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

Υλοποίηση Πολυπλεγματικού Αλγορίθμου σε Παράλληλες Αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Ν. Πράπας

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

Αθήνα, Οκτώβριος 2011

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

Υλοποίηση Πολυπλεγματικού Αλγορίθμου σε Παράλληλες Αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Ν. Πράπας

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Οκτωβρίου 2011.

.....	.....	.....
Ν. Κοζύρης	Ν. Παπασπύρου	Δ. Σούντρης
Αν. Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2011

.....

Δημήτριος Ν. Πράπας

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

Copyright © Δημήτριος Ν. Πράπας, 2011
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Σε έναν μεγάλο αριθμό φυσικών και μαθηματικών προβλημάτων, χρειάζεται να γίνεται μοντελοποίηση με μερικές διαφορικές εξισώσεις προκειμένου να καθιστάται δυνατή η επίλυση. Τα προβλήματα αυτά πιθανόν να αφορούν φυσικά φαινόμενα πολύ διαφορετικά μεταξύ τους αλλά μπορούν όλα να λυθούν με μερικές διαφορικές εξισώσεις. Στην εργασία αυτή ασχολούμαστε με τη μελέτη και την υλοποίηση αλγορίθμων που επιλύουν τέτοια προβλήματα. Το πρόβλημα που εξετάζουμε σε αυτή την εργασία αφορά την επίλυση γραμμικών ελλειπτικών μερικών διαφορικών εξισώσεων (εξίσωση του Poisson και εξίσωση του Laplace) και συγκεκριμένα είναι το πρόβλημα Dirichlet σε δύο και σε τρεις διαστάσεις.

Μελετάμε αρχικά τον αλγόριθμο επίλυσης που βασίζεται στη μέθοδο Jacobi, μια γραμμική επαναληπτική μέθοδο επίλυσης. Ωστόσο, ο αλγόριθμος που μας ενδιαφέρει κυρίως είναι αυτός που προκύπτει χρησιμοποιώντας πολυπλεγματικές τεχνικές. Τέτοιες τεχνικές χρησιμοποιούνται ευρύτατα σε εφαρμογές ελλειπτικών μερικών διαφορικών εξισώσεων. Συνεπώς, τα συμπεράσματα που θα εξάγουμε μπορούν με ασφάλεια να γενικευθούν για μια μεγάλη ομάδα εφαρμογών επίλυσης μερικών διαφορικών εξισώσεων. Οι πολυπλεγματικές τεχνικές χρησιμεύουν στην αποδοτική επίλυση μιας πληθώρας γραμμικών επαναληπτικών προβλημάτων και συνδυαζόμενες με διάφορες τεχνικές διακριτοποίησης, αναδεικνύουν μια από τις ταχύτερες μεθόδους επίλυσης φυσικών και μαθηματικών προβλημάτων.

Η υλοποίηση του αλγορίθμου γίνεται σε γλώσσα C και με χρήση των εργαλείων που προσφέρουν δύο πολύ διαδεδομένες διεπαφές προγραμματισμού εφαρμογών, το OpenMP και το MPI. Προσπαθούμε με αυτό τον τρόπο να εκμεταλλευτούμε τα περιθώρια παραλληλοποίησης του αλγορίθμου. Μας απασχολεί ιδιαίτερα η εκτέλεσή του σε υπολογιστικό σύστημα παράλληλης αρχιτεκτονικής, είτε κοινής είτε κατανεμημένης μνήμης. Το υπολογιστικό αυτό σύστημα είναι στην ουσία μια συστοιχία υπολογιστικών κόμβων, για την οποία μπορούμε να επιλέξουμε δίκτυο διασύνδεσης είτε το Ethernet είτε το Myrinet.

Λέξεις-Κλειδιά

Ελλειπτικές μερικές διαφορικές εξισώσεις, Εξίσωση Laplace, Εξίσωση Poisson, Πρόβλημα Dirichlet, Γραμμικές επαναληπτικές μέθοδοι, Μέθοδος Jacobi, Πολυπλεγματικές μέθοδοι, Διαπλεγματικές μεταβάσεις, Διπλεγματική επανάληψη, Εμφωλευμένη επανάληψη, Αλγόριθμος FMG, Παράλληλος προγραμματισμός, Αρχιτεκτονική κατανεμημένης μνήμης, Αρχιτεκτονική κοινής μνήμης, MPI, OpenMP.

Abstract

In a large number of physical and mathematical problems, modeling with use of partial differential equations becomes a necessity for their solution finding procedure. These problems may vary considerably in respect to the phenomena they deal with. However, all of them can be solved using partial differential equations. In this thesis we study and implement algorithms that solve such problems. The problem under examination concerns the solution of linear elliptic partial differential equations (Poisson's equation and Laplace's equation). More specifically, it is the Dirichlet problem in two and three dimensions.

We initially consider the algorithm based on the Jacobi method, a linear iterative solution method. However, our primary concern lies in the algorithm resulting from the use of multigrid methods. Such methods are being used widely in applications involving elliptic partial differential equations. Therefore, our conclusions can be safely generalized to concern a broad group of applications solving partial differential equations. Multigrid methods are used to efficiently solve a plethora of linear iterative problems. Combined with various disretization techniques, they generate one of the fastest solving methods for physical and mathematical problems.

We build the algorithm using the C programming language and additional tools offered by two widespread application programming interfaces, OpenMP and MPI. Thus, we attempt to exploit the parallelization potentialities of the algorithm. We mainly care for its implementation on a parallel computing system of either distributed memory or shared memory architecture. This computing system is merely an array of computing elements, for which we can select the interconnection network to be either Ethernet or Myrinet.

Key words

Elliptic partial differential equations, Laplace's equation, Poisson's equation, Dirichlet problem, Linear iterative methods, Jacobi method, Multigrid methods, Intergrid transitions, Two-grid iteration, Nested iteration, FMG algorithm, Parallel programming, Distributed memory architecture, Shared memory architecture, MPI, OpenMP.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

Σε πολλά προβλήματα της Τεχνολογίας και των Φυσικομαθηματικών Επιστημών έχουμε να κάνουμε με *μερικές διαφορικές εξισώσεις* (εν συντομία: ΜΔΕ). Μοντελοποιώντας μαθηματικά τα φυσικά αυτά προβλήματα με ΜΔΕ μπορούμε να έχουμε έναν τρόπο για να επιλύσουμε προβλήματα που αφορούν φυσικά φαινόμενα πολύ διαφορετικά μεταξύ τους. Προβλήματα, λοιπόν, που αφορούν διάδοση ήχου, θερμική αγωγιμότητα, ελαστικότητα, ηλεκτροστατική, ηλεκτροδυναμική, υδροδυναμική κ.ά. μπορούν όλα να λυθούν με ΜΔΕ. Μετά από κατάλληλη *διακριτοποίηση* της εκάστοτε *μοντελοποίησης*, η *επίλυση* του προβλήματος μπορεί να γίνει πολύ ταχύτερα σε ένα υπολογιστικό σύστημα. Παραπέμπουμε τον αναγνώστη για περισσότερα πάνω στις ΜΔΕ στη βιβλιογραφία [1, 2, 5, 6].

Πρόβλημα Dirichlet

Εξίσωση Poisson

Η ΜΔΕ που θέλουμε να επιλύσουμε εδώ είναι μια *ελλειπτική ΜΔΕ*, η *Poisson*. Φαίνεται παρακάτω για δύο (δεύτερης τάξης) και για τρεις διαστάσεις (τρίτης τάξης):

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f(x, y, z)$$

Έχοντας γνωστές τις *αρχικές* (ή *Cauchy*) και τις *συνοριακές* (ή *Dirichlet*) συνθήκες, ψάχνουμε τη λύση $u(x, y)$ - $u(x, y, z)$ για την τρισδιάστατη περίπτωση - που ισχύει στη μόνιμη κατάσταση. Έχουμε να κάνουμε, δηλαδή, με ένα *πρόβλημα Dirichlet* δεύτερης και τρίτης τάξης. [βλ. 2]

Εξίσωση Laplace

Στις εκτελέσεις των αλγορίθμων που θα ακολουθήσουν, θα θέσουμε $f(x,y)=0$ για δύο διαστάσεις και $f(x,y,z)=0$ για τρεις διαστάσεις. Δηλαδή, θα εργαστούμε με την αντίστοιχη *Laplace* ΜΔΕ. Στις δύο διαστάσεις αυτή έχει ως εξής:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

και στις τρεις διαστάσεις είναι η ακόλουθη:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0$$

Οι λύσεις της εξίσωσης Laplace χρησιμοποιούνται σε διάφορα επιστημονικά πεδία, όπως στον ηλεκτρομαγνητισμό, την αστρονομία και την υδροδυναμική. Επιπλέον, η εξίσωση Laplace αντιστοιχεί στην *εξίσωση θερμότητας μόνιμης κατάστασης* αφού περιγράφει πολύ καλά τα προβλήματα θερμικής αγωγιμότητας. Μπορεί να δώσει, λοιπόν, λύση σε μεγάλο αριθμό φυσικών προβλημάτων [βλ. 2]. Για λόγους απλότητας και ευκολίας, όμως, θα θεωρήσουμε στη συνέχεια ότι το φυσικό πρόβλημα προς επίλυση είναι ένα *πρόβλημα θερμικής αγωγιμότητας* και αυτό θα χρησιμοποιούμε ως κύριο παράδειγμα.

Παράμετροι του Προβλήματος Dirichlet

Στο πρόβλημα θερμικής αγωγιμότητας, λοιπόν, για τη δισδιάστατη (για συντομία θα χρησιμοποιούμε στη συνέχεια τη συντομογραφία 2Δ για τις δύο διαστάσεις) περίπτωση, θεωρούμε μια λεπτή επίπεδη και *ορθογώνια πλάκα*. Αυτή αντιστοιχεί στο διάστημα [0,a]×[0,b] του καρτεσιανού επιπέδου Oxy, όταν ισχύει $1.25 \le a/b \le 8$. Αυτός είναι ένας περιορισμός που θα βάλουμε στις εκτελέσεις ώστε το πρόβλημα να μην περιγράφει εξαιρετικά μακρόστενες περιοχές. Η πλάκα είναι μονωμένη από το περιβάλλον και έχει κατανομή θερμότητας τη χρονική στιγμή $t=0$ που δίνεται από τη συνάρτηση $in(x,y)$. Αυτή η συνάρτηση αφορά τις αρχικές συνθήκες. Στις εκτελέσεις θα θέσουμε $in(x,y)=0.0$. Στα σύνορά της πλάκας, η θερμοκρασία είναι δεδομένη και σταθερή. Στις εκτελέσεις που θα ακολουθήσουν, θα ορίσουμε τη θερμοκρασία πάνω και αριστερά ίση με 0.0 και κάτω και δεξιά ίση με 5.0. Αυτό που ψάχνουμε είναι η *κατανομή θερμότητας στην πλάκα στη μόνιμη κατάσταση.* Το πρόβλημα αυτό είναι γνωστό ως *πρόβλημα Dirichlet σε ορθογώνιο*. [βλ. 2]

Στις τρεις διαστάσεις (για συντομία θα χρησιμοποιούμε στη συνέχεια τη συντομογραφία 3Δ για τις τρεις διαστάσεις), το πρόβλημα επεκτείνεται σε ένα *ορθογώνιο παραλληλεπίπεδο* που αντιστοιχεί στο διάστημα [0,a]×[0,b]×[0,c] του καρτεσιανού επιπέδου Oxyz και για το οποίο ισχύει: $1.25 \le a/b, b/c, c/a \le 8$ (περιορισμός για τις εκτελέσεις). Όπως και πριν, το στερεό είναι μονωμένο από το περιβάλλον και έχει κατανομή θερμότητας τη χρονική στιγμή $t = 0$ που δίνεται από τη συνάρτηση $in(x, y, z)$ (στις εκτελέσεις έχουμε θέσει $in(x, y, z) = 0.0$). Στα σύνορά του, η θερμοκρασία είναι δεδομένη και σταθερή (στις εκτελέσεις έχουμε θεωρήσει πάνω, πίσω και αριστερά ίση με 0.0 και κάτω, εμπρός και δεξιά ίση με 5.0). Αυτό που ψάχνουμε είναι η *κατανομή θερμότητας στο στερεό στη μόνιμη κατάσταση*. Το πρόβλημα αυτό είναι γνωστό ως *πρόβλημα Dirichlet σε ορθογώνιο παραλληλεπίπεδο*. [βλ. 2]

Στη συνέχεια, θα αναπτύσσουμε παράλληλα τη μεθοδολογία επίλυσης τόσο του δισδιάστατου (2Δ) προβλήματος όσο και του αντίστοιχου τρισδιάστατου (3Δ). Γενικά θα αναφερόμαστε στο 2Δ πρόβλημα χωρίς ιδιαίτερη επισήμανση ότι πρόκειται για δύο διαστάσεις. Όπου, όμως, κριθεί κατάλληλο και αφού το επισημάνουμε, θα επεκτεινόμαστε και στο αντίστοιχο 3Δ.

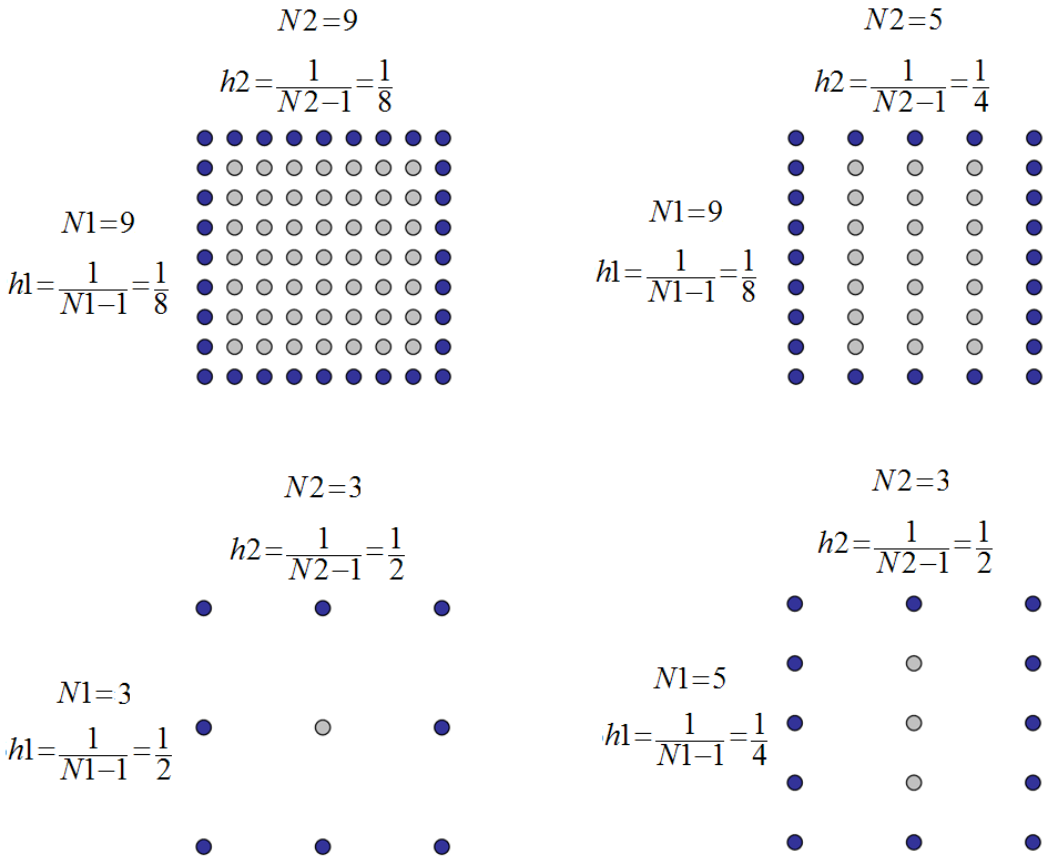
Διακριτοποίηση του Προβλήματος Dirichlet

Επειδή θα λύσουμε το πρόβλημα σε υπολογιστή, πρέπει σίγουρα να γίνει διακριτοποίηση (discretization) των μεταβλητών και της ΜΔΕ [βλ. 5]. Ας δούμε αρχικά τη δισδιάστατη (2Δ) περίπτωση και στη συνέχεια θα επεκτείνουμε τη λύση και για την αντίστοιχη τρισδιάστατη (3Δ). Θέλουμε να μπορούμε να σκεφτούμε τη 2Δ κατανομή θερμότητας πάνω στην πλάκα σαν έναν 2Δ πίνακα διαστάσεων N1×N2 με στοιχεία που θα έχουν πραγματικές τιμές ανάλογες με την πραγματική θερμοκρασία στα αντίστοιχα σημεία της πλάκας. Θεωρούμε για συντομία:

$u_{i,j}$ για $u(x_i, y_j)$ και $f_{i,j}$ για $f(x_i, y_j)$, όπου ισχύει:

$$x_i = x_0 + i\Delta_x \text{ για } i = 0,1,...,(N1-1) \text{ και } y_j = y_0 + j\Delta_y \text{ για } j = 0,1,...,(N2-1)$$

Τα Δ_x και Δ_y είναι τα *βήματα διακριτοποίησης* για τη διάσταση Ox και Oy αντίστοιχα [βλ. 5]. Στο Σχήμα 1 φαίνεται το βήμα διακριτοποίησης για διάφορα δισδιάστατα πλέγματα διακριτοποίησης:



Σχήμα 1: Βήμα διακριτοποίησης για διάφορα δισδιάστατα πλέγματα διακριτοποίησης

Στο παραπάνω σχήμα, τα μπλε σκούρα στοιχεία είναι τα *συνοριακά* και τα γκρι τα *εσωτερικά* του κάθε πίνακα. Ενδέχεται κάλλιστα και οι τέσσερις διατάξεις να αφορούν το ίδιο πρόβλημα. Απλώς ο πίνακας για την επίλυσή του θα έχει επέλθει από *διαφορετικής λεπτομέρειας διακριτοποίηση*.

Οργάνωση της Διπλωματικής Εργασίας

Στο **Κεφάλαιο 2** γίνεται αναφορά στις γραμμικές επαναληπτικές μεθόδους που μπορούν να χρησιμοποιηθούν για την επίλυση του προβλήματος Dirichlet. Από αυτές αναλύεται η μέθοδος Jacobi και υλοποιείται ο αντίστοιχος σειριακός αλγόριθμος ώστε να επιλύει το πρόβλημα.

Στο **Κεφάλαιο 3** γίνεται εισαγωγή στις πολυπλεγματικές μεθόδους και στα σημεία υπεροχής τους έναντι των μονοπλεγματικών. Μετά την περιγραφή του απλούστερου πολυπλεγματικού αλγορίθμου, παρουσιάζονται τα βασικά είδη των πολυπλεγματικών αλγορίθμων.

Στο **Κεφάλαιο 4** περιγράφεται αναλυτικά μια ειδική περίπτωση πολυπλεγματικού αλγορίθμου, ο αλγόριθμος εμφωλευμένης επανάληψης FMG. Υλοποιείται στη σειριακή του μορφή ώστε να επιλύει το πρόβλημα.

Στο **Κεφάλαιο 5** γίνεται μικρή εισαγωγή στον παράλληλο προγραμματισμό, τις παράλληλες αρχιτεκτονικές κοινής και κατανεμημένης μνήμης και τα προγραμματιστικά μοντέλα OpenMP και MPI.

Στο **Κεφάλαιο 6** περιγράφεται αναλυτικά η διαδικασία παραλληλοποίησης του σειριακού αλγορίθμου Jacobi με χρήση εργαλείων που προσφέρει το MPI.

Στο **Κεφάλαιο 7** περιγράφεται αναλυτικά η διαδικασία παραλληλοποίησης του σειριακού αλγορίθμου FMG με χρήση εργαλείων που προσφέρει το MPI.

Στο **Κεφάλαιο 8** παρουσιάζεται η διαδικασία παραλληλοποίησης του σειριακού αλγορίθμου FMG με χρήση εργαλείων που προσφέρει το OpenMP.

Στο **Κεφάλαιο 9** βρίσκεται το πειραματικό μέρος, όπου παρουσιάζονται διάφορες μετρήσεις και διαγράμματα που έχουν προκύψει από την εκτέλεση των υλοποιήσεων των προηγούμενών του Κεφαλαίων. Γίνεται ερμηνεία των αποτελεσμάτων μέσα από παρατηρήσεις και συγκρίσεις και εξάγονται σημαντικά συμπεράσματα για τους αλγορίθμους που προηγήθηκαν, τις υλοποιήσεις τους και τον βέλτιστο τρόπο εκτέλεσής τους.

Στο **Κεφάλαιο 10** συνοψίζεται η διαδικασία που ακολουθήθηκε στην εργασία αυτή και προτείνονται βελτιώσεις των παραπάνω υλοποιήσεων.

Πριν συνεχίσουμε στα επόμενα κεφάλαια, να σημειώσουμε ότι στις υλοποιήσεις των αλγορίθμων που θα παρουσιάσουμε, έχει γίνει προσαρμογή τους για καλύτερη κατανόηση. Ο αναγνώστης θα πρέπει να ανατρέξει στους κώδικες των προγραμμάτων για την πραγματική υλοποίηση. Τα προγράμματα έχουν υλοποιηθεί σε γλώσσα C ακολουθώντας την πρότυπη εμφάνιση κώδικα όπως αυτή περιγράφεται στο [12] για τον Linux Kernel v.2.6.37.

Κεφάλαιο 2

Γραμμικές Επαναληπτικές Μέθοδοι

Μια *γραμμική επαναληπτική μέθοδος* έχει στόχο να επιλύσει ένα *γραμμικό* πρόβλημα (που περιγράφεται από κάποια γραμμική ΜΔΕ). Χρησιμοποιεί ως βάση την *αλγεβρική εξίσωση επίλυσης* που προκύπτει μετά από τη διακριτοποίηση του προβλήματος. Σε αυτή την εξίσωση *επαναλαμβάνει* μια πράξη μέχρις ότου η προσεγγιστική λύση *συγκλίνει* [βλ. 5]. Οι γραμμικές επαναληπτικές μέθοδοι αποκαλούνται και *μέθοδοι εξομάλυνσης* (smoothing methods) ή *μέθοδοι χαλάρωσης* (relaxation methods) λύσης.

Υπάρχουν διάφορες μέθοδοι που μπορούν να χρησιμοποιηθούν για την επίλυση του προβλήματός μας σε κάποιο υπολογιστικό σύστημα. Γραμμικές επαναληπτικές μέθοδοι όπως είναι η *Jacobi*, η *Gauss-Seidel*, η *SOR* (Successive Over-Relaxation) και η *CG* (Conjugate Gradient) είναι συνυφασμένες με την επίλυση ελλειπτικών προβλημάτων. Για περισσότερες πληροφορίες για τις μεθόδους αυτές είναι χρήσιμες οι παραπομπές [1, 5, 6, 7]. Ωστόσο, η γραμμική επαναληπτική μέθοδος που θα αναλύσουμε και θα υλοποιήσουμε είναι η μέθοδος Jacobi.

Συνεχίζοντας από το Κεφάλαιο 1, με την εισαγωγή της $f(x,y)$ στο πρόβλημά μας, δηλαδή στην εξίσωση Poisson αντί της Laplace, είναι σαν να έχουμε επιπλέον μια πηγή θερμότητας που επηρεάζει την κατανομή θερμότητας στην πλάκα. Μετά τη διακριτοποίηση, πάντως, προκύπτει η *εξίσωση διαφορών* [βλ. 5]:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta_x} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta_y} = f_{i,j} \Leftrightarrow$$

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = \Delta_x \Delta_y f_{i,j}$$

Στη συνέχεια, θα δούμε πώς η μέθοδος Jacobi επιλύει το πρόβλημα χρησιμοποιώντας την παραπάνω εξίσωση διαφορών και θα υλοποιήσουμε τον αντίστοιχο *σειριακό* αλγόριθμο.

Αλγόριθμος Jacobi

Περιγραφή

Για την επίλυση με τη μέθοδο Jacobi, αρχικά θέτουμε σε όλα τα σημεία εκτός των συνοριακών μια αρχική τιμή (έστω 0.0) και υπολογίζουμε το:

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}) - \frac{\Delta_x \Delta_y}{4} f_{i,j}$$

Χρησιμοποιούμε τις νέες τιμές του u ως είσοδο στη δεξιά πλευρά και επαναλαμβάνουμε μέχρι το u να συγκλίνει [βλ. 5]. Στην περίπτωση των τριών διαστάσεων και με τον ίδιο ακριβώς τρόπο, καταλήγουμε στη σχέση υπολογισμού, η οποία έχει ως εξής:

$$u_{i,j,k}^{n+1} = \frac{1}{6}(u_{i+1,j,k}^{n} + u_{i-1,j,k}^{n} + u_{i,j+1,k}^{n} + u_{i,j-1,k}^{n} + u_{i,j,k+1}^{n} + u_{i,j,k-1}^{n}) - \frac{\Delta_x \Delta_y \Delta_z}{6} f_{i,j,k}$$

Με τις παραπάνω σχέσεις μπορούμε να υλοποιήσουμε τον πρώτο αλγόριθμο που θα επιλύει το πρόβλημα.

Υλοποίηση

Θεωρούμε ότι ο πίνακας u έχει διαστάσεις N1×N2 (ή N1×N2×N3 σε τρεις διαστάσεις). Ο μικρότερος πίνακας που έχει νόημα για τη μέθοδο αυτή είναι ο 3×3 (3×3×3 για 3Δ), για τον οποίο και υπάρχει μόνο ένα στοιχείο προς υπολογισμό: το κεντρικό. Σε αυτή την περίπτωση, τα βήματα διακριτοποίησης για κάθε διάσταση θα είναι ίσα με 0.5 (μέγιστο για το πρόβλημά μας). Αρκεί, λοιπόν, τα N1, N2 (και N3 για 3Δ) να είναι μεγαλύτερα του 2. Τότε η μέθοδος υλοποιείται ως εξής:

```
/* Jacobi Solver in 2D */
for (k = 0 to ITER) {

      /* interior points */
      for x = 1 to N1 - 1
            for y = 1 to N2 - 1
                  v^new[x][y] = (v^old[x - 1][y] + v^old[x + 1][y] +
                                 v^old[x][y - 1] + v^old[x][y + 1]) / 4 -
                                 h1h2f[x][y] / 4;

      pointer_swap(&v^new, &v^old);
```

```
        /* convergence check */
        if converged(v^old, v^new, cnverr)
            return;
    }
```

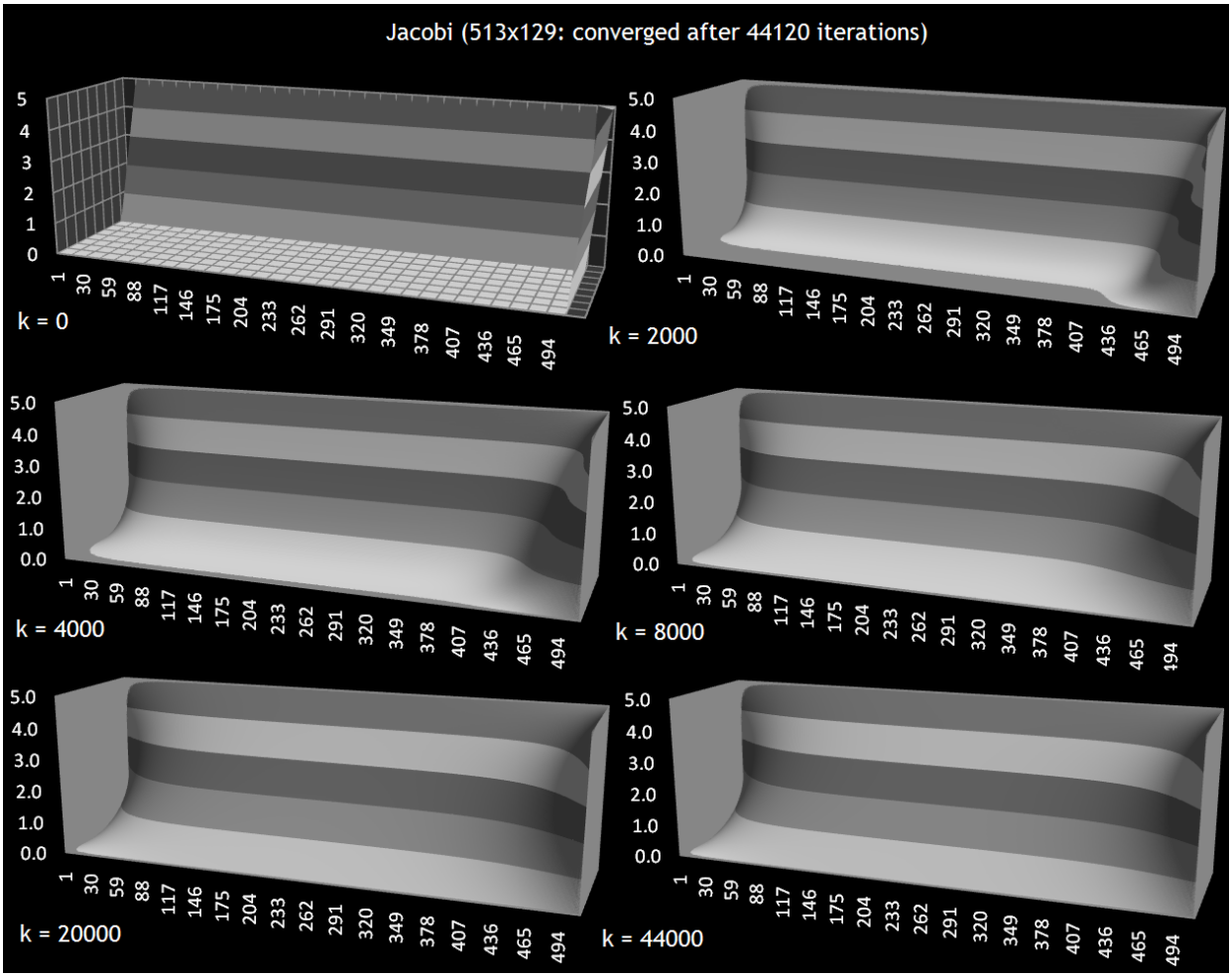
Αντίστοιχα, η τρισδιάστατη υλοποίηση έχει ως εξής:

```
    /* Jacobi Solver in 3D */
    for (k = 0 to ITER) {

        /* interior points */
        for x = 1 to N1 - 1
            for y = 1 to N2 - 1
                for z = 1 to N3 - 1
                    v^new[x][y][z] = (v^old[x - 1][y][z] +
                        v^old[x + 1][y][z] + v^old[x][y - 1][z] +
                        v^old[x][y + 1][z] + v^old[x][y][z - 1] +
                        v^old[x][y][z + 1]) / 6 –
                        h1h2h3f[x][y][z] / 6;

        pointer_swap(&v^new, &v^old);

        /* convergence check */
        if converged(v^old, v^new, cnverr)
            return;
}
```

Θεωρούμε ότι το σύστημα φτάνει στη *μόνιμη κατάσταση* όταν ο πίνακας u συγκλίνει. Αυτό συμβαίνει όταν μεταξύ δύο επαναλήψεων, η τιμή του κάθε στοιχείου (δηλαδή συγκρίνουμε τα αντίστοιχα στοιχεία των πινάκων v^{new} και v^{old}) δε μεταβάλλεται πέρα από ένα *ανώτατο όριο* (εμείς το έχουμε θέσει ίσο με 0.000001).

Στάδια του Αλγορίθμου

Για να κατανοήσουμε καλύτερα τη μέθοδο, στο Σχήμα 2 φαίνεται πώς μεταβάλλεται ένας 513×129 πίνακας (ή αλλιώς, η θερμοκρασία στην πλάκα) από τις αρχικές συνθήκες μέχρι τη μόνιμη κατάσταση (στις 44120 επαναλήψεις).



Σχήμα 2: Κατανομή θερμότητας στην πλάκα από τις αρχικές συνθήκες μέχρι τη μόνιμη κατάσταση (Jacobi)

Παρατηρώντας το κάθε διάγραμμα του παραπάνω σχήματος και τις επαναλήψεις (k) που χρειάστηκαν μέχρι αυτό, μπορούμε να διαπιστώσουμε κάτι πολύ σημαντικό. Στις πρώτες 4000 (αν όχι 2000) επαναλήψεις, το διάγραμμα φαίνεται να είναι πολύ κοντά στην τελική του μορφή. Ωστόσο, χρειάζεται να περάσουν άλλες 40000 επαναλήψεις προτού φτάσουμε στην απαραίτητη σύγκλιση. Μπορούμε να πούμε, λοιπόν, ότι η υψίσυχνη συνιστώσα του σφάλματος της λύσης εξαλείφεται σε ικανοποιητικό χρόνο αλλά η χαμηλόσυχνη συνιστώσα του μειώνεται εξαιρετικά αργά. Αυτή η συμπεριφορά προέρχεται από τον *τοπικό χαρακτήρα* των υπολογισμών στη μέθοδο Jacobi. [βλ. 3]

Προεκτάσεις

Καταλαβαίνουμε και θα το συμπεράνουμε και στο μέρος των πειραματικών αποτελεσμάτων ότι η μέθοδος Jacobi συγκλίνει πολύ αργά και ότι για μεγάλα μεγέθη πίνακα γίνεται απαγορευτική. Είναι λογική αυτή η συμπεριφορά καθώς η χρονική

πολυπλοκότητα του αλγορίθμου είναι πολυωνυμική δεύτερης τάξης. Ωστόσο, η μέθοδος Jacobi είναι πολύ απλή και αυτό την κάνει ιδιαίτερα *παραλληλοποιήσιμη*. Για αυτό, άλλωστε, τη χρησιμοποιούμε ως *βάση* για τους επόμενους αλγορίθμους.

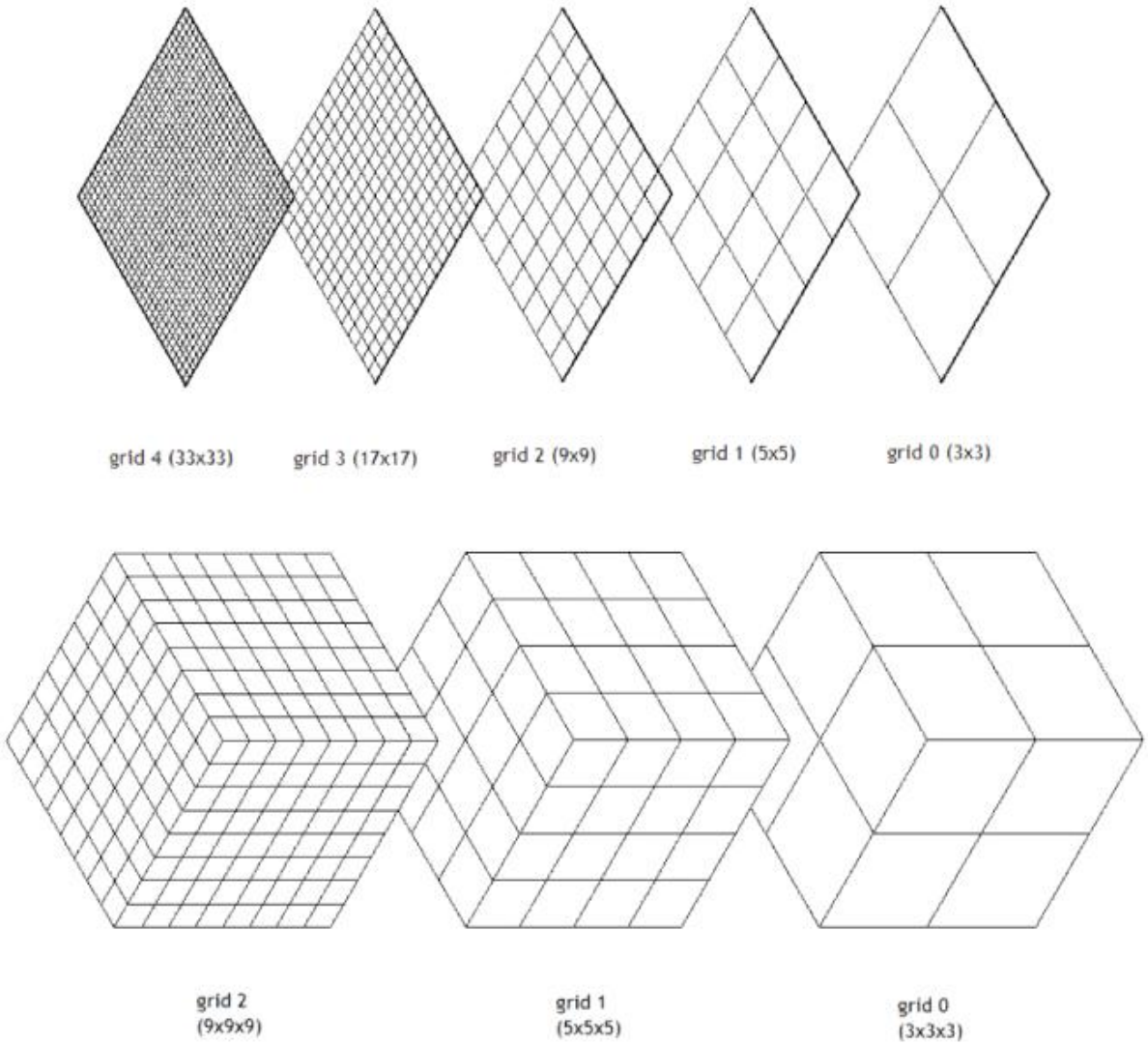
Κεφάλαιο 3

Πολυπλεγματικές Μέθοδοι

Οι *πολυπλεγματικές* (multigrid ή MG για συντομία) τεχνικές χρησιμεύουν στην αποδοτική επίλυση μιας πληθώρας γραμμικών επαναληπτικών προβλημάτων με τυπική εφαρμογή στην αριθμητική επίλυση ελλειπτικών ΜΔΕ σε δύο ή περισσότερες διαστάσεις (εμείς θα ασχοληθούμε με δύο και με τρεις διαστάσεις). Οι τεχνικές αυτές συνδυαζόμενες με διάφορες τεχνικές διακριτοποίησης αναδεικνύουν μια από τις ταχύτερες μεθόδους επίλυσης φυσικών και μαθηματικών προβλημάτων. [βλ. 6, 9]

Η MG είναι μια γενική μέθοδος, καθώς χειρίζεται αυθαίρετες περιοχές αλλά και συνοριακές συνθήκες, ενώ δε βασίζεται στις ξεχωριστές ιδιότητες της κάθε ΜΔΕ. Εφαρμόζεται, επίσης, άμεσα και σε πιο πολύπλοκα, μη συμμετρικά και μη γραμμικά συστήματα εξισώσεων (όπως σε προβλήματα ελαστικότητας) [βλ. 6, 9]. Η MG υπερβαίνει την τυπικά αργή σύγκλιση των *μονοπλεγματικών μεθόδων* (π.χ. η μέθοδος Jacobi) για επίλυση γραμμικών συστημάτων [βλ. 3]. Οι πολυπλεγματικές μέθοδοι θα γίνουν απόλυτα κατανοητές μέσα από αυτή την εργασία. Ωστόσο, παραπέμπουμε τον αναγνώστη και στη βιβλιογραφία [1, 3, 6, 7, 8, 9].

Αν και οι μονοπλεγματικές μέθοδοι εξαλείφουν αποδοτικά την υψίσυχνη συνιστώσα του σφάλματος της λύσης, μειώνουν πολύ αργά τη χαμηλόσυχνη συνιστώσα του. Αυτή η συμπεριφορά προέρχεται από τον τοπικό χαρακτήρα των υπολογισμών στις μονοπλεγματικές μεθόδους. Ο χαρακτήρας αυτός επιτρέπει τη γρήγορη εξάλειψη των συνιστωσών σφάλματος που μεταβάλλονται σε κλίμακες συγκρίσιμες με το βήμα διακριτοποίησης πλέγματος. Δυσχεραίνει, ωστόσο, την εξασθένιση των σφαλμάτων μεγαλύτερης κλίμακας. Όσο το βήμα διακριτοποίησης γίνεται μικρότερο, η πληροφορία διαδίδεται ακόμα πιο αργά και ο μυωπικός χαρακτήρας των μονοπλεγματικών επαναληπτικών μεθόδων γίνεται ακόμα πιο ισχυρός. [βλ. 3]



Σχήμα 3: Ιεραρχία πλεγμάτων (2Δ: ngrids=5, 3Δ: ngrids=3) από το λεπτομερέστερο μέχρι το αδρότερο

Οι πολυπλεγματικές τεχνικές υπερβαίνουν αυτές τις δυσκολίες χρησιμοποιώντας τέτοιες συμβατικές μεθόδους ως *συναρτήσεις εξομάλυνσης* (ή *χαλάρωσης*) σε μια *ιεραρχία πλεγμάτων* (Σχήμα 3) [βλ. 3]. Αυτές οι συναρτήσεις καλούνται και *εξομαλυντές* (smoothers) στην πολυπλεγματική ορολογία. Εμείς χρησιμοποιούμε τον αλγόριθμο Jacobi ως μια τέτοια συνάρτηση. Συνδέοντας το Σχήμα 1 με το Σχήμα 3, μπορούμε να πούμε ότι η πολυπλεγματική μέθοδος χειρίζεται πολλαπλές διαφορετικά διακριτοποιημένες εκδοχές του ίδιου προβλήματος. Εκ πρώτης όψεως, ίσως να φαίνεται ότι η πολυπλεγματική μέθοδος κάνει περισσότερη δουλειά από την αντίστοιχη μονοπλεγματική μέθοδο που χειρίζεται μόνο μία διακριτοποίηση του προβλήματος. Στην πραγματικότητα, η πολυπλεγματική μέθοδος ακολουθεί μια πιο έξυπνη, πιο στρατηγική και τελικά πολύ πιο *αποδοτική* πορεία προς την επίλυση του προβλήματος.

Καθώς οι επαναλήψεις εξομάλυνσης που πραγματοποιούνται στο πλέγμα του κάθε επιπέδου ομαλοποιούν τις συνιστώσες σφάλματος σε κλίμακα συγκρίσιμη με το βήμα διακριτοποίησης του συγκεκριμένου επιπέδου, ένας *πολυπλεγματικός κύκλος* (Σχήμα 4 – α και β) μπορεί να εξαλείψει αποδοτικά τις συνιστώσες σφάλματος σε όλο το εύρος επιφανειακών (ή χωρικών για 3Δ) συχνοτήτων. Επιπλέον, το κόστος εφαρμογής μεθόδων χαλάρωσης σε *αδρές* (coarse – μεγάλο βήμα διακριτοποίησης) κλίμακες είναι πολύ μικρότερο από αυτό σε *λεπτομερείς* (fine – μικρό βήμα διακριτοποίησης) κλίμακες, αφού οι εμπλεκόμενες μεταβλητές είναι σημαντικά λιγότερες [βλ. 3]. Η πολυπλεγματική μέθοδος, λοιπόν, είναι βέλτιστη για ένα ευρύ φάσμα ενδιαφέροντων προβλημάτων, αφού ο *ρυθμός σύγκλισής* της (convergence rate) είναι *ανεξάρτητος* από το μέγεθος του προβλήματος. Σε συνδυασμό και με την τεχνική της *εμφωλευμένης επανάληψης* (nested iteration – Σχήμα 4, γ και δ), την οποία θα αναλύσουμε αργότερα, μπορεί να επιλύσει προβλήματα στην επιθυμητή ακρίβεια σε χρόνο *ανάλογο* του αριθμού των αγνώστων ($O(N)$) [βλ. 7, 8].

Δυστυχώς, δεν υπάρχει ένας πολυπλεγματικός αλγόριθμος που να λύνει *όλα* τα ελλειπτικά προβλήματα. Υπάρχουν, όμως, πολυπλεγματικές τεχνικές που δίνουν το γενικότερο πλαίσιο για την επίλυση αυτών των προβλημάτων. Συνεπώς, με κατάλληλη αλλαγή των συνιστωσών του αλγορίθμου, μπορούμε να τον προσαρμόσουμε ώστε να επιλύει το πρόβημα που επιθυμούμε. [βλ. 1]

Αλγόριθμος Διπλεγματικής Επανάληψης [1]

Περιγραφή

Για να κατανοήσουμε καλύτερα την πολυπλεγματική μέθοδο, ας δούμε το *μαθηματικό υπόβαθρο* μιας πολύ απλής εκδοχής της: όταν τα πλέγματα είναι *δύο*. Θα διακρίνουμε τα δύο πλέγματα με βάση το βήμα διακριτοποίησης. Για ευκολία θα θεωρήσουμε σε αυτή την απλή εκδοχή ότι το βήμα διακριτοποίησης δε μεταβάλλεται ανά διάσταση (άρα $h1 = h2 = h$ και για 3Δ $h1 = h2 = h3 = h$). Είναι, όμως, διαφορετικό ανά πλέγμα. Άλλωστε, το βήμα διακριτοποίησης είναι αυτό που ορίζει τη διαφοροποίηση από πλέγμα σε πλέγμα. Έστω ότι είναι ίσο με h στο *λεπτομερές* πλέγμα και ίσο με H στο *αδρό* πλέγμα (άρα $H > h$ και συγκεκριμένα επιλέγουμε $H = 2h$). Το *πρόβλημα προς επίλυση* είναι το:

$$\nabla^2 u = f \tag{1}$$

Διακριτοποιούμε με βήμα διακριτοποίησης h:

$$\nabla_h^2 u_h = f_h \tag{2}$$

Έστω ότι το v_h δηλώνει κάποια *προσεγγιστική λύση* της εξίσωσης (2). Χρησιμοποιώντας το σύμβολο u_h για την *ακριβή λύση*, το *σφάλμα* της προσεγγιστικής λύσης (ή αντίστοιχα η *διόρθωση* που χρειάζεται) θα δίνεται από την εξίσωση (3):

$$e_h = u_h - v_h \tag{3}$$

Το *υπόλοιπο* (για την ακρίβεια, το *αντίθετο του υπολοίπου*) θα είναι:

$$r_h = -\nabla_h^2 v_h + f_h \tag{4}$$

Συνεπώς, θα έχουμε:

$$\nabla_h^2 e_h = \nabla_h^2 \left(u_h - v_h\right) = \nabla_h^2 u_h - \nabla_h^2 v_h = f_h + r_h - f_h = +r_h \tag{5}$$

Σε αυτό το σημείο, θα πρέπει να γίνει μια *προσέγγιση* του ∇_h^2 ώστε να βρούμε το e_h. Για αυτή την προσέγγιση μπορούμε κάλλιστα να χρησιμοποιήσουμε την επαναληπτική μέθοδο Jacobi που αναλύσαμε προηγουμένως. Θα βρίσκουμε, δηλαδή, σε κάθε στάδιο μια προσεγγιστική λύση της εξίσωσης (6) που ακολουθεί. Η απουσία μείον (–) στο δεύτερο μέλος είναι ο λόγος για τον οποίο επιλέξαμε η εξίσωση υπολοίπου (4) να μας δίνει το αντίθετο του υπολοίπου. Εφ' όσον έγινε πλέον κατανοητός ο λόγος αυτής της ιδιαιτερότητας, για απλότητα θα αναφερόμαστε στη συνέχεια στο αντίθετο του υπολοίπου απλά ως υπόλοιπο. Πρακτικά, μπορούμε να χρησιμοποιήσουμε για τον υπολογισμό του σφάλματος την *ίδια* συνάρτηση (Jacobi) που θα χρησιμοποιήσουμε και για την εξομάλυνση.

$$\nabla_h^2 e_h = r_h \tag{6}$$

Η *επόμενη προσέγγιση της λύσης* θα δίνεται από τον τύπο:

$$v_h^{new} = v_h^{old} + e_h \tag{7}$$

Ας δώσουμε τώρα έναν διαφορετικό τρόπο προσέγγισης της λύσης, όπου το βήμα διακριτοποίησης θα είναι μεγαλύτερο και ίσο με $H = 2h$. Έχουμε να κάνουμε, δηλαδή, με μια διαδικασία επίλυσης στο αδρότερο, πιο «χοντροκομμένο» πλέγμα.

Η εξίσωση υπολοίπου θα είναι:

$$\nabla_H^2 e_H = r_H \tag{8}$$

Αφού το ∇_H^2 είναι μικρότερων διαστάσεων, η εξίσωση (8) θα είναι ευκολότερο να λυθεί σε σχέση με την (5). Χρειαζόμαστε όμως το υπόλοιπο (όπως προείπαμε, στην πραγματικότητα είναι το αντίθετό του) r_H, το οποίο δεν έχουμε. Ωστόσο, έχουμε το r_h και με έναν *τελεστή περιορισμού* $\boldsymbol{R}$ μπορούμε να το περιορίσουμε στο αδρό πλέγμα ως εξής:

$$r_H = \boldsymbol{R}\, r_h \tag{9}$$

Αυτή η *πράξη περιορισμού* είναι μια από τις δύο *διαπλεγματικές μεταβάσεις* που θα χρησιμοποιήσουμε. Η άλλη είναι η *πράξη επέκτασης*. Ο τελεστής περιορισμού (restriction) λέγεται και τελεστής *λεπτομερούς-σε-αδρό* (fine-to-coarse) ή τελεστής *έγχυσης* (injection).

Χρησιμοποιούμε την εξίσωση (8) για να βρούμε το σφάλμα e_H και με έναν *τελεστή επέκτασης* $\boldsymbol{P}$ επεκτείνουμε τη διόρθωση στο λεπτομερές πλέγμα ως εξής:

$$e_h = \boldsymbol{P}\, e_H \tag{10}$$

Ο τελεστής επέκτασης (prolongation) λέγεται και *τελεστής αδρού-σε-λεπτομερές* (coarse-to-fine) ή τελεστής *παρεμβολής* (interpolation). Οι δύο τελεστές, $\boldsymbol{R}$ και $\boldsymbol{P}$, επιλέγονται γραμμικοί. Τελικά, χρησιμοποιώντας το αποτέλεσμα της (10) *ανανεώνουμε* την προσέγγιση με τη βοήθεια του τύπου (7).

Σύνοψη Διαδικασίας

Μπορούμε τώρα να περιγράψουμε συνοπτικά τη διαδικασία που ακολουθείται για την επίλυση του προβλήματος με δύο πλέγματα. Η διαδικασία είναι στην ουσία μια *διπλεγματική επανάληψη* (two-grid iteration) και αποτελεί τον απλούστερο *αλγόριθμο διόρθωσης* (θα εξετάσουμε παρακάτω τις βασικές διαφορές μεταξύ αλγορίθμων διόρθωσης και εμφωλευμένης επανάληψης). Αρχικά, κάνουμε $a_1 \geq 0$ βήματα εξομάλυνσης (a_1 επαναλήψεις της μεθόδου Jacobi – *πρότερη χαλάρωση*) στο λεπτομερές πλέγμα ώστε να εξομαλύνουμε την προσεγγιστική λύση v_h. Υπολογίζουμε στο λεπτομερές πλέγμα το υπόλοιπο (r_h) με τη βοήθεια της (4). Έπειτα, χρησιμοποιούμε τη συνάρτηση περιορισμού (9) και το προηγούμενο αποτέλεσμα (r_h) για να βρούμε το υπόλοιπο στο αδρό πλέγμα (r_H). Λύνουμε, με τη βοήθεια της μεθόδου Jacobi, την (8) ώστε να υπολογίσουμε την

απαιτούμενη διόρθωση (e_H). Αυτή τη διόρθωση (e_H) την επεκτείνουμε με τη βοήθεια της συνάρτησης επέκτασης (10) στο λεπτομερές πλέγμα (e_h). Χρησιμοποιούμε τη διόρθωση του λεπτομερούς πλέγματος (e_h) ώστε με τη βοήθεια του τύπου (7) να ανανεώσουμε την προσέγγιση. Τέλος, κάνουμε $a_2 \geq 0$ βήματα εξομάλυνσης (a_2 επαναλήψεις της μεθόδου Jacobi – *ύστερη χαλάρωση*) στο λεπτομερές πλέγμα ώστε να εξομαλύνουμε την προσεγγιστική λύση v_h.

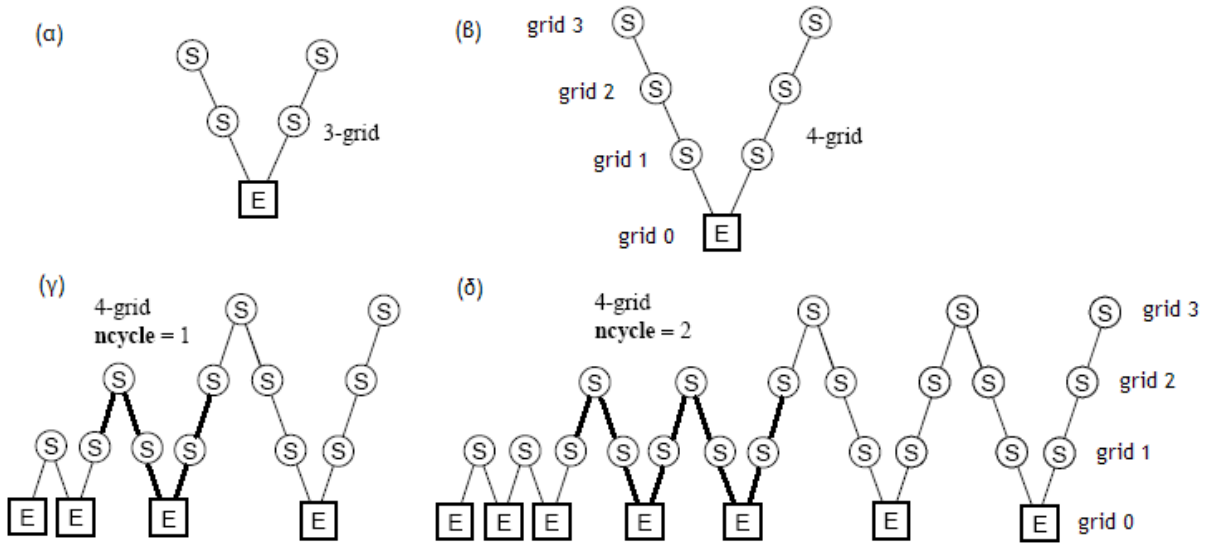
Πολυπλεγματικοί Αλγόριθμοι

Είναι απλή η μετάβαση από τη διπλεγματική μέθοδο στην *πολυπλεγματική*. Στο σημείο που υπολογίζουμε τη διόρθωση e_H μπορούμε αντί για ακριβή υπολογισμό να εισάγουμε ένα ακόμη αδρότερο πλέγμα και να κάνουμε χρήση της διπλεγματικής επανάληψης κ.ο.κ. μέχρι να φτάσουμε τον επιθυμητό αριθμό πλεγμάτων. Όλη αυτή η διαδικασία λέγεται *κύκλος* στην πολυπλεγματική ορολογία διότι αρχίζει από το λεπτομερέστερο πλέγμα και περνώντας από το αδρότερο καταλήγει πάλι στο λεπτομερέστερο. Το πόσες διπλεγματικές επαναλήψεις επιλέγουμε να γίνουν σε κάθε πλέγμα καθορίζει το *είδος του κύκλου* (για μια επανάληψη έχουμε *V-κύκλο*, για δύο επαναλήψεις έχουμε *W-κύκλο*). Για βελτίωση της σύγκλισης μπορούμε να κάνουμε επαναλήψεις και των ίδιων των πολυπλεγματικών κύκλων. [βλ. 1, 3]

Κατηγορίες Πολυπλεγματικών Αλγορίθμων

Υπάρχουν δύο κατηγορίες πολυπλεγματικών αλγορίθμων: οι αλγόριθμοι διόρθωσης και οι αλγόριθμοι εμφωλευμένης επανάληψης. Οι *αλγόριθμοι διόρθωσης*, όπως οι V-κύκλοι, ξεκινούν από το πιο λεπτομερές επίπεδο και χρησιμοποιούν τα αδρότερα επίπεδα για να εξαλείψουν το χαμηλόσυχνο σφάλμα. Δηλαδή κάνουν όσους πολυπλεγματικούς κύκλους (V ή W σχήματος συνήθως) χρειαστεί μέχρι να επιτευχθεί η επιθυμητή σύγκλιση. Από την άλλη, οι *αλγόριθμοι εμφωλευμένης επανάληψης*, όπως ο *Full MultiGrid* (FMG), ξεκινούν από τη λύση στα πιο αδρά πλέγματα και τη χρησιμοποιούν για να παράγουν μια αρχική εκτίμηση της λύσης στα πιο λεπτομερή πλέγματα προσφέροντας με αυτό τον τρόπο μεγαλύτερη αποδοτικότητα. [βλ. 3]

Παρακάτω, το πιο λεπτομερές πλέγμα βρίσκεται στην κορυφή του κάθε διαγράμματος:



Σχήμα 4: (α, β) Πολυπλεγματικοί κύκλοι (V-cycles) και (γ, δ) εμφωλευμένη επανάληψη (FMG)

Κεφάλαιο 4

Αλγόριθμος Εμφωλευμένης Επανάληψης FMG

Περιγραφή

Στο κεφάλαιο αυτό θα δώσουμε τον *Full MultiGrid* (FMG) αλγόριθμο που χρησιμοποιήσαμε για την επίλυση του προβλήματός μας. Πρώτα, όμως, θα περιγράψουμε τον βασικό κορμό του.

Αξίζει να τονίσουμε ότι στην περίπτωση της μεθόδου Jacobi, οι διαστάσεις που μας ένοιαζαν ήταν εκείνες του αρχικού πίνακα, N1×N2 (ή N1×N2×N3 για 3Δ). Στην περίπτωση της FMG μεθόδου, όμως, κάθε πλέγμα έχει το δικό του πίνακα. Άρα, τώρα μας ενδιαφέρουν οι διαστάσεις του πίνακα πλέγματος $N1_g \times N2_g$ (ή $N1_g \times N2_g \times N3_g$ για 3Δ).

Συνεχίζουμε να έχουμε το πρόβλημα της κατανομής θερμότητας σε μια ορθογώνια πλάκα (ή ορθογώνιο παραλληλεπίπεδο για 3Δ). Οι *παράμετροι* που δίνουμε στο πρόγραμμα είναι οι διαστάσεις N1×N2 (N1×N2×N3 για 3Δ) του πίνακα προς επίλυση, ο οποίος στην ουσία εκφράζει τη διακριτοποιημένη κατανομή θερμότητας πάνω στην πλάκα, ο *αριθμός των πλεγμάτων* (ngrids) και ο *αριθμός των πλεγματικών κύκλων* (ncycles) που θα γίνουν σε κάθε φάση. Η κάθε διάσταση επιλέγουμε να είναι της μορφής $N = 2^n + 1$, όπου n είναι ένας φυσικός αριθμός, ώστε με αυτό τον τρόπο να είναι δυνατές, απλές και λειτουργικές οι *διαπλεγματικές μεταβάσεις*, αφού έτσι οι διαστάσεις θα είναι σε κάθε πλέγμα της ίδιας μορφής και οι μεταβάσεις θα απαιτούν λίγους σχετικά υπολογισμούς [βλ. 1] - η μέθοδος Jacobi δεν απαιτεί τέτοιους περιορισμούς. Τέτοιες διαπλεγματικές μεταβάσεις είναι η *επέκταση* (prolongation) και ο *περιορισμός* (restriction). Για να μεταβούμε σε *λεπτομερέστερο* (finer) πλέγμα χρησιμοποιούμε τη ρουτίνα επέκτασης ενώ για *αδρότερο* (coarser) πλέγμα χρησιμοποιούμε τη ρουτίνα περιορισμού.

Η παράμετρος ngrids δεν έχει νόημα να είναι μεγαλύτερη από το n1 ($N1 = 2^{n1} + 1$) ή το n2 ($N2 = 2^{n2} + 1$) (ή το n3, όπου ισχύει αντίστοιχα $N3 = 2^{n3} + 1$, για 3Δ), αφού όταν σε μία διάσταση είναι N=3 δε μπορεί να υπάρξει αδρότερο πλέγμα. Στο πειραματικό μέρος θα δείξουμε ότι όσο μειώνεται το ngrids, τόσο η FMG μέθοδος *πλησιάζει* τη μονοπλεγματική μέθοδο, δηλαδή τη μέθοδο Jacobi. Αν είναι ngrids = 1, τότε οι δύο μέθοδοι *ταυτίζονται*. Όσο για το ncycles, αυτό δίνεται συνήθως ίσο με 1 ή 2. Ένας ακόμα τρόπος να παραμετροποιήσουμε το πρόγραμμά μας είναι να δώσουμε στις ρουτίνες εξομάλυνσης (είτε στην *άνοδο* είτε στην *κάθοδο* του V κύκλου – *πρότερη* και *ύστερη χαλάρωση*) τον

αριθμό επαναλήψεων που επιθυμούμε να γίνουν (εμείς έχουμε δώσει $a_1 = a_2 = 4$). Γενικά, ο αλγόριθμος που υλοποιούμε εδώ έχει πολλά κοινά στοιχεία με τον αντίστοιχο αλγόριθμο που παρουσιάζεται στο [1]. Έχει και πολλές σημαντικές διαφορές, όμως, που προκύπτουν κυρίως από την επιθυμία μας να είναι αρκετά παραλληλοποιήσιμος.

Καταρχάς, αρχικοποιούμε για κάθε πλέγμα τους πίνακες που θα χρειαστούμε. Θα χρειαστούμε σίγουρα δύο πίνακες (v και v^{new}) για τον εξομαλυντή και επιλυτή αδρότερου πλέγματος (Jacobi solver). Θα αναφέρονται στην προσέγγιση της ζητούμενης κατανομής. Δεσμεύουμε μνήμη για αυτούς και τους αρχικοποιούμε με τις συνοριακές αλλά και με τις αρχικές εσωτερικές τιμές τους. Θα χρειαστούμε ακόμα έναν πίνακα (f – right hand side function) για τον όρο που περιέχει τη συνάρτηση πηγής $f(x,y)$ (ή $f(x,y,z)$ για 3Δ). Δηλαδή, μέχρι στιγμής, κάνουμε ότι χρειάστηκε και στην περίπτωση του αλγορίθμου Jacobi αλλά για κάθε πλέγμα. Ορίζουμε τρεις ακόμα πίνακες σε κάθε πλέγμα (και τους αρχικοποιούμε με 0.0), τον πίνακα υπολοίπου (res - στα σύνορά του θα έχει συνέχεια 0.0 αφού οι συνοριακές συνθήκες είναι σταθερές) και τους πίνακες διόρθωσης (e και e^{new} – επίσης συνέχεια 0.0 στα σύνορα και ο δεύτερος ορίζεται μόνο στο αδρότερο πλέγμα) που εκφράζουν την απόκλιση από την προσεγγίσιμη λύση. Βρίσκουμε τη λύση στο αδρότερο πλέγμα με τη βοήθεια της μεθόδου Jacobi.

Έπειτα, ακολουθούμε μια επαναληπτική διαδικασία όπου σε κάθε επανάληψη θα γίνονται μια ρουτίνα επέκτασης και ngrids–1 επαναλήψεις ολοένα και μεγαλύτερων V κύκλων (σε κάθε επανάληψη τόσοι V κύκλοι όσους προβλέπει το ncycles). Μια τέτοια επανάληψη φαίνεται με τις έντονες γραμμές για ngrids = 4 και i = 2 στο Σχήμα 4 (γ, δ). Το S δηλώνει *εξομάλυνση* (smoothing) και το E δηλώνει *ακριβή λύση* (exact solution). Και τα δύο, βέβαια, γίνονται με την ίδια συνάρτηση, αυτή που έχει προκύψει από τη μέθοδο Jacobi. Στον V κύκλο, οι καθοδικές γραμμές (\) δηλώνουν υπολογισμό υπολοίπου (residual calculation) και διαπλεγματική μετάβαση περιορισμού υπολοίπου (residual restriction) ενώ οι ανοδικές γραμμές (/) δηλώνουν διαπλεγματική μετάβαση επέκτασης σφάλματος (error prolongation) και διόρθωση της προσεγγιστικής λύσης (approximated solution correction) με βάση το σφάλμα. [βλ. 1]

Υλοποίηση

Κορμός του Αλγορίθμου

Είμαστε πλέον έτοιμοι να παρουσιάσουμε τον FMG αλγόριθμο. Συμβολίζουμε τα βήματα διακριτοποίησης με τα οποία προέκυψε το αδρότερο grid με h_1, h_2 (και h_3 για 3Δ) και με H_1, H_2 (και H_3 για 3Δ) αντίστοιχα για το λεπτομερέστερο. Τα βήματα

διακριτοποίησης του κάθε πλέγματος μπορούν κάλλιστα να διαφέρουν μεταξύ τους. Ωστόσο, είναι συγκεκριμένα σε κάθε πλέγμα και οι διαπλεγματικές μεταβάσεις σημαίνουν *διπλασιασμό* (περιορισμός) ή *υποδιπλασιασμό* τους (επέκταση). Για ευκολία, στον παρακάτω αλγόριθμο τα χειριζόμαστε ως διανυσματικές συνιστώσες και συμβολίζουμε με H το διανυσματικό βήμα διακριτοποίησης του αδρότερου grid και με h εκείνο του λεπτομερέστερου. Στις δύο διαστάσεις θα είναι $H=(H_1, H_2)$ και $h=(h_1, h_2)$ ενώ στις τρεις θα είναι $H=(H_1, H_2, H_3)$ και $h=(h_1, h_2, h_3)$.

```
/* Full MultiGrid algorithm */
v^H = solve(f^H, FMG_ITER); // computing exact solution

/* going right to groups of higher v-cycles */
for (g = H; g > h; g /= 2) {
      fg = g / 2;

      /* going up right before every first v-cycle */
      interpolate(v^g → v^fg); // solution prolongation

      /* amount of v-cycles according to given ncycles */
      for (c = 0; c < ncycles; c++) {

            /* going down in v-cycle */
            for (gg = fg; gg < H; gg *= 2) {
                  v^gg = solve(f^gg, a_1); // solution relaxation
                  r^gg = calc_res(v^gg, f^gg); // residual calculation
                  restrict(r^gg → r^2gg); // residual restriction
                  initmtrx(e^2gg); // initialization with 0.0
            }

            e^H = solve(r^H, FMG_ITER); // computing exact error

            /* going up in v-cycle */
            for (gg = H / 2; gg > fg; gg /= 2) {
                  interpolate(e^2gg → e^gg); // error prolongation
                  correct(v^gg, e^gg); // solution correction
                  v^gg = solve(f^gg, a_2); // solution relaxation
            }
      }
}
```

Συναρτήσεις του Αλγορίθμου

Συνάρτηση επίλυσης και εξομάλυνσης

Θα παραθέσουμε και τις βασικές συναρτήσεις που χρησιμοποιούνται στη μέθοδο FMG. Τη συνάρτηση εξομάλυνσης και επίλυσης *jacobi_solve* την έχουμε ήδη παραθέσει στην προηγούμενη ενότητα αλλά θα την παραθέσουμε και εδώ για ευκολία. Η συνάρτηση

αυτή μπορεί να χρησιμοποιηθεί στη θέση τόσο της solve όσο και της relax. Ως συνάρτηση εξομάλυνσης/χαλάρωσης (relax) χρησιμοποιεί την επαναληπτική μέθοδο Jacobi για την πρότερη - a_1 επαναλήψεις της Jacobi κατά την κάθοδο (\) στον πλεγματικό κύκλο - και ύστερη - a_2 επαναλήψεις κατά την άνοδο (/) - *εξομάλυνση* της λύσης της $\nabla_g^2 u_g = f_g$ σε όλα τα πλέγματα πλην του αδρότερου. Ως συνάρτηση επίλυσης (solve) χρησιμοποιεί τη μέθοδο Jacobi για *ακριβή εύρεση* της λύσης (κάνει όσες επαναλήψεις χρειαστεί μέχρι την επιθυμητή σύγκλιση) της $\nabla_H^2 u_H = f_H$ (πρώτη εντολή του αλγορίθμου) και της $\nabla_H^2 e_H = r_H$ (εντολή στο κατώτατο σημείο του κάθε πλεγματικού κύκλου) στο αδρότερο μόνο πλέγμα. Στον αλγόριθμο που ακολουθεί έχουν επιλεχτεί συμβολισμοί για την περίπτωση της $\nabla_g^2 u_g = f_g$ χωρίς αυτό να σημαίνει ότι δεν επιλύει και την $\nabla_H^2 e_H = r_H$. Επίσης, για ευκολότερη ανάγνωση στους πίνακες v^{old}, v^{new} και f έχει παραλειφθεί ο δείκτης g που δηλώνει το πλέγμα στο οποίο ανήκει ο πίνακας.

```
/* Jacobi Solver & Smoother */

for (k = 0 to ITER) {

      /* interior points */
      for x = 1 to N1g - 1
            for y = 1 to N2g - 1
                  vnew[x][y] = (vold[x - 1][y] + vold[x + 1][y] +
                              vold[x][y - 1] + vold[x][y + 1]) / 4 -
                              h1h2f[x][y] / 4;

      pointer_swap(&vnew, &vold);

      /* convergence check */
      if converged(vold, vnew, cnverr)
            return;
}
```

Αντίστοιχα, η τρισδιάστατη υλοποίηση:

```
for (k = 0 to ITER) {

      /* interior points */
      for x = 1 to N1g - 1
            for y = 1 to N2g - 1
                  for z = 1 to N3g - 1
                        vnew[x][y][z] = (vold[x - 1][y][z] +
                              vold[x + 1][y][z] + vold[x][y - 1][z] +
                              vold[x][y + 1][z] + vold[x][y][z - 1] +
                              vold[x][y][z + 1]) / 6 –
                              h1h2h3f[x][y][z] / 6;

      pointer_swap(&vnew, &vold);
```

```
        /* convergence check */
        if converged(v^old, v^new, cnverr)
              return;
  }
```

Συνάρτηση υπολογισμού υπολοίπου

Η συνάρτηση υπολογισμού υπολοίπου *calc_res* υπολογίζει το αντίθετο του υπολοίπου - κατά την κάθοδο (\) στον πλεγματικό κύκλο - μέσω της $r_g = -\nabla_g^2 v_g + f_g$. Στις δύο διαστάσεις αυτό υπολογίζεται για κάθε στοιχείο του πίνακα ως εξής:

$$r_{i,j} = -\frac{1}{\Delta_x \Delta_y}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}) + f_{i,j}$$

Και ο αντίστοιχος αλγόριθμος:

```
/* Calculation of (minus) the residual */

/* interior points */
for x = 1 to N1_g - 1
      for y = 1 to N2_g - 1
            r_g[x][y] = - (v_g[x - 1][y] + v_g[x + 1][y] +
                                    v_g[x][y - 1] + v_g[x][y + 1] –
                                    4 * v_g[x][y]) / h_1h_2 + f_g[x][y];
```

Στις τρεις διαστάσεις θα έχουμε:

$$r_{i,j,k} = -\frac{1}{\Delta_x \Delta_y \Delta_z}(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1} - 6u_{i,j,k}) + f_{i,j,k}$$

Ο αντίστοιχος αλγόριθμος για τις τρεις διαστάσεις έχει ως εξής:

```
/* interior points */
for x = 1 to N1_g - 1
      for y = 1 to N2_g - 1
            for z = 1 to N3_g - 1
                  r_g[x][y][z] = - (v_g[x - 1][y][z] + v_g[x + 1][y][z] +
                              v_g[x][y - 1][z] + v_g[x][y + 1][z] +
                              v_g[x][y][z - 1] + v_g[x][y][z + 1] –
                              6 * v_g[x][y][z]) / h_1h_2h_3 + f_g[x][y][z];
```

Συνάρτηση διόρθωσης [1]

Πολύ απλός είναι και ο αλγόριθμος της συνάρτησης διόρθωσης *correct* που διορθώνει - στην άνοδο (/) κάθε πολυπλεγματικού κύκλου - κατάλληλα την προσεγγιστική λύση χρησιμοποιώντας την $v_g^{new} = v_g^{old} + e_g$. Στην ουσία προσθέτει τον πίνακα διόρθωσης e_g που έχει προκύψει μετά από επέκταση του πίνακα διόρθωσης e_H που βρέθηκε στο αδρότερο πλέγμα στα στοιχεία του πίνακα της προσεγγιστικής λύσης v_g.

```
/*
 * Correction of the approximated solution by
 * subtracting the calculated error
 */

/* interior points */
for x = 1 to N1g - 1
      for y = 1 to N2g - 1
            vg[x][y] += eg[x][y];
```

Εύκολα επεκτείνεται και στις τρεις διαστάσεις ως εξής:

```
/* interior points */
for x = 1 to N1g - 1
      for y = 1 to N2g - 1
            for z = 1 to N3g - 1
                  vg[x][y][z] += eg[x][y][z];
```

Συνάρτηση αρχικοποίησης

Η *initmtrx* απλώς θέτει όλα τα στοιχεία ενός πίνακα ίσα με 0 και χρησιμοποιείται για να μηδενίσει τον πίνακα διόρθωσης πριν τον υπολογισμό (και την επέκτασή) του στο (και από) το αδρότερο πλέγμα.

Συνάρτηση επέκτασης[1]

Για τη συνάρτηση επέκτασης (prolongation) *interpolate* χρησιμοποιούμε στην περίπτωση των δύο διαστάσεων *γραμμική παρεμβολή* (linear interpolation) και στην περίπτωση των τριών διαστάσεων *επιφανειακή παρεμβολή* (surface interpolation). Η συνάρτηση interpolate έχει δύο βασικές χρήσεις. Πρώτον, χρησιμοποιείται για επέκταση της λύσης από το προηγούμενο πλέγμα πριν από κάθε ομάδα πολυπλεγματικών κύκλων ίδιου μεγέθους. Στην ουσία αντιστοιχεί στην πρώτη άνοδο (/) σε πλέγμα που δεν έχει «επισκεφθεί» η μέθοδος FMG. Υλοποιεί, δηλαδή, την πράξη $v_g = P\ v_{2g}$. Επιπλέον, η interpolate χρησιμοποιείται στην άνοδο (/) του πολυπλεγματικού κύκλου για την επέκταση

του σφάλματος στα λεπτομερέστερα πλέγματα μετά τον υπολογισμό του στο αδρότατο. Σε αυτή την περίπτωση, υλοποιεί την πράξη $e_g = P e_{2g}$. Ο καθορισμός του τελεστή P εξαρτάται από το είδος της παρεμβολής που επιλέγεται να ακολουθηθεί. Συνεπώς, είναι δυνατόν να ποικίλλει από εφαρμογή σε εφαρμογή αλλά και μεταξύ εκδόσεων της ίδιας εφαρμογής.

Εμείς χρησιμοποιούμε στη δισδιάστατη έκδοση *διγραμμική* παρεμβολή:

$$P = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Προκύπτει ο παρακάτω αλγόριθμος για τη συνάρτηση επέκτασης στις δύο διαστάσεις. Στον αλγόριθμο αυτό έχουν επιλεχτεί συμβολισμοί για την περίπτωση της $v_g = P v_{2g}$ χωρίς αυτό να σημαίνει ότι δεν υλοποιεί και την $e_g = P e_{2g}$.

```
/* coarse-to-fine prolongation using interpolation */

/* elements that are copies */
for x2g = 1 to N12g - 1
      for x2g = 1 to N22g - 1
            vg[2 * x2g][2 * y2g] = v2g[x2g][y2g];

/* even-numbered columns, interpolating vertically */
for xg = 1 to (N1g – 1) with step 2
      for yg = 2 to N2g – 1 with step 2
            vg[xg][yg] = 0.5 * (vg[xg + 1][yg] + vg[xg - 1][yg]);

/* odd-numbered columns, interpolating horizontally */
for xg = 1 to N1g - 1
      for yg = 1 to N2g – 1 with step 2
            vg[xg][yg] = 0.5 * (vg[xg][yg + 1] + vg[xg][yg - 1]);
```

Αν είναι $P = P(\Delta x, \Delta y, \Delta z)$, όπου $\Delta x, \Delta y, \Delta z \in (-1, 0, 1)$, στις τρεις διαστάσεις χρησιμοποιούμε τον τελεστή P, για τον οποίο ισχύει:

$$P(\Delta z = 0) = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}, \; P(\Delta z = -1) = P(\Delta z = 1) = \begin{bmatrix} \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \end{bmatrix}$$

Και στις τρεις διαστάσεις έχουν επιλεχτεί στον αλγόριθμο συμβολισμοί για την περίπτωση της $v_g = P v_{2g}$ αλλά αυτό δε σημαίνει ότι δεν υλοποιεί και την $e_g = P e_{2g}$. Ο αλγόριθμος για την interpolate στις τρεις διαστάσεις έχει ως εξής:

```
/* elements that are copies */
for x2g = 1 to N12g - 1
      for y2g = 1 to N22g - 1
            for z2g = 1 to N32g – 1
                  vg[2 * x2g][2 * y2g][2 * z2g] = v2g[x2g][y2g][z2g];

/* even-x-even-y-numbered columns, interpolating vertically */
for xg = 1 to (N1g – 1) with step 2
      for yg = 2 to (N2g – 1) with step 2
            for zg = 2 to (N3g – 1) with step 2
                  uf[xg][yg][zg] = 0.5 * (vg[xg + 1][yg][zg] +
                                          vg[xg - 1][yg][zg]);

/* even-x-odd-y-numbered columns, interpolating z-horizontally */
for xg = 1 to N1g - 1
      for yg = 2 to (N2g – 1) with step 2
            for zg = 1 to (N3g – 1) with step 2
                  vg[xg][yg][zg] = 0.5 * (vg[xg][yg][zg + 1] +
                                          vg[xg][yg][zg - 1]);

/* odd-x-numbered columns, interpolating horizontally */
for xg = 1 to N1g - 1
      for yg = 1 to (N2g – 1) with step 2
            for zg = 1 to N3g - 1
                  vg[xg][yg][zg] = 0.5 * (vg[xg][yg + 1][zg] +
                                          vg[xg][yg - 1][zg]);
```

Συνάρτηση περιορισμού[1]

Για τη συνάρτηση περιορισμού (restriction) χρησιμοποιούμε *υποδειγματοληψία* (partial weighting sampling). Η συνάρτηση *restrict* χρησιμοποιείται για περιορισμό του υπολοίπου από το προηγούμενο πλέγμα στο αδρότερο κατά την κάθοδο (\) σε έναν πολυπλεγματικό κύκλο. Υλοποιεί, δηλαδή, την πράξη $r_{2g} = R\, r_g$. Θα μπορούσαμε πολύ απλοϊκά να επιλέξουμε τη μέθοδο της *ευθείας έγχυσης* (straight injection sampling) για τον τελεστή R. Αυτό θα σήμαινε να αντικαθιστούμε κάθε στοιχείο του αδρού πλέγματος με το αντίστοιχο στοιχείο του λεπτομερούς πλέγματος, οπότε θα είναι $R = [1]$. Στην πράξη, είναι καλύτερο οι τελεστές P και R να μην επιλέγονται ανεξάρτητα. Μια ασφαλής λύση είναι ο τελεστής R να είναι ο *συμπληρωματικός* του P . Στην περίπτωση που εξετάζουμε, προκύπτει ο *ανάστροφος* του P διαιρεμένος με 2^d, όπου d ο αριθμός των διαστάσεων. Ο καθορισμός του τελεστή R εξαρτάται και από το είδος της δειγματοληψίας που επιλέγεται να ακολουθηθεί. Συνεπώς, όπως και ο τελεστής P, είναι δυνατόν να ποικίλλει από εφαρμογή σε εφαρμογή αλλά και μεταξύ εκδόσεων της ίδιας εφαρμογής.

Στη δισδιάστατη εκδοχή θα είναι:

$$\boldsymbol{R} = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \xrightarrow[weighting]{half} \begin{bmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{2} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{bmatrix}$$

Προκύπτει ο παρακάτω αλγόριθμος για τη συνάρτηση περιορισμού στις δύο διαστάσεις:

```
/* half-weighting residual restriction from fine-grid to coarse-grid */

/* interior points */
x_g = 0;
for (x_2g = 1 to N1_2g - 1) {
      x_g += 2;
      y_g = 0;
      for (y_2g = 1 to N2_2g - 1) {
            y_g += 2;
            r_2g[x_2g][y_2g] = 0.5 * r_g[x_g][y_g] + 0.125 *
                               (r_g[x_g + 1][y_g] + r_g[x_g - 1][y_g] +
                               r_g[x_g][y_g + 1] + r_g[x_g][y_g - 1]);
      }
}
```

Αν είναι $\boldsymbol{R} = R(\Delta x, \Delta y, \Delta z)$, όπου $\Delta x, \Delta y, \Delta z \in (-1, 0, 1)$, στις τρεις διαστάσεις χρησιμοποιούμε τον τελεστή $\boldsymbol{R}$, για τον οποίο ισχύει:

$$R(\Delta z = -1) = R(\Delta z = 1) = \begin{bmatrix} \frac{1}{64} & \frac{1}{32} & \frac{1}{64} \\ \frac{1}{32} & \frac{1}{16} & \frac{1}{32} \\ \frac{1}{64} & \frac{1}{32} & \frac{1}{64} \end{bmatrix}, R(\Delta z = 0) = \begin{bmatrix} \frac{1}{32} & \frac{1}{16} & \frac{1}{32} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{32} & \frac{1}{16} & \frac{1}{32} \end{bmatrix}$$

Μετατοπίζοντας το βάρος στους «άξονες» του τελεστή, προκύπτει:

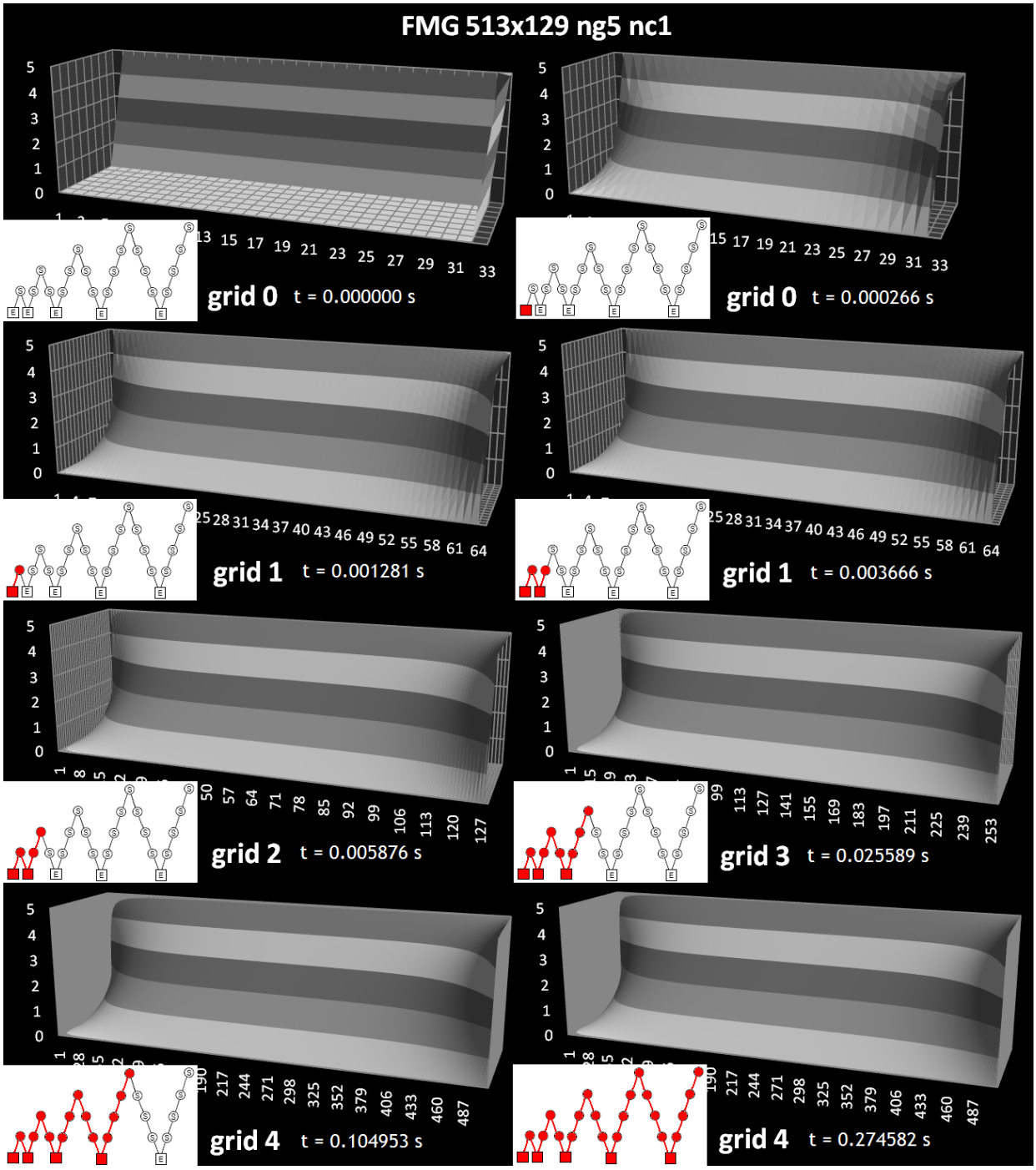
$$R(\Delta z = -1) = R(\Delta z = 1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{8} & 0 \\ 0 & 0 & 0 \end{bmatrix}, R(\Delta z = 0) = \begin{bmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{bmatrix}$$

Ο αλγόριθμος για τη restrict στις τρεις διαστάσεις έχει ως εξής:

```
/* interior points */
x_g = 0;
for (x_2g = 1 to N1_2g – 1) {
      x_g += 2;
      y_g = 0;
      for (y_2g = 1 to N2_2g - 1) {
            y_g += 2;
            z_g = 0;
            for (z_2g = 1 to N3_2g - 1) {
                  z_g += 2;
                  r_2g[x_2g][y_2g][z_2g] = 0.25 * r_g[x_g][y_g][z_g] + 0.125 *
                              (r_g[x_g + 1][y_g][z_g] + r_g[x_g - 1][y_g][z_g] +
                              r_g[x_g][y_g + 1][z_g] + r_g[x_g][y_g - 1][z_g] +
                              r_g[x_g][y_g][z_g + 1] + r_g[x_g][y_g][z_g - 1]);
            }
      }
}
```

Στάδια του Αλγορίθμου FMG

Για να καταλάβουμε καλύτερα την πορεία που ακολουθεί η μέθοδος FMG ως τη σύγκλιση, στο Σχήμα 5 φαίνεται πώς μεταβάλλεται ο 2Δ πίνακας (δηλαδή η κατανομή θερμότητας στην πλάκα για το 2Δ παράδειγμά μας) σε κάποια κομβικά στάδια από τις αρχικές συνθήκες μέχρι τη μόνιμη κατάσταση. Έχουμε διαστάσεις 513×129, ένα πλήθος πέντε (5) πλεγμάτων και έναν V κύκλο ανά επανάληψη.



Σχήμα 5: Κατανομή θερμότητας στην πλάκα από τις αρχικές συνθήκες μέχρι τη μόνιμη κατάσταση (FMG)

Από το παραπάνω σχήμα φαίνεται ότι οι πολυπλεγματικές μέθοδοι και ειδικότερα ο αλγόριθμος FMG εξαλείφουν πολύ αποδοτικά τις συνιστώσες σφάλματος σε όλο το εύρος χωρικών συχνοτήτων. Στο παράδειγμα αυτό, το αδρότερο πλέγμα (33×9) δεν είναι το αδρότερο δυνατό (9×3). Θα δούμε στο πειραματικό μέρος αν το αδρότερο πλέγμα πρέπει να είναι και το αδρότερο δυνατό. Πάντως, με τον FMG αλγόριθμο έχουμε καταφέρει σε 0.3 δευτερόλεπτα ό,τι χρειάστηκε 51.4 δευτερόλεπτα με τη μέθοδο Jacobi.

Προεκτάσεις

Μέχρι εδώ πρέπει να έχει γίνει σαφής η υπεροχή του FMG από τον μονοπλεγματικό αλγόριθμο Jacobi. Αναφερθήκαμε προηγουμένως στην *απλότητα* της μεθόδου Jacobi και είπαμε ότι αυτό την κάνει ιδιαίτερα παραλληλοποιήσιμη. Ξέρουμε, ακόμα, ότι ο αλγόριθμος FMG χρησιμοποιεί τον αλγόριθμο Jacobi ως βασικό επιλυτή αδρότερου πλέγματος αλλά και ως εξομαλυντή. Όπως, λοιπόν, χρησιμοποιήσαμε την πολυπλεγματική μέθοδο εμφωλευμένης επανάληψης για να λύσουμε ταχύτερα το πρόβλημά μας, τώρα θα χρησιμοποιήσουμε *τεχνικές παράλληλου προγραμματισμού* ώστε να βελτιώσουμε την ταχύτητα σύγκλισης και των δύο παραπάνω μεθόδων.

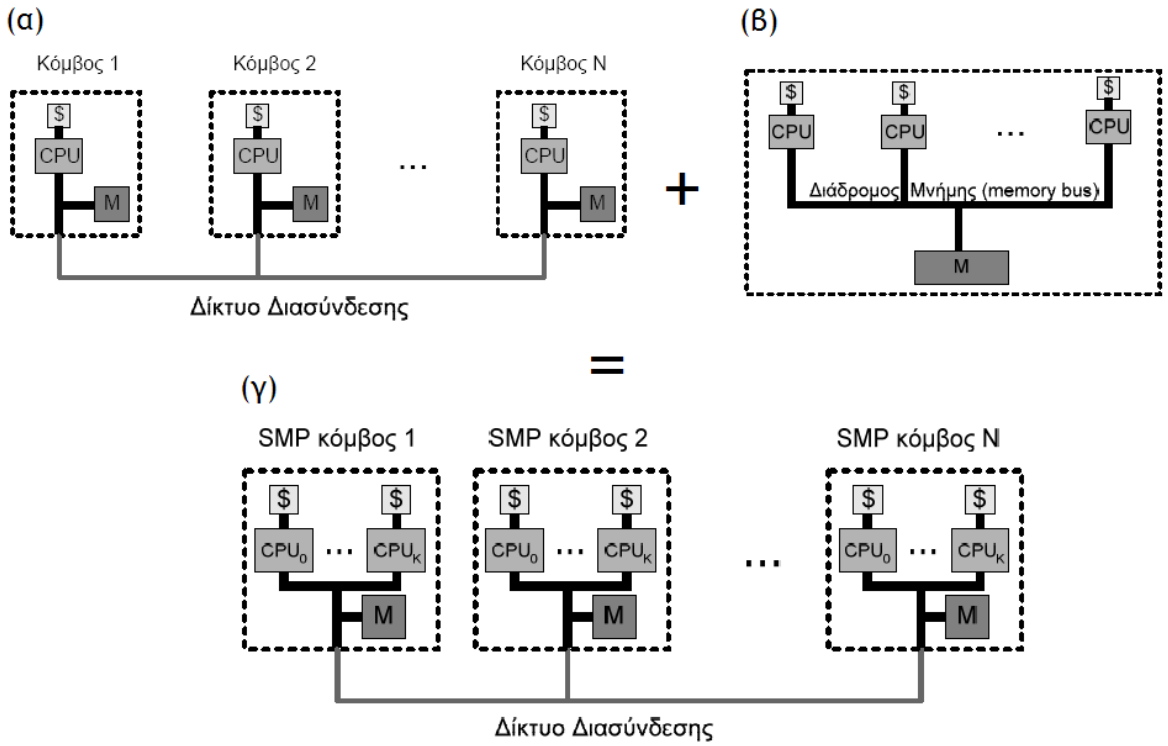
Κεφάλαιο 5

Παράλληλος προγραμματισμός

Προγραμματίζοντας παράλληλα στοχεύουμε να *κατανείμουμε* τους υπολογισμούς και τα δεδομένα (σε όποια κομμάτια του προγράμματος αυτό είναι δυνατό) σε επεξεργαστικά στοιχεία ώστε να προκύψει μικρότερος χρόνος εκτέλεσης από τον αντίστοιχο σειριακό αλγόριθμο. Αναφορικά με τον τρόπο οργάνωσης της μνήμης, υπάρχουν τρεις βασικές *κατηγορίες παράλληλων αρχιτεκτονικών*: η αρχιτεκτονική *κατανεμημένης μνήμης*, η αρχιτεκτονική *κοινής* μνήμης και η *υβριδική* αρχιτεκτονική.

Παράλληλες Αρχιτεκτονικές

Στην αρχιτεκτονική κατανεμημένης μνήμης (distributed memory) κάθε επεξεργαστής έχει τη δική του τοπική μνήμη και τοπική ιεραρχία κρυφών μνημών. Υπάρχει *δίκτυο διασύνδεσης* για την επικοινωνία των επεξεργαστών. Στην αρχιτεκτονική κοινής μνήμης (shared memory), οι επεξεργαστές έχουν κοινή μνήμη αλλά ο καθένας διαθέτει τοπική ιεραρχία κρυφών μνημών. Συνήθως, η διασύνδεση γίνεται μέσω *διαδρόμου μνήμης*. Η υβριδική αρχιτεκτονική συνδυάζει τις δύο πρώτες κατηγορίες ώστε κόμβοι με αρχιτεκτονική κοινής μνήμης να διασυνδέονται με ένα δίκτυο διασύνδεσης σε αρχιτεκτονική κατανεμημένης μνήμης.



Σχήμα 6: Κατηγορίες παράλληλων αρχιτεκτονικών: (α) Αρχιτεκτονική κατανεμημένης μνήμης, (β) Αρχιτεκτονική κοινής μνήμης και (γ) Υβριδική αρχιτεκτονική

Θα χρησιμοποιήσουμε τις δύο πρώτες κατηγορίες παράλληλων αρχιτεκτονικών ώστε να υλοποιήσουμε παράλληλα τις μεθόδους Jacobi και FMG. Σε αυτό το εγχείρημα θα μας βοηθήσουν να υλοποιήσουμε τις εφαρμογές μας δύο βασικοί εκπρόσωποι *προγραμματιστικών μοντέλων* που συνδυάζονται πολύ καλά με τις παραπάνω αρχιτεκτονικές: το *MPI* (message passing) και το *OpenMP* (shared address space). Και τα δύο είναι πρότυπα και όχι συγκεκριμένες υλοποιήσεις. Το MPI υλοποιείται συχνότερα σε αρχιτεκτονικές κατανεμημένης μνήμης, καθώς σε χαμηλό επίπεδο μπορεί να επικοινωνήσει με το δίκτυο διασύνδεσης. Το OpenMP υλοποιείται ευκολότερα σε αρχιτεκτονικές κοινής μνήμης. Σε υψηλό επίπεδο παρέχουν και τα δύο *διεπαφή* στον προγραμματιστή. [βλ. 10, 11]

Προγραμματιστικά Μοντέλα

Με το MPI η εκτέλεση ξεκινά με έναν αριθμό από MPI *διεργασίες* που εκτελούν το *ίδιο* πρόγραμμα. Κάθε διεργασία έχει *πρόσβαση* στα δικά της μόνο δεδομένα. Μπορεί να έχει πρόσβαση και σε δεδομένα άλλης διεργασίας με ρητές συναρτήσεις του MPI (αποστολής και λήψης). [βλ. 10]

Με το OpenMP η εκτέλεση ξεκινά ως ένα αρχικό νήμα και όταν η εκτέλεση συναντήσει μια *παράλληλη περιοχή*, τότε δημιουργείται ένα συγκεκριμένο πλήθος από OpenMP *νήματα* που έχουν πρόσβαση σε κοινό χώρο διευθύνσεων. Το ποια είναι παράλληλη περιοχή και ποια όχι, το δηλώνει ρητά ο προγραμματιστής. Στο τέλος της παράλληλης περιοχής, πάντως, τα νήματα *συγχρονίζονται*. [βλ. 11]

Προεκτάσεις

Μπορούμε να χρησιμοποιήσουμε και τα δύο πρότυπα (υβριδικός προγραμματισμός) ώστε σε μια υβριδική αρχιτεκτονική το OpenMP να ορίζει την παραλληλία στο *εσωτερικό* του κόμβου και το MPI να χρησιμοποιείται για την επικοινωνία *μεταξύ* των κόμβων.

Κεφάλαιο 6

Παραλληλοποίηση του Αλγορίθμου Jacobi με Χρήση του MPI

Περιγραφή

Σκοπός τώρα είναι να υλοποιήσουμε τη μέθοδο Jacobi με το πρότυπο MPI ώστε να τρέχει ως πολλαπλές MPI διεργασίες σε πολυεπεξεργαστικά συστήματα κατανεμημένης μνήμης. Οι διεργασίες αυτές, βέβαια, θα πρέπει σε κάποια σημεία να επικοινωνούν. Φυσικά, το *κόστος* αυτής της επικοινωνίας θα πρέπει να είναι μικρότερο από το *κέρδος* της χρησιμοποίησης παράλληλων πράξεων έναντι σειριακών ώστε η *παραλληλοποίηση* να έχει νόημα. Παραπέμπουμε τον αναγνώστη στο [10] για αναλυτική πληροφόρηση σχετικά με το προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων MPI.

Διαμοιρασμός Δεδομένων

Ας αρχίσουμε με την ιδέα για τον τρόπο παραλληλοποίησης της Jacobi. Πολύ αφηρημένα, η ιδέα είναι να *διαμοιράσουμε* τον πίνακα *κατά blocks* σε MPI διεργασίες, η κάθε διεργασία να *επεξεργάζεται* τα δικά της μόνο δεδομένα και όποτε χρειάζεται να *επικοινωνούν* μεταξύ τους. Για παράδειγμα, ένα σημείο όπου θα πρέπει σίγουρα να επικοινωνούν είναι ο έλεγχος σύγκλισης. Στόχος μας είναι να μην αλλάξει η ροή του σειριακού προγράμματος ή η ορθότητά του με την εισαγωγή της παραλληλίας. Στην ουσία, η κάθε διεργασία ακολουθεί την ίδια ροή, εκείνη του σειριακού αλγορίθμου, αλλά την απασχολεί ο υπολογισμός μόνο των δικών της δεδομένων.

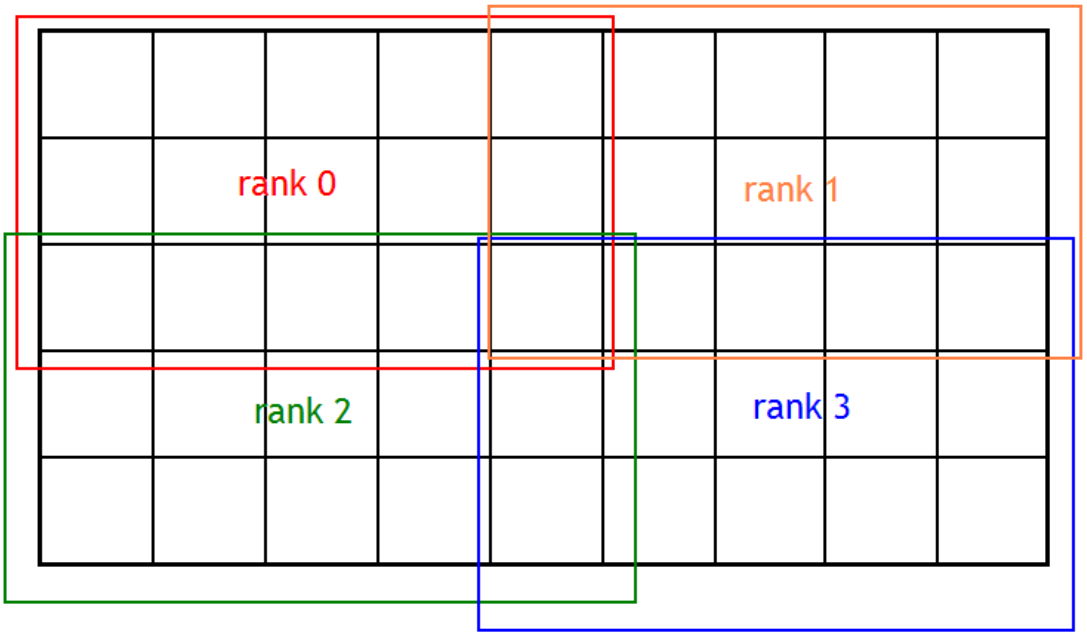
Μια πολύ σημαντική παράμετρος στην αποδοτικότητα του παράλληλου προγραμματισμού είναι η *συμμετρία*. Θέλουμε, λοιπόν, ανεξάρτητα από τον αριθμό των διεργασιών, να έχουν όλες περίπου το ίδιο *υπολογιστικό φορτίο*. Όπως αναλύσαμε προηγουμένως, στον αλγόριθμο Jacobi οι *περιορισμοί* για τις διαστάσεις του πίνακα προς εύρεση είναι να μην είναι μικρότερος του 3×3 (3×3×3 για 3Δ) και να πληροί τη συνθήκη $1.25 \le (N1-1)/(N2-1) \le 8$ (ή $1.25 \le (N1-1)/(N2-1), (N2-1)/(N3-1), (N3-1)/(N1-1) \le 8$ για 3Δ), όπου N1, N2 (και N3 για 3Δ) είναι οι πλευρές του πίνακα. Όμως, επειδή στην ουσία μας ενδιαφέρει η μέθοδος Jacobi ως συστατικό του FMG αλγορίθμου, θα λάβουμε υπ' όψιν και τους περιορισμούς του FMG. Ο βασικός επιπλέον περιορισμός του FMG (που

θα χρησιμοποιήσουμε και εδώ) αφορά τις διαπλεγματικές μεταβάσεις και είναι ότι οι διαστάσεις του πίνακα πρέπει να είναι της μορφής $N = 2^n + 1$, όπου n είναι ένας φυσικός αριθμός.

Αυτό δεν έχει επιρροή στη μέθοδο Jacobi αλλά με αυτό τον τρόπο, στη μέθοδο FMG θα είναι δυνατές και απλές οι διαπλεγματικές μεταβάσεις, καθώς έτσι οι διαστάσεις θα είναι σε κάθε πλέγμα της ίδιας απλής μορφής. Αυτός ακριβώς ο περιορισμός διευκολύνει τη συμμετρία και του παράλληλου προγράμματος Jacobi ώστε οι διαστάσεις του κάθε block (του κάθε πλέγματος) αυτή τη φορά να είναι της ίδιας μορφής. Άρα, μπορούμε να εκμεταλλευτούμε το γεγονός ότι μπορούμε να φτάσουμε μέχρι τον αδρότερο πίνακα με αλλεπάλληλους υποδιπλασιασμούς (σχεδόν) των διαστάσεων. Επιλέγοντας, λοιπόν, αριθμό MPI διεργασιών που να είναι κάποια δύναμη του 2 (αυτό για τους κατάλληλους «υποδιπλασιασμούς»), μπορούμε να κατανείμουμε απολύτως συμμετρικά τον όγκο εργασίας σε αυτές.

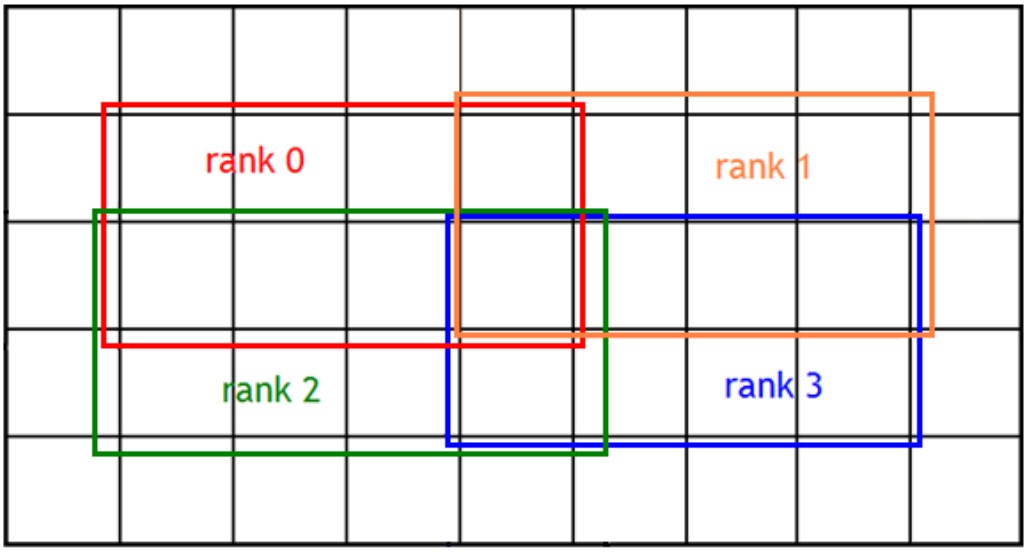
Για παράδειγμα, επιλέγοντας 8 MPI διεργασίες σε έναν πίνακα διαστάσεων 1025×257, θα πρέπει να χωρίσουμε τον πίνακα σε 8 blocks. Αυτά μπορεί να έχουν διάφορα μεγέθη σύμφωνα με τη μέθοδό μας. Μπορεί να είναι 8 γραμμές διαστάσεων 129×257 η καθεμιά (8×1 διεργασίες), 8 στήλες 1025×33 (1×8 διεργασίες), 8 blocks 513×65 (2×4 διεργασίες) ή 8 blocks 257×129 (4×2 διεργασίες). Θα εκτιμήσουμε αυτό τον τρόπο διαμοιρασμού εργασίας στην ανάλυση της παράλληλης (MPI) FMG μεθόδου. Ωστόσο, φαίνεται ότι ο παραπάνω πίνακας δεν έχει διαστάσεις *ακριβώς* πολλαπλάσιες των αντίστοιχων οποιουδήποτε εκ των δυνατών περιπτώσεων διαμοιρασμού. Μάλιστα, τα προτεινόμενα blocks έχουν επιπλέον στοιχεία στην περίμετρό τους.

Για έναν μικρότερο 2Δ πίνακα, διαστάσεων 5×9 και για 2×2 MPI διεργασίες, ο προτεινόμενος τρόπος διαμοιρασμού φαίνεται παρακάτω:



Σχήμα 7: Διαμοιρασμός δεδομένων πίνακα 5x9 σε 2x2 MPI διεργασίες

Όσο για το υπολογιστικό χωρίο της κάθε διεργασίας, αυτό φαίνεται στο Σχήμα 8. Δεν περιλαμβάνει στην ουσία τα συνοριακά στοιχεία αφού είναι σταθερά. Συνεπώς, τυχόν *εσωτερικά bloks* θα έχουν μεγαλύτερο υπολογιστικό χωρίο σε σχέση με τα *συνοριακά blocks.*

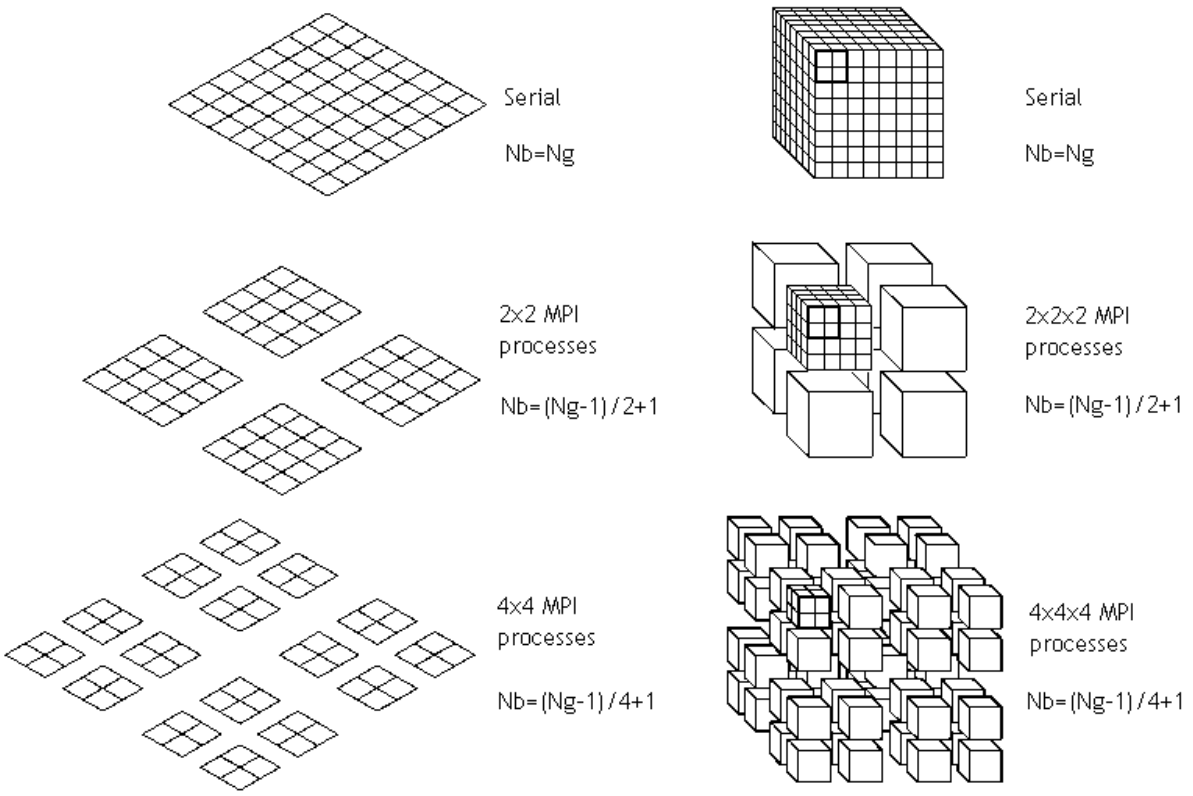


Σχήμα 8: Υπολογιστικό χωρίο κάθε διεργασίας σε πίνακα 5x9 και για 2x2 MPI διεργασίες

Βλέπουμε ότι το block κάθε διεργασίας έχει κοινά τα στοιχεία στην περίμετρό του όπου υπάρχει *γειτονικό* block (ή αλλιώς block γειτονικής διεργασίας). Μάλιστα, το κεντρικό στοιχείο υπάρχει και στα τέσσερα blocks στο παράδειγμά μας. Πρακτικά αυτό σημαίνει περισσότερη μνήμη από ό,τι είναι (εκ πρώτης όψεως τουλάχιστον) αναγκαίο να χρησιμοποιηθεί. Στην πραγματικότητα, αυτό συμβαίνει διότι ο «υποδιπλασιασμός» στον οποίο αναφερθήκαμε παραπάνω αναφέρεται στον τύπο N'=(N-1)/2+1. Να θυμίσουμε ότι αυτός ο τύπος περιγράφει τόσο τις διαπλεγματικές μεταβάσεις ($N_G=(N_g-1)/2+1$ για n φορές, όπου $G=2^n \cdot g$) όσο και τον καθορισμό των διαστάσεων του κάθε block σε ένα πλέγμα ($N_{Bg}=(N_g-1)/2+1$ για p φορές, όπου $P=2^p$ ο αριθμός των MPI διεργασιών που έχουν επιλεγεί στη συγκεκριμένη διάσταση).

Η FMG μέθοδος, λόγω της *αναγκαιότητας* των διαστάσεων του κάθε πλέγματος να είναι της μορφής $N = 2^n + 1$, δεν έχει περιθώρια να μη λάβει υπ' όψιν τέτοιους περιορισμούς. Αυτή η αναγκαιότητα, όπως προείπαμε, αφορά τις συναρτήσεις επέκτασης και περιορισμού και πηγάζει από την επιθυμία μας να είναι δυνατές, απλές και λειτουργικές οι διαπλεγματικές μεταβάσεις και άρα να απαιτούν λίγους σχετικά υπολογισμούς. Συνεπώς, είναι ένα αναγκαίο κακό για τη μέθοδο FMG, *όχι* όμως και για τη Jacobi. Η μέθοδος Jacobi, όπως αναλύσαμε παραπάνω, δεν έχει ανάγκη από τέτοιους περιορισμούς. Μπορεί να θυσιάσει στην MPI εκδοχή της λίγη συμμετρία χωρίς επιπλοκές προκειμένου να αποφύγει όλη αυτή τη διαδικασία. Όμως, επειδή η μέθοδος Jacobi αποτελεί βασική *συνιστώσα* του FMG αλγορίθμου, θα αποδεχτούμε αυτές τις ιδαιτερότητες της FMG και θα τις εφαρμόσουμε και στη μέθοδο Jacobi.

Σε αντιπαράθεση με το Σχήμα 3, όπου δείχνουμε την ιεραρχία πλεγμάτων *οριζόντια*, εδώ (Σχήμα 9) θα παρουσιάσουμε τη διαφοροποίηση των blocks ενός πλέγματος, καθώς πληθαίνουν οι διεργασίες, *κατακόρυφα*. Η διαφοροποίηση των δύο σχημάτων στοχεύει στην καλύτερη κατανόηση των διαφορών μεταξύ του *σειριακού μονοπλεγματικού* αλγορίθμου (1 πλέγμα, 1 διεργασία), του *παράλληλου (με MPI) μονοπλεγματικού* (1 πλέγμα, πολλαπλές διεργασίες), του σειριακού πολυπλεγματικού (πολλαπλά πλέγματα, 1 διεργασία) και του *παράλληλου (με MPI) πολυπλεγματικού* (πολλαπλά πλέγματα, πολλαπλές διεργασίες). Ας δούμε πως διαφοροποιούνται τα blocks με την αύξηση του πλήθους των MPI διεργασιών.



Σχήμα 9: Διαφοροποίηση των blocks για 2Δ πίνακα 9x9 και για 3Δ πίνακα 9x9x9 καθώς πληθαίνουν οι MPI διεργασίες

Στην περίπτωση της σειριακής μεθόδου Jacobi, οι διαστάσεις που μας ένοιαζαν για το υπολογιστικό κομμάτι ήταν εκείνες του αρχικού πίνακα, N1×N2 (ή N1×N2×N3 για 3Δ). Στην περίπτωση της σειριακής FMG μεθόδου, κάθε πλέγμα είχε το δικό του πίνακα και μας ενδιέφεραν οι διαστάσεις του πίνακα πλέγματος $N1_g \times N2_g$ (ή $N1_g \times N2_g \times N3_g$ για 3Δ). Στις παράλληλες MPI εκδόσεις πρέπει να προσθέσουμε έναν δείκτη b στις διαστάσεις για να δηλώσουμε ότι μας ενδιαφέρουν οι διαστάσεις του πίνακα (είτε αρχικού είτε πλεγματικού) του κάθε block. Επομένως, για την παράλληλη (MPI) Jacobi θα είναι $N1_b x N2_b$ (ή $N1_b x N2_b x N3_b$ για 3Δ) και για την παράλληλη (MPI) FMG $N1_{bg} x N2_{bg}$ (ή $N1_{bg} x N2_{bg} x N3_{bg}$ για 3Δ). Ο παρακάτω πίνακας αποσαφηνίζει αυτές τις διαφορές.

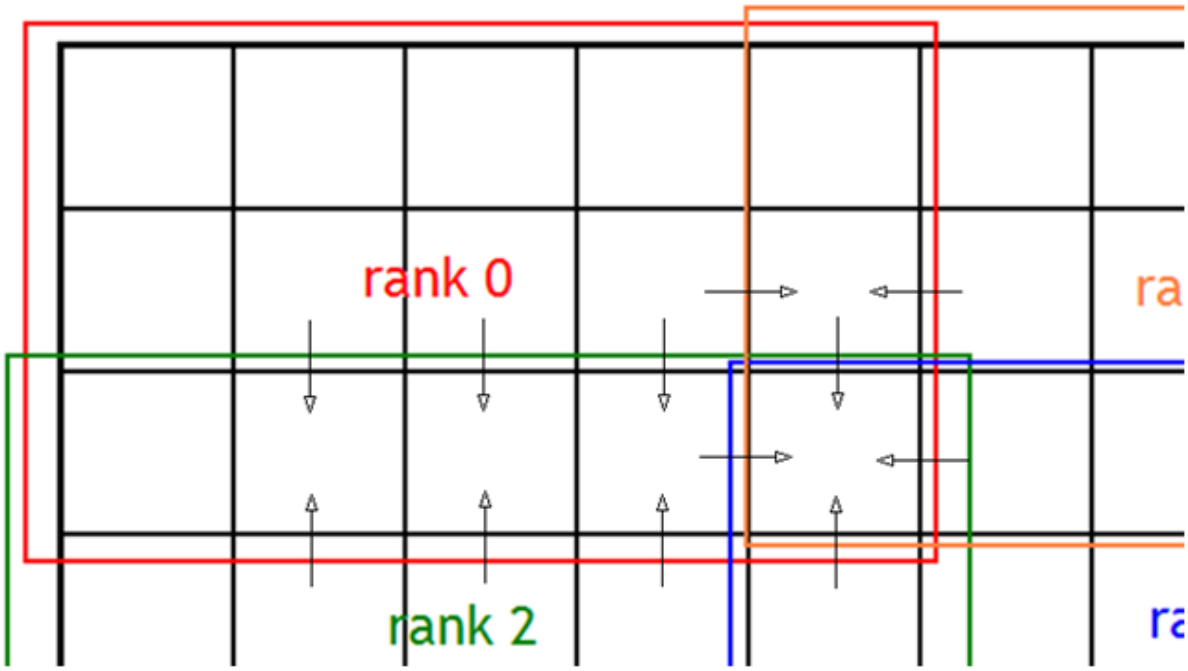
Πίνακας 1 – Διαστάσεις υπολογισμών ανάλογα με τον αλγόριθμο προς υλοποίηση

		ngrids = 1 (g = 1)		ngrids > 1, g = 1…ngrids	
		2D	3D	2D	3D
Serial Algorithm		*Jacobi2D* N1xN2	*Jacobi3D* N1xN2xN3	*FMG2D* $N1_g x N2_g$	*FMG3D* $N1_g x N2_g x N3_g$
Parallel Algorithms	MPI	*Jacobi2D_mpi* $N1_b x N2_b$	*Jacobi3D_mpi* $N1_b x N2_b x N3_b$	*FMG2D_mpi* $N1_{bg} x N2_{bg}$	*FMG3D_mpi* $N1_{bg} x N2_{bg} x N3_{bg}$
	OpenMP	*Jacobi2D_omp* N1xN2	*Jacobi3D_omp* N1xN2xN3	*FMG2D_omp* $N1_g x N2_g$	*FMG3D_omp* $N1_g x N2_g x N3_g$

Ανταλλαγή Δεδομένων

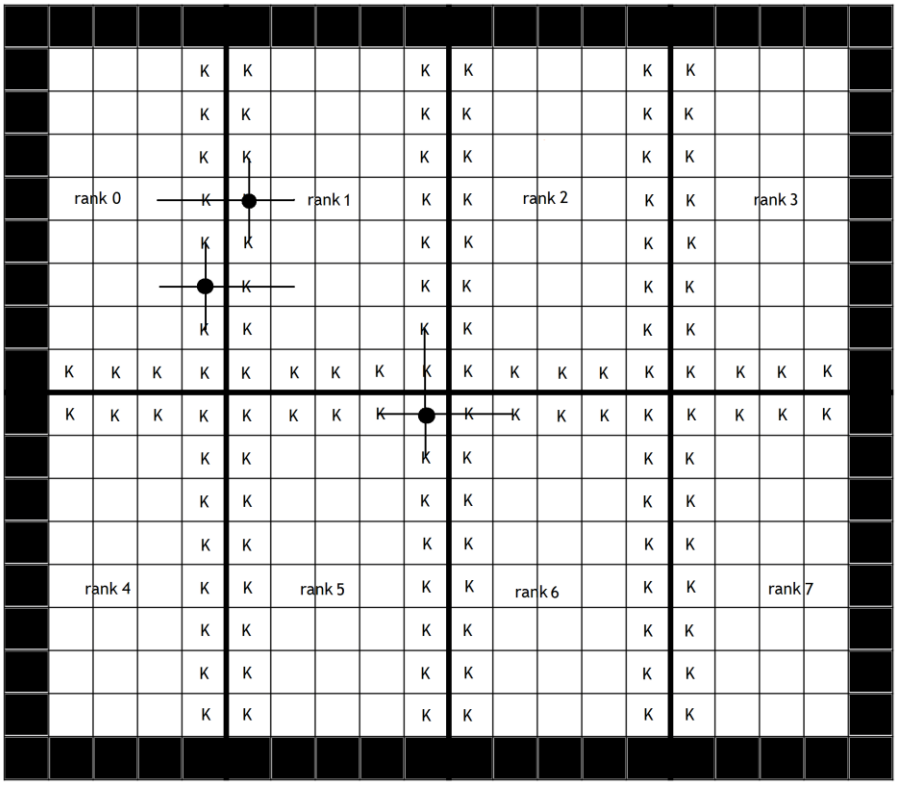
Ας προχωρήσουμε στην *υλοποίηση* της παραλληλίας. Τη βασική πράξη που εφαρμόζαμε στα στοιχεία του πίνακα στη σειριακή μέθοδο Jacobi θα την κάνει τώρα η κάθε MPI διεργασία για όλα τα στοιχεία του δικού της block (εκτός από εκείνα που αντιστοιχούν στα συνοριακά στοιχεία του αρχικού πίνακα), ακόμα και για τα στοιχεία που είναι κοινά με κάποιο άλλο block. Όμως, επειδή για την πράξη αυτή χρειάζονται για τον υπολογισμό κάθε στοιχείου και τα γειτονικά αυτού (πάνω, κάτω, αριστερά, δεξιά και για 3Δ και μπροστά, πίσω), στα στοιχεία που είναι κοινά για δύο blocks, θα πρέπει οι αντίστοιχες διεργασίες να επικοινωνήσουν ανταλλάσσοντας (με τις συναρτήσεις MPI_Send και MPI_Recv) τα δεδομένα που χρειάζεται η καθεμιά. Συνεπώς, για κάθε k θα κάνουμε πρώτα την απαραίτητη ανταλλαγή δεδομένων (με τη συνάρτηση *data_exchange*) και μετά τις πράξεις. Από το Σχήμα 9 μπορούμε εύκολα να διακρίνουμε ότι στις δύο διαστάσεις, τα blocks (οι διεργασίες στην ουσία) ανταλλάσσουν μεταξύ τους γραμμές στοιχείων ενώ στις τρεις διαστάσεις ανταλλάσσουν επίπεδα στοιχείων. Όπως το κόστος υπολογισμών, λοιπόν, έτσι και το κόστος επικοινωνίας είναι μιας τάξης μεγαλύτερο. Πάντως, μετά από τις ανταλλαγές αυτές και αν το k είναι κατάλληλο, η κάθε διεργασία θα πρέπει να κάνει έλεγχο σύγκλισης για το δικό της block. Έπειτα, θα επικοινωνήσουν (με τη βοήθεια της MPI_All_Reduce) όλες οι διεργασίες αυτή τη φορά, για να αποφανθούν αν έχει επιτευχθεί η ζητούμενη σύγκλιση. Εάν έστω και σε μια διεργασία δεν επιτεύχθηκε η σύγκλιση, θα συνεχίσουν όλες μέχρι τον επόμενο έλεγχο. Εν τω μεταξύ, προγραμματιστικά, ο έλεγχος αυτός θα γίνεται ανά έναν συγκεκριμένο αριθμό επαναλήψεων που καθορίζεται κατάλληλα ανάλογα με το μέγεθος του πίνακα επίλυσης και το κριτήριο σύγκλισης σε σχέση με τις συνοριακές τιμές, τις αρχικές τιμές και τις τιμές της «πηγής» f. Με αυτό τον τρόπο εξασφαλίζεται ότι θα χρειαστούν σχετικά λίγοι έλεγχοι σύγκλισης και λίγες επιπλέον επαναλήψεις μετά την πρώτη πραγματική σύγκλιση.

Στο παρακάτω σχήμα βλέπουμε τι γίνεται στο προηγούμενο παράδειγμα αλλά μόνο για τις ανταλλαγές δεδομένων στις οποίες παίρνει μέρος η MPI διεργασία με rank=0. Φαίνονται τα στοιχεία που έχει στη διάθεσή της και τα στοιχεία που χρειάζεται για τις πράξεις της:

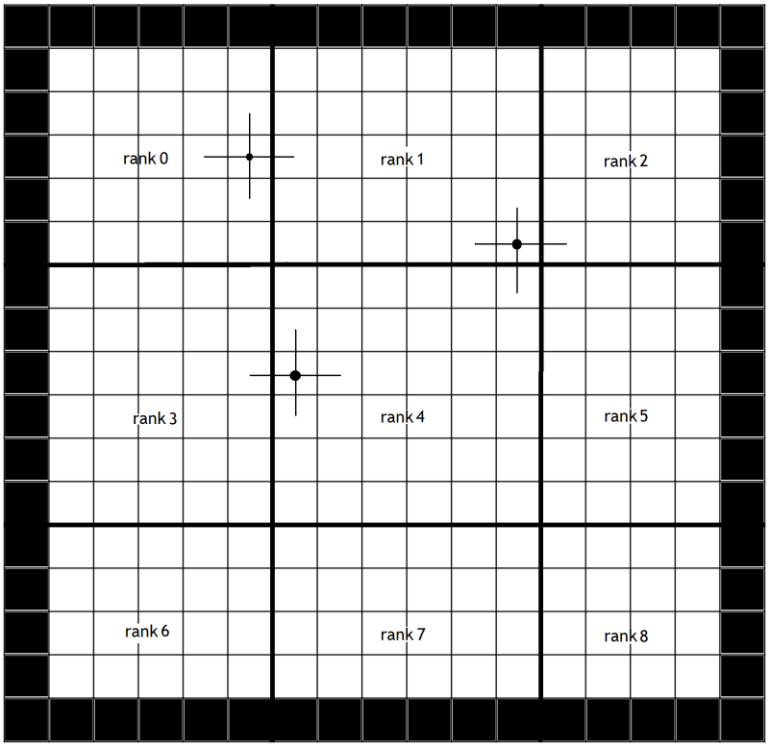


Σχήμα 10: Ανταλλαγή δεδομένων για τη διεργασία με rank=0 σε πίνακα 5x9 και για 2x2 MPI διεργασίες

Τα σύνορα είναι σταθερά και για αυτό δε χρειάζονται ανταλλαγές δεδομένων εκεί. Τα βέλη *καταλήγουν* στο στοιχείο για τον υπολογισμό του οποίου είναι απαραίτητο το στοιχείο από το οποίο *ξεκινά* το βέλος. Αξίζει να επισημάνουμε ότι με αυτό τον τρόπο τα στοιχεία που είναι κοινά για δύο διεργασίες υπολογίζονται δύο φορές, μια για κάθε διεργασία που τα διαθέτει. Το αποτέλεσμα είναι σαφώς το ίδιο αλλά γίνονται επιπλέον πράξεις. Μπορούμε να αποφύγουμε αυτές τις περιττές πράξεις αλλά θα πρέπει να εισάγουμε πολυπλοκότητα στον αλγόριθμο που έπειτα πιθανότατα να μας περιορίσει. Κάτι που σίγουρα *δε* μπορούμε να αποφύγουμε (λόγω της αναγκαιότητάς του στην FMG μέθοδο) είναι ο μεγαλύτερος όγκος ανταλλαγής δεδομένων σε κάθε επανάληψη λόγω των κοινών στοιχείων μεταξύ των διεργασιών. Για παράδειγμα, στα δύο επόμενα σχήματα βλέπουμε ότι για τον ίδιο αρχικό πίνακα (17×17) προκύπτει με τη μέθοδό μας και 8 MPI διεργασίες μεγαλύτερος συνολικός όγκος δεδομένων προς ανταλλαγή από ό,τι με τη μέθοδο χωρίς κοινές γραμμές ή στήλες και για 9 MPI διεργασίες. Στην πρώτη περίπτωση προκύπτουν 132 στοιχεία για ανταλλαγή σε κάθε επανάληψη ενώ αντίστοιχα στη δεύτερη (όπου έχουμε και μια επιπλέον MPI διεργασία) προκύπτουν 120.



Σχήμα 11: Ανταλλαγή δεδομένων για αρχικό πίνακα 17x17 και για 2x4 MPI διεργασίες (K: Κοινά στοιχεία)



Σχήμα 12: Ανταλλαγή δεδομένων για αρχικό πίνακα 17x17 και για 3x3 MPI διεργασίες

Παρατηρώντας τα δύο παραπάνω σχήματα, εύκολα παρατηρούμε την παντελή απουσία συμμετρίας στο δεύτερο σχήμα. Όπως έχουμε τονίσει αρκετές φορές, όμως, δεν είναι λόγοι αισθητικής που μας κάνουν να περιορίζουμε το πλήθος MPI διεργασιών σε δύναμη του 2 αλλά εγγενείς απαιτήσεις των συναρτήσεων διαπλεγματικών μεταβάσεων που έχουμε υλοποιήσει. Αν τις είχαμε υλοποιήσει διαφορετικά αλλά με ενιαίο τύπο μετάβασης από πλέγμα σε πλέγμα (έστω π.χ $N_G=(N_g-3)/5+3$) τότε πάλι θα θέλαμε συμμετρία αλλά τότε θα είχαμε διαφορετικά αποδεκτά μεγέθη πινάκων, διαφορετικό διαμοιρασμό των δεδομένων σε blocks, άρα και διαφορετικό περιορισμό στο πλήθος των MPI διεργασιών. Αν όμως χρησιμοποιούσαμε διαφορετικές συναρτήσεις διαπλεγματικών μεταβάσεων ανάλογα με το κάθε block, τότε η συμμετρία θα έπρεπε να υπάρχει μόνο για τα blocks που χρησιμοποιούν το ίδιο ζευγάρι interpolate και restrict.

Ας αφιερώσουμε και λίγο χρόνο για να απομυθοποιήσουμε λίγο τη συμμετρικότητα του πίνακα στο Σχήμα 11. Αρχικά, το υπολογιστικό χωρίο (ο φόρτος εργασίας) δεν είναι σε όλα τα blocks ίδιου μεγέθους. Τα *γωνιακά* blocks χρειάζεται να υπολογίσουν μιία στήλη λιγότερο από τα υπόλοιπα. Αυτό διότι στο πρόβλημά μας οι συνοριακές συνθήκες είναι σταθερές. Συνεπώς, αν ένα block έχει *συνοριακά* στοιχεία, αυτό δε θα κάνει υπολογισμούς για εκείνα τα στοιχεία. Επομένως, αν είχαμε blocks *χωρίς* συνοριακά στοιχεία (δηλαδή για πλήθη MPI διεργασιών από 4×4 και πάνω), τότε όλα αυτά θα είχαν το ίδιο υπολογιστικό φορτίο. Χάνουμε, λοιπόν, λίγη συμμετρία στο *υπολογιστικό* κομμάτι. Για τα περισσότερα μεγέθη πίνακα, βέβαια, και ειδικά στην τρισδιάστατη περίπτωση αυτό έχει μικρή σημασία καθώς οι υπολογισμοί στο εσωτερικό κάθε block έχουν κόστος μιας τάξης μεγαλύτερο από τους υπολογισμούς στα σύνορά του. Ούτε το κομμάτι της *επικοινωνίας* είναι απόλυτα συμμετρικό, καθώς τα γωνιακά σε σχέση με τα υπόλοιπα συνοριακά (και πόσο μάλλον σε σχέση με τα τυχόν μη συνοριακά) blocks χρειάζεται να ανταλλάξουν λιγότερα δεδομένα. Άρα, λιγότεροι υπολογισμοί και λιγότερη επικοινωνία για τα γωνιακά (αλλά και για όλα τα συνοριακά blocks. Αυτό δε μας ενοχλεί και σίγουρα όχι τόσο ώστε να το αλλάξουμε εισάγοντας επιπλέον πολυπλοκότητα.

Αυτό που μπορούμε να κάνουμε είναι να επεκτείνουμε κατά μια γραμμή ή/και στήλη (στην 3Δ περίπτωση επεκτείνουμε κατά ένα αντίστοιχο επίπεδο) το κάθε block προς τις μη συνοριακές πλευρές (ή επιφάνειές του για 3Δ) ώστε να δεχτεί εκεί τα δεδομένα που χρειάζεται κάθε φορά από τα γειτονικά του blocks. Με αυτό τον τρόπο χαλάμε και τη συμμετρία των συνοριακών blocks μεταξύ τους και σε σχέση με τα εσωτερικά blocks όσον αφορά το μέγεθος του πίνακά τους. Αυτό μας επηρεάζει ελάχιστα και μόνο στη χρησιμοποίηση μνήμης, ενώ έχει ένα πολύ μεγάλο όφελος: την αλγοριθμική *απλότητα* που προσδίδεται στο μέρος των υπολογισμών στις συναρτήσεις που χρειάζονται επικοινωνία. Εδώ αφορά μόνο τη Jacobi αλλά στην FMG μέθοδο απλοποιεί σημαντικά και τις calc_res και restrict.

Υλοποίηση

Κορμός του Αλγορίθμου

Τελικά, η μέθοδος υλοποιείται σε 2Δ με τη συνάρτηση *jacobi_solve_mpi* ως εξής:

```
/* Jacobi Solver in 2D MPI */
for (k = 0 to ITER) {

      /* exchange of necessary data */
      data_exchange_NB(v^old);

      /* computations in interior points */
      for x = U to N1_b - D
            for y = L to N2_b - R
                  v^new[x][y] = (v^old[x - 1][y] + v^old[x + 1][y] +
                              v^old[x][y - 1] + v^old[x][y + 1]) / 4 -
                              h1h2f[x][y] / 4;

      pointer_swap(&v^new, &v^old);

      /* convergence check */
      if all_converged(v^old, v^new, cnverr)
            return;
}
```

Για τρεις διαστάσεις θα είναι:

```
for (k = 0 to ITER) {

      /* exchange of necessary data */
      data_exchange_NB(v^old);

      /* interior points */
      for x = U to N1_b - D
            for y = L to N2_b - R
                  for z = F to N3_b - B
                        v^new[x][y][z] = (v^old[x - 1][y][z] +
                              v^old[x + 1][y][z] + v^old[x][y - 1][z] +
                              v^old[x][y + 1][z] + v^old[x][y][z - 1] +
                              v^old[x][y][z + 1]) / 6 –
                              h1h2h3f[x][y][z] / 6;

      pointer_swap(&v^new, &v^old);

      /* convergence check */
      if all_converged(v^old, v^new, cnverr)
            return;
}
```

Ο αλγόριθμος διαφοροποιείται σε σχέση με το σειριακό σε τρία βασικά σημεία, την *ανταλλαγή δεδομένων*, την αλλαγή στα *όρια του υπολογιστικού χωρίου* και την αλλαγή στον *έλεγχο σύγκλισης*. Ας ξεκινήσουμε από το τελευταίο. Τόσο για 2Δ όσο και για 3Δ, στην *all_converged* η κάθε MPI διεργασία κάνει έλεγχο σύγκλισης για το δικό της υπολογιστικό χωρίο. Κοινωνεί το αποτέλεσμα του ελέγχου ώστε να αποφανθούν εάν επιτεύχθηκε η επιθυμητή σύγκλιση συνολικά. Εάν έστω και μια διεργασία δεν συνέκλινε, όλες οι διεργασίες συνεχίζουν τη διαδικασία επίλυσης μέχρι τον επόμενο έλεγχο. Στην πράξη, η κάθε διεργασία εκχωρεί την τιμή 1 σε μια τοπική μεταβλητή (με αρχική τιμή 0) εάν έχει συγκλίνει στο δικό της χωρίο. Με τη βοήθεια της MPI_All_Reduce ελέγχεται εάν έστω και μια διεργασία δεν έχει εκχωρήσει την τιμή 1 στην αντίστοιχη μεταβλητή.

Η δεύτερη διαφοροποίηση αφορά την αλλαγή στα όρια του υπολογιστικού χωρίου της κάθε διεργασίας. Στο σειριακό αλγόριθμο είχαμε στην ουσία μια διεργασία κι άρα ένα μόνο υπολογιστικό χωρίο περιβαλλόμενο από τα συνοριακά στοιχεία. Εδώ έχουμε περισσότερες και ανάλογα με το πλήθος τους, το κάθε block μπορεί να έχει συνοριακά στοιχεία σε οποιαδήποτε πλευρά (για 3Δ επιφάνειά) του ή ακόμα και σε καμιά. Συνεπώς, το υπολογιστικό χωρίο δύναται να είναι διαφορετικά τοποθετημένο στο block δεδομένων από διεργασία σε διεργασία. Ορίζουμε, λοιπόν, τις μεταβλητές U, D, L, R (για 3Δ και F, B) για κάθε διεργασία, που περιέχουν στην ουσία τις *μεταβολές* που πρέπει να επιβληθούν στις συντεταγμένες ώστε να *ορίζουν* το υπολογιστικό μας χωρίο. Αυτές τις έχουμε ορίσει κατάλληλα στην αρχή του προγράμματος σύμφωνα με τον διαμοιρασμό δεδομένων.

Η ουσιαστικότερη διαφοροποίηση, φυσικά, είναι η συνάρτηση ανταλλαγής δεδομένων *data_exchange_NB*.

Συνάρτηση Ανταλλαγής Δεδομένων

Έχουμε αναφερθεί στα δεδομένα που χρειάζεται η κάθε διεργασία από τις γειτονικές της για τον υπολογισμό των αντίστοιχων κοινών στοιχείων τους. Σε δύο διαστάσεις (2Δ) αυτή η επικοινωνία αφορά γραμμές και στήλες ενώ σε τρεις διαστάσεις (3Δ) αφορά επιφάνειες. Με τη συνάρτηση data_exchange_NB θα γίνουν όλες οι απαραίτητες ανταλλαγές δεδομένων. Το «NB» δηλώνει non-blocking επικοινωνία, δηλαδή ότι δεν υπάρχει σειριακή αναμονή για *κάθε* εντολή αποστολής μέχρι την ολοκλήρωση της αντίστοιχης λήψης. Αντίθετα, θα εκτελούνται όλες οι εντολές αποστολής δεδομένων στην αρχή και η αναμονή θα είναι *συνολική*. Βεβαίως, οι υπολογισμοί θα συνεχίσουν μόλις γίνει η τελευταία λήψη. Ωστόσο, τα αντίστοιχα χρονικά διαστήματα αναμονής μετά την κάθε εντολή αποστολής δύναται να τέμνονται ή και να συμπίπτουν. Όσο για τα δεδομένα που

χρειάζεται η κάθε διεργασία, αυτά είπαμε ότι ζητούνται από τις διεργασίες με τις οποίες έχει κοινά στοιχεία και τα χρειάζεται για τον υπολογισμό αυτών ακριβώς των στοιχείων.

Σε 2Δ, η data_exchange_NB *υλοποιείται* ως εξής:

```
/* exchange of data between MPI processes in 2D */
if (neigh.LEFT >= 0) {

      /* sending my second column to my LEFT neighbour */
      send_NB(v_old[U...N1_b - D][1], neigh.LEFT);

      /* receiving my previous column from my LEFT neighbour */
      recv_NB(v_old[U...N1_b - D][-1], neigh.LEFT);
}
if (neigh.RIGHT >= 0) {

      /* sending my second-to-last column to my RIGHT neighbour */
      send_NB(v_old[U...N1_b - D][N2_b - 2], neigh.RIGHT);

      /* receiving my next column from my RIGHT neighbour */
      recv_NB(v_old[U...N1_b - D][N2_b], neigh.RIGHT);
}
if (neigh.UP >= 0) {

      /* sending my second line to my UP neighbour */
      send_NB(v_old[1][L...N2_b - R], neigh.UP);

      /* receiving my previous line from my UP neighbour */
      recv_NB(v_old[-1][L...N2_b - R], neigh.UP);
}
if (neigh.DOWN >= 0) {

      /* sending my second-to-last line to my DOWN neighbour */
      send_NB(v_old[N1_b - 2][L...N2_b - R], neigh.DOWN);

      /* receiving my next line from my DOWN neighbour */
      recv_NB(v_old[N1_b][L...N2_b - R], neigh.DOWN);
}
wait_all;
```

Σε 3Δ, η υλοποίηση έχει ως εξής:

```
if (neigh.FRONT >= 0) {

      /* sending my second plane to my FRONT neighbour */
      send_NB(v_old[U...N1_b - D][L...N2_b - R][1], neigh. FRONT);

      /* receiving my previous plane from my FRONT neighbour */
      recv_NB(v_old[U...N1_b - D][L...N2_b - R][-1], neigh. FRONT);
}
if (neigh.BACK >= 0) {
```

```
        /* sending my second-to-last plane to my BACK neighbour */
        send_NB(v_old[U...N1_b - D][L...N2_b - R][N3_b - 2], neigh. BACK);

        /* receiving my next plane from my BACK neighbour */
        recv_NB(v_old[U...N1_b - D][L...N2_b - R][N3_b], neigh. BACK);
    }
    if (neigh.LEFT >= 0) {

        /* sending my second plane to my LEFT neighbour */
        send_NB(v_old[U...N1_b - D][1][F...N3_b - B], neigh.LEFT);

        /* receiving my previous plane from my LEFT neighbour */
        recv_NB(v_old[U...N1_b - D][-1][F...N3_b - B], neigh.LEFT);
    }
    if (neigh.RIGHT >= 0) {

        /* sending my second-to-last plane to my RIGHT neighbour */
        send_NB(v_old[U...N1_b - D][N2_b - 2][F...N3_b - B], neigh.RIGHT);

        /* receiving my next plane from my RIGHT neighbour */
        recv_NB(v_old[U...N1_b - D][N2_b][F...N3_b - B], neigh.RIGHT);
    }
    if (neigh.UP >= 0) {

        /* sending my second plane to my UP neighbour */
        send_NB(v_old[1][L...N2_b - R][F...N3_b - B], neigh.UP);

        /* receiving my previous plane from my UP neighbour */
        recv_NB(v_old[-1][L...N2_b - R][F...N3_b - B], neigh.UP);
    }
    if (neigh.DOWN >= 0) {

        /* sending my second-to-last plane to my DOWN neighbour */
        send_NB(v_old[N1_b - 2][L...N2_b - R][F...N3_b - B], neigh.DOWN);

        /* receiving my next plane from my DOWN neighbour */
        recv_NB(v_old[N1_b][L...N2_b - R][F...N3_b - B], neigh.DOWN);
    }
    wait_all;
```

Το γεγονός ότι περνάμε τα στοιχεία που λαμβάνουμε εξωτερικά του πίνακά μας δίπλα στα «κοινά» στοιχεία βοηθά στην απλότητα του αλγορίθμου στο μέρος των υπολογισμών. Όσο για τη δομή neigh (neighbour), αυτή ορίζεται σύμφωνα με το διαμοιρασμό των δεδομένων στην αρχή. Οι μεταβλητές της μας δίνουν πληροφορίες σχετικά με την ύπαρξη ή όχι *γειτονικών* διεργασιών στην αντίστοιχη μεριά. Αν υπάρχει γειτονική διεργασία, τότε η αντίστοιχη μεταβλητή θα περιέχει το rank της.

Προεκτάσεις

Όπως συνέβη και με τις σειριακές εκδόσεις, έτσι και εδώ ο παράλληλος αλγόριθμος Jacobi θα αποτελέσει τον εξομαλυντή αλλά και τον βασικό επιλυτή αδρότερου πλέγματος στην *παράλληλη* έκδοση του FMG αλγορίθμου.

Κεφάλαιο 7

Παραλληλοποίηση του Αλγορίθμου FMG με Χρήση του MPI

Περιγραφή

Εδώ θα υλοποιήσουμε την πολυπλεγματική μέθοδο εμφωλευμένης επανάληψης FMG με το πρότυπο MPI. Την *παραλληλοποίηση* σε κατανεμημένη μνήμη της βασικής συνάρτησης του FMG αλγορίθμου, δηλαδή της jacobi_solve, την είδαμε στην ακριβώς προηγούμενη ενότητα. Με τον ίδιο τρόπο, θα παραλληλοποιήσουμε και τις υπόλοιπες συναρτήσεις του FMG.

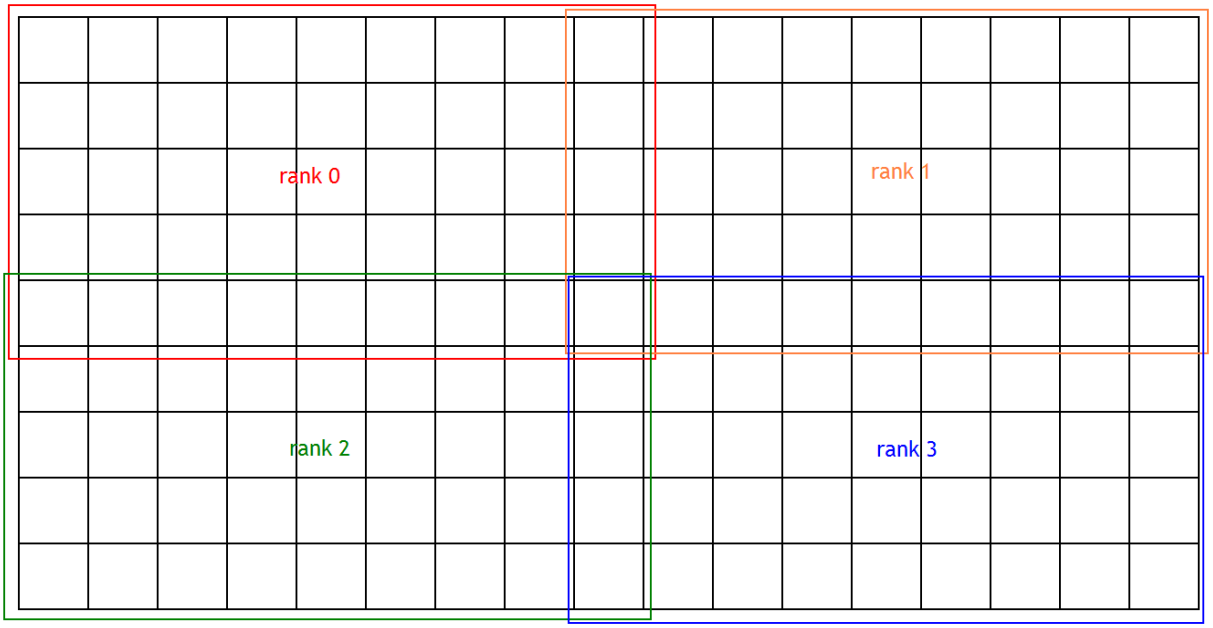
Όπως και πριν, η ιδέα είναι να *διαμοιράσουμε* τον πίνακα κατά blocks σε MPI διεργασίες. Η κάθε διεργασία θα ακολουθεί τη ροή του σειριακού αλγορίθμου αλλά θα εκτελεί τις όποιες πράξεις μόνο για τα δικά της δεδομένα. Όπου χρειαστεί, φυσικά, θα πρέπει να γίνεται και ανταλλαγή δεδομένων μεταξύ των διεργασιών. Τέτοιες ανταλλαγές ήδη είδαμε στη συνάρτηση jacobi_solve. Θα δούμε και στις συναρτήσεις διαπλεγματικού περιορισμού (restrict) και υπολογισμού υπολοίπου (calc_res). Μάλιστα, η ανταλλαγή δεδομένων data_exchange_NB που γίνεται στη jacobi_solve πριν τις πράξεις είναι η ίδια που θα γίνει και στις restrict και calc_res πριν από τις δικές τους αντίστοιχες πράξεις. Βεβαίως, επειδή δεν προβλέπεται ανταλλαγή δεδομένων στις συναρτήσεις interpolate και correct, αυτό δε σημαίνει ότι αυτές θα μείνουν ως είχαν στη σειριακή έκδοση. Θα πρέπει να γίνουν σε όλες τις συναρτήσεις αλλαγές ώστε ανάλογα με τη θέση του block στον πίνακα (και άρα τη θέση του υπολογιστικού χωρίου στο block αυτό) να γίνονται κατάλληλα οι πράξεις. Για παράδειγμα, ένα γωνιακό block δεν πρέπει να κάνει πράξεις σε όλα τα στοιχεία του αλλά μόνο στα εσωτερικά. Αντίθετα, ένα εσωτερικό block πρέπει να κάνει πράξεις σε όλα τα στοιχεία του αφού δεν περιλαμβάνει συνοριακά στοιχεία.

Διαμοιρασμός και Χωρισμός Δεδομένων

Είναι πολύ σημαντικό να κατανοήσουμε τις διαφορές μεταξύ του *διαμοιρασμού* δεδομένων, του *χωρισμού* δεδομένων και της *ανταλλαγής* δεδομένων. Ο χωρισμός δεδομένων αναφέρεται στη μορφή των πινάκων επίλυσης των διεργασιών για κάθε πλέγμα και κατ' επέκταση, στη μορφή των blocks. Ο διαμοιρασμός δεδομένων αναφέρεται στη

διανομή των αρχικών διακριτοποιημένων πινάκων επίλυσης στις αντίστοιχες διεργασίες, δηλαδή στην εκχώρηση των κατάλληλων δεδομένων στα στοιχεία των blocks. Η διανομή δεν είναι μέρος του αλγορίθμου FMG αλλά πιθανότατα προηγείται αυτού και βάσει αυτής προσδιορίζεται ο χωρισμός των δεδομένων. Η ανταλλαγή δεδομένων αφορά οποιαδήποτε επικοινωνία μεταξύ των διεργασιών. Κυρίως, εννοούμε την ανταλλαγή δεδομένων που υφίσταται μεταξύ γειτονικών διεργασιών (ή μάλλον διεργασιών με γειτονικά blocks).

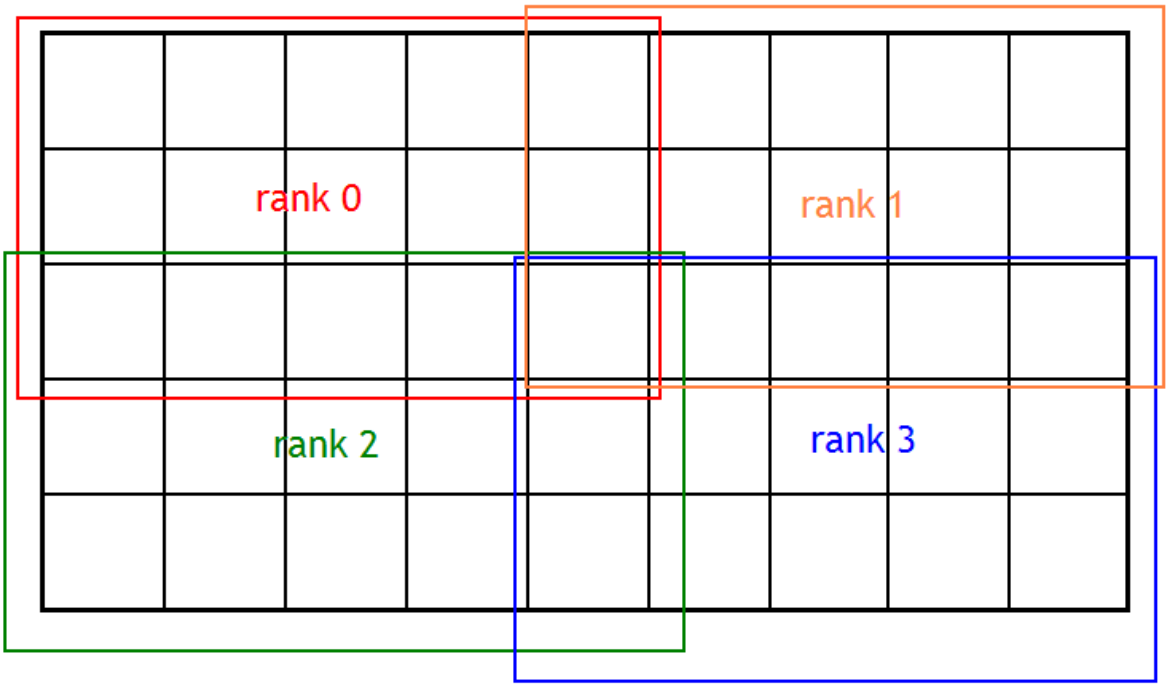
Ας δούμε κάποιες βασικές προϋποθέσεις που πρέπει ακόμα να πληροί ο παράλληλος FMG αλγόριθμος. Πρέπει να επισημάνουμε ότι αντίθετα με τη σειριακή έκδοση, δε μπορούμε να πάμε μέχρι το αδρότερο δυνατό πλέγμα (εκείνο που έχει μια διάσταση ίση με 3). Μάλιστα, αν είναι $N1=2^{n1}+1$, $N2=2^{n2}+1$ (για 3Δ και $N3=2^{n3}+1$) για τον αρχικό πίνακα $N1\times N2$ ($N1\times N2\times N3$ για 3Δ) και $P1=2^{p1}$ και $P2=2^{p2}$ (για 3Δ και $P3=2^{p3}$) για το σχήμα MPI διεργασιών $P1\times P2$ ($P1\times P2\times P3$ για 3Δ) που επιθυμούμε, τότε θα πρέπει να ισχύει: $ngrids \le n1-p1$ και $ngrids \le n2-p2$ (για 3Δ και $ngrids \le n3-p3$). Μπορούμε, δηλαδή, να πάμε μέχρι το αδρότερο δυνατό ανά διεργασία πλέγμα. Παραθέτουμε σαν παράδειγμα έναν αρχικό πίνακα 9×17 για 2×2 MPI διεργασίες:



Σχήμα 13: Διαμοιρασμός δεδομένων πίνακα 9x17 σε 2x2 MPI διεργασίες

Ο πίνακας έχει διαστάσεις $N1\times N2 = (2^{n1}+1)\times(2^{n2}+1) = (2^3+1)\times(2^4+1) = 9\times17$. Στην περίπτωση του σειριακού αλγορίθμου, θα μπορούσαμε να χρησιμοποιήσουμε μέχρι και $ngrids = \min(n1,n2) = 3$ πλέγματα για την εύρεση της λύσης. Επειδή θα μοιράσουμε τα δεδομένα σε $P1\times P2 = 2^{p1}\times2^{p2} = 2^1\times2^1 = 2\times2$ MPI διεργασίες όμως, μπορούμε να χρησιμοποιήσουμε μέχρι $\min(n1-p1,n2-p2) = \min(2,3) = 2$ πλέγματα. Όπως φαίνεται και

παρακάτω, στο αδρότερο πλέγμα (κι ας μην είναι το αδρότερο δυνατό για τον αρχικό πίνακα) η κάθε διεργασία έχει από έναν 3×5 πίνακα και συνεπώς δε μπορεί να προχωρήσει σε επόμενο, αδρότερο πλέγμα.



Σχήμα 14: Χωρισμός δεδομένων πίνακα αδρότερου πλέγματος σε 2x2 MPI διεργασίες

Υλοποίηση

Κορμός του Αλγορίθμου

Ας προχωρήσουμε τώρα στην υλοποίηση της παραλληλίας, παραθέτοντας το βασικό κορμό του FMG:

```
/* Full MultiGrid algorithm */
v^H = solve_mpi(f^H, FMG_ITER); // computing exact solution

/* going right to groups of higher v-cycles */
for (g = H; g > h; g /= 2) {
      fg = g / 2;

      /* going up right before every first v-cycle */
      interpolate_mpi(v^g → v^fg); // solution prolongation

      /* amount of v-cycles according to given ncycles */
      for (c = 0; c < ncycles; c++) {
```

```
                /* going down in v-cycle */
                for (gg = fg; gg < H; gg *= 2) {
                        v^gg = solve_mpi(f^gg, a_1); // solution relaxation
                        r^gg = calc_res_mpi(v^gg, f^gg); // residual calculation
                        restrict_mpi(r^gg → r^2gg); // residual restriction
                        initmtrx(e^2gg); // initialization with 0.0
                }

                e^H = solve_mpi(r^H, FMG_ITER); // computing exact error

                /* going up in v-cycle */
                for (gg = H / 2; gg > fg; gg /= 2) {
                        interpolate_mpi(e^2gg → e^gg); // error prolongation
                        correct_mpi(v^gg, e^gg); // solution correction
                        v^gg = solve_mpi(f^gg, a_2); // solution relaxation
                }
        }
}
```

Συναρτήσεις του Αλγορίθμου

Παραθέτουμε τις βασικές συναρτήσεις που χρησιμοποιεί ο FMG (εκτός από τη συνάρτηση-εξομαλυντή *jacobi_solve_mpi* και τη *data_exchange_NB* που έχουμε παραθέσει στην προηγούμενη ενότητα).

Συνάρτηση διόρθωσης

Η *correct_mpi* δεν προϋποθέτει επικοινωνία μεταξύ των διεργασιών:

```
/*
 * Correction of the approximated solution by
 * subtracting the calculated error
 */

/* interior points */
for x = U to N1_bg - D
        for y = L to N2_bg - R
                v_bg[x][y] += e_bg[x][y];
```

Στις τρεις διαστάσεις θα είναι:

```
/* interior points */
for x = U to N1_bg - D
        for y = L to N2_bg – R
                for z = F to N3_bg - B
                        v_bg[x][y][z] += e_bg[x][y][z];
```

Συνάρτηση υπολογισμού υπολοίπου

Η συνάρτηση υπολογισμού υπολοίπου *calc_res_mpi* προβλέπει επικοινωνία μεταξύ των διεργασιών πριν τους υπολογισμούς:

```
/* Calculation of (minus) the residual */

/* exchange of necessary data */
data_exchange_NB(v_bg);

/* interior points */
for x = U to N1_bg - D
      for y = L to N2_bg - R
            r_bg[x][y] = - (v_bg[x - 1][y] + v_bg[x + 1][y] +
                              v_bg[x][y - 1] + v_bg[x][y + 1] –
                              4 * v_bg[x][y]) / h1h2 + f_bg[x][y];
```

Στις τρεις διαστάσεις θα είναι:

```
/* exchange of necessary data */
data_exchange_NB(v_bg);

/* interior points */
for x = U to N1_bg - D
      for y = L to N2_bg – R
            for z = F to N3_bg - B
                  r_bg[x][y][z] = - (v_bg[x - 1][y][z] + v_bg[x + 1][y][z] +
                              v_bg[x][y - 1][z] + v_bg[x][y + 1][z] +
                              v_bg[x][y][z - 1] + v_bg[x][y][z + 1] –
                              6 * v_bg[x][y][z]) / h1h2h3 + f_bg[x][y][z];
```

Συνάρτηση επέκτασης

Η συνάρτηση επέκτασης *interpolate_mpi* δεν προϋποθέτει ανταλλαγή δεδομένων μεταξύ των blocks:

```
/* coarse-to-fine prolongation using interpolation */

/* elements that are copies */
for x_b2g = U to N1_b2g - D
      for x_b2g = L to N2_b2g - R
            v_bg[2 * x_b2g][2 * y_b2g] = v_b2g[x_b2g][y_b2g];

/* even-numbered columns, interpolating vertically */
for x_bg = 1 to (N1_bg – 1) with step 2
      for y_bg = 2 * L to N2_bg – R with step 2
            v_bg[x_bg][y_bg] = 0.5 * (v_bg[x_bg + 1][y_bg] + v_bg[x_bg - 1][y_bg]);
```

```
/* odd-numbered columns, interpolating horizontally */
for x_bg = U to N1_bg - D
      for y_bg = 1 to N2_bg – 1 with step 2
            v_bg[x_bg][y_bg] = 0.5 * (v_bg[x_bg][y_bg + 1] + v_bg[x_bg][y_bg - 1]);
```

Ο αλγόριθμος στις τρεις διαστάσεις έχει ως εξής:

```
/* elements that are copies */
for x_2g = U to N1_2g - D
      for y_2g = L to N2_2g - R
            for z_2g = F to N3_2g – B
                  v_g[2 * x_2g][2 * y_2g][2 * z_2g] = v_2g[x_2g][y_2g][z_2g];

/* even-x-even-y-numbered columns, interpolating vertically */
for x_g = 1 to (N1_g – 1) with step 2
      for y_g = 2 * L to (N2_g – R) with step 2
            for z_g = 2 * F to (N3_g – B) with step 2
                  uf[x_g][y_g][z_g] = 0.5 * (v_g[x_g + 1][y_g][z_g] +
                                             v_g[x_g - 1][y_g][z_g]);

/* even-x-odd-y-numbered columns, interpolating z-horizontally */
for x_g = U to N1_g - D
      for y_g = 2 * L to (N2_g – R) with step 2
            for z_g = 1 to (N3_g – 1) with step 2
                  v_g[x_g][y_g][z_g] = 0.5 * (v_g[x_g][y_g][z_g + 1] +
                                              v_g[x_g][y_g][z_g - 1]);

/* odd-x-numbered columns, interpolating horizontally */
for x_g = U to N1_g - D
      for y_g = 1 to (N2_g – 1) with step 2
            for z_g = F to N3_g - B
                  v_g[x_g][y_g][z_g] = 0.5 * (v_g[x_g][y_g + 1][z_g] +
                                              v_g[x_g][y_g - 1][z_g]);
```

Συνάρτηση περιορισμού

Ο παρακάτω αλγόριθμος είναι για τη συνάρτηση περιορισμού *restrict_mpi* στις δύο διαστάσεις:

```
/* half-weighting residual restriction from fine-grid to coarse-grid */

/* interior points */
x_g = 2 * (U - 1);
for (x_2g = U to N1_2g – D) {
      x_g += 2;
      y_g = 2 * (L - 1);
      for (y_2g = L to N2_2g - R) {
            y_g += 2;
            r_2g[x_2g][y_2g] = 0.5 * r_g[x_g][y_g] + 0.125 *
                               (r_g[x_g + 1][y_g] + r_g[x_g - 1][y_g] +
```

```
                                    r_g[x_g][y_g + 1] + r_g[x_g][y_g - 1]);
        }
    }
```

Ο αλγόριθμος στις τρεις διαστάσεις έχει ως εξής:

```
    /* interior points */
    x_g = 2 * (U - 1);
    for (x_2g = U to N1_2g – D) {
        x_g += 2;
        y_g = 2 * (L - 1);
        for (y_2g = L to N2_2g - R) {
            y_g += 2;
            z_g = 2 * (F – 1);
            for (z_2g = F to N3_2g - B) {
                z_g += 2;
                r_2g[x_2g][y_2g][z_2g] = 0.25 * r_g[x_g][y_g][z_g] + 0.125 *
                                (r_g[x_g + 1][y_g][z_g] + r_g[x_g - 1][y_g][z_g] +
                                r_g[x_g][y_g + 1][z_g] + r_g[x_g][y_g - 1][z_g] +
                                r_g[x_g][y_g][z_g + 1] + r_g[x_g][y_g][z_g - 1]);
            }
        }
    }
```

Προεκτάσεις

Για μεγάλα μεγέθη πίνακα επίλυσης είναι λογικό οι παράλληλες εκδόσεις να υπερέχουν σημαντικά έναντι των σειριακών. Για μικρότερα μεγέθη όμως, τόσο για τη μέθοδο Jacobi όσο και για τη μέθοδο FMG, ίσως συμφέρει να γίνει σειριακή εκτέλεση. Επίσης, αντίθετα με τη σειριακή μέθοδο, στην παράλληλα υλοποιημένη πολυπλεγματική μέθοδο έχουμε περιορισμό στον αριθμό των πλεγμάτων. Θα δούμε στο πειραματικό μέρος πώς αντικατοπτρίζονται τα παραπάνω θέματα στις μετρήσεις ώστε να αποφανθούμε για την κατάλληλη αντιμετώπισή τους.

Κεφάλαιο 8

Παραλληλοποίηση του Αλγορίθμου FMG με Χρήση του OpenMP

Περιγραφή

Σε αυτή την ενότητα χρησιμοποιούμε το πρότυπο OpenMP για να εκμεταλλευτούμε τις δυνατότητες *παραλληλοποίησης* σε μοιραζόμενη μνήμη που έχει ο αλγόριθμος FMG και να επιτύχουμε και με το πρότυπο αυτό καλύτερη επίδοση. Παραπέμπουμε τον αναγνώστη στο [11] για αναλυτική πληροφόρηση σχετικά με το προγραμματιστικό μοντέλο κοινού χώρου διευθύνσεων OpenMP.

Υλοποίηση

Κορμός του Αλγορίθμου

Ο αλγόριθμος εδώ έχει ως εξής:

```
/* Full MultiGrid algorithm */
v^H = solve_omp(f^H, FMG_ITER); // computing exact solution

/* going right to groups of higher v-cycles */
for (g = H; g > h; g /= 2) {
      fg = g / 2;

      /* going up right before every first v-cycle */
      interpolate_omp(v^g → v^fg); // solution prolongation

      /* amount of v-cycles according to given ncycles */
      for (c = 0; c < ncycles; c++) {

            /* going down in v-cycle */
            for (gg = fg; gg < H; gg *= 2) {
                  v^gg = solve_omp(f^gg, a_1); // solution relaxation
                  r^gg = calc_res_omp(v^gg, f^gg); // residual calculation
                  restrict_omp(r^gg → r^2gg); // residual restriction
                  initmtrx_omp(e^2gg); // initialization with 0.0
            }

            e^H = solve_mpi(r^H, FMG_ITER); // computing exact error
```

```
                /* going up in v-cycle */
                for (gg = H / 2; gg > fg; gg /= 2) {
                      interpolate_omp(e^2gg → e^gg); // error prolongation
                      correct_omp(v^gg, e^gg); // solution correction
                      v^gg = solve_omp(f^gg, a_2); // solution relaxation
                }
          }
    }
```

Συναρτήσεις του Αλγορίθμου

Όλες οι συναρτήσεις που φαίνονται παραπάνω με την κατάληξη *_omp* είναι ίδιες με τις αντίστοιχες σειριακές με μια μόνο διαφορά. Έχουμε ορίσει ως *παράλληλες περιοχές* τα εξωτερικά for των υπολογισμών τους. Σε εκείνα τα σημεία, λοιπόν, δημιουργείται ένα *πλήθος νημάτων* (όσα έχουμε ορίσει) τα οποία έχουν πρόσβαση σε *κοινό χώρο διευθύνσεων*. Αυτά θα συγχρονιστούν πάλι στο *τέλος* της παράλληλης περιοχής (ολοκλήρωση του for). Ένα σημαντικό ερώτημα είναι εάν θα έπρεπε σε όλες τις συναρτήσεις μας να γίνει αυτή η διαδικασία. Ενδιαφέρει αυτό το ερώτημα διότι θα μπορούσε κάλλιστα κάποια συνάρτηση από αυτές να έχει καλύτερη επίδοση εκτελούμενη σειριακά. Μετά από πειραματικές μετρήσεις αποδεικνύεται ότι η *επιτάχυνση* του παράλληλου αλγορίθμου για μεγάλα μεγέθη πίνακα μεγιστοποιείται όταν παραλληλοποιηθούν όλες με αυτό τον τρόπο.

Προεκτάσεις

Όσο για το πόσα πλέγματα μπορούμε να επιλέξουμε, εδώ δεν έχουμε τους περιορισμούς που επέβαλλε η υλοποίηση του FMG με το MPI. Μπορούμε κάλλιστα να πάμε μέχρι το αδρότερο δυνατό πλέγμα. Αν θέσουμε *ένα* μόνο πλέγμα, θα έχουμε στην ουσία τη μέθοδο Jacobi αλλά υλοποιημένη με το OpenMP.

Είναι έκδηλη η προγραμματιστική ευκολία που προσφέρει το πρότυπο OpenMP και δε μένουν πολλά για να εξηγήσουμε σε αυτή την υλοποίηση. Συνεχίζουμε, λοιπόν, με το πειραματικό μέρος, όπου θα μελετήσουμε την επίδοση των αλγορίθμων που περιγράψαμε. Συγκρίνοντας τα αποτελέσματα, θα εξάγουμε πολύτιμα συμπεράσματα.

Κεφάλαιο 9

Πειραματικό μέρος

Σειριακός Αλγόριθμος Jacobi

Μετρήσεις και Διαγράμματα Χρόνου Σύγκλισης

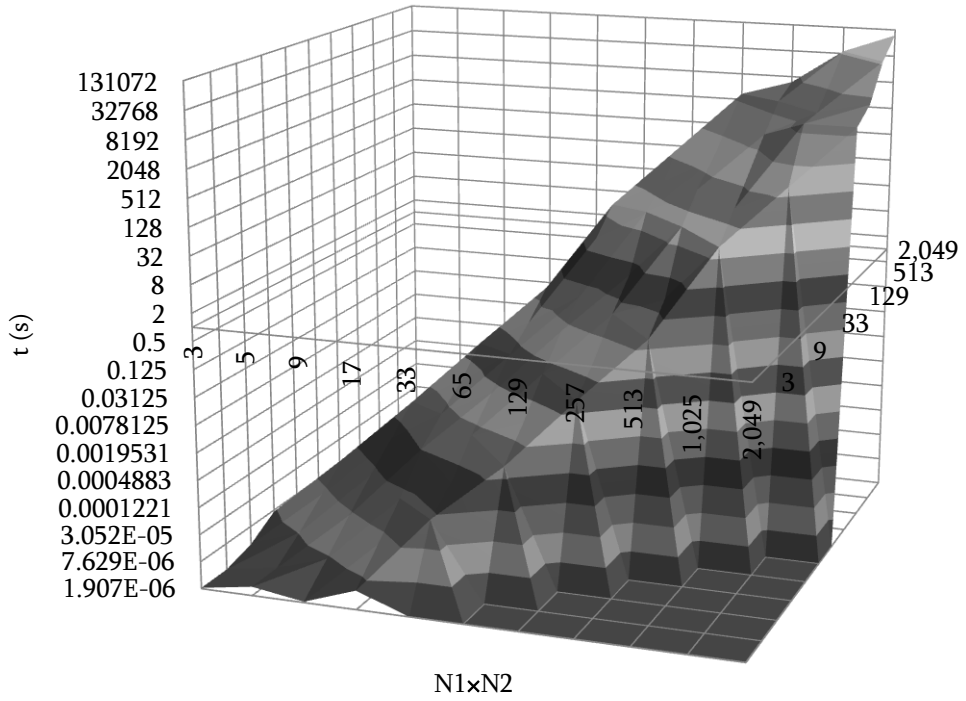
Για την επίλυση του προβλήματος με τη μέθοδο Jacobi, ας δούμε τον πίνακα χρόνου εύρεσης της τελικής κατανομής θερμότητας σε σχέση με το μέγεθος της πλάκας. Θυμίζουμε ότι οι πλευρές a και b πρέπει να πληρούν τη συνθήκη $1.25 \le a/b \le 8$.

Πίνακας 2:

Μετρήσεις χρόνου σύγκλισης του 2Δ αλγορίθμου Jacobi

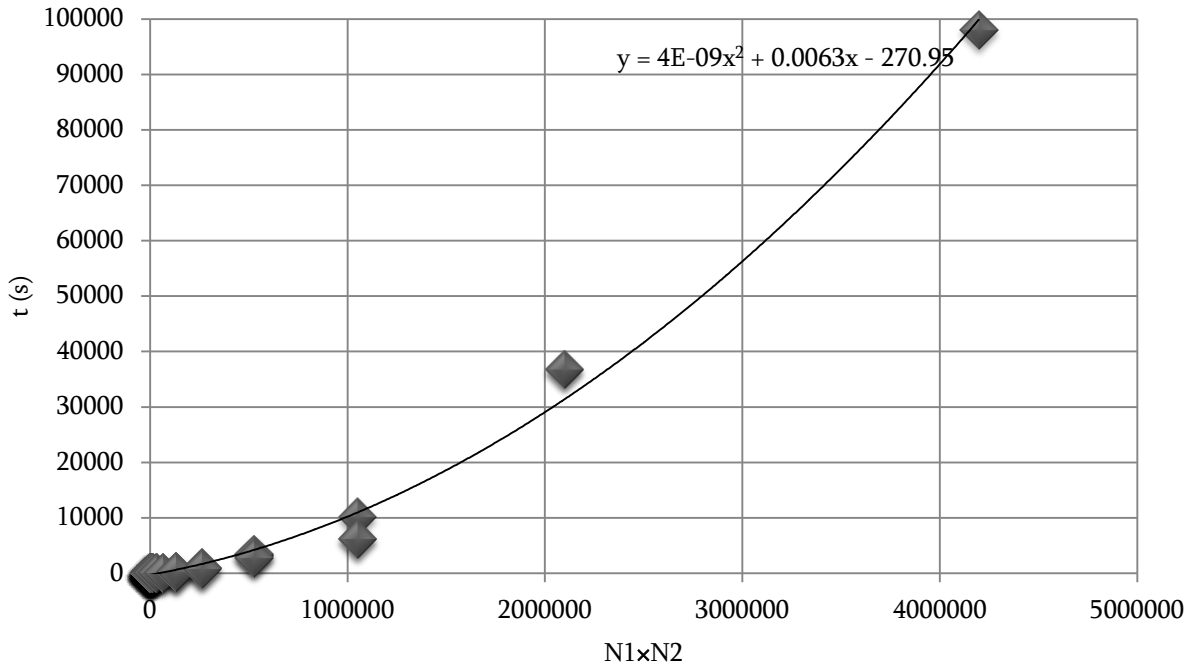
Jacobi	3	5	9	17	33	65	129	257	513	1025	2049
3	2.00E-06	3.00E-06	2.00E-06	5.00E-06							
5	2.00E-06	3.00E-06	7.00E-06	2.10E-05	1.01E-04						
9	3.00E-06	8.00E-06	2.70E-05	8.90E-05	2.64E-04	8.47E-04					
17	7.00E-06	2.50E-05	9.70E-05	4.89E-04	1.39E-03	3.60E-03	1.41E-02				
33		7.50E-05	2.95E-04	1.44E-03	5.72E-03	2.07E-02	5.81E-02	1.99E-01			
65			9.63E-04	3.76E-03	2.12E-02	9.24E-02	2.94E-01	8.08E-01	1.41E+01		
129				1.46E-02	5.76E-02	2.86E-01	1.19E+00	1.81E+01	5.70E+01	2.05E+02	
257					2.01E-01	3.54E+00	1.81E+01	7.27E+01	2.29E+02	3.05E+03	3.05E+03
513						1.26E+01	5.14E+01	2.07E+02	9.32E+02	3.36E+03	6.11E+03
1025							1.85E+02	7.55E+02	3.36E+03	1.01E+04	3.67E+04
2049								2.77E+03	6.11E+03	3.67E+04	9.80E+04

Για καλύτερα συμπεράσματα από τον παραπάνω πίνακα, παραθέτουμε και το αντίστοιχο χρονικό διάγραμμα σε σχέση με την επιφάνεια της πλάκας (οι άξονες είναι σε λογαριθμική κλίμακα):



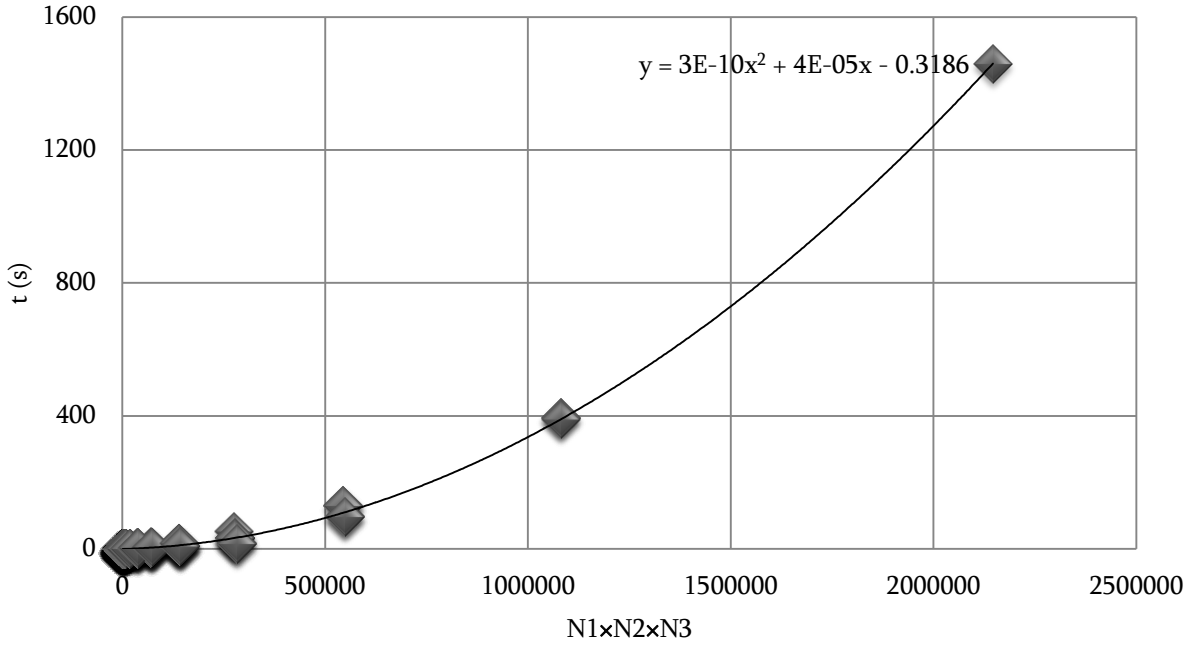
Σχήμα 15: Χρόνος σύγκλισης της 2Δ Jacobi για διάφορα μεγέθη του πίνακα

Βλέπουμε ότι η μέθοδος Jacobi συγκλίνει αρκετά αργά και για μεγάλα μεγέθη γίνεται απαγορευτική. Όπως φαίνεται και στο παρακάτω σχήμα, ο χρόνος σύγκλισής της προσεγγίζεται πολύ καλά από πολυώνυμο δευτέρου βαθμού του μεγέθους N1×N2 του πίνακα προς εύρεση:



Σχήμα 16: Χρόνος σύγκλισης της 2Δ Jacobi με αύξηση του μεγέθους του πίνακα

Παρόμοια αποτελέσματα προκύπτουν και για τη σύγκλιση της μεθόδου Jacobi στο τρισδιάστατο πρόβλημα. Όπως και για 2Δ, έτσι και εδώ, προκύπτει *πολυωνυμική* αύξηση δευτέρου βαθμού του χρόνου σύγκλισης σε σχέση με το μέγεθος N1×N2×N3 του 3Δ πίνακα.



Σχήμα 17: Χρόνος σύγκλισης της 3Δ Jacobi με αύξηση του μεγέθους του πίνακα

Δισδιάστατη και Τρισδιάστατη Υλοποίηση

Ωστόσο, συγκρίνοντας τα διαγράμματα για 2Δ και 3Δ, μπορούμε να συμπεράνουμε ότι για τον ίδιο αριθμό στοιχείων μεταξύ 2Δ και 3Δ πίνακα, στο 3Δ πρόβλημα έχουμε σημαντικά ταχύτερη (πάνω από μια τάξη μεγέθους) σύγκλιση. Αυτό το δείχνουν και οι εξισώσεις των καμπυλών παρεμβολής που φαίνονται στα δύο σχήματα. Ο συντελεστής δευτεροβάθμιου όρου είναι μιας τάξης μικρότερος και ο πρωτοβάθμιος συντελεστής δύο τάξεις μεγέθους μικρότερος. Παρόλο που το 3Δ πρόβλημα είναι επέκταση του 2Δ, τα δύο προβλήματα είναι διαφορετικά. Πέρα από την απαίτηση και προσδοκία για πολυωνυμική αύξηση του χρόνου σύγκλισης (λόγω της χρήσης της μεθόδου Jacobi και από τα δύο) και στα δύο προβλήματα, καθώς αυξάνεται το μέγεθος του πίνακα, δεν έχει νόημα να τα συγκρίνουμε περαιτέρω με τους ίδιους όρους. Ωστόσο, θα διαπιστώσουμε στη συνέχεια ότι οι δύο υλοποιήσεις έχουν πολλά περισσότερα κοινά, δικαιολογώντας το χαρακτηρισμό της τρισδιάστατης υλοποίησης ως (σχεδόν) *ευθεία επέκταση* της αντίστοιχης δισδιάστατης και για τις δύο μεθόδους.

Σειριακός Αλγόριθμος FMG

Μετρήσεις και Διαγράμματα Χρόνου Σύγκλισης

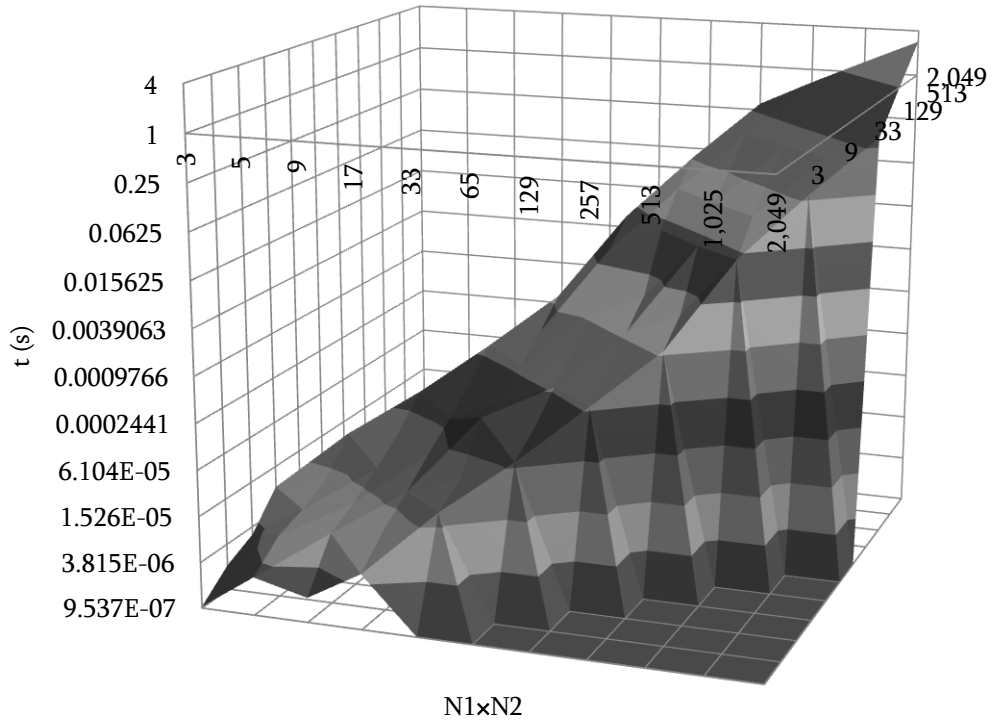
Σε αυτό το σημείο θα χρησιμοποιήσουμε τη μέθοδο FMG για την επίλυση του προβλήματος. Παρακάτω έχουμε τον πίνακα χρόνου εύρεσης της τελικής κατανομής θερμότητας σε σχέση με το μέγεθος της πλάκας όταν είναι ncycles=2.

Πίνακας 3:

Μετρήσεις χρόνου σύγκλισης του 2Δ αλγορίθμου FMG

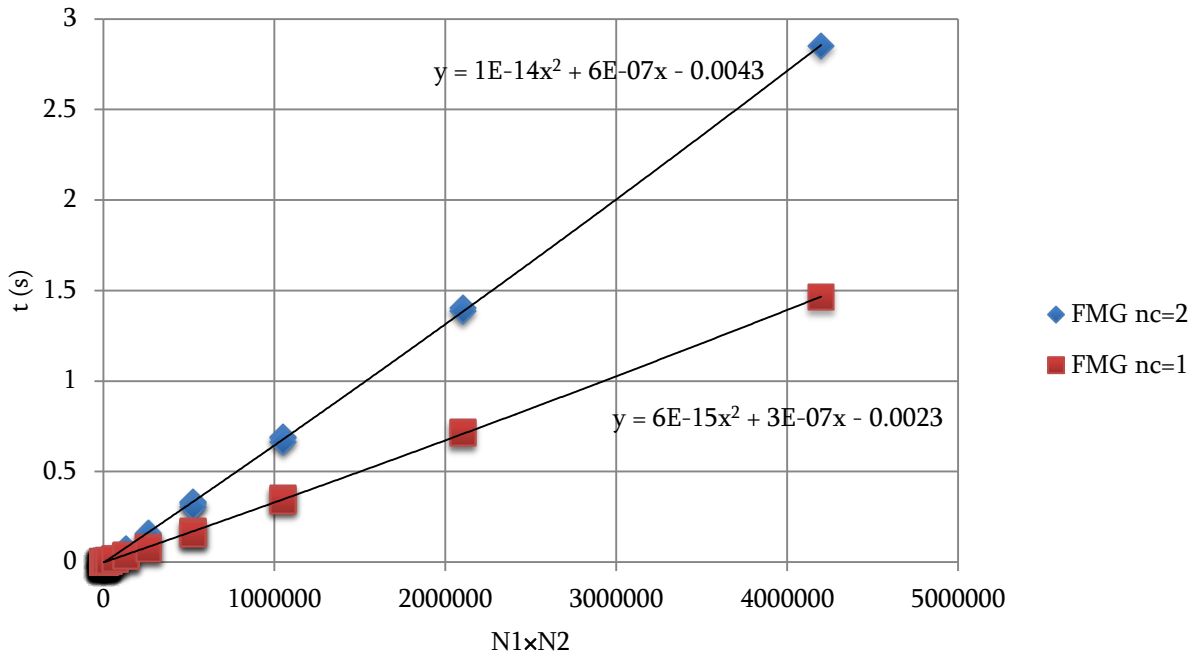
FMG nc2	3	5	9	17	33	65	129	257	513	1025	2049
3	1.00E-06	3.00E-06	2.00E-06	5.00E-06							
5	2.00E-06	6.00E-06	8.00E-06	1.10E-05	2.20E-05						
9	3.00E-06	8.00E-06	1.20E-05	2.00E-05	3.40E-05	7.10E-05					
17	8.00E-06	1.20E-05	2.10E-05	3.90E-05	1.41E-04	1.26E-04	2.75E-04				
33		3.20E-05	3.90E-05	7.20E-05	1.42E-04	2.67E-04	5.37E-04	1.09E-03			
65			9.00E-05	1.38E-04	2.80E-04	5.44E-04	1.11E-03	2.22E-03	1.31E-02		
129				3.06E-04	5.48E-04	1.11E-03	2.25E-03	1.21E-02	3.43E-02	7.30E-02	
257					1.16E-03	5.91E-03	1.22E-02	3.25E-02	7.07E-02	1.61E-01	3.33E-01
513						1.21E-02	3.23E-02	6.67E-02	1.56E-01	3.30E-01	6.93E-01
1025							6.63E-02	1.48E-01	3.20E-01	6.85E-01	1.40E+00
2049								3.04E-01	6.66E-01	1.39E+00	2.85E+00

Για καλύτερα συμπεράσματα από τον παραπάνω πίνακα, παραθέτουμε και το αντίστοιχο χρονικό διάγραμμα σε σχέση με την επιφάνεια της πλάκας (οι άξονες είναι σε λογαριθμική κλίμακα).



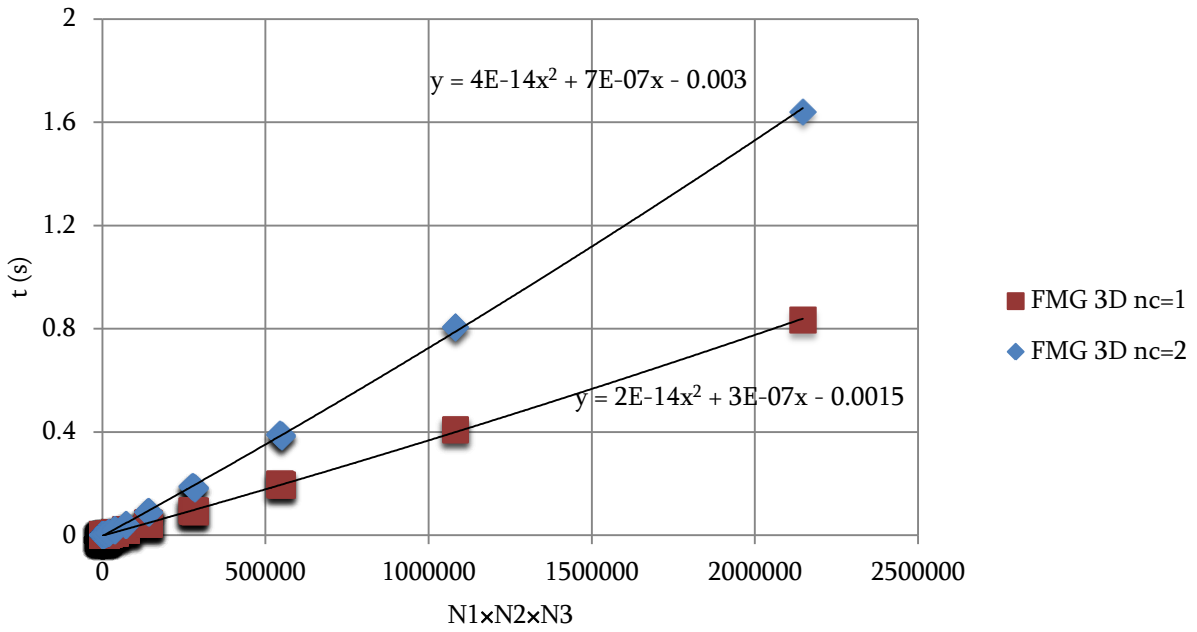
Σχήμα 18: Χρόνος σύγκλισης του 2Δ FMG για διάφορα μεγέθη του πίνακα (ncycles=2)

Αντίθετα με τη 2Δ μέθοδο Jacobi, ο χρόνος σύγκλισης της 2Δ μεθόδου FMG δείχνει να αυξάνεται (σχεδόν) γραμμικά με το μέγεθος N1×N2 του πίνακα προς εύρεση:



Σχήμα 19: Χρόνος σύγκλισης του 2Δ FMG καθώς αυξάνεται το μέγεθος του πίνακα (ncycles=1 & ncycles=2)

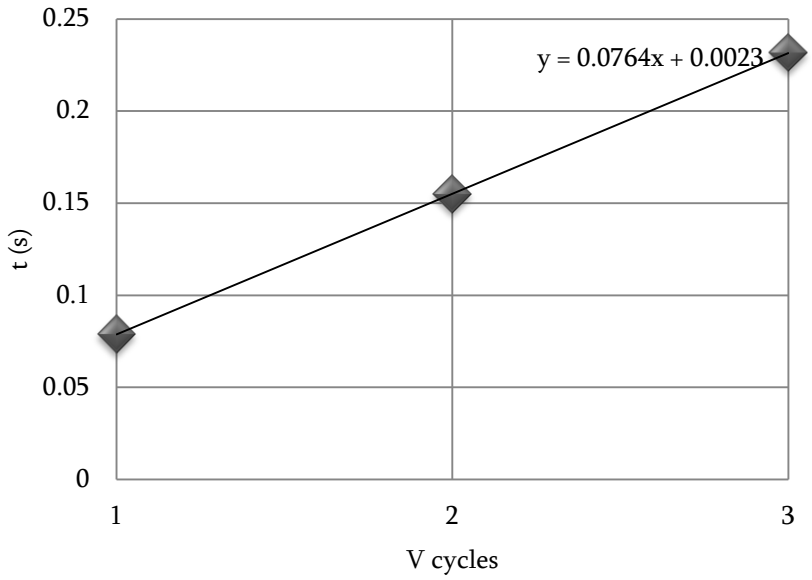
Αξιοσημείωτο είναι το γεγονός ότι και στην 3Δ περίπτωση η αύξηση του χρόνου σύγκλισης με την αύξηση του μεγέθους προάγεται σε (σχεδόν) *γραμμική* με χρήση του 3Δ FMG αλγορίθμου και μάλιστα με σχεδόν ίδιες εξισώσεις με τις αντίστοιχες 2Δ. Αυτό είναι πολύ σημαντικό, ειδικά αν αναλογιστούμε τη σημαντική διαφορά που είχαν η 2Δ Jacobi με την αντίστοιχη 3Δ.



Σχήμα 20: Χρόνος σύγκλισης του 3Δ FMG καθώς αυξάνεται το μέγεθος του πίνακα (ncycles=1 & ncycles=2)

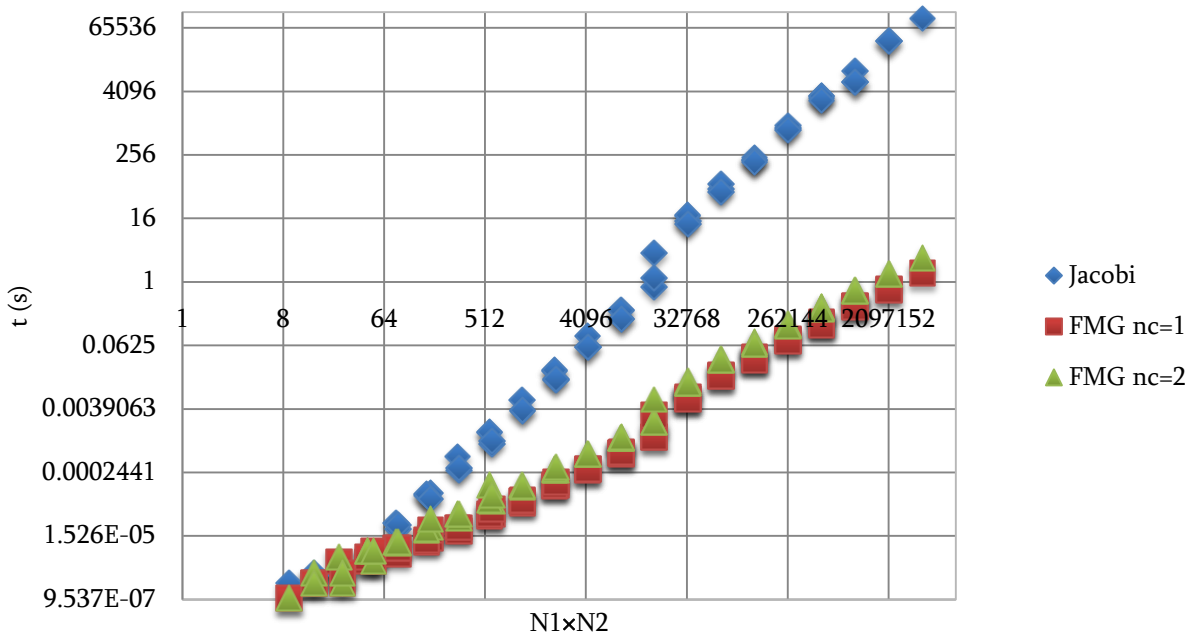
Φυσικά, δε μπορούμε να αγνοήσουμε τον δευτεροβάθμιο όρο των παραπάνω καμπυλών. Όμως, αξίζει να παρατηρήσουμε τη θεαματική μείωσή του σε σχέση με τις αντίστοιχες καμπύλες της μεθόδου Jacobi.

Όταν αυξάνεται το ncycles, αυξάνεται και ο χρόνος σύγκλισης αλλά λιγότερο από 100%. Αυτό φαίνεται και στο επόμενο διάγραμμα, όπου για μέγεθος 513×513 έχουμε τρέξει τον 2Δ αλγόριθμο FMG για τρία διαφορετικά ncycles.



Σχήμα 21: Χρόνος σύγκλισης του 2Δ FMG για διάφορα μεγέθη του ncycles (513x513, ncycles=2)

Πριν συνεχίσουμε στις παράλληλες εκδόσεις των προγραμμάτων μας, ας μελετήσουμε κάποια διαγράμματα επιπλέον για να εξετάσουμε από κοινού τις δύο παραπάνω μεθόδους (Jacobi και FMG). Στο παρακάτω διάγραμμα φαίνεται ο χρόνος σύγκλισης του 2Δ FMG αλγορίθμου για ncycles 1 και 2 και εκείνος του 2Δ αλγορίθμου Jacobi καθώς αλλάζει το μέγεθος του πίνακα προς εύρεση.



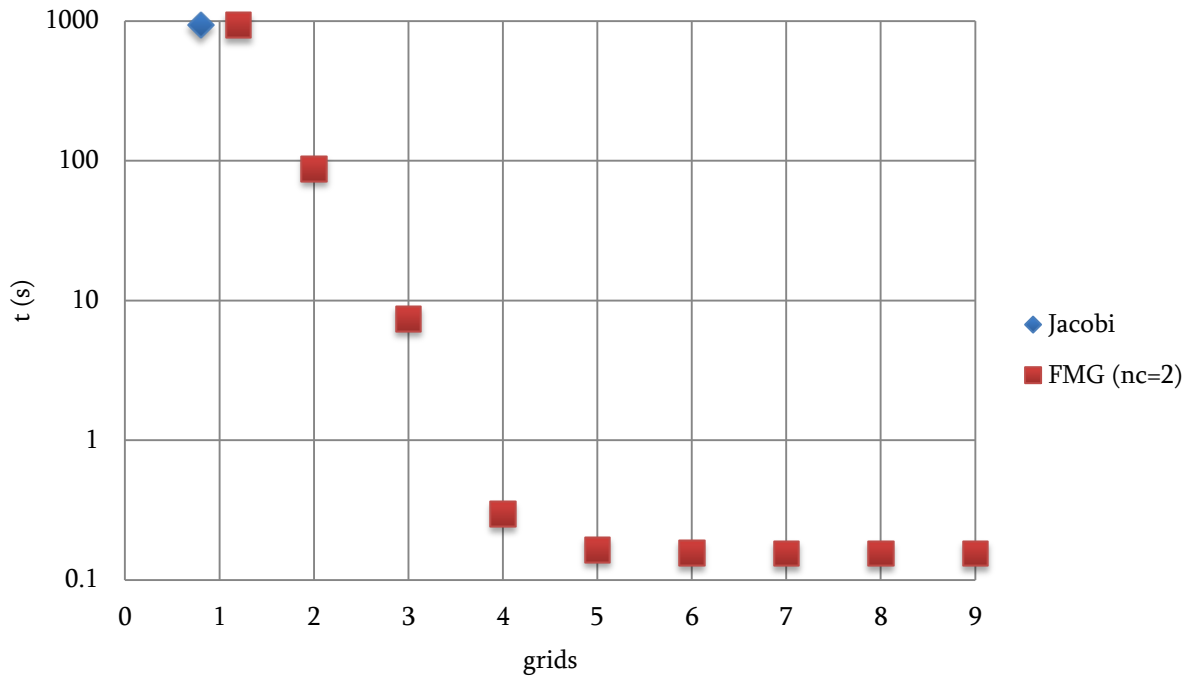
Σχήμα 22: Χρόνος σύγκλισης του 2Δ FMG (ncycles=1 & 2) και 2Δ Jacobi για διάφορα μεγέθη του πίνακα

Αφού παρατηρήσουμε ότι ο άξονας χρόνου είναι σε λογαριθμική κλίμακα, συμπεραίνουμε ότι υπάρχει μεγάλη διαφορά των δύο αλγορίθμων στην αποδοτικότητα. Σε όλα τα μεγέθη και ειδικότερα στα μεγαλύτερα, ο FMG υπερέχει συντριπτικά.

Πλήθος Πλεγμάτων και Σύγκλιση

Ένα βασικό σημείο για εξέταση είναι το πόσο «βαθιά» έχει νόημα να μπούμε με τον FMG. Γεγονός είναι πως αν θέσουμε ένα μόνο πλέγμα, δηλαδή το αδρότερο πλέγμα να είναι ο πίνακας προς εύρεση (άρα ταυτόχρονα και το λεπτομερέστερο), θα έχουμε στην ουσία χειριστεί την πολυπλεγματική μέθοδο ως μονοπλεγματική (τον FMG ως Jacobi). Άραγε όσο λιγότερα είναι τα πλέγματα, τόσο περισσότερο πλησιάζουμε τον αλγόριθμο Jacobi και τόσο πιο αργή είναι η σύγκλιση; Ή μήπως υπάρχει ένα πλέγμα ανάμεσα στο αδρότερο και το λεπτομερέστερο και αν φτάσουμε μέχρι αυτό μόνο, έχουμε τότε τη βέλτιστη ταχύτητα σύγκλισης;

Σε αυτά τα ερωτήματα θα μας βοηθήσει το παρακάτω διάγραμμα (ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα) για μέγεθος 513×513, ncycles ίσο με 2 και ngrids από 1 μέχρι 9:



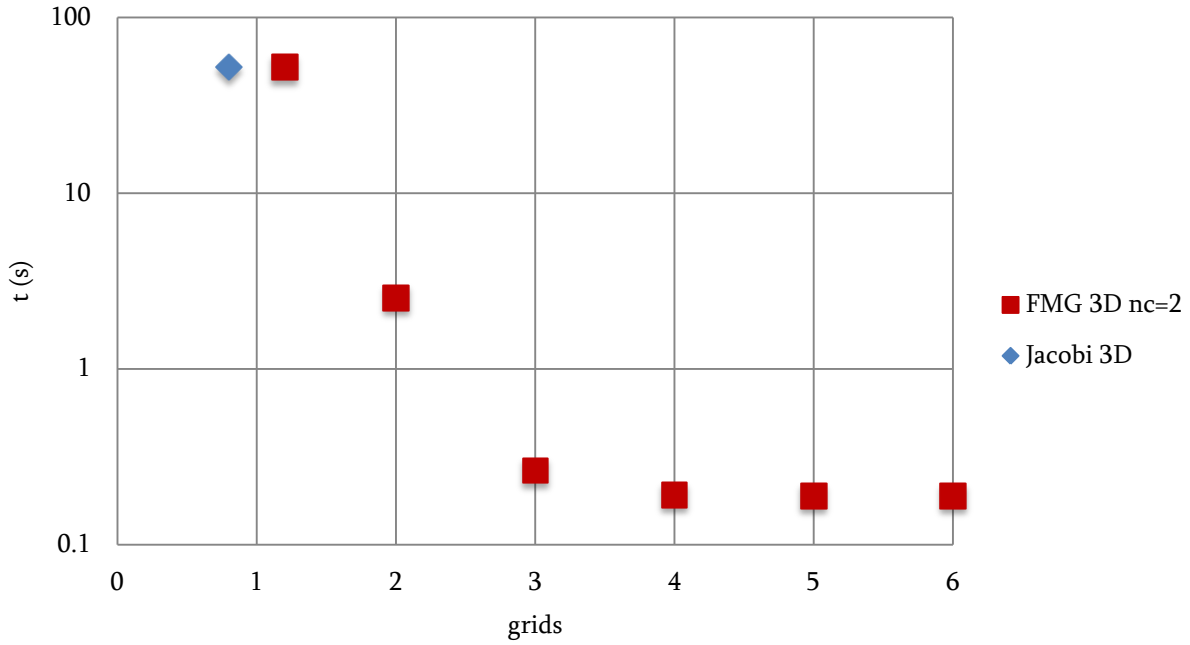
Σχήμα 23: Χρόνος σύγκλισης του 2Δ FMG για διάφορα μεγέθη του ngrids (513x513, ncycles=2)

Μπορούμε εύκολα να παρατηρήσουμε αυτό που ισχυριστήκαμε προηγουμένως, δηλαδή ότι υπάρχει ταύτιση των δύο μεθόδων για ngrids ίσο με 1. Η διαφορά τους όσο αυξάνεται ο αριθμός πλεγμάτων έγκειται στο πού (σε ποιο πλέγμα δηλαδή) εκτελείται η Jacobi αδρότερου πλέγματος. Αν εκτελείται στο ακριβώς λεπτομερέστερο πλέγμα από το αδρότερο δυνατό (ο πίνακας εκεί είναι ο 3×3), δηλαδή για πίνακα 5×5, δεν υπάρχει διαφορά γιατί σε αυτό το μέγεθος οι δύο μέθοδοι δεν έχουν χρονική διαφορά. Αν όμως εκτελείται έξι πλέγματα πιο πάνω, δηλαδή όταν ο πίνακας είναι ο 129×129, εκεί υπάρχει διαφορά γιατί για μέγεθος 129×129 η FMG (0.0022 sec) είναι σχεδόν 500 φορές ταχύτερη της Jacobi (1.1921 sec).

Γενικά, ο ταχύτερος αλγόριθμος για όλα τα μεγέθη θα υφίσταται για τη μεγαλύτερη δυνατή τιμή του ngrids. Από εκεί και πέρα μπορεί να θέλουμε να θυσιάσουμε λίγη ταχύτητα σύγκλισης προκειμένου να έχουμε λιγότερα πλέγματα. Αυτή η διαφορά, η θυσία, θα είναι ανεπαίσθητη για ένα συγκεκριμένο αριθμό πλεγμάτων και όταν γίνουν λιγότερα θα επηρεάσει. Είναι σχετικό, όμως, το πόση χρονική αύξηση θεωρούμε «ανεπαίσθητη», ανάλογα με την εφαρμογή μας και το λόγο που θέλουμε λιγότερα πλέγματα. Πάντως, αν μπορούμε να δεχτούμε τις φαινομενικά πολύ μικρές διαφορές (μέχρι 0.010 sec) για αριθμό πλεγμάτων από 5 μέχρι 9, μπορούμε να ισχυριστούμε ότι τουλάχιστον για μεγέθη από 33×33 και πάνω, ένας αριθμός πλεγμάτων (ngrids) μέχρι πέντε μικρότερος (δηλαδή για αδρότερο το πλέγμα με πίνακα 33×33 = $(2^5 + 1)\chi(2^5 + 1)$, για το οποίο η Jacobi εκτελείται σε 0.005715 sec ενώ η FMG σε 0.000071 sec) από τον μέγιστο δυνατό είναι ανεκτός. Βέβαια,

όσο μεγαλώνει το μέγεθος του πίνακα, άρα και ο χρόνος σύγκλισης, μεγαλύτερες διαφορές πιθανόν να γίνονται ανεκτές με αποτέλεσμα να μπορούμε να δεχτούμε επιπλέον ελάττωση του ngrids.

Τα παραπάνω ερωτήματα προφανώς αφορούν και την 3Δ περίπτωση και εδώ θα μας βοηθήσει το παρακάτω διάγραμμα (ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα) για μέγεθος 65×65×65, ncycles ίσο με 2 και ngrids από 1 μέχρι 6:



Σχήμα 24: Χρόνος σύγκλισης του 2Δ FMG για διάφορα μεγέθη του ngrids (513x513, ncycles=2)

Είναι πολλές οι ομοιότητες του παραπάνω σχήματος με το αντίστοιχο 2Δ. Καταρχάς, υπάρχει ταύτιση των 3Δ μεθόδων Jacobi και FMG για ngrids ίσο με 1. Εδώ, αν μπορούμε να δεχτούμε τις μικρές διαφορές (μέχρι 0.022 sec) για αριθμό πλεγμάτων από 4 μέχρι 6, μπορούμε να ισχυριστούμε ότι τουλάχιστον για μεγέθη από 9×9×9 και πάνω, ένας αριθμός πλεγμάτων (ngrids) μέχρι τρία μικρότερος (δηλαδή για αδρότερο το πλέγμα με πίνακα $9\times9\times9 = (2^3 + 1)\chi(2^3 + 1)\chi(2^3 + 1)$, για το οποίο η Jacobi εκτελείται σε 0.001022 sec ενώ η FMG σε 0.000152 sec) από τον μέγιστο δυνατό είναι ανεκτός.

Μπορεί κάποιος να ισχυριστεί (τόσο για 2Δ όσο και για 3Δ) ότι αν και η ταχύτητα σύγκλισης της Jacobi είναι πολύ μικρότερη της FMG, η σύγκλιση αυτή μπορεί να είναι καλύτερης ποιότητας αφού χρησιμοποιείται κριτήριο σύγκλισης. Αυτό έχει μια βάση ως επιχείρημα με την έννοια ότι ο αλγόριθμος FMG χρησιμοποιεί κριτήριο σύγκλισης μόνο στο αδρότερο πλέγμα. Στην ουσία, η σύγκλιση του FMG επιτυγχάνεται όταν ολοκληρωθούν τα στάδια του Σχήματος 4γ. Αν θέλουμε ακόμα καλύτερη σύγκλιση μπορούμε να αυξήσουμε

και το ncycles ή/και τις παραμέτρους πρότερης και ύστερης χαλάρωσης - a_1 και a_2. Αποδεικνύεται ότι είναι πολύ καλύτερη η σύγκλιση του FMG (ακόμα και για το χαμηλότερο ncycles) και ιδιαίτερα για μεγάλα μεγέθη πίνακα. Στη μονοπλεγματική μέθοδο, δηλαδή, όσο μεγαλώνει το μέγεθος του πίνακα, εκτός από το χρόνο σύγκλισης χειροτερεύει πολύ και η ποιότητα σύγκλισης (για ένα συγκεκριμένο κριτήριο σύγκλισης). Όπως είδαμε στα παραπάνω σχήματα, όσο λιγότερα επιλέγουμε να είναι τα πλέγματα, τόσο ο χρόνος σύγκλισης του FMG πλησιάζει το χρόνο σύγκλισης της Jacobi. Με τον ίδιο τρόπο, ο FMG συμπεριφέρεται και όσον αφορά την ποιότητα σύγκλισης. Για καλύτερη σύγκλιση, δηλαδή, μπορούμε να αυξήσουμε το ngrids αν δεν είναι το μεγαλύτερο δυνατό.

Προεκτάσεις

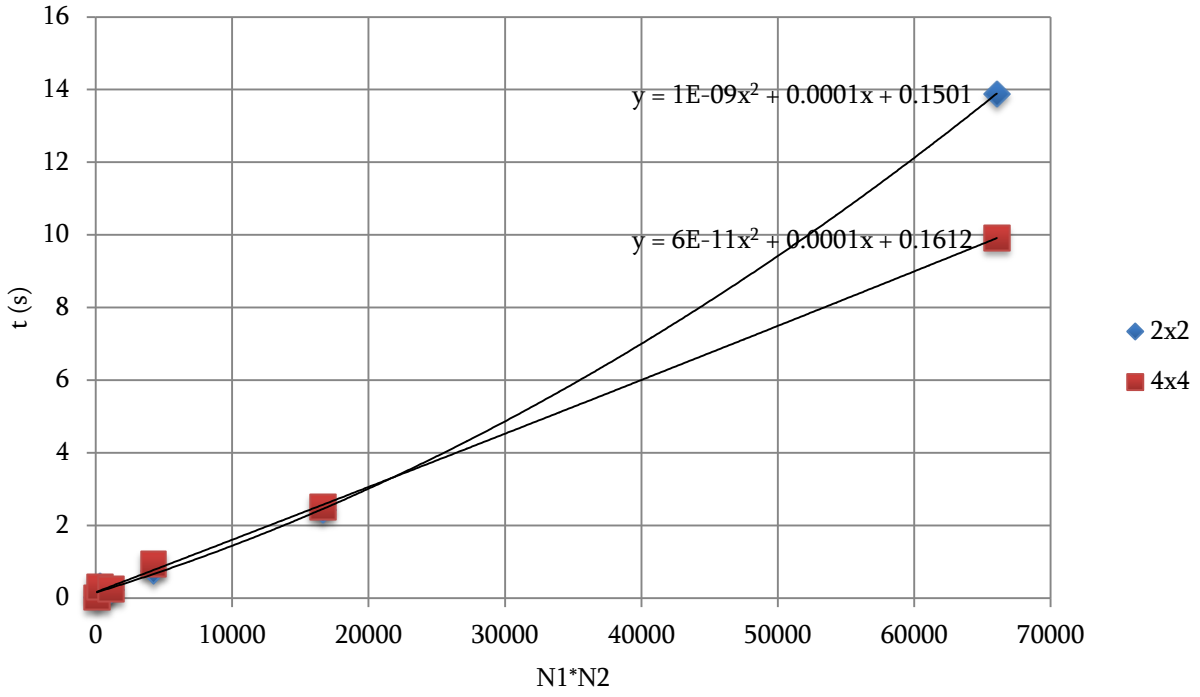
Γενικά, είναι καλύτερο να προτιμάμε το μέγιστο δυνατό ngrids, τόσο για την ταχύτητα σύγκλισης όσο και για την ποιότητά της, ακόμα και αν δεν έχει μεγάλο κέρδος σε σχέση με λίγο μικρότερες τιμές. Στην περίπτωση της παραλληλίας, όμως, αύξηση των διεργασιών μπορεί να σημαίνει ελάττωση του ngrids. Αν, λοιπόν, θέλουμε να αυξήσουμε τις διεργασίες, θα πρέπει να αναλογιστούμε πόση χρονική αύξηση και πόση υποβάθμιση της ποιότητας της σύγκλισης (λόγω του λεπτομερέστερου αδρότερου πλέγματος) μπορούμε να δεχτούμε.

Παράλληλος Αλγόριθμος Jacobi (MPI)

Διαγράμματα Χρόνου Σύγκλισης

Σε αυτό το σημείο θα χρησιμοποιήσουμε τη μέθοδο Jacobi υλοποιημένη με τη βοήθεια του MPI για την επίλυση του προβλήματος.

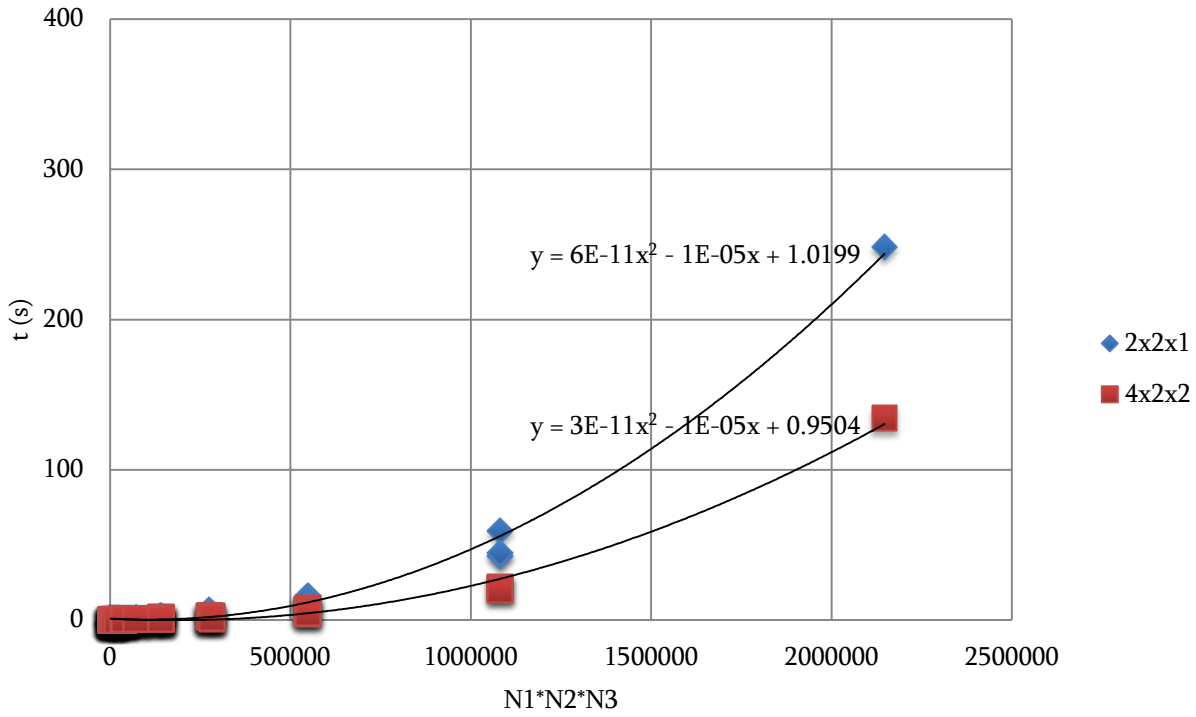
Παρακάτω φαίνεται πώς μεταβάλλεται ο χρόνος σύγκλισης του 2Δ αλγορίθμου, εκτελούμενου σε δεκαέξι επεξεργαστές, καθώς αυξάνεται το μέγεθος του πίνακα και για σχήματα MPI διεργασιών 2×2 και 4×4:



Σχήμα 25: Χρόνος σύγκλισης του 2Δ Jacobi αλγορίθμου υλοποιημένου με MPI και εκτελούμενου σε 16 επεξεργαστές καθώς αυξάνεται το μέγεθος του πίνακα (σχήματα MPI διεργασιών: 2x2 & 4x4)

Μπορούμε να παρατηρήσουμε ότι διατηρείται ο *πολυωνυμικός* (δευτέρου βαθμού) ρυθμός αύξησης του χρόνου σύγκλισης καθώς αυξάνεται το μέγεθος του πίνακα. Ωστόσο, ελαττώνεται σημαντικά η κλίση της κάθε καμπύλης και περισσότερο, όπως ήταν αναμενόμενο, για το σχήμα διεργασιών 4×4.

Όμοια συμπεριφορά παρατηρούμε και στην περίπτωση της αντίστοιχης 3Δ Jacobi υλοποιημένης με MPI και εκτελούμενης σε δεκαέξι επεξεργαστές. Εδώ τα σχήματα διεργασιών που μελετάμε είναι το 2×2×1 και το 4×2×2:



Σχήμα 26: Χρόνος σύγκλισης του 3Δ Jacobi αλγορίθμου υλοποιημένου με MPI και εκτελούμενου σε 16 επεξεργαστές καθώς αυξάνεται το μέγεθος του πίνακα (σχήματα MPI διεργασιών: 2x2x1 & 4x2x2)

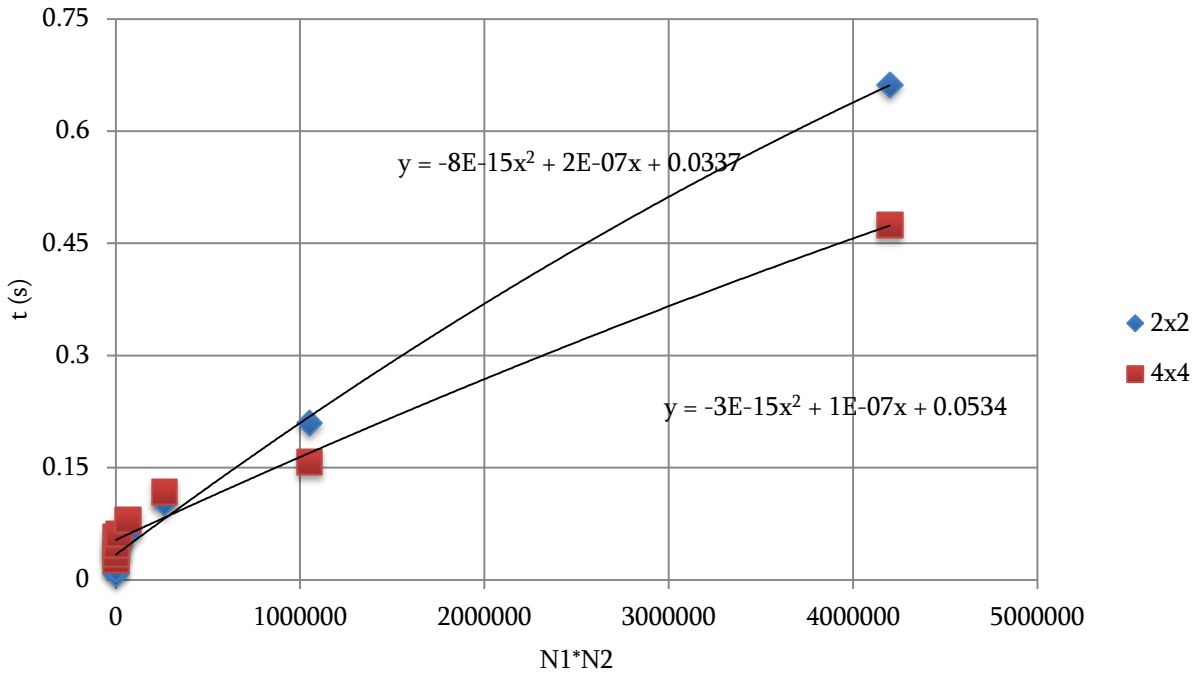
Όπως «γραμμικοποιήθηκε» η πολυωνυμική καμπύλη χρόνου σύγκλισης του σειριακού αλγορίθμου Jacobi με τη βοήθεια της μεθόδου FMG, αναμένουμε παρόμοια αποτελέσματα και με τους παραλληλοποιημένους αλγορίθμους.

Παράλληλος Αλγόριθμος FMG (MPI)

Διαγράμματα Χρόνου Σύγκλισης

Ας δούμε πώς μεταβάλλεται στο πρόβλημα ο χρόνος σύγκλισης με την αύξηση του μεγέθους του πίνακα προς επίλυση, όταν χρησιμοποιούμε τη μέθοδο FMG υλοποιημένη με τη βοήθεια του MPI.

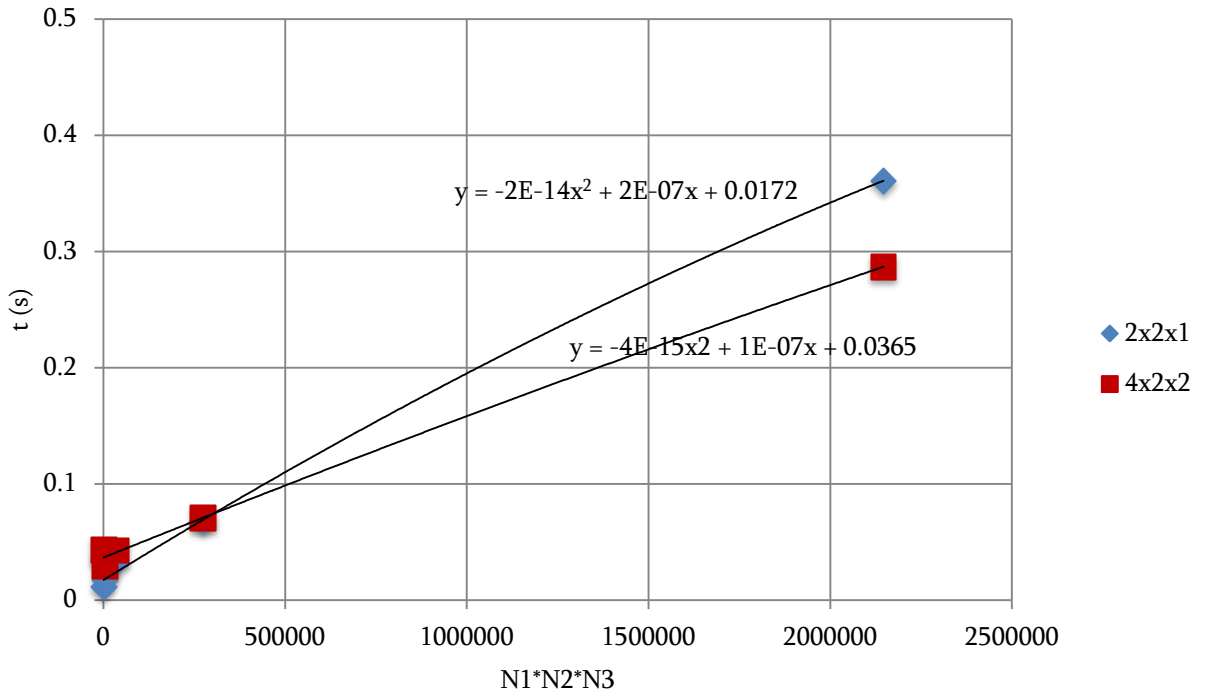
Παρακάτω έχουμε εκτέλεση για τη 2Δ περίπτωση σε δεκαέξι επεξεργαστές για σχήματα διεργασιών 2×2 και 4×4:



Σχήμα 27: Χρόνος σύγκλισης του 2Δ FMG αλγορίθμου υλοποιημένου με MPI και εκτελούμενου σε 16 επεξεργαστές καθώς αυξάνεται το μέγεθος του πίνακα (σχήματα MPI διεργασιών: 2x2 & 4x4)

Εκτός από τη θεαματική μείωση της κλίσης της καμπύλης σε σχέση με την αντίστοιχη της σειριακής έκδοσης, μπορούμε να παρατηρήσουμε ότι πετύχαμε κάτι πέρα από τη γραμμικότητα στην καμπύλη. Προέκυψε πολυωνυμική αύξηση δευτέρου βαθμού με μικρό μεν αλλά *αρνητικό* δευτεροβάθμιο όρο.

Όμοια συμπεριφορά παρατηρούμε και στην περίπτωση της αντίστοιχης 3Δ FMG μεθόδου υλοποιημένης με MPI και εκτελούμενης σε δεκαέξι επεξεργαστές. Εδώ τα σχήματα διεργασιών που μελετάμε είναι το 2×2×1 και το 4×2×2:



Σχήμα 28: Χρόνος σύγκλισης του 3Δ FMG αλγορίθμου υλοποιημένου με MPI και εκτελούμενου σε 16 επεξεργαστές καθώς αυξάνεται το μέγεθος του πίνακα (σχήματα MPI διεργασιών: 2x2 & 4x4)

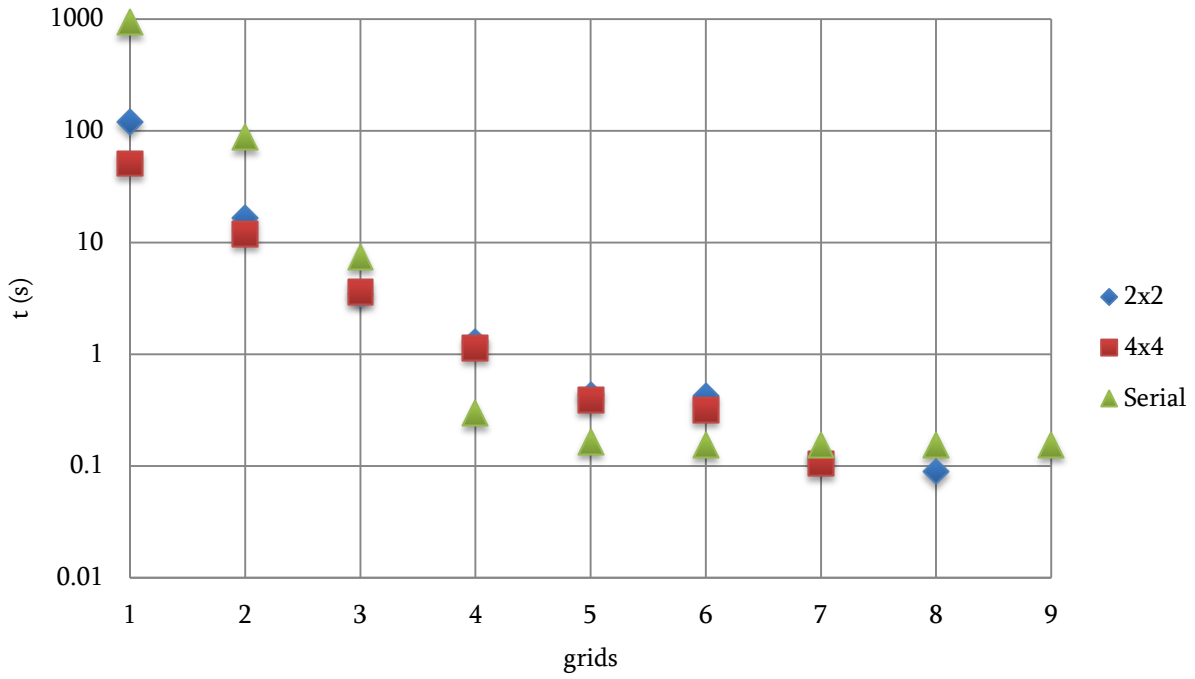
Όπως και στη 2Δ περίπτωση, παρατηρούμε μια θεαματική μείωση της κλίσης της καμπύλης σε σχέση με την αντίστοιχη της σειριακής έκδοσης. Επίσης, προέκυψε και εδώ πολυωνυμική αύξηση δευτέρου βαθμού με (μικρό μεν αλλά) *αρνητικό* δευτεροβάθμιο όρο.

Τόσο στη μέθοδο Jacobi όσο και στην FMG, φαίνεται ότι στα μικρά μεγέθη η επικοινωνία μεταξύ των διεργασιών κοστίζει περισσότερο από το κέρδος εισαγωγής περισσότερων διεργασιών στη διαδικασία επίλυσης. Για αυτό το λόγο, στα μικρά μεγέθη, και τόσο για 2Δ όσο και για 3Δ, η σειριακή έκδοση είναι ταχύτερη από τις παράλληλες και η παράλληλη με 4 διεργασίες είναι ταχύτερη εκείνης με 16. Λόγω εγγενών ιδιοτήτων του προβλήματος - όπως είναι για παράδειγμα η κατεύθυνση με την οποία διατρέχουν οι μέθοδοι τον πίνακα στο πλαίσιο της διαδικασίας επίλυσης, οι συνοριακές συνθήκες που έχουμε θέσει ή ο μέγιστος δυνατός αριθμός πλεγμάτων - είναι λογικό για διαφορετικά σχήματα πινάκων ή/και για διαφορετικά σχήματα MPI διεργασιών αλλά με τον ίδιο αριθμό διεργασιών η ταχύτητα σύγκλισης να είναι διαφορετική. Συνεπώς, στα παραπάνω διαγράμματα έχουμε λάβει υπ' όψιν μόνο μεγέθη πίνακα ίδιας αναλογίας (εδώ τετραγωνικά μεγέθη) για να αντιμετωπίσουμε τα προβλήματα επί ίσοις όροις.

Πλήθος Πλεγμάτων και Σύγκλιση

Έχουμε πει ότι για μοναδιαίο αριθμό πλεγμάτων η FMG και η Jacobi ταυτίζονται. Αυτό ισχύει και στις παράλληλες υλοποιήσεις. Είπαμε, ακόμα, ότι είναι καλύτερο να προτιμάμε το μέγιστο δυνατό ngrids, τόσο για την ταχύτητα σύγκλισης όσο και για την ποιότητα της σύγκλισης αυτής. Στην περίπτωση της παραλληλίας, όμως, αύξηση των διεργασιών μπορεί να σημαίνει ελάττωση του ngrids. Αυτό διότι ο περιορισμός μας δεν είναι πλέον το αδρότερο πλέγμα (εκείνο που έχει πίνακα με τουλάχιστον μια διάσταση ίση με 3) αλλά το αδρότερο ανά διεργασία πλέγμα (δηλαδή εκείνο που προβλέπει blocks με τουλάχιστον μια διάσταση ίση με 3). Στο δισδιάστατο πρόβλημα και για τετραγωνικά μεγέθη, για μια διεργασία το αδρότερο πλέγμα είναι το 3×3. Για σχήμα διεργασιών 2×2 είναι το 5×5 (αφού στο αδρότερο πλέγμα η καθε διεργασία θα έχει block 3×3), για σχήμα 4×4 διεργασιών είναι το 9×9 και ούτω καθεξής. Αν, λοιπόν, θέλουμε να αυξήσουμε τις διεργασίες, θα πρέπει να αναλογιστούμε πόση χρονική αύξηση και πόση υποβάθμιση της ποιότητας της σύγκλισης (λόγω του λεπτομερέστερου αδρότερου πλέγματος) έχουμε την πολυτέλεια να δεχτούμε.

Παρακάτω φαίνεται πώς μεταβάλλεται ο χρόνος σύγκλισης (ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα) για μέγεθος 513×513, ncycles ίσο με 2 και ngrids από 1 μέχρι 9 (8, για σχήμα διεργασιών 2×2 και 7 για 4×4). Υπενθυμίζουμε ότι πρόκειται για τον 2Δ αλγόριθμο FMG υλοποιημένο με MPI και εκτελούμενο σε δεκαέξι επεξεργαστές. Τα σχήματα διεργασιών είναι τα 2×2 και 4×4:



Σχήμα 29: Χρόνος σύγκλισης του 2Δ FMG (ncycles=2) για μέγεθος πίνακα 513x513, για διάφορες τιμές του ngrids. Ο αλγόριθμος είναι υλοποιημένος με MPI και εκτελείται σε 16 επεξεργαστές για δύο σχήματα MPI διεργασιών ενώ φαίνεται και η σειριακή υλοποίηση.

Στην περίπτωση των σειριακών εκδόσεων είχαμε δει για το ίδιο μέγεθος (513×513) ότι η διαφορά για μέχρι πέντε πλέγματα λιγότερα από το μέγιστο δυνατό ήταν πολύ μικρή. Όμως, αυτό συνέβαινε επειδή οι χρόνοι σύγκλισης Jacobi και FMG μέχρι και το πέμπτο αδρότερο πλέγμα (εκείνο με μέγεθος πίνακα 33×33) ήταν πολύ κοντά. Στις παράλληλες εκδόσεις, η επικοινωνία μεταξύ των διεργασιών προσφέρει καθοριστικής σημασίας αποδοτικότητα σε μεγάλα μεγέθη.

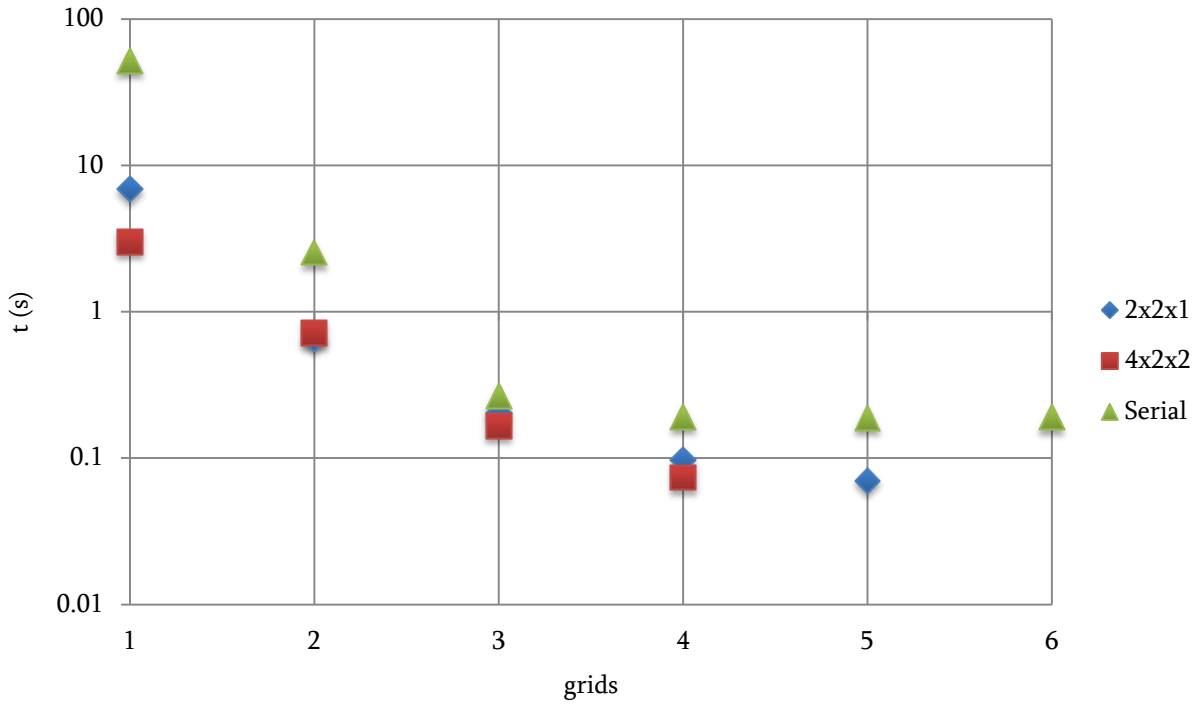
Σε μικρά μεγέθη (μικρότερα του 513×513), όμως, είναι τέτοιο το κόστος της (τουλάχιστον για Ethernet ως δίκτυο διασύνδεσης) που ακυρώνει το κέρδος της παραλληλοποίησης και ώστε η FMG (MPI) να είναι πιο αργή από τη σειριακή της έκδοση. Όσο το μέγεθος μεγαλώνει, το κόστος επικοινωνίας παρουσιάζει ήπια αύξηση ενώ ο χρόνος των υπολογισμών ελαττώνεται σημαντικά σε σχέση με τη σειριακή έκδοση. Αυτό συμβαίνει διότι το κόστος της επικοινωνίας είναι τάξης O(N) ενώ το κόστος των υπολογισμών είναι τάξης $O(N^2)$. Για τον ίδιο λόγο, για μεγάλα μεγέθη, αυξάνοντας τον αριθμό των διεργασιών (μέχρι ένα όριο), ο χρόνος επικοινωνίας παρουσιάζει μικρή διακύμανση ενώ ο χρόνος υπολογισμών μειώνεται ραγδαία. Άρα για το μέγεθος 513×513, ένα πλέγμα λιγότερο (δηλαδή για ngrids=8) ώστε να μπορούμε χρησιμοποιήσουμε την παράλληλη μέθοδο με σχήμα διεργασιών 2×2 έχει νόημα και έτσι προκύπτει καλύτερος χρόνος σύγκλισης σε σχέση με τη σειριακή μέθοδο. Ακόμα λιγότερα πλέγματα, όμως, προς

χάριν εκτέλεσης με περισσότερες διεργασίες δε θα έχουν καλύτερη επίδοση. Για μεγαλύτερα μεγέθη, όμως, θα αποδώσουν και όσο μεγαλώνει το μέγεθος τόσο λιγότερα πλέγματα σε σχέση με το μέγιστο θα έχει νόημα να χρησιμοποιούμε.

Στο βαθμό, λοιπόν, που για ένα μέγεθος μπορούμε να χρησιμοποιήσουμε έναν αριθμό πλεγμάτων λιγότερο από τον μέγιστο δυνατό χωρίς σημαντικό χρονικό κόστος, είναι πιθανότατα καλύτερο να το κάνουμε για να χρησιμοποιήσουμε περισσότερες διεργασίες.

Οι παραπάνω προβληματισμοί μας ισχύουν και στην 3Δ περίπτωση. Ωστόσο, υπάρχει μια σημαντική διαφορά στα χρονικά κόστη. Το κόστος της επικοινωνίας είναι τώρα τάξης $O(N^2)$ ενώ το κόστος των υπολογισμών είναι τάξης $O(N^3)$. Επίσης, για μεγέθη μικρότερα από 65×65×65, η 3Δ FMG (MPI) φαίνεται να είναι πιο αργή από τη σειριακή της έκδοση. Όσο το μέγεθος πίνακα μεγαλώνει, όμως, το κέρδος παραλληλοποίησης γίνεται όλο και πιο αισθητό, με αποτέλεσμα να υπάρχουν τα περιθώρια ώστε ολοένα και περισσότερες διεργασίες (ακόμα και αν αυτό σημαίνει λιγότερα πλέγματα) να έχει νόημα να χρησιμοποιηθούν.

Παρακάτω φαίνεται πώς μεταβάλλεται ο χρόνος σύγκλισης (ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα) για μέγεθος 65×65×65, ncycles ίσο με 2 και ngrids από 1 μέχρι 6 (5 για σχήμα διεργασιών 2×2×1 και 4 για 4×2×2). Πρόκειται για τον 3Δ αλγόριθμο FMG υλοποιημένο με MPI και εκτελούμενο σε δεκαέξι επεξεργαστές. Τα σχήματα διεργασιών είναι τα 2×2×1 και 4×2×2:

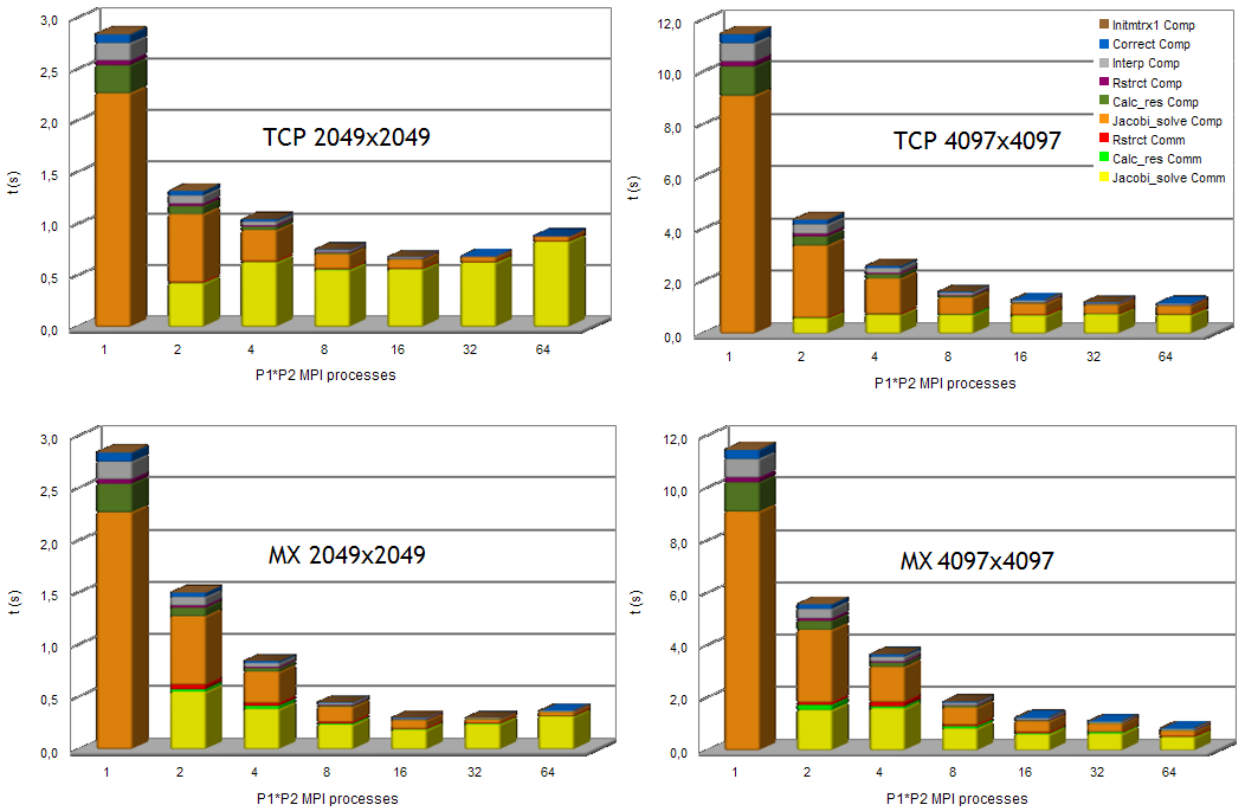


Σχήμα 30: Χρόνος σύγκλισης του 3Δ FMG (ncycles=2) για μέγεθος πίνακα 65x65x65, για διάφορες τιμές του ngrids. Ο αλγόριθμος είναι υλοποιημένος με MPI και εκτελείται σε 16 επεξεργαστές για δύο σχήματα MPI διεργασιών ενώ φαίνεται και η σειριακή υλοποίηση.

Εκτός από το κατώτατο όριο που υπάρχει στο μέγεθος του πίνακα ώστε να έχουμε κέρδος στην παράλληλη έκδοση, είδαμε ότι υπάρχει και ένα κατώτατο όριο στον αριθμό των πλεγμάτων για κάθε μέγεθος πίνακα. Αυτό ορίζει και ένα ανώτατο όριο στο πλήθος των διεργασιών που μπορούν να χρησιμοποιηθούν για κάθε μέγεθος πίνακα.

Κατανομή Χρόνου ανάμεσα στις Συναρτήσεις

Υπάρχει όμως και ένα δεύτερο όριο στο πλήθος των διεργασιών. Όταν ο χρόνος επικοινωνίας γίνει αρκετά μικρός, η αύξηση του πλήθους των διεργασιών θα προκαλεί μεγαλύτερη αύξηση στο χρόνο επικοινωνίας από τη μείωση στο χρόνο υπολογισμών (ακόμα και αν ποσοστιαία ισχύει το αντίστροφο) και τότε ο συνολικός χρόνος σύγκλισης θα μεγαλώνει. Αυτό μπορούμε να το παρατηρήσουμε για δύο διαφορετικά μεγέθη (2049×2049 για ngrids=7 και 4097×4097 για ngrids=8) και για δύο περιπτώσεις δικτύου διασύνδεσης (Ethernet & Myrinet) στο παρακάτω σχήμα. Στο κάτω μέρος (με εντονότερα χρώματα) έχουμε την κατανομή του χρόνου επικοινωνίας ενώ στο επάνω απεικονίζουμε την κατανομή του χρόνου υπολογισμών ανάμεσα στις συναρτήσεις του αλγορίθμου.



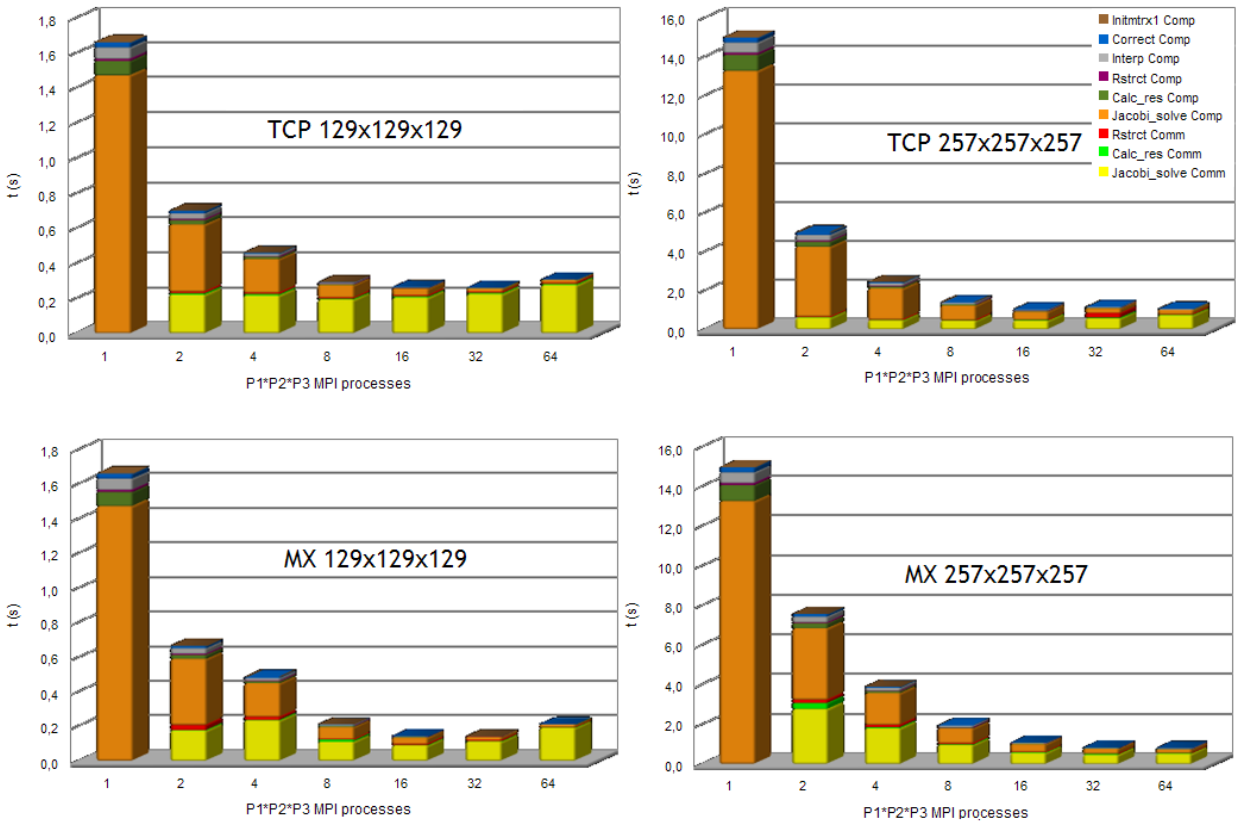
Σχήμα 31: Χρόνος σύγκλισης (κατανεμημένος σε επικοινωνία και υπολογισμούς για κάθε συνάρτηση) του 2Δ FMG (ncycles=2) για δύο μεγέθη πίνακα (2049x2049 για ngrids=7 & 4097x4097 για ngrids=8) καθώς πληθαίνουν οι MPI διεργασίες (1: Serial). Ο αλγόριθμος είναι υλοποιημένος με MPI και εκτελείται σε 64 επεξεργαστές και για δύο περιπτώσεις δικτύου διασύνδεσης (Ethernet & Myrinet).

Το παραπάνω σχήμα είναι αρκετά διαφωτιστικό και μας δίνει πολλές πληροφορίες. Καταρχάς, πιο *χρονοβόρα* συνάρτηση, τόσο στο πεδίο των υπολογισμών όσο και στο πεδίο της επικοινωνίας είναι σε όλες τις περιπτώσεις η συνάρτηση εξομάλυνσης και επίλυσης αδρότερου πλέγματος, jacobi_solve, ακολουθούμενη από τις συναρτήσεις calc_res και interpolate. Όσο για τα δύο δίκτυα διασύνδεσης, το δίκτυο Myrinet υπερέχει του Ethernet ως δίκτυο διασύνδεσης σε δύο βασικά σημεία. Έχει μεγαλύτερο *εύρος ζώνης* (bandwidth) και μικρότερο *χρόνο απόκρισης* (latency). Το Ethernet φέρεται να υπερέχει στα μικρά πλήθη MPI διεργασιών και αντίστοιχα το Myrinet στα μεγαλύτερα. Πάντως, στο μεγάλο μέγεθος φαίνονται αρκετά συγκρίσιμα.

Παρατηρούμε τη μικρή διακύμανση (αύξηση για το Ethernet) στο χρόνο επικοινωνίας καθώς πληθαίνουν οι MPI διεργασίες σε σχέση με τη ραγδαία μείωση στον χρόνο υπολογισμών. Επιπλέον, μπορούμε να διαπιστώσουμε ότι για μεγαλύτερο μέγεθος πίνακα έχουμε σημαντικά μεγαλύτερο κέρδος με την αύξηση του πλήθους των διεργασιών. Και οι δύο αυτές παρατηρήσεις έχουν να κάνουν με τη διαφορά μιας τάξης μεγέθους ανάμεσα στο κόστος επικοινωνίας και το κόστος υπολογισμών. Όταν, όμως, το κόστος των

υπολογισμών είναι αρκετά μικρό, περαιτέρω αύξηση των διεργασιών δεν επηρεάζει ορατά το ήδη μικρό κόστος υπολογισμών και έχει μάλιστα το αντίθετο από το επιθυμητό αποτέλεσμα με αποτέλεσμα ο συνολικός χρόνος να καθορίζεται κατά βάση από τη διακύμανση στο χρόνο επικοινωνίας. Κάπου εκεί σταματά να αποδίδει η αύξηση του πλήθους των MPI διεργασιών.

Ακολουθούν τα αντίστοιχα διαγράμματα για την 3Δ περίπτωση για τα μεγέθη 129×129×129 (για ngrids=4) και 257×257×257 (για ngrids=8):



Σχήμα 32: Χρόνος σύγκλισης (κατανεμημένος σε επικοινωνία και υπολογισμούς για κάθε συνάρτηση) του 3Δ FMG (ncycles=2) για δύο μεγέθη πίνακα (129x129x129 για ngrids=4 & 257x257x257 για ngrids=5) καθώς πληθαίνουν οι MPI διεργασίες (1: Serial). Ο αλγόριθμος είναι υλοποιημένος με MPI και εκτελείται σε 64 επεξεργαστές για δύο περιπτώσεις δικτύου διασύνδεσης (Ethernet & Myrinet).

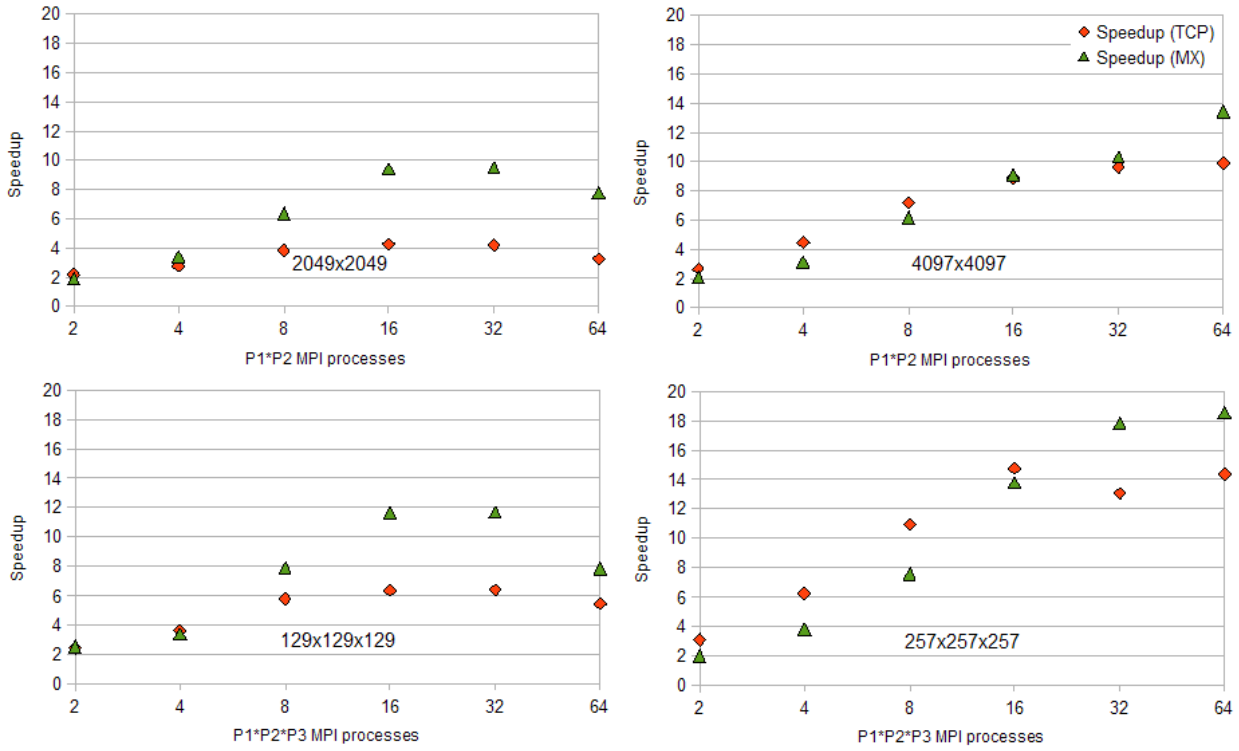
Πιο χρονοβόρα συνάρτηση, τόσο στο πεδίο των υπολογισμών όσο και στο πεδίο της επικοινωνίας είναι σαφώς η συνάρτηση εξομάλυνσης και επίλυσης αδρότερου πλέγματος, jacobi_solve, ακολουθούμενη από τις συναρτήσεις calc_res και interpolate. Το Ethernet φέρεται να υπερέχει και πάλι στα μικρά πλήθη MPI διεργασιών και αντίστοιχα το Myrinet στα μεγαλύτερα.

Όπως και στις 2Δ, υπάρχει η μικρή διακύμανση (αύξηση για το Ethernet) στο χρόνο επικοινωνίας καθώς πληθαίνουν οι MPI διεργασίες ενώ έχουμε απότομη μείωση στον

χρόνο υπολογισμών (περισσότερο από ό,τι στις 2Δ λόγω της διαφοράς μιας τάξης μεγέθους στο κόστος υπολογισμών). Για μεγαλύτερο μέγεθος πίνακα έχουμε σημαντικά μεγαλύτερο κέρδος με την αύξηση του πλήθους των διεργασιών. Όπως και στις 2Δ, έτσι και εδώ, οι δύο αυτές παρατηρήσεις έχουν να κάνουν με τη διαφορά μιας τάξης μεγέθους ανάμεσα στο κόστος επικοινωνίας και το κόστος υπολογισμών. Όταν, όμως, το κόστος των υπολογισμών γίνει αρκετά μικρό, περαιτέρω αύξηση των διεργασιών δεν επηρεάζει ορατά το ήδη μικρό κόστος υπολογισμών και ο συνολικός χρόνος καθορίζεται κατά βάση από τη διακύμανση στο χρόνο επικοινωνίας. Κάπου εκεί σταματά να αποδίδει η αύξηση του πλήθους των MPI διεργασιών.

Επιτάχυνση του Παράλληλου Αλγορίθμου

Ας δούμε τώρα την *επιτάχυνση* (speedup) των παράλληλων 2Δ και 3Δ FMG αλγορίθμων, όταν υλοποιούνται με το MPI και εκτελούνται σύμφωνα με το προηγούμενο σενάριο μετρήσεων:



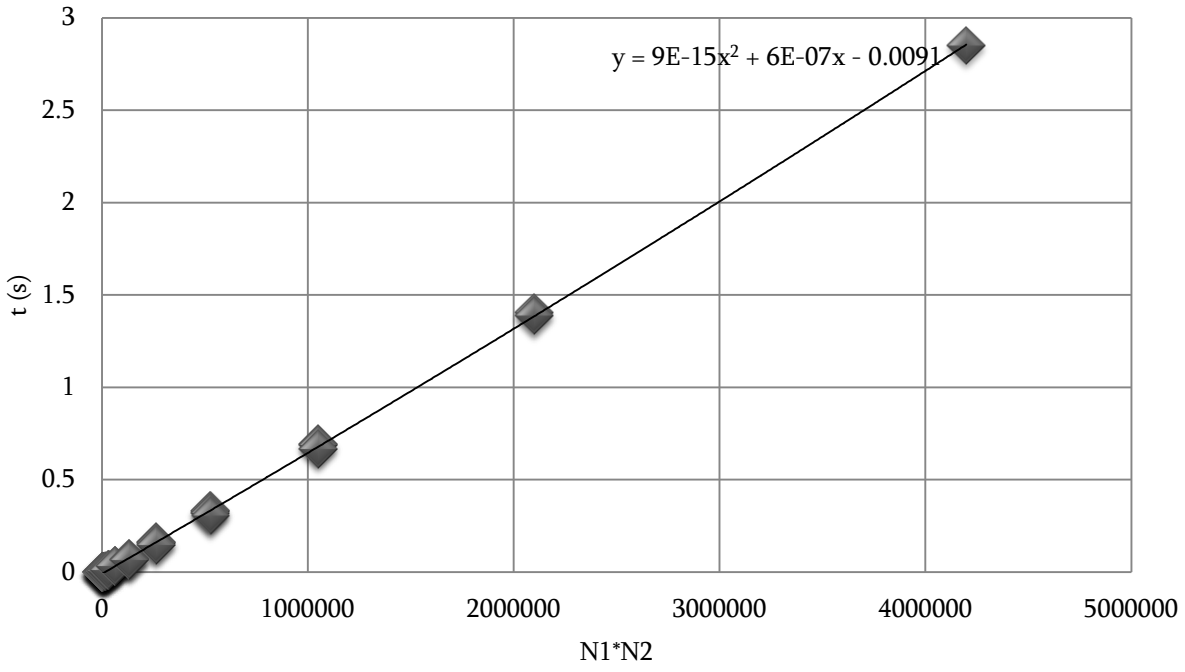
Σχήμα 33: Επιτάχυνση του FMG (ncycles=2, 2Δ & 3Δ) καθώς πληθαίνουν οι MPI διεργασίες, για μεγέθη πίνακα 2049x2049 με ngrids=7, 4097x4097 με ngrids=8 για 2Δ και 129x129x129 με ngrids=4, 257x257x257 με ngrids=5 για 3Δ. Ο αλγόριθμος είναι υλοποιημένος με MPI και εκτελείται σε 64 επεξεργαστές για δύο περιπτώσεις δικτύου διασύνδεσης (Ethernet & Myrinet).

Μπορούμε να παρατηρήσουμε και εδώ ότι για μικρά πλήθη MPI διεργασιών είναι προτιμότερη η χρησιμοποίηση του Ethernet ως δίκτυο διασύνδεσης των υπολογιστικών κόμβων ενώ για μεγάλα πλήθη MPI διεργασιών είναι καλύτερο το Myrinet. Επιπλέον, για το «μικρό» μέγεθος πίνακα, η υπεροχή του Myrinet έναντι του Ethernet υφίσταται σε ένα μεγαλύτερο φάσμα πλήθους MPI διεργασιών από ό,τι για το «μεγάλο». Όσο για την ίδια την επιτάχυνση, αυτή είναι καλύτερη για το μεγάλο μέγεθος πίνακα αφού αυξάνει (με μειούμενο ρυθμό) σε όλο το εύρος του πλήθους των διεργασιών ενώ στο μικρό μέγεθος παρατηρείται μείωση μετά τις 16 διεργασίες.

Παράλληλος Αλγόριθμος FMG (OpenMP)

Μετρήσεις και Διαγράμματα Χρόνου Σύγκλισης

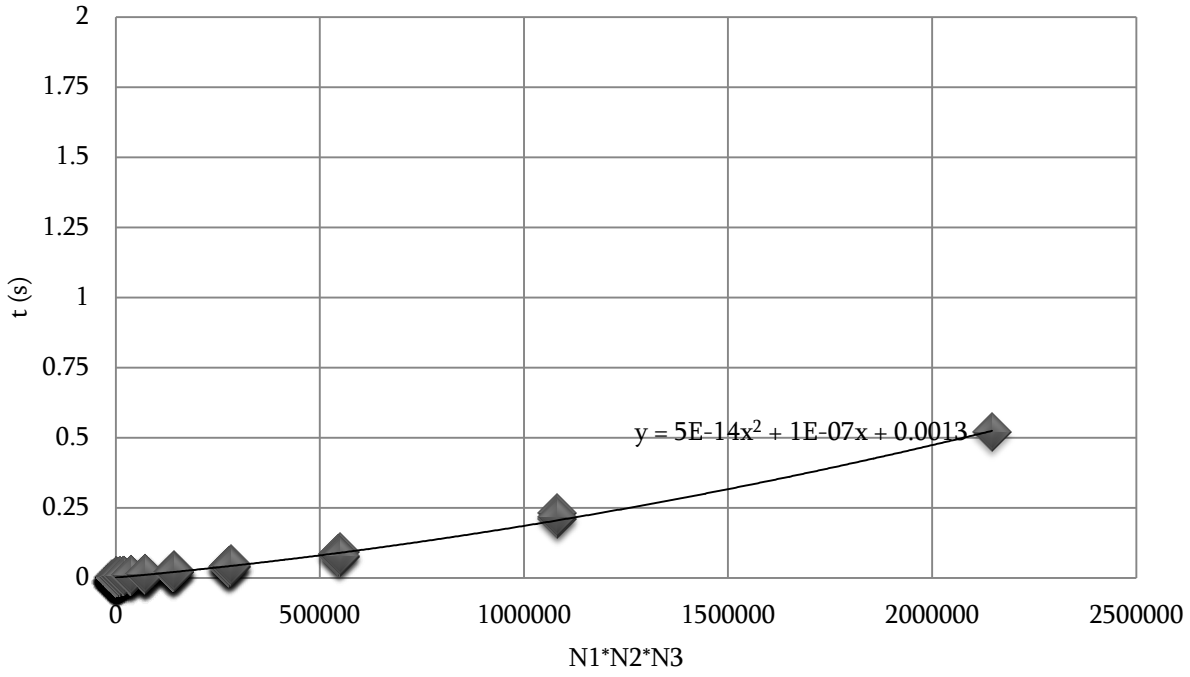
Σε αυτό το σημείο θα χρησιμοποιήσουμε τη μέθοδο FMG υλοποιημένη με τη βοήθεια του OpenMP για την επίλυση του προβλήματος. Παρακάτω απεικονίζεται η μεταβολή του χρόνου σύγκλισης με την αύξηση του μεγέθους του πίνακα στην παράλληλη υλοποίηση της μεθόδου FMG (ncycles=2). Η εκτέλεση γίνεται σε 8 επεξεργαστικά στοιχεία που βρίσκονται σε αρχιτεκτονική κοινής μνήμης και για 8 νήματα:



Σχήμα 34: Χρόνος σύγκλισης του 2Δ FMG (ncycles=2) με αύξηση του μεγέθους του πίνακα. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή για 8 νήματα.

Παρατηρούμε ότι το παραπάνω διάγραμμα ταιριάζει πάρα πολύ με εκείνα των προηγούμενων μεθόδων FMG. Και οι τρεις μέθοδοι (σειριακή, MPI, OpenMP) παρουσιάζουν «σχεδόν» *γραμμική* αύξηση του χρόνου σύγκλισης με την αύξηση του μεγέθους του πίνακα.

Παρόμοια συμπεριφορά παρατηρούμε και στην περίπτωση της αντίστοιχης 3Δ FMG μεθόδου υλοποιημένης με OpenMP. Εδώ η εκτέλεση γίνεται σε 8 επεξεργαστικά στοιχεία που βρίσκονται σε αρχιτεκτονική κοινής μνήμης και για 8 νήματα:



Σχήμα 35: Χρόνος σύγκλισης του 3Δ FMG (ncycles=2) με αύξηση του μεγέθους του πίνακα. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή για 8 νήματα.

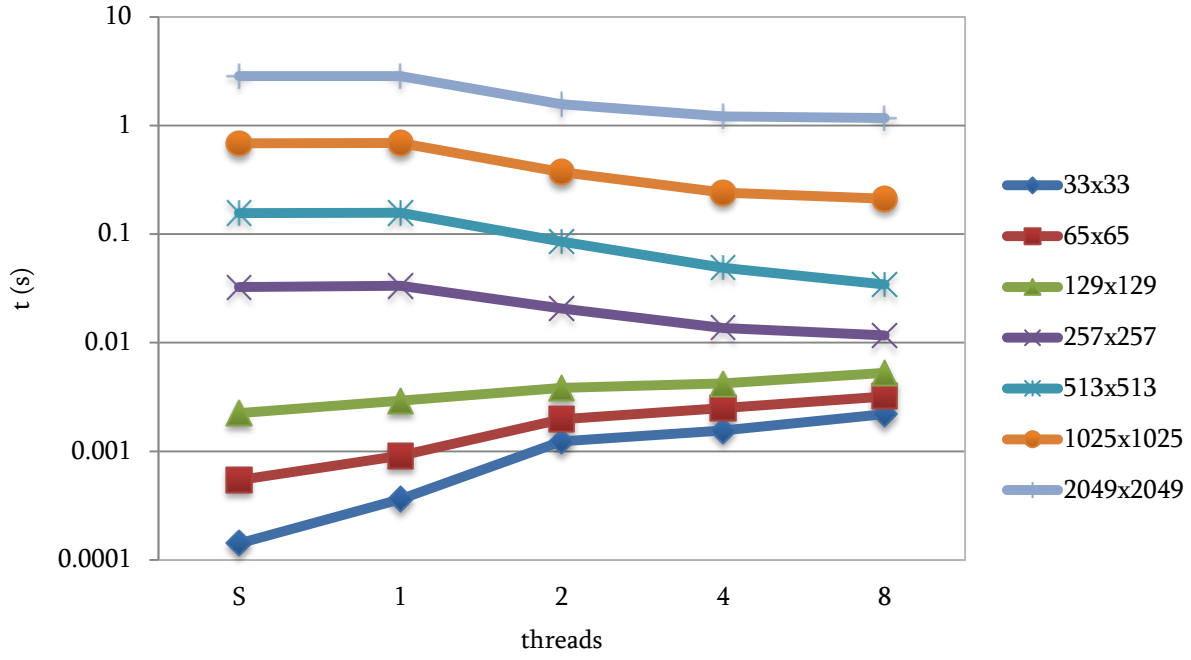
Ας δούμε τώρα έναν πίνακα χρόνου εύρεσης λύσης της τελικής λύσης για διάφορα τετραγωνικά μεγέθη του πίνακα επίλυσης σε σχέση με τον αριθμό των νημάτων που δημιουργούνται. Στη στήλη Serial έχουμε τα αποτελέσματα της σειριακής έκδοσης:

Πίνακας 4:

Μετρήσεις χρόνου σύγκλισης του 2Δ αλγορίθμου FMG (OpenMP) για τετραγωνικά μεγέθη

FMG (nc=2) array	Serial	OpenMP threads			
		1	2	4	8
33x33	0.000142	0.000363	0.001235	0.001555	0.002195
65x65	0.000544	0.000912	0.001976	0.002499	0.003201
129x129	0.002248	0.002925	0.003841	0.004213	0.005277
257x257	0.032496	0.033432	0.020675	0.013702	0.011683
513x513	0.156212	0.15715	0.085565	0.049037	0.03428
1025x1025	0.685437	0.687854	0.371773	0.240412	0.211318
2049x2049	2.849948	2.855417	1.569477	1.215522	1.172565

Για καλύτερα συμπεράσματα από τον παραπάνω πίνακα, παραθέτουμε και το αντίστοιχο χρονικό διάγραμμα σε σχέση με το πλήθος των νημάτων και για τα διάφορα μεγέθη πίνακα (ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα):



Σχήμα 36: Χρόνος σύγκλισης της 2Δ μεθόδου FMG (ncycles=2) σε σχέση με τον αριθμό των OpenMP νημάτων (S: Serial) και για διάφορα μεγέθη πίνακα. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή για 8 νήματα.

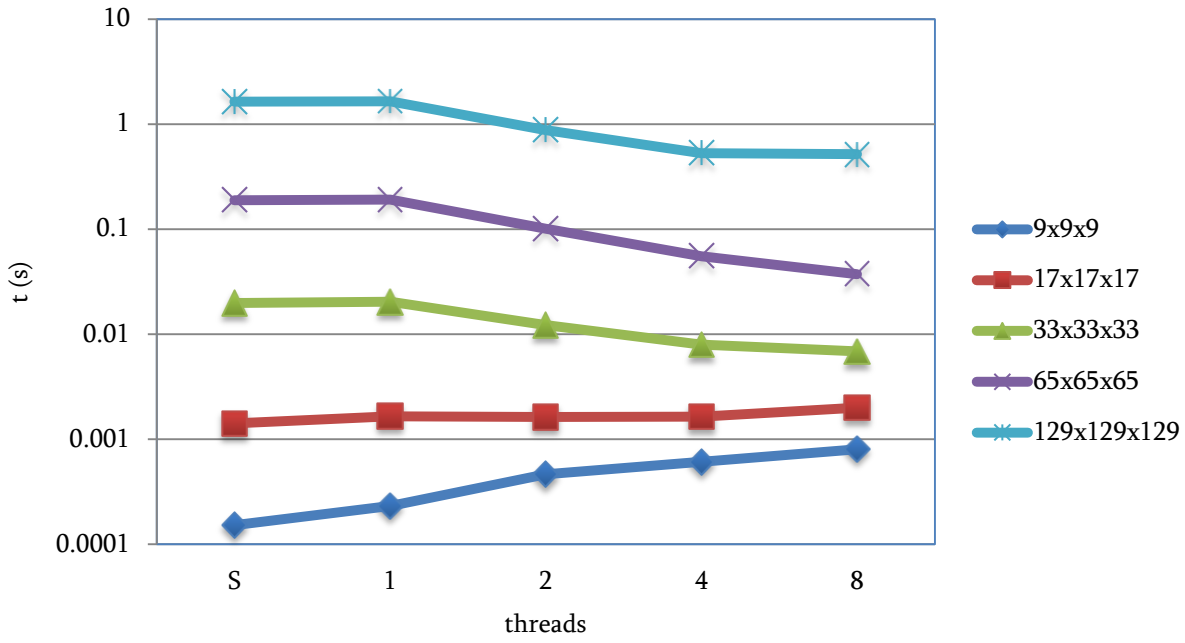
Για τρεις διαστάσεις, ο πίνακας για διάφορα κυβικά μεγέθη του πίνακα επίλυσης και το αντίστοιχο διάγραμμα έχουν ως εξής:

Πίνακας 5:

Μετρήσεις χρόνου σύγκλισης του 3Δ αλγορίθμου FMG (OpenMP) για κυβικά μεγέθη

FMG 3D (nc=2) array	Serial	OpenMP threads			
		1	2	4	8
9x9x9	0.000152	0.000231	0.000463	0.00061	0.000805
17x17x17	0.001413	0.001657	0.001625	0.00164	0.001993
33x33x33	0.019748	0.020357	0.012197	0.007974	0.006845
65x65x65	0.189042	0.191483	0.101254	0.055335	0.037272
129x129x129	1.641496	1.649274	0.878075	0.530525	0.517648

Ο άξονας του χρόνου είναι σε λογαριθμική κλίμακα:



Σχήμα 37: Χρόνος σύγκλισης της 3Δ μεθόδου FMG (ncycles=2) σε σχέση με τον αριθμό των OpenMP νημάτων (S: Serial) και για διάφορα μεγέθη πίνακα. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή για 8 νήματα.

Περισσότερα νήματα δε σημαίνουν απαραίτητα μικρότερο χρόνο σύγκλισης. Έχει σημασία το μέγεθος του πίνακα καθώς για μεγέθη μικρότερα του 257×257 (33×33×33 για 3Δ) παρατηρούμε ότι η δημιουργία νημάτων κοστίζει περισσότερο από τη σειριακή μέθοδο. Αξίζει να θυμίσουμε ότι και στις προηγούμενες παράλληλες μεθόδους, περίπου στο ίδιο

μέγεθος είχαμε το κομβικό σημείο όπου η παραλληλοποίηση για μικρότερα μεγέθη άρχιζε να γίνεται ασύμφορη ενώ για μεγαλύτερα βελτιωνόταν σημαντικά ο χρόνος σύγκλισης. Στο OpenMP, πάντως, φαίνεται ότι για μικρά μεγέθη, πέρα από τη δημιουργία τους, η ίδια η αύξηση των νημάτων καθυστερεί το πρόγραμμα ενώ για μεγάλα συμφέρει πολύ.

Πλήθος Πλεγμάτων και Σύγκλιση

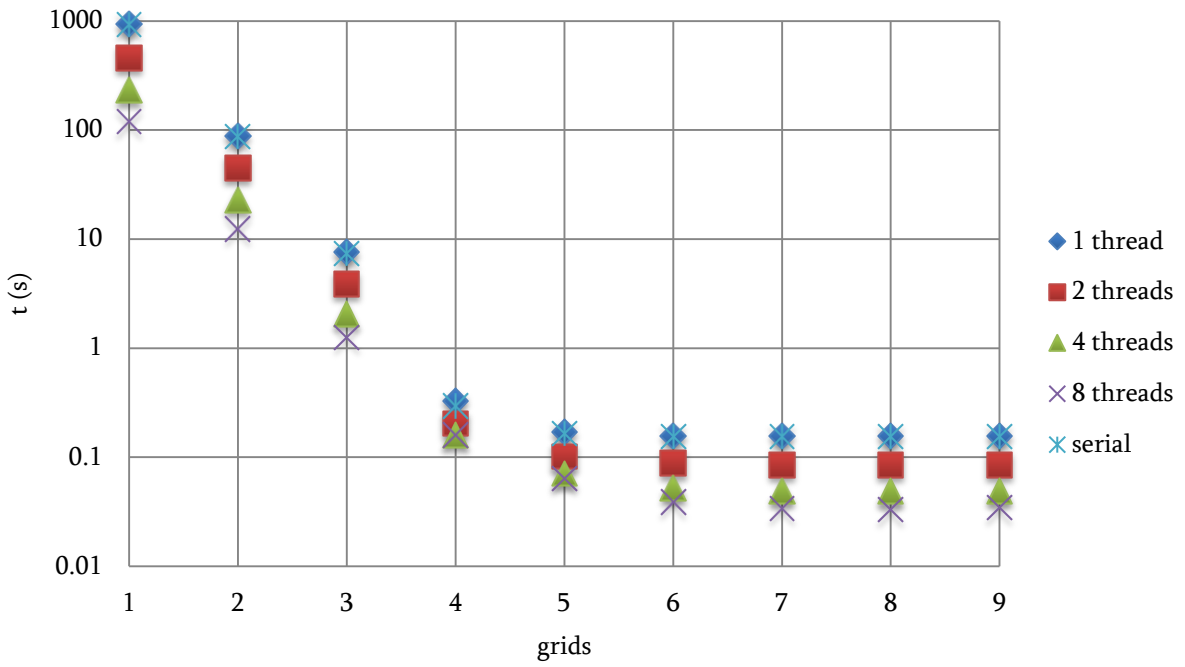
Όσο για το πόσα πλέγματα είναι καλύτερο να έχουμε, εδώ δεν έχουμε τους περιορισμούς που επέβαλλε η υλοποίηση με MPI. Μπορούμε κάλλιστα, όπως και στη σειριακή έκδοση, να πάμε μέχρι το αραιότερο δυνατό πλέγμα. Αν θέσουμε ένα μόνο πλέγμα, θα έχουμε στην ουσία την Jacobi OpenMP (αν και δεν την έχουμε αναλύσει ξεχωριστά). Όσο, μάλιστα, μειώνεται το πλήθος των πλεγμάτων, τόσο πλησιάζουμε τη Jacobi OpenMP στην ταχύτητα αλλά και στην ποιότητα σύγκλισης. Παραθέτουμε τον πίνακα και το αντίστοιχο διάγραμμα για μέγεθος πίνακα 513×513, ncycles ίσο με 2, για διάφορα πλήθη νημάτων και για πλέγματα από 1 μέχρι 9. Στη στήλη Serial έχουμε βάλει τους αντίστοιχους χρόνους για την αντίστοιχη σειριακή FMG.

Πίνακας 6:

Μετρήσεις χρόνου σύγκλισης του 2Δ αλγορίθμου FMG (OpenMP) για μέγεθος 513x513

FMG 2D (nc=2) 513x513	Serial	OpenMP threads			
grids		1	2	4	8
1	932.145008	930.693778	461.945961	231.172687	119.182214
2	87.276386	88.988805	44.859838	22.787218	12.384232
3	7.41393	7.576399	3.891948	2.055132	1.255931
4	0.29634	0.329211	0.20348	0.159993	0.159846
5	0.165232	0.170078	0.101822	0.071212	0.064296
6	0.155813	0.158094	0.088148	0.052304	0.038951
7	0.155137	0.156898	0.085319	0.04899	0.033993
8	0.154974	0.157078	0.085236	0.048547	0.033196
9	0.155002	0.15683	0.085153	0.048937	0.03435

Ακολουθεί το αντίστοιχο διάγραμμα (λογαριθμική κλίμακα χρόνου):



Σχήμα 38: Χρόνος σύγκλισης του 2Δ FMG (ncycles=2) για μέγεθος πίνακα 513x513, για διάφορες τιμές του ngrids και για διάφορα πλήθη νημάτων. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή ενώ φαίνεται και η σειριακή υλοποίηση.

Με αμελητέες διαφορές στις πέντε (ο ίδιος αριθμός ίσχυε και στην περίπτωση της σειριακής έκδοσης) τελευταίες περιπτώσεις πλεγμάτων, μπορούμε να συμπεράνουμε ό,τι και στη σειριακή έκδοση. Δηλαδή, ένας αριθμός πλεγμάτων (ngrids) μέχρι πέντε μικρότερος (δηλαδή για αδρότερο το πλέγμα με πίνακα $33\times33 = [2^5 + 1]\times[2^5 + 1]$) από τον μέγιστο δυνατό είναι γενικά ανεκτός. Βέβαια, επειδή τόσο στη σειριακή FMG όσο και στην υλοποιημένη με το OpenMP FMG με το μέγιστο δυνατό ngrids έχουμε και την ταχύτερη σύγκλιση, θα πρέπει να έχουμε κάποιο σημαντικό λόγο ώστε να μειώσουμε το ngrids. Στην περίπτωσή μας, ο λόγος αυτός αφορά την υλοποίηση με το MPI, όπου το αδρότερο πλέγμα αναπόφευκτα αλλάζει αφού αντιστοιχεί πια στο αδρότερο ανά διεργασία πλέγμα. Έτσι, έχουμε κέρδος από την εκτέλεση σε περισσότερες διεργασίες αλλά ζημία λόγω του χαμηλότερου (από το μέγιστο δυνατό) ngrids. Η διαφορά αυτών των δύο ορίζει το τελικό κέρδος μείωσης του ngrids. Βέβαια, όσο μεγαλώνει το μέγεθος του πίνακα, όλο και μικρότερες τιμές του ngrids μπορούν να γίνουν αποδεκτές για δύο λόγους. Πρώτον, η ζημία λόγω του ακόμα χαμηλότερου ngrids είναι μικρή σε σχέση με την αύξηση του χρόνου σύγκλισης για το μεγαλύτερο μέγεθος. Δεύτερον, αν σημαίνει χρησιμοποίηση περισσότερων διεργασιών, η ζημία θα είναι πιθανότατα μικρή σε σχέση με το κέρδος που θα αποφέρουν οι περισσότερες διεργασίες.

Με τον ίδιο τρόπο, στην 3Δ περίπτωση παραθέτουμε τον πίνακα και το αντίστοιχο διάγραμμα για μέγεθος πίνακα $65\times65\times65$, ncycles ίσο με 2, για διάφορα πλήθη νημάτων

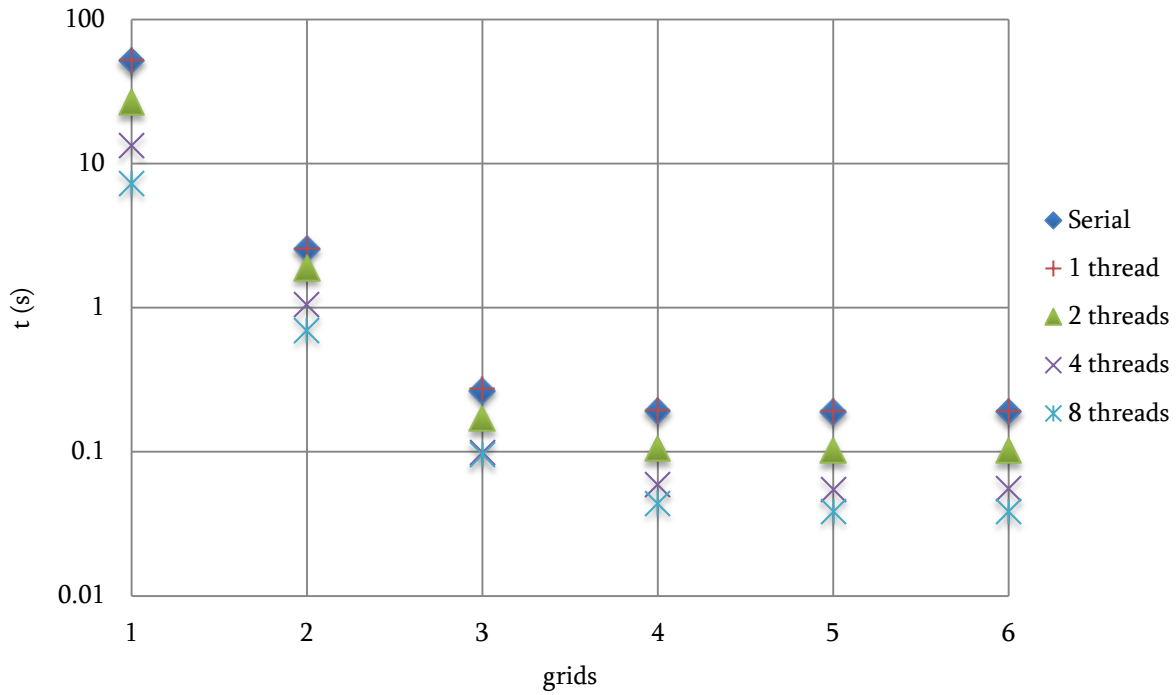
και για πλέγματα από 1 μέχρι 6. Στη στήλη Serial έχουμε βάλει τους αντίστοιχους χρόνους για την αντίστοιχη σειριακή FMG.

Πίνακας 7:

Μετρήσεις χρόνου σύγκλισης του 3Δ αλγορίθμου FMG (OpenMP) για μέγεθος 65x65x65

FMG 3D (nc=2) 65x65x65 grids	Serial	OpenMP threads			
		1	2	4	8
1	51.90904	52.24508	26.49617	13.25033	7.284955
2	2.541458	2.567969	1.875173	1.044408	0.696932
3	0.264855	0.273953	0.169405	0.098198	0.095194
4	0.191702	0.194313	0.105156	0.059079	0.043777
5	0.189389	0.191408	0.10166	0.054941	0.038781
6	0.189481	0.191819	0.101813	0.055376	0.03825

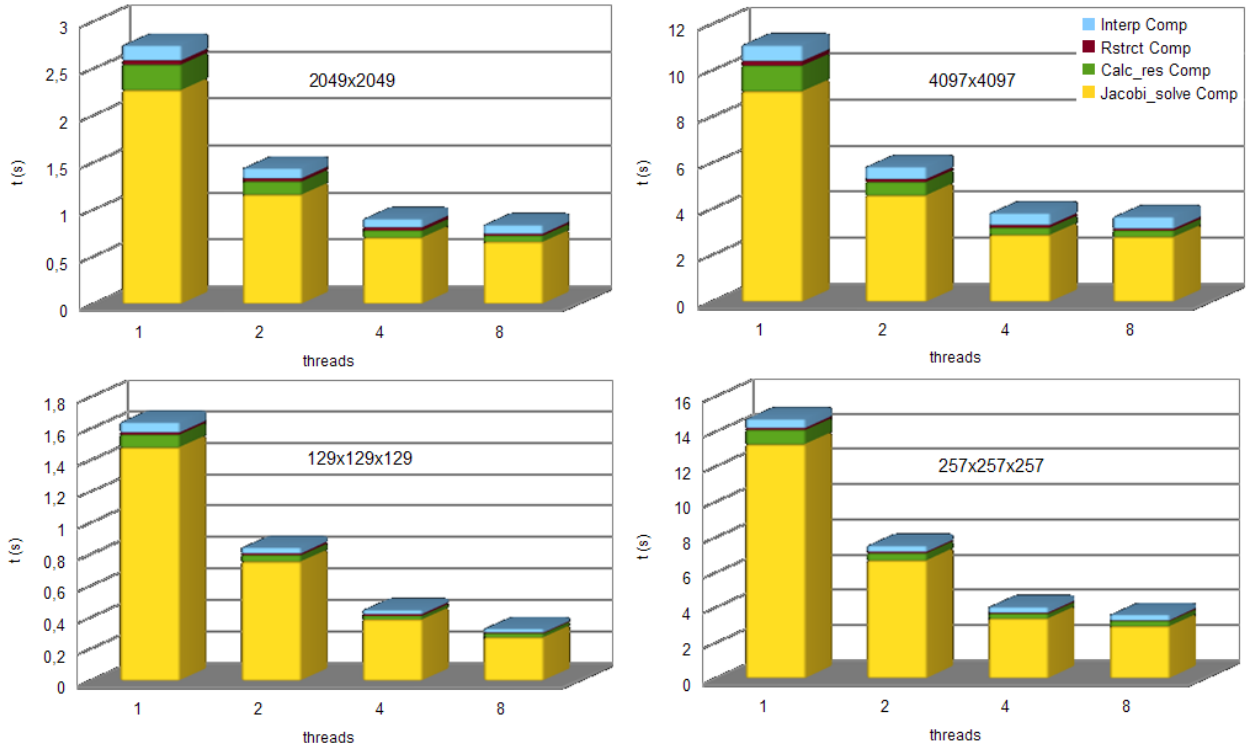
Ακολουθεί το αντίστοιχο διάγραμμα (λογαριθμική κλίμακα χρόνου):



Σχήμα 39: Χρόνος σύγκλισης του 3Δ FMG (ncycles=2) για μέγεθος πίνακα 65x65x65, για διάφορες τιμές του ngrids και για διάφορα πλήθη νημάτων. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε οκταπύρηνο επεξεργαστή ενώ φαίνεται και η σειριακή υλοποίηση.

Κατανομή Χρόνου ανάμεσα στις Συναρτήσεις

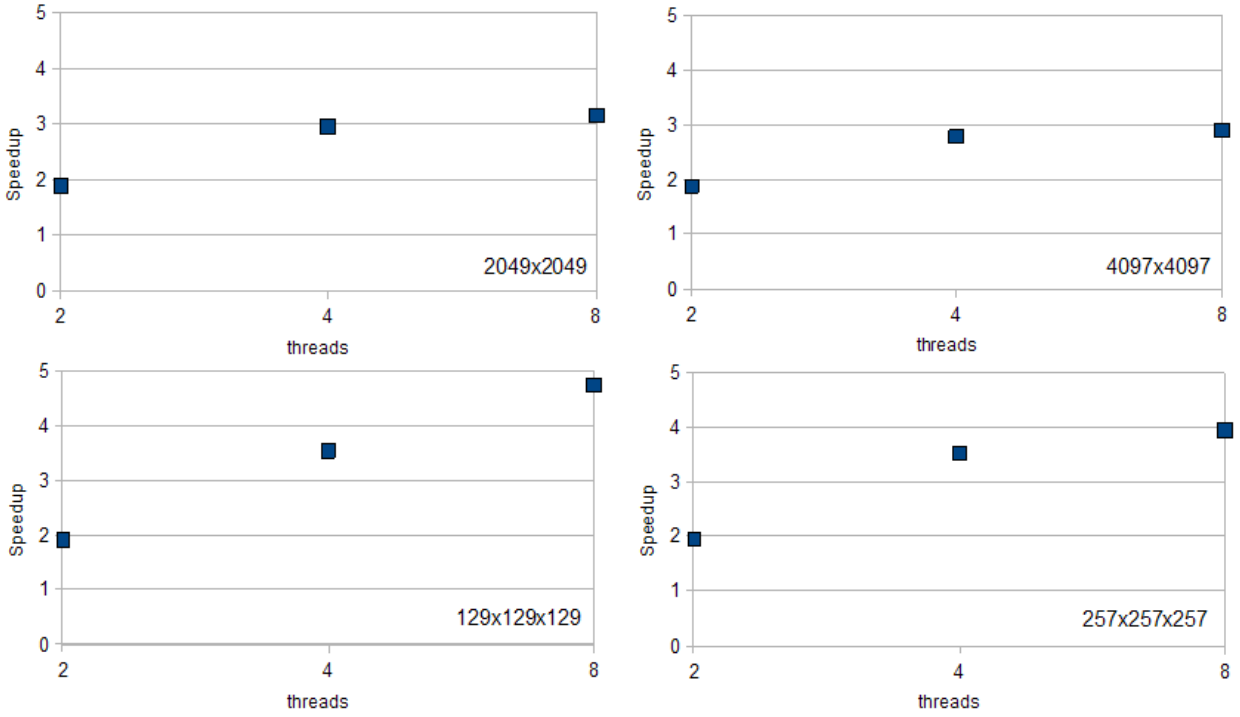
Το παρακάτω σχήμα δείχνει την κατανομή του χρόνου ανάμεσα στις συναρτήσεις του αλγορίθμου για δύο μεγέθη πίνακα σε κάθε περίπτωση διαστάσεων.



Σχήμα 40: Χρόνος σύγκλισης (κατανεμημένος στις συναρτήσεις) του FMG (ncycles=2) για υλοποίηση με OpenMP και εκτέλεση σε οκταπύρηνο επεξεργαστή για διάφορα πλήθη νημάτων (μεγέθη πίνακα: 2049x2049, 4097x4097 για 2Δ και 129x129x129, 257x257x257 για 3Δ)

Όπως και στην περίπτωση της υλοποίησης με MPI σε αρχιτεκτονική κατανεμημένης μνήμης, έτσι και με το OpenMP σε αρχιτεκτονική κοινής μνήμης, το κέρδος από την εκμετάλλευση της παραλληλίας του αλγορίθμου αυξάνεται όσο αυξάνεται το μέγεθος του πίνακα προς επίλυση. Για ένα δεδομένο μέγεθος πίνακα, αν είναι μεγαλύτερο από 129×129 για 2Δ ή από 17×17×17 για 3Δ, το κέρδος αυτό μεγαλώνει όσο πληθαίνουν τα νήματα εκτέλεσης.

Παρακάτω βλέπουμε την επιτάχυνση των 2Δ και 3Δ αλγορίθμων όταν υλοποιούνται με το OpenMP και εκτελούνται σύμφωνα με το παραπάνω σενάριο μετρήσεων:



Σχήμα 41: Επιτάχυνση του FMG (ncycles=2, 2Δ & 3Δ) καθώς πληθαίνουν τα OpenMP νήματα, για μεγέθη πίνακα 2049x2049, 4097x4097 για 2Δ και 129x129x129, 257x257x257 για 3Δ. Ο αλγόριθμος είναι υλοποιημένος με OpenMP και εκτελείται σε έναν οκταπύρηνο επεξεργαστή.

Εύκολα παρατηρούμε ότι η επιτάχυνση είναι μεγαλύτερη στην 3Δ περίπτωση. Αυτό συμβαίνει διότι το κέρδος παραλληλοποίησης είναι μεγαλύτερο στην 3Δ περίπτωση, όπου έχουμε κόστος υπολογισμών τάξης $O(N^3)$, από ό,τι είναι στην 2Δ περίπτωση, όπου το κόστος υπολογισμών είναι της τάξης του $O(N^2)$.

Συγκρίνοντας, επίσης, τα σχήματα που αφορούν την επιτάχυνση, για τις δύο παράλληλες FMG μεθόδους (OpenMP και MPI), διαπιστώνουμε τη σημαντική υπεροχή της χρησιμοποίησης διεργασιών έναντι της χρησιμοποίησης νημάτων, τουλάχιστον όσον αφορά την περίπτωση του FMG αλγορίθμου σε δύο και τρεις διαστάσεις.

Κεφάλαιο 10

Επίλογος

Σύνοψη

Είναι γεγονός ότι δεν υπάρχει ένας πολυπλεγματικός αλγόριθμος που να λύνει όλα ελλειπτικά προβλήματα. Υπάρχουν, όμως, συγκεκριμένες μέθοδοι ώστε να προσαρμόσουμε τη βασική μορφή ενός πολυπλεγματικού αλγορίθμου ώστε να επιλύει το πρόβλημα που επιθυμούμε. Εμείς εξετάσαμε την προσαρμογή του πολυπλεγματικού αλγορίθμου ώστε να επιλύει το πρόβλημα Dirichlet σε δύο και τρεις διαστάσεις. Κάποιες τεχνικές, όπως η τεχνική της εμφωλευμένης επανάληψης και κάποιες συναρτήσεις, όπως οι συναρτήσεις διαπεγματικών μεταβάσεων είναι κοινές στους περισσότερους πολυπλεγματικούς αλγορίθμους και παραμετροποιούνται μόνο κατάλληλα ανά περίπτωση. Άλλα στοιχεία, όμως, όπως η μορφή της συνάρτησης επίλυσης και εξομάλυνσης ή της συνάρτησης υπολογισμού υπολοίπου διαφέρουν πολύ από αλγόριθμο σε αλγόριθμο, καθώς εξαρτώνται από τη διαφορική εξίσωση του προβλήματος.

Επιπλέον, είδαμε τον τρόπο με τον οποίο μπορούμε να βελτιώσουμε την επίδοση ενός πολυπλεγματικού αλγορίθμου υλοποιώντας τον με χρήση των προγραμματιστικών μοντέλων OpenMP και MPI σε αρχιτεκτονικές κοινής και κατανεμημένης μνήμης αντίστοιχα. Έχουμε πλέον τις γνώσεις ώστε ανάλογα με το πρόβλημα, να παραμετροποιήσουμε κατάλληλα τον παράλληλο πολυπλεγματικό αλγόριθμο επίλυσης ώστε να έχει τη μέγιστη επιτάχυνση σε σχέση με τον αντίστοιχο σειριακό.

Προεκτάσεις

Βεβαίως, υπάρχουν ακόμα περιθώρια βελτίωσης. Καταρχάς, μπορεί να γίνει μια από κοινού υλοποίηση χρησιμοποιώντας και τα δύο πρότυπα, OpenMP και MPI. Με εκτέλεση ενός υλοποιημένου με αυτό τον τρόπο αλγορίθμου σε μια υβριδική αρχιτεκτονική αναμένεται να έχει ακόμα καλύτερη επίδοση από τους δύο προηγούμενους παράλληλους αλγορίθμους.

Επιπλέον, έγινε εκτενής ανάλυση του κόστος που θα έχει η χρησιμοποίηση λιγότερων πλεγμάτων από τον μέγιστο δυνατό αριθμό τους στις παραπάνω πολυπλεγματικές υλοποιήσεις. Τόσο στη σειριακή υλοποίηση όσο και στην παραλληλοποιημένη με το

OpenMP, δεν υπάρχει λόγος χρησιμοποίησης λιγότερων πλεγμάτων. Ο λόγος που πιθανότατα θα επιθυμούσαμε λιγότερα πλέγματα αφορά την ικανοποίηση των περιορισμών στον αριθμό τους, τους οποίους και θέτουν τα διάφορα σχήματα MPI διεργασιών. Χρησιμοποιώντας σε τετραγωνικά και κυβικά σχήματα διεργασιών όσο το δυνατόν περισσότερες διεργασίες, μείωση του πλήθους των πλεγμάτων κατά n, για τις δύο διαστάσεις θα σημαίνει πολλαπλασιασμό των διεργασιών επί 4^n ενώ ο αντίστοιχος πολλαπλασιαστής για τις τρεις διαστάσεις θα είναι ο 8^n. Επομένως, στο βαθμό που για ένα μέγεθος μπορούμε να χρησιμοποιήσουμε έναν αριθμό πλεγμάτων λιγότερο από τον μέγιστο δυνατό χωρίς σημαντικό χρονικό κόστος, είναι πιθανότατα καλύτερο να το κάνουμε για να χρησιμοποιήσουμε περισσότερες διεργασίες. Μάλιστα, λόγω της επίδρασης του παραπάνω πολλαπλασιαστή, το πλήθος των διεργασιών που θα μπορούμε να χρησιμοποιήσουμε θα αυξάνεται εκθετικά καθώς λιγοστεύουν τα πλέγματα.

Μέχρι εδώ, στην υλοποίηση με το MPI έχουμε αντιμετωπίσει τη δυνατότητα χρησιμοποίησης περισσότερων διεργασιών ως προνόμιο που απορρέει από τη χρήση μικρότερου πλήθους πλεγμάτων. Η σειριακή υλοποίηση, όμως, όπως και η παραλληλοποιημένη με το OpenMP δεν αντιμετωπίζουν κανέναν περιορισμό όσον αφορά το πλήθος των πλεγμάτων. Επιθυμώντας μια τέτοια ελευθερία και στην υλοποίηση με το πρότυπο MPI, το επόμενο στάδιο βελτίωσης θα είναι η σειριακή εκτέλεση του αλγορίθμου από ένα πλέγμα και μετά ώστε να μπορούμε να κερδίσουμε χρόνο και σε αυτή την υλοποίηση, φτάνοντας μέχρι το αδρότερο δυνατό πλέγμα. Φυσικά, αυτό θα γίνεται και για όλα τα αδρότερά του πλέγματα ενώ για τα λεπτομερέστερα θα επιστρέφουμε πάλι στον αριθμό των διεργασιών που είχαμε προηγουμένως. Τα πλέγματα, δηλαδή, θα είναι το δυνατόν περισσότερα και από ένα πλέγμα που θα οριστεί κατάλληλα και για τα αδρότερά του (δηλαδή για το κάτω μέρος του πλεγματικού κύκλου) μια μόνο διεργασία θα συλλέγει τα απαραίτητα δεδομένα από τις υπόλοιπες και μετά τους απαραίτητους υπολογισμούς (αφού εκτελέσει, δηλαδή, αυτό το κάτω μέρος του πλεγματικού κύκλου) θα τα επαναδιανέμει ανανεωμένα στις αντίστοιχες διεργασίες.

Για να υπάρχει ένα τέτοιο πλέγμα πρέπει προφανώς το κέρδος από τη σειριακή εκτέλεση να υπερκαλύπτει το κόστος επικοινωνίας που θα χρειαστεί για τη συλλογή από μια διεργασία των πινάκων των υπόλοιπων διεργασιών και για την εκ νέου διανομή τους μετά τους απαραίτητους υπολογισμούς. Αυτή η σειριακή εκτέλεση στη θέση μέρους της παράλληλης αναμένεται να έχει κέρδος με διττό τρόπο. Αρχικά, όπως αναφέραμε ήδη, θα σημαίνει, ανεξάρτητα από τον αριθμό των διεργασιών, δυνατότητα μετάβασης μέχρι το αδρότερο δυνατό πλέγμα. Επιπλέον, αυτή η αλλαγή πρόκειται να προσδώσει στη νέα μέθοδο το πλεονέκτημα που έχουν οι σειριακές εκδόσεις έναντι των παράλληλων στα μικρά μεγέθη πίνακα. Φυσικά, για πολύ μικρά μεγέθη πίνακα, αυτό θα σημαίνει προτίμηση εξαρχής της σειριακής εκτέλεσης έναντι της παράλληλης.

Βιβλιογραφία

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling και Brian P. Flannery: *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press and Numerical Recipes Software, URL http://www.nr.com/, 1988-1992

[2] Δ. Χ. Κραββαρίτης, Γ. Ν. Παντελίδης, *Εισαγωγή στις Διαφορικές Εξισώσεις Μερικών Παραγώγων*, κεφάλαια 1, 5, 7-9, 2003

[3] Γεώργιος Ν. Παπανδρέου, *Ανάλυση Εικόνας και Όραση Υπολογιστών: Θεωρία και Εφαρμογές στην Αποκατάσταση Αρχαίων Τοιχογραφιών*, Διδακτορική Διατριβή, κεφάλαιο 3, URL http://www.stat.ucla.edu/~gpapan/pubs/theses/Papandreou_PhD_ntua-thesis09.pdf, 2008

[4] A. Thekale, T. Gradl, K. Klamroth, U. Rüde, *Optimizing the Number of Multigrid Cycles in the Full Multigrid Algorithm*, 2009

[5] T. Wiegelmann, *Numerical Integration of PDEs*, στο International Max Planck Research School on Physical Processes in the Solar System and Beyond, Universities of Braunschweig and Göttingen, URL http://www.solar-system-school.de/, 2008

[6] Stuart Dalziel, *Numerical Methods Natural Sciences Lecture Notes*, University of Cambridge, URL http://www.damtp.cam.ac.uk/lab/people/sd/lectures/nummeth98/index.htm, 1998

[7] H. J. Sips, H. X. Lin, *Introduction to High Performance Computing Lecture Notes*, TU Delft, URL http://pds.twi.tudelft.nl/vakken/in4049TU/in4049_sheets_bb.htm, 2006

[8] X. P. Wang, *Advanced Numerical Methods II: A Multigrid Tutorial*, The Hong Kong University of Science and Technology, URL http://www.math.ust.hk/~mawang/teaching/math532/math532-a.html, 2011

[9] C. C. Douglas and M. B. Douglas, *MGNet Bibliography*, URL http://www.mgnet.org/bib/mgnet.bib, Yale University, Department of Computer Science, New Haven, CT (USA), 1991-2002

[10] Message Passing Interfac Forum, *MPI: A Message-Passing Interface Standard*, URL http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, 2009

[11] OpenMP Architecture Review Board, *OpenMP Application Program Interface*, URL http://www.openmp.org/mp-documents/spec25.pdf, 2005

[12] The Linux Cross Reference, *Linux Kernel Coding Style*, URL http://lxr.linux.no/linux+v2.6.37/Documentation/CodingStyle, 2007