ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΗΧΑΝΙΚΗ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του **Οικονόμου Μάριου**

Συμπλήρωμα Schur σε παράλληλες αρχιτεκτονικές πολλαπλών GPU/CPU πυρήνων. Εφαρμογή στη μέθοδο των πεπερασμένων στοιχείων.

ΕΠΙΒΛΕΠΩΝ

Ανδρέας Γ. Μπουντουβής, Καθηγητής

Σχολή Χημικών Μηχανικών ΕΜΠ

ΑΘΗΝΑ 2014

Θα ήθελα να ευχαριστήσω τον Κύριο Ανδρέα Μπουντουβή και τον Κύριο Αντώνη Σπυρόπουλο που χωρίς την συμβολή τους, θα ήταν αδύνατη η ολοκλήρωση αυτής της εργασίας. Θα ήθελα επίσης να ευχαριστήσω τους γονείς μου, για τη συμπαράσταση τους καθ΄ όλη τη διάρκεια εκπόνησης της.

Περίληψη

Η εργασία αυτή πραγματεύεται την μελέτη μιας παράλληλης εφαρμογής για την επίλυση προβλημάτων πεπερασμένων στοιχείων. Αναλυτικότερα, λύνεται το πρόβλημα ελαστικότητας, μιας τριδιάστατης δοκού. Βασικό χαρακτηριστικό της εφαρμογής είναι η χρήση του συμπληρώματος Schur. Εφαρμόζοντας στο αρχικό χωρίο του προβλήματος, μια προσαρμοσμένη διαμέριση, είναι πολύ εύκολο να δημιουργηθούν μικρότερα και ανεξάρτητα αλγεβρικά συστήματα τα οποία μπορούν να επιλυθούν παράλληλα.

Η εφαρμογή αυτή μπορεί να εκμεταλλευτεί τις δυνατότητες των σύγχρονων υβριδικών παράλληλων συστημάτων χρησιμοποιώντας το πρωτόκολλο MPI για την ανταλλαγή δεδομένων. Μπορεί επίσης να αξιοποιήσει τις ανεπτυγμένες δυνατότητες των GPU ώστε να επιτύχει πολύ καλύτερους χρόνους εκτέλεσης σε σχέση με τα συστήματα που χρησιμοποιούν μόνο CPUs.

Παρουσιάζονται μετρήσεις χρόνων και επιτάχυνσης της εφαρμογής για διάφορα σενάρια, από 2 επεξεργαστών μέχρι και συνδυασμό 2 GPUs μαζί με 8 CPUs. Αναλύεται η απόδοση του παράλληλου μοντέλου και αποτυπώνονται οι απαιτήσεις μιας διαμέρισης που οδηγεί σε επιτάχυνση της επίλυσης.

Schur Complement on multiple CPU/GPU parallel computer architectures. Application in the Finite Element Method.

Oikonomou Marios

Abstract

In this thesis is examined a parallel computer application used for the solution of finite element problems. Specifically, the problem that is solved is that of a 3D elastic beam. A key feature of the application is the usage of the Schur complement. A special partition, if applied to the initial domain, facilitates the formation of small and independent linear systems of equations which can be solved concurrently.

The application takes advantage of modern hybrid parallel computers using the MPI protocol for data exchange. Furthermore the Graphic processing unit (GPU) is utilized to increase performance compared to CPU only architectures.

The study presents the acceleration and performance of the application for different configurations, using two CPUs to eight CPUs combined with two GPUs. The efficiency of the parallel model is analyzed and the requirements of a partition, that accelerates the solution process, are established.

Περιεχόμενα

1 Εισαγωγή

Η παράλληλη επεξεργασία έχει ανοίξει το δρόμο στη επίλυση δύσκολων προβλημάτων, σε χρόνους που οι σειριακές μέθοδοι, αδυνατούν να επιτύχουν. Σε αρκετά προβλήματα διαφορικών εξισώσεων ή προβλήματα μηχανικής, όπου οι βαθμοί ελευθερίας είναι της τάξεως των χιλιάδων ή εκατομμυρίων, επιτυγχάνεται σημαντική βελτίωση στους χρόνους εκτέλεσης.

Ένας ακόμα λόγος που συντελεί στην επιλογή της παράλληλης επεξεργασίας είναι η εξέλιξη των επεξεργαστών. Πολλοί σχεδιαστές επεξεργαστών θεωρούν πως τα επόμενα χρόνια θα είναι αρκετά δύσκολο να αυξηθεί η ταχύτητα των σειριακών επεξεργαστών (αρχιτεκτονικής Single Instruction Single Data). Η βιομηχανία των υπολογιστών, λαμβάνοντας υπόψη το παραπάνω, έχει προχωρήσει στην ανάπτυξη πολυπύρηνων επεξεργαστών αλλά και τη χρήση των καρτών γραφικών (GPU) ως υπολογιστικούς επεξεργαστές.

Συνεπώς, η δημιουργία παράλληλων αλγορίθμων έχει γίνει αντικείμενο μελέτης. Υπάρχουν δύο προσεγγίσεις σε αυτό. Η μία είναι να γίνει προσπάθεια παραλληλοποίησης του σειριακού αλγορίθμου σε σημεία όπου επιτρέπεται, π.χ. αλγόριθμοι που χρησιμοποιούν απλές πράξεις γραμμικής άλγεβρας. Η άλλη περίπτωση είναι η δημιουργία ενός νέου αλγορίθμου όπου θα λειτουργεί εξαρχής με παράλληλη "λογική".

Μια τέτοια περίπτωση είναι και η μέθοδος συμπληρώματος Schur (Schur complement). Η βασική της ιδέα είναι να χωριστεί το αρχικό διακριτοποιημένο χωρίο σε υποχωρία και κατόπιν, να επιλυθούν οι αντίστοιχες εξισώσεις σε ξεχωριστούς επεξεργαστές. Σε αυτή την εργασία, θα επιλυθεί το πρόβλημα τρισδιάστατης ελαστικότητας με χρήση πεπερασμένων στοιχείων και του συμπληρώματος Schur. Θα γίνει μελέτη της επίδοσης της επίλυσης σε ένα σύστημα υβριδικού υπολογιστή που συνδυάζει κατανεμημένη αλλά και κοινή μνήμη μεταξύ των επεξεργαστών. Επιπρόσθετα, θα χρησιμοποιηθούν και κάρτες γραφικών στην επίλυση του προβλήματος.

2 Το συμπλήρωμα Schur

2.1 Ορισμός των υποχωρίων

Έστω ένα διακριτοποιημένο ορθογωνικό χωρίο Ω, στο οποίο είναι για επίλυση η εξίσωση :

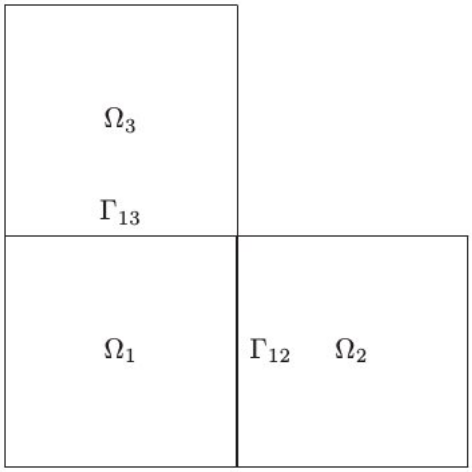
$$\triangle u = f \text{ στο } \Omega \tag{2.1}$$

$$u = u_\Gamma \text{ στο } \Gamma \tag{2.2}$$

Ο τελεστής $\triangle$ ορίζεται ως:

$$\triangle = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} \tag{2.3}$$

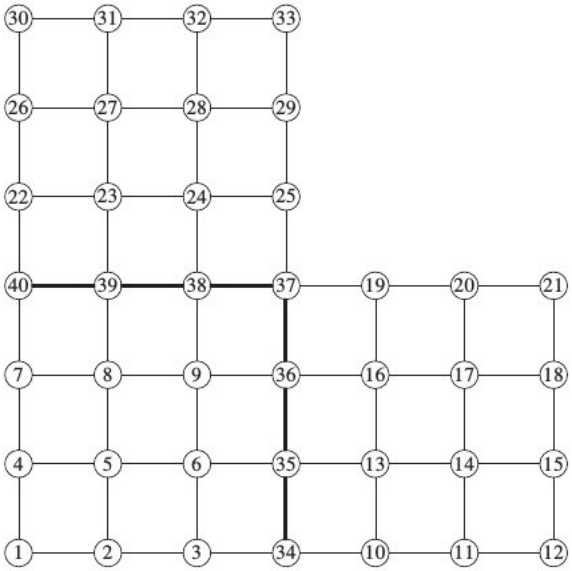
Χωρίζοντας το χωρίο σε s υποσύνολα δημιουργούνται τα σύνορα Γ_{13} και Γ_{12} (σχήμα 2.1).



Σχήμα 2.1: Χωρίο χωρισμένο σε τρία υποχωρία Saad [18]

Στην περίπτωση που ήταν γνωστές οι τιμές στα σύνορα Γ_{13} και Γ_{12}, θα μπορούσαν να χρησιμοποιηθούν σαν συνθήκες Dirichlet και έτσι να προκύψουν τρεις απεπλεγμένες εξισώσεις Poisson.

Επιλέγοντας να λυθεί το παραπάνω πρόβλημα με διακριτοποίηση πεπερασμένων στοιχείων, προκύπτει το πλέγμα του σχήματος 2.2. Κρίνεται αναγκαίο, κάθε πεπερασμένο στοιχείο να ανήκει σε ένα μόνο υποχωρίο. Επίσης πρέπει να αριθμηθούν πρώτα οι εσωτερικοί κόμβοι των χωρίων κι ύστερα οι κόμβοι του συνόρου.



Σχήμα 2.2: Αρίθμηση κόμβων [18]

Ανάμεσα στα υποχωρία, υπάρχει μια σειρά κόμβων (34-40) που αντιστοιχούν στα σύνορα Γ_{13} και Γ_{12}. Για να επιτευχθεί λύση στο αρχικό χωρίο, είναι απαραίτητο να βρεθεί η λύση στο σύνορο.

Το πρόβλημα των πεπερασμένων στοιχείων διατυπώνεται ως εξής: Να βρεθεί $u \in V_h$ τέτοιο ώστε $\alpha(u, v) = (f, v) \forall v \in V_h$ όπου το V_h είναι ο χώρος των συναρτήσεων σχήματος και το α είναι διγραμμικός τελεστής που ορίζεται ως:

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} dx$$

Εφόσον τα υποχωρία μεταξύ τους δεν είναι επικαλυπτόμενα, τότε

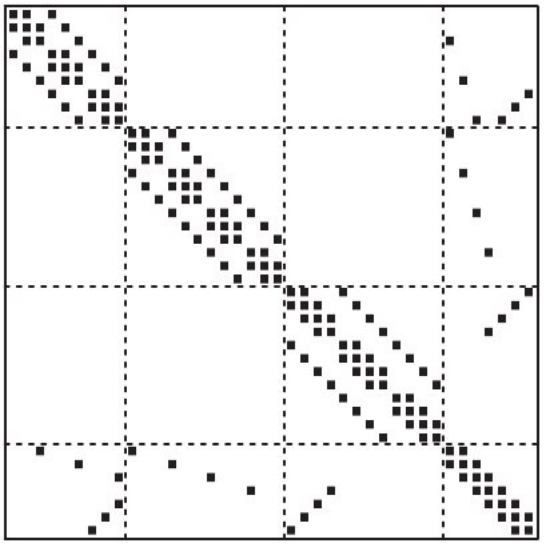
$$a(u, v) = \sum_{i=1}^{s} a_i(u, v)$$

με

$$a_i(u, v) = \int_{\Omega_i} \nabla u \cdot \nabla v dx$$

Ο παραπάνω τρόπος γραφής των εξισώσεων είναι ακριβώς ίδιος με τον τρόπο που χωρίζονται τα στοιχεία μεταξύ τους και δημιουργείται το μητρώο ακαμψίας.

Από την διακριτοποίηση προκύπτει το σύστημα $Au = b$ όπου το A είναι ένα μητρώο με την μορφή του σχήματος (2.3):



Σχήμα 2.3: Ενδεικτικός πίνακας που προκύπτει από διαμέριση του χωρίου για το πρόβλημα Poisson [18]

Διαπιστώνεται ότι ο πίνακας έχει χωριστεί σε διάφορα blocks κάποια από τα οποία έχουν στοιχεία, ενώ κάποια άλλα δεν έχουν. Στη συνέχεια της εργασίας, τα blocks αυτά θα έχουν τα εξής ονόματα:

$$\begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_S & E_S \\ F_1 & F_2 & \cdots & F_s & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{pmatrix} \tag{2.4}$$

όπου κάθε x_i είναι το υποδιάνυσμα των αγνώστων u που βρίσκονται εντός του χωρίου i, και το y είναι το υποδιάνυσμα των αγνώστων u του συνόρου. Αντίστοιχα, το κάθε f_i είναι το υποδιάνυσμα του b που αντιστοιχεί στο χωρίο i, ενώ το g είναι το υποδιάνυσμα του b που αντιστοιχεί στο σύνορο. Το σύστημα μπορεί να διατυπωθεί και στην εξής μορφή:

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \text{με } A = \begin{pmatrix} B & E \\ F & C \end{pmatrix} \tag{2.5}$$

2.2 Ορισμός του συστήματος Schur

Έχοντας το σύστημα της μορφής (2.5) και κάνοντας την υπόθεση ότι το μητρώο B αντιστρέφεται, υπολογίζεται το x :

$$x = B^{-1}(f - Ey) \tag{2.6}$$

Αντικαθιστώντας στην δεύτερη εξίσωση :

$$(C - FB^{-1}E)y = g - FB^{-1}f \tag{2.7}$$

Ο πίνακας

$$S = C - FB^{-1}E \tag{2.8}$$

ονομάζεται συμπλήρωμα Schur. Αν μπορεί να λυθεί το σύστημα 2.7, τότε υπολογίζονται οι άγνωστοι που βρίσκονται πάνω στο σύνορο. Αντικαθιστώντας στην εξίσωση (2.6), υπολογίζονται όλοι οι άγνωστοι εσωτερικά των χωρίων. Το γεγονός ότι το μητρώο B έχει αυτή τη μορφή, προσφέρει τη δυνατότητα να επιλυθούν τα συστήματα του κάθε χωρίου ανεξάρτητα το ένα από το άλλο. Πρακτικά αυτό σημαίνει ότι τα συστήματα $B_i x_i = f_i$:

- μπορούν να λυθούν παράλληλα σε διαφορετικούς επεξεργαστές ταυτόχρονα.
- μπορούν να λυθούν μέχρι και σε υπολογιστές που δεν έχουν επαρκή μνήμη για να λύσουν περισσότερα του ενός, συστήματα.

Ένας γενικευμένος αλγόριθμος για την επίλυση των Schur συστημάτων θα μπορούσε να είναι:

1. Υπολόγισε το δεξί μέλος $g' = g - FB^{-1}f$
2. Λύσε το σύστημα $Sy = g'$
3. Γνωρίζοντας πλέον το y, λύσε το σύστημα $x = B^{-1}(f - Ey)$

Η επίλυση του συστήματος (2.7) μπορεί να γίνει με δύο τρόπους όπου θα αναλυθούν στις επόμενες παραγράφους

2.2.1 Σχηματισμός του μητρώου S

Ο πρώτος τρόπος είναι ο ρητός σχηματισμός του μητρώου S. Ορίζοντας το μητρώο $E' = B^{-1}E$ και το διάνυσμα $f' = B^{-1}f$ ο αλγόριθμος για την επίλυση του Schur συστήματος, γίνεται[18]:

1. Λύσε τα $BE' = E$ και $B'f = f$
2. Υπολόγισε το $g' = g - Ff'$
3. Σχημάτισε το $S = C - FE'$
4. Λύσε το $Sy = g'$
5. Υπολόγισε το $x = f' - E'y$

Σε αυτή την περίπτωση, η επίλυση των επιμέρους συστημάτων B_i μπορεί να γίνει είτε με επαναληπτική μέθοδο ή με LU παραγοντοποίηση [18]. Όταν το σύνορο μεταξύ των χωρίων είναι μεγάλο, τότε θα επιλυθούν αρκετές φορές συστήματα με το μητρώο B_i. Οπότε θα είναι προτιμότερο να γίνει παραγοντοποίηση στην αρχή ώστε να γίνει γρήγορα η επίλυση των υπόλοιπων συστημάτων. Παρόλα αυτά τίποτα δεν προσφέρεται χωρίς το ανάλογο κόστος. Στα περισσότερα προβλήματα μηχανικής, το μητρώο του συστήματος $Ax = b$ είναι αραιό και μπορεί να αποθηκευτεί, με οικονομικές σε μνήμη μεθόδους. Το αντίστοιχο συμβαίνει και με τα μητρώα B_iαφού είναι υποσύνολα του ίδιου προβλήματος. Κάθε απόπειρα παραγοντοποίησης αυτών των μητρώων, οδηγεί σε προσθήκη πολλών νέων στοιχείων με αποτέλεσμα να αυξάνονται κατά πολύ οι απαιτήσεις σε μνήμη.

Μια σημαντική παρατήρηση είναι ότι η κατασκευή του μητρώου S είναι αρκετά απαιτητική σε μνήμη με αποτέλεσμα να χάνεται το πλεονέκτημα επίλυσης μεγάλων συστημάτων.

2.2.2 Επίλυση του συστήματος με μεθόδους Krylov

Οι επαναληπτικές μέθοδοι που βασίζονται σε υποχώρους Krylov απαιτούν τον πολλαπλασιασμό του μητρώου A με ένα διάνυσμα βάσης q. Υπάρχει τρόπος να πολλαπλασιαστεί ο πίνακας S με οποιοδήποτε διάνυσμα, εμμέσως, δηλαδή χωρίς να σχηματιστεί:

$$Sq = (C - FB^{-1}E)q = Cq - FB^{-1}Eq \tag{2.9}$$

Ορίζοντας τα διανύσματα $q' = Eq$, $z = B^{-1}q'$ και $w = Cq + Fz$ διαπιστώνεται ότι $w = Sq$.
Τα διανύσματα αυτά μπορούν να χωριστούν σε blocks, δηλαδή:

$$q'_i = E_i q \tag{2.10}$$

$$B_i z_i = q'_i \tag{2.11}$$

$$w = Cq + \sum_{\forall i \in \Omega} F_i z_i \tag{2.12}$$

Με αυτόν τον τρόπο, τα συστήματα της σχέσης (2.11) μπορούν να επιλυθούν παράλληλα. Μέσω αυτής της υπολογιστικής διαδικασίας, μπορεί να υπολογιστεί το διάνυσμα y. Παρότι το γινόμενο Sq δεν παρουσιάζει ιδιαίτερες δυσκολίες, η χρήση προσταθεροποιητή (preconditioner) στο μητρώο S είναι αρκετά πιο δύσκολη ([18]).

2.3 Ιδιότητες του συμπληρώματος Schur

Δίνεται το μητρώο A όπου δεν είναι ιδιάζον και χωρισμένο όπως στη σχέση (2.4) και ο υποπίνακας B που και αυτός δεν είναι ιδιάζων. Ορίζεται ο τελεστής επιλογής συνόρου ως: $R_y \begin{matrix} x \\ y \end{matrix} = y$ με τις εξής ιδιότητες[18]:

1. Ο πίνακας του συμπληρώματος Schur S δεν είναι ιδιάζων
2. Αν ο A είναι θετικά ορισμένος, τότε είναι και ο S

3. Για κάθε y, $S^{-1}y = R_y A^{-1} \begin{matrix} 0 \\ y \end{matrix}$

Η ιδιότητα (2) είναι αρκετά σημαντική καθώς αν το αρχικό πρόβλημα είναι θετικά ορισμένο, τότε για τον υπολογισμό του συνόρου μπορεί να χρησιμοποιηθεί η ίδια επαναληπτική μέθοδος που θα χρησιμοποιούνταν για το αρχικό πρόβλημα.

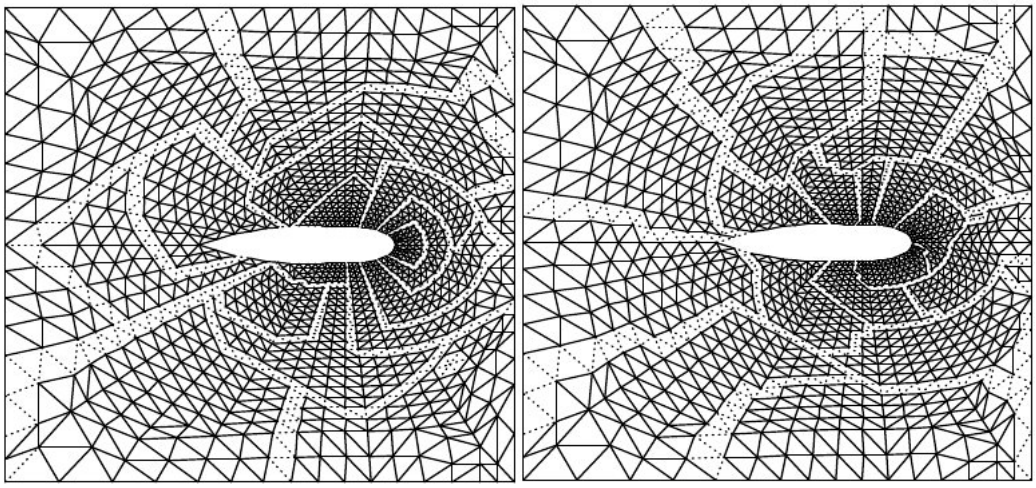
2.4 Διαμέριση του αρχικού χωρίου

Από τα αρχικά στάδια εφαρμογής της μεθόδου Schur, ένα από τα ζητήματα που καλείται κανείς να αντιμετωπίσει είναι της διαμέρισης του χωρίου. Υπάρχουν διάφορα κριτήρια που καθορίζουν αν μια διαμέριση είναι καλή ή κακή:

1. Είναι αναγκαίο τα σύνορα μεταξύ των χωρίων να αποτελούνται από λίγους κόμβους. Όσο πιο μεγάλο είναι το όριο, τόσο πιο πολύ αυξάνεται το πλήθος των συστημάτων που πρέπει να επιλυθούν, είτε σχηματιστεί το μητρώο S είτε όχι.

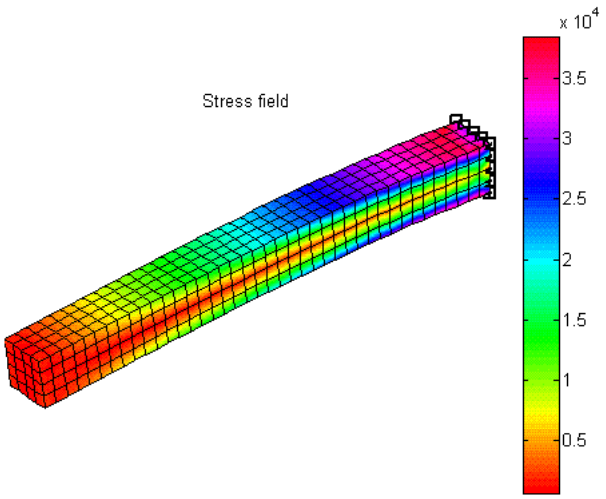
2. Τα μεγέθη των χωρίων πρέπει να είναι τέτοια ώστε να ευνοούν την κατανομή φορτίου μεταξύ των επεξεργαστών. Σε οποιαδήποτε άλλη περίπτωση, ο ένας επεξεργαστής θα περιμένει τον άλλον.

Σε πραγματικά προβλήματα, ο χωρισμός γίνεται είτε από τον προγραμματιστή, είτε χρησιμοποιούνται άλλες τεχνικές, όπως γεωμετρικές, χωρικές ή θεωρίας γράφων.



Σχήμα 2.4: Δύο διαφορετικές διαμερίσεις για το πρόβλημα της αεροτομής[18]

3 Το πρόβλημα ελαστικότητας του τριδιάστατου στερεού



Σχήμα 3.1: Τρισδιάστατη πακτωμένη δοκός [5]

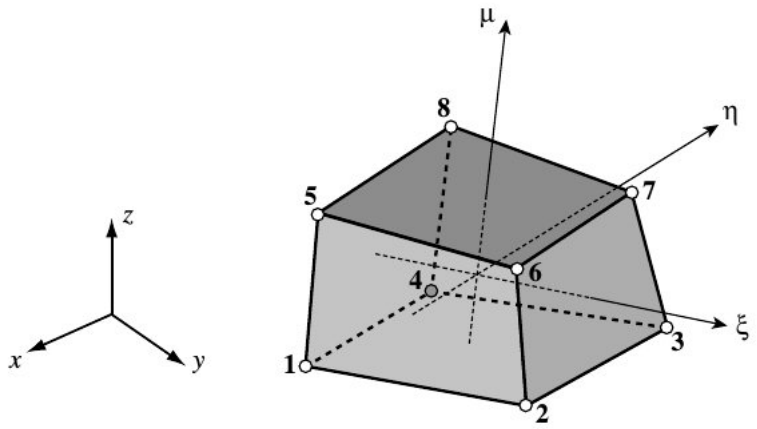
Για την επίλυση του προβλήματος του τριδιάστατου στερεού, χρησιμοποιήθηκαν οχτακομβικά τρισδιάστατα στοιχεία. Κάθε ένα απ΄ αυτά τα στοιχεία έχει διατυπωθεί σε φυσικές συντεταγμένες. Το πολυώνυμο παρεμβολής που χρησιμοποιούν τα στοιχεία αυτά είναι $u(x, y, z) = a_1 + a_2x + a_3y + a_4z + a_5xy + a_6yz + a_7zx + a_8xyz$.

Αν ορίσουμε τις συντεταγμένες κάθε σημείου του στοιχείου ως

$$\alpha_i = \begin{Bmatrix} \xi_i \\ \eta_i \\ \mu_i \end{Bmatrix}$$

μπορεί να γραφεί το διάνυσμα μετατοπίσεων ως

$$u = Na^e \tag{3.1}$$



Σχήμα 3.2: Οχτακομβικό στοιχείο σε φυσικές συνεταγμένες[1, :4]

Οι συντεταγμένες των ξ, η, μ μπορούν να διατυπωθούν ως:

- ξπαίρνει την τιμή -1 στην έδρα 1485 μέχρι την $+1$ 2376
- ηπαίρνει την τιμή -1 στην έδρα 1265 μέχρι την $+1$ 3487
- μπαίρνει την τιμή -1 στην έδρα 1234 μέχρι την $+1$ 5678

Οι συναρτήσεις σχήματος του κάθε στοιχείου είναι:

$$\begin{array}{ll} N_1^e = \frac{1}{8}(1-\xi)(1-\eta)(1-\mu) & N_2^e = \frac{1}{8}(1+\xi)(1-\eta)(1-\mu) \\ N_3^e = \frac{1}{8}(1+\xi)(1+\eta)(1-\mu) & N_4^e = \frac{1}{8}(1-\xi)(1+\eta)(1-\mu) \\ N_5^e = \frac{1}{8}(1-\xi)(1-\eta)(1+\mu) & N_6^e = \frac{1}{8}(1+\xi)(1-\eta)(1+\mu) \\ N_7^e = \frac{1}{8}(1+\xi)(1+\eta)(1+\mu) & N_2^e = \frac{1}{8}(1-\xi)(1+\eta)(1+\mu) \end{array} \tag{3.2}$$

Το μητρώο των τροπών (strains) στον τριδιάστατο χώρο είναι

$$\epsilon = \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \end{Bmatrix} = \mathbf{S}u \tag{3.3}$$

Αντικαθιστώντας την (3.1) προκύπτει:

$$\epsilon = \mathbf{SN}\alpha^e = \mathbf{B}\alpha^e \tag{3.4}$$

όπου

$$\mathbf{B_i} = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \end{bmatrix} \tag{3.5}$$

Όταν το υλικό είναι γραμμικό και οι μετατοπίσεις μικρές:

$$\sigma = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix} = \mathbf{D}\epsilon \tag{3.6}$$

Επίσης, όταν το υλικό είναι ισότροπο, ο τανυστής D ορίζεται ως:

$$\mathbf{D} = \frac{\mathbf{E}}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ & 1-\nu & \nu & 0 & 0 & 0 \\ & & 1-\nu & 0 & 0 & 0 \\ & & & \frac{1-2\nu}{2} & 0 & 0 \\ & \text{sym.} & & & \frac{1-2\nu}{2} & 0 \\ & & & & & \frac{1-2\nu}{2} \end{bmatrix} \tag{3.7}$$

Άρα το μητρώο ακαμψίας που προκύπτει από την αρχή των δυνατών έργων είναι:

$$\mathbf{K^e} = \int_{V^e} \mathbf{B^T E B} dV^e \tag{3.8}$$

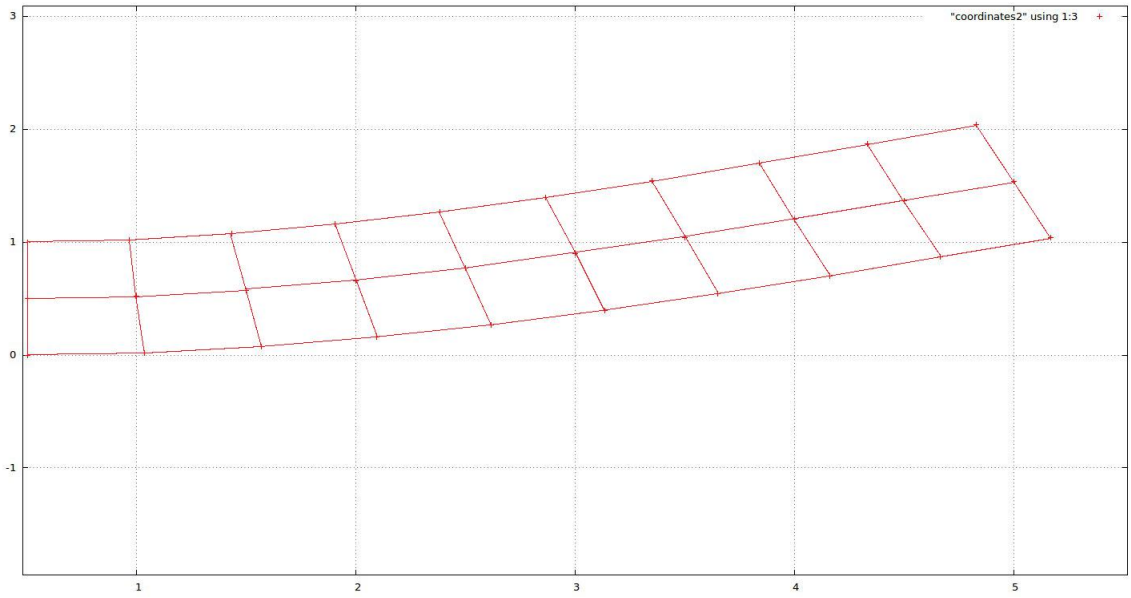
Αφού τα στοιχεία είναι σε φυσικές συντεταγμένες, ορίζεται η Ιακωβιανή του μετασχηματισμού ως:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{pmatrix} \tag{3.9}$$

και το μητρώο δυσκαμψίας ως

$$\mathbf{K^e} = \iiint_{-1}^{+1} \mathbf{B^T E B} det(\mathbf{J}) d\xi d\eta d\mu \tag{3.10}$$

Η δημιουργία του τοπικού μητρώου έγινε στο πρόγραμμα Mathematica και εισήχθη στον κώδικα των πεπερασμένων στοιχείων ως αρχείο κειμένου. Η γεωμετρία του στερεού είναι όμοια με της δοκού. Η δοκός επιλέχθηκε γιατί είναι εύκολο να χωριστεί σε υποχωρία δημιουργώντας εγκάρσιες τομές. Στο σχήμα (3.3) φαίνονται τα αποτελέσματα των μετατοπίσεων ενώ η δοκός είναι σε κατακόρυφη κάμψη.



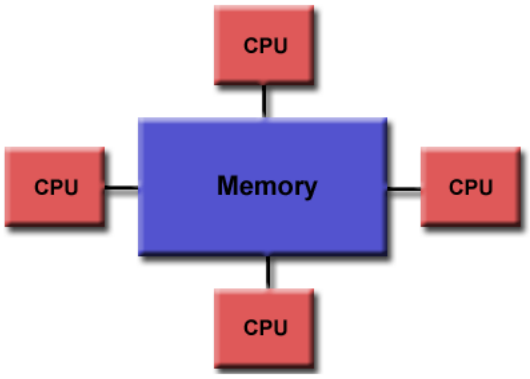
Σχήμα 3.3: Μετατοπίσεις δοκού - Αποτελέσματα από τον κώδικα πεπερασμένων στοιχείων

4 Παράλληλη επεξεργασία

4.1 Αρχιτεκτονικές μνήμης παράλληλων υπολογιστών

Υπάρχουν τρία είδη αρχιτεκτονικής:

Κοινής μνήμης

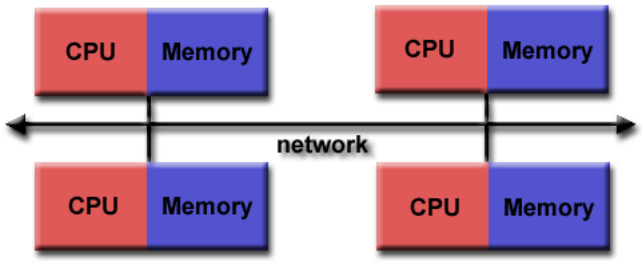


Σχήμα 4.1: Σύστημα κοινής μνήμης

Το σύστημα έχει παραπάνω από έναν επεξεργαστές οι οποίοι μπορούν να προσπελάσουν την ίδια (ομοιογενή - UMA) μνήμη. Οι επεξεργαστές μπορούν να εργάζονται ανεξάρτητα ο ένας από τον άλλο, αλλά οποιαδήποτε αλλαγή κάνει κάποιος στη μνήμη, είναι ορατή σε όλους τους επεξεργαστές. Τα πλεονεκτήματα αυτής της αρχιτεκτονικής είναι ότι παρέχεται ένα εύκολο περιβάλλον για προγραμματισμό αλλά και ότι η μεταφορά δεδομένων από επεξεργαστή σε επεξεργαστή είναι ταχύτατη, λόγω των γρήγορων δίαυλων. Από την άλλη πλευρά, τα μειονεκτήματα της μεθόδου είναι ότι δεν επεκτείνεται εύκολα σε περισσότερους επεξεργαστές καθώς οι προσπελάσεις της μνήμης μπορεί να αυξηθούν μέχρι και σε γεωμετρικό βαθμό.

Τα τελευταία χρόνια αυτά τα συστήματα έχουν γίνει αρκετά κοινά στην αγορά καθώς έχουν δημιουργηθεί chip που έχουν περισσότερους από έναν επεξεργαστές (πολυπύρηνα). Επίσης, έχουν εμφανιστεί και αρχιτεκτονικές μη ομοιογενούς μνήμης (NUMA). Σε αυτές τις αρχιτεκτονικές, οι επεξεργαστές και η μνήμη χωρίζονται σε ομάδες. Κάθε ομάδα επεξεργαστών έχει πάλι πρόσβαση σε όλες της ομάδες μνήμης, με τη διαφορά ότι η προσπέλαση στη δική της ομάδα είναι ταχύτερη.

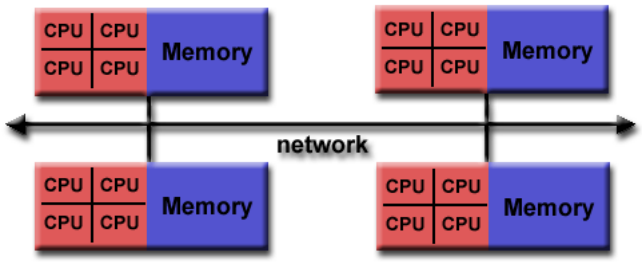
Κατανεμημένης μνήμης



Σχήμα 4.2: Σύστημα μη κατανεμημένης μνήμης

Σε αυτό σύστημα κάθε ομάδα επεξεργαστή-μνήμης ονομάζεται κόμβος. Οι κόμβοι μπορεί να είναι πολύ διαφορετικοί μεταξύ τους. Κάθε κόμβος έχει την δική του τοπική μνήμη και έχει πρόσβαση μόνο σε αυτή. Με αυτόν τον τρόπο κάθε επεξεργαστής λειτουργεί ανεξάρτητα και κάθε αλλαγή στην μνήμη του, δεν είναι ορατή στους υπόλοιπους. Η ανταλλαγή δεδομένων μεταξύ των κόμβων, είναι ευθύνη του προγραμματιστή, ο οποίος πρέπει να ορίσει ρητά με ποιόν τρόπο και πότε πρέπει να γίνει αυτή η επικοινωνία. Ένα από τα πλεονεκτήματα της αρχιτεκτονικής αυτής είναι ότι είναι επεκτάσιμη αφού αυξάνοντας τον αριθμό των επεξεργαστών, αυξάνεται αυτομάτως και ο χώρος της μνήμης. Επιπρόσθετα, κάθε επεξεργαστής μπορεί να προσπελάσει την δική του μνήμη με μεγάλη ταχύτητα και χωρίς να επηρεάζει την ταχύτητα προσπέλασης των υπολοίπων. Ένα από τα σημαντικότερα μειονεκτήματα της μεθόδου, είναι ότι ο προγραμματιστής είναι υπεύθυνος για αρκετές λεπτομέρειες της επικοινωνίας των επεξεργαστών, κάτι που δεν αποτελεί ζήτημα στις αρχιτεκτονικές κοινής μνήμης. Τέλος, η ταχύτητα ανταλλαγής δεδομένων μεταξύ των επεξεργαστών εξαρτάται από την ταχύτητα του δικτύου. Σχεδόν σε όλες τις περιπτώσεις, τα δίκτυα είναι πολύ πιο αργά από τους διαύλους επικοινωνίας επεξεργαστή-μνήμης με αποτέλεσμα οι αρχιτεκτονικές κοινής μνήμης να είναι ταχύτερες σε αυτό το κομμάτι.

Υβριδικά συστήματα



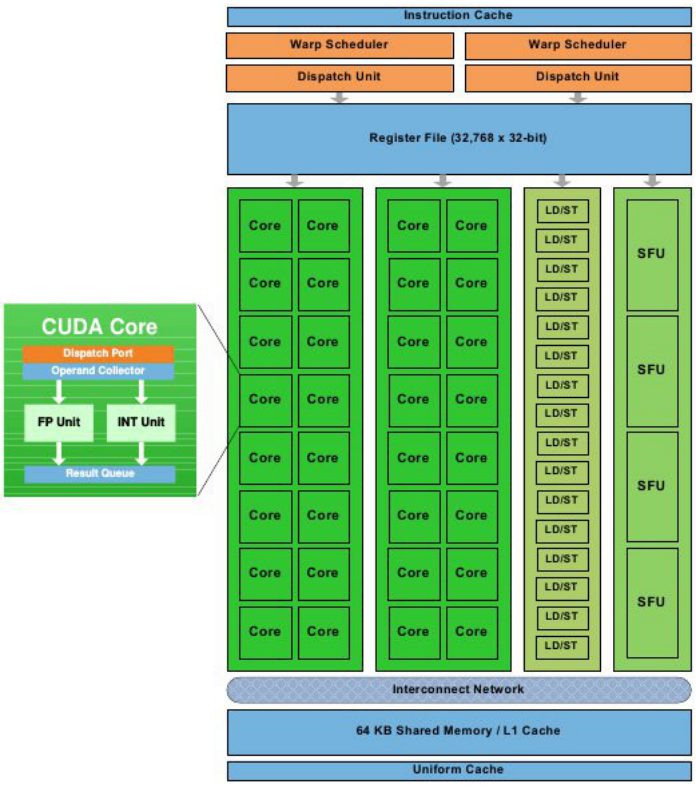
Σχήμα 4.3: Υβριδικά συστήματα

Τα πιο γρήγορα συστήματα υπολογιστών σήμερα, έχουν αυτήν την αρχιτεκτονική μνήμης. Αποτελούνται από κόμβους που ο κάθε ένας έχει περισσότερους από έναν επεξεργαστές με κοινή μνήμη.

4.2 Υπολογισμοί σε κάρτες γραφικών

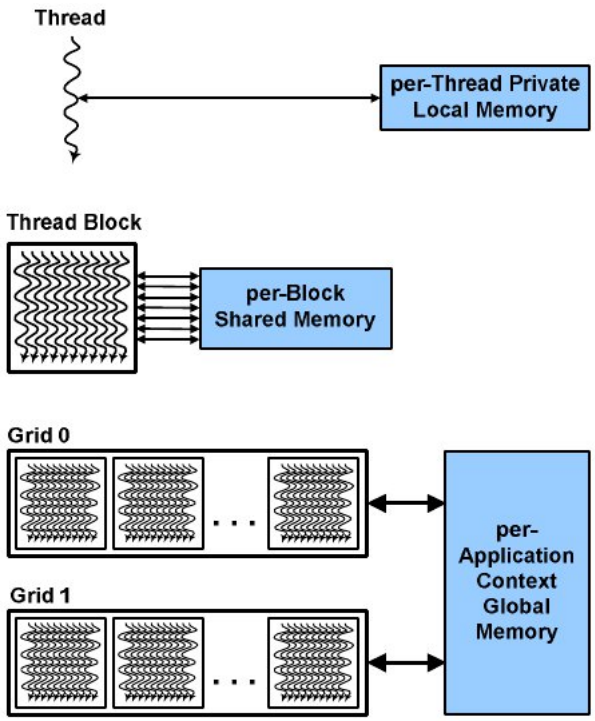
Από το έτος 2003 [2] οι κάρτες γραφικών άρχισαν να χρησιμοποιούνται για επεξεργασία γενικών και αριθμητικών δεδομένων, όπως ανάλυση πρωτεϊνών, ανάλυση ρίσκου ή ερωτήματα σε μεγάλες βάσεις δεδομένων. Οι κάρτες γραφικών είναι κατασκευασμένες με τέτοιο τρόπο ώστε να μπορούν να κάνουν πράξεις υπολογιστικής γεωμετρίας σε τέτοια ταχύτητα που να μπορούν να παρουσιάσουν σε πραγματικό χρόνο, τριδιάστατα γραφικά. Οπότε, κατά κάποιο τρόπο, θα μπορούσαν να χαρακτηριστούν ως διανυσματικοί επεξεργαστές (οι διανυσματικοί επεξεργαστές μπορούν σε έναν "κύκλο ρολογιού", να κάνουν πράξεις μεταξύ διανυσμάτων). Σημειώνεται ότι οι κάρτες γραφικών έχουν τη δική τους μνήμη, η οποία είναι συνήθως πιο γρήγορη από αυτή των επεξεργαστών.

Μία από τις πιο διαδεδομένες τεχνολογίες χρήσης καρτών γραφικών για γενικούς υπολογισμούς, είναι η CUDA της NVIDIA. Θα ακολουθήσει μια γρήγορη επεξήγηση της αρχιτεκτονικής FERMI η οποία υλοποιεί την CUDA. Κάθε κάρτα γραφικών έχει μια σειρά από Stream Multiprocessors. Κάθε Multiprocessor έχει μια σειρά από Stream Processors ή Cuda cores, όπου κάθε ένας μπορεί να εκτελεί, μία πράξη τη φορά, σε ένα βαθμωτό μέγεθος (θερμοκρασία για παράδειγμα) είτε είναι ακέραιος, είτε αριθμός κινητής υποδιαστολής. Στην αρχιτεκτονική FERMI, συνήθως κάθε Multiprocessor έχει 32 Cuda cores. Όλοι οι πυρήνες ενός Stream Multiprocessor δέχονται μια κοινή εντολή και την εκτελούν την ίδια χρονική στιγμή. Για παράδειγμα, μπορούν να προσθέσουν δύο διανύσματα με 32 στοιχεία το κάθε ένα, σε έναν κύκλο ρολογιού.



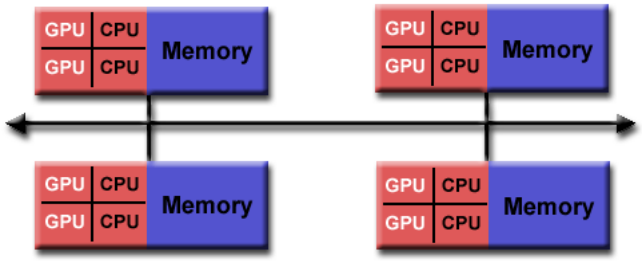
Σχήμα 4.4: Σχεδιάγραμμα ενός Stream Multiprocessor

Κάθε φορά που χρειάζεται να γίνει ένας υπολογισμός στην κάρτα γραφικών, πρέπει να μεταφερθούν τα δεδομένα, από την κύρια μνήμη, στην μνήμη της κάρτας γραφικών. Το αντίστροφο πρέπει να συμβεί όταν έχει τελειώσει ο υπολογισμός και γίνεται συλλογή των αποτελεσμάτων. Ο κώδικας που εκτελείται στην κάρτα γραφικών ονομάζεται kernel. Κάθε kernel αντιστοιχίζεται σε ένα grid όπου αυτό περιέχει blocks από νήματα. Κάθε block προγραμματίζεται να εκτελεστεί σε έναν και μοναδικό Multiprocessor. Αυτό σημαίνει ότι όλα τα νήματα σε αυτό το block έχουν τον ίδιο κώδικα εκτέλεσης. Ο Multiprocessor με τη σειρά του, ομαδοποιεί τα νήματα του block σε ομάδες των 32 που ονομάζονται wraps. Με αυτόν τον τρόπο, εκτελούνται και τα 32 νήματα του κάθε wrap ταυτόχρονα.



Σχήμα 4.5: Το μοντέλο προγραμματισμού του CUDA

Μετά την εμφάνιση αυτής της τεχνολογίας, πολλά από τα παράλληλα συστήματα υπολογιστών, έχουν αρχίσει να την ενσωματώνουν. Έτσι δημιουργείται μια νέα κατηγορία υβριδικών συστημάτων η οποία φαίνεται στο σχήμα 4.6.



Σχήμα 4.6: Υβριδικά συστήματα με Επεξεργαστές (CPU) και κάρτες γραφικών (GPU)

5 Επιλογή μοντέλου παράλληλης επεξεργασίας

5.1 Επιλογή αρχιτεκτονικής για το συμπλήρωμα Schur

Τα σύγχρονα προβλήματα μηχανικής, ανάγονται σε προβλήματα εξισώσεων με πολλές χιλιάδες βαθμούς ελευθερίας. Με τη μέθοδο Schur complement επιδιώκεται:

- Η επίλυση ενός προβλήματος σε παράλληλους υπολογιστές
- Η επίλυση ενός προβλήματος το οποίο είναι τόσο μεγάλο που η μνήμη ενός υπολογιστή δεν επαρκεί.

Γι΄ αυτούς τους λόγους επιλέχθηκε να αναπτυχθεί κώδικας ο οποίος δεν θα κάνει ρητό σχηματισμό του μητρώου S και για την επίλυση στο σύνορο θα χρησιμοποιεί μια επαναληπτική μέθοδο Krylov. Η επιλογή της υβριδικής παράλληλης αρχιτεκτονικής, προσφέρει επεκτασιμότητα σε μεγαλύτερα παράλληλα συστήματα, με περισσότερη μνήμη, ώστε να μπορούν να επιλυθούν ακόμα μεγαλύτερα προβλήματα. Δόθηκε ιδιαίτερη προσοχή ώστε να μπορούν να αξιοποιηθούν και κάρτες γραφικών, αν υπάρχουν σε τέτοια συστήματα.

5.2 Message Passing Interface - MPI

Το πρωτόκολλο MPI [7] είναι από τα πιο καθιερωμένα περιβάλλοντα παράλληλου προγραμματισμού. Η πιο βασική του λειτουργία, είναι ότι παρέχει στον προγραμματιστή ένα interface όπου του επιτρέπει να στέλνει πακέτα με δεδομένα από τον έναν επεξεργαστή στον άλλο. Το MPI αποκρύπτει σχεδόν όλες τις λεπτομέρειες που αφορούν το δίκτυο μεταφοράς των δεδομένων μεταξύ των επεξεργαστών.Ίσως μια σημαντική εξαίρεση σε αυτό είναι η τοπολογία του δικτύου (πολλοί παράλληλοι αλγόριθμοι μπορούν να εκμεταλλευτούν καλύτερα συγκεκριμένες τοπολογίες).

Το MPI προσφέρει την δυνατότητα να μεταφερθούν πακέτα δεδομένων, και μεταξύ επεξεργαστών που έχουν κοινή μνήμη. Αυτό το κάνει ιδανικό στη χρήση υβριδικών συστημάτων αφού απλοποιεί αρκετά την υλοποίηση της εφαρμογής και την κάνει αρκετά φορητή ώστε να μπορεί να εκτελεστεί ακόμη και σε ετερογενή συστήματα υπολογιστών.

Εκτός από το MPI υπάρχουν και άλλα περιβάλλοντα ανάπτυξης παράλληλων εφαρμογών. Ένα από τα πιο διαδεδομένα είναι το OpenMP. Βασική διαφορά του OpenMP από το MPI είναι ότι παραλληλοποιεί μια διαδικασία δημιουργώντας νήματα και όχι νέες διεργασίες. Βασικό πλεονέκτημα του είναι ότι τα νήματα δημιουργούνται πολύ πιο γρήγορα από τις διεργασίες αλλά και ότι η ανταλλαγή δεδομένων μεταξύ των νημάτων είναι πιο γρήγορη. Παρ΄ όλα αυτά, το OpenMP μπορεί να λειτουργήσει μόνο σε συστήματα κοινής μνήμης. Εφόσον η εφαρμογή που θα δημιουργηθεί, θα χρησιμοποιηθεί σε υβριδικά συστήματα, θα μπορούσε να γίνει συνδυασμός του MPI και του OpenMP. Λόγω του παράλληλου μοντέλου που επιλέχθηκε (παράγραφος 5.4) κρίθηκε ότι το OpenMP θα προσέφερε ελάχιστα σε ταχύτητα ενώ θα έκανε πολύ πιο δύσκολο τον προγραμματισμό της εφαρμογής. Παρ΄ όλα αυτά, με το τωρινό μοντέλο παραλληλίας, είναι πιθανό οι διαφορετικοί επεξεργαστές να έχουν ένα κοινό υποσύνολο από blocks (εξ. 2.11). Αυτό σημαίνει πως σε ένα κοινής μνήμης σύστημα, έχει σπαταληθεί μνήμη. Το OpenMP θα μπορούσε να βοηθήσει σε αυτήν την περίπτωση ώστε να μην υπάρχουν αντίγραφα των ίδιων block στη μνήμη.

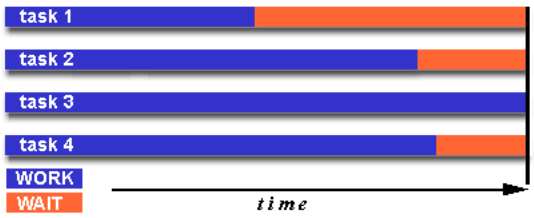
5.3 Το πρόβλημα κατανομής του φορτίου

Ένα από τα ζητήματα που ανακύπτουν στη μέθοδο του Schur Complement είναι η κατανομή του φορτίου μεταξύ των επεξεργαστών. Στη γενική περίπτωση των επαναληπτικών μεθόδων ισχύει $\vec{x}_{i+1} = \vec{x}_i + a\vec{d}$. Για να υπολογιστεί το νέο x_i θα πρέπει σε κάθε επανάληψη να λυθούν όλα τα συστήματα της εξίσωσης (2.11). Είναι λοιπόν φυσικό επόμενο, να μην επιλυθούν όλα τα συστήματα στον ίδιο χρόνο, καθώς μπορεί να έχουν διαφορετικό μέγεθος ή να εκτελούνται σε ετερογενές παράλληλο σύστημα όπου δεν έχουν όλοι οι επεξεργαστές την ίδια ισχύ.

Όταν χρησιμοποιούνται επαναληπτικές μέθοδοι για την επίλυση των συστημάτων (εξ. 2.11), υπάρχει ένας ακόμα λόγος που εμφανίζονται διαφορετικοί χρόνοι επίλυσης και έχει να κάνει με τον αριθμό κατάστασης (condition number). Ο αριθμός κατάστασης ορίζεται ως:

$$\kappa(A) = \|A\|\|A^{-1}\|$$

και δείχνει την ευαισθησία της λύσης για μικρές αλλαγές των τιμών του A. Όταν το condition number είναι μεγάλο, οι επαναληπτικές μέθοδοι εμφανίζουν πιο αργή σύγκλιση και κατά συνέπεια αυξάνεται ο χρόνος επίλυσης του κάθε συστήματος. [8, 16].



Σχήμα 5.1: Στιγμιότυπο παράλληλης επίλυσης όπου δεν υπάρχει κατανομή φορτίου

Ως αποτέλεσμα των διαφορετικών χρόνων επίλυσης των συστημάτων (εξ 2.11), ο χρόνος κάθε επανάληψης επίλυσης του συνόρου (εξ. 2.9) καθορίζεται από τον πιο αργό επεξεργαστή (σχήμα 5.1). Έτσι σχεδόν όλοι οι επεξεργαστές αναμένουν τον τελευταίο, χωρίς να κάνουν κάποιο υπολογισμό.

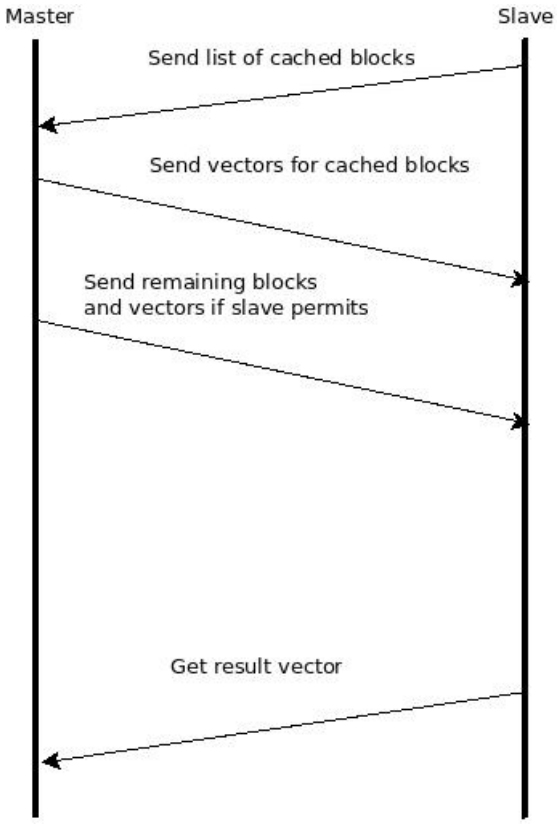
Για να περιοριστεί αυτό το φαινόμενο, είναι αναγκαίο να έχουν οριστεί blocks τα οποία δεν θα αναλάβει κανείς επεξεργαστής από την αρχή. Με αυτόν τον τρόπο, αυτός που έχει ολοκληρώσει την επίλυση των συστημάτων που είχε, θα αναλάβει ένα ή περισσότερα από τα περισσευούμενα. Έτσι, όλοι οι επεξεργαστές θα πραγματοποιούν επίλυση και θα ελαχιστοποιηθεί ο χρόνος αναμονής μεταξύ τους.

5.4 Αναλυτική περιγραφή του μοντέλου

Σε αυτή την υλοποίηση επιλέχθηκε το μοντέλο Master - Slave. Ο Master έχει στη μνήμη του όλα τα blocks και εκτελεί την επαναληπτική διαδικασία Krylov για να υπολογιστούν οι τιμές του συνόρου. Κάθε Slave δέχεται όσα block (εξ. 2.11) μπορεί να χωρέσει η μνήμη του ή όσα του οριστούν. Κάθε φορά που ο Master εκτελεί μια επανάληψη Krylov, στέλνει τα block με τα διανύσματα που τα συνοδεύουν στον πρώτο ελεύθερο slave. Αυτό γίνεται επαναληπτικά μέχρι να εξαντληθούν όλα τα blocks. Σε αυτή τη φάση ο Master αναμένει τα αποτελέσματα από τους Slaves για να μπορέσει να προχωρήσει στην επόμενη επανάληψη. Αυτό ονομάζεται φράγμα (barrier) και είναι το πρόβλημα που προσπαθεί να λύσει η κατανομή φορτίου (load balancing).

Υπάρχουν ακόμα δύο σημαντικά ζητήματα, όσον αφορά τον αλγόριθμο. Για την επίτευξη του ελάχιστου χρόνου, ο αλγόριθμος δημιουργεί στην αρχή μια ταξινομημένη λίστα με τα blocks από το μεγαλύτερο προς το μικρότερο. Έπειτα, ταξινομεί τους slaves από τον πιο γρήγορο, στον πιο αργό. Στη συνέχεια ζητά από τον πρώτο διαθέσιμο slave, να επιλύσει το πρώτο block της λίστας που δεν έχει υπολογιστεί, αν αυτό είναι δυνατόν. Αυτού του είδους η προσέγγιση, έχει σαν αποτέλεσμα την επεξεργασία των πιο μεγάλων block από τους πιο γρήγορους slaves και επίσης παρέχει σε αυτούς περισσότερο χρόνο απ΄ ότι στους αργούς. Ακόμη είναι αναγκαίο να αναφερθεί ότι ο Master ελέγχει

ποια block έχει στη μνήμη ο Slave και έτσι προτιμά να του στέλνει τα διανύσματα που αφορούν αυτά με σκοπό την μείωση της επικοινωνίας μεταξύ τους.



Σχήμα 5.2: Το μοντέλο Master Slave

6 Λεπτομέρειες υλοποίησης και δοκιμών

6.1 Λεπτομέρειες του παράλληλου υπολογιστή

Όλες οι μετρήσεις έγιναν σε ένα υβριδικό cluster της σχολής Χημικών Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου. Αποτελείται από τέσσερις κόμβους HP ProLiant SL390s G7[4]. Τα χαρακτηριστικά κάθε κόμβου είναι:

- 2 Xeon CPUs X5660 @ 2.80GHz - Σύνολο: 12 πυρήνες (επεξεργαστές) και 16 GB RAM.
- 2 nVIDIA GPUs Tesla M2050 - Σύνολο: 896 Cuda cores με 6 GB RAM στις κάρτες γραφικών.

Οι κόμβοι είναι συνδεδεμένοι μεταξύ τους με δίκτυο Gigabit Ethernet.

Η εφαρμογή δοκιμάστηκε με αρκετούς συνδυασμούς επεξεργαστών και κόμβων. Παρ΄ όλα αυτά, οι μετρήσεις επίδοσης της εφαρμογής έγιναν σε έναν κόμβο για λόγους διαθεσιμότητας του cluster. Εξαίρεση σε αυτό, αποτελούν οι δοκιμές με περισσότερες από 2 GPUs.

6.2 Σχετικά με το λογισμικό

Η εφαρμογή αυτή αναπτύχθηκε σε γλώσσα C++ (του προτύπου 2003 [6]) με χρήση αντικειμενοστραφούς προγραμματισμού. Η επιλογή της γλώσσας έγινε με σκοπό την αναγνωσιμότητα του κώδικα (κάτι που το πετυχαίνει αρκετά καλά το αντικειμενοστραφές μοντέλο) αλλά και τις επιδόσεις της εφαρμογής. Παρόλο που η C++ είναι εξελιγμένη γλώσσα γενικού προγραμματισμού, τα εκτελέσιμα αρχεία που παράγει είναι native, δηλαδή γλώσσας μηχανής. Σε συνδυασμό με τις βελτιστοποιήσεις που παρέχουν οι σημερινοί μεταγλωττιστές, την καθιστούν ως μια πολύ ισχυρή και γρήγορη γλώσσα.

Μια από τις βασικές βιβλιοθήκες που χρησιμοποιήθηκαν, ως κορμός, για την εφαρμογή είναι η Eigen 3.2 [14]. Η βιβλιοθήκη αυτή παρέχει χειρισμό πινάκων (αραιών και πυκνών) αλλά και αρκετές από τις πράξεις της γραμμικής άλγεβρας. Η εκτεταμένη χρήση template που διαθέτει, την κάνουν αρκετά ευέλικτη και εύκολη στη χρήση. Διαθέτει επίσης και κάποιους επιλύτες γραμμικών συστημάτων.

Η βιβλιοθήκη που χρησιμοποιήθηκε για την αξιοποίηση των καρτών γραφικών είναι η Cula 4.0 [15]. Παρέχει αρκετούς επιλύτες αραιών πινάκων αλλά και τους preconditioners που μπορούν να εκμεταλλευτούν. Η Cula φροντίζει από μόνη της την μεταφορά των συστημάτων στην μνήμη της

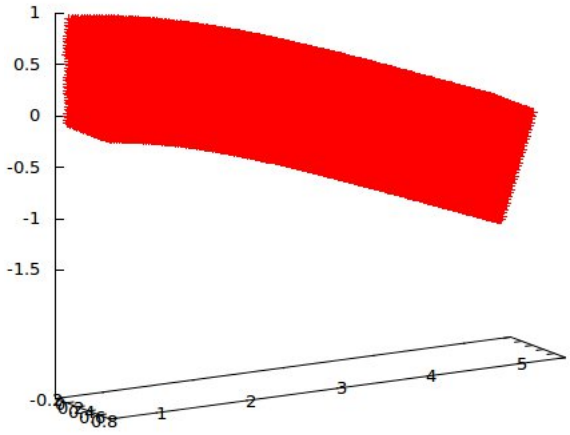
κάρτας γραφικών. Ένα μειονέκτημα που παρουσιάζει, είναι ότι δεν μπορεί να διατηρηθεί στην μνήμη ένα μητρώο για δεύτερη επίλυση του. Παρ΄ όλα αυτά, το κόστος μεταφοράς του παραμένει χαμηλό.

Η τελευταία βιβλιοθήκη που χρησιμοποιήθηκε, ήταν η boost [3]. Αυτή παρέχει containers στα οποία μπορούν να τοποθετηθούν εύκολα, διάφορα αντικείμενα της C++ και να σταλούν μέσω του MPI σε κάποιον άλλο επεξεργαστή.

6.3 Λεπτομέρειες του προβλήματος

Κάθε φορά που εκτελείται μια εφαρμογή MPI, δημιουργούνται στο λειτουργικό σύστημα τόσες νέες διεργασίες, όσες είναι και οι διαθέσιμες CPUs. Η δημιουργία μιας νέας διεργασίας έχει μια σταθερή χρονική επιβάρυνση. Κάτι αντίστοιχο συμβαίνει και με τα μηνύματα που στέλνουν οι CPUs μεταξύ τους. Αναλυτικότερα, κάθε μήνυμα περιέχει μια σταθερού μεγέθους κεφαλίδα η οποία έχει και αυτή σταθερή χρονική επιβάρυνση. Για να εμφανιστούν καλές επιδόσεις σε μια παράλληλη εφαρμογή, πρέπει ο συνολικός χρόνος εκτέλεσης να είναι πολύ μεγαλύτερος από την επιβάρυνση δημιουργίας νέων διεργασιών. Αντίστοιχα τα μηνύματα που ανταλλάσσουν οι CPUs πρέπει να είναι αρκετά μεγαλύτερα από τις κεφαλίδες τους.

Δεδομένης της επεξεργαστικής ισχύς του κόμβου και για να μην εμφανιστούν φαινόμενα καθυστέρησης από αρχικοποίηση, επιλέχθηκε να δημιουργηθεί πλέγμα διαστάσεων $[161, 33, 33]$. Αυτό σημαίνει πως το μητρώο δυσκαμψίας έχει διαστάσεις 525.987 επί 525.987 και έχει 40.230.009 μη μηδενικά στοιχεία. Η λύση του προβλήματος με την επαναληπτική διαδικασία Biconjugate gradient stabilized (BicgStab) για υπόλοιπο 10^{-5} φαίνεται παρακάτω:



Σχήμα 6.1: Λύση του προβλήματος της δοκού

Η επίλυση του έγινε σε 259 επαναλήψεις και διήρκεσε 99 δευτερόλεπτα.

6.4 Επιλύτες των block και του συνόρου

Ο επιλύτης που επιλέχθηκε και για τα block (εξ. 2.11) αλλά και για το σύνορο (εξ. 2.9) είναι η BicgStab [11]. Η επιθυμητή ακρίβεια επίλυσης ολόκληρου του συστήματος ορίζεται στο 10^{-5}(όσες δοκιμές κατέληξαν με μεγαλύτερο υπόλοιπο δεν καταγράφηκαν). Έτσι επιλέχθηκε η ακρίβεια 10^{-6} για την επίλυση του συνόρου. Μετά τις μετρήσεις, φάνηκε πως η ακρίβεια των επιλυτών των εξισώσεων 2.11, παίζει πολύ σημαντικό ρόλο στη σύγκλιση του συνόρου αλλά και στην ακρίβεια των αποτελεσμάτων όλου του συστήματος. Στην αρχή δοκιμάστηκαν επιλύτες με ακρίβεια 10^{-14} αλλά παρουσιάστηκε δυσκολία σύγκλισης όταν επιλύθηκαν στις κάρτες γραφικών. Όταν η επιθυμητή ακρίβεια μειώθηκε στο 10^{-13}, δεν υπήρξαν ιδιαίτερα προβλήματα στη σύγκλιση.

Το γεγονός ότι παρουσιάζονται διαφορετικά αποτελέσματα από τους επεξεργαστές και τις κάρτες γραφικών είναι απολύτως φυσιολογικό λόγω των διαφορών στην αρχιτεκτονική τους [17]. Αυτό σημαίνει πως οι υπολογισμοί που γίνονται στην GPU έχουν διαφορετικό round-off error από αυτούς που γίνονται στην CPU. Αυτό μπορεί να συμβαίνει επειδή οι GPUs χρησιμοποιούν πάρα πολλά νήματα για να πραγματοποιήσουν υπολογισμούς ενώ οι CPUs πολύ λιγότερα ή κανένα. Επίσης, η σύγκλιση των επαναληπτικών μεθόδων αλλάζει αρκετά από βιβλιοθήκη σε βιβλιοθήκη.

7 Επιδόσεις της εφαρμογής

Στις παρακάτω μετρήσεις ορίζονται οι εξής δείκτες :

- Χρόνος μεταφοράς: Είναι ο χρόνος που δαπανάται στις μεταφορές των blocks (εξ. 2.11) και των αποτελεσμάτων, από τον master στους slaves.
- Χρόνος αναμονής: Είναι ο χρόνος από τη στιγμή που ένας επεξεργαστής έχει τελειώσει με τους υπολογισμούς και αναμένει τους υπόλοιπους επεξεργαστές για να περάσουν το φράγμα και μετά όλοι μαζί, να προχωρήσουν στην επόμενη επανάληψη. Υψηλός χρόνος αναμονής σημαίνει ότι δεν έχει γίνει σωστή κατανομή φορτίου.
- Χρόνος υπολογισμού: Είναι ο χρόνος που αφιερώνει κάθε επεξεργαστής στην επίλυση των blocks (εξίσωση 2.11). Για να γίνεται σωστή κατανομή φορτίου, πρέπει όλοι οι επεξεργαστές που συμμετέχουν στον υπολογισμό, να έχουν περίπου ίδιο χρόνο υπολογισμού.
- Συνολικός χρόνος υπολογισμού: είναι ο χρόνος που χρειάζεται η εφαρμογή να λύσει με τη μέθοδο του Schur Complement το πρόβλημα της δοκού. Ο συνολικός χρόνος είναι πάντα μεγαλύτερος από τον μέγιστο χρόνο υπολογισμού των slaves και είναι ίσος με το άθροισμα του χρόνου μεταφοράς, αναμονής και υπολογισμού.
- Επιτάχυνση ή Speed-up: Ορίζεται ως

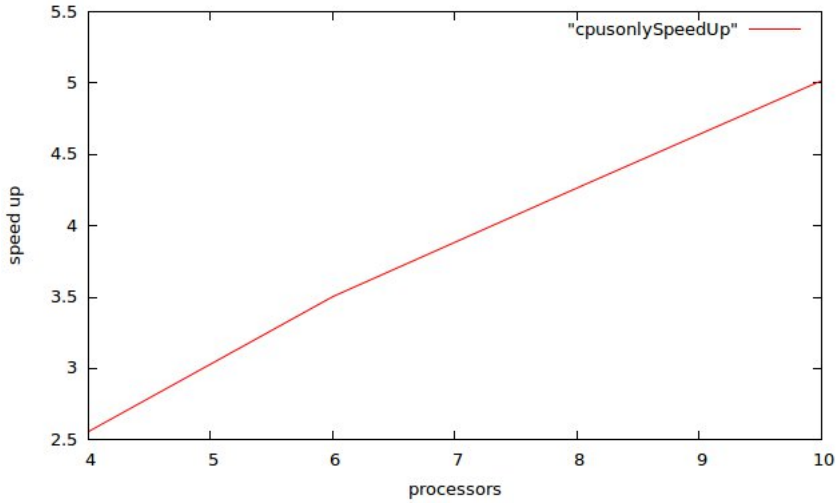
$$S = \frac{T_2}{T_i}$$

  Σε όλες τις περιπτώσεις το T_2 είναι ο χρόνος που χρειάζεται για να λύσουν το πρόβλημα της δοκού, δύο επεξεργαστές με ισομερή διαμέριση δύο Block. Το T_i είναι ο συνολικός χρόνος εκτέλεσης κάθε περίπτωσης με i πλήθος CPUs ή GPUs.

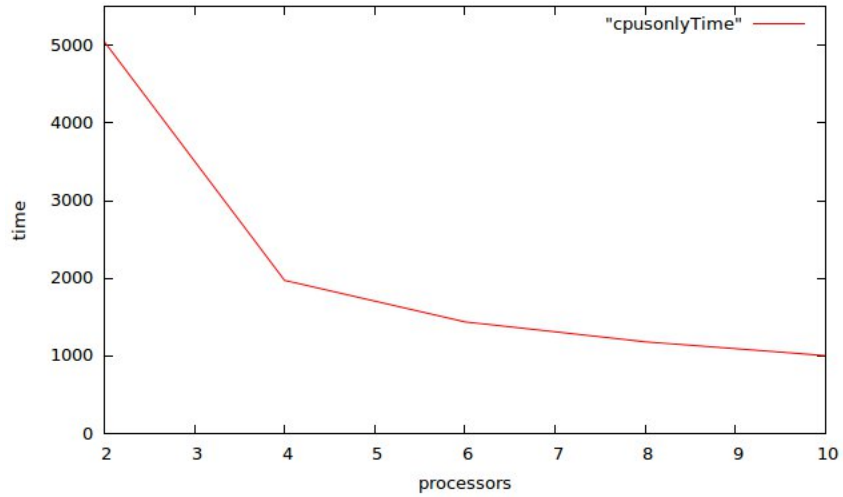
7.1 Αποκλειστική χρήση επεξεργαστών

Σε αυτό το σενάριο φαίνονται οι χρόνοι εκτέλεσης της εφαρμογής, όταν εκτελείται μόνο σε επεξεργαστές. Κάθε επεξεργαστής λαμβάνει ένα block (εξίσωση 2.11) και συνεχίζει να λύνει μόνο αυτό

το block (λειτουργία cached block). Έτσι μειώνεται ο χρόνος σε των μεταφορών στο ελάχιστο. Η δοκός χωρίστηκε σε ίσου μεγέθους block. Στο διάγραμμα 7.1 φαίνεται το speed-up της εφαρμογής.



Σχήμα 7.1: Διάγραμμα SpeedUp μόνο με χρήση Cpu

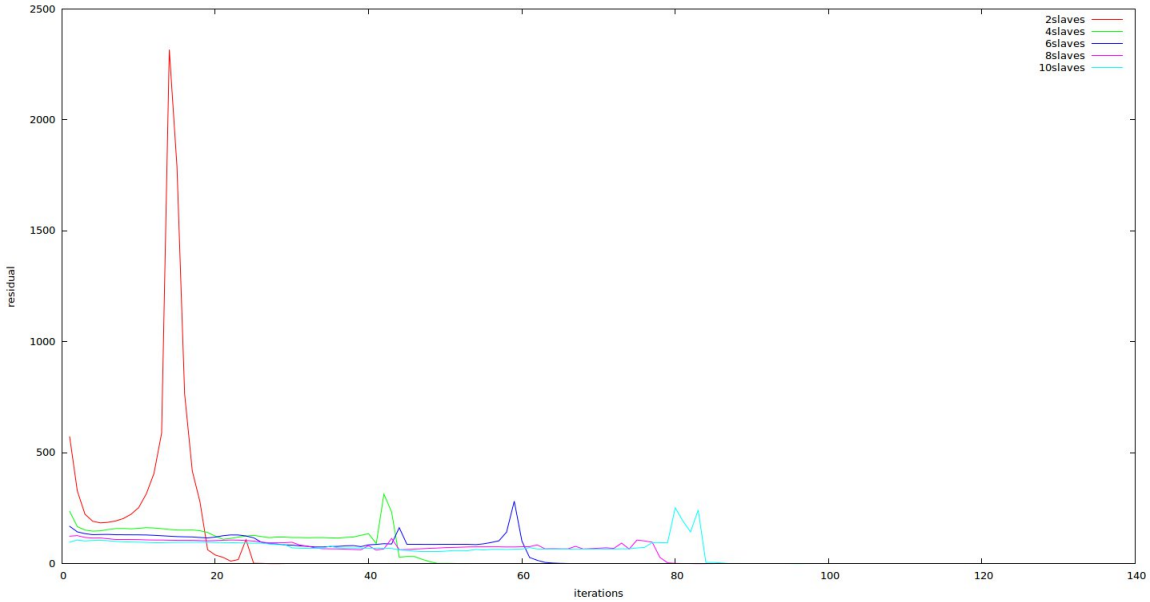


Σχήμα 7.2: Χρόνοι εκτέλεσης των CPU

Στο διάγραμμα 7.3 φαίνονται οι σχέσεις των residuals

$$r_i = Sy_i - g + FB^{-1}f \tag{7.1}$$

που προκύπτουν από τις επαναλήψεις krylov της εξίσωσης 2.9 για κάθε περίπτωση. Γίνεται λοιπόν φανερό πως όσο περισσότερα block υπάρχουν, ή διαφορετικά όσο περισσότεροι άγνωστοι υπάρχουν στο σύνορο, τόσο πιο δύσκολη είναι η σύγκλιση της μεθόδου.



Σχήμα 7.3: Μόνο CPU residuals - επαναλήψεις

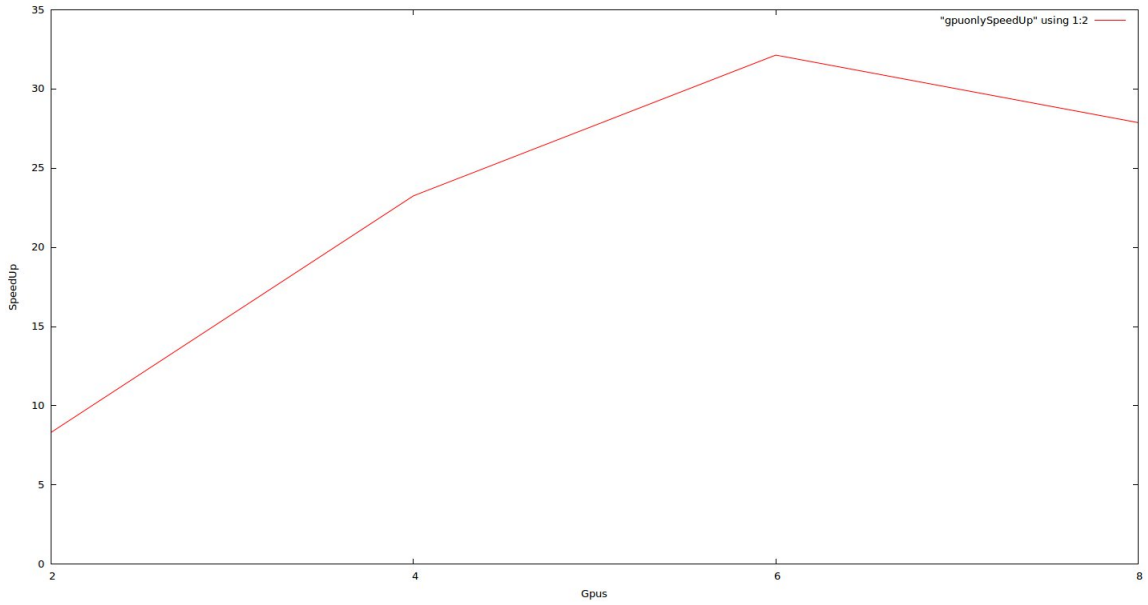
Επεξεργαστής	1	2	3	4	5	6	7	8	9	10
Μεταφορά	0.94	1.13	0.97	0.55	1.15	1.07	1	1.06	0.97	1.03
Αναμονή	115.5	60.87	354.56	646.45	30.27	42.83	354.66	40.85	353.62	44.93
Υπολογισμός	889.69	944.08	650.61	358.69	974.7	962.16	650.49	964.19	651.57	960.1

Πίνακας 7.1: Χρόνοι σε sec. για την περίπτωση των 10 CPU

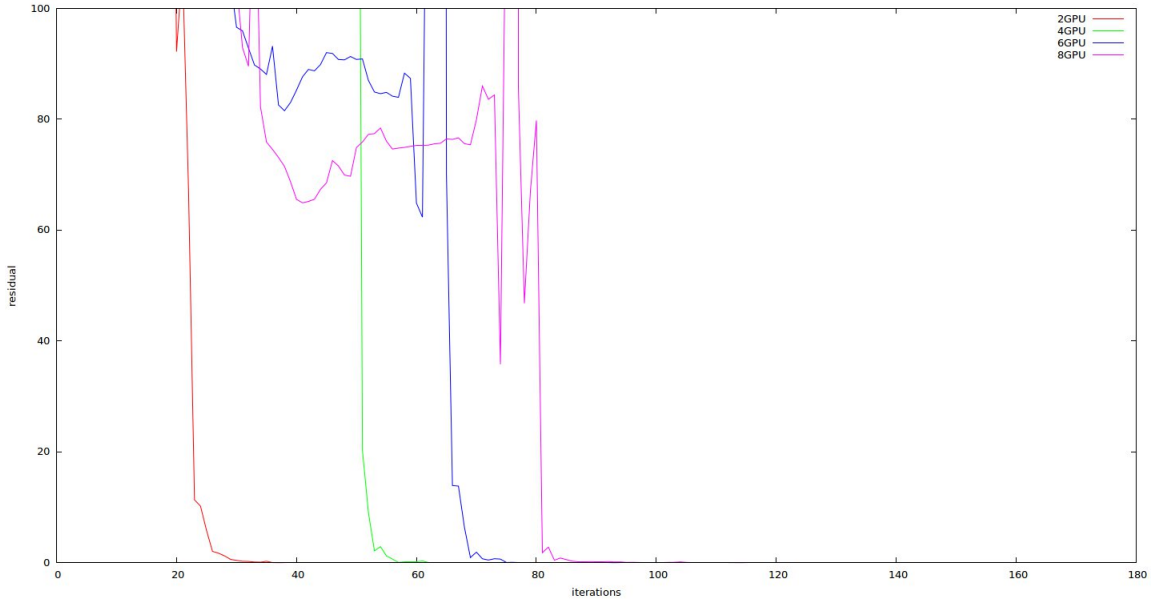
Στον πίνακα 7.1 φαίνεται όπως αναμενόταν, πως οι χρόνοι που δαπανήθηκαν στις μεταφορές δεδομένων, είναι αμελητέοι. Όμως είναι ξεκάθαρο ότι κάποιοι επεξεργαστές κάνανε αρκετά περισσότερο χρόνο να λύσουν τα συστήματα που τους ανατέθηκαν και έτσι όλοι οι υπόλοιποι βρίσκονταν σε κατάσταση αναμονής ενώ θα μπορούσαν να επιλύουν κάτι άλλο. Αυτό είναι ένα από τα βασικά προβλήματα του παράλληλου προγραμματισμού και ονομάζεται load balancing.

7.2 Επίλυση στις κάρτες γραφικών

Με αντίστοιχες συνθήκες, σαν της παραγράφου 7.1, έγιναν και οι μετρήσεις για τις κάρτες γραφικών.



Σχήμα 7.4: Διάγραμμα Speed-up για αποκλειστική χρήση GPU



Σχήμα 7.5: Αποκλειστική χρήση GPU, Residual - επαναλήψεις

GPU	1	2
Μεταφορά	4.68	4.48
Αναμονή	321.66	5.54
Υπολογισμός	280.07	596.48

Πίνακας 7.2: Χρόνοι σε sec. για την περίπτωση των 2 GPU με 2 blocks

Στην περίπτωση των 2 GPU - 2 blocks, (πίνακας 7.2) ο συνολικός χρόνος εκτέλεσης είναι 607 δευτερόλεπτα και το Speed-up 8.3. Παρ΄ ότι τα blocks έχουν το ίδιο μέγεθος, οι χρόνοι υπολογισμού είναι σχεδόν διπλάσιοι από τη μία κάρτα στην άλλη με αποτέλεσμα να μην υπάρχει σωστή κατανομή φορτίου.

GPU	1	2	3	4
Μεταφορά	2.24	2.31	3.11	4.17
Αναμονή	75.38	0.33	80.71	66.74
Υπολογισμός	140.07	215.17	133.96	146.8

Πίνακας 7.3: Χρόνοι σε sec. για την περίπτωση των 4 GPU με 4 blocks

Για την περίπτωση των 4 GPU - 4Blocks (πίνακας7.3), ο συνολικός χρόνος εκτέλεσης είναι 217.82 και το Speed-up είναι 23.25.

GPU	1	2	3	4	5	6
Μεταφορά	1.42	5.76	2.42	5.24	4.86	3.3
Αναμονή	60.83	38.7	52.33	46.15	19.12	44.48
Υπολογισμός	95.88	113.53	103.25	106.57	133.6	109.63

Πίνακας 7.4: Χρόνοι σε sec. για την περίπτωση των 6 GPU με 6 blocks

Ο συνολικός χρόνος εκτέλεσης της περίπτωσης των 6 GPU (πίνακας 7.4) 157.84 sec και το Speed-up είναι 32.14. Και σε αυτή την περίπτωση είναι φανερό πως η κατανομή φορτίου δεν είναι ιδανική. Στην καλύτερη των περιπτώσεων, ο συνολικός χρόνος εκτέλεσης θα μπορούσε να είναι κοντά στο χρόνο υπολογισμού του πέμπτου slave.

Τελευταία περίπτωση είναι με τις 8 GPU - 8 block όπου παρατηρείται πτώση του Speed-up και αδυναμία σύγκλισης στην τιμή που συγκλίνουν οι προηγούμενες περιπτώσεις.

7.2.1 Κατανομή φορτίου στις GPU

Μιας και παρουσιάζονται μεγάλοι χρόνοι αναμονής, όταν κάθε GPU υπολογίζει ένα μόνο block τη φορά, παρατίθεται η παρακάτω περίπτωση όπου το χωρίο διαμερίζεται σε 4 blocks που τα έχουν και οι δύο κάρτες γραφικών στη μνήμη.

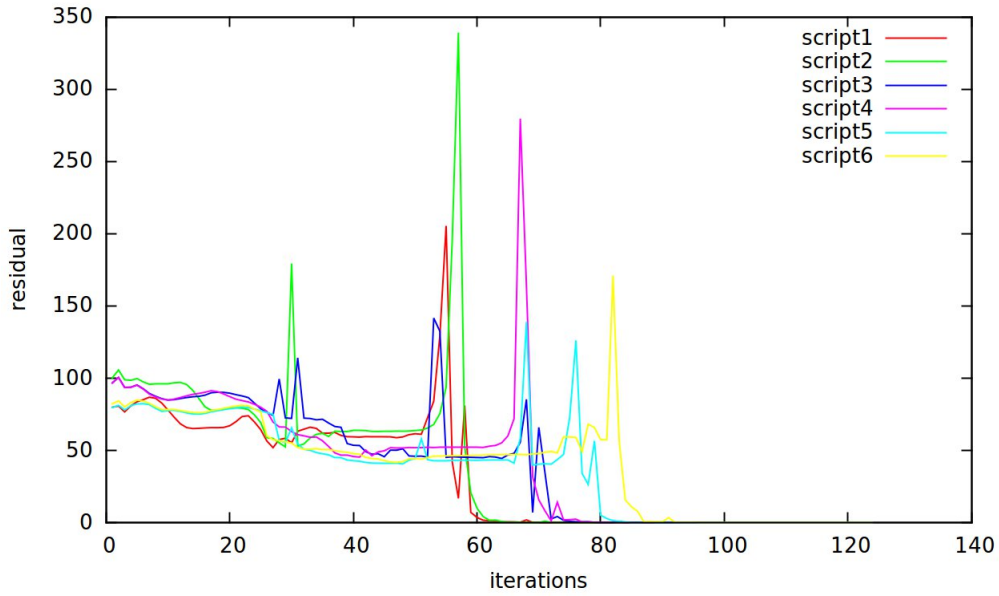
GPU	1	2
Μεταφορά	6.72	6.76
Αναμονή	17.14	26.24
Υπολογισμός	253.99	244.69

Πίνακας 7.5: Χρόνοι σε sec για την περίπτωση των 2 GPU με load balancing

Παρατηρείται μεγάλη βελτίωση στο χρόνο εκτέλεσης ο οποίος είναι 278.97 και το Speed-up είναι 18.08. Αυτό που είναι σημαντικό είναι ότι οι κάρτες γραφικών πλέον αφιερώνουν σχεδόν όλο το χρόνο εκτέλεσης σε υπολογισμούς και όχι σε αναμονή ή μεταφορές δεδομένων, δεδομένου ότι δεν αυξάνονται κατά πολύ οι επαναλήψεις για την εύρεση του ορίου.

7.3 Επίλυση με συνδυασμό CPU - GPU

Από τις προηγούμενες παραγράφους είναι φανερό πως οι κάρτες γραφικών κάνουν πολύ ταχύτερους υπολογισμούς από τους κοινούς επεξεργαστές. Κάθε απόπειρα συνδυασμού αυτών, απαιτεί να γίνει και η ανάλογη διαμέριση του χωρίου. Σε όλες τις περιπτώσεις αυτού του κεφαλαίου, τα μεγάλα blocks στέλνονται στις κάρτες γραφικών, ενώ τα μικρά στους επεξεργαστές.



Σχήμα 7.6: residuals - επαναλήψεις για το υβριδικό μοντέλο

7.3.1 2 GPUs - 4 Cpus (Σενάριο 1)

Σε αυτή την περίπτωση έγινε η παρακάτω διαμέριση:

Βαθμοί ελευθερίας	Επεξεργαστής
209088	GPU1
209088	GPU2
22869	CPU1
22869	CPU2
22869	CPU3
22869	CPU4

Πίνακας 7.6: Διαμέριση Σεναρίου 1

Οι άγνωστοι που βρίσκονται στο σύνορο είναι 16335. Ο συνολικός χρόνος εκτέλεσης είναι 419 δευτερόλεπτα και το Speed-up 12.04.

slave	1	2	3	4	5	6
Μεταφορά	3.64	3.8	0.42	0.48	0.48	0.44
Αναμονή	28.94	15.6	262.82	317.46	316.14	344.98
Υπολογισμός	385.88	398.79	156.27	101.56	102.89	74.09

Πίνακας 7.7: Χρόνοι σε sec για την περίπτωση του σεναρίου 1

Από τον πίνακα 7.7 είναι φανερό ότι οι CPUs (3-6) δεν έχουν τόσο μεγάλα blocks και έτσι αναμένουν τις κάρτες γραφικών.

7.3.2 2 GPUs - 4 Cpus (Σενάριο 2)

Δεδομένου ότι πρέπει να φορτιστούν λίγο περισσότερο οι CPUs προκύπτει η παρακάτω διαμέριση με σύνορο μεγέθους 16335:

Βαθμοί ελευθερίας	Επεξεργαστής
179685	GPU1
179685	GPU2
39204	CPU1
39204	CPU2
39204	CPU3
32670	CPU4

Πίνακας 7.8: Διαμέριση Σεναρίου 2

slave	1	2	3	4	5	6
Μεταφορά	3.1	3.24	0,61	0.78	0.72	0.78
Αναμονή	27.36	12.66	90.37	64.72	63.61	67.72
Υπολογισμός	274.63	288.79	215.26	240.74	241,91	237.75

Πίνακας 7.9: Χρόνοι σε sec για την περίπτωση του σεναρίου 2

Με αυτή τη διαμέριση, ο συνολικός χρόνος εκτέλεσης είναι 306 δευτερόλεπτα ο οποίος έχει βελτιωθεί αρκετά. Το Speed-up είναι 16.04.

7.3.3 2 GPUs - 6 Cpus (Σενάριο 3)

Σε αυτό το σενάριο το σύνορο είναι 22869 αγνώστων και η διαμέριση είναι:

Βαθμοί ελευθερίας	Επεξεργαστής
153549	GPU1
153549	GPU2
32670	CPU1
32670	CPU2
32670	CPU3
32670	CPU4
32670	CPU5
32670	CPU6

Πίνακας 7.10: Διαμέριση σεναρίου 3

slave	1	2	3	4	5	6	7	8
Μεταφορά	2.68	2.81	0.64	0.65	0.69	0.67	0.64	0.66
Αναμονή	105.91	97.53	126.24	10.78	126.12	126.65	212.21	207.18
Υπολογισμός	302.24	312.67	290.14	405.59	290.19	289.89	204.18	209.19

Σχήμα 7.7: Χρόνοι σε sec για την περίπτωση του σεναρίου 3

Ο συνολικός χρόνος εκτέλεσης είναι 416.89 και το Speed-up 12.10. Είναι φανερό ότι ο slave 4 καθυστερεί πολύ την επίλυση καθώς έχει τον κατά πολύ μεγαλύτερο χρόνο υπολογισμού.

7.3.4 2 GPUs - 6 Cpus (Σενάριο 4)

Βαθμοί ελευθερίας	Επεξεργαστής
160083	GPU1
160083	GPU2
26136	CPU1
26136	CPU2
32670	CPU3
32670	CPU4
32670	CPU5
32670	CPU6

Πίνακας 7.11: Διαμέριση σεναρίου 4

slave	1	2	3	4	5	6	7	8
Μεταφορά	2.79	2.88	0.55	0.59	0.71	0.6	0.55	0.64
Αναμονή	28.52	29.34	128.42	22.1	69.97	113.07	129.23	70.94
Υπολογισμός	253.76	253.2	158.25	264.5	216.52	173.54	157.43	215.63

Σχήμα 7.8: Χρόνοι σε sec για την περίπτωση του σεναρίου 4

Με αυτή τη διαμέριση ο συνολικός χρόνος είναι 286.96 και το Speed-up 17.58.

7.3.5 2 GPUs - 8 Cpus (Σενάριο 5)

Χρησιμοποιώντας την διαμέριση με σύνορο 29403 αγνώστων:

Βαθμοί ελευθερίας	Επεξεργαστής
147015	GPU1
147015	GPU2
26136	CPU1
26136	CPU2
26136	CPU3
26136	CPU4
26136	CPU5
26136	CPU6
26136	CPU7
19602	CPU8

Πίνακας 7.12: Διαμέριση σεναρίου 5

slave	1	2	3	4	5	6	7	8	9	10
Μεταφορά	2.66	2.7	0.57	0.56	0.52	0.55	0.59	0.57	0.56	0.46
Αναμονή	31.17	13.4	118.12	127.93	117.92	125.84	127.52	128.4	115.07	117.83
Υπολογισμός	277.9	296.7	196	186.21	196.24	188.3	186.6	185.74	199.07	196.39

Πίνακας 7.13: Χρόνοι σε sec για την περίπτωση του σεναρίου 5

Ο συνολικός χρόνος εκτέλεσης είναι 314.53 sec ο οποίος είναι αρκετά καλός δεδομένου ότι οι CPU έχουν περιθώριο αύξησης φορτίου και ότι πλέον υπάρχουν αρκετά blocks. Σε αυτό το σενάριο το Speed-up είναι 16.

7.3.6 2 GPUs - 8 Cpus (Σενάριο 6)

Η επόμενη διαμέριση έγινε με σύνορο 29403 αγνώστων:

Βαθμοί ελευθερίας	Επεξεργαστής
147015	GPU1
143748	GPU2
26136	CPU1
26136	CPU2
26136	CPU3
26136	CPU4
26136	CPU5
26136	CPU6
26136	CPU7
22869	CPU8

Πίνακας 7.14: Διαμέριση σεναρίου 6

slave	1	2	3	4	5	6	7	8	9	10
Μεταφορά	2.62	2.61	0.55	0.52	0.55	0.53	0.57	0.55	0.58	0.47
Αναμονή	23.46	12.34	50.35	131.12	51.83	54.2	54.88	54.47	126.2	93.39
Υπολογισμός	255.46	267.31	232.69	151.95	231.2	228.85	228.13	228.55	156.84	189.71

Πίνακας 7.15: Χρόνοι σε sec για την περίπτωση του σεναρίου 6

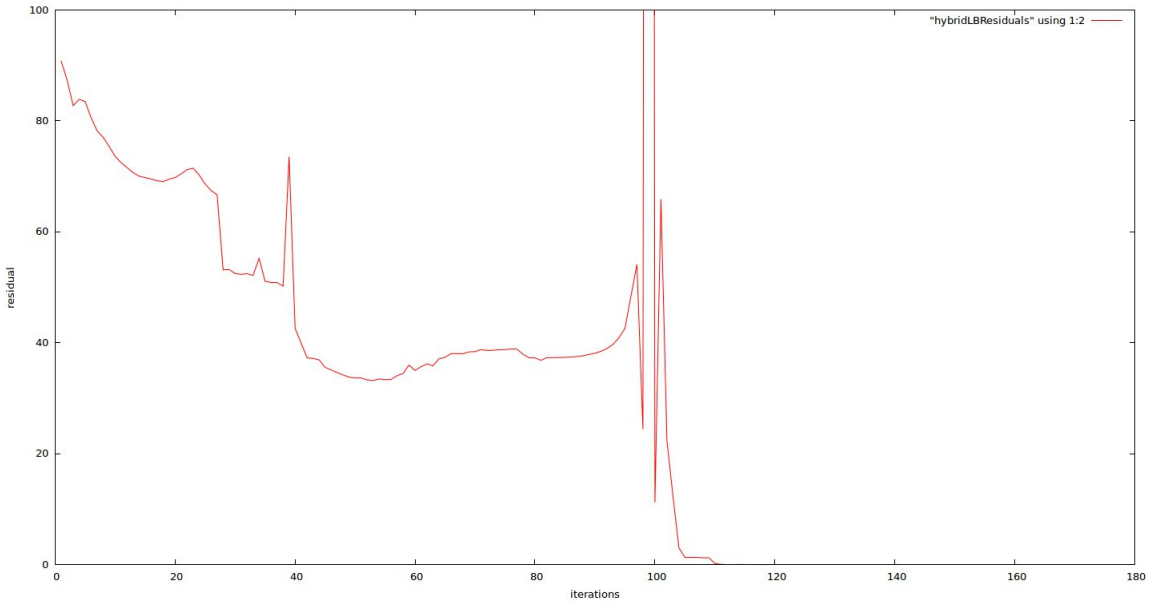
Ο τελικός χρόνος εκτέλεσης είναι 283.46 δευτερόλεπτα και το Speed-up 17.8.

7.3.7 2 GPUs - 8 Cpus (Load balancing)

Στα προηγούμενα σενάρια οι διαμερίσεις προκύψανε μετά από αρκετές δοκιμές ώστε να έχουν γρήγορους χρόνους εκτέλεσης. Σε αυτή την δοκιμή θα δημιουργηθούν πολλά και μικρότερα blocks (15 στο σύνολο) ώστε να μπορεί ο master να κάνει κατανομή φορτίου. Η διαμέριση που έγινε είναι με σύνορο 49005 αγνώστων:

Βαθμοί ελευθερίας
68607
68607
68607
68607
16335
16335
16335
16335
16335
16335
16335
16335
16335
16335
39204

Πίνακας 7.16: Διαμέριση σεναρίου load balancing



Σχήμα 7.9: residuals - επαναλήψεις για το υβριδικό μοντέλο (load balancing)

Σε αυτό το σενάριο κάθε slave μπορεί να έχει στη μνήμη του μέχρι 10 blocks. Τα τέσσερα

μεγάλα blocks μπορούν να χωρέσουν μόνο στην μνήμη των GPU, επιβάλλοντας έτσι τον υπολογισμό τους μόνο από αυτές. Ο τελικός χρόνος εκτέλεσης είναι 392.98 sec και το Speed-up είναι 12.84. Συγκρίνοντας αυτό το σενάριο με το σενάριο 6 (7.3.6), θα μπορούσε να πει κάποιος ότι η κατανομή φορτίου δουλεύει αρκετά καλά. Αυτό το σενάριο χρειάστηκε λίγες επαναλήψεις παραπάνω απ΄ τις διπλάσιες (του σεναρίου 6) για να συγκλίνει. Παρ΄ όλα αυτά, ό χρόνος αυξήθηκε μόλις κατά 38% και όχι 100%.

slave	1	2	3	4	5	6	7	8	9	10
Μεταφορά	6.2	6.71	1.11	1.21	1.03	1.28	1.11	1.04	1.33	1.45
Αναμονή	162.42	112.67	279.66	278.3	320.19	278.59	321.82	279.55	254.62	170.78
Υπολογισμός	224	272.36	112.48	113.71	72	113.37	70.2	112.56	137.27	220.93

Πίνακας 7.17: Χρόνοι σε sec για την περίπτωση του σεναρίου load balancing

7.3.8 2 GPUs - 8 Cpus (Load balancing - τυχαία διαμέριση)

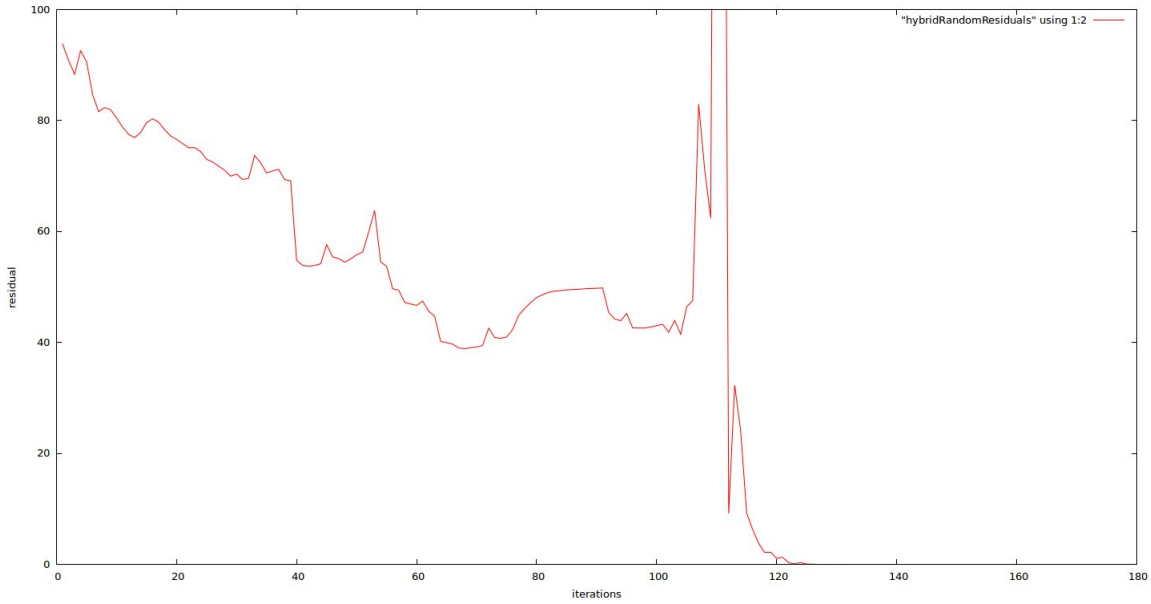
Σε πραγματική επίλυση προβλημάτων, συνήθως δεν υπάρχει η πολυτέλεια χρόνου να γίνουν δοκιμές ώστε να προκύψει μια αποδοτική διαμέριση. Σε αυτό το σενάριο θα γίνει επίλυση του προβλήματος με διαμέριση των 18 block τυχαίου μεγέθους (χρησιμοποιώντας την συνάρτηση rand() της C++). Η διαμέριση που προέκυψε, με σύνορο 58806 αγνώστων, είναι η παρακάτω:

Βαθμοί ελευθερίας
45738
16335
26136
9801
16335
42471
29403
13068
16335
35937
32670
22869
29403
16335
29403
29403
13068
42471

Πίνακας 7.18: Διαμέριση σεναρίου load balancing τυχαίας διαμέρισης

slave	1	2	3	4	5	6	7	8	9	10
Μεταφορά	8.26	7.09	2.51	2.01	2.8	1.2	1.62	1.54	1.67	0.76
Αναμονή	466.16	466.58	18.96	352.95	272.87	294.68	138.21	255.84	249.27	298.23
Υπολογισμός	170.86	172.64	625.22	291.87	371.09	350.96	507.06	389.45	395.93	347.82

Πίνακας 7.19: Χρόνοι σε sec για την περίπτωση του σεναρίου load balancing τυχαίας διαμέρισης



Σχήμα 7.10: residuals - επαναλήψεις για το υβριδικό μοντέλο (load balancing) τυχαίας διαμέρισης

Ο συνολικός χρόνος εκτέλεσης είναι ήταν 646.12 sec και το Speed-up είναι 7.8. Είναι λοιπόν φανερό πως η διαμέριση του χωρίου παίζει πολύ σημαντικό ρόλο στον τελικό χρόνο εκτέλεσης.

Σενάριο	Συνολικός χρόνος	Speed-up
1	419	12.04
2	306	16.04
3	416.89	12.1
4	286.96	17.58
5	314.53	16
6	283.46	17.8
load balancing	392.98	12.84
load balancing – random	646.12	7.8

Πίνακας 7.20: Συγκεντρωτικός πίνακας σεναρίων

7.4 Επίλυση σε slaves όπου δεν έχουν αρκετή μνήμη

Σε αυτό το σενάριο η διαμέριση θα γίνει σε τέσσερα blocks και θα επιλυθεί στις κάρτες γραφικών. Η ιδιαιτερότητα του σεναρίου αυτού είναι ότι η κάθε κάρτα θα μπορεί να αποθηκεύσει ένα μόνο block

τη φορά. Αυτό σημαίνει πως σε κάθε υπολογισμό θα πρέπει να ξαναστέλνεται από τον master.

GPU	1	2
Μεταφορά	590.7	575.91
Αναμονή	150.08	289.52
Υπολογισμός	573.78	449.31

Πίνακας 7.21: Χρόνοι για την περίπτωση των 2 GPU χωρίς αρκετή μνήμη

Ο συνολικός χρόνος εκτέλεσης είναι 1316.93 δευτερόλεπτα και το Speed up μόλις 3.83. Είναι ξεκάθαρο πως όταν τα blocks δεν διατηρούνται στη μνήμη των slaves, τότε ο χρόνος μεταφοράς παίζει ιδιαίτερο ρόλο. Βέβαια, αυτού του είδους η πρακτική, επιτρέπει την επίλυση συστημάτων και σε cluster όπου δεν έχουν αρκετή μνήμη ώστε να διατηρήσουν ολόκληρο το πρόβλημα.

8 Συμπεράσματα

Από τις μετρήσεις που έγιναν στο κεφάλαιο 7 , διαπιστώνεται ότι με την αύξηση του πλήθους των στοιχείων του συνόρου, αυξάνεται και ο αριθμός των επαναλήψεων για να επιλυθεί το σύνορο. Αποτέλεσμα αυτού, είναι ότι ο συνολικός χρόνος εκτέλεσης αυξάνεται σημαντικά. Δικαιολογημένα λοιπόν, δεν παρουσιάζεται ιδιαίτερη αύξηση του Speed-up όταν προστίθενται περισσότερα CPUs ή GPUs.

Παρατηρείται επίσης ότι είναι εκπληκτική η ταχύτητα με την οποία λύνονται τα συστήματα από τις GPUs. Το μεγαλύτερο Speed-up επετεύχθη, όταν το πρόβλημα επιλύθηκε με δύο κάρτες γραφικών και διαμέριση τεσσάρων blocks με load balancing. Γίνεται πλέον σαφές ότι οι κάρτες γραφικών γίνονται απαραίτητες στον χώρο της υπολογιστικής μηχανικής και στον ευρύτερο χώρο του επιστημονικού υπολογισμού. Δεν μπορούν σε καμία περίπτωση βέβαια να αντικαταστήσουν τις CPUs καθώς δεν μπορούν να υπολογίσουν με τόση μεγάλη ακρίβεια (αυτό φαίνεται από την καθυστέρηση στη σύγκληση του γραφήματος 7.5 σε σχέση με αυτό των CPU) και δεν έχουν τόσο πολλή μνήμη όσο οι CPUs. Το τελευταίο είναι αρκετά μεγάλο πρόβλημα όταν υπάρχουν μεγάλα blocks. Στην περίπτωση που δεν υπάρχουν ισχυρές CPUs στο cluster τότε η επίλυση θα είναι πολύ αργή. Το ίδιο ακριβώς θα συμβεί αν γίνουν επιπρόσθετες διαμερίσεις στα μεγάλα block.

Ένα ακόμα ζήτημα που αποκτά ιδιαίτερο ρόλο στους χρόνους εκτέλεσης είναι η συσχέτιση της κατανομής φορτίου με την ίδια τη διαμέριση. Από τις παραπάνω μετρήσεις, φαίνεται πως είναι επιθυμητό να έχουμε μικρό αριθμό αγνώστων στα σύνορα. Στις περισσότερες περιπτώσεις, αυτό μπορεί να συμβεί όταν έχουμε λίγα blocks. Για να γίνει σωστή κατανομή φορτίου, πρέπει το κάθε block να έχει το ανάλογο μέγεθος με αυτό των δυνατοτήτων του επεξεργαστή που πρόκειται να το επιλύσει. Αυτό από μόνο του καθιστά πολύ δύσκολη την δημιουργία μιας αυτοματοποιημένης διαδικασίας που θα δημιουργεί διαμερίσεις. Ένα βήμα περισσότερο όταν κατά την δημιουργία της διαμέρισης πρέπει να ληφθεί υπ΄ όψη και ο αριθμός κατάστασης του συστήματος που προκύπτει.

Η άλλη προσέγγισή είναι να δημιουργούνται περισσότερα blocks από τις CPUs/GPUs. Έτσι είναι πιο πιθανό να γίνει καλύτερη κατανομή φορτίου. Στη περίπτωση της δοκού, αυτό αυξάνει πολύ τον πλήθος τον αγνώστων στο σύνορο κάνοντας πολύ πιο αργή την επίλυση. Πιθανόν σε κάποιο άλλο πρόβλημα αυτή η προσέγγιση να λειτουργούσε καλύτερα.

References

[1] Advanced finite element methods-chapter 11. http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch11.d/.

[2] Whitepaper: Nvidia next generation cuda compute architecture: Fermi. Technical report.

[3] Boost c++ libraries. http://www.boost.org/.

[4] Ntua chemical engineering cluster. http://hpc.chemeng.ntua.gr.

[5] Beam image. www.codedevelopment.net.

[6] C++ standard. http://www.iso.org/iso/catalogue_detail.htm?csnumber=38110.

[7] Open source high performance computing. http://www.open-mpi.org/.

[8] Jaroslaw Bylina Anna pyzara, Beata Bylina. The influence of a matrix condition number on iterative methods' convergence. *2011 Federated Conference on Computer Science and Information Systems,FedCSIS 2011*, (6078297):459–464, 2011.

[9] Klaus Jurgen Bathe. *Finite Element Procedures*. Prentice Hall, 1996.

[10] Lawrence Livermore National Laboratory Blaise Barney. Introduction to parallel computing. https://computing.llnl.gov/tutorials/parallel_comp/.

[11] D. R. Fokkema G. L. G. Sleijpen, H. A. van der Vorst. Bicgstab(l) and other hybrid bi-cg methods. *Numerical Algorithms*, 7:75–109, 1994.

[12] Robert M. Kirby II George Em Karniadakis. *Parallel Scientific Computing in C++ and MPI*. Cambridge University Press, 1996.

[13] Zuo X.-Y. Liu X.-P. Li P.-L. Gu, T.-X. An improved parallel hybrid bi-conjugate gradient method suitable for distributed parallel computing. *Journal of Computational and Applied Mathematics*, 226:55–65, 2009.

[14] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[15] K. E. Spagnoli A. L. Paolini E. J. Kelmelis J. R. Humphrey, D. K. Price. Cula: Hybrid gpu accelerated linear algebra routines. SPIE Defense and Security Symposium (DSS), 2010.

[16] Petr Tichý Jörg Liessen. Convergence analysis of krylov subspace methods. Technical report.

[17] Alex Fit-Florea Nathan Whitehead. Precision & performance: Floating point and ieee 754 compliance for nvidia gpus. Technical report, 2011.

[18] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[19] O.C. Zienkiewicz & R.L. Taylor. *The Finite Element Method*, volume 1:The Basis. Butterworth-Heinemann, fifth edition, 2000.

[20] Gu T.-X. Liu X.-P. Zhu, S.-X. Minimizing synchronizations in sparse iterative solvers for distributed supercomputers. *Computers and Mathematics with Applications*, 67:199–209, 2014.