέχεια της λύσης πέρα απ' τη διεπιφάνεια των υποχωρίων ενισχύεται με την απεικόνιση της τιμή της σε όλα τα γειτονικά υποχωρία. Στις δυικές μεθόδους (dual), όπως είναι η FETI, η συνέχεια της λύσης στα γειτονικά υποχωρία ενισχύεται με πολλαπλασιαστές Lagrange. Εκτός αυτών υπάρχουν και υβριδικές μέθοδοι όπως είναι η FETI-DP [I6].

Οι παραπάνω μέθοδοι είναι πολλά υποσχόμενες και έχει γίνει εκτενής έρευνα τις δύο τελευταίες δεκαετίες πάνω σε αυτές, αλλά δεν θα μας απασχολήσουν στην παρούσα διπλωματική.

3. Κώδικες και Αποτελέσματα

3.1 Το πρόβλημα

Έχουμε το εξής απλό πρόβλημα αγωγής θερμότητας σε δι-διάστατο πτερύγιο:

Σε μόνιμη κατάσταση, η κατανομή θερμοκρασίας, T, σε ορθογώνιο πτερύγιο ικανοποιεί την εξίσωση Laplace:

$$\nabla^2 T(x,y) = 0 \,,\ 0 < \mathrm{x} < \mathrm{L},\ 0 < \mathrm{y} < \mathrm{w} \,. \qquad (3.1\alpha)$$

Οι συνοριακές συνθήκες είναι:

$\mathrm{x} = 0 \,,\ \dfrac{\partial T}{\partial x} = 0$ (μόνωση) (3.1β)

$\mathrm{x} = \mathrm{L} \,,\ \dfrac{\partial T}{\partial x} = -h(T - T_0)$ (μεταφορά θερμότητας προς το περιβάλλον) (3.1γ)

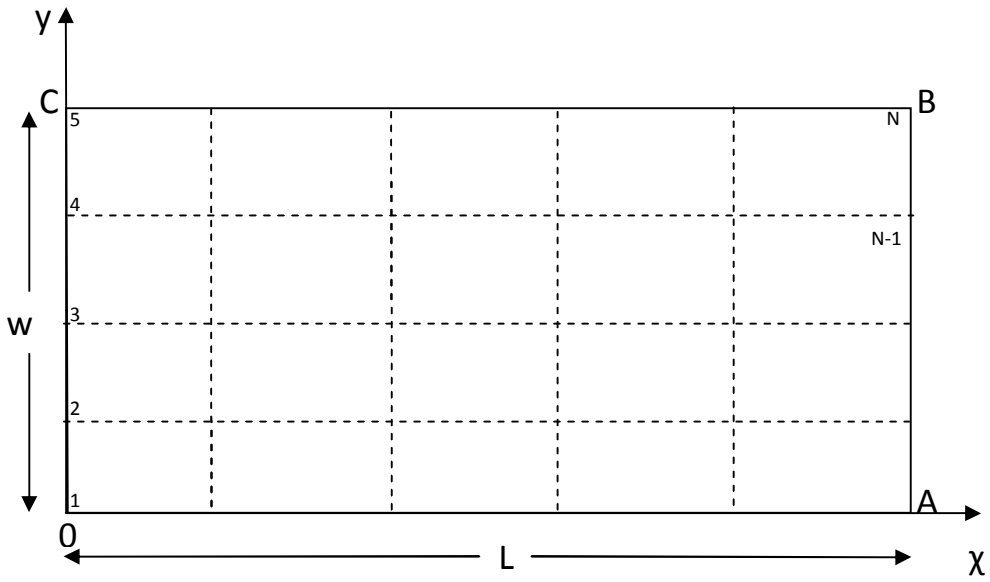
$\mathrm{y} = 0 \,,\ \mathrm{T} = \mathrm{T}_1$ (σταθερή θερμοκρασία) (3.1δ)

$\mathrm{y} = \mathrm{w} \,,\ \dfrac{\partial T}{\partial y} = -h(T - T_0)$ (μεταφορά θερμότητας προς το περιβάλλον) (3.1ε)

T_0 είναι η θερμοκρασία περιβάλλοντος, T_1 είναι σταθερή θερμοκρασία και h είναι συντελεστής μεταφοράς θερμότητας. Δίνονται:

$$T_0 = 20^{\mathrm{o}}\mathrm{C},\ T_1 = 200^{\mathrm{o}}\mathrm{C},\ \mathrm{h} = 1\ \mathrm{m}^{-1},\ \mathrm{L} = 4\ \mathrm{m},\ \mathrm{w} = 2\ \mathrm{m}.$$

Να βρεθεί η κατανομή θερμοκρασίας στο πτερύγιο με χρήση διωνυμικών συναρτήσεων βάσης.



Εικόνα 3.1. Το χωρίο Ω στο οποίο αναφέρεται το πρόβλημα. Η εσωτερική αρίθμηση είναι ενδεικτική και αναφέρεται στους κόμβους (Ν το πλήθος).

3.2 Προσεγγιστική επίλυση με Μέθοδο Galerkin/Πεπερασμένων Στοιχείων

Το παραπάνω πρόβλημα είναι ένα πρόβλημα συνοριακών τιμών της μορφής:

$$Lu = f \text{ στο } \Omega \qquad (3.2\alpha)$$

$$Bu = g \text{ στο } \partial\Omega \qquad (3.2\beta)$$

Όπου στην περίπτωσή μας ισχύει ότι:

- $L = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$,
- $f = 0$,
- $\Omega = (0,4) \times (0,2)$,
- $B = \frac{\partial}{\partial x}$ και $g = 1$ στο χ=0
- $B = \frac{\partial}{\partial x} + h\ell$ και $g = hT_0$ στο χ=L,
- $B = \ell$ και $g = T_1$ στο y=0,
- $B = \frac{\partial}{\partial y} + h\ell$ και $g = hT_0$ στο y=w

όπου ℓ είναι ο ταυτοτικός συντελεστής.

Προκειμένου να εφαρμοστεί η μέθοδος Galerkin [Μπουντουβής (1992), Strang & Fix (2008)] στο παραπάνω πρόβλημα θα πρέπει να υπολογίσουμε τα σταθμισμένα υπόλοιπα Galerkin (Galerkin weighted residuals) R_i και να βρούμε για ποιες τιμές του u μηδενίζονται:

$$R_i \equiv \int_{\Omega} (Lu - f)\varphi^i dS \tag{3.3}$$

Στην περίπτωσή μας:

$$\begin{aligned} R_i &= \iint_{\Omega} \varphi^i \nabla^2 T dxdy - \cancel{\iint_{\Omega} \varphi^i f dxdy} = \iint_{\Omega} \nabla \cdot \left(\varphi^i \nabla T\right) dxdy - \iint_{\Omega} \nabla \varphi^i \cdot \nabla T dxdy \\ &= \iint_{\Omega} \left\{ \frac{\partial}{\partial x}\left(\varphi^i \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\varphi^i \frac{\partial T}{\partial y}\right) \right\} dxdy - \iint_{\Omega} \frac{\partial \varphi^i}{\partial x}\frac{\partial T}{\partial x} dxdy - \iint_{\Omega} \frac{\partial \varphi^i}{\partial y}\frac{\partial T}{\partial y} dxdy \end{aligned} \tag{3.4}$$

όπου i η αρίθμηση των N το πλήθος κόμβων.

Το πρώτο ολοκλήρωμα στο τρίτο σκέλος της εξίσωσης (3.4) μετατρέπεται σε επικαμπύλιο με εφαρμογή του θεωρήματος Green στο επίπεδο:

$$\iint_{\Omega} \left(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} \right) dxdy = \oint_{\partial\Omega} \left(Pdy - Qdx \right),$$

όπου P=P(x, y) και Q=Q(x, y) και το σύνορο $\partial\Omega$ διαγράφεται με φορά αντίθετη της φοράς των δεικτών του ρολογιού: 0A→AB→BC→C0. Άρα η εξίσωση (3.4) γράφεται:

$$\begin{aligned} R_i &= \oint_{\partial\Omega} \varphi^i \left(\frac{\partial T}{\partial x} dy - \frac{\partial T}{\partial y} dy \right) - I_{x,i} - I_{y,i} \\ &= -\int_0^A \varphi^i \frac{\partial T}{\partial y} dx + \int_A^B \varphi^i \frac{\partial T}{\partial x} dy - \int_B^C \varphi^i \frac{\partial T}{\partial y} dx + \int_C^0 \varphi^i \frac{\partial T}{\partial x} dy - I_{x,i} - I_{y,i} \end{aligned} \tag{3.5}$$

όπου

$$I_{x,i} = \iint_{\Omega} \frac{\partial \varphi^i}{\partial x}\frac{\partial T}{\partial x} dxdy = \sum_{j=1}^{N} T_j \iint_{\Omega} \frac{\partial \varphi^i}{\partial x}\frac{\partial \varphi^j}{\partial x} dxdy \tag{3.6α}$$

$$I_{y,i} = \iint_{\Omega} \frac{\partial \varphi^i}{\partial y}\frac{\partial T}{\partial y} dxdy = \sum_{j=1}^{N} T_j \iint_{\Omega} \frac{\partial \varphi^i}{\partial y}\frac{\partial \varphi^j}{\partial y} dxdy \tag{3.6β}$$

Από τις συνοριακές συνθήκες (3.1δ), οι εξισώσεις διακριτοποίησης στους συνοριακούς κόμβους που ανήκουν στην πλευρά 0A του χωρίου Ω είναι:

$R_i = T_1 = 200$, όπου i οι αριθμοί των κόμβων που ανήκουν στην 0A πλευρά και τα αντίστοιχα στοιχεία του πίνακα $\underline{\underline{A}}$ και του διανύσματος $\underline{b}$ στο γραμμικό σύστημα $\underline{\underline{A}}\underline{u} = \underline{b}$, είναι:

$a_{ii}=1$, $b_i=200$, $a_{ij}=0$ για $i \neq j$, όπου i οι αριθμοί των κόμβων που ανήκουν στην 0A πλευρά και j=1,2,…,N.

Οι εξισώσεις διακριτοποίησης για τους συνοριακούς κόμβους που ανήκουν στην πλευρά 0C είναι:

$$R_i = \int_C^0 \varphi^i \frac{\partial T}{\partial x} dy - I_{x,i} - I_{y,i} = -I_{x,i} - I_{y,i} = 0$$

και επομένως:

$a_{ij} = -\left\{ \iint_\Omega \frac{\partial \varphi^i}{\partial x} \frac{\partial \varphi^j}{\partial x} dxdy + \iint_\Omega \frac{\partial \varphi^i}{\partial y} \frac{\partial \varphi^j}{\partial y} dxdy \right\}$, $b_i=0$, όπου i οι αριθμοί των κόμβων που ανήκουν στην 0C πλευρά και j=1,2,…,N. Όπως θα δούμε παρακάτω οι εξισώσεις διακριτοποίησης των κόμβων αυτών ταυτίζονται με τις εξισώσεις διακριτοποίησης των εσωτερικών κόμβων.

Οι εξισώσεις διακριτοποίησης για τους συνοριακούς κόμβους που ανήκουν στην πλευρά AB προκύπτουν με αντικατάσταση των μερικών παραγώγων στα επικαμπύλια ολοκληρώματα στις εξισώσεις (3.5) από τις συνοριακές συνθήκες (3.1γ):

$$R_i = \int_A^B \varphi^i \frac{\partial T}{\partial x} dy - I_{x,i} - I_{y,i} = \int_A^B \varphi^i h\left(T_0 - T\right) dy - I_{x,i} - I_{y,i} = 0$$

άρα:

$$a_{ij} = -\left\{ \iint_\Omega \frac{\partial \varphi^i}{\partial x} \frac{\partial \varphi^j}{\partial x} dxdy + \iint_\Omega \frac{\partial \varphi^i}{\partial y} \frac{\partial \varphi^j}{\partial y} dxdy + h\int_0^w \varphi^i \varphi^j dy \right\},$$

$b_i = -T_0 \int_0^w \varphi^i dy$, όπου i οι αριθμοί των κόμβων που ανήκουν στην AB πλευρά και j=1,2,…,N.

Ομοίως, για τις εξισώσεις διακριτοποίησης των συνοριακών κόμβων που ανήκουν στην πλευρά BC:

$$R_i = -\int_B^C \varphi^i \frac{\partial T}{\partial x} dx - I_{x,i} - I_{y,i} = \int_C^B \varphi^i h\left(T_0 - T\right) dx - I_{x,i} - I_{y,i} = 0$$

άρα:

$$a_{ij} = -\left\{ \iint_{\Omega} \frac{\partial \varphi^i}{\partial x} \frac{\partial \varphi^j}{\partial x} dxdy + \iint_{\Omega} \frac{\partial \varphi^i}{\partial y} \frac{\partial \varphi^j}{\partial y} dxdy + h \int_0^L \varphi^i \varphi^j dx \right\},$$

$b_i = -T_0 \int_0^L \varphi^i dx$, όπου i οι αριθμοί των κόμβων που ανήκουν στην BC πλευρά και j=1,2,…,N.

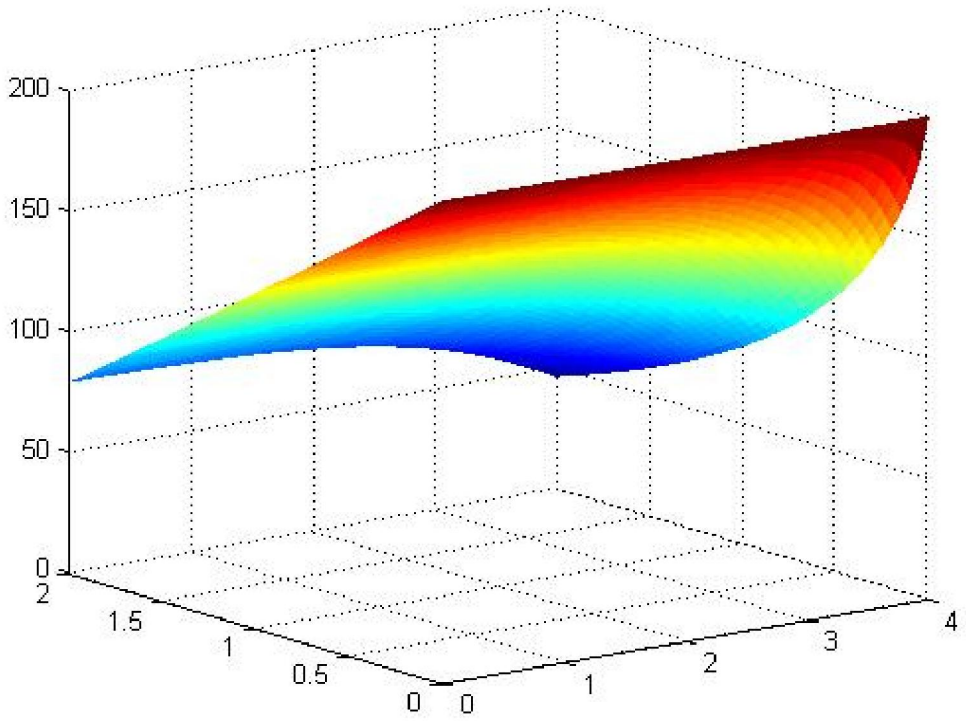
Στους εσωτερικούς κόμβους του πλέγματος οι εξισώσεις διακριτοποίησης είναι:

$$R_i = -I_{x,i} - I_{y,i} = 0$$

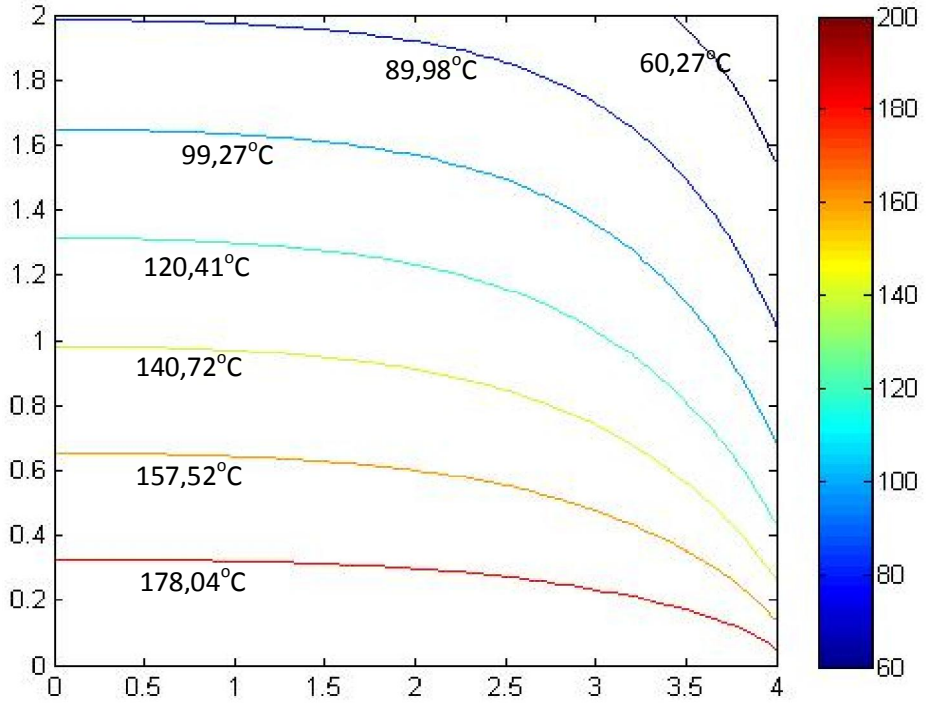
και επομένως

$a_{ij} = -\left\{ \iint_{\Omega} \frac{\partial \varphi^i}{\partial x} \frac{\partial \varphi^j}{\partial x} dxdy + \iint_{\Omega} \frac{\partial \varphi^i}{\partial y} \frac{\partial \varphi^j}{\partial y} dxdy \right\}$, $b_i=0$, όπου i οι αριθμοί των εσωτερικών κόμβων του χωρίου Ω και j=1,2,…,N.

Έτσι λοιπόν με βάση τα παραπάνω και χρησιμοποιώντας διωνυμικές συναρτήσεις βάσης, εννέα κόμβους ανά στοιχείο και τρία σημεία Gauss για την ολοκλήρωση στο στοιχείο αναφοράς (η οποία γίνεται με τη μέθοδο Gauss Quadrature) κατασκευάζουμε κώδικα σε Fortran 90 ο οποίος επιλύει το παραπάνω πρόβλημα. Τα αποτελέσματα που παίρνουμε για τη θερμοκρασία στο χωρίο Ω φαίνονται στα παρακάτω γραφήματα:



Εικόνα 3.2. Κατανομή θερμοκρασίας στο δι-διάστατο χωρίο. Η θερμοκρασία φαίνεται στον κάθετο άξονα και χρωματικά.



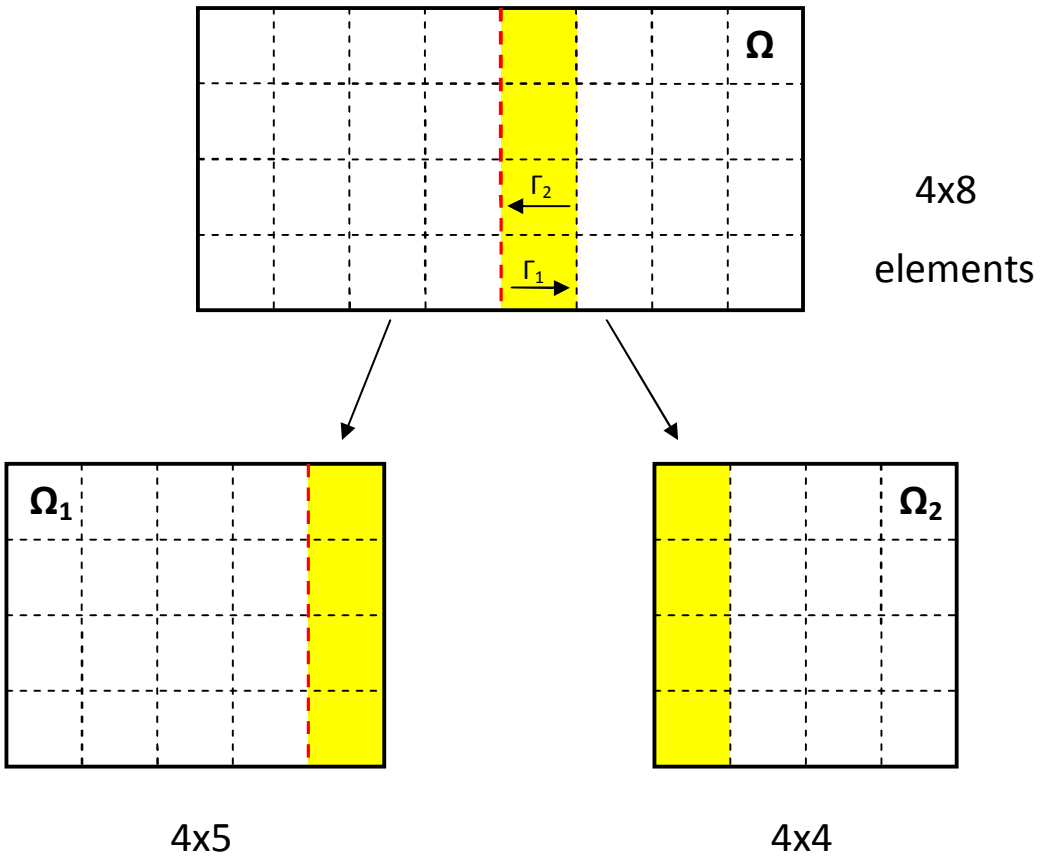
Εικόνα 3.3. Κατανομή ισοθερμοκρασιακών καμπυλών στο χωρίο.

3.3 Υλοποίηση της μεθόδου Alternating Schwarz σε σειριακό κώδικα

Όπως είδαμε στο προηγούμενο κεφάλαιο η μέθοδος Alternating Schwarz αφορά την επίλυση ενός προβλήματος συνοριακών τιμών με διαχωρισμό του χωρίου σε επιμέρους υποχωρία και την επίλυση των επιμέρους προβλημάτων συνοριακών τιμών που δημιουργούνται. Για τα καινούργια σύνορα που δημιουργούνται στα υποχωρία και προέκυψαν από το εσωτερικό του αρχικού χωρίου (Γ_i) θα χρησιμοποιήσουμε συνοριακές συνθήκες τύπου Dirichlet.

Όπως βλέπουμε και στο παρακάτω σχήμα το χωρίο του προβλήματός μας είναι αρκετά απλό και τα πλέγματα των υποχωρίων μας στην περιοχή επικάλυψης ταυτίζονται. Άρα αυτό απλοποιεί την μέθοδο μας σε σχέση με την γενικότερη περίπτωση που είχε μελετήσει ο Schwarz.

Αυτή η απλοποίηση ήταν κάτι το αναμενόμενο καθώς ο στόχος μας διαφέρει από το στόχο που είχε ο Schwarz όταν ανέπτυξε αυτή τη μέθοδο. Αρχικός στόχος της ήταν να βοηθήσει την επίλυση προβλημάτων συνοριακών τιμών σε δύσκολες γεωμετρίες χωρίων διαχωρίζοντας τα σε υποχωρία με ευκολότερη γεωμετρία. Εμείς, όμως, θα προσπαθήσουμε να εκμεταλλευτούμε τη μέθοδο με απώτερο στόχο, να παραλληλοποιήσουμε τον αλγόριθμο και να διαμοιράσουμε την επίλυση σε επιμέρους επεξεργαστές. Γι' αυτό το λόγο δεν σκοπεύουμε να ασχοληθούμε με δύσκολες γεωμετρίες αλλά να επικεντρωθούμε στην ταχύτητα επίλυσης.



Εικόνα 3.4. Παράδειγμα διαχωρισμού ενός χωρίου σε δύο υποχωρία.

Κωδικοποιούμε λοιπόν τα βήματα του αλγορίθμου της μεθόδου αυτής για την περίπτωση των δύο υποχωρίων (με n συμβολίζεται ο αριθμός της επανάληψης) [Smith et al (1996)]:

1. Διαχωρίζω το αρχικό χωρίο σε δύο επιμέρους υποχωρία.
2. Κάνω μία αρχική εκτίμηση για το u_2 (u_2^0) στα σημεία του Ω_2 που συμπίπτουν με το εσωτερικό σύνορο Γ_1 του Ω_1.
3. Επιλύω το πρόβλημα:

$$Lu_1^n = f \quad \text{στο } \Omega_1 \qquad (3.7\alpha)$$

$$Bu_1^n = g \quad \text{στο } \partial\Omega_1 \setminus \Gamma_1 \qquad (3.7\beta)$$

$$u_1^n = u_2^{n-1}\Big|_{\Gamma_1} \quad \text{στο } \Gamma_1 \qquad (3.7\gamma)$$

4. Επιλύω το πρόβλημα:

$$Lu_2^n = f \quad \text{στο } \Omega_2 \qquad (3.8\alpha)$$

$$Bu_2^n = g \ \text{στο} \ \partial\Omega_2 \setminus \Gamma_2 \qquad (3.8\beta)$$

$$u_2^n = u_1^n\big|_{\Gamma_2} \ \text{στο} \ \Gamma_2 \qquad (3.8\gamma)$$

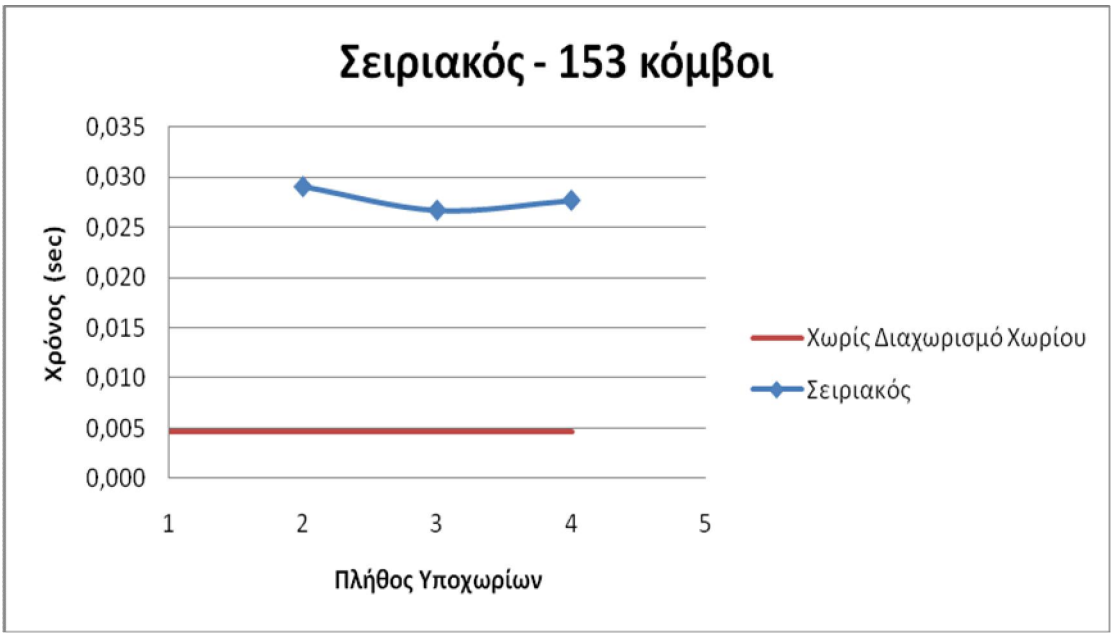
5. Επαναλαμβάνω τα βήματα 2 έως 3 μέχρι οι τιμές των αποτελεσμάτων της επικαλυπτόμενης περιοχής των γειτονικών χωρίων να αποκτήσουν μία επιθυμητά μικρή απόκλιση.

Κατασκευάσαμε λοιπόν σειριακό κώδικα που υλοποιεί τον παραπάνω αλγόριθμο και διακριτοποιεί τα προβλήματα των επιμέρους υποχωρίων χρησιμοποιώντας την μέθοδο πεπερασμένων στοιχείων όπως περιγράψαμε παραπάνω.

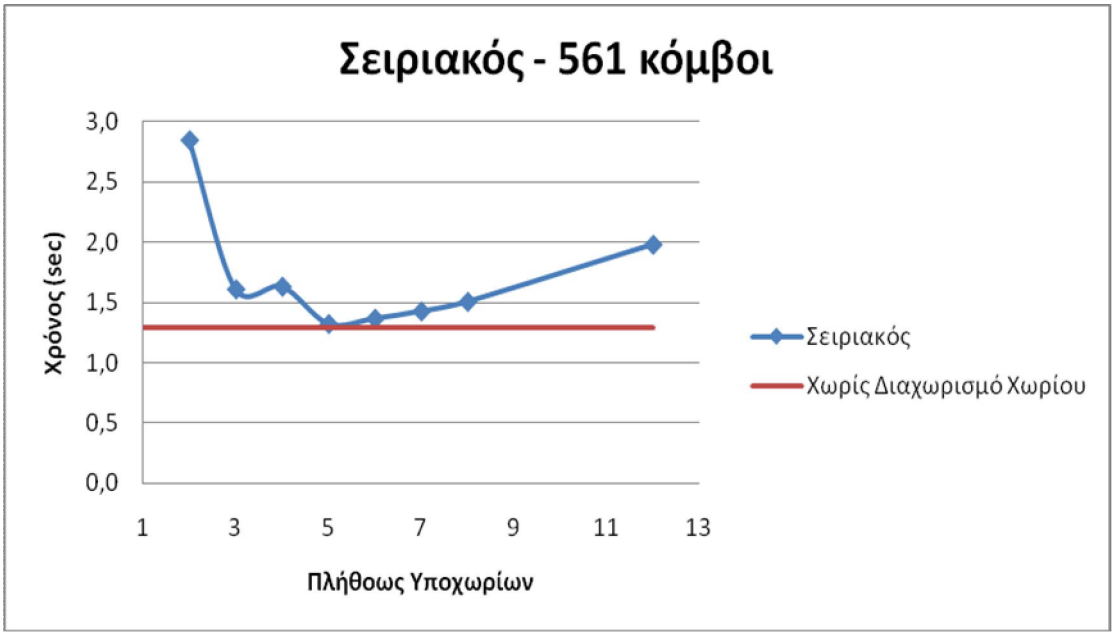
Παρατηρήσαμε ότι ο αλγόριθμος αυτός συγκλίνει ανεξαρτήτως της αρχικής εκτίμησης για τα εσωτερικά σύνορα και μάλιστα ο αριθμός των επαναλήψεων δεν επηρεάζεται ιδιαίτερα από αυτήν την εκτίμηση.

Χάρη σε αυτή την παρατήρηση καταφέραμε να γενικεύσουμε την περίπτωση των δύο υποχωρίων [Badea (1989)] ώστε ο κώδικας να λαμβάνει ως μεταβλητή το πλήθος των υποχωρίων που επιθυμούμε να διαχωρίσουμε το χωρίο μας, χρησιμοποιώντας ως αρχική εκτίμηση για όλους τους κόμβους των εσωτερικών συνόρων θερμοκρασία μηδέν. Έτσι μπόρεσε να γίνει ευκολότερα η ανάλυση της επίδρασης του πλήθους των υποχωρίων στην ταχύτητα εκτέλεσης του κώδικα.

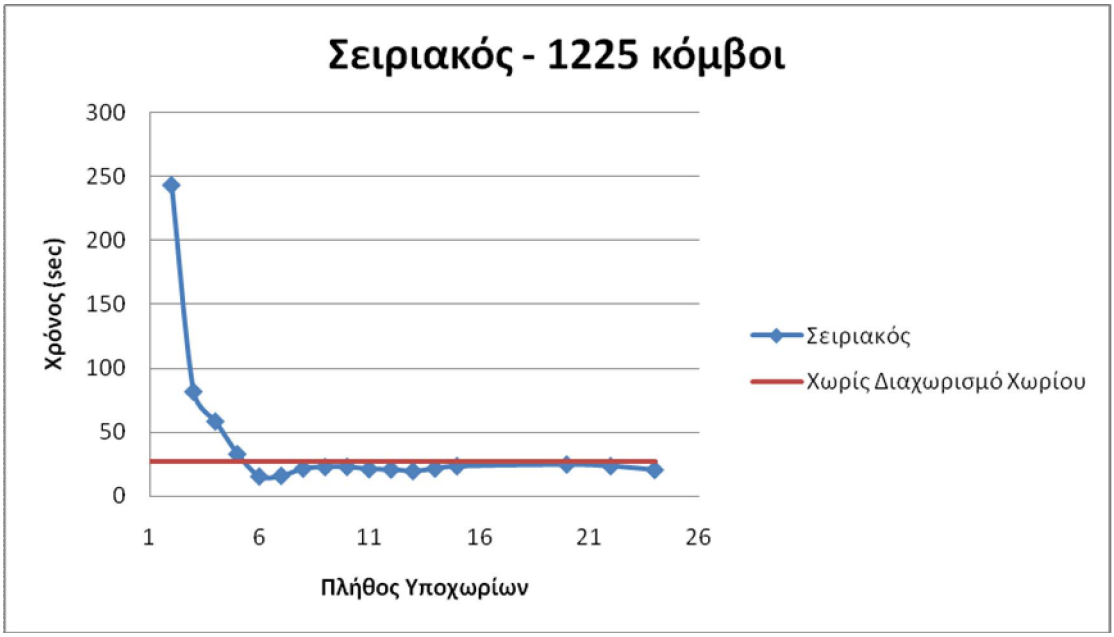
3.4 Αποτελέσματα σειριακού κώδικα



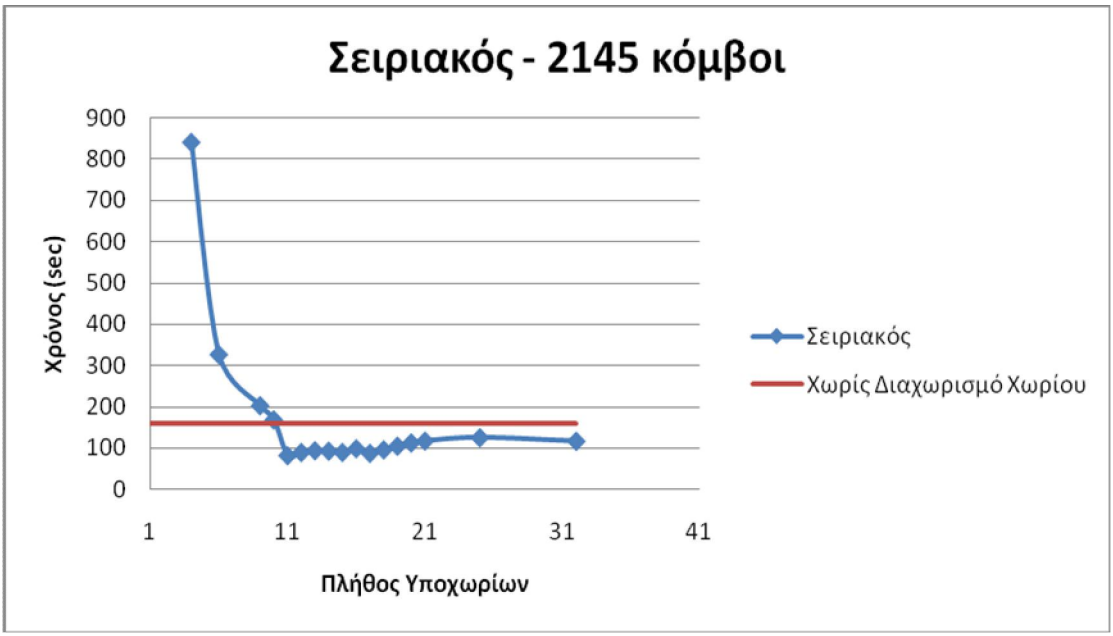
Εικόνα 3.5. Χρόνοι επίλυσης σειριακού κώδικα για 153 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



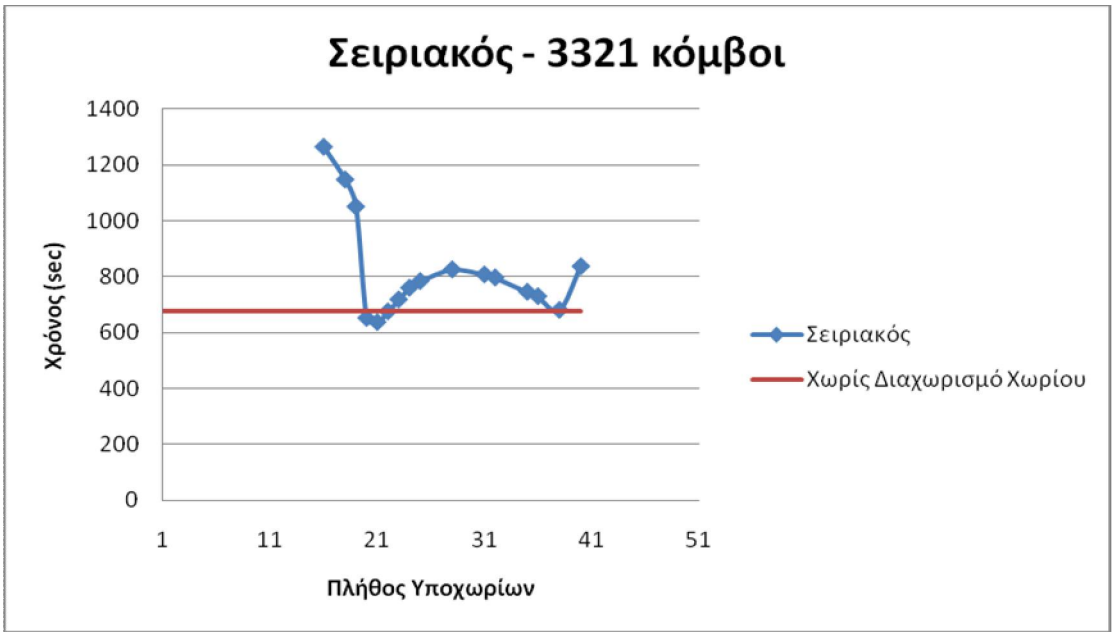
Εικόνα 3.6. Χρόνοι επίλυσης σειριακού κώδικα για 561 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



Εικόνα 3.7. Χρόνοι επίλυσης σειριακού κώδικα για 1225 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



Εικόνα 3.8. Χρόνοι επίλυσης σειριακού κώδικα για 2145 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



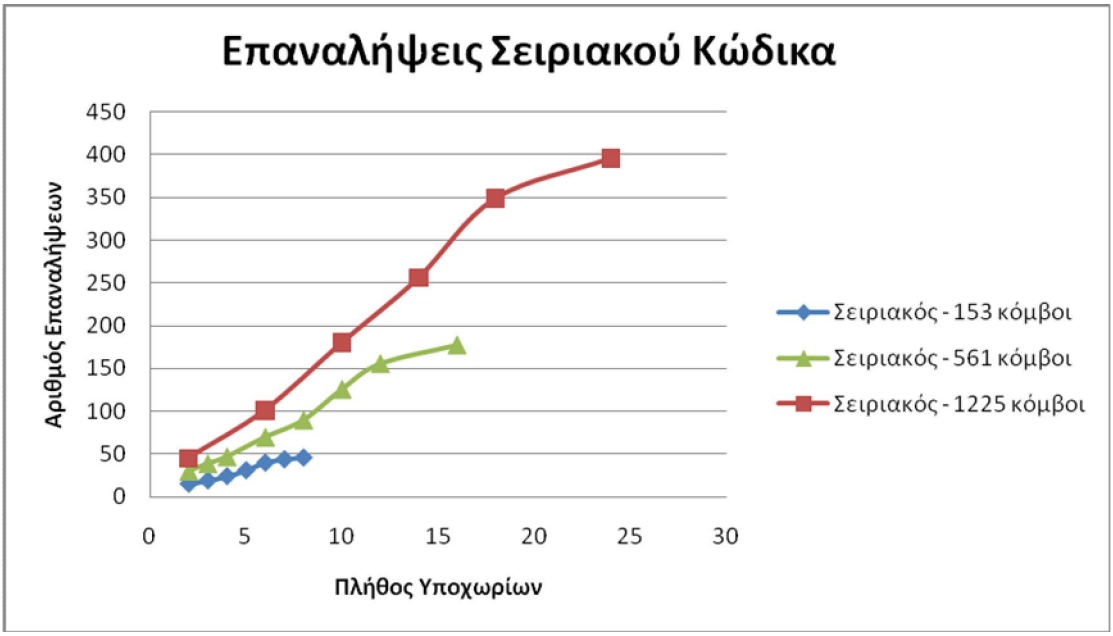
Εικόνα 3.9. Χρόνοι επίλυσης σειριακού κώδικα για 3321 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.

Από τα παραπάνω διαγράμματα μπορούμε να βγάλουμε κάποια χρήσιμα συμπεράσματα για τον σειριακό κώδικά μας:

- Για μικρά προβλήματα (153, 561 κόμβοι) η χρήση του σειριακού κώδικα για την επίλυση του προβλήματος δεν έχει νόημα γιατί ο χρόνος εκτέλεσης του απλού κώδικα πεπερασμένων στοιχείων είναι μικρότερος και επίσης ο χρόνος εκτέλεσης τέτοιων προβλημάτων σε έναν επεξεργαστή είναι αρκετά μικρός όπως επίσης και η απαίτηση τους σε μνήμη (με βάση τις δυνατότητες των σύγχρονων υπολογιστικών συστημάτων).
- Για λίγο μεγαλύτερα προβλήματα (1225 κόμβοι) η χρήση του κώδικα αρχίζει να έχει νόημα αφού βλέπουμε ότι για περισσότερα από πέντε υποχωρία η λύση επιταχύνεται. Αυτό συμβαίνει λόγω της μείωσης του μεγέθους των επιμέρους γραμμικών συστημάτων. Λόγω της πολυπλοκότητας της απαλοιφής Gauss ($O(n^3)$) η λύση επιταχύνεται παρ' όλο που αύξηση των υποχωρίων συνεπάγεται αύξηση του πλήθους των γραμμικών συστημάτων και του πλήθους των επαναλήψεων (εικόνα 3.10). Αυτό είναι εντονότερο στον παράλληλο κώδικα καθώς τα επιμέρους γραμμικά συστήματα κατανέμονται στους επεξεργαστές (βλέπε παράγραφο 3.6.2). Η μέγιστη εξοικονόμηση χρόνου που πετύχαμε στην συγκεκριμένη περίπτωση ήταν περίπου 43% ενώ η

μέση εξοικονόμηση χρόνου επιλέγοντας περισσότερα των πέντε υποχωρίων είναι περίπου 19%.

- Στο πρόβλημα των 2145 κόμβων το κέρδος μας σε χρόνο είναι περισσότερο προφανές αλλά για να επιτευχθεί αυτό πρέπει να διαχωρίσουμε το χωρίο μας σε περισσότερα από δέκα υποχωρία, περισσότερα δηλαδή από πριν. Η μέγιστη εξοικονόμηση χρόνου που πετύχαμε εδώ ήταν περίπου 47% ενώ η μέση εξοικονόμηση χρόνου επιλέγοντας περισσότερα των δέκα υποχωρίων είναι περίπου 31%.
- Όταν όμως πάμε σε ακόμα μεγαλύτερα προβλήματα (3321 κόμβοι) στα οποία η χρήση του κώδικα αποκτάει μεγαλύτερο νόημα βλέπουμε ότι **το πλεονέκτημα της εξοικονόμησης χρόνου που είχαμε στις προηγούμενες περιπτώσεις τώρα χάνεται**. Αυτό μπορεί να εξηγηθεί αν ρίξουμε μια ματιά στο πλήθος των επαναλήψεων. Βλέπουμε στο παρακάτω γράφημα ότι το πλήθος των επαναλήψεων που απαιτείται για να φτάσουμε στην επιθυμητή ακρίβεια αυξάνεται με την αύξηση του πλήθους των κόμβων αλλά και με την αύξηση των υποχωρίων για σταθερό αριθμό κόμβων. Αν λάβουμε επίσης υπόψιν το συμπέρασμα απ' τις δύο προηγούμενες παρατηρήσεις, ότι δηλαδή όσο αυξάνεται το μέγεθος του προβλήματος απαιτείται μεγαλύτερος αριθμός υποχωρίων για να πάρουμε λύσεις γρηγορότερα απ' τον αντίστοιχο κώδικα πεπερασμένων στοιχείων, τότε καταλαβαίνουμε ότι είναι φυσιολογικό αυτό που συμβαίνει γιατί για μεγάλα προβλήματα οι επαναλήψεις που απαιτούνται αυξάνονται κατά πολύ.



Εικόνα 3.10. Πλήθος επαναλήψεων σειριακού κώδικα για διάφορους αριθμούς κόμβων.

- Πέρα απ' το χρόνο όμως, που τελικά δεν εξοικονομείται πάντα, πρέπει να τονίσουμε ότι το μεγάλο πλεονέκτημα του σειριακού μας κώδικα είναι η **μείωση της απαίτησης σε μνήμη**. Με την επιλογή του μέγιστου δυνατού αριθμού υποχωρίων (που ουσιαστικά εδώ είναι ίσος με τον αριθμό των στοιχείων στον άξονα χ, λόγω του ότι το χωρίο επιλέξαμε να διαχωρίζεται σε υποχωρία μόνο στον οριζόντιο άξονα - ας ονομάσουμε τον αριθμό αυτό n_{max}) καταφέρνουμε να μειώσουμε την απαίτηση σε μνήμη των μονοδιάστατων πινάκων δια $n_{max}/2$, ενώ για την αποθήκευση του πίνακα A του γραμμικού συστήματος (που καταλαμβάνει και το μεγαλύτερο χώρο) η απαίτηση μειώνεται δια $n_{max}^2/4$. Αυτό σημαίνει ότι για παράδειγμα στο πρόβλημα των 3321 κόμβων στον κώδικα πεπερασμένων στοιχείων, όταν χρησιμοποιούμε αριθμούς διπλής ακρίβειας 8 bytes (double precision) το μέγεθος του πίνακα A (που καταλαμβάνει και το μεγαλύτερο μέρος της μνήμης) μειώνεται **από τα 88 MB στα 220 KB**.

3.5 Κατασκευή παράλληλου κώδικα

Ο παραπάνω αλγόριθμος μετατράπηκε σε παράλληλο κώδικα που χρησιμοποιεί τη βιβλιοθήκη MPI προκειμένου να μοιράσει την κατασκευή και επίλυση των γραμμικών συστημάτων του κάθε υποχωρίου σε διαφορετικό επεξεργαστή.

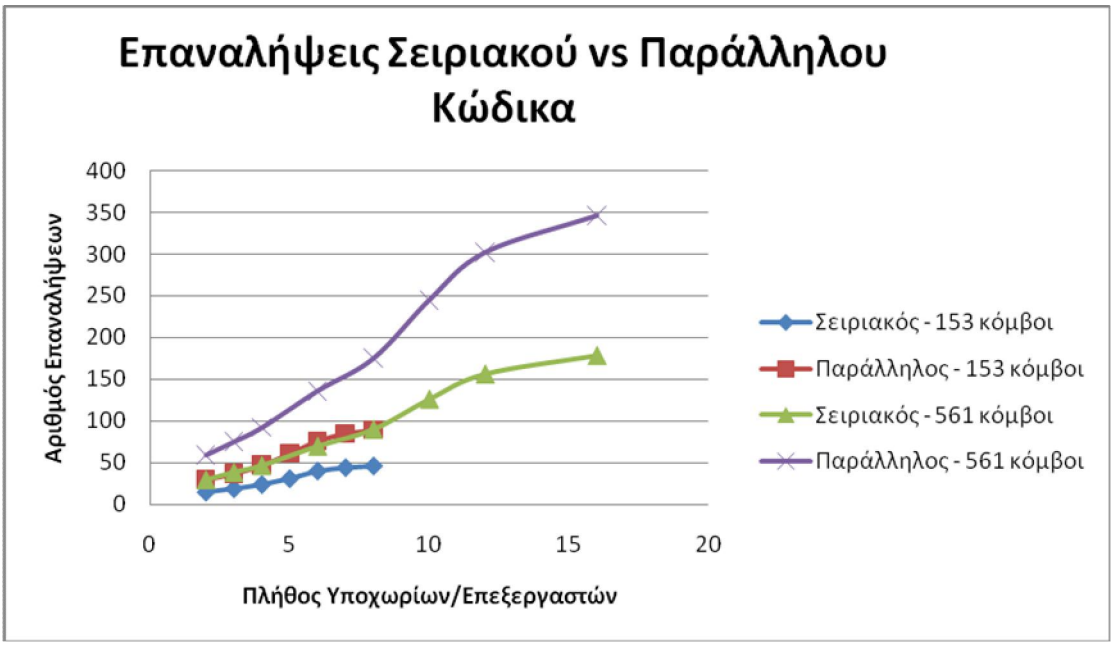
Επομένως, ουσιαστικά όπου βλέπουμε αναφορά σε αριθμό επεξεργαστών παρακάτω, αυτός θα είναι ίσος με τον αριθμό των υποχωρίων και αντίστροφα.

Το κύριο πρόβλημα όμως που αντιμετωπίσαμε κατά την μετατροπή αυτή ήταν ότι εξ' αιτίας της εξίσωσης (3.8γ) σε κάθε επανάληψη το πρόβλημα που επιλύεται στο κάθε υποχωρίο εξαρτάται απ' το αριστερό του υποχωρίο, άρα κάθε υποεπανάληψη εξαρτάται από την προηγούμενη υποεπενάληψη. Αυτό δημιουργεί μία σειριακότητα που δεν είναι αποδεκτή και αν δεν γίνει κάποια αλλαγή τότε η χρήση περισσότερων του ενός επεξεργαστή ουσιαστικά δεν θα έχει νόημα και δεν θα προσφέρει τίποτα.

Η αλλαγή που αναγκαστήκαμε να κάνουμε προκειμένου να παραλληλοποιηθεί ήταν η εξίσωση (3.8γ) τελικά να γίνει: $u_2^n = u_1^{n-1}\Big|_{\Gamma_2}$. Δηλαδή οι τιμές της θερμοκρασίας για τα εσωτερικά σύνορα θα γίνονται ίσες με την τιμή των αντίστοιχων κόμβων του αριστερού τους υποχωρίου, όπως αυτές ήταν στην προηγούμενη επανάληψη. Αυτό γενικεύεται για περισσότερα των δύο υποχωρίων και έτσι σε κάθε επανάληψη οι υπολογισμοί στο κάθε υποχωρίο μπορούν να ξεκινάνε ανεξάρτητα απ' τα διπλανά του.

Ο αλγόριθμός συνεχίζει να συγκλίνει παρά αυτή την αλλαγή αλλά η επίπτωση που είχαμε στον αριθμό των επαναλήψεων ήταν σχεδόν να διπλασιαστούν σε σχέση με τον σειριακό κώδικα για ίδιο αριθμό κόμβων, όπως φαίνεται και παρακάτω.

3.5.1 Επίδραση στο πλήθος επαναλήψεων



Εικόνα 3.11 Πλήθος επαναλήψεων σειριακού και παράλληλου κώδικα για 153 και 561 κόμβους.

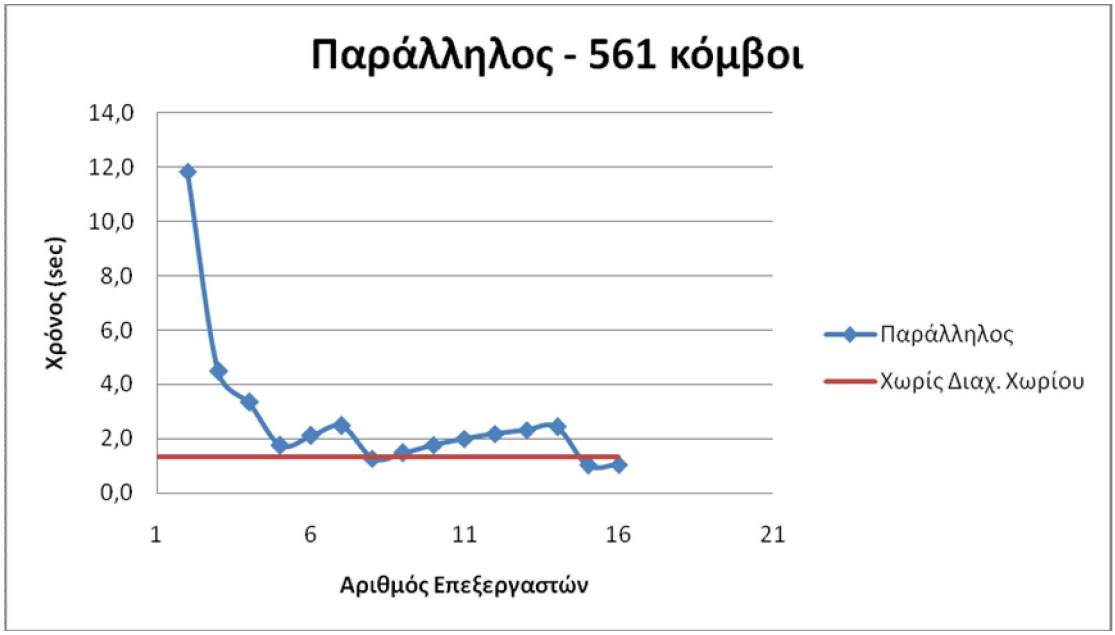
Εκτός τη μεταβολή των επαναλήψεων λόγω της παραλληλοποίησης του κώδικα μπορούμε να παρατηρήσουμε και αύξηση των επαναλήψεων με αύξηση του αριθμού των κόμβων (που είναι ίσοι με το πλήθος των γραμμικών εξισώσεων) αλλά και για σταθερό αριθμό κόμβων με αύξηση του αριθμού των υποχωρίων. Επίσης παρατηρούμε ότι αν κρατήσουμε σταθερό τον αριθμό των υποχωρίων και αυξήσουμε το πλήθος των κόμβων τότε για μεγάλο πλήθος υποχωρίων η αύξηση των επαναλήψεων που παρατηρείται είναι μεγαλύτερη. Άρα ίσως μπορούμε να αναμένουμε ότι ο κώδικάς μας δεν θα δίνει πολύ καλά αποτελέσματα χρόνου εκτέλεσης για μεγάλα προβλήματα. Κάποια ενδεικτικά αποτελέσματα πλήθους επαναλήψεων φαίνονται στον παρακάτω πίνακα:

Κόμβοι	Υποχωρία	Σειριακός Κώδικας	Παράλληλος Κώδικας
153	2	15	30
	4	24	47
	8	46	89
561	2	30	59
	8	90	175
	16	178	346
1225	2	45	87
	14	256	497
	24	396	769

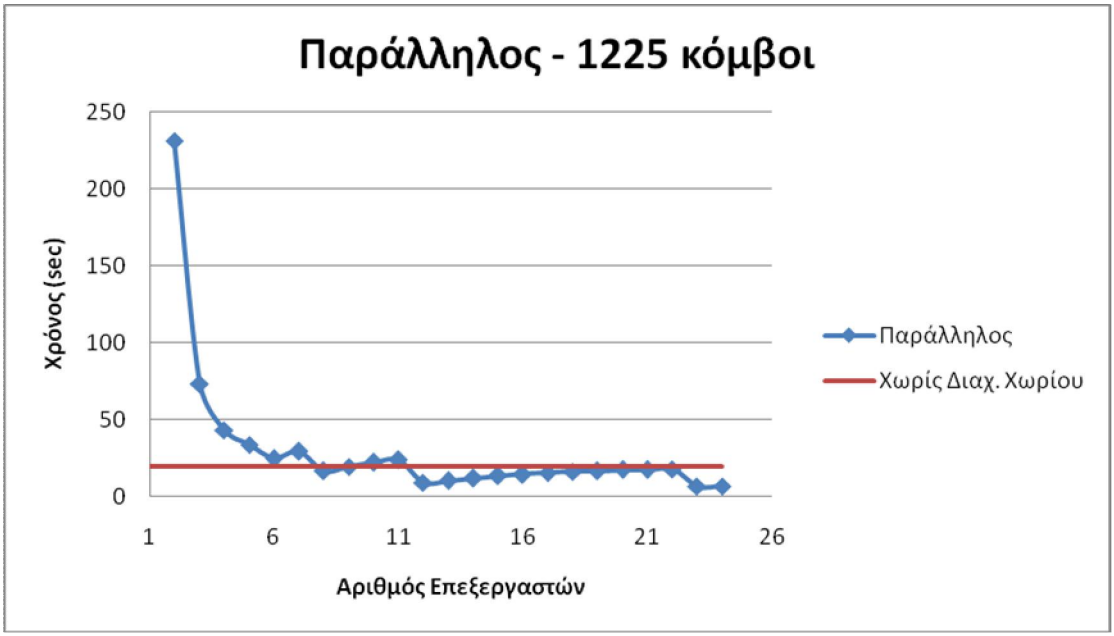
Πίνακας 3.1. Πλήθος επαναλήψεων για διάφορους αριθμούς κόμβων και υποχωρίων.

3.6 Αποτελέσματα παράλληλου κώδικα

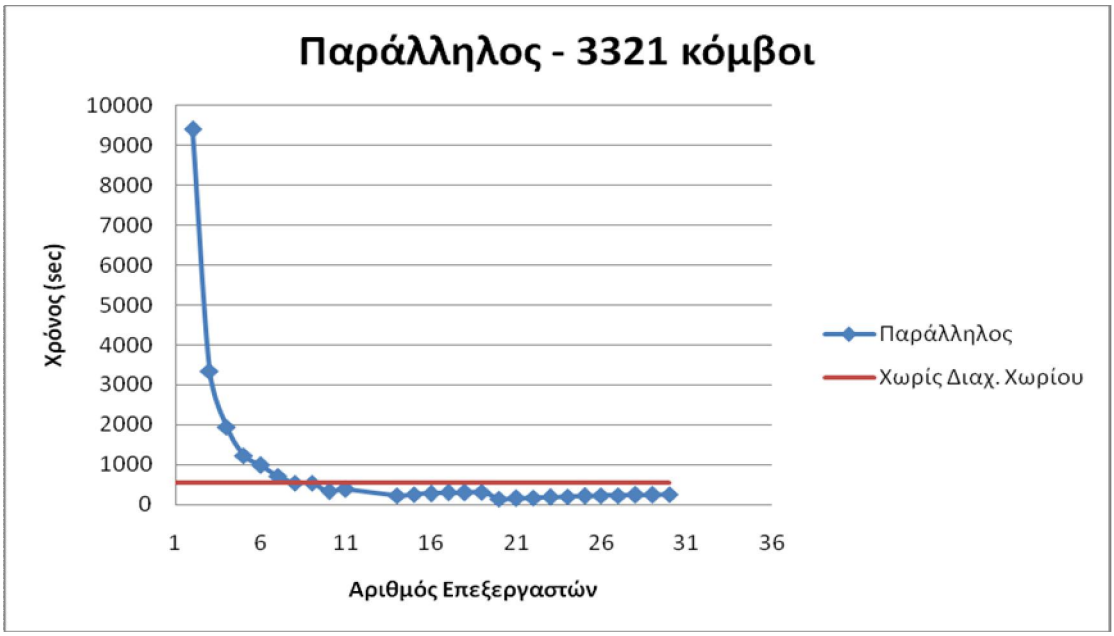
3.6.1 Χρόνοι εκτέλεσης



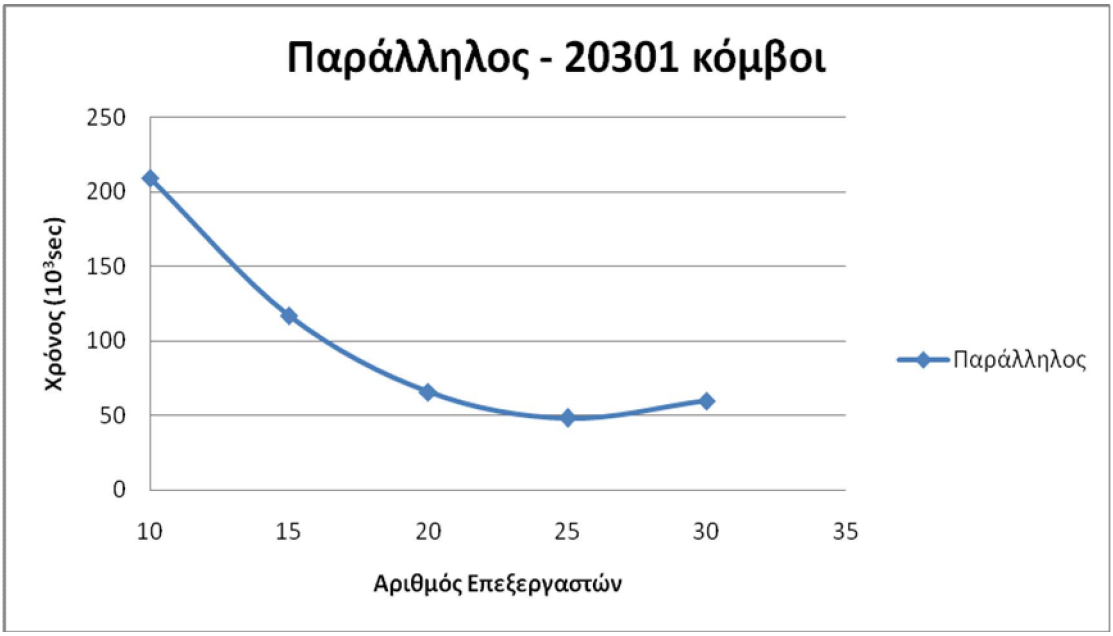
Εικόνα 3.12. Χρόνοι επίλυσης παράλληλου κώδικα για 561 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



Εικόνα 3.13. Χρόνοι επίλυσης παράλληλου κώδικα για 1225 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



Εικόνα 3.14. Χρόνοι επίλυσης παράλληλου κώδικα για 3321 κόμβους και σύγκριση με τον απλό κώδικα πεπερασμένων στοιχείων.



Εικόνα 3.15 Χρόνοι επίλυσης παράλληλου κώδικα για 20301 κόμβους.

Τα συμπεράσματα που βγάζουμε από τα παραπάνω διαγράμματα είναι τα εξής:

- Για πολύ μικρά προβλήματα (561 κόμβοι) ο διαχωρισμός του χωρίου δεν έχει νόημα. Παρόλο που μπορεί να εξοικονομήσουμε χρόνο σε σχέση με το σειριακό κώδικα πεπερασμένων στοιχείων (έως 22% περίπου), αν δεν γίνει σωστά η επιλογή επεξεργαστών μπορεί και να μην εξοικονομηθεί χρόνος.
- Για μεγαλύτερα προβλήματα (1225 και 3321 κόμβοι) η χρήση των παράλληλων επεξεργαστών έχει νόημα και μπορούμε να εξοικονομήσουμε χρόνο επίλυσης έως 67 και 74% αντίστοιχα σε σχέση με τον απλό κώδικα πεπερασμένων στοιχείων.
- Όταν πάμε σε προβλήματα της τάξης των 20.000 κόμβων ο κώδικας πεπερασμένων στοιχείων δεν μπορεί πλέον να επιλυθεί σε έναν επεξεργαστή της συστοιχίας Pegasus λόγω μη επαρκούς μνήμης (βλέπε παράγραφο 1.3.1), έτσι δεν έχουμε ένα μέτρο σύγκρισης όπως στα προηγούμενα διαγράμματα. Αντιλαμβανόμαστε όμως ότι χρόνοι επίλυσης είναι πολύ μεγάλοι: για την περίπτωση των δέκα επεξεργαστών είναι περίπου 58 ώρες, για είκοσι και τριάντα επεξεργαστές είναι 18 και 16,5 ώρες αντίστοιχα.
- Στα παραπάνω διαγράμματα, και ειδικά στα τρία πρώτα που οι μετρήσεις είναι πιο πυκνές παρατηρούμε κοινή συμπεριφορά στους χρόνους με την αύξηση των υποχωρίων (άρα και των επεξεργαστών). Δηλαδή βλέπουμε μία απότομη πτώση του χρόνου αρχικά και μετά κάποιες απότομες πτώσεις ανά

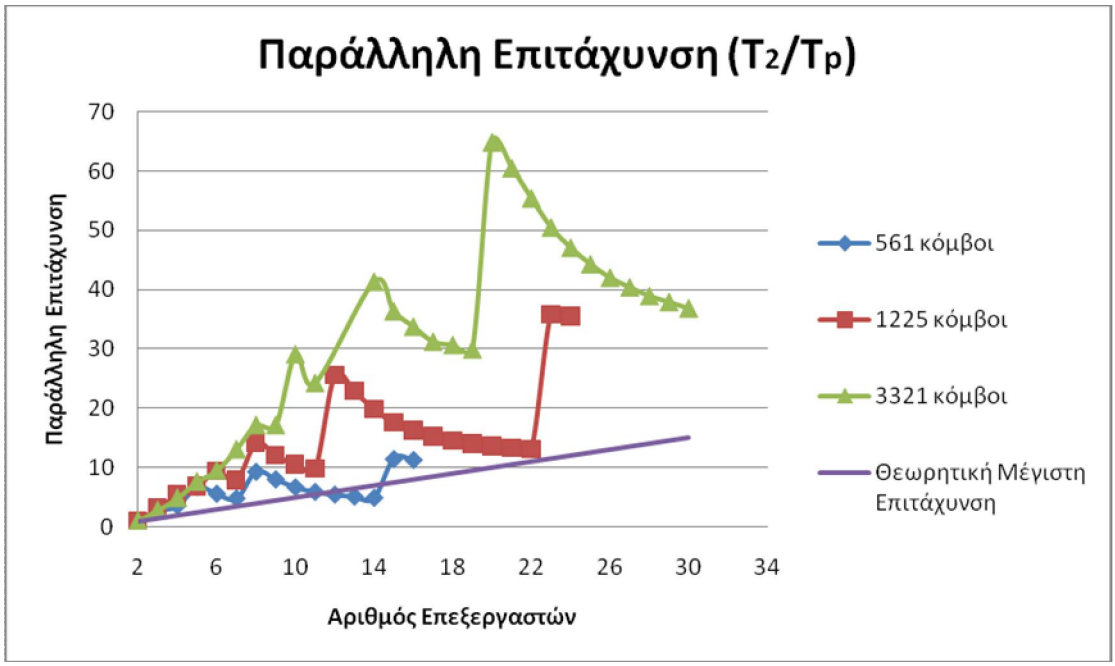
διαστήματα που ακολουθούνται από αργές ανόδους. Αυτή η διακύμανση δεν είναι τυχαία και μπορεί να εξηγηθεί από τον τρόπο που το χωρίο διαχωρίζεται:

- Η αρχική απότομη πτώση οφείλεται στο ότι όταν μιλάμε για μικρό αριθμό επεξεργαστών και η προσθήκη ενός ακόμη έχει μεγάλη επίδραση στη μείωση του προβλήματος που επιλύει ο κάθε επεξεργαστής.
- Η διακύμανση που ακολουθεί της απότομης πτώσης έχει και αυτή εξήγηση. Γενικά πρέπει να καταλάβουμε ότι με τον τρόπο που διαχωρίζεται το χωρίο η αύξηση των υποχωρίων και άρα και των επεξεργαστών δεν μειώνει πάντα το πρόβλημα που επιλύουν οι επιμέρους επεξεργαστές. Για παράδειγμα ας πάμε στην περίπωση των 561 κόμβων του παράλληλου κώδικα. Εδώ ο αριθμός των στοιχείων του αρχικού χωρίου ήταν δεκάξι στον χ άξονα και οχτώ στον y. Όταν διαμοιράζουμε το χωρίο σε οχτώ υποχωρία (το χωρίο διαμοιράζεται κάθετα άρα ουσιαστικά ο κάθε επεξεργαστής αναλαμβάνει κάποιες στήλες στοιχείων) τότε ουσιαστικά το κάθε υποχωρίο περιέχει τρεις στήλες στοιχείων για επίλυση (δύο που του αντιστοιχούν και μία που επικαλύπτει το δεξί υποχωρίο) εκτός από το τελευταίο που περιέχει δύο. Όσο αυξάνουμε τα υποχωρία σε εννέα, δέκα κτλ. πάντα υπάρχουν κάποιοι επεξεργαστές που αναλαμβάνουν να επιλύσουν τρεις σειρές στοιχείων στον άξονα των χ και αυτοί οι επεξεργαστές καθυστερούν και τους υπόλοιπους οι οποίοι έχουν να λύσουν μικρότερο πρόβλημα, δηλαδή παρουσιάζεται **ανισοκατανομή υπολογιστικού φορτίου** (load imbalance). Άρα η αύξηση των υποχωρίων δεν έχει επίδραση στην μείωση του μεγέθους του προβλήματος των επιμέρους επεξεργαστών και επειδή επιπλεόν η αύξηση των υποχωρίων αύξησε το πλήθος των επαναλήψεων αλλά και την επικοινωνία μεταξύ των επεξεργαστών, παρατηρούμε αυτή την σταδιακή αύξηση στην ταχύτητα εκτέλεσης. Όταν όμως φτάνουμε στα δεκαπέντε υποχωρία βλέπουμε απότομη μείωση χρόνου γιατί για αυτό το πλήθος υποχωρίων ουσιαστικά όλα τα υποχωρία περιλαμβάνουν δύο σειρές στοιχείων στον άξονα των χ, άρα μειώθηκε το υπολογιστικό φορτίο όλων των επεξεργαστών. Η

λύση για την μείωση του χρόνου στην περίπτωση της ανισοκατανομής υπολογιστικού φορτίου ίσως είναι οι ασύγχονες επαναλήψεις [Chau et al (2007), Frommer & Szyld (2000)].

- Να διευκρινήσουμε σε αυτό το σημείο ότι ο παράλληλος κώδικας δεν μειώνει την συνολική απαίτηση σε μνήμη του συστήματος όπως κάνει ο σειριακός, αλλά μειώνει την επιμέρους απαίτηση σε κάθε τοπική μνήμη σε συστήματα κατανεμημένης μνήμης.

3.6.2 Παράλληλη Επιτάχυνση



Εικόνα 3.16. Παράλληλη επιτάχυνση με βάση το χρόνο επίλυσης για δύο επεξεργαστές, για διάφορους αριθμούς κόμβων σε σύγκριση με τη θεωρητικά μέγιστη.

Για την παράλληλη επιτάχυνση παρατηρούμε ότι είναι πολύ μεγαλύτερη από τη θεωρητικά μέγιστη. Σε τέτοιες περιπτώσεις λέμε ότι έχουμε υπεργραμμική παράλληλη επιτάχυνση (super linear speedup). Αυτό είναι φυσιολογικό όμως γιατί ο επιλύτης που χρησιμοποιούμε (απαλοιφή Gauss) έχει πολυπλοκότητα $O(n^3)$ [I6]. Αυτό σημαίνει ότι όταν το μέγεθος του γραμμικού συστήματος μειωθεί κατά k τότε οι πράξεις που πρέπει να γίνουν απ' τον υπολογιστή για την επίλυσή του μειώνονται κατά k^3.

Με την αύξηση του μεγέθους του προβλήματος το κομμάτι της επίλυσης των γραμμικών συστημάτων απ' τους επεξεργαστές καταλαμβάνει όλο και μεγαλύτερο

ποσοστό απ' τον υπολογιστικό χρόνο λόγω της πολυπλοκότητας της απαλοιφής Gauss. Αυτός είναι ο λόγος που ο ρυθμός αύξησης της παράλληλης επιτάχυνσης αυξάνεται με την αύξηση των κόμβων του χωρίου.

Οι διακυμάνσεις που παρατηρούνται οφείλονται στην ανισοκατανομή του υπολογιστικού φορτίου για κάποιες τιμές υποχωρίων, όπως αναφέραμε παραπάνω.

3.7 Συμπεράσματα

- Τα αποτελέσματα που παίρνουμε από την παραπάνω μέθοδο γενικά δεν μπορούμε να τα χαρακτηρίσουμε ως ενθαρρυντικά.
    - Ο αριθμός των επαναλήψεων που παίρνουμε από τη μέθοδο μας είναι αρκετά μεγάλος (πίνακας 3.1). Μία σύγκριση που γίνεται ανάμεσα στην Additive, την Multiplicative και την Alternating Schwarz στο [Smith et al (1996)] μας βοηθάει να συμπεράνουμε ότι πρέπει να εγκαταλείψουμε την Alternating Schwarz αν θέλουμε να ξεφύγουμε από την μεγάλη επίδραση που έχει ο αριθμός των κόμβων και των υποχωρίων στο πλήθος των επαναλήψεων.
    - Οι χρόνοι που παίρνουμε δεν είναι καλοί και αυτό φαίνεται κυρίως στην περίπτωση που λύνουμε προβλήματα της τάξης των 20.000 κόμβων. Βέβαια, η χρήση της απαλοιφής Gauss η οποία δεν αντικαταστάθηκε με κάποιον καλύτερο επιλύτη έχει σημαντική συνεισφορά σε αυτό.
    - Η παράλληλη επιτάχυνση που προκύπτει από τον αλγόριθμο είναι πολύ καλή, πολύ καλύτερη από την θεωρητικά μέγιστη, αλλά δεν πρέπει να ξεχνάμε ότι η πολυπλοκότητα της απαλοιφής Gauss ήταν ο κύριος λόγος γι' αυτό.
- Πρότασή μας για τη βελτίωση του αριθμού των επαναλήψεων του παράλληλου κώδικα είναι να εισάγουμε τη λογική του σειριακού μέσα στον παράλληλο. Δηλαδή, ο κάθε επεξεργαστής να αναλαμβάνει περισσότερα του ενός υποχωρία τα οποία να είναι γειτονικά μεταξύ τους και να επιλύονται με σειριακό τρόπο σε κάθε επανάληψη από τον επεξεργαστή ενώ η κάθε ομάδα υποχωρίων θα επιλύεται παράλληλα ως προς τις υπόλοιπες. Έτσι θα περιοριστούν οι απαιτήσεις επικοινωνίας και θα μειωθούν οι επαναλήψεις. Το

αρνητικό όμως είναι ότι ό,τι αλλαγές και να κάνουμε στον παράλληλο κώδικα δεν θα καταφέρουμε να μειώσουμε τις επαναλήψεις κάτω από τις επαναλήψεις του σειριακού. Αυτό το όριο επαναλήψεων που μας θέτει ο σειριακός κώδικας είναι ίσως ο ισχυρότερος λόγος για να στραφούμε σε άλλους αλγορίθμους.

- Το θετικότερο στοιχείο από τα παραπάνω είναι ίσως η μείωση της απαίτησης σε μνήμη που καταφέραμε με τον σειριακό κώδικα. Αυτό μας δίνει ξεκάθαρα την δυνατότητα να λύνουμε προβλήματα σε έναν επεξεργαστή με μεγάλη απαίτηση σε μνήμη που δεν μπορούσαμε να λύσουμε με την απλή μέθοδο πεπερασμένων στοιχείων.
- Αν επιχειρήσουμε να βελτιστοποιήσουμε την παραπάνω μέθοδο ως προς την ταχύτητα επίλυσης τότε κάποιοι παράμετροι οι οποίοι μπορούν να αλλάξουν είναι:
    - Διαφορετικό ποσοστό επικάλυψης μεταξύ των υποχωρίων. Από τη βιβλιογραφία [Smith et al (1996)] ξέρουμε ότι η επικάλυψη περισσότερων κόμβων θα φέρει σύγκλιση σε λιγότερες επαναλήψεις αλλά θα μεγαλώσει και τα επιμέρους γραμμικά συστήματα. Άρα το αποτέλεσμα που θα έχω από αύξηση του της επικάλυψης μπορεί να είναι και θετικό.
    - Στα τεχνητά σύνορα (Γ_i) μπορούν να χρησιμοποιηθούν άλλες συνθήκες εκτός των Dirichlet [Kim et al (1996)], όπως συνθήκες συνέχειας.
    - Εναλλακτικό διαχωρισμό του χωρίου σε υποχωρία, όχι απαραίτητα κατά τη διεύθυνση του συστήματος συντεταγμένων.
    - Αν ζητούμενο είναι μόνο η ταχύτητα επίλυσης και όχι η ακρίβεια τότε μπορεί να γίνει και αύξηση της ανοχής του σφάλματος.
- Άποψη μας είναι ότι η συγκεκριμένη μέθοδος διαχωρισμού χωρίου μειονεκτεί έναντι άλλων και προτείνουμε στους ενδιαφερόμενους να στραφούν σε μεθόδους όπως Additive Schwarz πολλαπλών επιπέδων αλλά και μη-επικαλυπτόμενων υποχωρίων όπως η FETI κ.α.

4. Παράμετροι Κώδικα

4.1 Διαχωρισμός σε υποχωρία

Το σημαντικότερο πράγμα το οποίο πρέπει να μάθει ο χρήστης του κώδικα πριν τον χρησιμοποιήσει είναι ο τρόπος που διαχωρίζεται το χωρίο σε υποχωρία. Ο τρόπος αυτός είναι κοινός και για τον σειριακό και για τον παράλληλο κώδικα. Η υπορουτίνα xydiscr περιέχει τις εντολές για το διαχωρισμό του:

```
      nd=2        !number of subdomains in x
      if (nd>nex) then
            write(*,*) 'more domains than allowed'
      else if (nd<=1) then
            write(*,*) 'less domains than allowed'
      end if
!*** split into subdomains
      nexx=nex

      do i=1,nd-1
            nex1(i)=int(nexx/real(nd-i+1))+1
            nexx=nexx-(nex1(i)-1)
      end do
      nex1(nd)=nexx
```

Όπως βλέπουμε παραπάνω ο αριθμός των υποχωρίων ρυθμίζεται από τον χρήστη με την μεταβλητή nd. Ωστόσο εξ' αιτίας του ότι το υποχωρίο διαχωρίζεται κάθετα, δηλαδή σε στήλες στοιχείων (nex), υπάρχουν δύο περιορισμοί στον αριθμό του nd. Πρώτον, δεν μπορεί να υπερβαίνει το πλήθος των στοιχείων στα οποία έχουμε χωρίσει τον άξονα χ (nex). Δεύτερον, πρέπει να είναι οπωσδήποτε πάνω από ένα και αυτό γιατί αν πάρει την τιμή ένα ο αλγόριθμος της Alternating Schwarz ουσιαστικά εκφυλίζεται και πάμε σε ένα απλό κώδικα πεπερασμένων στοιχείων.

Το πλήθος των στηλών στοιχείων τα οποία περιέχει το κάθε υποχωρίο αποθηκεύονται στον πίνακα nex1 και κάθε στοιχείο του πίνακα περιέχει την πληροφορία για το αντίστοιχο υποχωρίο.

Η λογική του διαχωρισμού των στηλών στα υποχωρία είναι η εξής:

(α) Σε περίπτωση που τα διαχωρίσουμε σε αριθμό υποχωρίων που διαιρεί ακριβώς το πλήθος των στηλών όλα θα πάρουν στήλες ίσες με το πηλίκο της διαίρεσης συν ένα, λόγω της επικάλυψης, και το τελευταίο υποχωρίο (δεξιά) δεν θα χρειαστεί αυτή τη μία επιπλέον στήλη επικάλυψης.

(β) Σε περίπτωση που η ακέραια διαίρεση δεν δίνει μηδενικό υπόλοιπο τότε τις επιπλέον στήλες θα τις επιμεριστούν τα υποχωρία που βρίσκονται πιο δεξιά.

Παράδειγμα αν nex=8 και αποφασίσω ο αριθμός των υποχωρίων να είναι τέσσερα τότε το πρώτο, δεύτερο και τρίτο υποχωρίο θα πάρουν τρεις στήλες στοιχείων και το τέταρτο δύο. Ενώ αν αποφασίσω τα υποχωρία να είναι πέντε τότε το πρώτο και δεύτερο παίρνουν δύο στήλες στοιχείων, το τρίτο και τέταρτο παίρνουν τρεις και το πέμπτο παίρνει δύο στήλες. Άρα βλέπουμε με το παραπάνω παράδειγμα πώς προκύπτει η ανισοκατανομή υπολογιστικού φορτίου όπως την περιγράψαμε στην παράγραφο 3.6.1, καθώς ο διαχωρισμός από τέσσερα σε πέντε υποχωρία δεν μειώνει το υπολογιστικό φορτίο κάποιων επεξεργαστών και αυτό δημιουργεί καθυστέρηση σε όλους τους επεξεργαστές.

4.2 Σειριακός Κώδικας

Ο αλγόριθμός που παρουσιάσαμε στην 3.3 υλοποιήθηκε προγραμματιστικά ως εξής:

```
!*** initial guess for internal boundary conditions
            if (n==1) then
                  do i=np1(n)-nny+1,np1(n)
                        r1(i)=0.    !guess essential conditions
                        do j=1,np1(n)
                              sk1(i,j)=0.
                        end do
                        sk1(i,i)=1.
                  end do
            else if (n==nd) then
                  do i=1,nny
                        r1(i)=u(np1(n-1)-3*nny+i,n-1)
                        do j=1,np1(n)
                              sk1(i,j)=0.
                        end do
                        sk1(i,i)=1.
                  end do
            else
                  do i=np1(n)-nny+1,np1(n)
                        r1(i)=0.    !guess essential conditions
                        do j=1,np1(n)
                              sk1(i,j)=0.
                        end do
```

```
                        sk1(i,i)=1.
                  end do
                  do i=1,nny
                        r1(i)=u(np1(n-1)-3*nny+i,n-1)
                        do j=1,np1(n)
                              sk1(i,j)=0.
                        end do
                        sk1(i,i)=1.
                  end do
            end if

!*** solve the system of equations by Gauss elimination
            call gelim(n)

      end do

!*** solve the system of equations iteratively
      e=10.
      k=0.
      Do While (e>1.E-5)
            do n=1,nd
!*** initialization
                  do i=1,np1(n)
                        r1(i)=0.
                        do j=1,np1(n)
                              sk1(i,j)=0.
                        end do
                  end do
!*** matrix assembly
                  do nell=nestart(n),nestart(n)+ne1(n)-1
                        call abfind(nell,n)
                  end do
!*** impose essential boundary conditions
                  do i=1,np1(n)
                        if(ncod(i+nnstart(n)-1)==1) then
                              r1(i)=bc(i+nnstart(n)-1)
                              do j=1,np1(n)
                                    sk1(i,j)=0.
                              end do
                              sk1(i,i)=1.
                        end if
                  end do

!*** impose internal boundary conditions
                  if (n==1) then
                        do i=np1(n)-nny+1,np1(n)
```

```
                              r1(i)=u(i-np1(n)+3*nny,n+1)
                              do j=1,np1(n)
                                    sk1(i,j)=0.
                              end do
                              sk1(i,i)=1.
                        end do
                  else if (n==nd) then
                        do i=1,nny
                              r1(i)=u(np1(n-1)-3*nny+i,n-1)
                              do j=1,np1(n)
                                    sk1(i,j)=0.
                              end do
                              sk1(i,i)=1.
                        end do
                  else
                        do i=np1(n)-nny+1,np1(n)
                              r1(i)=u(i-np1(n)+3*nny,n+1)
                              do j=1,np1(n)
                                    sk1(i,j)=0.
                              end do
                              sk1(i,i)=1.
                        end do
                        do i=1,nny
                              r1(i)=u(np1(n-1)-3*nny+i,n-1)
                              do j=1,np1(n)
                                    sk1(i,j)=0.
                              end do
                              sk1(i,i)=1.
                        end do
                  end if

!*** solve the system of equations by Gauss elimination
                  call gelim(n)

            end do


!******* deviation calculation
            e=0.
            do n=1,nd-1
                  do i=np1(n)-3*nny+1,np1(n)
                        e=e+(u(i,n)-u(i-np1(n)+3*nny,n+1))**2
                        write(*,*) i+nnstart(n)-1,(u(i,n)-u(i-%
&np1(n)+3*nny,n+1))**2
                  end do
            end do
```

```
        k=k+1
        e=sqrt(e)
        write(*,*) 'sum','  ',e
    end do
```

Το κομμάτι αυτό του σειριακού κώδικα είναι το βασικότερο γιατί εδώ ουσιαστικά υλοποιούμε την μέθοδο Alternating Schwarz.

Βλέπουμε ότι αρχικά ορίζουμε κάποιες τυχαίες τιμές για τα εσωτερικά σύνορα (είχε προηγηθεί εισαγωγή συνοριακών συνθηκών για τα φυσικά σύνορα του χωρίου). Για τα εσωτερικά σύνορα εισάγουμε με συνθήκες Dirichlet αρχική τιμή μηδέν για τα δεξιά σύνορα των υποχωρίων και στα αριστερά εισάγουμε πάλι με Dirichlet την τιμή που έχει ήδη υπολογιστεί για τους αντίστοιχους κόμβους του αριστερού υποχωρίου στην ίδια επανάληψη (όπου r_1 και sk_1 είναι οι πίνακες b και A του γραμμικού συστήματος $\underline{\underline{A}}\underline{u} = \underline{b}$). Με αυτό τον τρόπο εκμεταλλευόμαστε τη σειριακότητα για να μειώσουμε το πλήθος των επαναλήψεων.

Μετά από αυτή την πρώτη επίλυση των επιμέρους συστημάτων ξεκινάει η επαναληπτική μέθοδος. Ως κριτήριο σύγκλισης χρησιμοποιούμε τη δεύτερη νόρμα της διαφοράς των τιμών των κόμβων των επικαλυπτόμενων περιοχών των γειτονικών υποχωρίων και οι επαναλήψεις τερματίζουν όταν η νόρμα γίνει μικρότερη ή ίση του 10^{-5}. Σε κάθε επανάληψη το γραμμικό σύστημα ουσιαστικά ξαναχτίζεται καλώντας την υπορουτίνα abfind και επιλύεται με την υπορουτίνα gelim. Αυτό σημαίνει πολύ δουλειά για τον μοναδικό επεξεργαστή που επιλύει το πρόβλημα, αλλά παράλληλα σημαίνει και μεγάλη εξοικονόμηση χώρου αφού οι τιμές των πινάκων A και b αποθηκεύονται στις ίδιες θέσεις μνήμης για κάθε υποχωρίο.

Σε κάθε επανάληψη τα τεχνητά σύνορα που βρίσκονται στα δεξιά των υποχωρίων παίρνουν τιμή από την n-1 επανάληψη και τα αριστερά από την n επανάληψη.

4.3 Παράλληλος Κώδικας

Παρουσιάζουμε το αντίστοιχο κομμάτι του παράλληλου κώδικα σε σχέση με του σειριακού που παρουσιάστηκε παραπάνω (mm είναι η μεταβλητή που αντιπροσωπεύει κάθε επεξεργαστή):

```
!*** initial guess for internal boundary conditions
      if (mm==1) then
            do i=np1(mm)-nny+1,np1(mm)
                  r1(i)=0.    !guess essential conditions
                  do j=1,np1(mm)
                        sk1(i,j)=0.
                  end do
                  sk1(i,i)=1.
            end do
      else if (mm==nd) then
            do i=1,nny
                  r1(i)=0.    !guess essential conditions
                  do j=1,np1(mm)
                        sk1(i,j)=0.
                  end do
                  sk1(i,i)=1.
            end do
      else
            do i=np1(mm)-nny+1,np1(mm)
                  r1(i)=0.    !guess essential conditions
                  do j=1,np1(mm)
                        sk1(i,j)=0.
                  end do
                  sk1(i,i)=1.
            end do
            do i=1,nny
                  r1(i)=0.    !guess essential conditions
                  do j=1,np1(mm)
                        sk1(i,j)=0.
                  end do
                  sk1(i,i)=1.
            end do
      end if

!*** solve the system of equations by Gauss elimination
      call gelim

!*** solve the system of equations iteratively
      e=10.
      k=0.
      Do While (e>1.E-5)
!*** initialization
            do i=1,np1(mm)
                  r1(i)=0.
                  do j=1,np1(mm)
                        sk1(i,j)=0.
```

```
                end do
            end do
!*** matrix assembly
            do nell=nestart(mm),nestart(mm)+ne1(mm)-1
                call abfind(nell)
            end do
!*** impose essential boundary conditions
            do i=1,np1(mm)
                if(ncod(i+nnstart(mm)-1)==1) then
                    r1(i)=bc(i+nnstart(mm)-1)
                    do j=1,np1(mm)
                        sk1(i,j)=0.
                    end do
                    sk1(i,i)=1.
                end if
            end do

!*** impose internal boundary conditions
            if (mm==1) then
                call MPI_IRECV(r1(np1(mm)-nny+1:np1(mm)), nny,
MPI_DOUBLE_PRECISION, me+1, tag(1), MPI_COMM_WORLD, requests(1),ierror)
                do i=np1(mm)-nny+1,np1(mm)
                    do j=1,np1(mm)
                        sk1(i,j)=0.
                    end do
                    sk1(i,i)=1.
                end do
                call MPI_ISEND(u(np1(mm)-3*nny+1:np1(mm)-2*nny), nny,
MPI_DOUBLE_PRECISION, me+1, tag(2), MPI_COMM_WORLD, requests(2),ierror)
            else if (mm==nd) then
                call MPI_IRECV(r1(1:nny), nny, MPI_DOUBLE_PRECISION,
me-1, tag((mm-2)*2+2), MPI_COMM_WORLD,requests((mm-2)*4+3), ierror)
                do i=1,nny
                    do j=1,np1(mm)
                        sk1(i,j)=0.
                    end do
                    sk1(i,i)=1.
                end do
                call MPI_ISEND(u(2*nny+1:3*nny), nny,
MPI_DOUBLE_PRECISION, me-1, tag((mm-2)*2+1), MPI_COMM_WORLD,
requests((mm-2)*4+4), ierror)
            else
                call MPI_IRECV(r1(np1(mm)-nny+1:np1(mm)), nny,
MPI_DOUBLE_PRECISION, me+1, tag((mm-2)*2+3), MPI_COMM_WORLD,
requests((mm-2)*4+3), ierror)
                do i=np1(mm)-nny+1,np1(mm)
```

```
                        do j=1,np1(mm)
                              sk1(i,j)=0.
                        end do
                        sk1(i,i)=1.
                  end do
                  call MPI_IRECV(r1(1:nny), nny, MPI_DOUBLE_PRECISION,
me-1, tag((mm-2)*2+2), MPI_COMM_WORLD, requests((mm-2)*4+4),ierror)
                  do i=1,nny
                        do j=1,np1(mm)
                              sk1(i,j)=0.
                        end do
                        sk1(i,i)=1.
                  end do
                  call MPI_ISEND(u(2*nny+1:3*nny), nny,
MPI_DOUBLE_PRECISION, me-1, tag((mm-2)*2+1), MPI_COMM_WORLD,
requests((mm-2)*4+5), ierror)
                  call MPI_ISEND(u(np1(mm)-3*nny+1:np1(mm)-2*nny), nny,
MPI_DOUBLE_PRECISION, me+1, tag((mm-2)*2+4), MPI_COMM_WORLD,
requests((mm-2)*4+6), ierror)
            end if

            call MPI_WAITALL((nn-1)*4,requests,statuses,ierror)

!*** solve the system of equations by Gauss elimination
            call gelim

!******* deviation calculation
            d=0.
            if (mm<nn) call MPI_IRECV(unext(1:3*nny), 3*nny,
MPI_DOUBLE_PRECISION, me+1, tag(mm), MPI_COMM_WORLD, requests(2*mm),
ierror)
            if (mm>1) call MPI_ISEND(u(1:3*nny), 3*nny,
MPI_DOUBLE_PRECISION, me-1, tag(mm-1), MPI_COMM_WORLD, requests(2*mm-
3), ierror)
            call MPI_WAITALL((nn-1)*2, requests, statuses, ierror)
            if (mm<nn) then
                  do i=np1(mm)-3*nny+1,np1(mm)
                        d=d+(u(i)-unext(i-np1(mm)+3*nny))**2
                        write(*,*) 'process',mm,':',i+nnstart(mm)-1,
(u(i)-unext(i-np1(mm)+3*nny))**2
                  end do
            end if
            call
MPI_ALLREDUCE(d,e,1,MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierror)
            k=k+1
            e=sqrt(e)
```

```
        If (me==0) write(*,*) 'sum',' ',e
    end do
```

Εδώ βλέπουμε ότι οι τιμές που ορίζουμε για τα εσωτερικά σύνορα πριν την έναρξη της επανάληψης είναι μηδενικές και για τα δύο τεχνητά σύνορα των εσωτερικών υποχωρίων. Αυτό συμβαίνει γιατί αν είχαμε ακολουθήσει την ίδια πρακτική με πριν θα είχαμε σειριακότητα και η χρήση παράλληλων επεξεργαστών δεν θα είχε νόημα. Ωστόσο θα μπορούσαμε να είχαμε ξεκινήσει και με σειριακούς υπολογισμούς πριν την επανάληψη για να πάρουμε κάποιες καλύτερες αρχικές τιμές και στην επανάληψη να γυρίζαμε στην παράλληλη εκδοχή, αλλά βρέθηκε ότι αυτό ελάχιστη επίδραση έχει στο πλήθος των επαναλήψεων.

Όταν μπαίνουμε στην επαναληπτική διαδικασία ξεκινάμε και τη χρήση των εντολών της βιβλιοθήκης MPI για να επιτευχθεί η ανταλλαγή δεδομένων μεταξύ των επεξεργαστών. Η ανταλλαγή γίνεται σε δύο κύρια σημεία. Πρώτον, όταν χτίζονται οι πίνακες των επιμέρους υποχωρίων και δεύτερον, όταν γίνεται ο υπολογισμός της απόκλισης.

Σε αυτά τα δύο σημεία στα οποία γίνεται ανταλλαγή δεδομένων μεταξύ των επικαλυπτόμενων υποχωρίων είναι που “γεννιέται” και η καθυστέρηση. Εξ’ αιτίας της χρήσης του MPI_WAITALL σε κάθε επανάληψη οι επεξεργαστές που τελειώνουν νωρίτερα πρέπει να περιμένουν να τελειώσουν όλοι οι υπόλοιποι προκειμένου να ανταλλαγούν τα δεδομένα. Έτσι στην περίπτωση που δημιουργείτε ανισοκατανομή του υπολογιστικού φορτίου αυτές οι εντολές προκαλούν το φαινόμενο που περιγράψαμε στην παράγραφο 3.6.1.

Η μεταβολή της εξίσωσης (3.8γ) η οποία είναι απαραίτητο να γίνει για να παραλληλοποιηθεί ο κώδικας (βλέπε παράγραφο 3.5), υλοποιείται προγραμματιστικά με μία απλή αλλαγή του σημείου κλήσης της υπορουτίνας gelim.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Διεθνής Βιβλιογραφία

1) Badea L., "A generalization of the Schwarz alternating method to an arbitrary number of subdomains", *Numer. Math.*, 55, 1989, pp. 61-81
2) Bjorstad P. E., Dryja M., Vainikko E., "Additive Schwarz methods without subdomain overlap", John Wiley & Sons, 1996, pp. 67-83
3) Chau M., El Baz D., Guivarch R., Spiteri P., "MPI implementation of parallel subdomain methods for linear and nonlinear convection-diffusion problems", *Journal of Parallel and Distributed Computing*, 67, 2007, pp. 581-591
4) Dongarra J., Foster I., Fox G., Gropp W., Kennedy K., Torczon L., White A., "Sourcebook of Parallel Computing", Morgan Kaufmann Publishers, 2003
5) Flemisch B., "The Alternating Schwarz Method", Master thesis Iowa State University, Ames Iowa 2001
6) Frommer A., Szyld D. B., "On asynchronous iterations", Journal of *Computational and Applied Mathematics*, 123, 2000, pp. 201-216
7) Jordan H. F., Alaghband G., "Fundamentals of Parallel Processing", Prentice Hall, 2003
8) Kim S.-B., Hadjidimos A., Houstis E.N., Rice J.R., "Multi-parameterized Schwarz alternating methods for elliptic boundary value problems", *Mathematics and Computers in Simulation*, 42, 1996, pp. 47-76
9) Moldovan D. I., "Parallel Processing, From Applications to Systems", Morgan Kaufmann Publishers, Los Angeles, 1993
10) Schwartz H.A., "Über einen Grenzübergang durch alternierendes Verfahren", *Vierteljahrsschrift der Naturforschenden Gesellschaft*, 50, Zürich 1870, pp. 272-286
11) Smith B., Bjorstad P., Gropp W., "Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations", Cambridge University Press 1996
12) Strang G., Fix G., "Finite Element Method", 2nd edition, Wellesley-Cambridge, 2008

13) Wohlmuth B. I., “Lecture Notes in Computational Science and Engineering 17: Discretization Methods and Iterative Solvers Based on Domain Decomposition”, Springer publications, Augsburg 2000

Ελληνική Βιβλιογραφία

1) Μπουντουβής Α., «Υπολογιστική Ανάλυση με την Μέθοδο των Πεπερασμένων Στοιχείων, Εισαγωγικές Σημειώσεις», Εκδόσεις ΕΜΠ, Αθήνα 1992
2) Σπυρόπουλος Α., «Υπολογισμοί Μεγάλης Κλίμακας με Μεθόδους Παράλληλης Επεξεργασίας σε μη Γραμμικά Προβλήματα Διεπιφανειακής Μαγνητο-ρευστομηχανικής», Διδακτορική Διατριβή ΕΜΠ, Αθήνα 2003
3) Χειμαριός Ν., «Ανάλυση Πρότυπων Διεργασιών Χημικής Απόθεσης από Ατμό με τον Υπολογιστικό Κώδικα Fluent σε συστοιχίες Παράλληλης Επεξεργασίας», Μεταπτυχιακή Εργασία ΕΜΠ, Αθήνα 2008

Ιστοσελίδες

I1) Το Top 500 των ταχύτερων υπερυπολογιστών: http://www.top500.org/
I2) Η επίσημη σελίδα των μεθόδων διαχωρισμού χωρίου: http://www.ddm.org/
I3) Η ιστοσελίδα του cluster Pegasus: http://febui.chemeng.ntua.gr/pegasus.htm
I4) Διδακτικό κείμενο σχετικό με την Additive Schwarz απ’ το τμήμα μαθηματικών του Louisiana State University: https://www.math.lsu.edu/~kimn/DD.pdf
I5) Διάλεξη για την Παράλληλη Επεξεργασία του τμήματος ΜΗΥΠ του πανεπιστημίου πατρών: parallel.hpclab.ceid.upatras.gr/Lecture-1.pdf
I6) Σελίδες της Wikipedia: http://en.wikipedia.org/wiki/Parallel_computing
http://en.wikipedia.org/wiki/Domain_decomposition_methods
http://en.wikipedia.org/wiki/Gaussian_elimination
http://en.wikipedia.org/wiki/Flynn’s_taxonomy
http://en.wikipedia.org/wiki/Beowulf_cluster
http://en.wikipedia.org/wiki/Speedup
I7) Το πρόγραμμα μέτρησης επίδοσης Linpack για υπολογιστές κατανεμημένης μνήμης: http://www.netlib.org/linpack/