ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΓΕΝΙΚΟ ΤΜΗΜΑ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΦΑΡΜΟΣΜΕΝΕΣ ΕΠΙΣΤΗΜΕΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΔΙΑΤΡΙΒΗ ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ
ΚΑΤΕΥΘΥΝΣΗ : «ΕΦΑΡΜΟΣΜΕΝΑ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΑ ΜΑΘΗΜΑΤΙΚΑ»

ΕΠΙΛΥΣΗ ΜΕΓΑΛΩΝ ΓΡΑΜΜΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΡΙΘΜΗΤΙΚΩΝ ΜΕΘΟΔΩΝ ΠΕΠΕΡΑΣΜΕΝΩΝ ΣΤΟΙΧΕΙΩΝ ΣΕ ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ ΜΕ ΓΡΑΦΙΚΑ ΥΠΟΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΩΝ (GPUs)

ΝΙΚΟΛΑΟΣ Δ. ΒΙΛΑΝΑΚΗΣ

Επιβλέπων : Επ. Καθηγητής **Εμμανουήλ Μαθιουδάκης**

ΧΑΝΙΑ , 2013

Η διατριβή αυτή εξετάστηκε επιτυχώς από τη παρακάτω Τριμελή Επιτροπή:

- Επικ. Καθηγητή Εμμανούηλ Μαθιουδάκη (Επιβλέπων)
- Καθηγητή Ιωάννη Σαριδάκη
- Καθηγήτρια Έλενα Παπαδοπούλου

η οποία ορίστηκε κατά τη 13^{η}/ 6.12.2012 συνεδρίαση της Ειδικής Σύνθεσης του Γενικού Τμήματος του Πολυτεχνείου Κρήτης.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω όλους όσους με υποστήριξαν κατά τη διάρκεια των σπουδών μου, αλλά και με βοήθησαν για την ολοκλήρωση αυτής της διατριβής.

Ιδιαίτερα ευχαριστώ τον επιβλέποντα της διατριβής Επίκουρο Καθηγητή Εμμανουήλ Μαθιουδάκη για την επιστημονική καθοδήγηση και τις πολύτιμες γνώσεις που απλόχερα μου διέθεσε για την ολοκλήρωση της διατριβής.

Ευχαριστώ τους Καθηγητή Ιωάννη Σαριδάκη και Καθηγήτρια Έλενα Παπαδοπούλου για τη συμμετοχή τους στη τριμελή Επιτροπή, καθώς και τη βοήθεια τους κατά τη συγγραφή της πρώτης ερευνητικής μου εργασίας.

Οι συμφοιτητές μου Δημήτρης Μπομπολάκης και Νίκη Χαραλαμπάκη αξίζουν ιδιαίτερης αναφοράς για τις αξέχαστες στιγμές που ζήσαμε κατά την διάρκεια των μεταπτυχιακών μας σπουδών.

Τέλος, ένα μεγάλο ευχαριστώ δεν είναι ποτέ αρκετό προς την οικογένεια μου για την ηθική και υλική υποστήριξη που μου παρείχε κατά τη διάρκεια όλων των σπουδών μου.

Περίληψη

Για τη διεξαγωγή επιστημονικών υπολογισμών η χρήση υποσυστημάτων επιτάχυνσης υπολογισμών έχει γίνει απαραίτητη τα τελευταία χρόνια σε υπολογιστικά συστήματα υψηλών επιδόσεων. Τέτοια μηχανήματα συμπεριλαμβάνονται στη κορυφή της λίστας των ισχυρότερων υπολογιστικών συστημάτων του κόσμου για το 2013. Τα υποσυστήματα τους αυτά διαθέτουν πολλαπλούς υπολογιστικούς πυρήνες και αυτόνομη μνήμη κοινής διαχείρισης, ενώ η διασύνδεση τους με το κυρίως υπολογιστικό σύστημα επιτυγχάνεται μέσω διαύλων PCI. Αποτελούν συμπληρωματικές συσκευές επιτάχυνσης υπολογισμών και διακρίνονται σε δύο βασικές κατηγορίες, αυτή των γραφικών υποσυστημάτων (GPUs) και των πρόσθετων επεξεργαστικών υποσυστημάτων (coprocessors accelerators). Σε αυτή τη διατριβή κατασκευάστηκαν αποδοτικοί αλγόριθμοι για παράλληλες αρχιτεκτονικές υπολογισμών αριθμητικής επίλυσης μεγάλων και αραιών γραμμικών συστημάτων, τα οποία παράγονται από την εφαρμογή μεθόδων Πεπερασμένων Στοιχείων εφαρμογών για την επίλυση ελλειπτικών Προβλημάτων Συνοριακών Τιμών δεύτερης τάξης με σταθερούς συντελεστές. Η κατασκευή αλγορίθμων των επαναληπτικών μεθόδων του υπολοίπου Schur και Newton με χρήση διπλής /μικτής ακρίβειας υπολογισμών βασίστηκε στην κατάλληλη οργάνωση των υπολογισμών για παράλληλες αρχιτεκτονικές κοινής μνήμης πολυεπεξεργαστικών μηχανημάτων. Σε αυτά τα υπολογιστικά περιβάλλοντα γίνεται ταυτόχρονη χρήση υπολογιστικών πυρήνων από γραφικά υποσυστήματα (GPUs). Συγκεκριμένα, η υλοποίηση των αλγορίθμων πραγματοποιήθηκε για υπολογιστικά περιβάλλοντα κατανεμημένης και κοινής μνήμης σύμφωνα με τα πρότυπα MPI, OpenMP και OpenACC.

Η διατριβή είναι δομημένη σε έξι κεφάλαια. Το πρώτο κεφάλαιο αναφέρει σύντομα την ιστορική εξέλιξη των γραφικών υποσυστημάτων, καθώς και τα πλεονεκτήματα των υπολογιστικών δυνατοτήτων τους σε σχέση με την αποκλειστική χρήση CPU υπολογιστικών πυρήνων. Στη συνέχεια παρουσιάζεται η αρχιτεκτονική γραφικών υποσυστημάτων CUDA, και ειδικότερα η αρχιτεκτονική GPU τύπου Fermi. Στο δεύτερο κεφάλαιο παρουσιάζονται τα διαθέσιμα προγραμματιστικά εργαλεία ανάπτυξης παραλληλοποιήσιμων εφαρμογών που χρησιμοποιήθηκαν στις υλοποιήσεις της διατριβής. Αρχικά, παρουσιάζεται το πρότυπο ανταλλαγής μηνυμάτων για αρχιτεκτονικές κατανεμημένης μνήμης MPI, στη συνέχεια το πρότυπο παράλληλου προγραμματισμού αρχιτεκτονικών κοινής μνήμης OpenMP και στο τέλος το πρότυπο ανάπτυξης εφαρμογών με χρήση GPUs, OpenACC. Στο τρίτο κεφάλαιο παρουσιάζεται συνοπτικά η αριθμητική μέθοδος πεπερασμένων στοιχείων Hermite Collocation καθώς και η κατασκευή του παραγόμενου γραμμικού συστήματος. Στο τέταρτο κεφάλαιο περιγράφεται η κατασκευή αλγορίθμων για την εφαρμογή της μεθόδου Newton με χρήση διπλής και μικτής ακρίβειας υπολογισμών για την επίλυση του γραμμικού συστήματος. Στη συνέχεια, κατασκευάζονται αλγόριθμοι υλοποίησης της μεθόδου Schur Complement με την εφαρμογή της επαναληπτικής μεθόδου BiCGSTAB για αρχιτεκτονικές υπολογισμών κοινής/κατανεμημένης μνήμης που περιλαμβάνουν συνεργασίες υπολογισμών GPU-CPU. Το πέμπτο κεφάλαιο παρουσιάζει τη μελέτη της συμπεριφοράς των υλοποιήσεων των παραπάνω μεθόδων σε υπολογιστικά περιβάλλοντα κοινής/κατανεμημένης μνήμης με γραφικά υποσυστήματα. Στο τελευταίο κεφάλαιο αναφέρονται τα συμπεράσματα που προέκυψαν από τη μελέτη συμπεριφοράς των υλοποιήσεων των παράλληλων αλγορίθμων.

Στο παράρτημα παρατίθενται οι κώδικες των εφαρμογών που αναπτύχθηκαν στη παρούσα διατριβή, με χρήση της γλώσσας προγραμματισμού Fortran και των προτύπων MPI, OpenMP και OpenACC.

Η κατασκευή του παράλληλου αλγορίθμου καθώς και η υλοποίηση του για τη μέθοδο Schur Complement για υπολογιστικά περιβάλλοντα με γραφικά υποσυστήματα παρουσιάστηκε στο διεθνές συνέδριο ¨The 2013 International Conference of Parallel and Distributed Computing¨ το οποίο διεξήχθη 3-5 Ιουλίου 2013 στο Imperial College του Λονδίνου και η σχετική εργασία απέσπασε τη διάκριση *Best Paper Award of The 2013 International Conference of Parallel and Distributed Computing*. Η επέκταση του αλγορίθμου για αρχιτεκτονικές υπολογισμών με χρήση πολλαπλών γραφικών υποσυστημάτων παρουσιάστηκε στο διεθνές συνέδριο ¨International Conference on Mathematical Modeling in Physical Sciences¨, το οποίο διεξήχθη 1-5 Σεπτεμβρίου 2013 στη Πράγα και η αντίστοιχη εργασία έχει γίνει δεκτή για δημοσίευση στο Journal of Physics: Conference Series.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

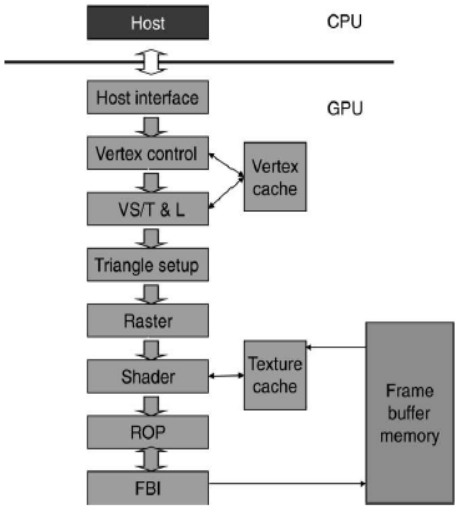
Γραφικά υποσυστήματα υπολογισμών

Γραφικό σύστημα υπολογισμών (GPU, Graphics Processor Unit) θεωρείται η συσκευή του υπολογιστικού συστήματος που χρησιμοποιείται για την επεξεργασία εικόνων, λαμβάνοντας δεδομένα (πολυγωνικά μοντέλα σε τρείς διαστάσεις) από την κεντρική μονάδα επεξεργασίας (CPU), τα οποία μετατρέπει σε εικόνα που στη συνέχεια προβάλεται στη συσκευή προβολής του μηχανήματος. Οι κάρτες γραφικών, όπως συνηθίζεται να αποκαλούνται οι GPUs, συνδέονται μέσω διαύλου PCI Express στη μητρική πλακέτα του υπολογιστή και έχουν, στη πλειοψηφία τους, αυτόνομη μνήμη (υπάρχουν κάρτες γραφικών που χρησιμοποιούν τμήματα της μνήμης του υπολογιστή, οι οποίες δε συμπεριλαμβάνονται στη κατηγορία των καρτών που μελετώνται στη παρούσα διατριβή εξαιτίας των περιορισμένων υπολογιστικών δυνατοτήτων τους). Τα τελευταία χρόνια οι GPUs εξέλιχθηκαν σε πανίσχυρες πολυπύρηνες υπολογιστικές μονάδες, με αποτέλεσμα να αυξηθεί κατακόρυφα και το ενδιαφέρον της επιστημονικής κοινότητας για ανάπτυξη λογισμικού ώστε να είναι δυνατή η διεξαγωγή υπολογισμών σε αυτές. Στη παρούσα διατριβή αναπτύχθηκαν εφαρμογές με χρήση καρτών γραφικών, οπότε στόχος αυτού του κεφαλαίου είναι, αφού γίνει μια σύντομη ιστορική αναδρομή, να παρουσιαστεί η τεχνολογία των GPUs και οι υπολογιστικές δυνατότητες αυτών που χρησιμοποιήθηκαν.

1.1 Χρονική εξέλιξη γραφικών υποσυστημάτων

Η κάρτα γραφικών εξελίχθηκε σταδιακά τα τελευταία είκοσι περίπου χρόνια. Αρχικά, τη γραφική αναπαράσταση σε ένα υπολογιστή αναλάμβαναν συσκευές τύπου CGA(Colour Graphics Adapter), EGA(Enhanced Graphics Adapter) και, αργότερα, VGA(Video Graphics Array Controller) με περιορισμένες δυνατότητες εμφάνισης χρωμάτων και ανάλυσης. Στις αρχές της δεκαετίας του $'90$, εμφανίζονται οι πρώτες ρυθμιζόμενες, αλλά μη−προγραμματιζόμενες κάρτες γραφικών με ξεχωριστούς υπολογιστικούς πυρήνες για τα επιμέρους τμήματα της ροής επεξεργασίας γραφικών, [7]. Ταυτόχρονα αναπτύσσονται και οι πρώτες διεπαφές προγραμματισμού για γραφικά, όπως το DirectX της Microsoft και το OpenGL της SGI. Στα τέλη της δεκαετίας, με την τεχνολογία των ημιαγωγών να εξελίσσεται διαρκώς, κατέστη δυνατόν να ενσωματωθούν όλοι οι πυρήνες σε ένα ενιαίο επεξεργαστή, τον shader, ο οποίος πλέον μπορούσε να εφαρμόσει ολοκληρωμένα τα βήματα της ροής επεξεργασίας γραφικών τριών διαστάσεων (graphics pipeline): μετασχηματισμούς, πλαισιοποίηση, υφή, δοκιμή βάθους και απεικόνιση χωρίς να υπάρχει μεταφορά δεδομένων από ένα πυρήνα σε άλλο, [62]. Η διαδικασία αυτή αποτελείται από επιμέρους βήματα, που περιγράφονται στο επόμενο σχήμα. Απο τις πρώτες κάρτες οι οποίες μπορούσαν να αναλάβουν εξόλοκλήρου τη ροή επεξεργασίας γραφικών ήταν η GeForce256 της εταιρείας NVIDIA, η οποία χαρακτηρίστηκε από την εταιρεία και ως η πρώτη GPU, όρος που καθιερώθηκε στη συνέχεια για να χαρακτηρίζει ολοκληρωμένες συσκευές επεξεργασίας γραφικών.

Στις GPUs που κατασκευάστηκαν στη συνέχεια, χαρακτηριστική ήταν η διαρκής αύξηση του αριθμού των υπολογιστικών πυρήνων ώστε να είναι αποδοτική η πολλαπλή ροή επεξεργασίας γραφικών, ενώ για πρώτη φορά ήταν δυνατή και η επέμβαση, προγραμματιστικά, σε συγκεκριμένα τμήματα της ροής επεξεργασίας, με τη GPU GeForce 7800. Με την αύξηση του αριθμού των πυρήνων σε μια GPU, βελτιωνόταν η ταχύτητα επεξεργασίας των γραφικών και κατ′επέκτασιν η απόδοση τους στην οθόνη, αφού αυξανόταν ο αριθμός των πολυγώνων που μπορούσαν να επεξεργαστούν παράλληλα και ανεξάρτητα (ίδιες εν-



Σχήμα 1.1: Ροή επεξεργασίας γραφικών σε GPU της εταιρείας NVIDIA, [40]. Αρχικά, η κάρτα γραφικών υποδέχεται τα δεδομένα σε μορφή πολυγώνων από το CPU, ο χειριστής κορυφών (vertex control) επεξεργάζεται τα πολύγωνα αλλάζοντας τους θέσεις και συντεταγμένες. Στη συνέχεια τα πολύγωνα αντιστοιχίζονται σε εικονοστοιχεία (pixels), καθορίζεται ο χρωματισμός τους, η διαφάνεια και το βάθος και προβάλλονται στην οθόνη.

τολές επεξεργασίας σε διαφορετικά δεδομένα−SIMD) στους πυρήνες της. Παρατηρώντας όμως ότι η ροή επεξεργασίας γραφικών δεν απέχει, όσον αφορά τη ροή των πληροφοριών, σε σχέση με έναν αλγόριθμο επαναλαμβανόμενων εκτελέσεων, έγινε αντιληπτό ότι οι πυρήνες των GPUs θα μπορούσαν να χρησιμοποιηθούν και για τη διεξαγωγή επιστημονικών υπολογισμών. Η χρήση GPUs σε τέτοιες εφαρμογές περιγράφεται διεθνώς με τον όρο GPGPUs, (General Purpose computation on Graphics Processor Units), [22].

Στην εξάπλωση και την πλήρη εκμετάλλευση τέτοιων αρχιτεκτονικών, ανασταλτικό παράγοντα αποτέλεσαν οι περιορισμοί του διαθέσιμου λογισμικού. Η συνήθης πρακτική ήθελε το προγραμματιστή να διαμορφώνει το προς επίλυση πρόβλημα σε πρόβλημα αναπαράστασης γραφικών. Για παράδειγμα, για την επίλυση ένός προβλήματος αεροδυναμικής δύο διαστάσεων, το πλέγμα που έπρεπε να χρησιμοποιηθεί θα είχε διάσταση τη μέγιστη ανάλυση της οθόνης (π.χ. 1024x768) και κάθε εικονοστοιχείο θα αντιστοιχούσε σε ένα κόμβο του πλέγματος, ενώ οι μεταβλητές σε κάθε κόμβο θα καθόριζαν το χρωματισμό του εικονοστοιχείου στη βάση RGBA (**R**ed**G**reen**B**lue**A**lpha), όπου 'R' η πυκνότητα, 'G', 'B' οι συνιστώσες της ταχύτητας και 'A' η πίεση στο κόμβο. Συνεπώς, η ανάπτυξη ενός

Σχήμα 1.2: Η πρώτη κάρτα γραφικών που χαρακτηρίστηκε ως GPU, GeForce 256 της εταιρείας NVIDIA. Ήταν η πρώτη κάρτα στην οποία γεωμετρικοί μετασχηματισμοί, επεξεργασία φωτισμού, παρεμβολή χρωματισμών εικονοστοιχείων και λοιπές λειτουργίες εκτελούνταν στον ίδιο υπολογιστικό πυρήνα, [62].

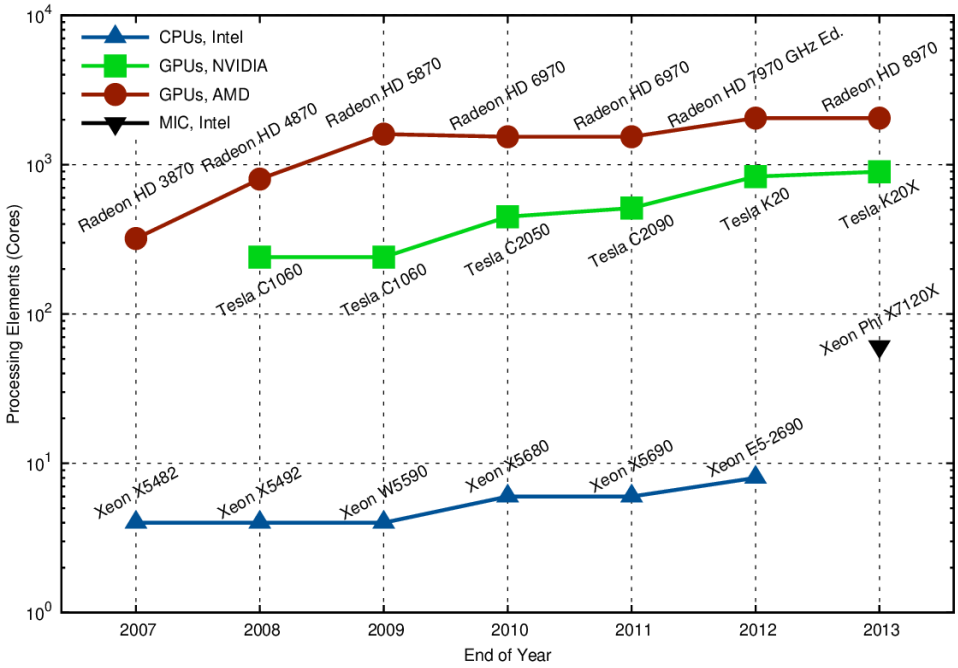
κώδικα σε GPU, έστω και χαμηλής παράλληλης απόδοσης, απαιτούσε εξαιρετική γνώση του τρόπου λειτουργίας της κάρτας, [55]. Τα τελευταία χρόνια αναπτύχθηκαν δεκάδες γλώσσες προγραμματισμού και προγραμματιστικά περιβάλλοντα, με χρήση των οποίων μπορεί ο προγραμματιστής να αναπτύξει εφαρμογές που εκτελούνται σε ετερογενή συστήματα GPU−CPU, χωρίς να απαιτείται εξαιρετική γνώση του τρόπου λειτουργίας της GPU. Το πιο χαρακτηριστικό προγραμματιστικό περιβάλλον είναι η CUDA της εταιρείας NVIDIA, που παρουσιάστηκε για πρώτη φορά με τη κάρτα Geforce 8800 το 2006, [30], και στηρίζεται στην ομώνυμη αρχιτεκτονική των GPUs της εταιρείας, ενώ επίσης διαδεδομένο είναι και το, μεταγενέστερο, OpenCL, [25], που αναπτύχθηκε από την εταιρεία Khronos και παρουσιάστηκε το 2010. Το πλέον πρόσφατο προγραμματιστικό πρότυπο για GPUs, που χρησιμοποιήθηκε και στην ανάπτυξη εφαρμογών της παρούσας διατριβής είναι το OpenACC, [33].

Σήμερα, οι GPUs που κυκλοφορούν στην αγορά μπορούν να μετατρέψουν ένα οποιοδήποτε υπολογιστή με κατάλληλη μητρική κάρτα και ικανή πηγή τροφοδοσίας σε ένα σύγχρονο παράλληλο υπολογιστικό σύστημα, με δυσανάλογα μικρό κόστος. Αποτέλε-

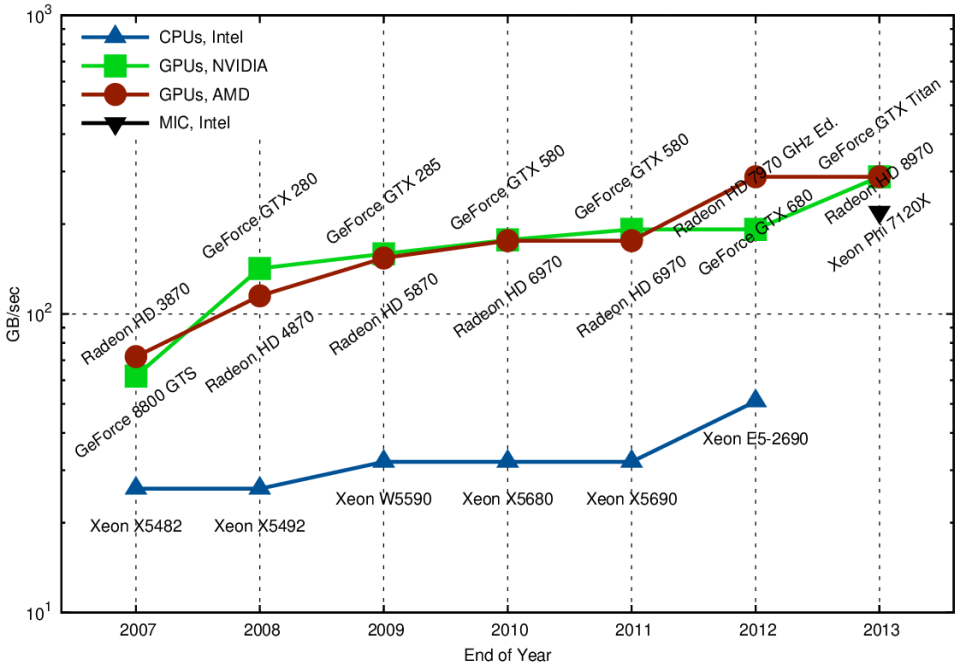
σμα είναι η συνεχής εξάπλωση της χρήσης τους και η διαρκής τεχνολογική τους εξέλιξη. Ακαδημαϊκά και εμπορικά έχουν αναπτυχθεί εκατοντάδες εφαρμογές που κάνουν χρήση των υπολογιστικών πυρήνων των GPUs, σε τομείς όπως υπολογιστική μηχανική και βιολογία, πρόγνωση κλιματικών φαινομένων, υπολογιστικά οικονομικά και άλλους, και παρέχουν αξιόλογη επιτάχυνση στην εκτέλεση τους σε σχέση με το CPU της τάξης του $1.1 - 400$x, [32].

1.2 Υπολογιστικές δυνατότητες GPU-CPU

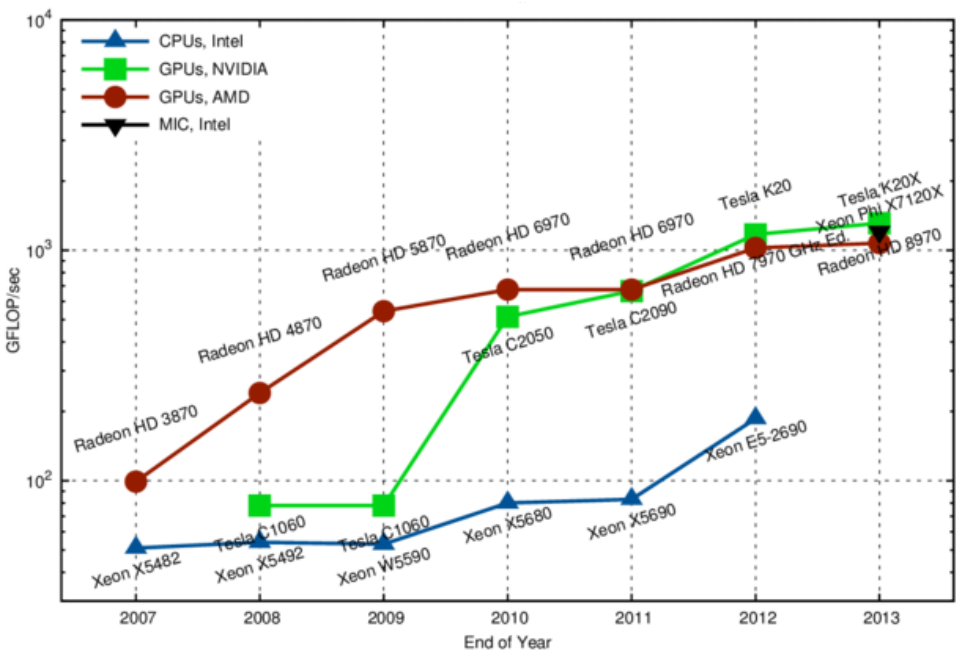
Την τελευταία δεκαετία παρατηρείται ραγδαία αύξηση της υπολογιστικής ισχύος των GPUs, που συνεπάγεται την ταυτόχρονη αύξηση της χρήσης τους για την επίλυση επιστημονικών προβλημάτων. Είναι χαρακτηριστικό ότι την περίοδο συγγραφής της διατριβής, η εξειδικευμένη για επιστημονικούς υπολογισμούς κάρτα γραφικών Tesla K20X της εταιρείας NVIDIA αποτελείται, μεταξύ άλλων, από 2688 υπολογιστικούς πυρήνες με δυνατότητα εκτέλεσης 1.31 Tflops πράξεων διπλής ακρίβειας και 3.95 Tflops πράξεων απλής ακρίβειας, διαθέτει 6 GB μνήμης και μέγιστη ταχύτητα προσπέλασης μνήμης 250 GB/sec[31]. Αντίθετα, ο εκ των κορυφαίων της σημερινής αγοράς CPU, Intel Xeon E5-2690 αποτελείται από 8 υπολογιστικούς πυρήνες, διαθέτει 384 GB μνήμης, μνήμη cache 20MB, μέγιστη ταχύτητα προσπέλασης μνήμης 51.2 GB/sec και δυνατότητα εκτέλεσης 2.8 Gflops πράξεων διπλής ακρίβειας, [23]. Το υπολογιστικό σύστημα του ΕΕ-ΜΗΥ, μοντέλο HP SL390s διαθέτει κάρτες Tesla M2070 με 448 υπολογιστικούς πυρήνες, δυνατότητα εκτέλεσης 515 Gflops πράξεων διπλής ακρίβειας και 1.03 Tflops πράξεων απλής ακρίβειας, 6 GB τοπικής μνήμης και μέγιστη ταχύτητα προσπέλασης μνήμης 150 GB/sec. Τα σχήματα παρουσιάζουν την αύξηση του αριθμού των πυρήνων, της μέγιστης ταχύτητας προσπέλασης μνήμης και της ταχύτητας αριθμητικών πράξεων απλής και διπλής ακρίβειας σε CPUs και GPUs τη τελευταία δεκαετία.



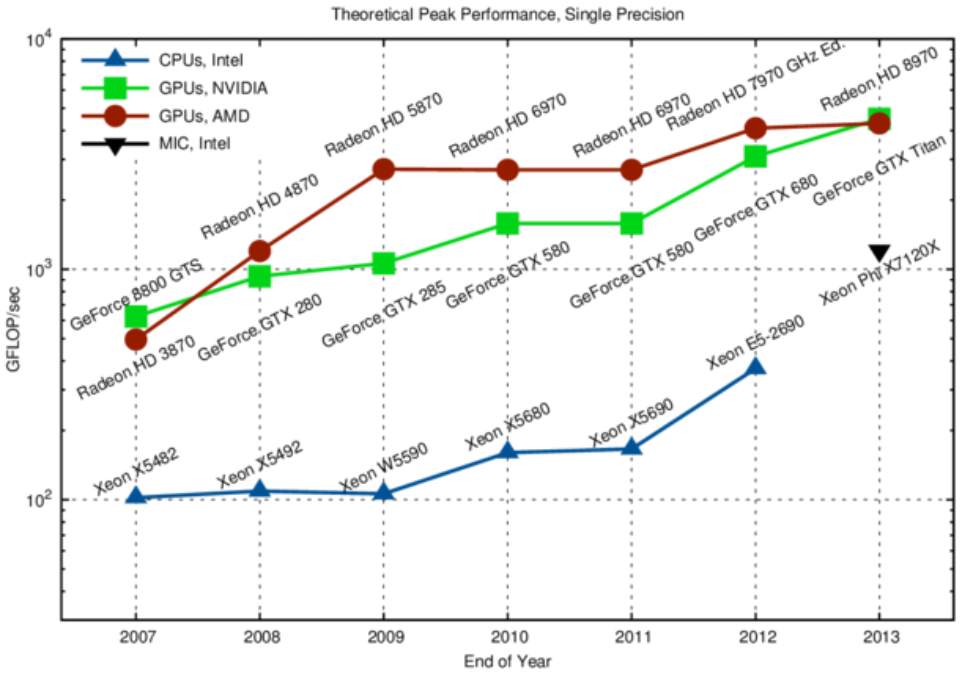
Σχήμα 1.3: Αριθμός πυρήνων GPU-CPU,[24].



Σχήμα 1.4: Σύγκριση θεωρητικής μέγιστης ταχύτητας προσπέλασης μνήμης GPU-CPU,[24].

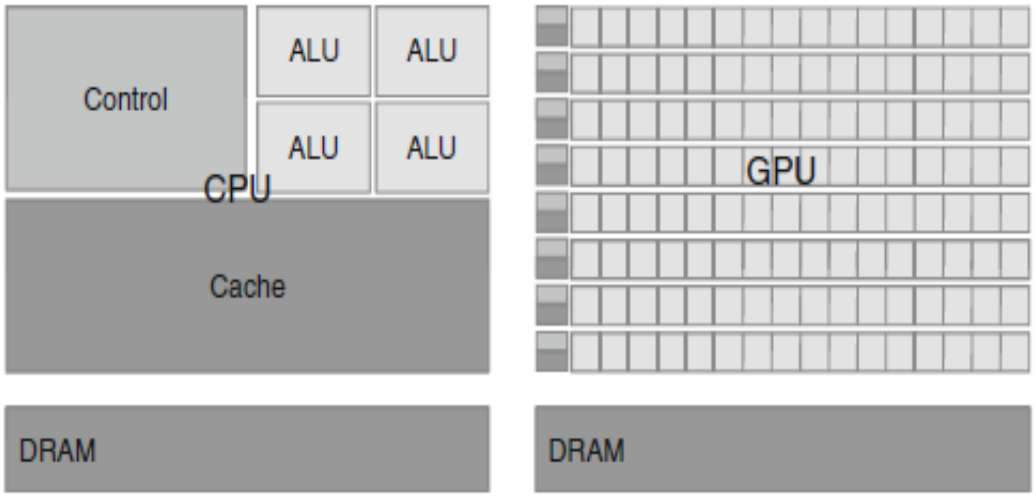


Σχήμα 1.5: Σύγκριση θεωρητικής μέγιστης ταχύτητας αριθμητικών πράξεων διπλής ακρίβειας GPU-CPU, [24].



Σχήμα 1.6: Σύγκριση θεωρητικής μέγιστης ταχύτητας αριθμητικών πράξεων απλής ακρίβειας GPU-CPU, [24].

Η μεγάλη διαφορά στη ταχύτητα εκτέλεσης αριθμητικών πράξεων μεταξύ μιας GPU νέας γενιάς και ενός πολυπύρηνου CPU, στην οποία έγινε αναφορά παραπάνω, οφείλεται στη διαφορετική σχεδιαστική φιλοσοφία που διέπει τα δύο υπολογιστικά συστήματα. Η εξέλιξη της αρχιτεκτονικής των CPUs στηρίχθηκε στη βασική λειτουργία τους σε ένα υπολογιστή, δηλαδή τη διαχείριση του υλικού του υπολογιστή και στην εκτέλεση εντολών από χρήστη και λειτουργικό σύστημα. Για το λόγο αυτό έχουν πολλά ολοκληρωμένα κυκλώματα αφιερωμένα στον ελέγχο ροής και οργάνωσης των δεδομένων, λίγους αλλά γρήγορους πυρήνες για την εκτέλεση απλών (πράξεις λογικής, κινητής υποδιαστολής) και σύνθετων οδηγιών (εκτέλεση out of order για την αποφυγή καθυστερήσεων στην ανάγνωση/εγγραφή δεδομένων, πρόγνωση διακλαδώσεων, branch prediction, εκτέλεσης) και, επιπλέον, σημαντικού μεγέθους πολυεπίπεδη cache μνήμη. Οι πυρήνες ενός CPU ακολουθούν τη παράλληλη αρχιτεκτονική MIMD (Multiple Instruction Multiple Data), μπορούν δηλαδή να απασχολούνται με διαφορετικές ομάδες δεδομένων εκτελώντας διαφορετικές λειτουργίες με αυτές, [51].



Σχήμα 1.7: Αρχιτεκτονικός σχεδιασμός CPU και GPU, [40].

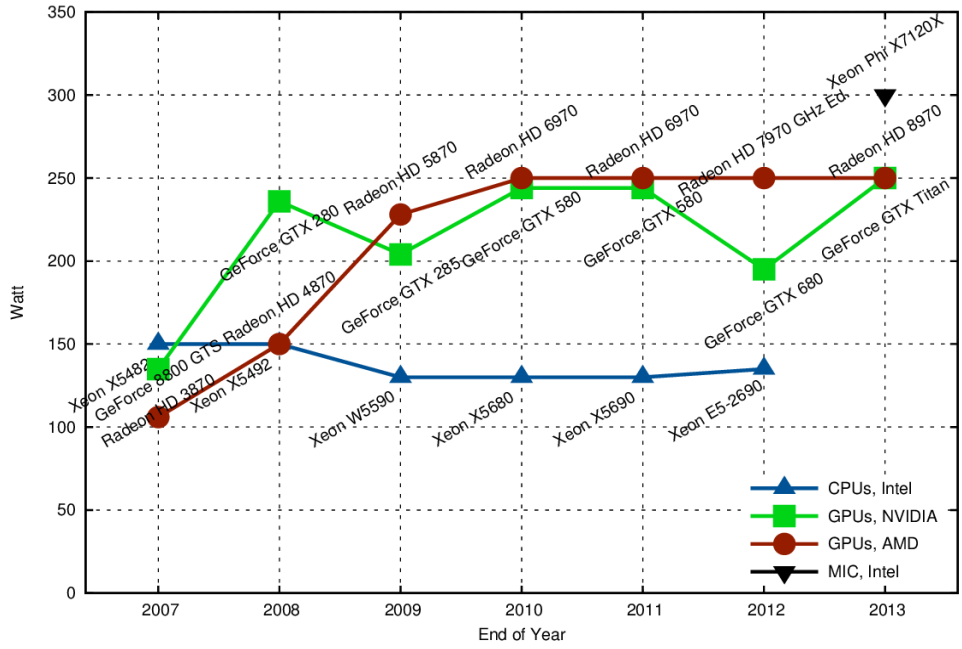
Αντίθετα, τα κυκλώματα μιας GPU είναι αφιερωμένα σχεδόν αποκλειστικά στην ε-

κτέλεση αριθμητικών πράξεων κινητής υποδιαστολής, όπως αυτό προκύπτει και από τη θεμελιώδη ανάγκη για επεξεργασία γραφικών και τη επακόλουθη σχεδίαση της συσκευής. Το μέγεθος της cache μνήμης είναι ελάχιστο, ενώ οι πυρήνες εκτελούν εντολές σύμφωνα με το πρότυπο SIMD.

Συμπερασματικά, GPUs και CPUs έχουν χαρακτηριστικές διαφορές στον αρχιτεκτονικό τους σχεδιασμό, που κάνουν το κάθε μηχάνημα καταλληλότερο από το άλλο, για διαφορετικές εργασίες. Οι GPUs αποτελούν υποσυστήματα υπολογιστικών μηχανημάτων, κατάλληλα να αναλάβουν αυξημένου υπολογιστικού κόστους διεργασίες ενός προγράμματος, όπου απαιτείται βασική επεξεργασία των δεδομένων, τα οποία πρέπει να μπορούν να εκτελεστούν παράλληλα. Ο CPU είναι διαχειριστής στις διαφορετικές εργασίες που πρέπει να εκτελεστούν σε ένα πρόγραμμα, αναλαμβάνοντας να ενορχηστρώσει, να στείλει τα *επιβαρυμένα* τμήματα προς εκτέλεση, να λάβει πίσω τα αποτελέσματα και να αξιολογήσει, [40]. Συνεπώς, γίνεται αντιληπτό ότι ενδείκνυται προγραμματιστικά η ανάπτυξη εφαρμογών όπως της παρούσας διατριβής, όπου χρησιμοποιούνται υβριδικά συστήματα CPU−GPU και, τη σειριακή εκτέλεση αναλαμβάνει ο CPU ενώ τα απαιτητικά υπολογιστικά μέρη του κώδικα, που μπορούν να παραλληλοποιηθούν, η GPU[55]. Το πρότυπο ανάπτυξης εφαρμογών OpenACC που παρουσιάζεται στο επόμενο κεφάλαιο, έχει σχεδιαστεί, ώστε να υποστηρίζει την από κοινού εκτέλεση σε αυτά τα ετερογενή συστήματα CPU−GPU.

Επικοινωνία CPU−GPU Το κόστος επικοινωνίας μεταξύ CPU και GPU είναι μια καθοριστική παράμετρος στην ανάπτυξη εφαρμογών σε τέτοια ετερογενή συστήματα. Καθώς η μέγιστη ταχύτητα προσπέλασης της μνήμης του CPU από τη μνήμη της GPU είναι μόλις 6 GB/sec, γίνεται αντιληπτό ότι η χρονική επιβάρυνση αυξάνεται σημαντικά όσο αυξάνεται και ο αριθμός των προσβάσεων στη κύρια μνήμη της GPU. Η ελαχιστοποίηση της μεταφοράς δεδομένων από και πρός τη GPU σε συνδυασμό με τη μεγαλύτερη δυνατή διάσταση προβλήματος, έτσι ώστε να αυξάνεται το πλήθος των προς εκτέλεση πράξεων,

είναι βασικές προυποθέσεις για ανάπτυξη GPU κώδικα υψηλής απόδοσης.



Σχήμα 1.8: Συγκριτικό διάγραμμα μέγιστης κατανάλωσης ισχύος GPU/CPU, [24].

Κατανάλωση ισχύος CPU−GPU Ένας από τους βασικούς λόγους που οδήγησαν στην προώθηση του παράλληλου προγραμματισμού ήταν οι περιορισμοί στην κατανάλωση ισχύος ενός υπολογιστικού συστήματος. Σήμερα, όλες οι αρχιτεκτονικές GPUs περιορίζουν την κατανάλωση ισχύος στα 300 Watt περίπου (μέγιστη τιμή ισχύος, με μέση τιμή σε κατάσταση αδράνειας τα 180 W περίπου) με βάση το κανόνα μέτρησης TDP (**T**hermal **D**esign **P**ower). Πρέπει βέβαια να σημειωθεί ότι ο TDP αναφέρεται στη μέγιστη ισχύ που θα χρειαστεί το μηχάνημα ψύξης για να απαγάγει τη θερμότητα που εκλείει ένα μηχάνημα, ώστε να το διατηρήσει σε λειτουργική κατάσταση, και όχι στη ακριβή κατανάλωση ισχύος του μηχανήματος. Οι GPUs γενικά χρησιμοποιούν τη διπλάσια TDP σε σχέση με ένα CPU, όπως φαίνεται και στο σχήμα 1.2. Για το λόγο αυτό, σε εφαρμογές επιστημονικών υπολογισμών, οι compute GPUs που δεν έχουν επιφορτιστεί με εκτέλεση υπολογισμών ή δε συμμετέχουν στην υλοποίηση, παραμένουν σε κατάσταση αδράνειας ώστε να εξοικονομείται ισχύς για άλλους πόρους του συστήματος. Η ενεργοποίηση και η

έναρξη της συμμετοχής τους στην εκτέλεση έχει χρονικό κόστος, το οποίο πρέπει να συνυπολογίζεται σε περιπτώσεις που οι μετρήσεις των χρονικών αποτελεσμάτων πρέπει να είναι ακριβείς. Με χρήση του προτύπου OpenACC ενδείκνυται η χρήση της διαδικασίας !$acc_init.

1.3 Ακρίβεια αριθμητικών πράξεων σε GPUs

Όπως αναφέρθηκε στην προηγούμενη ενότητα, οι πρώτες προγραμματιζόμενες κάρτες γραφικών δεν υποστήριζαν την αριθμητική διπλής ακρίβειας, ούτε ενσωμάτωναν πλήρως το πρότυπο IEEE 754 για την αποθήκευση και αναπαράσταση πραγματικών αριθμών. Στο πλαίσιο χρήσης μιας GPU με απλή ακρίβεια, όπως αποδεικνύεται και στην εργασία [55], τα αποτελέσματα υστερούν αισθητά σε σχέση ακόμα και με αποτελέσματα ενός κώδικα σε σύγχρονο CPU. Αργότερα, σε μεταγενέστερες GPU που υποστήριζαν αριθμητική διπλής ακρίβειας, η ταχύτητα εκτέλεσης τέτοιων πράξεων ήταν περίπου στο $1/8$ αυτών μεταξύ πραγματικών απλής ακρίβειας. Αυτό αποτελούσε τροχοπέδη για την ανάπτυξη εφαρμογών που θα εκτελούνταν σε GPUs αλλά θα απαιτούσαν υψηλή ακρίβεια αποτελεσμάτων. Έτσι, βρήκε πρόσφορο έδαφος η χρήση μικτής ακρίβειας (mixed precision method) στους υπολογισμούς της επίλυσης ενός προβλήματος, σε ένα ετερογενές σύστημα GPU−CPU, με εφαρμογή της διόρθωσης υπολοίπου (error correction). Η βασική ιδέα στην υλοποίηση της μεθόδου μικτής ακρίβειας είναι να χρησιμοποιηθεί διαφορετική ακρίβεια σε διαφορετικά μέρη του αλγορίθμου. Συγκεκριμένα ο GPU χρησιμοποιεί αριθμητική απλής ακρίβειας, ενώ ο CPU αριθμητική διπλής ακρίβειας, με σκοπό να επιτευχθεί καλύτερη χρονική απόδοση με εξίσου ακριβή αποτελέσματα, σε σχέση με την εκτέλεση αποκλειστικά σε CPU. Στη παρούσα διατριβή γίνεται χρήση της μεθόδου μικτής ακρίβειας στην υλοποίηση της μεθόδου πεπερασμένων στοιχείων collocation.

1.4 Η αρχιτεκτονική CUDA

Η αρχιτεκτονική CUDA (Compute Unified Device Architecture) αποτελεί το πρότυπο κατασκευής και προγραμματιστικό περιβάλλον καρτών γραφικών της εταιρείας NVIDIA. Παρουσιάστηκε για πρώτη φορά το 2006 στη κάρτα GeForce 8800 της εταιρείας και χαρακτηρίστηκε ως το σημαντικότερο βήμα στη μετάβαση στην εποχή των (GPGPU). Οι συσκευές που υποστηρίζουν τη CUDA είναι πλήρως διαμορφώσιμες μέσω του περιβάλλοντος εκτέλεσης με χρήση βιβλιοθηκών CUDA, όπως cuBLAS [16], CULA [21], MAGMA [19], IMSL FNL [39], cuSPARSE [17], με χρήση κατάλληλων οδηγιών στο μεταγλωτιστή όπως το πρότυπο OpenACC ή με κατάλληλες επεκτάσεις σε γλώσσες προγραμματισμού C, C++ και Fortran. Υπενθυμίζεται ότι στην ανάπτυξη εφαρμογών της παρούσας διατριβής, για τη διαμόρφωση των επιλογών εκτέλεσης στις κάρτες γραφικών χρησιμοποιήθηκε το πρότυπο OpenACC, [33], με το μεταγλωτιστή pgf90 της εταιρείας PGI, [36], θυγατρικής εταιρείας της NVIDIA.

Στη συνέχεια ακολουθεί συνοπτική παρουσίαση του τρόπου λειτουργίας της CUDA, ενώ περιγράφεται η αρχιτεκτονική καρτών γραφικών Fermi της εταιρείας NVIDIA, που υποστηρίζεται από τη CUDA.

1.4.1 Βασικές έννοιες

Η αρχιτεκτονική CUDA στηρίζεται σε ένα σύνολο υπολογιστικών πολυεπεξεργαστών, που αποτελούνται από δεκάδες πυρήνες, και κάθε ένας αναλαμβάνει να εκτελεί ακολουθίες εντολών, τις οποίες απαιτεί ένα πρόγραμμα CUDA. Οι ακολουθίες αυτές εκτελούνται πολλές φορές κατά τη διάρκεια ενός προγράμματος, με διαφορετικές ομάδες δεδομένων και, κατ'επέκταση, διαφορετικές ομάδες αποτελεσμάτων. Κάθε τέτοια ακολουθία εντολών αναφέρεται ως **νήμα** ή **thread** και ορίζεται ως η βασική μονάδα επεξεργασίας. Κάθε κάρτα γραφικών CUDA, έχει τη δυνατότητα να δημιουργεί και να χειρίζεται ένα αριθμό από threads, ανάλογα την υπολογιστική της δυνατότητα, στα οποία ανα-

θέτει την εκτέλεση μιας διεργασίας—μέρος του κυρίως προγράμματος, που ονομάζεται **kernel**(υπολογιστικός πυρήνας).

Κάθε πολυεπεξεργαστής μπορεί να εκτελεί ταυτόχρονα εκατοντάδες threads, έτσι ώστε κάθε kernel να εκτελείται από μια ομάδα από threads, με κάθε thread να διαχειρίζεται διαφορετικό όγκο δεδομένων. Για τη διαχείριση των threads εφαρμόζεται η λογική **SIMT** (Single Instruction Multiple Thread), σύμφωνα με την οποία πολλαπλά threads εκτελούν την ίδια λειτουργία με διαφορετικά δεδομένα την ίδια χρονική στιγμή. Η SIMT λογική είναι παραπλήσια της ευρύτερα γνωστής SIMD (Single Instruction Multiple Data).

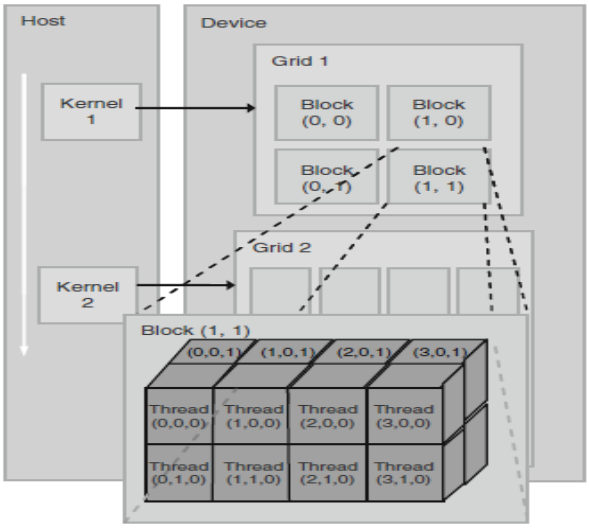
Συμπερασματικά, η εκτέλεση ενός τυπικού CUDA προγράμματος ξεκινά με την εκτέλεση του κώδικα από το CPU. Όταν ζητηθεί η εκτέλεση ενός kernel, η εκτέλεση θα συνεχιστεί στη GPU. Εκεί, θα δημιουργηθεί ένας αριθμός από threads που αφού οργανωθούν θα εκτελέσουν παράλληλα τις εντολές που έχουν ζητηθεί. Όταν όλα ολοκληρωθούν, ο kernel θα τερματιστεί και η εκτέλεση θα συνεχιστεί στο CPU, μέχρι να ζητηθεί η εκτέλεση ενός νέου kernel.

1.4.2 Οργάνωση των υπολογιστικών νημάτων στην αρχιτεκτονική CUDA

Όταν ζητηθεί η εκτέλεση ενός kernel, τα threads που δημιουργούνται και εκτελούνται στους πολυεπεξεργαστές δεν κατανέμονται τυχαία. Αντίθετα, η διαδικασία και ο τρόπος οργάνωσης τους περιλαμβάνει αρκετά επίπεδα ομαδοποίησης. Αρχικά, τα threads οργανώνονται σε επιμέρους ομάδες, τα **blocks**. Ανάλογα την υπολογιστική δυνατότητα της GPU, ένα block έχει ανώτερο όριο από threads που μπορούν να το συνθέτουν. Στις κάρτες υπολογιστικής δυνατότητας $2.x$ για παράδειγμα, κάθε block μπορεί να αποτελείται από το πολύ 1024 threads. Κάθε πολυεπεξεργαστής μπορεί να εκτελέσει ταυτόχρονα μέχρι 8 blocks, ανάλογα τις ανάγκες του kernel σε registers και κοινή (shared) μνήμη. Σε περίπτωση που σε κάποια χρονική στιγμή υπάρχει μεγαλύτερος αριθμός blocks προς εκτέλεση, τα πλεονάζοντα blocks περιμένουν την ολοκλήρωση της εκτέλεσης άλλων, παραμένοντας ανενεργά. Δημιουργείται έτσι μια σειριακή διαδικασία ομάδων από

blocks.

Στη συνέχεια, τα blocks που δημιουργούνται σε ένα kernel οργανώνονται σε ένα ευρύτερο υπολογιστικό σύνολο, που ονομάζεται πλέγμα ή **grid**, όπως παρουσιάζεται στο σχήμα 1.9. Κάθε thread χαρακτηρίζεται από έναν αύξοντα αριθμό (threadIdx) που δείχνει που βρίσκεται μέσα στο block, και κάθε block αντίστοιχα από ένα αριθμό blockIdx, για να προσδιορίζεται η θέση του μέσα στο grid. Για παράδειγμα, σε ένα grid με 64 blocks όπου κάθε block έχει 32 threads, υπάρχουν συνολικά 2048 threads. Το thread 4 στο block 0 θα έχει τον αύξοντα αριθμό $0 * 32 + 4 = 4$, ενώ το thread 14 στο block 27 θα έχει τον αύξοντα αριθμό $27 * 32 + 14 = 878$. Γενικά, τα blocks διατάσσονται σε 2 διαστάσεις μέσα στο grid και τα threads σε 2 (ή 3 σε Fermi) διαστάσεις μέσα σε κάθε block. Ο προγραμματιστής έχει τη δυνατότητα να επιλέξει τον τρόπο διάταξης, ανάλογα το πρόβλημα που επιλύεται, [40, 54, 55].



Σχήμα 1.9: Παράδειγμα πολυδιάστατης οργάνωσης threads σε CUDA kernel, [40].

Η πολυεπίπεδη οργάνωση των εκτελούμενων threads σε ένα πρόγραμμα CUDA και η δυνατότητα διαμόρφωσης του αριθμού τους σε κάθε επίπεδο, εξυπηρετεί συγκεκριμένους σκοπούς. Ειδικότερα, με αυτό τον τρόπο διάταξης, κάθε thread σχετίζεται με συγκεκριμένες θέσεις μνήμης και το πρόγραμμα γνωρίζει σε ποιό thread απευθύνεται για μεταφορά δεδομένων, εκτέλεση, κ.ο.κ. Επιπλέον, τα threads που ανήκουν στο ίδιο

block χρησιμοποιούν τη κοινή μνήμη του block, η οποία είναι ταχείας προσπέλασης, πετυχαίνοντας μεγαλύτερη επιτάχυνση στην εκτέλεση τους. Όπως σημειώθηκε, ο προγραμματιστής θα πρέπει να επιλέξει το κατάλληλο πλήθος από threads σε block σε κάθε kernel του CUDA προγράμματος, ανάλογα πάντα τη φύση του προς επίλυση προβλήματος και δεδομένου του ανώτερου ορίου που έχει κάθε κάρτα. Στην ανάπτυξη εφαρμογών αυτής της διατριβής, χρησιμοποιήθηκε το πρότυπο OpenACC, στο οποίο, όπως περιγράφεται στη συνέχεια, την επιλογή του αριθμού των εκτελούμενων threads αναλαμβάνει ο μεταγλωττιστής, με δυνατότητα διαμόρφωσης από το χρήστη, αν θεωρείται χρήσιμο.

1.4.3 Πολυεπεξεργαστές της κάρτας γραφικών

Η CUDA στηρίζεται ουσιαστικά στους πολυεπεξεργαστές της κάρτας γραφικών, που αναλαμβάνουν να εκτελέσουν ομάδες από εκατοντάδες threads ο καθένας, ταυτόχρονα. Κάθε πολυεπεξεργαστής (SM, Streaming Multiprocessor) αποτελείται από ένα αριθμό υπολογιστικών πυρήνων, ειδικές μονάδες μνήμης και ελέγχου ροής, όπως παρουσιάζονται παρακάτω, για την αρχιτεκτονική τύπου Fermi.

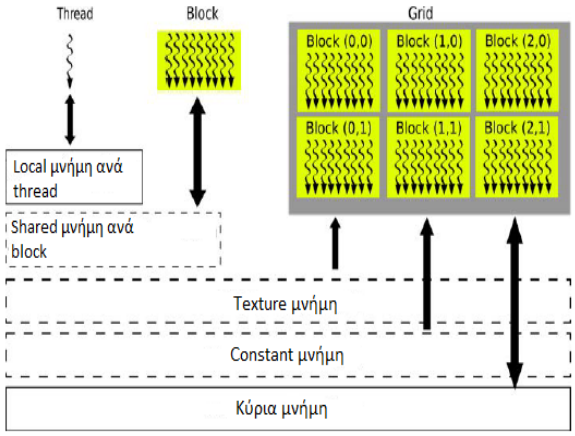
Η διαδρομή εκτέλεσης στον πολυεπεξεργαστή ξεκινά με την ομαδοποίηση των threads. Συγκεκριμένα, όταν δοθούν ένα ή περισσότερα blocks από threads προς εκτέλεση, ο πολυεπεξεργαστής τα ομαδοποιεί σε ομάδες των 32 threads που ονομάζονται **warps** (ή στημόνια) και προγραμματίζονται για εκτέλεση από κατάλληλες μονάδες χειρισμού (warp schedulers). Για να χωριστεί ένα block σε warp, ακολουθείται σταθερή διαδικασία. Για να συμπληρωθεί ένα warp περιλαμβάνει διαδοχικά τα threads με τους αντίστοιχους αύξοντες αριθμούς, π.χ. στο warp 0 περιλαμβάνονται τα threads από 0 ώς 31, κ.ο.κ. Τα threads ενός warp εκτελούν την ίδια σειρά εντολών. Προφανώς, βέλτιστη απόδοση ενός kernel επιτυγχάνεται όταν και τα 32 threads ενός warp συμφωνούν στην πορεία εκτέλεσης τους.

1.4.4 Μνήμη σε κάρτες γραφικών

Σε μια κάρτα γραφικών βρίσκονται ενσωματωμένες τα ακόλουθα είδη μνήμης

- *Κύρια μνήμη*
- *Shared μνήμη*
- *Constant μνήμη*
- *Texture μνήμη*
- *Local μνήμη*
- *Καταχωρητές (registers)*

Όπως παρουσιάζεται και στο σχήμα 1.10, πρόσβαση σε κάθε είδος μνήμης έχουν συγκεκριμένα στοιχεία κατά την εκτέλεση.



Σχήμα 1.10: Δυνατότητα πρόσβασης σε κάθε μνήμη της GPU, [55].

Έτσι,

- στη κύρια μνήμη διαβάζει και γράφει το κάθε grid
- στη shared μνήμη διαβάζει και γράφει το κάθε block
- στη local μνήμη διαβάζει και γράφει το κάθε thread

- στην constant μνήμη μόνο διαβάζει το κάθε grid
- στην texture μνήμη μόνο διαβάζει το κάθε grid
- στους καταχωρητές (registers) διαβάζει και γράφει το κάθε thread

Κάθε μια από τις παραπάνω μνήμες της κάρτας γραφικών έχει διαφορετικά χαρακτηριστικά σε σχέση με τις άλλες. Η κύρια μνήμη είναι η μεγαλύτερη σε μέγεθος μνήμη της κάρτας γραφικών. Έχουν πρόσβαση σε αυτή όλα τα threads, αλλά καθότι δεν αποτελεί μνήμη cache, η συνεχής πρόσβαση σε αυτή κοστίζει χρονικά. Οι μνήμες texture και constant είναι μνήμες cache, ταχείας προσπέλασης, που βρίσκονται στον ίδιο χώρο με τη κύρια μνήμη. Η πρόσβαση στη constant μνήμη και η ανάγνωση κάθε thread απο αυτή είναι ταχύτατη (΄κοστίζει΄ χρονικά όσο και η ανάγνωση ενός καταχωρητή). Η texture μνήμη είναι σχεδιασμένη ώστε ΄γειτονικά΄ threads (για παράδειγμα, τα threads ενός warp) να διαβάζουν από αυτή ΄γειτονικές΄ θέσεις μνήμης. Η shared μνήμη είναι επίσης μνήμη ταχείας προσπέλασης, βρίσκεται στο chip των πολυεπεξεργαστών και εξυπηρετεί την επικοινωνία μεταξύ των threads ενός block. Με τον όρο local (τοπική) μνήμη, εννοείται ένα μέρος της κύριας μνήμης που αντιστοιχεί σε κάθε thread. Χρησιμοποιείται από τη κάρτα όταν ένα kernel απαιτεί περισσότερους καταχωρητές από τους διαθέσιμους ενός πολυεπεξεργαστή. Επειδή αποτελεί μέρος της κύριας μνήμης, η προσπέλαση της χαρακτηρίζεται από υψηλούς χρόνους.

Γίνεται αντιληπτό, ότι κατά την εκτέλεση ενός kernel η προσπέλαση της κύριας μνήμης πρέπει να περιορίζεται όσο περισσότερο γίνεται, λόγω του χρονικού κόστους προσπέλασης. Αντίθετα, αύξηση της χρήσης των μνημών cache της κάρτας γραφικών μπορεί να αυξήσει και την απόδοση της εκτέλεσης του προγράμματος. Βέβαια, το μικρό μέγεθος των συγκεκριμένων μνημών (η Tesla M2070 αρχιτεκτονικής Fermi, έχει $128\ KB$ cache μνήμες), συνδέει άμεσα τη διαχείριση της μνήμης της κάρτας γραφικών σε ένα πρόγραμμα με την βελτιστοποίηση της απόδοσης του.

1.4.5 Αρχιτεκτονική των γραφικών υποσυστημάτων τύπου Fermi

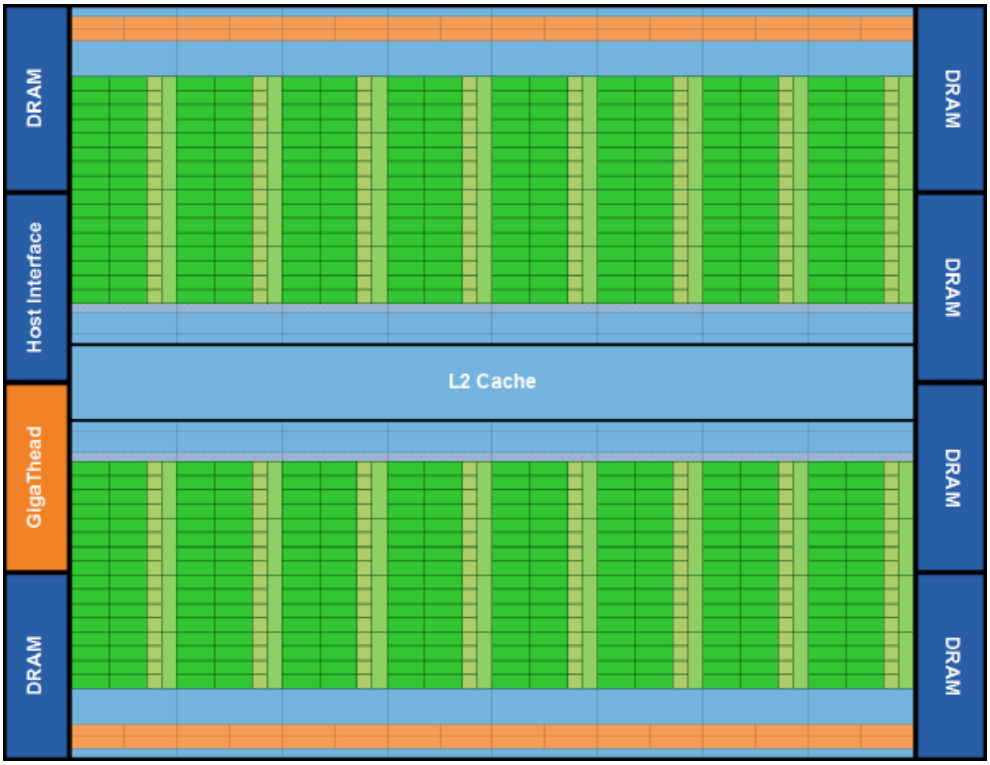
Η πρώτη σειρά καρτών γραφικών αρχιτεκτονικής Fermi, [50],• από την εταιρεία NVIDIA κυκλοφόρησε το 2010, με το μοντέλο GeForce 400. Η καινοτομία στο σχεδιασμό, η αυξημένη υπολογιστική δυνατότητα και ένα πλήθος επιπρόσθετων λειτουργιών σε σχέση με προηγούμενες σειρές καρτών, κατέστησε τις κάρτες γραφικών τύπου Fermi σημείο αναφοράς τόσο σε επίπεδο επιστημονικών υπολογισμών με τις σειρές Quadro και Tesla, όσο και σε επίπεδο ευρύτερης χρήσης με άλλες σειρές.

Μια κάρτα αρχιτεκτονικής Fermi αποτελείται από 512 (ή 768) CUDA υπολογιστικούς πυρήνες, οργανωμένους σε 16 πολυεπεξεργαστές (SM's, Streaming Multiprocessors), από 32 (ή 48) πυρήνες ο καθένας. Ο αριθμός των πυρήνων σε κάθε πολυεπεξεργαστή εξαρτάται από την υπολογιστική δυνατότητα της κάρτας γραφικών. Οι πολυεπεξεργαστές βρίσκονται περιμετρικά και υποστηρίζονται από μια κοινή L2 cache μνήμη, η οποία επιταχύνει την προσπέλαση της κύριας μνήμης. Επιπλέον, όπως παρουσιάζεται και στην ακόλουθη σχηματική αναπαράσταση, η κάρτα διαθέτει ένα δίαυλο επικοινωνίας (Host Interface) για τη μεταφορά δεδομένων από και πρός τον κεντρικό υπολογιστή, ένα προγραμματιστή (Gigathread) που διανέμει τις ομάδες υπολογιστικών νημάτων στους πολυεπεξεργαστές και 6 64-bit τομείς μνήμης DRAM με συνολική κύρια μνήμη 6 GB.

Πρέπει να αναφερθεί ότι σε αρκετά μοντέλα αρχιτεκτονικής Fermi, η εταιρεία NVIDIA έχει απενεργοποιήσει ένα αριθμό πολυεπεξεργαστών για πρακτικούς λόγους (κατανάλωση ισχύος, μείωση θερμότητας). Για παράδειγμα η σειρά Tesla έχει ενεργούς μόνο τους 14 από τους 16 πολυεπεξεργαστές.

Πολυεπεξεργαστής σε κάρτα Tesla αρχιτεκτονικής Fermi

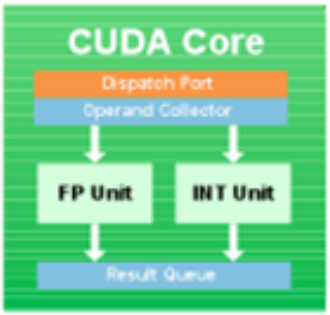
Κάθε πολυεπεξεργαστής σε μια κάρτα Tesla τύπου Fermi[50], υπολογιστικής δυνατότητας 2.0 αποτελείται από:



Σχήμα 1.11: Σχηματική αναπαράσταση μιας κάρτας γραφικών αρχιτεκτονικής Fermi. Οι 16 πολυεπεξεργαστές βρίσκονται περιμετρικά της κοινής L2 cache μνήμης. Κάθε πολυπεξεργαστής αποτελείται από 32 CUDA πυρήνες (με πράσινο χρώμα), 2 μονάδες χειρισμού warps (με πορτοκαλί χρώμα), μνήμες και registers (με γαλάζιο χρώμα), [50].

- 32 υπολογιστικούς πυρήνες CUDA
- 4 ειδικές μονάδες εκτέλεσης μαθηματικών συναρτήσεων (SFU)
- 2 μονάδες χειρισμού warp (warp schedulers)
- 1 constant cache μνήμη
- 1 texture cache μνήμη
- 1 L1 cache μνήμη
- 1 shared μνήμη
- 16 μονάδες load-store
- 32768 32-bit καταχωρητές (registers)

Κάθε υπολογιστικός πυρήνας CUDA διαθέτει μια μονάδα FPU (Floating Point Unit) και μια μονάδα ALU (Arithmetic Logic Unit), με τις οποίες μπορεί να εκτελέσει πράξεις κινητής υποδιαστολής (floating point) και ακεραίων (integer) αντίστοιχα. Στις κάρτες



Σχήμα 1.12: Σχηματική αναπαράσταση ενός υπολογιστικού πυρήνα CUDA, [50].

τύπου Fermi, η μονάδα AL υποστηρίζει τις συνήθεις λογικές και μαθηματικές πράξεις με ακρίβεια 32 και 64 bit για λειτουργίες όπως μετατροπή, σύγκριση και αλλαγή. Επιπρόσθετα, στη μονάδα FP οι πράξεις κινητής υποδιαστολής υποστηρίζουν πλέον το πρότυπο IEEE 754-2008. Οι αριθμοί που βρίσκονται πολύ κοντά στο 0 (subnormal numbers), ώστε να μην μπορούν να αναπαρασταθούν σε κανονικοποιημένη μορφή, δεν αντικαθίστανται με 0, αλλά τυγχάνουν ορθότερου χειρισμού, ενώ εξασφαλίζονται 4 τρόποι στρογγυλοποίησης. Επίσης, εισάγεται η λειτουργία FMA (Fused Multiply-ADD), μια βελτιωμένη έκδοση της λειτουργίας MAD (Multiply-Addition), που παρέχει υψηλότερη ακρίβεια στα αριθμητικά αποτελέσματα προσθέσεων−πολλαπλασιασμών.

Οι τέσσερις ειδικές μονάδες εκτέλεσης μαθηματικών συναρτήσεων (SFU) του πολυεπεξεργαστή εκτελούν υπερβατικές οδηγίες όπως τριγωνομετρικές, εκθετικές και αντίστροφες. Κάθε ειδική μονάδα εκτελεί μια οδηγία ανά thread σε κάθε κύκλο του ρολογιού, συνεπώς για την εκτέλεση ενός warp απαιτούνται 8 κύκλοι.

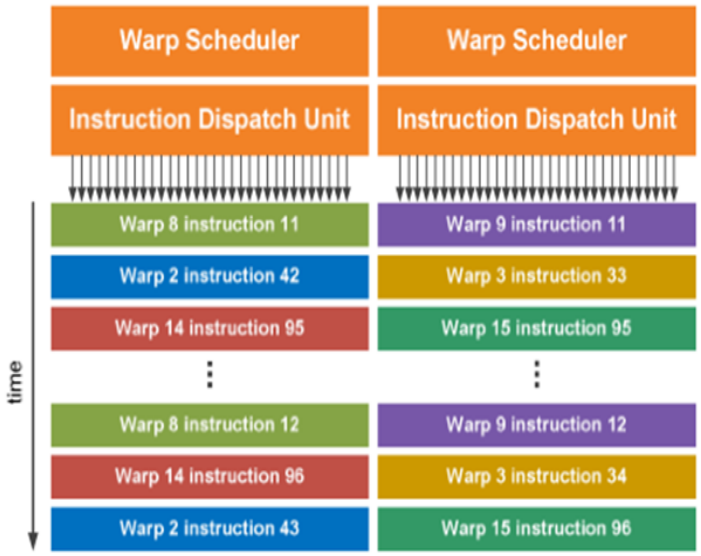
Κάθε πολυεπεξεργαστής διαθέτει επίσης 2 μονάδες χειρισμού των warps, που ορίζουν στα warps τις εργασίες που θα εκτελέσουν. Τα warps, αφού συμπληρωθούν με 32 threads, παίρνουν ένα αύξοντα αριθμό (warp 0, warp 1, κ.ο.κ). Οι 32 υπολογιστικοί

Σχήμα 1.13: Σχηματική αναπαράσταση της λειτουργίας FMA σε σχέση με τη λειτουργία MAD, σε Fermi-based GPU, [50].

πυρήνες χωρίζονται σε 2 ομάδες των 16. Η πρώτη μονάδα αναλαμβάνει να αναθέσει τα warps με άρτιο αριθμό στη μια ομάδα υπολογιστικών πυρήνων ή στις ειδικές μονάδες μαθηματικών συναρτήσεων ή στις μονάδες load-store, ενώ η δεύτερη αναθέτει τα warps με περιττό αριθμό αντίστοιχα.

Οι δύο μονάδες εκτελούν τα warps ανεξάρτητα στις περισσότερες περιπτώσεις. Για παράδειγμα, μπορούν να εκτελεστούν ταυτόχρονα δύο αριθμητικές πράξεις μεταξύ ακεραίων αριθμών, δύο απλής ακρίβειας, συνδυασμός απλής ακρίβειας και SFU, κ.ο.κ. Σε περίπτωση όμως που ένα warp πρέπει να πραγματοποιήσει αριθμητικές πράξεις με πραγματικούς διπλής ακρίβειας η μια μονάδα δεσμεύει το σύνολο των πυρήνων του πολυεπεξεργαστή, αφήνοντας ελεύθερες για χρήση απο άλλα warps τις SFU και τη δυνατότητα προσπέλασης της μνήμης.

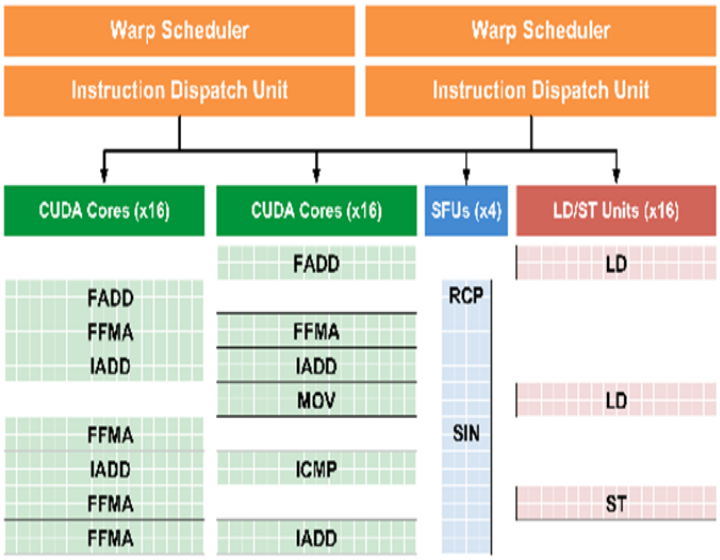
Γίνεται αντιληπτό ότι η εκτέλεση αριθμητικών πράξεων με διπλής ακρίβειας πραγματικούς αριθμούς πραγματοποιείται με τη μισή ταχύτητα σε σχέση με την εκτέλεση πράξεων με αριθμούς απλής ακρίβειας, αφού χρησιμοποιούνται οι μισοί πυρήνες κάθε πολυεπεξεργαστή. Συγκεκριμένα, στη κάρτα Tesla M2070 η ταχύτητα εκτέλεσης πρά-



Σχήμα 1.14: Η μια μονάδα χειρισμού των warps αναλαμβάνει την ανάθεση αυτών με άρτιο αριθμό στους υπολογιστικούς πυρήνες και η άλλη αυτών με περιττό αριθμό, [50].

ξεων διπλής ακρίβειας φτάνει τα 515 Gflops ενώ σε απλή ακρίβεια τα 1.03 Tflops. Σε προηγούμενες αρχιτεκτονικές (π.χ. GT200), η αντίστοιχη ταχύτητα διπλής ακρίβειας διαμορφωνόταν στα 86 Gflops, δίνοντας ένα λόγο ταχυτήτων εκτέλεσης περίπου ίσο με $1/12$, ενώ πλέον αντιστοιχεί σε $1/2$.

Οι 16 μονάδες load-store κάθε υπολογιστικού πυρήνα χειρίζονται τις λειτουργίες μνήμης. Η constant cache μνήμη και η texture cache μνήμη είναι ενδιάμεσες μνήμες που επιταχύνουν την προσπέλαση των constant και texture μνημών, που βρίσκονται μαζί με την κύρια μνήμη της κάρτας γραφικών. Το μέγεθος τους σε μια Tesla M2070 είναι $8KB$ και $8KB$ αντίστοιχα. Η shared και L1 cache βρίσκονται στον ίδιο χώρο του πολυεπεξεργαστή και αποτελούν την τοπική του μνήμη. Η shared μνήμη παρέχει τη δυνατότητα πρόσβασης σε ενδιάμεσα αποτελέσματα υπολογισμών και μικρές ποσότητες δεδομένων, ενώ η L1 cache μνήμη χρησιμοποιείται για άμεση προσπέλαση θέσεων της κύριας μνήμης. Στις Fermi-based κάρτες η τοπική μνήμη είναι 64 KB και ο προγραμματιστής έχει τη δυνατότητα να επιλέξει ρύθμιση μεταξύ των 16KB/48KB ή 48KB/16KB για shared και L1 cache αντίστοιχα. Η επιλογή εξαρτάται από τις απαιτήσεις σε shared μνήμη ενός



Σχήμα 1.15: Η εκτέλεση των warps γίνεται ταυτόχρονα, εκτός της περίπτωσης όπου εκτελούνται πράξεις αριθμητικής διπλής ακρίβειας οπότε δεσμεύεται το σύνολο των πυρήνων του πολυπεξεργαστή, [50].

kernel και τη συχνότητα πρόσβασης του στη κύρια μνήμη (ώστε να είναι αποδοτική η αύξηση του μεγέθους της L1 cache). Σημειώνεται ότι σε κάρτες υπολογιστικής δυνατότητας 2.0, δίνεται η δυνατότητα στον προγραμματιστή να ζητήσει τη μη χρησιμοποίηση της L1 cache μνήμης[50]. Τέλος, κάθε πολυεπεξεργαστής διαθέτει 32768 καταχωρητές των 32-bit. Η ανάγκη κάθε kernel σε καταχωρητές διαμορφώνει τον αριθμό των threads σε κάθε block όπως περιγράφεται σε επόμενη παράγραφο.

Συνοψίζοντας, οι κάρτες αρχιτεκτονικής Fermi παρέχουν τη δυνατότητα εκτέλεσης αριθμητικών πράξεων διπλής ακρίβειας με μεγαλύτερη ταχύτητα σε σχέση με αυτές προηγούμενων αρχιτεκτονικών, που είναι απαραίτητη για ακριβείς υπολογισμούς σε διάφορες επιστημονικές εφαρμογές. Επιπλέον, η κύρια μνήμη της κάρτας υποστηρίζεται από τις ενδιάμεσες μνήμες επιπέδου L1, L2 cache που επιταχύνουν την πρόσβαση σε αυτήν.

Κεφάλαιο 2

Πρότυπα ανάπτυξης παραλληλοποιήσιμων εφαρμογών

Η αρχιτεκτονική των σημερινών υπερυπολογιστικών συστημάτων είναι βασισμένη σε ένα εξειδικευμένο ταχύτατο δίκτυο διασύνδεσης εκατοντάδων ή και χιλιάδων υπολογιστικών κόμβων[38]. Κάθε υπολογιστικός κόμβος αποτελείται από ένα πολυεπεξεργαστικό σύστημα, το οποίο μπορεί να διαθέτει και ένα μονοψήφιο αριθμό γραφικών υποσυστημάτων. Άρα το μοντέλο ανάπτυξης των εφαρμογών για τη πλήρη εκμετάλλευση ενός τέτοιου υπολογιστικού μηχανήματος πρέπει να βασιστεί σε αυτό της κοινής-κατανεμημένης μνήμης. Αυτό συμβαίνει, διότι οι υπολογιστικοί κόμβοι διαθέτουν αποκλειστικής χρήσης μνήμη, η οποία είναι φυσικά κατανεμημένη στο κάθε κόμβο, ενώ θεωρείται κοινή για όλους τους επεξεργαστές του. Οπότε κατά την ανάπτυξη των εφαρμογών θα πρέπει ο χρήστης να εφαρμόσει ένα πρότυπο ανταλλαγής πληροφοριών-μηνυμάτων μεταξύ των υπολογιστικών κόμβων, ενώ για καθένα από αυτούς η διαχείριση μνήμης/υπολογιστικών πυρήνων είναι διαφορετική. Τα πρότυπα ανάπτυξης εφαρμογών, τα οποία έχουν καθιερωθεί μέχρι σήμερα είναι το MPI [26, 27, 35] για τη πρώτη περίπτωση, ενώ για τη δεύτερη το OpenMP [34]. Για τη περίπτωση χρήσης γραφικών υποσυτημάτων κατά την ανάπτυξη εφαρμογών είναι εφικτή επιπλέον η χρήση των γλωσσών προγραμματισμού CUDA [30], OpenCL [25] ή του προτύπου OpenACC [33]. Η υλοποίηση των αριθμητικών μεθόδων της παρούσας διατριβής πραγματοποιήθηκε σε γλώσσα Fortran με χρήση των MPI, OpenMP και OpenACC προτύπων. Στη συνέχεια παρουσιάζονται τα τρία πρότυπα και

ιδιαίτερα τα υποπρογράμματα και οι απαιτούμενες ρυθμίσεις εκτέλεσης των εφαρμογών που χρησιμοποιήθηκαν.

2.1 Το πρότυπο MPI

2.1.1 Εισαγωγή

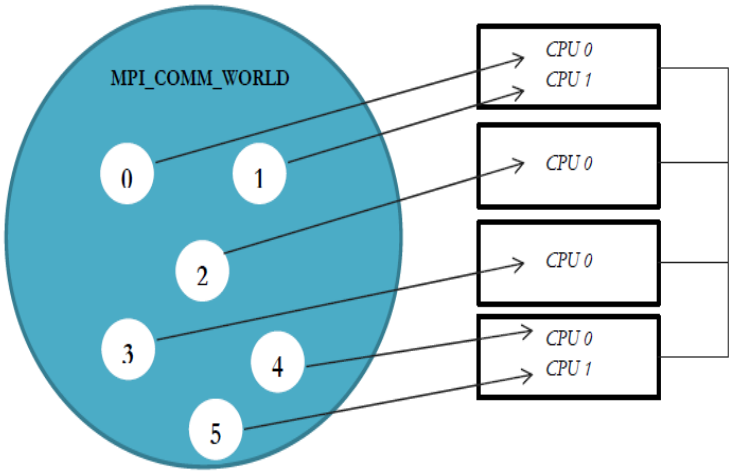
Το πρότυπο διεπαφής ανταλλαγής μηνυμάτων MPI (Message Passing Interface) είναι μια βιβλιοθήκη συναρτήσεων και υποπρογραμμάτων, μέσω των οποίων είναι εφικτή η διαχείριση της επικοινωνίας μεταξύ επεξεργαστών σε υπολογιστικά συστήματα παράλληλης αρχιτεκτονικής. Αποτελείται από ένα πρωτόκολλο επικοινωνίας που υποστηρίζει θύρα προς θύρα (point−to−point) και συλλογική (collective) επικοινωνία σε επεξεργαστές οργανωμένους και συνδεδεμένους, έτσι ώστε να θεωρούνται ως ένα ενιαίο σύστημα (computer cluster). Η κύρια βιβλιοθήκη του MPI περιλαμβάνει υποπρογράμματα αποστολής/λήψης δεδομένων (send/receive operations), διαμοιρασμού δεδομένων (scatter operation), συνδυασμού αποτελεσμάτων (gather/reduce operations), συγχρονισμού των πυρήνων (barrier) [26, 27, 35].

2.1.2 Μοντέλο Ανάπτυξης Εφαρμογών MPI

Κατά την ανάπτυξη εφαρμογών με χρήση του προτύπου MPI, στην εκτέλεση συμμετέχουν περισσότεροι από ένας επεξεργαστικοί−υπολογιστικοί πυρήνες, οι οποίοι επικοινωνούν καλώντας υποπρογραμμάτα του προτύπου, με αποστολή και λήψη μηνυμάτων. Ο αριθμός των υπολογιστικών πυρήνων που συμμετέχουν καθορίζεται στην αρχή του προγράμματος και καθένας αναλαμβάνει να εκτελέσει και μια ξεχωριστή διεργασία, ορίζοντας έτσι το προγραμματιστικό μοντέλο του MPI να είναι γενικά MIMD (Multiple Instructions Multiple Data) από το σύνηθες SIMD, όπου όλοι οι επεξεργαστές εκτελούν τις ίδιες διεργασίες σε διαφορετικά δεδομένα. Κάθε διεργασία χρησιμοποιεί το δικό της χώρο διευθύνσεων μνήμης για τα δεδομένα της, η μεταφορά των οποίων απο τη μια στην άλλη γίνεται όποτε καθοριστεί αναγκαίο από τον προγραμματιστή με ανταλλαγή μηνυ-

μάτων. Η ανταλλαγή είναι δυνατή μεταξύ δύο πυρήνων, όπου ο ένας λαμβάνει, ο άλλος αποστέλλει και ανάποδα (point−to−point, ή συλλογικά (collective) όταν ένας πυρήνας αποστέλλει ή λαμβάνει απο τους υπόλοιπους.

Την οργάνωση της επικοινωνίας μεταξύ των υπολογιστικών πυρήνων αναλαμβάνει ένας χειριστής (communicator), ο οποίος αρχικά αριθμεί τους πυρήνες και τους ταξινομεί. Ο αριθμός που αντιστοιχίζεται σε κάθε πυρήνα καλείται τάξη του (rank) στον συγκεκριμένο communicator και αποτελεί καθοριστική παράμετρο στην ανταλλαγή των μηνυμάτων. Όσοι πυρήνες ανήκουν στον ίδιο communicator μπορούν να επικοινωνούν μεταξύ τους ακόμα και αν ανήκουν σε διαφορετικά MPI προγράμματα. Ο εξ' ορισμού communicator στο MPI είναι ο `MPI_COMM_WORLD`, ο οποίος καλείται αυτόματα απο το MPI στην έναρξη του προγράμματος, και περιέχει όλους τους διαθέσιμους πυρήνες που δηλώνονται απο το χρήστη. Ο αριθμός τους, δηλαδή το μέγεθος του communicator επιστρέφεται από τη συνάρτηση `MPI_COMM_SIZE`. Στο σχήμα 2.1 εμφανίζεται ένα παράδειγμα ταξινόμησης απο δύο διπλοπύρηνα και δύο μονοεπεξεργαστικά υπολογιστικά συστήματα συνδεδεμένα σε ένα δίκτυο.



Σχήμα 2.1: Παράδειγμα ταξινόμησης επεξεργαστών ενός cluster στο MPI_COMM_WORLD

Για να χρησιμοποιηθεί το MPI σε ένα πρόγραμμα εντολών της γλώσσας προγραμματισμού Fortran πρέπει πρώτα να δηλωθεί η χρήση του με την εντολή `#include"mpi.h"`, ώστε να φορτωθεί η βιβλιοθήκη με όλα τα πρωτότυπα των συναρτήσεων και τις δομές των δεδομένων που χρειάζεται ένα MPI πρόγραμμα.

Για την εκκίνηση του προγράμματος είναι απαραίτητη η κλήση της συνάρτησης `MPI_Init`, η οποία ενεργοποιεί το πρότυπο, ενώ για τον τερματισμό του προγράμματος καλείται η `MPI_Finalize`, από όλες τις διεργασίες, η οποία απελευθερώνει όλους τους πόρους που είχαν δεσμευθεί για το πρότυπο.

Σχήμα 2.2: Σχηματική αναπαράσταση της δομής λειτουργίας του προτύπου MPI

2.1.3 Βασικά Υποπρογράμματα του προτύπου MPI

Στη συνέχεια παρουσιάζονται συνοπτικά τα κυριότερα υποπρογράμματα του MPI που χρησιμοποιήθηκαν στις αναπτύξεις εφαρμογών της παρούσας διατριβής. Η κλήση κάθε υποπρογράμματος απαιτεί συγκεκριμένες παραμέτρους για ορίσματα, των οποίων ο τύπος και η σειρά δήλωσης περιγράφονται αναλυτικά στο εγχειρίδιο του MPI [26, 27, 35].

- **MPI_SEND** Με κλήση της αποστέλλονται μέσω μηνύματος δεδομένα από ένα ή περισσότερους υπολογιστικούς πυρήνες. Το μήνυμα έχει συγκεκριμένη ταυτότητα−ετικέτα (tag) και απευθύνεται σε αντίστοιχο αριθμό πυρήνων, που έχουν οριστεί σε κατάσταση παραλαβής (με το MPI_RECV) και ανήκουν στον ίδιο communicator.

- **MPI_RECV** Με κλήση της ένας ή περισσότεροι επεξεργαστές λαμβάνουν το μήνυμα που απέστειλε αντίστοιχος αριθμός επεξεργαστών με την MPI_SEND.

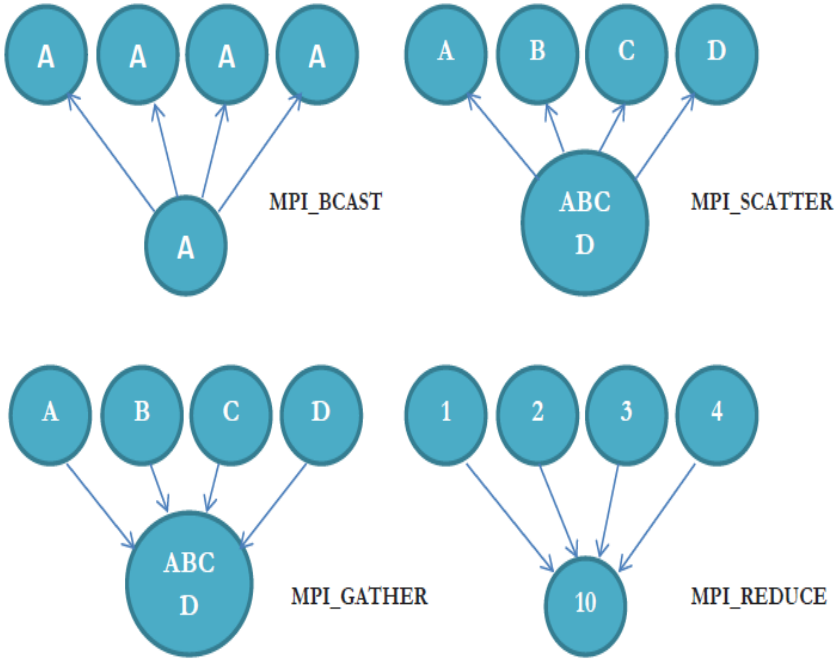
- **MPI_BCAST** Με την κλήση της ένας επεξεργαστής αναμεταδίδει μέσω μήνυματος δεδομένα προς όλους τους υπόλοιπους επεξεργαστές του ίδιου communicator, καθένας απο τους οποίους θα έχει πλέον ένα πλήρες αντίγραφο αυτών των δεδομένων.

- **MPI_GATHER** Υποδεικνύει σε ένα επεξεργαστή να συγκεντρώσει τιμές δεδομένων από τους υπόλοιπους, εντός ίδιου communicator σε μια εννιαία μορφή δεδομένων.

- **MPI_SCATTER** Υποδεικνύει σε ένα επεξεργαστή να μοιράσει τιμές δεδομένων στους υπόλοιπους επεξεργαστές της ομάδας. Τα δεδομένα, οι τιμές ενός διανύσματος για παράδειγμα, θα μοιραστούν ισομερώς στους υπόλοιπους επεξεργαστές ανάλογα το πλήθος τους, συνεπώς καθένας τους δεν γνωρίζει όλες τις τιμές αλλά το τμήμα που του αναλογεί.

- **MPI_REDUCE** Μετατρέπει τιμές δεδομένων σε κάθε επεξεργαστή σε μια μοναδική τιμή, εφαρμόζοντας κατάλληλες πράξεις όπως άθροισμα, γινόμενο, εύρεση μεγίστου, εύρεση ελαχίστου.

Το σχήμα 2.3 παρουσιάζει σχηματικά τη λειτουργία των βασικών υποπρογραμμάτων του MPI.



Σχήμα 2.3: Σχηματική αναπαράσταση υποπρογραμμάτων ανταλλαγής μηνυμάτων του προτύπου MPI

2.2 Το πρότυπο OpenMP

2.2.1 Εισαγωγή

Στην ενότητα αυτή θα παρουσιαστεί συνοπτικά το μοντέλο προτύπου OpenMP το οποίο χρησιμοποιήθηκε για την εκτέλεση εφαρμογών στην παρούσα διατριβή, σε επίπεδο host-only αλλά και ετερογενή (heterogenous) multi-GPU συστημάτων.

Το OpenMP (**Open** specifications for **MultiProcessing**) είναι μια διεπαφή προγραμματισμού εφαρμογών (API, Application Programming Interface) που παρέχει τη δυνατότητα παράλληλου προγραμματισμού σε αρχιτεκτονικές κοινής μνήμης. Είναι συμβατό με τις γλώσσες προγραμματισμού C/C++ και Fortran. Χαρακτηρίζεται από τα παρακάτω στοιχεία

- **Οδηγίες (Compiler Directives):** Σύνολο από οδηγίες προς τον μεταγλωττιστή (compiler), οι οποίες ενσωματώνονται στον κώδικα με τη μορφή σχολίων, και υποδεικνύουν στον compiler τις περιοχές παράλληλης εκτέλεσης, διαχείρισης δεδομέ-

νων, συγχρονισμού ή καταμερισμού εργασιών.

- **Συναρτήσεις Βιβλιοθήκης (Library Routines):** Συναρτήσεις βιβλιοθήκης με τη χρήση των οποίων διαμορφώνεται ανάλογα η παράλληλη εκτέλεση ή επιστρέφονται πληροφορίες σχετικές με τον τρόπο εκτέλεσης που έχει επιλεγεί.

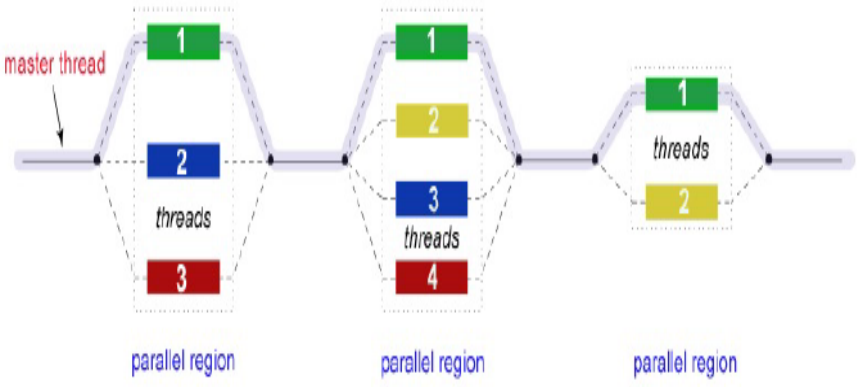
- **Μεταβλητές Περιβάλλοντος (Environment Variables):** Μεταβλητές περιβάλλοντος χρηστών για τον έλεγχο της παράλληλης εκτέλεσης στο περιβάλλον εργασίας.

Το OpenMP API δημιουργήθηκε με βάση την ανάγκη για ύπαρξη ενός κοινού προτύπου για τον παράλληλο προγραμματισμό σε αρχιτεκτονικές υπολογιστών κοινής μνήμης. Στον καθορισμό του συμμετέχουν εταιρείες κατασκευής υλικού και λογισμικού υπολογιστικών συστημάτων, ενώ στην καθολική αποδοχή του συνετέλεσε το γεγονός ότι παρείχε τη δυνατότητα της μερικής παραλληλοποίησης, σε αντίθεση με το πρότυπο MPI, όπου ο παραλληλισμός είναι ολοκληρωτικός ή μηδενικός. Η πρώτη έκδοση του προτύπου κυκλοφόρησε το 1997, ενώ η τρέχουσα έκδοση είναι η 3.1 [15, 34].

2.2.2 Μοντέλο Ανάπτυξης Εφαρμογών

Το μοντέλο παραλληλισμού που εφαρμόζεται είναι thread based, στηρίζεται δηλαδή στην ύπαρξη πολλών υπολογιστικών νημάτων (threads) που μοιράζονται τον ίδιο χώρο μνήμης. Ο αριθμός των threads καθορίζεται απο το χρήστη, ανάλογα τις δυνατότητες του μηχανήματος, με χρήση της κατάλληλης συνάρτησης βιβλιοθήκης ή θέτοντας την κατάλληλη μεταβλητή περιβάλλοντος του κελύφους. Το πρόγραμμα στο οποίο γίνεται χρήση του προτύπου OpenMP, ξεκινά να εκτελείται σειριακά από ένα thread (master thread) σαν μια διεργασία, μέχρις ότου αυτό συναντήσει οδηγία παράλληλης εκτέλεσης. Τότε το master thread δημιουργεί μια ομάδα από threads, με το πλήθος που όρισε ο προγραμματιστής, τα οποία εκτελούν παράλληλα τις εντολές που περιέχονται στην περιοχή αυτή του κώδικα. Το master thread αποτελεί επίσης μέλος της ομάδας υπολογισμών. Εντός της παραλληλοποιήσιμης περιοχής δεν υπάρχει συγχρονισμός μεταξύ threads, καθένα

μπορεί να φτάσει σε οποιοδήποτε σημείο της οποιαδήποτε χρονική στιγμή. Στο τέλος της περιοχής, έχει δημιουργηθεί ένα φράγμα (barrier) όπου τα threads συγχρονίζονται και αφήνουν τη συνέχεια της εκτέλεσης στο master thread, παραμένοντας ανενεργά μέχρι την επόμενη περιοχή παράλληλης εκτέλεσης[15]. Το μοντέλο αυτό εκτέλεσης είναι γνωστό και ως FORK–JOIN.



Σχήμα 2.4: Fork–Join

Οι περιοχές παραλληλοποίησης ορίζονται με τις κατάλληλες οδηγίες προς το μεταγλωττιστή, η σύνταξη των οποίων περιγράφεται στη συνέχεια.

2.2.3 Μοντέλο Διαχείρισης Μνήμης στο OpenMP

Το πρότυπο OpenMP είναι σχεδιασμένο, όπως προαναφέρθηκε, για πολυεπεξεργαστικά μηχανήματα με κοινή μνήμη. Όλα τα threads έχουν πρόσβαση στις θέσεις μνήμης, ενώ το καθένα μπορεί να έχει και μια αποκλειστική εικόνα μερους της, σε περίπτωση που τα δεδομένα σε αυτές τις θέσεις αφορούν κάθε thread ξεχωριστά. Κάθε μεταβλητή εντός παράλληλης περιοχής μπορεί να είναι κοινής(shared) ή αποκλειστικής(private) χρήσης, να είναι δηλαδή προσβάσιμη, είτε από όλα τα threads, είτε καθένα να έχει ξεχωριστό αντίγραφο αυτής της μεταβλητής. Εξ' ορισμού, εντός παράλληλων περιοχών οι μεταβλητές είναι κοινές για όλα threads. Ωστόσο, ο καταμερισμός αυτών των δεδομένων οδηγεί ορισμένες φορές σε ένα ιδιότυπο ανταγωνισμό μεταξύ των threads. Για τον έλεγχο και

την αποφυγή τέτοιας συμπεριφοράς, που οδηγεί συνήθως σε λάθος υλοποιήσεις, επιβάλλεται η οργάνωση των δεδομένων, όπως αλλαγή της εμβέλειας των μεταβλητών (από shared σε private), ή οδηγία συγχρονισμού μεταξύ των threads, με χρήση κατάλληλων οδηγιών(directives). Γενικότερα, σε μοντέλα ανάπτυξης εφαρμογών για αρχιτεκτονικές κοινής μνήμης, τα δύο βασικά ζητήματα της συμπεριφοράς της μνήμης του συστήματος είναι η συνάφεια (coherence) και η συνέπεια (consistency). Η συνάφεια αναφέρεται στη συμπεριφορά του συστήματος μνήμης, όταν πολλαπλά threads προσπαθούν να αποκτήσουν πρόσβαση σε μια συγκεκριμένη θέση μνήμης, ενώ η συνέπεια στην σειρά πρόσβασης που θα έχουν τα threads σε διαφορετικές θέσεις μνήμης. Το OpenMP, όπως αναφέρθηκε, δεν εξασφαλίζει την ορθότητα των αποτελεσμάτων σε περιπτώσεις ανταγωνισμού των threads για μια θέση μνήμης −συνάφεια− όμως εξασφαλίζει συγκεκριμένη συμπεριφορά συνέπειας, με χρήση κατάλληλων οδηγιών [15, 34].

2.2.4 Οδηγίες OpenMP

Η χρήση του προτύπου OpenMP μπορεί να γίνει σε γλώσσες προγραμματισμού C, C++ και Fortran. Η γενική σύνταξη μιας OpenMP directive γίνεται με διαφορετικό τρόπο στις C/C++ και Fortran. Οι παρακάτω directives που θα περιγραφούν, θα έχουν τη σύνταξη που απαιτείται στη γλώσσα Fortran και χρησιμοποιήθηκαν στην ανάπτυξη των εφαρμογών αυτής της διατριβής [15, 34].

Μια OpenMP directive εισάγεται σε κώδικα γλώσσας Fortran με τη μορφή

```
!$omp directive clause1(var1,var2,...), clause2(var3,...),...
```

Ο συμβολισμός `!$omp` γράφεται σαν μια λέξη, χωρίς κενά, στις στήλες $1-5$ του κώδικα, ακολουθούμενος από το όνομα της directive και, για κάποιες directives, τους **όρους** (clauses) με τις απαιτούμενες μεταβλητές (**var**iables). Σε περίπτωση που απαιτείται συνέχιση της directive στην επόμενη γραμμή, τον συμβολισμό `!$omp` ακολουθεί άμεσα

το συμπλεκτικό &. Οι directives μπορούν να υποδεικνύουν παράλληλες περιοχές, να ορίζουν την εμβέλεια των δεδομένων ή να συγχρονίζουν τα threads. Κάθε directive εφαρμόζεται μόνο στο τμήμα (block) εντολών που ακολουθεί, που πρέπει να είναι δομημένα τμήματα κώδικα, δηλαδή να έχουν ένα σημείο εισόδου αρχικά και ένα σημείο εξόδου στο τέλος.

Η οδηγία !$omp parallel

Η θεμελιώδης directive στο OpenMP είναι η !$omp parallel, με σύνταξη

```
!$omp parallel clause1(var1,var2,...), clause2(var3,...),...

...
δομημένο block κώδικα
...

!$omp end parallel
```

την οποία όταν συναντήσει το master thread, δημιουργεί την ομάδα των threads που εκτελούν παράλληλα τις ακολουθούμενες εντολές, όπως προαναφέρθηκε στο προγραμματιστικό μοντέλο. Η αρίθμηση των threads εντός παράλληλης περιοχής ξεκινά απο το 0, που αντιστοιχεί στον master thread, εώς και τον αριθμό $N-1$, με N αριθμό threads.

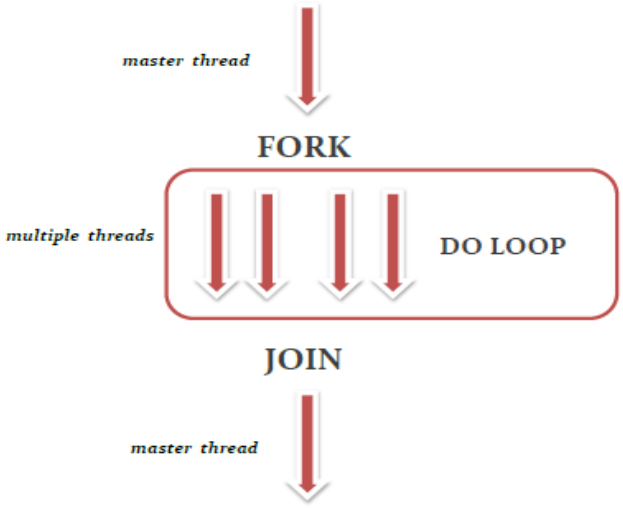
Η οδηγία !$omp do

Η οδηγία !$omp do είναι μια από τις directives διαχείρισης εργασιών (work-sharing loops) στο OpenMP. Συντάσσεται ως

```
!$omp do clause1(var1,var2,...), clause2(var3,...),...

...
δομημένο block κώδικα
...

!$omp end do
```



Σχήμα 2.5: Εκτέλεση do loop στο πρότυπο OpenMP

και υποδεικνύει ότι οι επαναλήψεις του βρόγχου εργασίας που ακολουθεί θα εκτελεστούν παράλληλα από τα υπολογιστικά threads. Στην έναρξη της περιοχής, τα threads εισέρχονται ακανόνιστα, ενώ στο τέλος της υπονοείται φράγμα αναμονής στο οποίο συγχρονίζονται όλα τα threads για την ολοκλήρωση της παράλληλης εργασίας.

Η οδηγία !$omp sections

Η οδηγία !$omp sections ανήκει και αυτή στην κατηγορία των οδηγιών διαχείρισης εργασιών (work-sharing loops directives) στο OpenMP. Συντάσσεται ως

```
!$omp sections clause1(var1,var2,...), clause2(var3,...),...

!$omp section
δομημένο block κώδικα
!$omp section
δομημένο block κώδικα

!$omp end sections
```

και υποδεικνύει ότι τα τμήματα του κώδικα που καθορίζονται από τη section οδηγία θα διαμοιραστούν και θα εκτελεστούν από τα threads της ομάδας υπολογισμών. Κάθε

section θα εκτελεστεί μια φορά από ένα thread. Διαφορετικά sections εκτελούνται απο διαφορετικά threads.

Όροι(Clauses)

Όροι που χρησιμοποιήθηκαν στη συγκεκριμένη διατριβή και μπορούν να χρησιμοποιηθούν στις παραπάνω directives είναι οι εξής

- `shared(var1,var2,..)` Οι τιμές των μεταβλητών που περιέχονται θα είναι κοινές για όλα τα threads (τα threads θα προσπελάσουν τις ίδιες θέσεις μνήμης)

- `private(var1,var2,..)` Κάθε thread θα έχει δικό του αντίγραφο για τις μεταβλητές που έχουν δηλωθεί στην συγκεκριμένη directive.

- `if (True—False)` Η παραλληλοποιήση θα εφαρμοστεί ή όχι, ανάλογα τη δεδομένη λογική έκφραση.

Η οδηγία !$omp barrier

Η !$omp barrier συγχρονίζει όλα τα threads της ομάδας σε δεδομένο σημείο του κώδικα. Συντάσσεται ως

```
!$omp barrier
```

και ουσιαστικά, το πρώτο thread που φτάνει σε αυτή την εντολή περιμένει όλα τα υπόλοιπα να φτάσουν και κατόπιν συνεχίζουν όλα μαζί την παράλληλη εκτέλεση που ακολουθεί.

2.2.5 Συναρτήσεις Βιβλιοθήκης

Οι συναρτήσεις βιβλιοθήκης παρέχουν λειτουργίες σχετικές με την εκτέλεση του προγράμματος όπως τον ορισμό του αριθμού των threads, πληροφορίες σχετικά με τον αριθμό των threads ή τον αριθμό των διαθέσιμων επξεργαστών. Στη διατριβή χρησιμοποιήθηκαν σε κάποιες εκτελέσεις η συνάρτηση **omp_get_num_threads**, η οποία επιστρέφει τον αριθμό των threads που είναι ενεργά στην συγκεκριμένη παράλληλη περιοχή, και η διαδικασία **omp_set_num_threads** *(αριθμός threads)*, η κλήση της οποίας θέτει τον αριθμό που θα χρησιμοποιθούν στην εκτέλεση που ακολουθεί.

2.2.6 Μεταβλητές Περιβάλλοντος Χρηστών

Το πρότυπο OpenMP παρέχει τη δυνατότητα διαμόρφωσης των μεταβλητών κελύφους του περιβάλλοντος εκτέλεσης, με σκοπό την παραμετρικοποίηση της εκτέλεσης του παράλληλου κώδικα. Οι εντολές τους γράφονται με κεφαλαία γράμματα. Στη διατριβή επιλέχθηκε η χρήση της OMP_NUM_THREADS με την εντολή **export OMP_NUM_THREADS** = *αριθμός threads*, η οποία όπως γίνεται κατανοητό επιλέγει τον αριθμό των threads που θα χρησιμοποιηθούν.

2.3 Το πρότυπο OpenACC

2.3.1 Εισαγωγή

Η ενότητα αυτή παρουσιάζει συνοπτικά το πρότυπο ανάπτυξης εφαρμογών OpenACC, το οποίο χρησιμοποιήθηκε στην παρούσα διατριβή για την υλοποίηση αλγορίθμων σε γραφικά υποσυστήματα υπολογισμών.

Το πρότυπο OpenACC [33], είναι μια διεπαφή προγραμματισμού εφαρμογών (API, Application Programming Interface), με την οποία μπορεί ο προγραμματιστής να με-

ταθέσει την εκτέλεση μέρους του κώδικα προγράμματος (σε γλώσσες προγραμματισμού C, C++ ή Fortran) απο τον κεντρικό επεξεργαστή (host−CPU) σε μια ή περισσότερες συσκευές επιτάχυνσης (accelerator devices). Η έννοια *επιταχυντής* αναφέρεται σε οποιοδήποτε μέσο εκτέλεσης υπολογισμών διαφορετικό του κεντρικού επεξεργαστή. Όπως έχει ήδη αναφερθεί, στην παρούσα διατριβή ως επιταχυντές χρησιμοποιήθηκαν κάρτες γραφικών υποσυστημάτων (GPUs), κατά συνέπεια όλες οι αναφορές για GPUs που ακολουθούν και σχετίζονται με το πρότυπο, ισχύουν και για κάθε άλλο επιταχυντή.

Το πρότυπο OpenACC χαρακτηρίζεται απο τα παρακάτω στοιχεία:

- **Οδηγίες** (Compiler Directives): Ένα σύνολο από οδηγίες προς τον μεταγλωττιστή (compiler), γνωστές ως directives, οι οποίες ενσωματώνονται στον κώδικα με τη μορφή σχολίων, και υποδεικνύουν στον compiler τα τμήματα του κώδικα που θα εκτελεστούν στη GPU, χωρίς να χρειάζεται ο προγραμματιστής να διαμορφώσει εξόλοκλήρου τον κώδικα για να εκτελεστεί σε κάποια συσκευή GPU.

- **Συναρτήσεις Βιβλιοθήκης** (Library Routines): Οι συναρτήσεις βιβλιοθήκης χρόνου εκτέλεσης, με χρήση των οποίων μπορεί ο προγραμματιστής να εκκινήσει, να διαμορφώσει ή να τερματίσει τη συμμετοχή των GPUs στην εκτέλεση του προγράμματος.

- **Μεταβλητές Περιβάλλοντος** (Environment Variables): Οι μεταβλητές περιβάλλοντος χρηστών, με τις οποίες μπορεί ο χρήστης να καθορίσει τον τύπο και το πλήθος των GPUs που χρησιμοποιούνται κατά την εκτέλεση του προγράμματος, μέσω του περιβάλλοντος εκτέλεσης μιας εφαρμογής.

Το OpenACC ξεκίνησε να αναπτύσσεται, το 2010, από μια συνεργασία των εταιρειών NVIDIA, Cray, PGI και CAPS Enterprise [18, 33, 36]. Παρά την ύπαρξη αρκετών άλλων γλωσσών προγραμματισμού για GPUs, σε όλες θεωρούνταν αναγκαία η καλή γνώση των τεχνικών χαρακτηριστικών της GPU και η μετατροπή του κώδικα που θα εκτελεστεί στη GPU στην κατάλληλη γλώσσα, δημιουργώντας πλήθος διαφορετικών τμημάτων κώδικα.

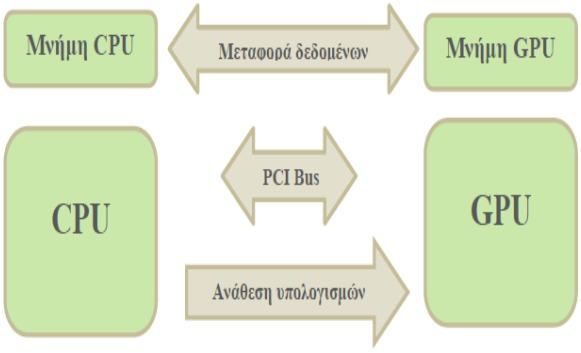
Ταυτόχρονα, ήταν επίπονη διαδικασία η εκ των υστέρων διαμόρφωση του κώδικα (π.χ. νέες προσθήκες), ενώ και η εκτέλεση σε νεότερης τεχνολογίας GPU, με άλλα χαρακτηριστικά, απαιτούσε σημαντικές αναπροσαρμογές. Έτσι, η ανάπτυξη ενός τέτοιου προτύπου είχε ως απώτερο σκοπό την απλούστευση της διαδικασίας παράλληλου προγραμματισμού ετερογενών συστημάτων (CPU/GPU), χρησιμοποιώντας αντίστοιχη υψηλού προγραμματιστικού επιπέδου προσέγγιση με αυτήν του προτύπου OpenMP (directive-based approach). Η πρώτη επίσημη έκδοση του OpenACC API (ver. 1.0) κυκλοφόρησε το Νοέμβριο του 2011, με την υποστήριξη των παραπάνω εταιρειών. Η NVIDIA κατασκευάζει κάρτες γραφικών αρχιτεκτονικής CUDA στην οποία στηρίζεται το πρότυπο, ενώ οι Cray, PGI και CAPS αναπτύσσουν σχετικό λογισμικό που υποστηρίζει το πρότυπο και διατίθεται στην αγορά. Το Μάρτιο του 2013 ανακοινώθηκε και το προσχέδιο της δεύτερης έκδοσης (ver. 2.0), με σημαντικές αλλαγές και προσθήκες, κυρίως όσον αφορά τις κλήσεις υποπρογραμμάτων εντός της περιοχής παραλληλοποίησης, την ύπαρξη παράλληλων διεργασιών μέσα σε ήδη υπάρχουσες,κ.α. Η έκδοση 2.0 είναι διαθέσιμη από τον Ιούλιο του 2013 [36].

2.3.2 Μοντέλο Ανάπτυξης Εφαρμογών

Κατά την εκτέλεση εφαρμογών με το πρότυπο OpenACC συμμετέχουν ο κεντρικός επεξεργαστής (CPU) και μια ή περισσότερες κάρτες γραφικών. Την καθοδήγηση και τον κύριο έλεγχο του προγράμματος αναλαμβάνει ο επεξεργαστής (CPU), στον οποίο εκτελείται και το κύριο μέρος του προγράμματος, ενώ τα τμήματα με μεγάλο υπολογιστικό φόρτο και αυξημένες ιδιότητες παραλληλοποίησης ανατίθενται για εκτέλεση στη GPU [59].

Αναλυτικότερα, ο CPU αναλαμβάνει να:

- Δεσμεύσει θέσεις στη μνήμη της GPU για τα δεδομένα



Σχήμα 2.6: Μοντέλο Εκτέλεσης εφαρμογής στο πρότυπο OpenACC

- Μεταφέρει τα δεδομένα στις θέσεις μνήμης της GPU
- Στείλει το κατάλληλο τμήμα του κώδικα για εκτέλεση στην GPU
- Περιμένει την ολοκλήρωση των διεργασιών της GPU
- Μεταφέρει τα δεδομένα από την μνήμη της GPU
- Απελευθερώσει τις θέσεις μνήμης της GPU

Απο την άλλη μεριά η GPU, εκτελεί το συγκεκριμένα τμήματα του κώδικα σε περιοχές παράλληλης εκτέλεσης (parallel regions ή kernels regions), όπου περιέχονται βρόγχοι εργασίας (work-sharing loops). Οι βρόγχοι αντιστοιχίζονται σε κατάλληλα νήματα υπολογισμών(GPU threads), τα οποία αναλαμβάνουν την εκτέλεση του προγράμματος.

2.3.3 Μοντέλο Διαχείρισης Μνήμης

Η διαχείριση της μνήμης είναι μια σημαντική παράμετρος στην εκτέλεση ενός προγράμματος, είτε αυτό εκτελείται αποκλειστικά σε CPU (host-only program), είτε όταν εκτελείται σε CPU και GPU (host-accelerator program). Ο λόγος είναι ότι η πλειοψηφία των GPUs που κυκλοφορούν σήμερα, όπως προαναφέρθηκε, έχουν αυτόνομη μνήμη, διαφορετική απο αυτή του CPU. Σε αυτή την περίπτωση, πρέπει να πραγματοποιηθεί μεταφορά των δεδομένων, την οποία αναλαμβάνει ο CPU, χρησιμοποιώντας το πρότυπο Άμεσης Προσπέλασης Μνήμης (DMA). Στις χαμηλού προγραμματιστικού επιπέδου

γλώσσες CUDA και OpenCL, τη μεταφορά των δεδομένων διαχειρίζεται ο χρήστης με κατάλληλες εντολές, οι οποίες συνήθως επιβαρύνουν υπερβολικά τον κώδικα. Τα δεδομένα μεταφέρονται απο τον CPU στην GPU, άμεσα (explicit). Στο OpenACC η μεταφορά γίνεται έμμεσα (implicit) και την αναλαμβάνει ο compiler, με δυνατότητα καθοδήγησης απο τον προγραμματιστή μέσω directives όπου θεωρεί ότι χρειάζεται παρέμβαση του. Για την άρτια διαχείριση στην μεταφορά των δεδομένων (αποφυγή περιττών μετακινήσεων) από και πρός την GPU, χρησιμοποιούνται κατάλληλες για το σκοπό directives, καθώς και οι αντίστοιχοι όροι τους (clauses), οι οποίες παρουσιάζονται συνοπτικά στη συνέχεια.

Θα πρέπει να τονιστεί ότι εξακολουθούν να υφίστανται παράμετροι σχετικές με τη διαχείριση της μνήμης και στο μοντέλο του προτύπου OpenACC, οι οποίες μπορούν να διαμορφώσουν την απόδοση μιας παράλληλης εκτέλεσης της εφαρμογής. Οι πιο σημαντικές είναι η μέγιστη ταχύτητα προσπέλασης της μνήμης της GPU από το CPU (bandwidth) και η σχετικά περιορισμένη, σε σχέση με την μνήμη του CPU, κύρια μνήμη της GPU (global memory). Η περιορισμένη ταχύτητα προσπέλασης της μνήμης καθορίζει το πόσο έντονη υπολογιστικά πρέπει να είναι η περιοχή που θέλει ο προγραμματιστής να παραλληλοποιήσει, ώστε να είναι αποδοτική η διαδικασία μεταφοράς και εκτέλεσης στην GPU. Όσον αφορά δε την κύρια μνήμη της GPU, η χωρητικότητα της είναι αποτρεπτικός παράγοντας για μεταφορά και παραλληλοποίηση τμημάτων διεργασιών που απαιτούν πολύ μεγάλο χώρο αποθήκευσης λόγω του όγκου των δεδομένων. Σε επίπεδο GPU, πρέπει να αναφερθεί ότι, οι πλέον πρόσφατες συσκευές δεν υποστηρίζουν τη συνοχή της μνήμης μεταξύ των εργασιών που εκτελούνται από διαφορετικές μονάδες εκτέλεσης SMs, ενώ και στην ίδια μονάδα εκτέλεσης η συνοχή εξασφαλίζεται όταν μεταξύ των εργασιών υπάρχει σαφές φράγμα αναμονής. Διαφορετικά, σε περιπτώσεις όπου μια διεργασία διαβάζει δεδομένα από μια θέση μνήμης, ενώ μια άλλη διαβάζει την ίδια θέση, ή δύο διαφορετικές διεργασίες αποθηκεύουν στη ίδια θέση, τα αποτελέσματα ενδεχομένως να είναι ασύμβατα. Ουσιαστικά εφαρμόζεται ένα πιο αδύναμο μοντέλο μνήμης, χωρίς να υπάρχει δηλαδή συγχρονισμός μεταξύ των διαφορετικών μονάδων εκτέλεσης. Τέλος,

πρέπει να αναφερθεί ότι η διαχείριση της μνήμης cache της GPU στο OpenACC γίνεται επίσης απο τον compiler, με δυνατότητα οδηγιών (directives) απο τον χρήστη, σε αντίθεση με τα πρότυπα CUDA και OpenCL.

2.3.4 Οδηγίες OpenACC

Όπως προαναφέρθηκε, χρήση του προτύπου OpenACC μπορεί να γίνει από τις γλώσσες προγραμματισμού C, C++ και Fortran. Η γενική σύνταξη μιας OpenACC οδηγίας γίνεται με διαφορετικό τρόπο στις C/C++ και Fortran. Στην παρούσα διατριβή χρησιμοποιήθηκε η γλώσσα Fortran, συνεπώς οι directives που θα περιγραφούν, θα έχουν τη σύνταξη που απαιτεί η συγκεκριμένη γλώσσα.
Μια OpenACC directive εισάγεται σε κώδικα γλώσσας Fortran με τη μορφή

```
!$acc directive clause1(var1,var2,...), clause2(var3,...),...
```

Ο συμβολισμός `!$acc` γράφεται σαν μια λέξη, χωρίς κενά, στις στήλες $1-5$ του κώδικα, ακολουθούμενος από το όνομα της directive και, για κάποιες directives, τους όρους (clauses) με τις απαιτούμενες μεταβλητές. Σε περίπτωση που απαιτείται συνέχιση της directive στην επόμενη γραμμή, τον συμβολισμό `!$acc` ακολουθεί άμεσα το συμπλεκτικό &. Ακολουθεί το μέρος του αρχικού κώδικα, που θα εκτελεστεί παράλληλα, ενώ κάποιες directives απαιτούν τερματισμό, με τρόπο που περιγράφεται παρακάτω [59, 36].

Οι directives ανάλογα το σκοπό που εξυπηρετούν διακρίνονται σε **βασικές** (construct directives), **εκτέλεσης** (execution directives) και **δεδομένων** (data directives). Σε όλες μπορούν ή πρέπει να προστεθούν συγκεκριμένοι όροι (clauses) με ή χωρίς ορίσματα, που παρέχουν συγκεκριμένες οδηγίες εκτέλεσης στον compiler. Τα ορίσματα των όρων μπορεί να είναι μεταβλητές, πίνακες, διανύσματα ή σταθερές και χωρίζονται με κόμμα, ενώ η σειρά τους δεν έχει σημασία. Οι directives που χρησιμοποιήθηκαν στην ανάπτυξη εφαρμογών της παρούσας διατριβής παρουσιάζονται συνοπτικά στη συνέχεια, ενώ για

περισσότερες λεπτομέρειες μπορεί κάποιος να ανατρέξει στο εγχειρίδιο του OpenACC [;]

Οι Βασικές Οδηγίες !$acc parallel και !$acc kernels

Οι θεμελιώδεις directives στο πρότυπο OpenACC είναι οι !$acc parallel και !$acc kernels. Πρόκειται για τις directives, οι οποίες ορίζουν την περιοχή του προγράμματος που θα εκτελεστεί παράλληλα στην GPU. Ο τρόπος με τον οποίο λειτουργεί καθεμία είναι διαφορετικός, όπως γίνεται αντιληπτό απο την περιγραφή που ακολουθεί.

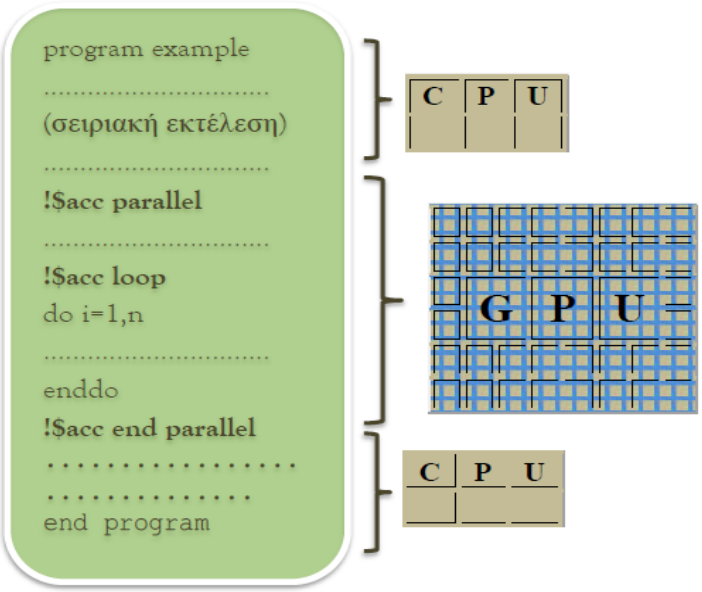
!$acc parallel Η !$acc parallel καθορίζει την περιοχή του κώδικα η οποία θα εκτελεστεί παράλληλα στην GPU και έχει τη σύνταξη

```
!$acc parallel clause1(var1,var2,...), clause2(var3,...),...
```

δομημένο block κώδικα

```
!$acc end parallel
```

Η λειτουργία της αντιστοιχεί σε αυτήν του προτύπου OpenMP Parallel construct, όπου θυμίζεται ότι με την έναρξη της παραλληλοποιήσιμης περιοχής(parallel region) της διεργασίας ένας συγκεκριμένος αριθμός από threads εκτελεί ταυτόχρονα το μέρος αυτό του κώδικα, ενώ τους βρόγχους εργασίας (work-sharing loops) εντός της parallel region εκτελούν τα threads διαδοχικά. Αντίστοιχα, και η οδηγία !$acc parallel ορίζει την περιοχή για την οποία δημιουργείται συγκεκριμένος αριθμός ομάδων (**gangs**) από threads , με συγκεκριμένο αριθμό threads η καθεμία (που παραμένει σταθερός), και όλοι οι βρόγχοι εργασίας εκτελούνται από αυτές, επίσης με διαδοχικό τρόπο.
Ουσιαστικά, ολόκληρη η περιοχή αντιμετωπίζεται ως μια διεργασία, που εκτελείται από ομάδες από threads. Το πέρας της περιοχής και η παύση της εκτέλεσης στη GPU προσδιορίζεται από τον προγραμματιστή με την εντολή !$acc end parallel, η οποία ορίζει και ένα φράγμα αναμονής (barrier) στον κώδικα, όπου ο host περιμένει όλα τα gangs να



Σχήμα 2.7: Σχηματική χρήση της οδηγίας parallel

ολοκληρώσουν την εκτέλεση που ανέλαβαν, πριν προχωρήσει στην εκτέλεση των επόμενων εντολών του προγράμματος. Τέλος, οι όροι της !$acc parallel μπορούν να είναι όροι δεδομένων, ασύγχρονης εκτέλεσης, επιλογής του αριθμού των ομάδων των threads,κ.α.

!$acc kernels Η οδηγία !$acc kernels καθορίζει επίσης την περιοχή του κώδικα η οποία θα παραλληλοποιηθεί και συντάσσεται ως

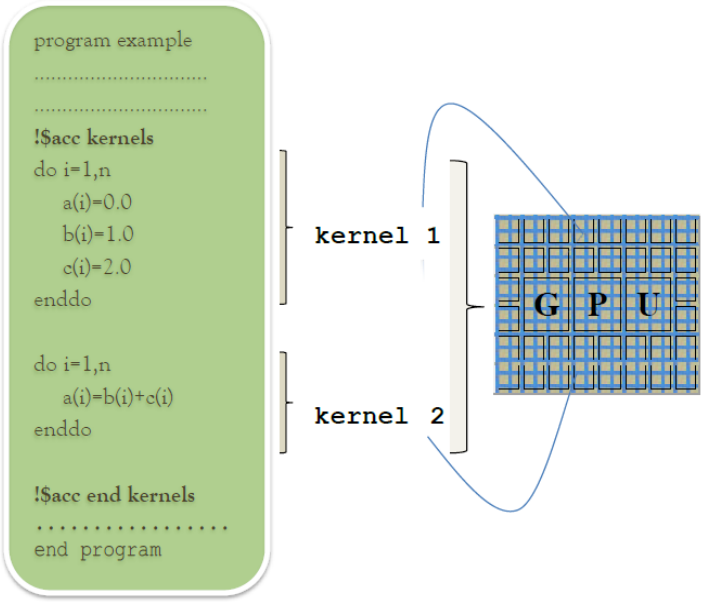
```
!$acc kernels clause1(var1,var2,...), clause2(var3,...),...
```

δομημένο block κώδικα

```
!$acc end kernels
```

Σε αντίθεση όμως με την !$acc parallel, ο compiler εδώ θα χωρίσει την περιοχή σε ξεχωριστές διεργασίες, οι οποίες θα εκτελεστούν παράλληλα, αν αυτό είναι εφικτό. Ουσιαστικά, κάθε βρόγχος εργασίας εντός της περιοχής θα είναι μια διαφορετική διεργασία,

και κάθε μια μπορεί να εκτελεστεί από διαφορετικό αριθμό ομάδων threads και με διαφορετικό αριθμό threads στην κάθε ομάδα. Επομένως, εδώ η περιοχή εκτελείται ως ακολουθία διεργασιών στην GPU.



Σχήμα 2.8: Σχηματική χρήση της οδηγίας kernels

Πρέπει να αναφερθεί ότι προκειμένου να εκτελεστεί μια kernels περιοχή, ο compiler πρώτα προσδιορίζει ποιοι βρόγχοι μπορούν να εκτελεστούν παράλληλα, και στη συνέχεια επιλέγει το κατάλληλο επίπεδο παραλληλισμού για κάθε βρόγχο- το grid level αντιστοιχεί στο gang ενώ το thread level αντιστοιχεί στο vector του προτύπου OpenACC - ή πολλαπλά επίπεδα παραλληλισμού με χρήση strip-mining [59]. Το πέρας της περιοχής και η παύση της εκτέλεσης στην GPU προσδιορίζεται από τον προγραμματιστή με την !$acc end kernels, η οποία επίσης ορίζει και ένα φράγμα αναμονής (barrier) στον κώδικα, όπως αυτό περιγράφεται και παραπάνω. Τέλος, ως όροι μπορούν να χρησιμοποιηθούν διάφοροι όροι δεδομένων (data clauses) ή ασύγχρονης εκτέλεσης, μερικοί εκ των οποίων περιγράφονται παρακάτω, για την αποφυγή άσκοπων μετακινήσεων δεδομένων από τον compiler, ή για την ακύρωση του φράγματος αναμονής.

Οι περιοχές οι οποίες ορίζονται από kernels ή parallel constructs, δεν μπορούν να περιέχουν άλλες περιοχές παράλληλης εκτέλεσης kernels ή parallel.

Συμπερασματικά, αν και οι δυο οδηγίες χρησιμοποιηθούν για να παραλληλοποιήσουν την ίδια περιοχή ενός κώδικα, το αποτέλεσμα που θα παράξουν ενδέχεται να είναι διαφορετικό και ανάλογο του κώδικα που ζητείται να παραλληλοποιηθεί. Όπως περιγράφει η [37], η !$acc kernels παρέχει την ελευθερία στο compiler να επιλέξει άν θα παραλληλοποιήσει μια περιοχή και ποιός είναι ο κατάλληλος τρόπος, ανάλογα τις δυνατότητες GPU και τις απαιτήσεις του προγράμματος. Η περιοχή αυτή θα αντιμετωπιστεί ως ακολουθία διεργασιών. Από την άλλη, στην !$acc parallel απαιτείται απο τον προγραμματιστή να έχει αναλύσει και να επιλέξει ποια είναι η καταλληλότερη προσέγγιση στην παράλληλη εκτέλεση που θα ακολουθήσει εντός της ορισμένης περιοχής, που θα αντιμετωπιστεί ως μια και μοναδική διεργασία.

Η Οδηγία Δεδομένων !$acc data

Έχει σύνταξη

`!$acc data` *`clause1(var1,var2,...), clause2(var3,...),...`*

δομημένο block κώδικα

`!$acc end data`

και καθορίζει την περιοχή του προγράμματος, στην αρχή της οποίας μεταφέρονται τα δεδομένα από την μνήμη του CPU στην μνήμη της GPU (host to device transfer), ενώ στο τέλος της τα δεδομένα μεταφέρονται πίσω στη μνήμη του CPU (device to host transfer). Υπενθυμίζεται ότι την ανάθεση στη μνήμη της CPU και την μεταφορά των δεδομένων αναλαμβάνει ο host. Οι κυριότεροι όροι της οδηγίας data που εξυπηρετούν την ανάθεση ή μεταφορά δεδομένων είναι οι παρακάτω

- `copyin(var1,var2,..)` Τα δεδομένα που περιέχονται στη copyin clause αντιγράφονται από το CPU στη GPU, αλλά δεν επιστρέφουν στον CPU.

- `copy(var1,var2,..)` Τα δεδομένα που περιέχονται στη copy clause αντιγράφονται από το CPU στη GPU, στην αρχή της περιοχής και επιστρέφουν από την GPU στον CPU στο τέλος της περιοχής.

- `copyout(var1,var2,..)` Τα δεδομένα που περιέχονται στη copyout clause αντιγράφονται από τη GPU στο CPU στο τέλος της περιοχής εντολών.

- `create(var1,var2,..)` Για τα δεδομένα που περιέχονται στην create clause ο CPU θα δεσμεύσει νέες θέσεις μνήμης στη GPU. Δεν υπάρχει μεταφορά δεδομένων μεταξύ της μνήμης του CPU και της GPU.

- `present(var1,var2,..)` Για τα δεδομένα που περιέχονται στη present clause, ο compiler ενημερώνεται ότι βρίσκονται ήδη στη μνήμη της GPU, οπότε επιβάλλεται να μην πραγματοποιηθεί αντιγραφή.

- `pcopyin, pcopy, pcopyout, pcreate` Το αρχικό γράμμα p είναι συντόμευση του present. Ο compiler ελέγχει πρώτα αν τα δεδομένα βρίσκονται ήδη στη μνήμη της GPU, και αν δεν συμβαίνει αυτό, τότε πραγματοποιείται η αντιγραφή.

Η Οδηγία Εκτέλεσης !$acc loop

Η οδηγία !$acc loop προηγείται κάθε βρόγχου, ο οποίος βρίσκεται εντός περιοχής παράλληλοποίησης και έχει τη μορφή

`!$acc loop` *`clause1(var1,var2,...), clause2(var3,...),...`*

δομημένο block κώδικα

Οι συμπληρωματικοί όροι διαμορφώνουν τον τρόπο παραλληλοποίησης που θα ακολουθήσει ο compiler. Μπορεί να τεθεί είτε εντός kernel περιοχής, είτε εντός parallel

περιοχής, όμως οι όροι, και ο τρόπος με τον οποίο χρησιμοποιούνται σε κάθε περίπτωση είναι διαφορετικοί λόγω της φύσης των συγκεκριμένων directives. Ενδεικτικά οι κυριότεροι όροι είναι

- `gang [ ]` Καθορίζει των αριθμό των CUDA blocks από threads που θα εκτελέσουν τον βρόγχο, με κατάλληλο όρισμα να μεταβάλλεται απο τον προγραμματιστή.

- `vector [ ]` Καθορίζει των αριθμό των threads σε κάθε block, με κατάλληλο όρισμα να εισάγεται απο τον προγραμματιστή.

- `independent` Δηλώνει ότι οι επαναλήψεις εντός του βρόγχου διαθέτουν ανεξάρτητα δεδομένα. Η εξάρτηση μεταξύ των δεδομένων αποδίδει πολλές φορές λανθασμένα αποτέλεσματα. Για το λόγο αυτό ο compiler αποφεύγει να εκτελέσει παράλληλα βρόγχο για τον οποίο θεωρεί ότι υπάρχει εξάρτηση δεδομένων, εκτελώντας τον σειριακά. Συνεπώς με τη χρήση της, επιβάλλεται στον compiler να θεωρήσει την ανεξαρτησία των δεδομένων.

- `collapse [ ]` Ο όρος δέχεται σαν όρισμα ένα φυσικό αριθμό, που υποδεικνύει τους βρόγχους που ακολουθούν, για τους οποίους είναι σε ισχύ η !$acc loop και οι όροι που έχουν δηλωθεί σε αυτήν, χωρίς να χρειάζεται να χρησιμοποιηθεί εκ νέου.

- `private [ ]` Ο όρος υποδεικνύει στον compiler να κρατήσει την συγκεκριμένη τιμή σε κάθε επανάληψη, για τις μεταβλητές που δηλώνονται σε αυτόν.

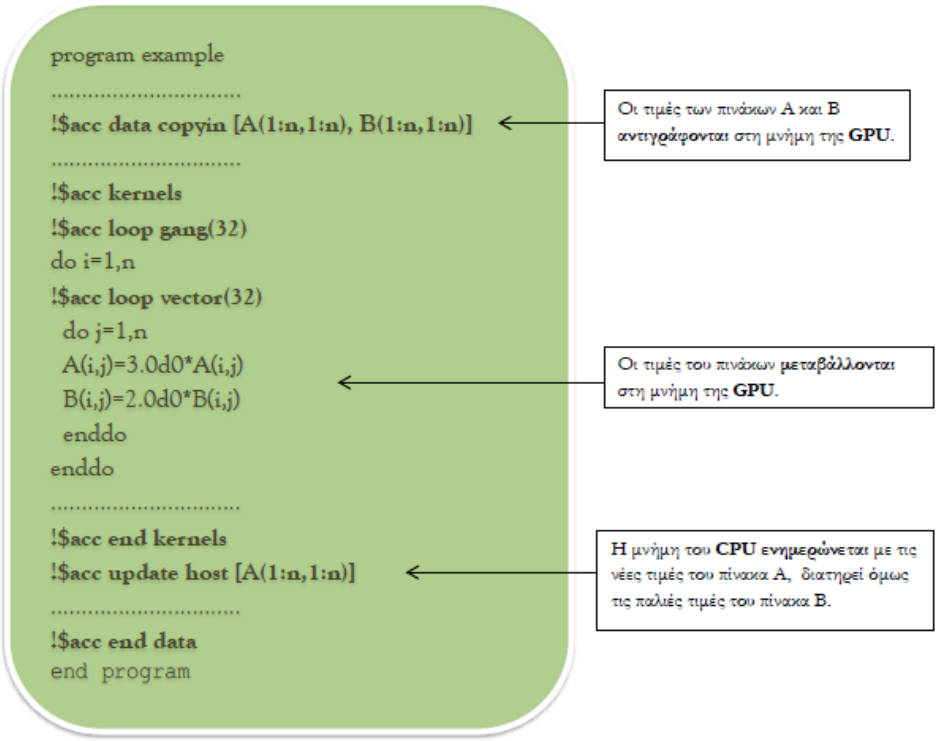
Οι παραπάνω όροι χρησιμοποιούνται μόνο εντός kernel περιοχής.

Η Οδηγία Ενημέρωσης !$acc update

Η οδηγία !$acc update συντάσσεται

```
!$acc update clause(var1,var2,...)
```

με κύριους όρους τους device και host. Ανάλογα με τον όρο που χρησιμοποιείται, δηλώνει ότι οι τιμές των μεταβλητών που περιέχονται, θα ενημερώσουν την μνήμη της GPU (από τις τιμές που έχει ο CPU) ή τη μνήμη του CPU (από τις τιμές που έχει η GPU).



Σχήμα 2.9: Σχηματική χρήση της οδηγίας update directive

2.3.5 Συναρτήσεις Βιβλιοθήκης

Οι συναρτήσεις βιβλιοθήκης του προτύπου OpenACC παρέχουν τη δυνατότητα έναρξης, τερματισμού, επιλογής και διαμόρφωσης των παραμέτρων που χρησιμοποιεί το API κατά τη διάρκεια της εκτέλεσης του προγράμματος. Δηλαδή ο αριθμός και ο τύπος των GPUs που θα χρησιμοποιηθεί, η έναρξη και ο τερματισμός της συμμετοχής μιας GPU στην εκτέλεση,κ.α. Οι κλήσεις των συναρτήσεων αρχικοποιούν και στη συνέχεια μπορούν να μεταβάλλουν τις ICVs, Internal Control Variables (εσωτερικές μεταβλητές ελέγχου), με τις οποίες ελέγχεται το περιβάλλον εκτέλεσης.

Η διαδικασία !$acc_init

Η κλήση της διαδικασίας !$acc_init ορίζει την έναρξη της συμμετοχής της GPU στην εκτέλεση της εφαρμογής. Τονίζεται ότι κάθε compute GPU παραμένει σε ανενεργή κατάσταση, όταν δεν υπάρχει διεργασία CUDA σε εξέλιξη για αυτήν, και ενεργοποιείται μόνο όταν το πρόγραμμα εντοπίσει περιοχή παράλληλης εκτέλεσης. Ο χρόνος εκκίνησης της είναι περίπου 1 δευτερόλεπτο. Σε περιπτώσεις που οι μετρήσεις του χρόνου εκτέλεσης πρέπει να είναι ακριβείς, θα πρέπει να αποφεύγεται η προσμέτρηση αυτού του χρόνου. Συντάσσεται με όρισμα τον τύπο της GPU που ορίζεται να εκκινήσει. Για GPUs κατασκευασμένες από την NVIDIA το όρισμα αυτό είναι το acc_device_nvidia.

Η διαδικασία !$acc_shutdown

Η κλήση της τερματίζει τη συμμετοχή της GPU στην εκτέλεση της εφαρμογής, καθώς και η επικοινωνία της με τον κεντρικό επεξεργαστή. Συντάσσεται επίσης με όρισμα τον τύπο της GPU που ορίζεται να τερματιστεί.

Και οι δύο παραπάνω διαδικασίες καλούνται εκτός περιοχής παράλληλης εκτέλεσης.

Η διαδικασία !$acc_set_device_num

Με την κλήση της επιλέγεται η GPU που θα χρησιμοποιηθεί. Η επιλογή και ορισμός της καθορίζεται με ορίσματα τον αριθμό της GPU, που είναι ακέραιος αριθμός μεγαλύτερος ίσος του 0 (το 0 αντιστοιχεί στην προεπιλεγμένη απο το πρόγραμμα GPU) και τον τύπο της GPU, όπως αναφέρθηκε και παραπάνω.

Η συνάρτηση !$acc_get_device_num

Η συγκεκριμένη συνάρτηση επιστρέφει τον αριθμό της GPU που θα χρησιμοποιηθεί για εκτέλεση εντολών στο επόμενο τμήμα παράλληλης εκτέλεσης του κώδικα.

2.3.6 Μεταβλητές Περιβάλλοντος Χρηστών

Ο προγραμματιστής έχει τη δυνατότητα να τροποποιήσει τις μεταβλητές περιβάλλοντος, διαμορφώνοντας κατάλληλα τη συμπεριφορά του προγράμματος. Παρακάτω ακολουθούν οι κυριότερες εντολές σε επίπεδο κελύφους τύπου bash, στις οποίες τονίζεται ότι υπάρχει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων.

- export PGI_ACC_TIME $= 0$ *ή* 1 Όταν ενεργοποιηθεί παρέχει αναλυτική αναφορά για το χρόνο εκκίνησης, χρόνο παραμονής σε περιοχή παράλληλης εκτέλεσης καθώς και εκτέλεσης κάθε διεργασίας.

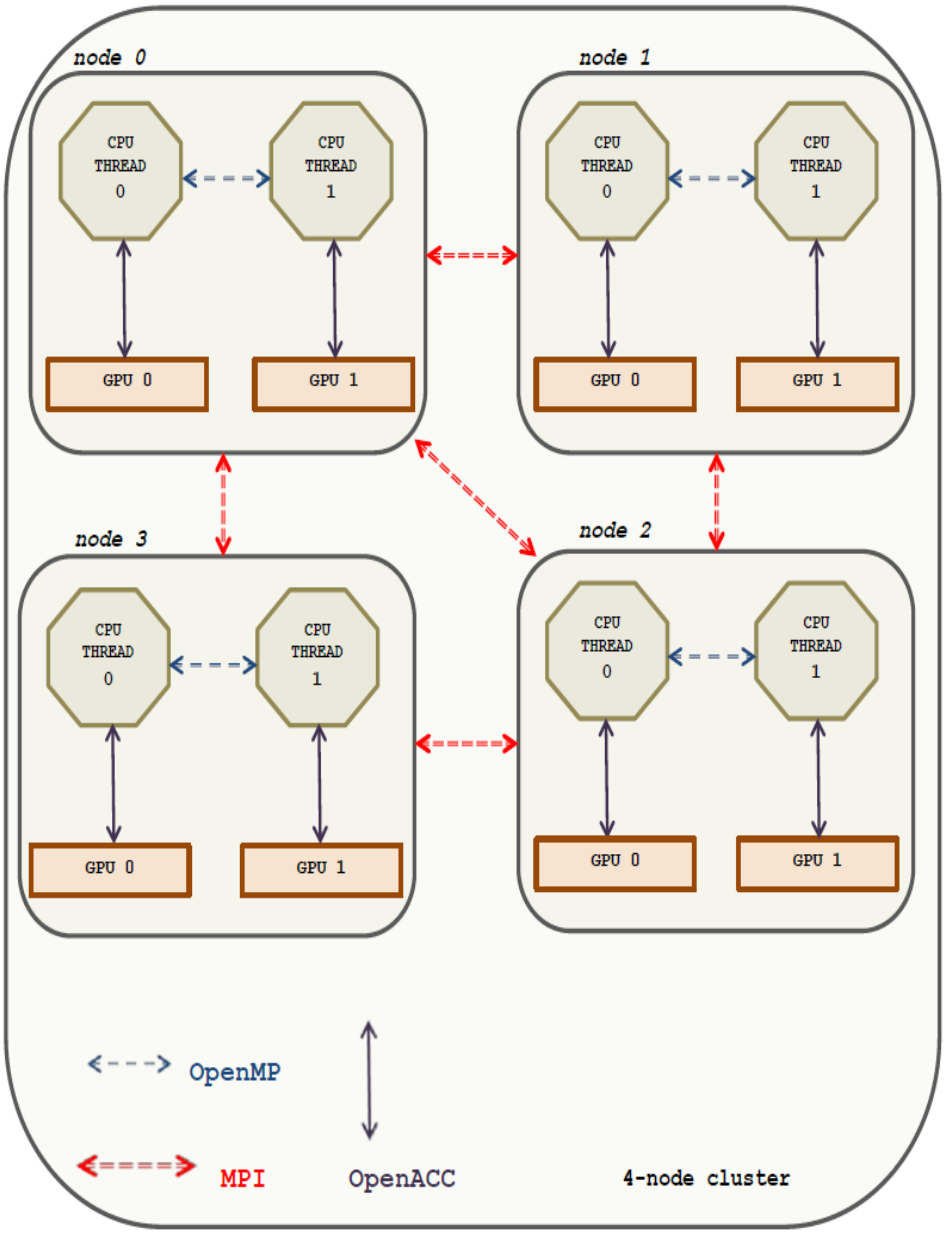
- export PGI_ACC_SYNCHRONOUS =*0 ή 1* Στις πλέον πρόσφατες εκδόσεις του compiler γίνεται προεπιλεγμένη χρήση της δεσμευμένης μνήμης (pinned memory) με σκοπό τη βελτίωση της ασύγχρονης μεταφοράς δεδομένων. Σε κάποιες περιπτώσεις εκτελέσεων, παρατηρείται μια επιβράδυνση που οφείλεται στο συγχρονισμό του host και της GPU κατά τη διάρκεια της απελευθέρωσης της pinned memory που γίνεται για να ολοκληρωθούν όλες οι τυχόν ατελείς μεταφορές δεδομένων. Με την ενεργοποίηση της ο compiler θα επαναφέρει την χρήση της non-pinned memory. Αξίζει να τονιστεί ότι, σύμφωνα με [20], η χρήση της pinned memory είναι ευνοική ανάλογα το μέγεθος των δεδομένων που μεταφέρεται και το είδος της μεταφοράς (από και πρός, μόνο πρός).

- export ACC_DEVICE_NUM=*ακέραια τιμή μεγαλύτερη του* 0 Ορίζει τον αριθμό της προεπιλεγμένης GPU.

- export ACC_DEVICE_TYPE=NVIDIA Ορίζει τον τύπο της προεπιλεγμένης GPU.

2.4 MPI-OpenMP-OpenACC σε υπερυπολογιστικά συστήματα GPU/CPU

Όπως αναφέρθηκε σε προηγούμενη ενότητα, η σημερινή αρχιτεκτονική των υπερυπολογιστικών συστημάτων ακολουθεί το μοντέλο της κατανεμημένης κοινής μνήμης. Αυτό έχει ως άμεση συνέπεια τη χρήση τουλάχιστον δύο προγραμματιστικών προτύπων κατά την ανάπτυξη εφαρμογών. Ειδικότερα για εφαρμογές χρήσης άνω του ενός υπολογιστικού κόμβου, όπου χρειάζεται να γίνει ανταλλαγή δεδομένων και συγχρονισμός μεταξύ τους, απαιτείται η υλοποίηση παράλληλων αλγορίθμων με χρήση του προτύπου MPI. Έτσι, ιεραρχικά ανώτερα θα βρίσκονται οι διαδικασίες αυτού του προτύπου, οι οποίες θα καθορίζουν και θα υλοποιούν την ανταλλαγή πληροφοριών καθώς και την κατανομή υπολογιστικού φόρτου σε κάθε κόμβο του υπολογιστικού συστήματος. Στη συνέχεια θα υλοποιούνται σε κάθε υπολογιστικό κόμβο διαδικασίες του προτύπου OpenMP σύμφωνα με το μοντέλο κοινής διαχείρισης της μνήμης του. Στην περίπτωση όπου κάθε υπολογιστικός κόμβος διαθέτει ένα αριθμό από GPUs, την αμέσως επόμενη προτεραιότητα διαδικασιών θα υλοποιεί το πρότυπο OpenACC. Επειδή οι διαδικασίες που υλοποιούνται σε μια GPU χρειάζεται να δημιουργούνται, να επιβλέπονται και να τερματίζονται από μια CPU διαδικασία, είναι φανερό ότι κατά τη χρήση πολλαπλών GPUs σε ένα υπολογιστικό κόμβο απαιτείται η δημιουργία ίσου αριθμού CPU διαδικασιών. Αυτές οι διαδικασίες παράγονται από OpenMP εκτελέσιμες εντολές. Έτσι κατά τη χρήση πολλαπλών GPUs θα χρειαστεί να υπάρξει ταυτόχρονα εκτέλεση ίδιου πλήθους CPU διαδικασιών, με αποτέλεσμα να είναι αναγκαία, για την αποδοτική εκτέλεση του αλγορίθμου, η ύπαρξη τουλάχιστον ίσου αριθμού CPU υπολογιστικών πυρήνων με τον αριθμό των GPUs σε κάθε υπολογιστικό κόμβο.



Σχήμα 2.10: Σχηματική αναπαράσταση υπερυπολογιστικού συστήματος με 4 κόμβους, που επικοινωνούν με χρήση MPI, σε κάθε ένα από τους οποίους εκτελούνται 2 threads, με χρήση OpenMP, οι οποίοι και ελέγχουν τις GPUs κάθε κόμβου, δίνοντας οδηγίες με χρήση OpenACC.

Κεφάλαιο 3

Μέθοδος Πεπερασμένων Στοιχείων

3.1 Εισαγωγή

Η μοντελοποίηση φυσικών προβλημάτων συχνά οδηγεί στη δημιουργία Προβλημάτων Συνοριακών Τιμών (ΠΣΤ), τα οποία με τη σειρά τους επιλύονται με κάποια αριθμητική μέθοδο. Αυτό συμβαίνει είτε γιατί η αναλυτική τους επίλυση δεν είναι εφικτή, είτε γιατί είναι επιθυμητή κάποια προσεγγιστική τιμή της ακριβούς λύσης. Οι αριθμητικές μέθοδοι που χρησιμοποιούνται σε αυτές τις περιπτώσεις είναι οι φασματικές μέθοδοι και οι μέθοδοι πεπερασμένων Στοιχείων, Όγκων και Διαφορών[2, 13, 47].

Ένα ΠΣΤ το οποίο ορίζεται στο χωρίο Ω μπορεί να περιγραφεί μέσω των παρακάτω σχέσεων :

$$\text{(ΠΣΤ)} \quad \begin{cases} \mathcal{L}u(\mathbf{x}) \;=\; f(\mathbf{x}) \;\;,\;\; \mathbf{x} \in \Omega \\ \\ \mathcal{B}u(\mathbf{x}) \;=\; g(\mathbf{x}) \;\;,\;\; \mathbf{x} \in \partial\Omega \end{cases} \;, \tag{3.1}$$

όπου ο τελεστής $\mathcal{L}$ ισχύει εντός του χωρίου Ω, ενώ ο τελεστής $\mathcal{B}$ περιγράφει τις συνοριακές συνθήκες του προβλήματος και εφαρμόζεται πάνω στο σύνορο $\partial\Omega$.

Μια από τις πιο διαδομένες επιλογές για την επίλυση ενός ΠΣΤ είναι η μέθοδος Πεπερασμένων Στοιχείων (Finite Element Method).

Η ευρεία αποδοχή της μεθόδου αποδίδεται στη γενικότητα από την οποία χαρακτηρίζεται, καθώς πλήθος ΠΣΤ από διάφορους επιστημονικούς κλάδους μπορεί να αναλυθεί και να επιλυθεί με ένα συγκεκριμένο πλαίσιο επίλυσης, καθώς επίσης και από την ευελιξία που παρέχει στην επιλογή της μεθόδου διακριτοποίησης.
Η μέθοδος Πεπερασμένων Στοιχείων μπορεί να περιγραφεί με τη βοήθεια των παρακάτω βημάτων :

- **Βήμα 0:** Γεωμετρική Διαμέριση του χωρίου Ω σε πεπερασμένο πλήθος υπολογιστικών κελιών T_h= { T } τέτοια ώστε $U_{T \in T_h}$ T = Ω. Τα κελιά στο σύνολο τους σχηματίζουν ένα *πλέγμα* του χωρίου Ω και είναι κατά κανόνα απλά πολυγωνικά σχήματα, όπως τρίγωνα, τετράπλευρα, κ.ο.κ. Είναι όμως δυνατή και η δημιουργία άλλων σχημάτων κελιών, συγκεκριμένα καμπυλόγραμμων, σε περίπτωση που πρέπει να αποτυπωθεί ορθά το σύνορο ενός μή−πολυγωνικού χωρίου.

- **Βήμα 1:** Επιλογή n γραμμικά ανεξάρτητων κατά τμήματα συνεχών πολυωνυμικών συναρτήσεων $\Phi_1, \ldots, \Phi_n$, σε κάθε κελί T, , οι οποίες ονομάζονται συναρτήσεις **βάσης** και κατασκευάζουν τη προσέγγιση $u_n(\mathbf{x})$ της πραγματικής λύσης $u(\mathbf{x})$ του ΠΣΤ ως εξής

$$u(\mathbf{x}) \simeq u_n(\mathbf{x}) = a_1\Phi_1(\mathbf{x}) + \cdots + a_n\Phi_n(\mathbf{x}) = \sum_{k=1}^{n} a_k\Phi_k(\mathbf{x}). \tag{3.2}$$

  Κάθε κελί T μαζί με το χώρο των συναρτήσεων βάσης V και ένα σύνολο κανόνων που περιγράφουν τις συναρτήσεις στον V καλείται *πεπερασμένο στοιχείο.* σύμφωνα με τον ορισμό του Ciarlet (1976) [42].

- **Βήμα 2:** Επιλογή μεθόδου διακριτοποίησης (Galerkin, Rayleigh-Ritz, Least Squares, Collocation) [2] για μετάβαση από το **συνεχή** χώρο στο **διακριτό**, δηλαδή μετατροπή του ΠΣΤ σε ένα πρόβλημα επίλυσης του γραμμικού συστήματος

$$C\vec{\mathbf{a}} = \vec{\mathbf{b}} \tag{3.3}$$

όπου $C \in \mathbb{R}^{n,n}$, $\vec{\mathbf{a}} = [a_1 \;\; a_2 \;\; \cdots \;\; a_n]^T$ και $\vec{\mathbf{b}} = [b_1 \;\; b_2 \;\; \cdots \;\; b_n]^T$.

- **Βήμα 3:** Επιλογή μεθόδου για την επίλυση του γραμμικού συστήματος και προσδιορισμός των αγνώστων $a_1, \ldots, a_n$. Η επιλογή αυτή βασίζεται στο μέγεθος και στις ιδιότητες του παραγόμενου γραμμικού συστήματος.

3.2 Μέθοδος Hermite Collocation για ΠΣΤ ελλειπτικού τύπου

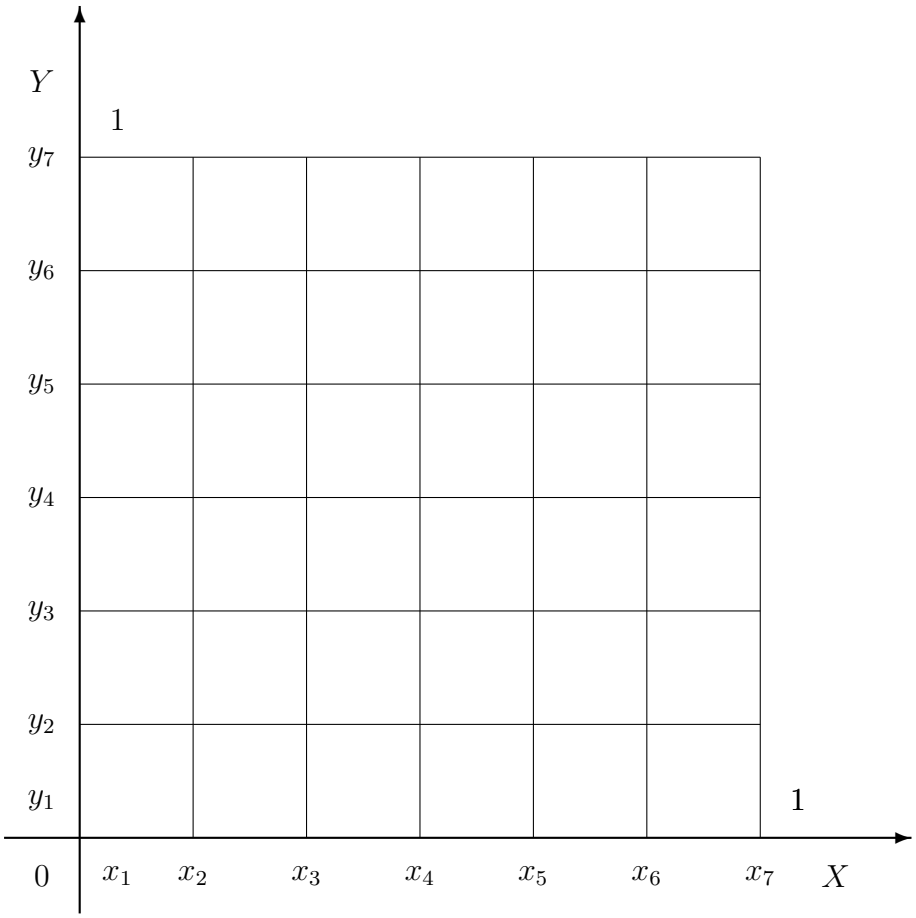
Η μέθοδος Hermite Collocation είναι μια αριθμητική μέθοδος υψηλής ακρίβειας, αφού επιτρέπει την προσέγγιση της λύσης ΠΣΤ με υψηλή τάξη σφάλματος. Κατά την εφαρμογή της αποφεύγεται η διαδικασία της αριθμητικής ολοκλήρωσης, η οποία απαιτείται κατά την υλοποίηση της κλασσικής μεθόδου πεπερασμένων στοιχείων Galerkin. Επίσης, η χρήση των πολυωνύμων Hermite ως συναρτήσεις βάσης της μεθόδου επιτρέπει τον άμεσο προσδιορισμό τιμής και παραγώγου της ζητούμενης συνάρτησης. Ειδικότερα κατά την επίλυση ενός γραμμικού ΠΣΤ δεύτερης τάξης ελλειπτικού τύπου ακολουθείται η παρακάτω μεθοδολογία:

Θεωρούμε το ΠΣΤ το οποίο ορίζεται σε ένα χωρίο Ω, που είναι μια ορθογώνια περιοχή $\Omega \equiv [a,b] \times [c,d]$ και τους τελεστές $\mathcal{L}$ και $\mathcal{B}$ ως :

$$\begin{cases} \mathcal{L} \equiv a(x,y)\frac{\partial^2}{\partial x^2} + 2b(x,y)\frac{\partial^2}{\partial x \partial y} + c(x,y)\frac{\partial^2}{\partial y^2} + d(x,y)\frac{\partial}{\partial x} + e(x,y)\frac{\partial}{\partial y} + h(x,y) \\ \mathcal{B} \equiv \alpha(x,y) + \beta(x,y)\frac{\partial}{\partial \vec{n}} \end{cases} \tag{3.4}$$

Η συνθήκη $a(x,y)c(x,y) > b^2(x,y)$ χαρακτηρίζει τον **ελλειπτικό** τύπο του προβλήματος και συνεπάγεται ότι οι συναρτήσεις $a(x,y)$ και $c(x,y)$ είναι ομόσημες και μη μηδενικές. Για την αριθμητική επίλυση του ΠΣΤ με τη μέθοδο πεπερασμένων στοιχείων Collocation ακολουθούμε τα παρακάτω βήματα θεωρώντας ότι το πεδίο ορισμού του ΠΣΤ έχει μετασχηματιστεί στο μοναδιαίο τετράγωνο.

- **Βήμα 0:** Θεωρούμε ομοιόμορφο διαμερισμό των διαστημάτων $I^x = I^y = [0,1]$ σε n_s υποδιαστήματα $I^x_m = I^y_m\ ,\ m = 1\ ,\ \ldots\ ,\ n_s$ τα οποία παράγουν ένα ομοιόμορφο πλέγμα με βήμα διακριτοποίησης $h = \frac{1}{n_s}$ και συντεταγμένες κόμβων (x_i, y_j), όπου $x_i = (i-1)h$ και $y_j = (j-1)h$, με $i, j = 1\ ,\ \ldots\ ,\ (n_s+1)$. Το Σχήμα 3.1 εμφανίζει την διαμέριση του Ω για $n_s = 6$.



Σχήμα 3.1: Γεωμετρική απεικόνιση της διαμέρισης του πεδίου Ω

- **Βήμα 1:** Ως συναρτήσεις βάσης επιλέγονται τα Hermite Bicubic πολυώνυμα $\Phi_{i,j}(x,y)$ και η συνάρτηση $u(x,y)$ προσεγγίζεται από την

$$u(x,y)\ \simeq\ u_n(x,y)\ =\ \sum_{i=1}^{\tilde{n}}\sum_{j=1}^{\tilde{n}} a_{i,j}\Phi_{i,j}(x,y)\ \ , \tag{3.5}$$

όπου $\tilde{n} = 2(n_s + 1)$.

- **Βήμα 2:** Ως μέθοδος διακριτοποίησης επιλέγεται η μέθοδος της **Collocation**, η οποία κατασκευάζει το γραμμικό σύστημα $C\tilde{\mathbf{a}} = \tilde{\mathbf{b}}$ απαιτώντας οι συνθήκες $\mathcal{L}u_n - f = 0$ και $\mathcal{B}u_n - g = 0$ να ισχύουν για n καθορισμένα εσωτερικά και συνοριακά collocation σημεία.

- **Βήμα 3:** Για την επίλυση του αραιού και γενικού γραμμικού συστήματος $C\tilde{\mathbf{a}} = \tilde{\mathbf{b}}$ επιλέγεται κάποια επαναληπτική μέθοδος.

3.2.1 Πολυώνυμα Hermite

Τα τμηματικά κυβικά πολυώνυμα Hermite μέσω των οποίων θα κατασκευαστεί η βάση του χώρου στον οποίο θα αναζητηθεί η λύση, ορίζονται ως εξής[4, 41] :

$$\Phi(x) \doteq \begin{cases} \Phi_+(x) & , \quad x \in [0,1] \\ \Phi_-(x) & , \quad x \in [-1,0] \\ 0 & , \quad x \notin [-1,1] \end{cases} , \tag{3.6}$$

$$\Psi(x) \doteq \begin{cases} \Psi_+(x) & , \quad x \in [0,1] \\ \Psi_-(x) & , \quad x \in [-1,0] \\ 0 & , \quad x \notin [-1,1] \end{cases} , \tag{3.7}$$

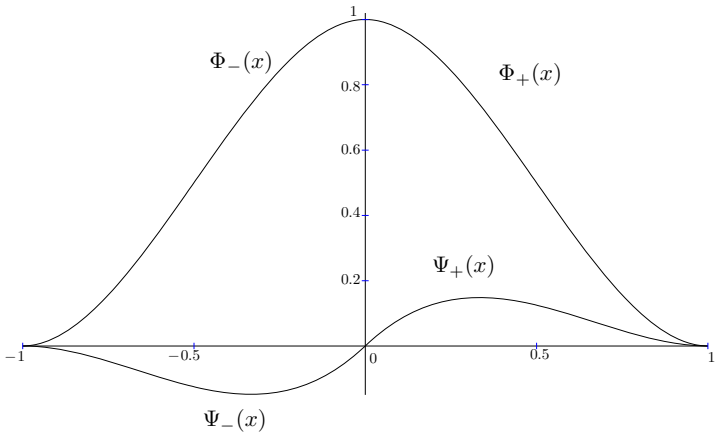
όπου

$$\Phi_+(x) \doteq \begin{cases} (1-x)^2(1+2x) & , \quad x \in [0,1] \\ 0 & , \quad x \notin [0,1] \end{cases} , \tag{3.8}$$

$$\Phi_-(x) = \Phi_+(-x) \doteq \begin{cases} (1+x)^2(1-2x) & , \quad x \in [-1,0] \\ 0 & , \quad x \notin [-1,0] \end{cases} , \tag{3.9}$$

$$\Psi_+(x) \doteq \begin{cases} x(1-x)^2 & , \quad x \in [0,1] \\ \\ 0 & , \quad x \notin [0,1] \end{cases} , \tag{3.10}$$

$$\Psi_-(x) = -\Psi_+(-x) \doteq \begin{cases} x(1+x)^2 & , \quad x \in [-1,0] \\ \\ 0 & , \quad x \notin [-1,0] \end{cases} . \tag{3.11}$$

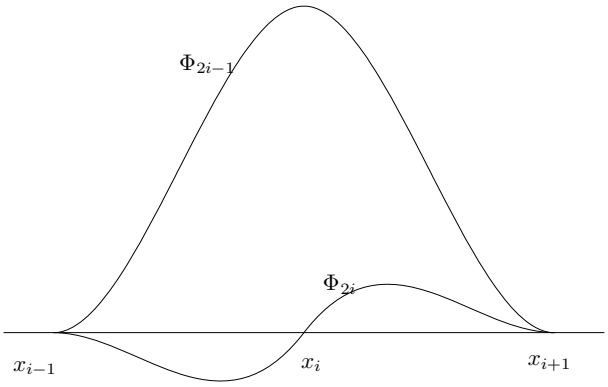


Σχήμα 3.2: Κυβικά πολυώνυμα Hermite.

Το Σχήμα 3.2 εμφανίζει τα κυβικά πολυώνυμα Hermite, όπως αυτά ορίζονται στο $[-1,1]$. Αν θεωρήσουμε τη διακριτοποίηση ως προς τη μία κατεύθυνση, τότε σε κάθε κόμβο x_m αντιστοιχούν δύο συναρτήσεις και ορίζονται ως εξής:

$$\Phi_{2m-1}(x) \doteq \begin{cases} \Phi\left(\frac{x-x_m}{h}\right) & , \quad x \in I_{m-1} \cup I_m \\ \\ 0 & , \quad \text{διαφορετικά} \end{cases} , \tag{3.12}$$

$$\Phi_{2m}(x) \doteq \begin{cases} \Psi\left(\frac{x-x_m}{h}\right) & , \quad x \in I_m \cup I_{m-1} \\ \\ 0 & , \quad \text{διαφορετικά} \end{cases} , \tag{3.13}$$



Σχήμα 3.3: Πολυώνυμα Hermite ορισμένα στον κόμβο x_i.

όπου $m = 1, \ldots, (n_s + 1)$, $I_i \equiv [x_i, x_{i+1}]$, $i = 1, \ldots, n_s$ μπορεί να θεωρηθούν μονοδιάστατα πεπερασμένα στοιχεία ως προς την x κατεύθυνση για παράδειγμα.

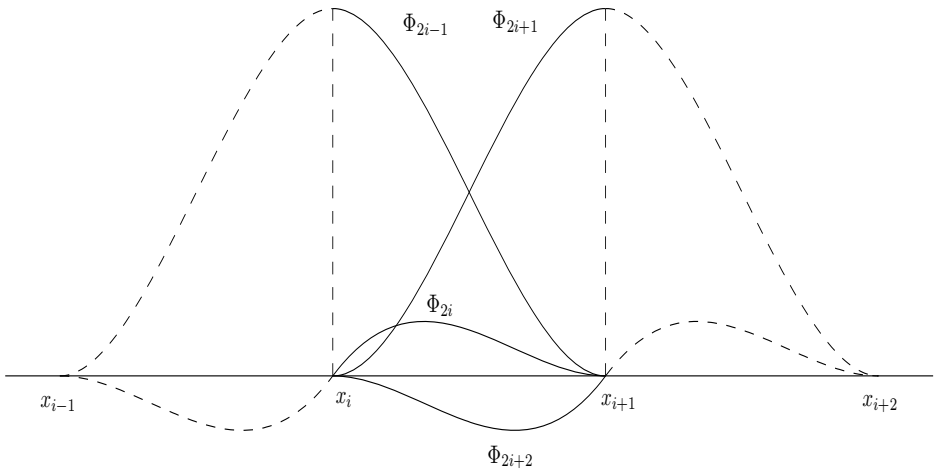
Για να ισχύουν οι ορισμοί (3.12) και (3.13) για $m = 1$ και $m = n_s + 1$, θεωρούμε δύο εικονικούς κόμβους $x_0 := -h$ και $x_{n_s+2} := 1 + h$.

Στο Σχήμα 3.3 εμφανίζεται ένας τυχαίος μονοδιάστατος κόμβος x_i και παρουσιάζονται οι αντίστοιχες συναρτήσεις, όπως αυτές ορίζονται σε αυτό το κόμβο. Θα ισχύουν οι εξής ιδιότητες

$$\begin{cases} \Phi_{2m-1}(x_i) &= h\frac{d}{dt}\Phi_{2m}(x_i) \;=\; \delta^i_m \\ \Phi_{2m}(x_i) &= \frac{d}{dt}\Phi_{2m-1}(x_i) \;=\; 0 \end{cases} \tag{3.14}$$

για όλα τα $m = 1, \ldots, (n_s + 1)$ και όπου δ^i_m : Δέλτα του Kronecker. Είναι φανερό ότι σε κάθε υποδιάστημα I_i διέρχονται τέσσερα μόνο μη μηδενικά πολυώνυμα Hermite με δείκτες : Φ_{2i-1} , Φ_{2i} , Φ_{2i+1} και Φ_{2i+2} όπου εμφανίζονται στο Σχήμα 3.4.

Βασισμένοι στις ιδιότητες των μονοδιάστατων πολυωνύμων Hermite που παρουσιάστηκαν παραπάνω προκύπτουν οι ιδιότητες των διδιάστατων bicubic πολυωνύμων Hermite. Έτσι, παρατηρούμε ότι σε κάθε διδιάστατο κόμβο (x_i, y_j) ορίζονται τα παρακάτω τέσσερα



Σχήμα 3.4: Μη μηδενικά Πολυώνυμα Hermite στο υποδιάστημα $[x_i, x_{i+1}]$.

Hermite Bicubic πολυώνυμα :

$$\begin{cases} \Phi_{2i-1,2j-1}(x,y) &= \Phi_{2i-1}(x)\Phi_{2j-1}(y) \\ \Phi_{2i-1,2j}(x,y) &= \Phi_{2i-1}(x)\Phi_{2j}(y) \\ \Phi_{2i,2j-1}(x,y) &= \Phi_{2i}(x)\Phi_{2j-1}(y) \\ \Phi_{2i,2j}(x,y) &= \Phi_{2i}(x)\Phi_{2j}(y) \end{cases} \tag{3.15}$$

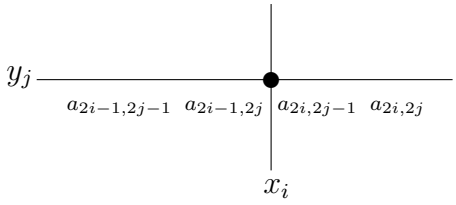
με τις εξής ιδιότητες :

$$\begin{cases} \Phi_{2i-1,2j-1}(x_i,y_j) = 1 \quad , \quad h\frac{\partial}{\partial y}\Phi_{2i-1,2j}(x_i,y_j) = 1 \\ h\frac{\partial}{\partial x}\Phi_{2i,2j-1}(x_i,y_j) = 1 \quad , \quad h^2\frac{\partial^2}{\partial x \partial y}\Phi_{2i,2j}(x_i,y_j) = 1 \end{cases} . \tag{3.16}$$

Σαν άμεση συνέπεια των προηγουμένων σχέσεων προκύπτει ότι

$$\begin{cases} u_n(x_i,y_j) = a_{2i-1,2j-1} \quad , \quad h\frac{\partial}{\partial y}u_n(x_i,y_j) = a_{2i-1,2j} \\ h\frac{\partial}{\partial x}u_n(x_i,y_j) = a_{2i,2j-1} \quad , \quad h^2\frac{\partial^2}{\partial x \partial y}u_n(x_i,y_j) = a_{2i,2j} \end{cases} , \tag{3.17}$$

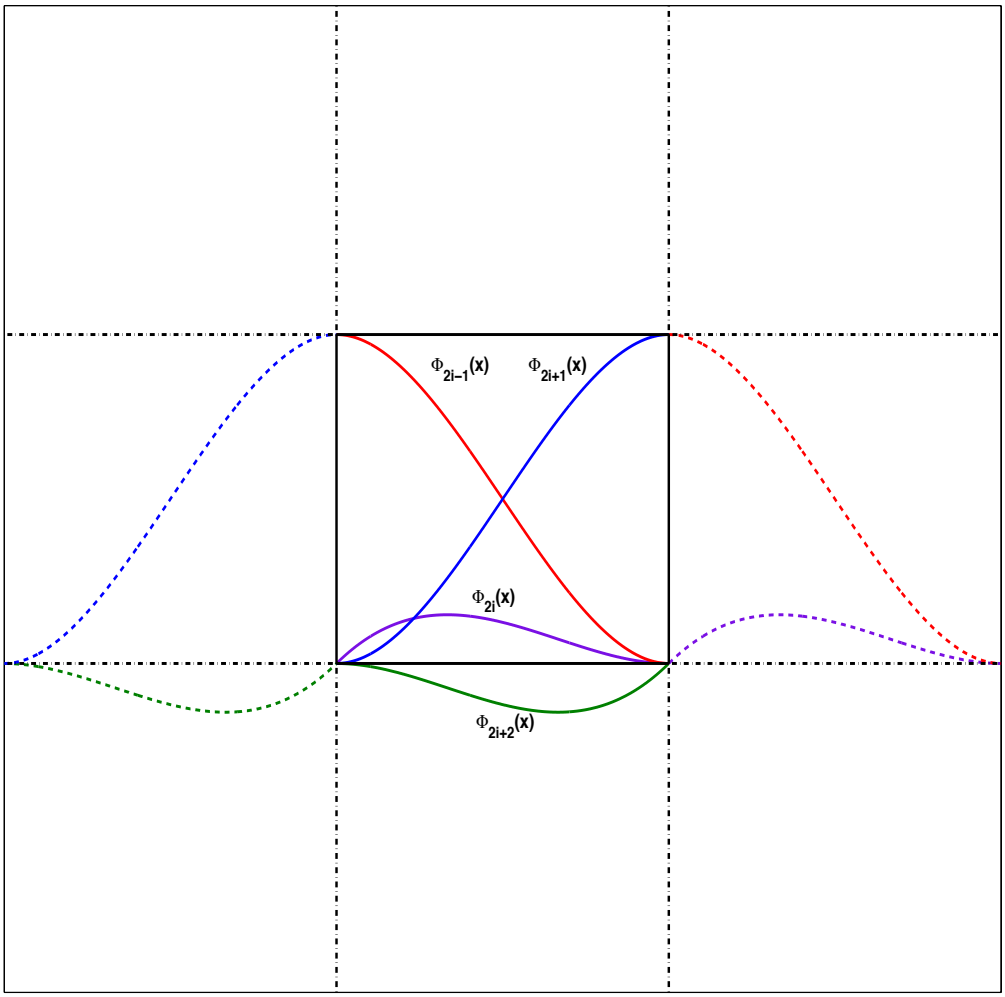
όπου $i\ ,\ j\ =\ 1,\ldots,(n_s+1)$ και στο παρακάτω σχήμα εμφανίζονται οι τέσσερις άγνωστοι όπως αυτοί αντιστοιχούν στον κόμβο (x_i, y_i).



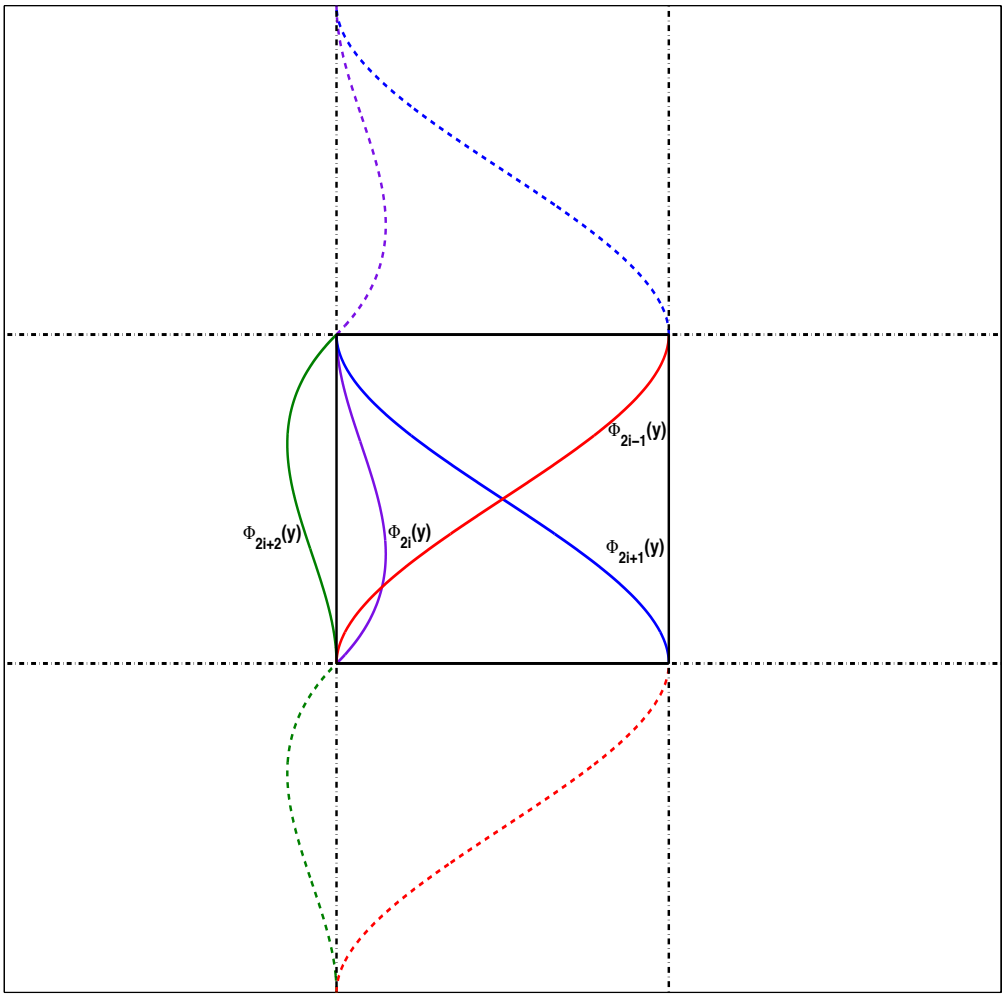
Επιπλέον παρατηρούμε ότι σε κάθε πεπερασμένο στοιχείο I_{ij}^{xy} αντιστοιχούν 16 μη μηδενικές συναρτήσεις βάσης (4 από κάθε κατεύθυνση), όπως εμφανίζει το σχήμα και επομένως για $(x, y) \in I_{ij}^{xy}$ ισχύει ότι :

$$u_n(x,y)\ =\ \sum_{k=2i-1}^{2i+2}\sum_{l=2j-1}^{2j+2}\alpha_{k,l}\Phi_k(x)\Phi_l(y)\ \ . \tag{3.18}$$

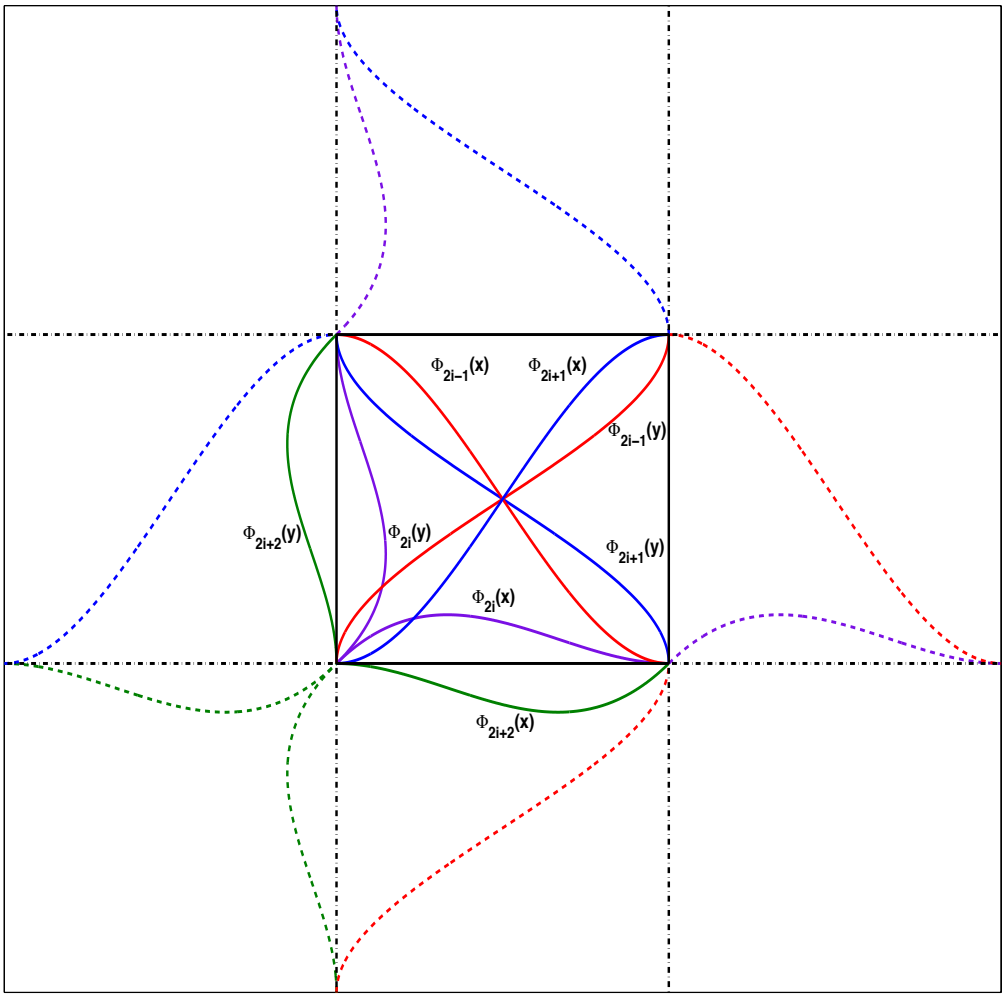
Γιάυτό το λόγο κάθε πεπερασμένο στοιχείο I_{ij}^{xy} είναι στοιχείο με 16 βαθμούς ελευθερίας. Στα σχήματα 3.5, 3.6 και 3.7 που ακολουθούν παρουσιάζονται οι συναρτήσεις βάσης σε κάθε κατεύθυνση και συνολικά στις δύο κατευθύνσεις, σε ένα πεπερασμένο στοιχείο.



Σχήμα 3.5: Συναρτήσεις βάσης στη διάσταση x σε ένα πεπερασμένο στοιχείο.



Σχήμα 3.6: Συναρτήσεις βάσης στη διάσταση y σε ένα πεπερασμένο στοιχείο.



Σχήμα 3.7: Συναρτήσεις βάσης σε 2 διαστάσεις σε ένα πεπερασμένο στοιχείο.

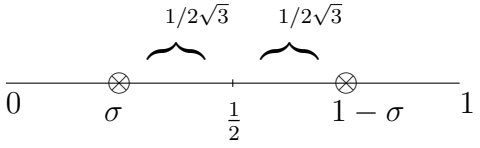
3.2.2 Σημεία Collocation

Χρειάζονται $n_I = 4n_s^2$ **εσωτερικά** collocation σημεία και $n_b = 4(2n_s+1)$ **συνοριακά** collocation σημεία. Το πλήθος $n_I + n_b$ των collocation εξισώσεων θα ισούται με τον αριθμό των αγνώστων. Εφαρμόζοντας τις σχέσεις των συνοριακών συνθηκών με τη ταυτόχρονη χρήση τέταρτης τάξης σφάλματος προσεγγιστικών σχέσεων πεπερασμένων διαφορών των παραγώγων (κεντρικών, προς τα εμπρός και πίσω) είναι εφικτός ο άμεσος προσδιορισμός της προσέγγισης όλων των τιμών της λύσης του ΠΣΤ, καθώς και των κατά κατεύθυνση τιμών των παραγώγων πάνω στα σημεία διακριτοποίησης του πλέγματος του συνόρου $\partial\Omega$. Από τις συνοριακές συνθήκες δηλαδή προσδιορίζονται άμεσα $4(2n_s + 1)$ το πλήθος άγνωστοι και απαλοίφονται από το γραμμικό σύστημα. Έτσι έχουμε να υπολογίσουμε $n = 4n_s^2$ αγνώστους, όσο δηλαδή και το πλήθος των εσωτερικών collocation σημείων. Είναι γνωστό [5] ότι η επιλογή των Gauss σημείων ως σημείων Collocation για προβλήματα ελλειπτικού τύπου μπορεί να θεωρηθεί ως μια άριστη επιλογή. Τα σημεία Gauss στο διάστημα $[-1, 1]$ είναι η απεικόνιση των ρίζων του Legendre πολυωνύμου δευτέρου βαθμού $\frac{1}{2}(3x^2-1) = 0$, δηλαδή τα σημεία $\mp\frac{\sqrt{3}}{3}$. Ο μετασχηματισμός των σημείων Gauss στο πεπερασμένο στοιχείο $I_i = [x_i, x_{i+1}]$ οδηγεί στις σχέσεις

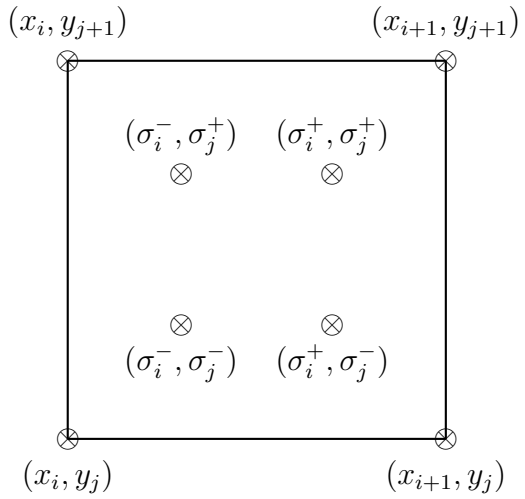
$$\sigma_{2i} \;=\; \tfrac{x_i+x_{i+1}}{2} - \tfrac{1}{\sqrt{3}}\tfrac{h_i}{2} \quad , \quad \sigma_{2i+1} \;=\; \tfrac{x_i+x_{i+1}}{2} + \tfrac{1}{\sqrt{3}}\tfrac{h_i}{2} \;. \tag{3.19}$$

Για την περίπτωση ομοιόμορφου διαμερισμού του Ω τα σημεία Gauss προκύπτουν από τη σχέση $\sigma_i^{\pm} = \frac{h}{2}(2i - 1 \pm \frac{\sqrt{3}}{3})$.

Παρατηρούμε από το σχήμα που ακολουθεί ότι τα σημεία Gauss στο $[0, 1]$ έχουν συντεταγμένες σ και $1 - \sigma$



όπου $\sigma = \frac{1}{2} - \frac{1}{2\sqrt{3}}$.



Σχήμα 3.8: Τα τέσσερα σημεία Gauss στο πεπερασμένο στοιχείο I_{ij}^{xy}.

Θα ισχύουν οι παρακάτω σχέσεις για τις παραγώγους k-τάξης των κυβικών πολυωνύμων Hermite ($D^k = \frac{d^k}{dx^k}$)

$$\begin{cases} D^k\Phi_{2i-1}(\sigma_{2i}) = \frac{1}{h^k}D^k\Phi(\sigma) \quad , \quad D^k\Phi_{2i-1}(\sigma_{2i+1}) = \frac{1}{h^k}D^k\Phi(1-\sigma) \\ D^k\Phi_{2i}(\sigma_{2i}) = \frac{1}{h^{k-1}}D^k\Psi(\sigma) \quad , \quad D^k\Psi_{2i-1}(\sigma_{2i+1}) = \frac{1}{h^{k-1}}D^k\Psi(1-\sigma) \\ D^k\Phi(1-\sigma) = D^k(1-\Phi(\sigma)) = (-1)^k D^k\Phi(\sigma) \end{cases} \tag{3.20}$$

για κάθε $i = 1, \ldots, n_s$.

3.2.3 Βασικοί Collocation πίνακες

Αν επιλέξουμε τα σημεία Gauss ως εσωτερικά collocation σημεία, ορίζονται οι παρακάτω πίνακες, για $l = 0, 1, 2$

$$k_i^l = D^l\Phi_k(\sigma_j)_{k=2i-1,j=2i}^{2i+2\quad 2i+1} \tag{3.21}$$

Από την τελευταία σχέση υπολογίζουμε τους βασικούς πίνακες (element matrices) για κάθε πεπερασμένο στοιχείο μιας διάστασης I_i:

$$k^0 = \begin{bmatrix} a & hb & 1-a & -h\bar{b} \\ 1-a & h\bar{b} & a & -hb \end{bmatrix} \quad , \tag{3.22}$$

$$a = \tfrac{9+4\sqrt{3}}{18}, b = \tfrac{3+\sqrt{3}}{36}, \bar{b} = \tfrac{3-\sqrt{3}}{36}$$

$$k^1 = \frac{1}{h} \begin{bmatrix} -1 & h\hat{b} & 1 & -h\hat{b} \\ -1 & -h\hat{b} & 1 & h\hat{b} \end{bmatrix} \quad , \tag{3.23}$$

$$\hat{b} = \tfrac{\sqrt{3}}{6}$$

$$k^2 = \frac{1}{h^2} \begin{bmatrix} -a' & -hb' & a' & -h\bar{b}' \\ a' & h\bar{b}' & -a' & hb \end{bmatrix} \quad , \tag{3.24}$$

$$a' = 2\sqrt{3}, \bar{b}' = \sqrt{3}-1, b' = \sqrt{3}+1$$

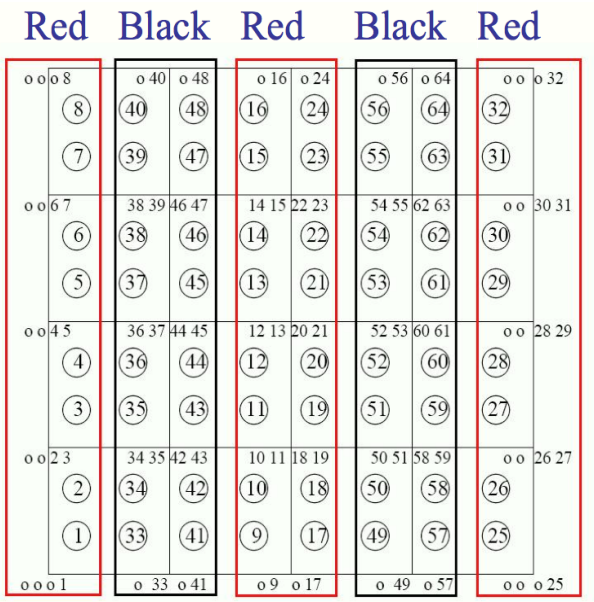
Με την βοήθεια των βασικών πινάκων k θα κατασκευαστούν οι K^l πίνακες, οι οποίοι αντιστοιχούν στο σύνολο των πεπερασμένων μονοδιάστατων στοιχείων

$$K^l = \begin{bmatrix} \kappa_2^l & \kappa_3^l & \kappa_4^l & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & \kappa_1^l & \kappa_2^l & \kappa_3^l & \kappa_4^l & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & \kappa_1^l & \kappa_2^l & \kappa_3^l & \kappa_4^l & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \kappa_1^l & \kappa_2^l & \kappa_3^l \end{bmatrix} . \tag{3.25}$$

Ο συμβολισμός κ_i^l αναφέρεται στην i-στήλη του βασικού πίνακα k^l, για $l = 0, 1, 2$.

3.2.4 Το Collocation γραμμικό σύστημα

Η δομή του Collocation πίνακα καθορίζεται από την επιλογή του τρόπου αρίθμησης αγνώστων και εξισώσεων στο πλέγμα διακριτοποίησης. Είναι γνωστό ότι η κατασκευή αποδοτικού παράλληλου αλγορίθμου επίλυσης του γραμμικού συστήματος προϋποθέτει την αυξημένη απεξάρτηση ομάδων των αγνώστων. Έτσι επιλέγεται η γνωστή red-black αρίθμηση αγνώστων και εξισώσεων, που ομαδοποιεί αγνώστους και εξισώσεις, ώστε να αλληλοαπεξαρτοποιούνται[52]. Αναλυτικότερα, γίνεται ομαδοποίηση και χρωματισμός αγνώστων που αντιστοιχούν σε κάθε κάθετη γραμμή πλέγματος, έτσι ώστε να μην υπάρχουν γειτονικές ομάδες ίδιου χρώματος. Στη συνέχεια γίνεται αρίθμηση αγνώστων και εξισώσεων σύμφωνα με τη κάθετη λεξικογραφική μέθοδο για κάθε χρώμα ξεχωριστά, αρχίζοντας από αυτό της πρώτης στήλης. Το σχήμα 3.9 εμφανίζει τη red- black αρίθμηση για την περίπτωση τεσσάρων πεπερασμένων στοιχείων ανά κατεύθυνση.

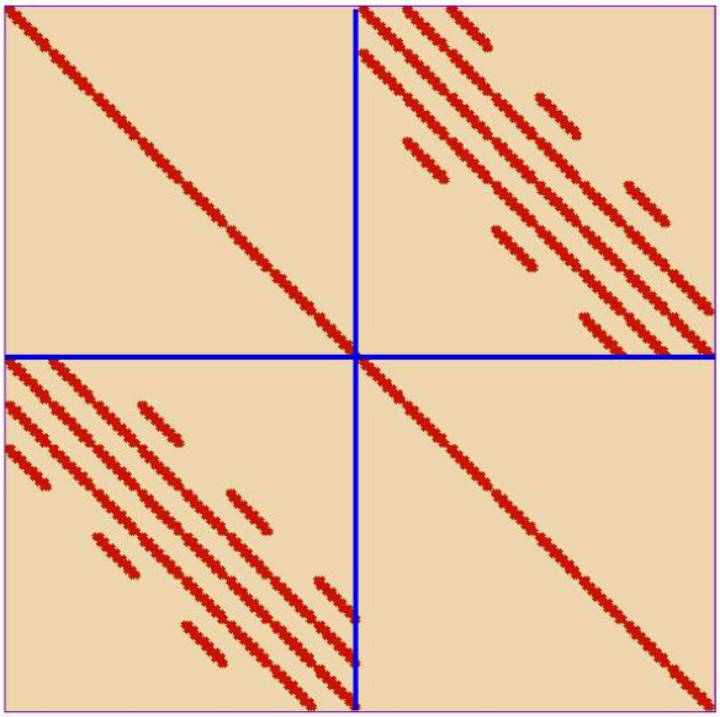


Σχήμα 3.9: Red-Black αρίθμηση των collocation αγνώστων και εξισώσεων για $n_s = 4$[44].

Οι μικροί σε μέγεθος αριθμοί απεικονίζουν την αρίθμηση των αγνώστων σε κάθε κόμβο, ενώ οι αριθμοί εντός των κύκλων την αρίθμηση των εξισώσεων ανά πεπερασμένο στοιχείο. Οι μικροί κύκλοι στο σύνορο υποδεικνύουν την ύπαρξη αγνώστων που υπολογίστηκαν

με χρήση των συνοριακών συνθηκών στη θέση αυτή.

Στο σχήμα 3.10 παρουσιάζεται σχηματικά η δομή του παραγόμενου Collocation πίνακα μετά την εφαρμογή κατάλληλου μετασχηματισμού ομοιότητας [45, 47].



Σχήμα 3.10: Η δομή του Collocation πίνακα για $n_s = 4$.

Εύκολα καταλήγει κανείς στο συμπέρασμα ότι ο Collocation πίνακας θα έχει τη παρακάτω μορφή

$$C = \begin{bmatrix} D_R & H_B \\ H_R & D_B \end{bmatrix} , \tag{3.26}$$

όπου D_R και D_B είναι αντιστρέψιμοι block διαγώνιοι πίνακες. Στην περίπτωση όπου $n_s = 2p$ αυτοί θα έχουν τη μορφή

$$D_R = \text{diag}[\underbrace{A_2 \;\; 2A_1 \;\; 2A_2 \;\; \cdots \;\; 2A_1 \;\; 2A_2}_{2p-blocks} \;\; -A_2] , \tag{3.27}$$

$$D_B \;=\; 2\,\mathrm{diag}[\underbrace{A_1 \;\; A_2 \;\cdots\; A_1 \;\; A_2}_{2p-blocks}] \tag{3.28}$$

$$H_R \;=\; \begin{bmatrix} R_1 & R_2 & & & & \\ R_3 & R_1 & R_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & R_3 & R_1 & R_2 \\ & & & & R_3 & \hat{R}_1 \end{bmatrix} \tag{3.29}$$

$$H_B \;=\; \begin{bmatrix} B_1 & B_2 & & & & \\ B_3 & B_1 & B_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & B_3 & B_1 & B_2 \\ & & & & B_3 & B_1 \end{bmatrix} \tag{3.30}$$

όπου

$$R_1 \;=\; \begin{bmatrix} A_4 & A_3 \\ -A_4 & A_3 \end{bmatrix},\; \hat{R}_1 \;=\; \begin{bmatrix} A_4 & -A_4 \\ -A_4 & -A_4 \end{bmatrix},$$

$$R_2 \;=\; -\begin{bmatrix} A_4 & 0 \\ A_4 & 0 \end{bmatrix},\; R_3 \;=\; \begin{bmatrix} 0 & A_3 \\ 0 & -A_3 \end{bmatrix},$$

και

$$B_1 \;=\; \begin{bmatrix} A_3 & -A_4 \\ A_3 & A_4 \end{bmatrix},\; B_2 \;=\; \begin{bmatrix} 0 & 0 \\ A_3 & -A_4 \end{bmatrix},\; B_3 \;=\; -\begin{bmatrix} A_3 & A_4 \\ 0 & 0 \end{bmatrix}.$$

Η block μορφή των παραπάνω πινάκων εμπλέκει τέσσερις διάστασης $2n_s \times 2n_s$ πενταδιαγώνιους βασικούς πίνακες A_i για $i = 1, \ldots, 4$ [49].

Στις προηγούμενες εργασίες [45, 46, 49, 48] έχει παρουσιαστεί η αποδοτική επίλυση του red-black collocation γραμμικού συστήματος, με χρήση της επαναληπτικής μεθόδου SOR και των επαναληπτικών μεθόδων υποχώρων Krylov, βασισμένη στην ακόλουθη διάσπαση του πίνακα

$$A \;=\; D_A \;-\; L_A \;-\; U_A \;\;, \tag{3.31}$$

όπου

$$D_A = \begin{bmatrix} D_R & O \\ O & D_B \end{bmatrix} , \; L_A = \begin{bmatrix} O & O \\ -H_R & O \end{bmatrix} ,$$
$$U_A = \begin{bmatrix} O & -H_B \\ O & O \end{bmatrix} , \tag{3.32}$$

και για τα υπόλοιπα μέρη του γραμμικού συστήματος μπορεί να θεωρηθεί η ανάλογη διαμέριση των διανυσμάτων $\boldsymbol{x}$ και $\boldsymbol{b}$ σε

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_R \\ \boldsymbol{x}_B \end{bmatrix} \text{ και } \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_R \\ \boldsymbol{b}_B \end{bmatrix} . \tag{3.33}$$

Σε αυτές τις εργασίες, όπως επίσης και στην [6], η επαναληπτική μέθοδος BiCGSTAB, προρυθμισμένη είτε με το Symmetric Gauss-Seidel (SGS) είτε από το Gauss-Seidel (GS) επαναληπτικό σχήμα, επιτύγχανε ταχύτερα σύγκλιση σε σχέση με τη κλασσική μέθοδο SOR ή από οποιαδήποτε άλλη τύπου Krylov επαναληπτική μέθοδο.

Με σκοπό τη μείωση του χρόνου επίλυσης, αλλά και την αύξηση του βαθμού παραλληλοποίησης της μεθόδου BiCGSTAB, μπορεί να εφαρμοστεί η παρακάτω προρύθμιση

$$M_1^{-1} A M_2^{-1} M_2 \boldsymbol{x} = M_1^{-1} \boldsymbol{b} , \tag{3.34}$$

όπου M_1 είναι ο επαναληπτικός πίνακας της μεθόδου Gauss Seidel, βασισμένος στη διάσπαση

$$M_1 = D_A - L_A = D_A(I - D_A^{-1} L_A) \tag{3.35}$$

και

$$M_2 = I - D_A^{-1} U_A . \tag{3.36}$$

Οπότε το collocation γραμμικό σύστημα θα έχει τη παρακάτω μορφή

$$\begin{bmatrix} I & O \\ O & S \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_R + D_R^{-1} H_R \boldsymbol{x}_B \\ \boldsymbol{x}_B \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{b}}_R \\ \hat{\boldsymbol{b}}_B \end{bmatrix} , \tag{3.37}$$

όπου

$$S = D_B - H_R D_R^{-1} H_B \tag{3.38}$$

είναι το συμπλήρωμα του Schur του collocation πίνακα όσο αφορά το πίνακα D_B και

$$\hat{\boldsymbol{b}}_R \;=\; D_R^{-1}\boldsymbol{b}_R \quad \text{και} \quad \hat{\boldsymbol{b}}_B \;=\; \boldsymbol{b}_B - H_R\hat{\boldsymbol{b}}_R \;\;. \tag{3.39}$$

Ο ακόλουθος αλγόριθμος περιγράφει αναλυτικά τις διαδοχικές φάσεις υπολογισμών για την επίλυση του γραμμικού συστήματος:

Αλγόριθμος για τις collocation εξισώσεις του συμπληρώματος Schur

B1: *Επίλυση* $D_R\hat{\boldsymbol{b}}_R = \boldsymbol{b}_R$

B2: *Υπολογισμός* $\hat{\boldsymbol{b}}_B \;=\; \boldsymbol{b}_B - H_R\hat{\boldsymbol{b}}_R$

B3: *Επίλυση με BiCGSTAB* $S\,\boldsymbol{x}_B \;=\; \hat{\boldsymbol{b}}_B$

B4: *Υπολογισμός* $\hat{\boldsymbol{x}}_B \;=\; H_B\boldsymbol{x}_B$

B5: *Επίλυση* $D_R\hat{\boldsymbol{x}}_R = \hat{\boldsymbol{x}}_B$

B6: *Υπολογισμός* $\boldsymbol{x}_R \;=\; \hat{\boldsymbol{b}}_R - \hat{\boldsymbol{x}}_R$ [44, 46, 48, 49]

Κεφάλαιο 4

Επίλυση του Schur-Collocation γραμμικού συστήματος σε αρχιτεκτονικές υψηλών επιδόσεων

Για την επίλυση του Collocation γραμμικού συστήματος σε παράλληλες αρχιτεκτονικές υψηλών επιδόσεων σύμφωνα με την επαναληπτική μέθοδο του συμπληρώματος Schur, χρειάζεται να κατασκευαστεί κατάλληλος αλγόριθμος, στον οποίο θα υπάρχει αποδοτική οργάνωση των υπολογισμών. Βασικός παράγοντας για κάθε τέτοιο αλγόριθμο αποτελεί η γνώση της αρχιτεκτονικής των υπολογισμών. Στην εργασία [49] παρουσιάστηκε ένας αποδοτικός αλγόριθμος για την επίλυση του γραμμικού συστήματος με τη μέθοδο BiCGSTAB για αρχιτεκτονικές κατανεμημένης μνήμης.

Η πρώτη ενότητα αυτού του κεφαλαίου παρουσιάζει τη μέθοδο του Newton με επαναληπτική βελτίωση του υπολοίπου για το Schur-Collocation γραμμικό σύστημα. Η μέθοδος αυτή επιλύει το γραμμικό σύστημα του υπολοίπου Schur κάνοντας χρήση της μεθόδου BiCGSTAB για δικτυακές-κατανεμημένες αρχιτεκτονικές υπολογισμών και χρήση μικτής ακρίβειας στην απεικόνιση των αριθμητικών τιμών, υλοποιώντας ένα γενικευμένο αλγόριθμο σε σχέση με αυτόν της εργασίας [49]. Στη δεύτερη ενότητα παρουσιάζεται η κατασκευή παράλληλου αλγορίθμου για αρχιτεκτονικές κοινής μνήμης, στις οποίες συμμετέχουν πολυεπεξεργαστικά συστήματα και γραφικά υποσυστήματα.

4.1 Επαναληπτική βελτίωση υπολοίπου με τη μέθοδο Newton με χρήση μικτής ακρίβειας υπολογισμών

Στην ενότητα 1.3 έγινε αναφορά στη διαδικασία χρήσης μικτής ακρίβειας υπολογισμών σε ένα υβριδικό κώδικα GPU-CPU. Η μέθοδος ουσιαστικά εφαρμόζει διαφορετική ακρίβεια στην απεικόνιση δεδομένων σε διαφορετικά τμήματα του αλγορίθμου, και πιο συγκεκριμένα χρήση διπλής ακρίβειας σε υπολογισμούς που δεν έχουν μεγάλο φόρτο εργασίας κι έτσι δεν κοστίζουν χρονικά, ενώ γίνεται χρήση απλής ακρίβειας στα αυξημένου φόρτου υπολογιστικά τμήματα του αλγορίθμου. Η ιδέα της συγκεκριμένης μεθόδου στηρίχθηκε στο γεγονός ότι στις πρώτες GPUs υπήρχε μόνο η δυνατότητα αριθμητικών πράξεων πραγματικών αριθμών απλής ακρίβειας (με ταχύτητα εκτέλεσης πολλαπλάσια σε σχέση με ένα CPU (σχήμα 1.6) ενώ οι πρώτες προγραμματιζόμενες GPUs που υποστήριζαν αριθμητικές πράξεις πραγματικών αριθμών διπλής ακρίβειας είχαν ταχύτητα διεξαγωγής κατά περίπου $1/8$ μικρότερη σε σχέση με τη ταχύτητα εκτέλεσης υπολογισμών μεταξύ πραγματικών αριθμών απλής ακρίβειας (σχήμα 1.5). Συνεπώς, κρίθηκε ενδιαφέρουσα η ανάπτυξη ενός τέτοιου μοντέλου επίλυσης, το οποίο θα μπορούσε να εκμεταλλευτεί την υπολογιστική ισχύ των GPUs χωρίς να θυσιαστεί η ακρίβεια των αποτελεσμάτων της επίλυσης. Για την υλοποίηση αυτής της ιδέας, το τμήμα του αλγορίθμου που εκτελείται στο CPU διεξάγει υπολογισμούς διπλής ακρίβειας, ενώ τα τμήματα που εκτελούνται στη GPU διεξάγουν υπολογισμούς απλής ακρίβειας. Έτσι, επιτυγχάνεται και μείωση του κόστους μεταφοράς των δεδομένων μεταξύ CPU-GPU, αλλά και μείωση της χρήσης μνήμης στη GPU, αφού γίνεται χρήση απλής και όχι διπλής ακρίβειας αριθμητικών δεδομένων.

Μια ιδιαίτερα γνωστή και απλή μέθοδος χρήσης μικτής ακρίβειας υπολογισμών για την επίλυση γραμμικών συστημάτων είναι η **επαναληπτική βελτίωση του υπολοίπου**, σύμφωνα με την οποία το υπόλοιπο μιας διαδικασίας επίλυσης χρησιμοποιείται ως δεξί μέλος σε ένα γραμμικό σύστημα διόρθωσης της λύσης.

Η παραπάνω διαδικασία στηρίζεται στη **μέθοδο Newton**, με την οποία επιτυγχάνονται προοδευτικά καλύτερες προσεγγίσεις μιας συνάρτησης $f(\cdot)$, διορθώνοντας τη προσέγγιση της λύσης x_i μέσω της σχέσης της μεθόδου Newton

$$x_{i+1} \;=\; x_i - (\nabla f(x_i))^{-1} f(x_i) \tag{4.1}$$

Η παραπάνω σχέση, για τη συνάρτηση υπολοίπου $f(x) = b - Ax$, όπου $\nabla f = -A$, γράφεται ως

$$x_{i+1} \;=\; x_i + A^{-1}(b - Ax_i) \tag{4.2}$$

οπότε ορίζοντας το υπόλοιπο στο επαναληπτικό βήμα i ως $r_i = b - Ax$, προκύπτει ότι

$$x_{i+1} \;=\; x_i + A^{-1} r_i \tag{4.3}$$

ενώ, αν θεωρηθεί $c_i = A^{-1} r_i$ στη διόρθωση της λύσης x_i, ο αλγόριθμος της διόρθωσης του υπολοίπου περιγράφεται με τα παρακάτω βήματα

Αλγόριθμος της διόρθωσης υπολοίπου

B1: *Επιλογή αρχικής τιμής για* x_0

B2: *Υπολογισμός αρχικού υπολοίπου* $r_0 = b - Ax_0$

B3: **while** $(\|Ax_i - b\| > \epsilon \|r_0\|$ **do**

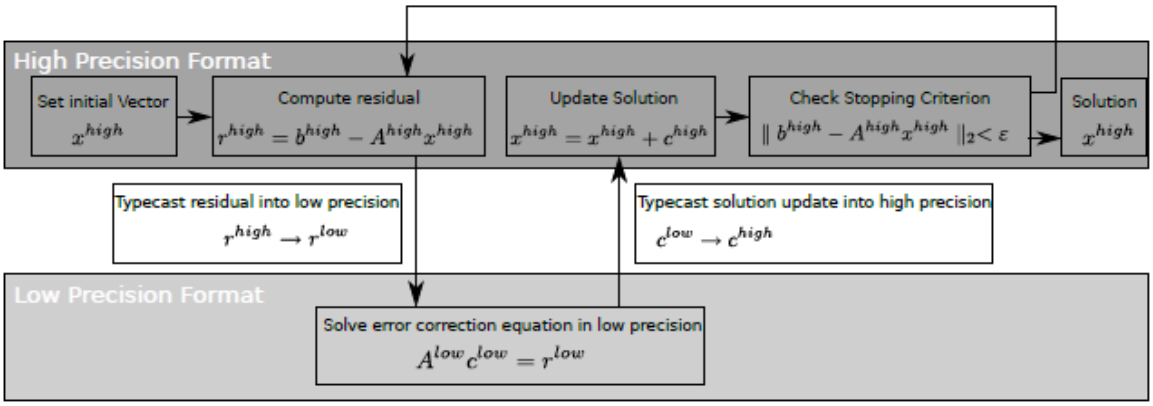
B4: *Υπολογισμός* $r_i = b - Ax_i$

B5: *Επίλυση:* $Ac_i = r_i$

B6: *Διόρθωση λύσης:* $x_{i+1} \;=\; x_i + c_i$

B7: **end while**

Η μέθοδος επαναληπτικής βελτίωσης μικτής ακρίβειας (mixed precision iterative refinement), μπορεί να αποτυπωθεί αλγοριθμικά στο σχήμα 4.1.



Σχήμα 4.1: Αλγόριθμος της μεθόδου επαναληπτικής βελτίωσης μικτής ακρίβειας, [1].

Κατά την υλοποίηση του παραπάνω αλγορίθμου, δημιουργείται ένας επιλύτης μικτής ακρίβειας, στον οποίο η επίλυση του γραμμικού συστήματος διόρθωσης του υπολοίπου $Ac = r$ πραγματοποιείται σε απλή ακρίβεια ($.^{low}$), ενώ η διόρθωση της προσέγγισης της λύσης και όλες οι υπόλοιπες διαδικασίες σε διπλή ακρίβεια ($.^{high}$). Για την επίλυση του γραμμικού συστήματος $Ac = r$ μπορούν να χρησιμοποιηθούν μέθοδοι υποχώρων Krylov ή άμεσες μέθοδοι επίλυσης, ανάλογα τη δομή του πίνακα A. Με επιλογή μιας επαναληπτικής μεθόδου υποχώρων Krylov (CG, GMRES, BiCG) δημιουργείται μια **εσωτερική** επαναληπτική διαδικασία για την επίλυση του $Ac_i = r_i$, η οποία εκτελείται σε κάθε **εξωτερική** επανάληψη i διόρθωσης του επιλύτη μικτής ακρίβειας. Για τον τερματισμό κάθε επαναληπτικής διαδικασίας χρησιμοποιούνται κριτήρια τερματισμού, εσωτερικό και εξωτερικό, ο κατάλληλος συνδυασμός των οποίων έχει καταλυτικό ρόλο στην απόδοση του επιλύτη μικτής ακρίβειας. Μεγάλος αριθμός εσωτερικών επαναλήψεων σε απλή ακρίβεια οδηγεί σε λιγότερες εξωτερικές επαναλήψεις σε διπλή ακρίβεια, ενώ ελάττωση του αριθμού οδηγεί σε περισσότερες επανεκκινήσεις του επιλύτη. Για τη βελτιστοποίηση της απόδοσης, δεν μπορεί να γίνει εύκολα ολοκληρωμένη θεωρητική ανάλυση, είναι δεδομένο όμως ότι απαιτείται καλή γνώση των ιδιοτήτων και της συμπεριφοράς της ε-

παναληπτικής μεθόδου και της δομής του γραμμικού συστήματος που επιλύεται [1]. Στη παρούσα επίλυση του γραμμικού συστήματος διόρθωσης υπολοίπου επιλέχθηκε η μέθοδος BiCGSTAB με χρήση του συμπληρώματος Schur.

Ο αντίστοιχος αλγόριθμος θα είναι:

Αλγόριθμος Newton του συμπληρώματος Schur με χρήση BiCGSTAB

B1: *Επιλογή αρχικής τιμής για* ${x_0}^{high}$

B2: *Υπολογισμός αρχικού υπολοίπου* ${r_0}^{high} = b - A^{high}{x_0}^{high}$, $A^{high} \rightarrow A^{low}$

B3: **while** $(\|A^{high}{x_i}^{high} - b\| > \epsilon \, \|{r_0}^{high}\|$ **do**

B4: *Υπολογισμός* $r_i^{high} = b - A^{high}{x_i}^{high}$

B5: ${r_i}^{high} \rightarrow {r_i}^{low}$

B6: *Επίλυση* $A^{low}{c_i}^{low} = {r_i}^{low}$ *με τη μέθοδο Schur Complement*

$$A^{low} = \begin{bmatrix} D_R & H_B \\ H_R & D_B \end{bmatrix}, c_i^{low} = \begin{bmatrix} c_R^{(i)} \\ c_B^{(i)} \end{bmatrix} \text{ και } r_i^{low} = \begin{bmatrix} r_R^{(i)} \\ r_B^{(i)} \end{bmatrix}$$

SC1: *Επίλυση* $D_R\hat{r}_R^{(i)} = r_R^{(i)}$

SC2: *Υπολογισμός* $\hat{r}_B^{(i)} = r_B^{(i)} - H_R\hat{r}_R^{(i)}$

SC3: *Επίλυση με* $BiCGSTAB$ $S\, c_B^{(i)} = \hat{r}_B^{(i)}$

SC4: *Υπολογισμός* $\hat{c}_B^{(i)} = H_B c_B^{(i)}$

SC5: *Επίλυση* $D_R\hat{c}_R^{(i)} = \hat{c}_B^{(i)}$

SC6: *Υπολογισμός* $c_R^{(i)} = \hat{r}_R^{(i)} - \hat{c}_R^{(i)}$

B7: $c_i^{low} \rightarrow c_i^{high}$

B8: *Διόρθωση λύσης:* ${x_{i+1}}^{high} = {x_i}^{high} + {c_i}^{high}$

B9: **end while**

Γίνεται αντιληπτό ότι, συγκρίνοντας το συγκεκριμένο επιλύτη μικτής ακρίβειας με την απλή επαναληπτική μέθοδο του Schur, οι υπολογισμοί που πραγματοποιούνται στον επιλύτη μικτής ακρίβειας είναι περισσότεροι. Αυτό συμβαίνει διότι σε κάθε εξωτερική επανάληψη i του επιλύτη μικτής ακρίβειας υπολογίζεται η τιμή του υπολοίπου r_i, με την εκτέλεση ενός πολλαπλασιασμού πίνακα με διάνυσμα και μιας πρόσθεσης διανυσμάτων, ενημερώνεται η προσέγγιση της λύσης x_i, ελέγχεται το κριτήριο τερματισμού, ενώ πραγματοποιούνται και δύο μετατροπές διανυσμάτων από απλή σε διπλή ακρίβεια και αντίστροφα. Συνεπώς, για να θεωρηθεί συμφέρουσα η χρήση του επιλύτη μικτής ακρίβειας θα πρέπει η χρονική επιβάρυνση που προκύπτει απο τους επιπλέον υπολογισμούς να υπερκαλύπτεται από το όφελος που προκύπτει από τη ταχύτερη εκτέλεση υπολογισμών σε αριθμητική απλής ακρίβειας της επίλυσης του γραμμικού συστήματος με τη διόρθωση της λύσης.

Στη περίπτωση δε υλοποίησης του αλγορίθμου σε αρχιτεκτονικές CPU-GPU, θα πρέπει να συνυπολογιστεί και η συχνή μεταφορά δεδομένων μεταξύ της μνήμης της GPU και της μνήμης του CPU, η περιορισμένη ταχύτητας προσπέλασης των οποίων (περίπου 6 Gb/sec) μειώνει την απόδοση του επιλύτη. Επιπλέον, τονίζεται ότι ο πίνακας A αποθηκεύεται σε δύο διαφορετικές μορφές, με τιμές απλής A^{low} και διπλής ακρίβειας A^{high}, για να αποφευχθεί το υψηλό κόστος της μετατροπής του σε κάθε επαναληπτικό βήμα της διαδικασίας. Για το λόγο αυτό, συμβαίνει το ίδιο και με έναν αριθμό βοηθητικών διανυσμάτων, συνεπώς αυξάνονται ανάλογα και οι απαιτήσεις σε μνήμη.

Οι νεότερες GPUs υποστηρίζουν πράξεις αριθμητικής διπλής ακρίβειας, με την ταχύτητα εκτέλεσης τους να είναι περίπου υποδιπλάσια αυτών της απλής ακρίβειας, ενώ ενσωματώνουν πλήρως το πρότυπο IEEE 754 για την αναπαράσταση πραγματικών αριθμών. Με βάση αυτή την πρόοδο στη τεχνολογία των GPUs, στην εργασία [10] προτείνεται η πλήρης εκτέλεση της επίλυσης ενός προβλήματος σε GPUs, διαχωρίζοντας το σε υποπροβλήματα με βάση την επιθυμητή ακρίβεια στους υπολογισμούς. Με τον τρόπο αυτό μειώνεται σε πολύ μεγάλο βαθμό το κόστος επικοινωνίας μεταξύ των μνημών CPU-GPU.

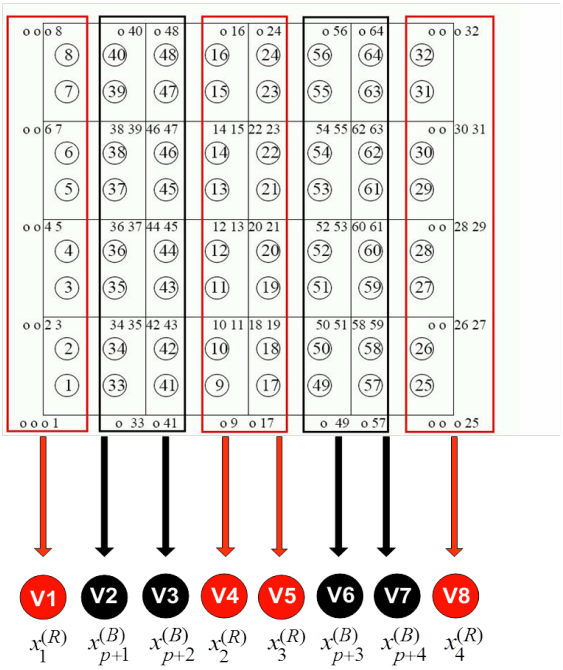
Έγινε υλοποίηση της επίλυσης του Collocation γραμμικού συστήματος με χρήση του συμπληρώματος Schur εφαρμόζοντας την επαναληπτική διαδικασία της μεθόδου Newton. Η μελέτη της συμπεριφοράς τόσο του σειριακού, αλλά και του παράλληλου αλγόριθμου που προέκυψε χρησιμοποιώντας τις επιμέρους παράλληλες διαδικασίες της εργασίας [;], οδήγησε στο συμπέρασμα ότι η χρήση μικτής ακρίβειας υπολογισμών επιβάρυνε αισθητά με επαναληπτικά βήματα τη διαδικασία επίλυσης. Έτσι, αν και ο παράλληλος αλγόριθμος της μεθόδου Newton για κατανεμημένης μνήμης αρχιτεκτονικές είναι αποδοτικός σε σχέση με τη σειριακή έκδοση του, η χρήση της μεθόδου Newton υστερεί σε απόδοση σε σχέση με την αντίστοιχη αποκλειστική χρήση της επαναληπτικής μεθόδου του συμπληρώματος Schur για την επίλυση του Collocation γραμμικού συστήματος. Για αυτό το λόγο στην επόμενη ενότητα παρουσιάζεται η κατασκευή αλγορίθμου για την επαναληπτική μέθοδο του συμπληρώματος Schur για αρχιτεκτονικές πολλαπλών υπολογιστικών πυρήνων κοινής μνήμης για CPU-GPU επιστημονικούς υπολογισμούς.

4.2 Επίλυση σε CPU-GPU αρχιτεκτονική υπολογισμών κοινής μνήμης

Οι σημαντικότεροι παράγοντες που λαμβάνονται υπ'οψη στο σχεδιασμό και την οργάνωση των υπολογισμών ενός παράλληλου αλγορίθμου είναι η αρχιτεκτονική διασύνδεσης επεξεργαστών του διαθέσιμου μηχανήματος καθώς και ο αριθμός και το είδος των επεξεργαστών του. Τα σημερινά υπολογιστικά συστήματα κοινής μνήμης αποτελούνται από λίγους σε πλήθος, αλλά ισχυρούς επεξεργαστικούς πυρήνες. Αντίθετα, μια GPU διαθέτει μερικές εκατοντάδες υπολογιστικών πυρήνων με δυνατότητα εκτέλεσης βασικών αριθμητικών πράξεων. Οι υπολογιστικοί αυτοί πυρήνες έχουν τη δική τους μνήμη κι έτσι θα πρέπει να ληφθεί υπ'οψη η συγκεκριμένη αρχιτεκτονική στο διαχωρισμό δεδομένων και υπολογισμών,ενώ επίσης βασική επιδίωξη είναι όλοι οι πυρήνες να παραμένουν απασχολημένοι κατά τη διάρκεια των υπολογισμών. Αυτό το *no idle core* υπολογιστικό μοντέλο απαιτεί ισοκατανομή και ισορροπία μεταξύ φόρτου επικοινωνίας και υπολογισμών.

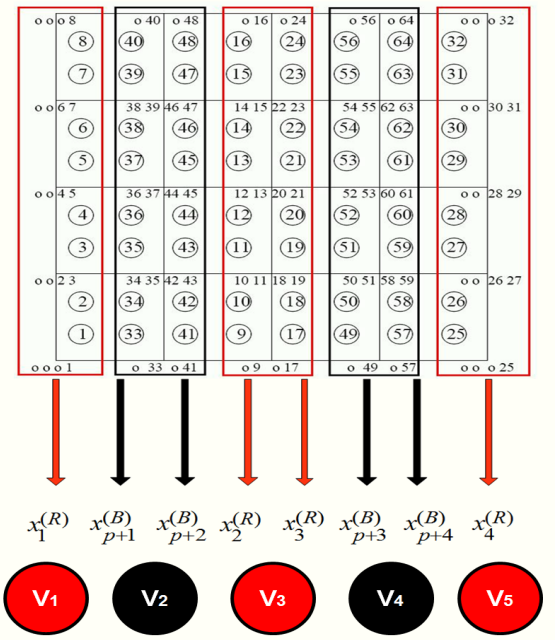
4.2.1 Αλγόριθμος για CPU-GPU υπολογιστικά περιβάλλοντα

Για τη κατασκευή αποδοτικών αλγορίθμων σε αρχιτεκτονικές CPU-GPU είναι η διευθέτηση της ανάθεσης των υπολογιστικών νημάτων (threads) σε κατάλληλους υπολογιστικούς πυρήνες. Αρχικά, μπορεί να θεωρηθεί μια εικονική αρχιτεκτονική με απεριόριστο αριθμό επεξεργαστών. Λαμβάνοντας υπ'όψη την αναγκαιότητα να έχουν τα threads ανεξάρτητα μεταξύ τους δεδομένα, την επιδίωξη για ελαχιστοποίηση του κόστους επικοινωνίας με τη μνήμη και το γεγονός ότι ο αριθμός των υποδιαστημάτων $n_s = 2p$ της διακριτοποίησης και προς τις δύο κατευθύνσεις x και y είναι άρτιος, γίνεται αντιληπτό ότι η κατάλληλη κατανομή επιτυγχάνεται με τη αντιστοίχιση ενός thread για κάθε μια από τις $2p+1$ κάθετες γραμμές του πλέγματος διακριτοποίησης. Αυτό παρουσιάζεται σχηματικά στην εικόνα 4.2 για τη περίπτωση $n_s = 4$ πεπερασμένων στοιχείων σε κάθε κατεύθυνση.



Σχήμα 4.2: Αντιστοίχιση των collocation αγνώστων σε υπολογιστικά νήματα για $n_s = 4$, [57].

Έτσι, τα *περιττής* αρίθμησης threads αντιστοιχούν σε *red* αγνώστους από τις κάθετες γραμμές πλέγματος, ενώ τα *άρτια* threads αντιπροσωπεύουν υπολογισμούς των *black* γραμμών πλέγματος. Εξαιτίας της $2p$-block διαμέρισης όλων των διανυσμάτων στους υπολογισμούς, είναι ξεκάθαρο ότι σε καθένα από τα περιττά threads V_{2i-1} , $i = 1, \ldots, p+1$, έχει ανατεθεί ο υπολογισμός των διανυσμάτων $\boldsymbol{t}_{2i-2}$ και $\boldsymbol{t}_{2i-1}$ της λύσης, ενώ κάθε άρτιο thread V_{2i} , $i = 1, \ldots, p$, θα πρέπει να υπολογίσει τα διανύσματα $\boldsymbol{t}_{2p+2i-1}$ και $\boldsymbol{t}_{2p+2i}$ της λύσης. Η ιδιαιτερότητα που παρουσιάζεται στα threads V_1 και V_{2p+1} οφείλεται στις συνοριακές συνθήκες. Θα πρέπει να σημειωθεί ότι αν γινόταν αντιστοίχιση αγνώστων που ανήκουν στην ίδια κάθετη γραμμή πλέγματος σε διαφορετικά threads θα υπήρχε αυξημένο κόστος επικοινωνίας μεταξύ των threads και της κύριας μνήμης επειδή υπάρχει ισχυρή εξάρτηση μεταξύ των αγνώστων που ανήκουν σε ίδιους υπολογιστικούς κόμβους πλέγματος. Για τον ίδιο λόγο οι υπολογισμοί αγνώστων που αντιστοιχούν στη δεξιά και αριστερή στήλη κάθε κάθετης γραμμής του πλέγματος θα χρειαστεί να διεξαχθούν από το ίδιο thread. Αυτό μας οδηγεί στη νέα αρίθμηση των red και black threads όπως εμφανίζει η εικόνα 4.3.



Σχήμα 4.3: Αντιστοίχιση των collocation αγνώστων σε threads ανάλογα την αρίθμηση για $n_s = 4$, [57].

Στον παράλληλο αλγόριθμο, όλες οι βασικές πράξεις διανυσμάτων γραμμικής άλγεβρας όπως εσωτερικά γινόμενα, πρόσθεση διανυσμάτων και πολλαπλασιασμοί τους με βαθμωτά μεγέθη, μπορούν να εκτελεστούν παράλληλα, βάσει της παραπάνω $2p$-block διαμέρισης όλων των διανυσμάτων. Ο λόγος που μπορούν να εκτελεστούν παράλληλα οι παραπάνω διαδικασίες είναι η ανεξαρτησία μεταξύ των δεδομένων που έχει το κάθε thread. Σύμφωνα με τον παραπάνω τρόπο αντιστοίχισης δεδομένων σε threads, οι παράλληλες διαδικασίες είναι ανεξάρτητες ως προς τα δεδομένα για όλους τους πολλαπλασιασμούς πίνακα με διάνυσμα και για τις πράξεις άμεσης επίλυσης κατά τους red και black κύκλους υπολογισμών.

Οι παρακάτω παράλληλοι αλγόριθμοι ενσωματώνουν όλες τις παραπάνω ιδιότητες για τον υπολογισμό ενός τυχαίου διανύσματος $\boldsymbol{t}$ διάστασης ${4n_s}^2$.

Παράλληλη διαδικασία RED Υπολογισμών

!$OMP PARALLEL DO DEFAULT(SHARED)

do $i = 0$ *έως* p

V_{2i+1} *υπολογίζει* $\boldsymbol{t}_{2i}$, $\boldsymbol{t}_{2i+1}$

enddo

!$OMP END PARALLEL DO

Παράλληλη διαδικασία BLACK Υπολογισμών

!$OMP PARALLEL DO DEFAULT(SHARED)

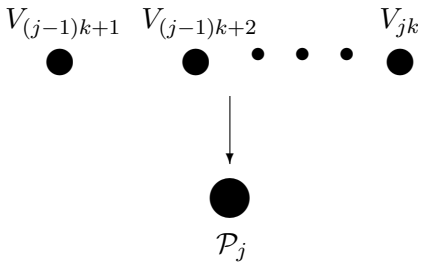
do $i = 0$ *έως* p

V_{2i} *υπολογίζει* $\boldsymbol{t}_{2p+2i}$, $\boldsymbol{t}_{2p+2i-1}$

enddo

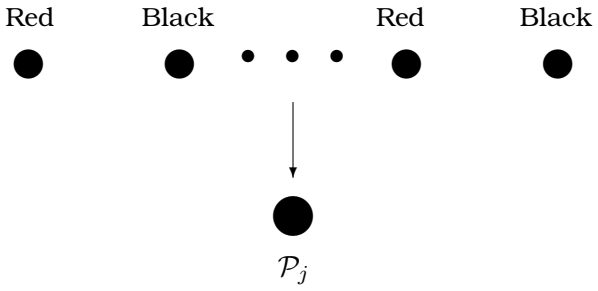
!$OMP END PARALLEL DO

Για την εκτέλεση του αλγορίθμου σε ένα παράλληλο υπολογιστικό σύστημα, που αποτελείται από P υπολογιστικούς πυρήνες, ομάδες των threads πρέπει να αντιστοιχηθούν σε πυρήνες της GPU και σε αυτούς του κεντρικού επεξεργαστή. Η ενότητα που ακολουθεί περιγράφει τη διαδικασία αντιστοίχισης για τη περίπτωση όπου $n_s = kP$, καθώς για τις υπόλοιπες περιπτώσεις υπάρχει όμοια αντιμετώπιση. Στη συγκεκριμένη περίπτωση, ωστόσο, το υπολογιστικό κόστος είναι το ίδιο για όλους τους πυρήνες υπολογισμών, οπότε υπάρχει ισοκατανομή του υπολογιστικού φόρτου.
Σύμφωνα με τον τρόπο αντιστοίχισης που αναφέρθηκε παραπάνω, κάθε k διαδοχικά threads αντιστοιχίζονται σε κάθε έναν από τους $\mathcal{P}_j \ , \ (j = 1, \ldots, P)$ πυρήνες, με βάση την αρχιτεκτονική του συστήματος.



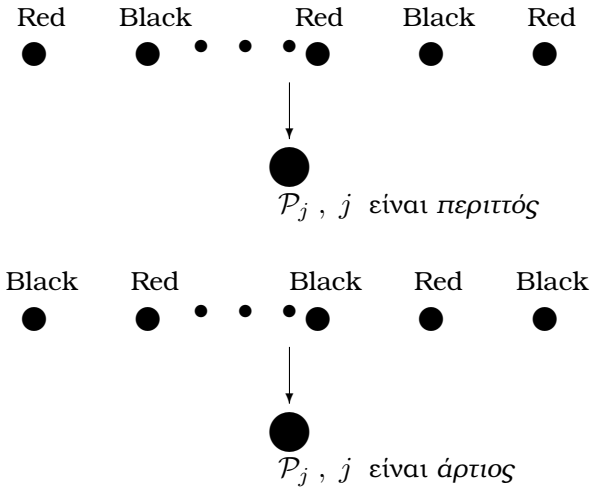
Οπότε, ακολουθώντας την παραπάνω διαδικασία σε κάθε $\mathcal{P}_j$ πυρήνα μπορούμε να αντιστοιχίσουμε k threads $V_{(j-1)k+1}, \ldots, V_{jk}$. Παρατηρούμε ότι:

- Όταν ο δείκτης k είναι *άρτιος* οι δείκτες $(j-1)k+1$ και jk που ικανοποιούν τη συνθήκη $(j-1)k+1$ είναι *περιττοί,* ενώ οι jk είναι *άρτιοι.* Οπότε τα υπολογιστικά threads $V_{(j-1)k+1}$ και V_{jk} είναι αντίστοιχα ***red (περιττά)*** και ***black (άρτια)*** threads κι έτσι σχηματικά θα ισχύει ότι:



Οπότε, στους πυρήνες $\mathcal{P}_j$ αντιστοιχίζονται τα k ***red*** διανύσματα $\boldsymbol{t}_l$, $l = (j-1)k, \ldots, jk-1$ και τα k ***black*** διανύσματα $\boldsymbol{t}_{2p+l}$, $l = (j-1)k+1, \ldots, jk$.

- Όταν ο δείκτης k είναι *περιττός* οι δείκτες $(j-1)k+1$ και jk που ικανοποιούν τη συνθήκη $(j-1)k+1$ και jk είναι *περιττοί* όταν j είναι *περιττός*, ενώ οι $(j-1)k+1$ και jk είναι *άρτιοι* όταν το j είναι *άρτιος*. Έτσι τα threads $V_{(j-1)k+1}$ και V_{jk} είναι μαζί ***red*** *(περιττά)* όταν το j είναι *περιττός* ενώ και τα δυο είναι ***black*** *(άρτια)* όταν το j είναι *άρτιος* κι έτσι θα έχουμε το σχήμα:



Οπότε, όταν το j είναι *περιττός*, στους πυρήνες $\mathcal{P}_j$ αντιστοιχίζονται τα $k+1$ ***red*** διανύσματα $\boldsymbol{t}_l$, $l = (j-1)k, \ldots, jk$ και τα $k-1$ ***black*** διανύσματα $\boldsymbol{t}_{2p+l}$, $l = (j-1)k+1, \ldots, jk-1$, ενώ, όταν το j είναι *άρτιος*, στους πυρήνες $\mathcal{P}_j$ αντιστοιχίζονται τα $k-1$ ***red*** διανύσματα $\boldsymbol{t}_l$, $l = (j-1)k+1, \ldots, jk-1$ και τα $k-1$ ***black*** διανύσματα $\boldsymbol{t}_{2p+l}$, $l = (j-1)k, \ldots, jk$.

Η υλοποίηση των παραπάνω αλγορίθμων με CPU threads μπορεί να πραγματοποιηθεί με χρήση αποδοτικών διαδικασιών από υπάρχουσες αριθμητικές βιβλιοθήκες. Για παράδειγμα, στην περίπτωση της εμπρός και πίσω αντικατάστασης κατά τη διάρκεια της block άμεσης επίλυσης του γραμμικού συστήματος με τους πίνακες D_R και D_B, μπορεί να επιλεγεί η κατάλληλη διαδικασία από την βιβλιοθήκη Lapack [29], ενώ για τον πολλαπλασιασμό πίνακα με διάνυσμα, που εμπλέκει τους πίνακες H_B και H_R, μπορεί να

χρησιμοποιηθεί η κατάλληλη διαδικασία από τη βιβλιοθήκη BLAS, [28]. Ωστόσο, για την υλοποίηση με GPU threads κρίνεται αναγκαία η σχεδίαση κατάλληλων αλγορίθμων, έτσι ώστε να είναι εφικτή η εκμετάλλευση της ιδιαίτερης δομής των πινάκων που εμπλέκονται στους υπολογισμούς. Η σειριακότητα των υπολογισμών κατά τη διάρκεια της εμπρός και πίσω αντικατάστασης στη φάση επίλυσης του γραμμικού συστήματος, σε συνδυασμό με τους περιορισμούς που υπάρχουν λόγω μεγέθους μνήμης των GPUs, άμεσα μας υποδεικνύει ότι οι επιλύσεις κατά τους red και black κύκλους υπολογισμών, με τους πίνακες συντελεστών D_R και D_B αντίστοιχα, πρέπει να εκτελεστούν από CPU threads. Από την άλλη μεριά η ανεξαρτησία των δεδομένων που εμπλέκονται στις βασικές πράξεις μεταξύ πινάκων και διανυσμάτων, σε συνδυασμό με το γεγονός ότι οι εκατοντάδες υπολογιστικοί πυρήνες των GPUs είναι οργανωμένοι σε υπολογιστικές ομάδες και, μέσω της εκτέλεσης εφαρμογών κατά SIMD, μπορούν εκτελέσουν ταυτόχρονα βασικές αριθμητικές πράξεις ταχύτερα από ότι οι πυρήνες του κεντρικού επεξεργαστή του μηχανήματος. Κατά συνέπεια, κρίνεται κατάλληλη η εκτέλεση των διαδικασιών που εμπλέκουν πολλαπλασιασμό πίνακα-διάνυσμα κατά τη διάρκεια των red και black κύκλων, με τους block πίνακες H_R και H_B, στη GPU. Περαιτέρω βελτίωση της απόδοσης μπορεί να επιτευχθεί με την εκμετάλλευση της block δομής των H_R και H_B, από τις σχέσεις (3.29) και (3.30) αντίστοιχα, επειδή εμπλέκονται αποκλειστικά δύο πενταδιαγώνιοι πίνακες A_3 και A_4 τάξης $2n_s$. Ο αλγόριθμος της επαναληπτικής διαδικασίας θα έχει τη μορφή

Παράλληλος αλγόριθμος για επίλυση Schur collocation εξισώσεων

B1: *Επίλυση* $D_R\hat{b}_R = b_R$ *παράλληλα* CPU

B2: *Αποστολή πινάκων* A_3, A_4 *και διανύσματος* $\hat{b}_R$ CPU $\rightarrow$ GPU

B3: *Υπολογισμός* $\hat{b}_B = b_B - H_R\hat{b}_R$ *παράλληλα* GPU

B4: *Επίλυση με BiCGSTAB* $S\, x_B = \hat{b}_B$ *παράλληλα*

B5: *Υπολογισμός* $\hat{x}_B = H_B x_B$ *παράλληλα* GPU

B6: *Αποστολή διανύσματος* $\hat{x}_B$ $\boxed{\text{GPU}} \rightarrow \boxed{\text{GPU}}$

B7: *Επίλυση* $D_R\hat{x}_R = \hat{x}_B$ *παράλληλα* $\boxed{\text{CPU}}$

B8: *Υπολογισμός* $x_R = \hat{b}_R - \hat{x}_R$ *παράλληλα* $\boxed{\text{CPU}}$

Οι υπολογισμοί του βήματος B4 της μεθόδου BiCGSTAB πραγματοποιούνται από τους επεξεργαστές του υπολογιστικού συστήματος, εκτός από τους υπολογισμούς των δυο πολλαπλασιασμών πινάκων-διανυσμάτων σε κάθε επαναληπτικό βήμα όπου εμπλέκεται ο Schur πίνακας του συμπληρώματος Schur S. Πιο αναλυτικά, οι υπολογισμοί του βήματος B4, που εμπλέκουν τους πίνακες H_R και H_B ως πολλαπλασιαστές, θα διεξάγονται στη GPU σύμφωνα με τον ακόλουθο αλγόριθμο:

Υπολογισμός του $t = Sp$

BS1: *Αποστολή* p $\boxed{\text{CPU}} \rightarrow \boxed{\text{GPU}}$

BS2: *Υπολογισμός* $t = H_B\, p$ *παράλληλα* $\boxed{\text{GPU}}$

BS3: *Αποστολή* t $\boxed{\text{GPU}} \rightarrow \boxed{\text{CPU}}$

BS4: *Επίλυση* $D_R s = t$ *παράλληλα* $\boxed{\text{CPU}}$

BS5: *Αποστολή* s $\boxed{\text{CPU}} \rightarrow \boxed{\text{GPU}}$

BS6: *Υπολογισμός* $q = H_R\, s$ *παράλληλα* $\boxed{\text{GPU}}$

BS7: *Αποστολή* q $\boxed{\text{GPU}} \rightarrow \boxed{\text{CPU}}$

BS8: *Υπολογισμός* $t = D_B\, p - q$ *παράλληλα* $\boxed{\text{CPU}}$

Το κόστος επικοινωνίας για τη μεταφορά δεδομένων μεταξύ της κύριας μνήμης του υπολογιστικού συστήματος και του γραφικού υποσυστήματος είναι αυτό της μεταφοράς δυο διανυσμάτων μεγέθους $2n_s^2$ ανά κατεύθυνση. Δηλαδή, το κόστος επικοινωνίας κάθε επαναληπτικού βήματος της BiCGSTAB είναι το κόστος μεταφοράς οκτώ διανυσμάτων μεγέθους $2n_s^2$ προς κάθε κατευθυνση, αφού οι πίνακες A_3 και A_4 θα μεταφερθούν και

θα αποθηκευτούν στη μνήμη της GPU μία φορά μόνο κατά την έναρξη της διαδικασίας επίλυσης.

Ο παρακάτω αλγόριθμος περιγράφει ένα black κύκλο υπολογισμών της $\boldsymbol{t} = H_B\boldsymbol{z}$ διαδικασίας πολλαπλασιασμού πίνακα-διανύσματος στη GPU.

Διαδικασία Υπολογισμού t=$H_B z$ στη GPU

!$ACC DATA COPYIN($\boldsymbol{z}$) CREATE($\boldsymbol{temp}$) COPYOUT($\boldsymbol{t}$)
!$ACC KERNELS

!$ACC LOOP
do $i = 1$ *έως* $2n_s$
$t(i) = A_3(3,i)z(i)$
enddo

!$ACC LOOP
do $i = 2$ *έως* $2n_s$
$t(i-1) = t(i-1) + A_3(2,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 1$
$t(i+1) = t(i+1) + A_3(4,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 2$
$t(i) = t(i) + A_3(1,i+2)z(i+2)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 2$
$t(i+2) = t(i+2) + A_3(5,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s$
$t(i) = t(i) - A_4(3,i)z(i+2n_s)$
enddo

!$ACC LOOP

$\underline{do} \quad i = 2 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$
$t(i-1) = t(i-1) - A_4(2,i)z(i+2n_s)$
enddo

!$ACC LOOP
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 1$
$t(i+1) = t(i+1) - A_4(4,i)z(i+2n_s)$
enddo

!$ACC LOOP
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 2$
$t(i) = t(i) - A_4(1,i+2)z(i+2+2n_s)$
enddo

!$ACC LOOP
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 2$
$t(i+2) = t(i+2) - A_4(5,i)z(i+2n_s)$
enddo

$\underline{do} \quad k = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad n_s - 3 \quad \underline{\mu\varepsilon\,\beta\acute{\eta}\mu\alpha} \quad 2$
$k_1 = (k-1)2n_s \;\;,\;\; k_2 = k2n_s \;\;,\;\; k_3 = (k+1)2n_s \;\;,\;\; k_4 = (k+2)2n_s$

!$ACC LOOP INDEPENDENT
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$
$t(k_2+i) = A_3(3,i)z(k_1+i) + A_4(3,i)z(k_2+i)$
enddo

!$ACC LOOP INDEPENDENT
$\underline{do} \quad i = 2 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$
$t(k_2+i-1) = t(k_2+i-1) + A_3(2,i)z(k_1+i-1) + A_4(2,i)z(k_2+i)$
enddo

!$ACC LOOP INDEPENDENT
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 1$
$t(k_2+i+1) = t(k_2+i+1) + A_3(4,i)z(k_1+i) + A_4(4,i)z(k_2+i)$
enddo

!$ACC LOOP INDEPENDENT
$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 2$
$t(k_2+i) = t(k_2+i) + A_3(1,i+2)z(k_1+i+2) + A_4(1,i+2)z(k_2+i+2)$
enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_2 + i + 2) = t(k_2 + i + 2) + A_3(5, i)z(k_1 + i) + A_4(5, i)z(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_3 + i) = A_3(3, i)z(k_3 + i) - A_4(3, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 2$ *έως* $2n_s$

$t(k_3 + i - 1) = t(k_3 + i - 1) + A_3(2, i)z(k_3 + i - 1) - A_4(2, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 1$

$t(k_3 + i + 1) = t(k_3 + i + 1) + A_3(4, i)z(k_3 + i) - A_4(4, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_3 + i) = t(k_3 + i) + A_3(1, i + 2)z(k_3 + i + 2) - A_4(1, i + 2)z(k_4 + i + 2)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_3 + i + 2) = t(k_3 + i + 2) + A_3(5, i)z(k_3 + i) - A_4(5, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$temp(i) = t(k_3 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_3 + i) = t(k_3 + i) - t(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$

$t(k_2 + i) = t(k_2 + i) + temp(k_3 + i)$

$\underline{enddo}$

$\underline{enddo}$

$k_2 = 2n^2{}_s - 2n_s \;\; , \;\; k_3 = 2n^2{}_s - 2 \cdot 2n_s$

!$ACC LOOP

$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$

$t(k3 + i) = A3(3, i)z(k2 + i) + A4(3, i)z(k3 + i)$

$\underline{enddo}$

!$ACC LOOP

$\underline{do} \quad i = 2 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s$

$t(k3 + i - 1) = t(k3 + i - 1) + A3(2, i)z(k2 + i) + A4(2, i)z(k3 + i)$

$\underline{enddo}$

!$ACC LOOP

$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 1$

$t(k3 + i + 1) = t(k3 + i + 1) + A3(4, i)z(k2 + i) + A4(4, i)z(k3 + i)$

$\underline{enddo}$

!$ACC LOOP

$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 2$

$t(k3 + i) = t(k3 + i) + A3(1, i + 2)z(k2 + i + 2) + A4(1, i + 2)z(k3 + i + 2)$

$\underline{enddo}$

!$ACC LOOP

$\underline{do} \quad i = 1 \quad \underline{\acute{\epsilon}\omega\varsigma} \quad 2n_s - 2$

$t(k3 + i + 2) = t(k3 + i + 2) + A3(5, i)z(k2 + i) + A4(5, i)z(k3 + i)$

$\underline{enddo}$

!$ACC END KERNELS

!$ACC END DATA

Η πραγματοποίηση του αντίστοιχου red κύκλου υπολογισμών για GPU μπορεί να περιγραφεί με το παρακάτω αλγόριθμο.

Διαδικασία Υπολογισμού t=$H_R z$ στη GPU

!$ACC DATA COPYIN($z$) CREATE($temp$) COPYOUT($t$)
!$ACC KERNELS

!$ACC LOOP
do $i = 1$ *έως* $2n_s$
$t(i) = A_4(3,i)z(i)$
enddo

!$ACC LOOP
do $i = 2$ *έως* $2n_s$
$t(i-1) = t(i-1) + A_4(2,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 1$
$t(i+1) = t(i+1) + A_4(4,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 2$
$t(i) = t(i) + A_4(4,i+2)z(i+2)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s - 2$
$t(i+2) = t(i+2) + A_4(5,i)z(i)$
enddo

!$ACC LOOP
do $i = 1$ *έως* $2n_s$
$t(i+2n_s) = A_3(3,i)z(i+2\cdot 2n_s) - A_4(3,i)z(i+2\cdot 2n_s)$
enddo

!$ACC LOOP
do $i = 2$ *έως* $2n_s$
$t(i-1+2n_s) = t(i-1+2n_s) + A_3(2,i)z(i+2n_s) - A_4(2,i)z(i+2\cdot 2n_s)$
enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 1$

$t(i + 1 + 2n_s) = t(i + 1 + 2n_s) + A_3(4, i)z(i + 2n_s) - A_4(4, i)z(i + 2 \cdot 2n_s)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(i + 2n_s) = t(i + 2n_s) + A_3(1, i + 2)z(i + 2 + 2n_s) - A_4(1, i + 2)z(i + 2 + 2 \cdot 2n_s)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(i + 2 + 2n_s) = t(i + 2 + 2n_s) + A_3(5, i)z(i + 2n_s) - A_4(5, i)z(i + 2 \cdot 2n_s)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$temp(i) = t(i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(i) = t(2n_s + i) + t(i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(2n_s + i) = t(2n_s + i) - temp(i)$

enddo

do $k = 1$ *έως* $n_s - 3$ *με βήμα* 2

$k_1 = (k - 1)2n_s \ , \ k_2 = k2n_s \ , \ k_3 = (k + 1)2n_s \ , \ k_4 = (k + 2)2n_s$

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_2 + i) = A_3(3, i)z(k_1 + i) + A_4(3, i)z(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 2$ *έως* $2n_s$

$t(k_2 + i - 1) = t(k_2 + i - 1) + A_3(2, i)z(k_1 + i - 1) + A_4(2, i)z(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 1$

$t(k_2 + i + 1) = t(k_2 + i + 1) + A_3(4, i)z(k_1 + i) + A_4(4, i)z(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_2 + i) = t(k_2 + i) + A_3(1, i + 2)z(k_1 + i + 2) + A_4(1, i + 2)z(k_2 + i + 2)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_2 + i + 2) = t(k_2 + i + 2) + A_3(5, i)z(k_1 + i) + A_4(5, i)z(k_2 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_3 + i) = A_3(3, i)z(k_3 + i) - A_4(3, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 2$ *έως* $2n_s$

$t(k_3 + i - 1) = t(k_3 + i - 1) + A_3(2, i)z(k_3 + i - 1) - A_4(2, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 1$

$t(k_3 + i + 1) = t(k_3 + i + 1) + A_3(4, i)z(k_3 + i) - A_4(4, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_3 + i) = t(k_3 + i) + A_3(1, i + 2)z(k_3 + i + 2) - A_4(1, i + 2)z(k_4 + i + 2)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s - 2$

$t(k_3 + i + 2) = t(k_3 + i + 2) + A_3(5, i)z(k_3 + i) - A_4(5, i)z(k_4 + i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$temp(k2+i) = t(k_2+i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_2+i) = t(k_2+i) + t(k_3+i)$

enddo

!$ACC LOOP INDEPENDENT

do $i = 1$ *έως* $2n_s$

$t(k_3+i) = t(k_3+i) - temp(k_3+i)$

enddo

enddo

$k_1 = 2n^2{}_s - 2n_s \;\;,\;\; k_2 = 2n^2{}_s - 2\cdot 2n_s \;\;,\;\; k_3 = 2n^2{}_s - 3\cdot 2n_s$

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(k1+i) = -A_4(3,i)z(k1+i)$

enddo

!$ACC LOOP

do $i = 2$ *έως* $2n_s$

$t(k1+i-1) = t(k1+i-1) - A_4(2,i)z(k1+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 1$

$t(k1+i+1) = t(k1+i+1) - A_4(4,i)z(k1+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(k1+i) = t(k1+i) - A_4(4,i+2)z(k1+i+2)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(k1+i+2) = t(k1+i+2) - A_4(5,i)z(k1+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$temp(i) = t(k1+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(k2+i) = A_3(3,i)z(k3+i) + A_4(3,i)z(k2+i)$

enddo

!$ACC LOOP

do $i = 2$ *έως* $2n_s$

$t(k2+i-1) = t(k2+i-1) + A_3(2,i)z(k3+i) + A_4(2,i)z(k2+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 1$

$t(k2+i+1) = t(k2+i+1) + A_3(4,i)z(k3+i) + A_4(4,i)z(k2+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(k2+i) = t(k2+i) + A_3(1,i+2)z(k3+i+2) + A_4(1,i+2)z(k2+i+2)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s - 2$

$t(k2+i+2) = t(k2+i+2) + A_3(5,i)z(k3+i) + A_4(5,i)z(k2+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(k1+i) = t(k1+i) - t(k2+i)$

enddo

!$ACC LOOP

do $i = 1$ *έως* $2n_s$

$t(k2+i) = temp(i) + t(k2+i)$

enddo

!$ACC END KERNELS

!$ACC END DATA

4.3 Αλγόριθμος για αρχιτεκτονικές πολλαπλών υπολογιστικών πυρήνων με πολλαπλές GPUs

Η εξέλιξη της τεχνολογίας των υπολογιστικών συστημάτων και των γραφικών υποσυστημάτων τους έχει επιτρέψει σήμερα την ύπαρξη μηχανημάτων με πολλαπλές GPUs. Αν και αρχικά όλες οι GPUs αποτελούσαν ένα ανεξάρτητο τμήμα του υλικού κατασκευής του υπολογιστικού συστήματος, το οποίο ήταν συνδεδεμένο σε ένα δίαυλο PCI Express, σήμερα η ύπαρξη πολλαπλών τέτοιων διαύλων έχει επιτρέψει τη διασύνδεση πολλαπλών GPUs. Ο μόνος περιορισμός είναι ότι το υπολογιστικό θα διαθέτει τουλάχιστον ίσο αριθμό CPU πυρήνων με τον αριθμό των GPUs. Αυτό συμβαίνει διότι κατά την εκτέλεση εφαρμογών με χρήση των υπολογιστικών πυρήνων των GPU απαιτείται μια CPU διαδικασία ελέγχου-διαχείρισης για τη κάθε κάρτα γραφικών. Επίσης, η μνήμη της κάθε GPU δεν είναι άμεσα προσβάσιμη από μια άλλη, οπότε το μοντέλο ανάπτυξης εφαρμογών σε ένα τέτοιου τύπου υπολογιστικό περιβάλλον είναι κοινής και ταυτόχρονα κατανεμημένης μνήμης.

Για την επίλυση του γραμμικού συστήματος της μεθόδου Collocation σε ένα παράλληλο περιβάλλον υπολογιμών πολλαπλών υπολογιστικών πυρήνων CPU με πολλαπλές GPUs χρειάζεται η κατάλληλη τροποποίηση και επέκταση του αλγορίθμου της προηγούμενης ενότητας. Ειδικότερα, κατά τις διαδικασίες πολλαπλασιασμού διανυσμάτων με τους πίνακες H_R και H_B, οι υπολογισμοί των οποίων διεξάγονται από τους υπολογιστικούς πυρήνες των GPUs, χρειάζεται να γίνει αρχικά μια ομοιόμορφη διαμέριση των διανυσμάτων που είναι αποθηκευμένα στη κύρια μνήμη του υπολογιστικού συστήματος. Στη συνέχεια κάθε CPU thread το οποίο διαχειρίζεται την ανάλογη GPU θα αναλάβει να στείλει το κατάλληλο τμήμα των διανυσμάτων στη μνήμη της GPU. Όμως η δομή των πινάκων H_R και H_B είναι τέτοια όπου απαιτούνται δεδομένα από τα γειτονικά τμήματα του διανύσματος που θα πολλαπλασιαστεί με κάθε πίνακα, τα οποία θα βρίσκονται στη κύρια μνήμη κάποιας γειτονικής GPU. Οπότε για την ολοκλήρωση της διαδικασίας διεξαγωγής των πράξεων του πολλαπλασιασμού των πινάκων αυτών με τα διανύσματα

χρειάζεται κάθε φορά η δημιουργία πολλαπλών αντιγράφων των δεδομένων εξάρτησης των διαδικασιών αυτών στη κύρια μνήμη του υπολογιστικού συστήματος και η αποστολή τους στη συνέχεια στη κύρια μνήμη της κατάλληλης GPU. Αυτό όμως έχει ως συνέπεια την αύξηση του μεγέθους μεταφοράς των δεδομένων μεταξύ CPU και GPU με μια σειριακή διαδικασία, αφού η πρόσβαση στα δεδομένα της κύριας μνήμης του υπολογιστή για τη δημιουργία των πολλαπλών αντιγράφων επιτρέπεται μόνο από ένα CPU thread ανά χρονική στιγμή.

Μια άλλη αποδοτικότερη λύση σε αυτό το πρόβλημα είναι η διεξαγωγή μόνο των πράξεων στις GPUs που δεν έχουν εξαρτώμενα δεδομένα. Στη συνέχεια θα διεξάγονται οι πράξεις που διαθέτουν τα εξαρτημένα δεδομένα από τα CPU threads, για την ολοκλήρωση των διαδικασιών πολλαπλασιασμών των πινάκων H_R και H_B με διανύσματα. Η κατάλληλη οργάνωση των υπολογισμών με τα εξαρτώμενα δεδομένα απο τα CPU threads μπορεί να αποτρέψει τη ταυτόχρονη πρόσβαση αυτών των threads στα ίδια δεδομένα.

Ειδικότερα αν θεωρήσουμε ότι το πλήθος των GPUs είναι άρτιος αριθμός $nGPU$ τότε θα χρειαστεί το ίδιο πλήθος CPU threads για τη διαχείριση τους. Επειδή το συνολικό μέγεθος των διανυσμάτων που πολλαπλασιάζονται με τους πίνακες H_R και H_B είναι $2n_s^2$, η διαμέριση τους για την αποστολή τους σε κάθε GPU θα έχει μέγεθος $\frac{2n_s^2}{nGPU}$. Οπότε κάθε GPU θα υπολογίζει το τμήμα κάθε διανύσματος που αντιστοιχεί στις θέσεις από $2n_s^2 iGPU + 1$ μέχρι $2n_s^2(iGPU + 1)$ για $iGPU = 0, \ldots, nGPU - 1$. Η δημιουργία του συνολικού διανύσματος κάθε πολλαπλασιασμού θα ολοκληρώνεται με τη διεξαγωγή των πράξεων στις GPUs, στις οποίες εμπλέκονται τα εξαρτημένα δεδομένα σε κάθε περίπτωση. Για τη περίπτωση του πολλαπλασιασμού $t = H_R z$ θα χρειαστεί να γίνουν δυο πολλαπλασιασμοί με τους βασικούς πίνακες A_3 και A_4 με τα τμήματα των διανυσμάτων $z(2n_s^2 iGPU + 1, \ldots, 2n_s^2 iGPU + 2n_s)$ και $z(2n_s^2 iGPU + 2n_s(n_s - 1), \ldots, 2n_s^2(iGPU + 1))$ για τον υπολογισμό των τμημάτων των διανυσμάτων $t(2n_s^2 iGPU + 1, \ldots, 2n_s^2 iGPU + 4n_s)$ και $t(2n_s^2 iGPU + 2n_s(n_s - 2), \ldots, 2n_s^2(iGPU + 1))$. Για το πολλαπλασιασμό $t = H_B z$ θα χρειαστεί να γίνουν δυο πολλαπλασιασμοί με τους ίδιους πίνακες A_3 και A_4 με τα τμή-

ματα του διανύσματος $z(2n_s^2 iGPU + 1, \ldots, 2n_s^2 iGPU + 4n_s)$ και $z(2n_s^2 iGPU + 2n_s(n_s - 2), \ldots, 2n_s^2(iGPU + 1))$ για τη κατασκευή των τμημάτων $t(2n_s^2 iGPU + 1, \ldots, 2n_s^2 iGPU + 2n_s)$ και $t(2n_s^2 iGPU + 2n_s(n_s - 1), \ldots, 2n_s^2(iGPU + 1))$ από κάθε CPU thread.

Ο παρακάτω αλγόριθμος περιγράφει τη διαδικασία του πολλαπλασιασμού $t = Sp$ για τη περίπτωση ύπαρξης πολλαπλών γραφικών υποσυστημάτων.

Υπολογισμός του $t = Sp$

!$OMP PARALLEL DO

do $iGPU = 0$ *έως* $nGPU - 1$

S1: *Αποστολή* $p(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ CPU $\rightarrow$ GPU

S2: *Υπολογισμός* $t(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ *για* $t = H_B p$ *παράλληλα* GPU

S3: *Αποστολή* $t(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ GPU $\rightarrow$ CPU

S3A: *Υπολογισμός* $t(2n_s^2 iGPU + 1 : 2n_s^2 iGPU + 2n_s)$

και $t(2n_s^2 iGPU + 2n_s(n_s - 1) : 2n_s^2(iGPU + 1))$ CPU

enddo

!$OMP END PARALLEL DO

S4: *Επίλυση* $D_R s = t$ *παράλληλα* CPU

!$OMP PARALLEL DO

do $iGPU = 0$ *έως* $nGPU - 1$

S5: *Αποστολή* $s(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ CPU $\rightarrow$ GPU

S6: *Υπολογισμός* $q(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ *για* $q = H_R s$ *παράλληλα* GPU

S7: *Αποστολή* $q(2n_s^2 iGPU + 1 : 2n_s^2(iGPU + 1))$ GPU $\rightarrow$ CPU

S7A: *Υπολογισμός* $q(2n_s^2 iGPU + 1 : 2n_s^2 iGPU + 2n_s)$

και $q(2n_s^2 iGPU + 2n_s(n_s-2) : 2n_s^2(iGPU+1))$ CPU

enddo

!$OMP END PARALLEL DO

S8: *Υπολογισμός* $t = D_B\, p - q$ *παράλληλα* CPU

Κεφάλαιο 5

Μελέτη συμπεριφοράς υλοποίησης των αλγορίθμων

Οι υλοποιήσεις των αλγορίθμων πραγματοποιήθηκαν σε μηχάνημα τύπου HP SL390s G7, το οποίο ανήκει στο Εργαστήριο Εφαρμοσμένων Μαθηματικών και Ηλεκτρονικών Υπολογιστών του Πολυτεχνείου Κρήτης, καθώς και στον Υπολογιστή Πλέγματος του Πολυτεχνείου Κρήτης.

Το μηχάνημα HP SL390s G7 είναι ένα υπολογιστικό σύστημα αρχιτεκτονικής κοινής μνήμης, που αποτελείται από δύο 6-πύρηνους τύπου Xeon X5660@2.8GHz επεξεργαστές με 12 MB Level 3 μνήμης cache για το καθένα. Η συνολική μνήμη είναι 24 GB και το λειτουργικό σύστημα είναι Oracle Linux της έκδοσης 6.2. Επίσης, διαθέτει διπλό γραφικό υποσύστημα GPU τύπου Tesla M2070 αρχιτεκτονικής Fermi [31], συνδεδεμένο μέσω ξεχωριστών διαύλων PCI-gen2 για κάθε GPU, η οποία διαθέτει 6GB μνήμης και 448 πυρήνες σε 14 πολυεπεξεργαστές.

Σχήμα 5.1: GPU τύπου Tesla M2070 αρχιτεκτονικής Fermi από την εταιρεία NVIDIA.

Ο υπολογιστής πλέγματος αποτελείται από 43 υπολογιστικούς κόμβους κατανεμημένους σε 3 συστάδες. Κάθε υπολογιστικός κόμβος διαθέτει 2 επεξεργαστές AMD Opteron τύπου 2218@2.6GHz και μνήμη 4GB. Η συνολική μνήμη του μηχανήματος είναι 1152GB, ενώ το λειτουργικό σύστημα είναι Scientific Linux έκδοσης 5.x. Η ταχύτητα επικοινωνίας κάθε υπολογιστικού κόμβου είναι 10 Gbit μεταξύ κόμβων της ίδιας συστάδας και 1Gbit για οποιοδήποτε άλλο.

Οι εφαρμογές αναπτύχθηκαν με χρήση γλώσσας προγραμματισμού Fortran, με υπολογισμούς διπλής και απλής ακρίβειας, χρησιμοποιώντας τα πρότυπα OpenMP [34] και OpenACC [35]. Το σύστημα με τις GPUs διαθέτει τους μεταγλωτιστές της εταιρείας PGI στην έκδοση 12.9, ενώ ο υπολογιστής πλέγματος τους gnu έκδοση 4.4. Σε αυτό το μηχάνημα χρησιμοποιήθηκε το πρότυπο MPI [33] για την ανταλλαγή μηνυμάτων και δεδομένων στην υλοποίηση του OpenMPI 1.4. Για τις βασικές πράξεις γραμμικής άλγεβρας, χρησιμοποιούνται υλοποιήσεις διαδικασιών από τις επιστημονικές βιβλιοθήκες BLAS [28] και LAPACK [29] με τις κατάλληλες υλοποιήσεις τους για τις δυο παραπάνω πλατφόρμες υπολογισμών.

Στην υλοποίηση των παράλληλων αλγορίθμων που αναφέρονται στο προηγούμενο κεφάλαιο, επιλύεται το πρόβλημα δοκιμής Dirichlet Modified Helmholtz

$$\begin{cases} \nabla^2 u(x,y) - \lambda u(x,y) &= f(x,y) \ , \ (x,y) \in \Omega \\ u(x,y) &= g(x,y) \ , \ (x,y) \in \partial\Omega \end{cases}$$

όπου η παράμετρος $\lambda = 1$ στο ορθογώνιο χωρίο $\Omega \equiv (0,1) \times (0,1)$, το οποίο δέχεται την παρακάτω αναλυτική λύση

$$u(x,y) = 10 \ \phi(x) \ \phi(y) \ , \ \ \phi(x) = e^{-100(x-0.1)^2}(x^2 - x),$$

5.1 Υλοποίηση της μεθόδου Newton με χρήση διπλής και μικτής ακρίβειας υπολογισμών σε αρχιτεκτονικές κατανεμημένης μνήμης

Για τη μελέτη της συμπεριφοράς του παράλληλου αλγορίθμου της μεθόδου Newton χρησιμοποιήθηκε ο υπολογιστής πλέγματος. Αρχικά, πραγματοποιήθηκαν δοκιμές για την εύρεση των βέλτιστων τιμών μεταξύ των εξωτερικών και των εσωτερικών επαναλήψεων βημάτων της μεθόδου με χρήση διπλής και μικτής ακρίβειας στους υπολογισμούς για τη παραγωγή παραπλήσιας νόρμας σφάλματος του γραμμικού συστήματος σε κάθε πρόβλημα δοκιμής. Ο παρακάτω πίνακας παρουσιάζει τη συμπεριφορά της μεθόδου Newton για διακριτοποίησεις μεγέθους έως και 2048 πεπερασμένων στοιχείων ανά κατεύθυνση, με αναφορά στα απαιτούμενα βήματα σύγκλισης της επαναληπτικής διαδικασίας διόρθωσης υπολοίπου (εξωτερικές επαναλήψεις, **Outer-Newton**) καθώς και στα βήματα της μεθόδου BiCGSTAB (**Inner-BiCGSTAB**) που χρησιμοποιείται για την επίλυση της εξίσωσης υπολοίπου. Επίσης, αναφέρεται η Ευκλείδια νόρμα του σφάλματος του γραμμικού συστήματος για τη χρήση διπλής και μικτής ακρίβειας καθώς και ο συνολικός αριθμός των βαθμών ελευθερίας κάθε προβλήματος (DOF).

		Iterations		Double	Mixed
n_s	DOF	Inner-BiCGSTAB	Outer-Newton	$\|\|b - Ax^{(m)}\|\|_2$	$\|\|b - Ax^{(m)}\|\|_2$
128	262.144	50	3	5.38e-06	1.11e-05
256	1.048.576	70	6	3.01e-08	9.26e-09
512	4.194.304	40	26	1.21e-08	4.12e-09
1024	16.777.216	30	99	5.46e-09	5.35e-09
2048	67.108.864	40	280	3.35e-09	2.05e-09

Στη συνέχεια έγινε μελέτη της συμπεριφοράς του παράλληλου αλγορίθμου ως προς τον αριθμό των επεξεργαστών. Οι πίνακες που ακολουθούν παρουσιάζουν τις μετρήσεις του χρόνου εκτέλεσης της υλοποίησης της μεθόδου επαναληπτικής βελτίωσης υπολοίπου με τη μέθοδο Newton για τις παραπάνω διακριτοποιήσεις με χρήση 2, 4, 8, 16, 32, 64 και 128 υπολογιστικών πυρήνων. Αναφέρονται, ο χρόνος υπολογισμών (Computation Time) και ο χρόνος επικοινωνίας μεταξύ των πυρήνων (Communication Time) για κάθε δοκιμή, ο συνολικός χρόνος εκτέλεσης (Total Time) καθώς και η επιτάχυνση (speedup) που προέκυψε. Τα τέσσερα γραφήματα που ακολουθούν στη συνέχεια παρουσιάζουν τη συνολική διακύμανση της επιτάχυνσης και τη διακύμανση της απόδοσης ως προς το μέγεθος του προβλήματος και το πλήθος των επεξεργαστών για τη περίπτωση χρήσης διπλής και μικτής ακρίβειας υπολογισμών.

$n_s = 64$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	0.292	–	–	–	0.304	–
2	0.152	0.005	0.157	1.85	0.157	0.017	0.174	1.74
4	0.075	0.041	0.116	2.50	0.079	0.018	0.098	3.09
8	0.034	0.199	0.238	1.22	0.039	0.163	0.203	1.49
16	0.024	0.195	0.220	1.32	0.028	3.276	3.299	0.09

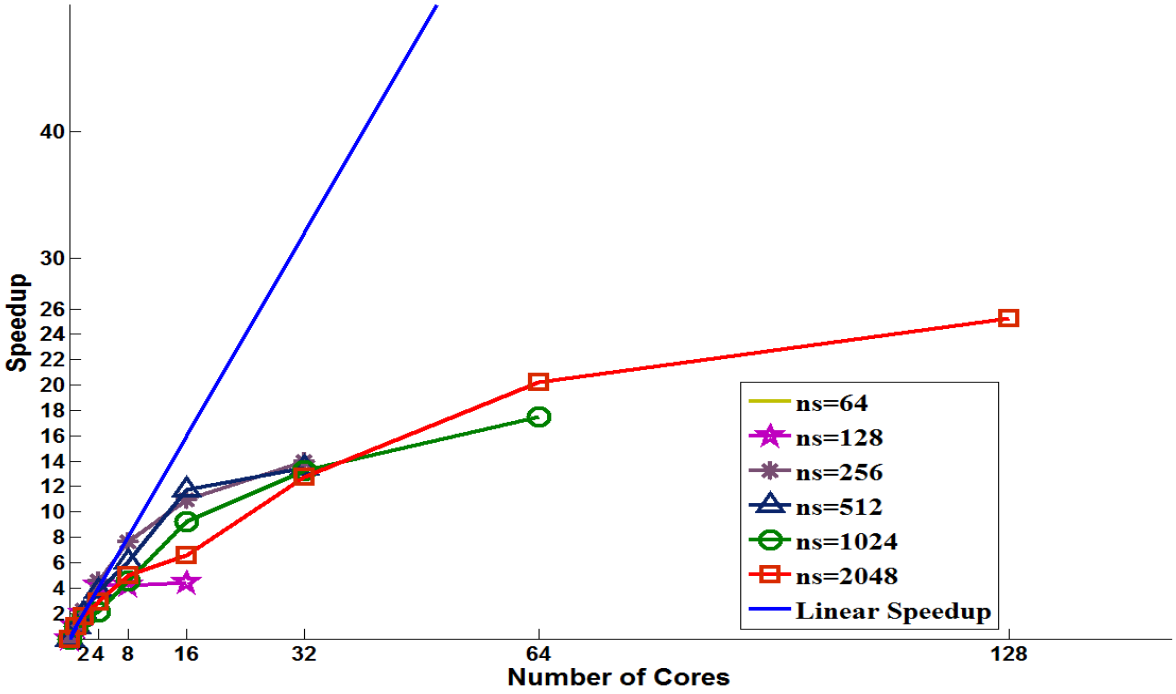
$n_s = 128$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	2.145	–	–	–	2.054	–
2	1.071	0.021	1.093	1,96	0.873	0.164	1.037	1,97
4	0.485	0.026	0.511	4,19	0.456	0.143	0.600	3,42
8	0.233	0.279	0.513	4,18	0.271	0.273	0.544	3,77
16	0.132	0.356	0.489	4,38	0.118	0.370	0.488	4,20
32	0.064	0.488	0.552	3,88	0.061	6.476	6.537	0,31

$n_s = 256$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	28.08	–	–	–	18.54	–
2	12.19	0.141	12.33	2,27	10.83	1.554	12.38	1,49
4	5.835	0.301	6.137	4,57	4.273	1.474	5.747	3,22
8	2.616	1.043	3.659	7,67	2.016	1.359	3.376	5,49
16	1.245	1.317	2.563	10,95	1.299	1.583	2.882	6,43
32	0.606	1.406	2.012	13,94	0.556	4.667	5.223	3,54
64	0.367	2.924	3.292	8,52	0.331	11.21	11.55	1,60

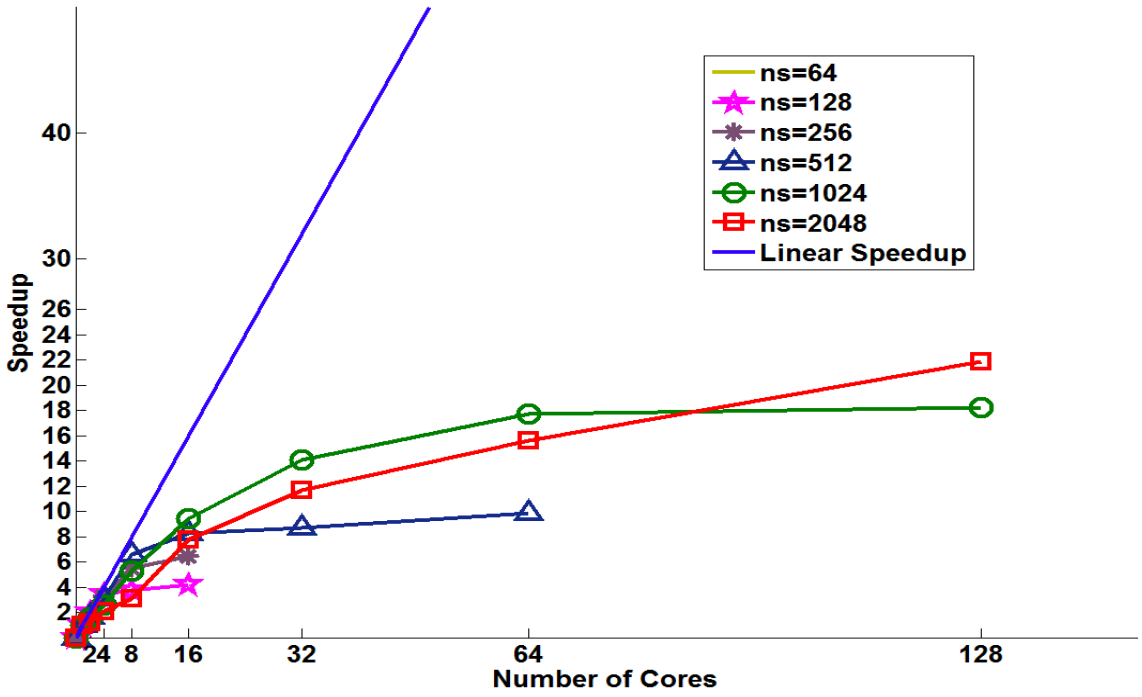
$n_s = 512$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	269.5	–	–	–	229.2	–
2	134.1	0.946	135.1	1,99	126.1	16.54	142.6	1,60
4	62.84	9.05	71.89	3,74	62.50	12.31	74.82	3,06
8	34.65	9.62	44.26	6,09	24.17	10.65	34.83	6,58
16	12.40	10.47	22.88	11,78	14.71	12.93	27.65	8,28
32	7.764	12.25	20.02	13,46	6.765	19.60	26.36	8,69
64	3.935	28.45	32.39	8,32	3.181	20.02	23.21	9,87
128	1.991	32.29	34.28	7,86	1.877	38.96	40.84	5,61

$n_s = 1024$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	2670	–	–	–	2482	–
2	1545	6.149	1551	1,72	1409	156.0	1565	1,58
4	1230	47.94	1277	2,08	828.7	129.1	957.8	2,59
8	509.2	74.86	584.1	4,57	366.7	106.9	473.7	5,24
16	230.7	58.37	289.1	9,23	174.8	88.69	263.5	9,41
32	123.7	77.51	201.2	13,26	88.97	86.87	175.8	14,11
64	49.10	103.8	152.9	17,46	44.66	95.21	139.8	17,74
128	24.14	126.6	150.7	17,71	20.63	115.5	136.1	18,22

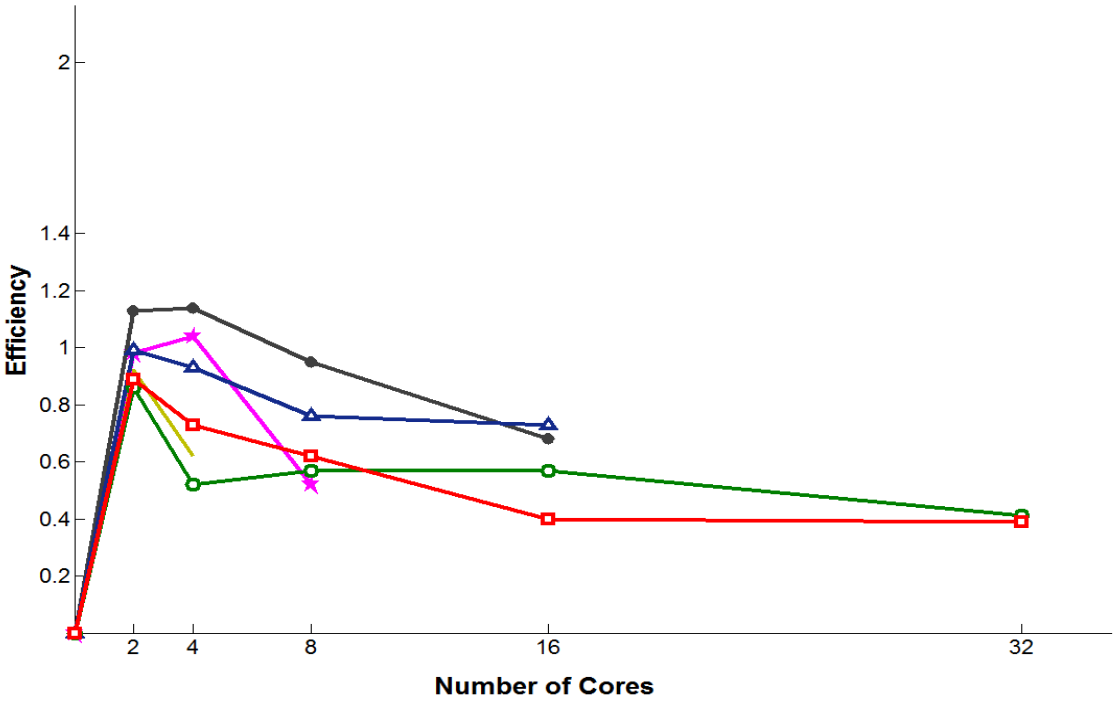
$n_s = 2048$	Time Measurements for Double Precision Computations				Time Measurements for Mixed Precision Computations			
#cores	Computation	Communication	Total	Speedup	Computation	Communication	Total	Speedup
1	–	–	33792	–	–	–	23134	–
2	18908	52	18961	1,78	17097	1968	19065	1,21
4	11331	227	11559	2,92	10228	1004	11232	2,05
8	6450	334	6785	4,98	6261	1173	7434	3,11
16	4511	649	5160	12,72	2185	796	2981	7,75
32	1998	656	2655	12,72	1297	685	1982	11,66
64	921	752	1673	20,18	726	755	1482	15,60
128	371	964	1336	25,28	279	779	1059	21,84



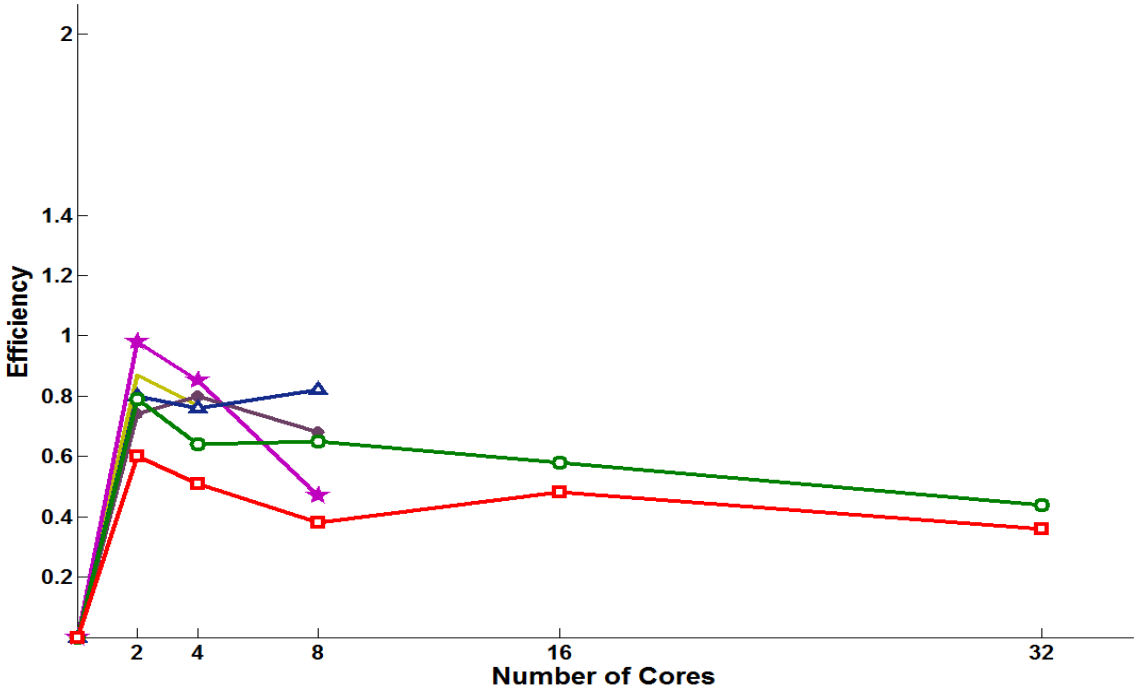
Σχήμα 5.2: Επιτάχυνση με χρήση διπλής ακρίβειας υπολογισμών



Σχήμα 5.3: Επιτάχυνση με χρήση μικτής ακρίβειας υπολογισμών



Σχήμα 5.4: Απόδοση με χρήση διπλής ακρίβειας υπολογισμών



Σχήμα 5.5: Απόδοση με χρήση μικτής ακρίβειας υπολογισμών

5.2 Υλοποίηση των μεθόδων Newton και Schur Complement σε αρχιτεκτονικές κοινής μνήμης με χρήση γραφικών υποσυστημάτων

Σε αυτή την ενότητα παρουσιάζεται η μελέτη της συμπεριφοράς παράλληλων αλγορίθμων για υπολογιστικές αρχιτεκτονικές κοινής μνήμης με γραφικά υποσυστήματα υπολογισμών. Κατά την υλοποίηση των αλγορίθμων έχουν ληφθεί υπόψη παράμετροι, οι οποίες σχετίζονται με την ανάπτυξη εφαρμογών για GPUs αρχιτεκτονικής τύπου Fermi και η ρύθμιση των οποίων μπορεί να διαμορφώσει την απόδοση του παράλληλου επιλύτη.

5.2.1 Ρύθμιση παραμέτρων στη GPU για αύξηση της παράλληλης απόδοσης

- Η κάρτα γραφικών Tesla M2070 υπολογιστικής δυνατότητας 2.0 δίνει δυνατότητα στον προγραμματιστή να επιλέξει το μέγεθος των μνημών L1 cache και shared cache, μεταξύ 16 και 48 KB ή 48 και 16 KB αντίστοιχα, ανάλογα τις ανάγκες κάθε kernel σε shared μνήμη ή σε ταχεία προσπέλαση της κύριας μνήμης μέσω της L1 cache.

- Ο προγραμματιστής μπορεί να ζητήσει τη μη χρησιμοποίηση της L1 cache μνήμης για την προσπέλαση της κύριας μνήμης.

- Ο προγραμματιστής μπορεί να ορίσει το μέγιστο αριθμό καταχωρητών που μπορεί να χρησιμοποιήσει ένα kernel. Με τη μείωση του αριθμού των καταχωρητών, που επιτυγχάνεται με τη μείωση των μεταβλητών μέσα στο kernel, αυξάνεται ο αριθμός των warps που εκτελούνται σε κάθε πολυεπεξεργαστή. Απαιτείται όμως ορθή διαχείριση μιας τέτοιας ρύθμισης, καθώς σε περίπτωση έλλειψης καταχωρητών, τα threads του block απευθύνονται στη τοπική (local) μνήμη, η πρόσβαση στην οποία κοστίζει χρονικά.

Στην επόμενη παράγραφο παρουσιάζεται ο υπολογισμός των threads σε κάθε block με βάση την ανάγκη του kernel σε καταχωρητές.

Υπολογισμός των threads ανά block: Στην ανάπτυξη εφαρμογών της παρούσας διατριβής, με χρήση του προτύπου OpenACC και της παραμετρικής εντολής `-Minfo=accel` στον μεταγλωττιστή της PGI `pgf90`, ver.12.9, ελήφθησαν υπόψιν οι ανάγκες των kernel σε καταχωρητές, με σκοπό να υπολογιστεί ο βέλτιστος αριθμός threads σε κάθε block. Υπενθυμίζεται ότι στο OpenACC ο μεταγλωττιστής επιλέγει τον αριθμό των threads που θεωρεί κατάλληλο για κάθε block στην εκτέλεση ενός kernel και ο προγραμματιστής μπορεί να το διαμορφώσει με τις κατάλληλες οδηγίες και παραμέτρους.

Υποθέτουμε ότι ένα kernel απαιτεί 17 καταχωρητές ανά thread. Αφού κάθε warp αποτελείται από 32 threads, για την εκτέλεση ενός warp απαιτούνται 544 καταχωρητές. Κάθε πολυεπεξεργαστής διαθέτει 32768 καταχωρητές, οπότε μπορεί να εκτελέσει περίπου 60 warps. Υπενθυμίζεται ότι ο μέγιστος αριθμός warps που μπορεί να χειριστεί ένας πολυπεξεργαστής είναι 48. Άρα, γίνεται αντιληπτό ότι για να είναι μέγιστη η απόδοση του πολυεπεξεργαστή πρέπει ο αριθμός των threads που θα εκτελούνται σε αυτόν να είναι πολλαπλάσιο του 48.

Δοκιμάστηκαν διάφορες τιμές για τον αριθμό των threads σε διαφορετικά kernels. Επιλέγοντας 256 threads σε κάθε block ζητείται η εκτέλεση 8 warps δηλαδή $48/8 = 6$ blocks σε κάθε πολυεπεξεργαστή. Το πλήθος των threads ανά πολυεπεξεργαστή είναι $256 * 6 = 1536$, ενώ χρησιμοποιούνται $256 * 6 * 17 = 26112$ καταχωρητές, δηλαδή το 94% του συνόλου. Επιλέγοντας 32 threads σε αντίστοιχο παράδειγμα, η πληρότητα (occupancy) μειώνεται μόλις στο 17%. Στις εκτελέσεις της διατριβής παρατηρήθηκε βελτίωση της απόδοσης του GPU κώδικα με την επιλογή μειωμένου αριθμού threads και μειωμένης πληρότητας στους καταχωρητές των πολυπεξεργαστών. Η χαμηλή πληρότητα (low occupancy) αύξησε τον αριθμό των καταχωρητών ανά thread με αποτέλεσμα σημαντικός όγκος δεδομένων κάθε thread να παραμένει στους καταχωρητές, αποφεύγοντας την

μεταφορά από και πρός άλλες μνήμες, για παράδειγμα τη διαρκή πρόσβαση στη κοινή (shared) μνήμη της GPU.

Συμπερασματικά, η επιλογή του αριθμού των threads ανά block που χρησιμοποιούνται για κάθε kernel διαμορφώνει ανάλογα τον αριθμό των καταχωρητών που χρησιμοποιούνται σε κάθε πολυεπεξεργαστή και επηρεάζει καθοριστικά την απόδοση τους. Συνεπώς, η επιλογή του προγραμματιστή θα πρέπει να γίνεται προσεκτικά για κάθε kernel και λαμβάνοντας υπόψη τους περιορισμούς του υλικού (υπενθυμίζεται ότι στις κάρτες με υπολογιστική δυνατότητα 2.x το ανώτατο όριο είναι 1024 threads ανά block και κάθε πολυεπεξεργαστής διαχειρίζεται μέχρι 8 blocks). Το πρότυπο OpenACC παρέχει τη δυνατότητα με χρήση των παραμέτρων gang και vector στην οδηγία acc kernels να διαμορφωθούν σε κάθε kernel ξεχωριστά ο αριθμός των threads και ο αριθμός των blocks, ενώ ο μεταγλωττιστής της PGI, έκδοση 12.9, παρέχει με την παραμετρική εντολή -Minfo=accel, πληροφορίες σχετικά με τον αριθμό των καταχωρητών και το μέγεθος της shared μνήμης που παραχωρούνται σε ένα block. Ο υπολογισμός των παραπάνω σε κάθε περίπτωση βασίζεται στα παρακάτω:

Ο συνολικός αριθμός καταχωρητών που παραχωρείται για ένα block σε μια κάρτα υπολογιστικής δυνατότητας 2.x είναι

$$R_{block} = \lceil W_{size}, R_k, W_{block} \rceil$$

όπου W_{size} ίσο με 32, δηλαδή ο αριθμός των threads σε κάθε warp, R_k ο αριθμός των καταχωρητών που χρησιμοποιούνται από το kernel και $W_{block} = \lceil T/W_{size}, 1 \rceil$, ενώ το μέγεθος της shared μνήμης για κάθε block, σε bytes είναι

$$\mathbf{Size_{block}} = \lceil S_k, G_s \rceil$$

όπου S_k το μέγεθος της shared μνήμης που χρησιμοποείται από το kernel και G_s ίσο με 128 (στις κάρτες 2.x η μνήμη αντιμετωπίζεται ως σύνολο από τμήματα μήκους 128 bytes), [63, 55].

5.2.2 Αποτελέσματα υλοποίησης των μεθόδων Newton και Schur Complement

Αυτή η ενότητα παρουσιάζει τα αποτελέσματα της μελέτης από την υλοποίηση των αλγορίθμων σε υπολογιστικά περιβάλλοντα κοινής μνήμης. Οι δοκιμές πραγματοποιήθηκαν στο μηχάνημα HP SL390. Αρχικά, μελετήθηκε η συμπεριφορά σε σειριακή μορφή των μεθόδων Newton με χρήση διπλής και μικτής ακρίβειας και της επαναληπτικής διαδικασίας του συμπληρώματος Schur. Οι παρακάτω δυο πίνακες παρουσιάζουν τη συμπεριφορά της κάθε μεθόδου για διακριτοποιήσεις μεγέθους έως και 2048 πεπερασμένων στοιχείων ανά κατεύθυνση, με αναφορά στα απαιτούμενα βήματα σύγκλισης, την Ευκλείδια νόρμα του σφάλματος του γραμμικού συστήματος και το χρόνο εκτέλεσης.

Newton

	Iterations		Double		Mixed	
n_s	Inner-BiCGSTAB	Outer-Newton	$\|\|b - Ax^{(m)}\|\|_2$	Time	$\|\|b - Ax^{(m)}\|\|_2$	Time
256	70	6	3.48e-09	18.12	3.49e-08	16.23
512	40	26	2.76e-10	172.4	3.05e-08	148.9
1024	30	99	4.24e-09	2033	1.28e-08	1736
2048	40	280	2.63e-10	28894	1.34e-10	26219

Schur

n_s	Iterations	$\|\|b - Ax^{(m)}\|\|_2$	Time
256	294	6.06e-11	12.24
512	589	2.85e-11	88.83
1024	1161	1.39e-11	750.35
2048	3726	9.59e-12	9176

Στη συνέχεια έγιναν δοκιμές των δύο παράλληλων αλγορίθμων για αρχιτεκτονικές υπολογισμών με GPUs των μεθόδων Newton με χρήση μικτής ακρίβειας υπολογισμών και της μεθόδου του συμπληρώματος Schur. Ο παρακάτω πίνακας παρουσιάζει τους χρόνους εκτέλεσης σε δευτερόλεπτα των εφαρμογών των μεθόδων με χρήση αποκλειστικά CPU διαδικασιών καθώς και με συνδυαστική χρήση CPU-GPU.

	Schur		Newton	
n_s	CPU	CPU+GPU	CPU	CPU+GPU
256	12.24	11.18	16.23	12.48
512	88.83	71.25	148.9	112.9
1024	750.3	549.8	1736	1265
2048	9176	6770	26219	18877

Στη μελέτη της συμπεριφοράς εκτέλεσης του παράλληλου αλγορίθμου της μεθόδου Schur για την υλοποίηση στο συγκεκριμένο υπολογιστικό περιβάλλον, έγινε συλλογή μετρήσεων των χρόνων εκτέλεσης για διάφορες παραμέτρους του προβλήματος, όπως η χρήση του γραφικού υποστήματος υπολογισμών και του πλήθους των CPU πυρήνων. Οι παρακάτω πίνακες εμφανίζουν αυτές τις μετρήσεις σε δευτερόλεπτα για διαφορετικού μεγέθους προβλήματα με διακριτοποιήσεις από $n_s = 256$ έως και $n_s = 2048$, με υλοποιήσεις για τη περίπτωση αποκλειστικής χρήσης CPU και για τη περίπτωση ταυτόχρονης χρήσης CPU+GPU threads.

$n_s = 256$	CPU	GPU + CPU
CPU cores	Total Time	Total Time
1	12.24	11.18
2	8.12	7.53
4	4.87	5.63

$n_s = 512$	CPU	GPU + CPU
CPU cores	Total Time	Total Time
1	88.83	71.25
2	52.94	46.59
4	34.25	32.63

$n_s = 1024$	CPU	GPU + CPU
CPU cores	Total Time	Total Time
1	750.35	549.82
2	448.76	352.95
4	283.14	250.64

$n_s = 2048$	CPU	GPU + CPU
CPU cores	Total Time	Total Time
1	9176	6770
2	5001	4387
4	2999	2839

Οι χρόνοι μεταφοράς δεδομένων μεταξύ μνήμης των CPU και GPU και αντίστροφα παρουσιάζονται στο παρακάτω πίνακα για όλα τα προβλήματα διακριτοποιήσεων και για κάθε διαθέσιμο αριθμό CPU υπολογιστικών πυρήνων.

n_s	GPU - CPU	Computation Time		
	Comm. Time	1 Core	2 Cores	4 Cores
256	1.52	9.66	6.01	4.11
512	6.81	64.44	39.78	25.82
1024	44.3	505.5	308.6	206.3
2048	541	6229	3846	2298

Πρέπει να αναφερθεί ότι ο χρόνος επικοινωνίας μεταξύ των CPU και GPU είναι ανεξάρτητος του αριθμού των πυρήνων της CPU, διότι στον αλγόριθμο η διαδικασία μεταφοράς των δεδομένων μεταξύ μνήμης CPU) και GPU πραγματοποιείται από ένα μόνο υπολογιστικό thread για λόγους ισοκατανομής του υπολογιστικού φορτίου. Έτσι αποφεύγονται φαινόμενα συνωστισμού των δεδομένων κατά τη κίνηση τους μεταξύ πολλαπλών CPU threads προς τη μνήμη του γραφικού υποσυστήματος διαμέσου του διαύλου PCI του μηχανήματος.

Ο παρακάτω πίνακας περιλαμβάνει τις μετρήσεις του χρόνου επίλυσης των προβλημάτων με βαθμούς ελευθερίας από 1048576 μέχρι 67108860 κάνοντας χρήση υπολογιστικών πυρήνων από τη CPU προσθέτοντας πυρήνες από μια GPU και στη συνέχεια και από τη δεύτερη διαθέσιμη GPU. Σε κάθε περίπτωση εμφανίζεται η επιτάχυνση της μεθόδου σε σχέση με την αποκλειστική χρήση CPU διαδικασιών, [57].

DOF	n_s	CPU time	CPU + GPU time	GPU speedup	CPU + 2GPUs time	2GPUs speedup
1.048.576	256	12.24	11.18	1.09	8.58	1.42
4.194.304	512	88.83	71.25	1.25	54.61	1.63
16.777.216	1024	750.35	549.82	1.37	399.42	1.88
67.108.864	2048	9176.02	6770.11	1.36	5209.01	1.76

Κεφάλαιο 6

Συμπεράσματα

Τα επιστημονικά αποτελέσματα που προέκυψαν από την ερευνητική διαδικασία που περιγράφηκε στη παρούσα διατριβή μπορούν να ομαδοποιηθούν σε δυο κατηγορίες. Στη πρώτη μπορούν να συμπεριληφθούν τα συμπεράσματα από τη μελέτη της συμπεριφοράς υλοποίησης των επαναληπτικών διαδικασιών για την επίλυση του Collocation γραμμικού συστήματος σε σειριακό επίπεδο. Συγκεκριμένα

- Η χρήση μικτής ακρίβειας υπολογισμών της επαναληπτικής βελτίωσης υπολοίπου Newton που παράγεται από τη χρήση της μεθόδου Schur Complement επιλύει ικανοποιητικά και προβλήματα μικρής διακριτοποίησης.
- Η χρήση της μεθόδου Schur-BiCGSTAB επιτυγχάνει ταχύτερη σύγκλιση έναντι της μεθόδου Newton σε CPU αρχιτεκτονικές με χρήση CPU υπολογιστικών πυρήνων.

Στην επόμενη κατηγορία συμπερασμάτων περιλαμβάνονται τα αποτελέσματα που προέκυψαν από την υλοποίηση των παράλληλων αλγορίθμων επίλυσης που κατασκευάστηκαν. Ειδικότερα

- Ο παράλληλος αλγόριθμος Schur-BiCGSTAB για αρχιτεκτονικές κατανεμημένης μνήμης είναι αποδοτικός και στην εφαρμογή του με τη μέθοδο Newton.
- Η χρήση της μεθόδου Schur-BiCGSTAB επιτυγχάνει ταχύτερη σύγκλιση έναντι της μεθόδου Newton σε CPU-GPU αρχιτεκτονικές.

- Η μέθοδος Schur Complement υλοποιήθηκε μέσω παράλληλου αλγορίθμου, όπου η χρήση πολλαπλών CPU threads στη περίπτωση μιας ή πολλαπλών GPUs δεν δημιουργεί συνωστισμό δεδομένων (bottleneck) κατά την επικοινωνία μεταξύ CPU-GPU.

- Η χρήση πολλαπλών CPU υπολογιστικών πυρήνων επιταχύνει τη διαδικασία επίλυσης ακόμα και με τη χρήση υπολογιστικών πυρήνων από μια GPU.

- Η χρήση GPU υπολογιστικών πυρήνων επιταχύνει τη διαδικασία επίλυσης. Για παράδειγμα η συμμετοχή υπολογιστικών πυρήνων στη διεξαγωγή των πράξεων από μια GPU επιτάχυνε τη διαδικασία κατά 1.3 φορές, ενώ από δύο GPUs κατά 1.8 φορές.

Παράρτημα Α΄

Κώδικας σε γλώσσα προγραμματισμού Fortran

Α΄.1 Επίλυση του Schur Complement με τη μέθοδο Newton σε πολυεπεξεργαστικό σύστημα κατανεμημένης μνήμης

Α΄.1.1 Χρήση διπλής ακρίβειας υπολογισμών

Κυρίως πρόγραμμα

```
      parameter (ns=64,np=1,inh=2*ns,inh2=4*ns,
     +           n=inh*inh,n2=n/2,ip=n2/np)
      implicit real*8 (a-h,o-z)
      include 'mpif.h'
      real*8 b(n),x(n),t(n2),xb(n2),bb(n2),
     +  a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),M1(7,inh),M2(7,inh),
     +  bm(inh),v1(inh),v2(inh),tmp1(2*inh),tmp2(2*inh),
     +  resid,temp(inh),w(ip),tt(ip),pi(ip),ui(ip),s(ip),
     +  bbpart(ip),brpart(ip),rpart(ip),tpart(ip),rhpart(ip),
     +  cr(ip),cb(ip),xhr(ip),xhb(ip),rnr(ip),rnb(ip),dsqrt,
     +  qr(ip),qb(ip),xeb1(ip),xeb2(ip),xer1(ip),xer2(ip),
     +  MPI_Wtime,tim1,tim2,tmc1,tmc2,tcomp,tcomm

      integer ipvt1(inh),ipvt2(inh),p,status(MPI_STATUS_SIZE)

      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD,p,ierr)
      call start(ns,a1,a2,a3,a4,M1,M2,ipvt1,ipvt2)
      maxstep=3
```

```
      if (myrank.eq.0) then
       tolex=1.0d-9
       toa=0.0d0

       print*,'-----------------------------------------------------'
       print*,'Error Correction Double Precision Iterative Refinenement'
       print*,'-----------------------------------------------------'
       print*,'   Ns    = ',ns
       print*,'   Procs    = ',np
       print*,'   Total External Steps    =',maxstep
       tcomp=0.0d0
       tcomm=0.0d0

       call makeb(ns,b,bm)
       call redblack(ns,b,x)
       call makeredblack(ns,x,b)

       tmc1=MPI_Wtime()
      endif

c------To 2o melos xwrizetai se np kommatia---------------------

        call MPI_Scatter(b,ip,MPI_DOUBLE_PRECISION,brpart,ip,
     +                   MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
        call MPI_Scatter(b(n2+1),ip,MPI_DOUBLE_PRECISION,bbpart,ip,
     +                   MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

      if (myrank.eq.0) then
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
       tim1=MPI_Wtime()
      endif

c-------Dhmiourgia rn kai xh se Red & Black----------------------

        call dcopy(ip,brpart,1,qr,1)
        call dcopy(ip,bbpart,1,qb,1)
        call dcopy(ip,brpart,1,xhr,1)
        call dcopy(ip,bbpart,1,xhb,1)
        call dcopy(ip,qr,1,rnr,1)
        call dcopy(ip,qb,1,rnb,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
        endif
```

```
c---------------------------------------
c       Ekkinhsh epanalhptikhs diadikasias
c---------------------------------------

        do k=1,maxstep

        if (myrank.eq.0) then
         tim1=MPI_Wtime()
        endif

        call matdred(ns,xhr,xer1,a1,a2,myrank,ip,np)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

c---------------------------------------
c             Epikoinwnia gia thn Matblack
c---------------------------------------

       if (mod(myrank,2).eq.1) then
  call MPI_SEND(xhb,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  111,MPI_COMM_WORLD,ierr)
           call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  11,MPI_COMM_WORLD,status,ierr)
       if (myrank.lt.np-2) then
           call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,
     +                  myrank+1,11,MPI_COMM_WORLD,status,ierr)
           call MPI_SEND(xhb(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
     +                  myrank+1,12,MPI_COMM_WORLD,ierr)
       endif
       else
  call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
     +                  111,MPI_COMM_WORLD,status,ierr)
           call MPI_SEND(xhb(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
     +                  myrank+1,11,MPI_COMM_WORLD,ierr)
        if (myrank.gt.0) then
    call MPI_SEND(xhb,inh2,MPI_DOUBLE_PRECISION,
     +                  myrank-1,11,MPI_COMM_WORLD,ierr)
           call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  12,MPI_COMM_WORLD,status,ierr)
       endif
       endif

        if (myrank.eq.0) then
```

```
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
         endif

          call matblack(xhb,xeb1,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
          call daxpy(ip,1.0d0,xeb1,1,xer1,1)

         if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
         endif


c----------------------------------------
c               Epikoinwnia gia thn Matred
c----------------------------------------

         if (mod(myrank,2).eq.1) then
   call MPI_SEND(xhr,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +                   111,MPI_COMM_WORLD,ierr)
            call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
     +                   MPI_COMM_WORLD,status,ierr)
         if (myrank.lt.np-2) then
            call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,
     +                   myrank+1,11,MPI_COMM_WORLD,status,ierr)
            call MPI_SEND(xhr(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
     +                   myrank+1,12,MPI_COMM_WORLD,ierr)
         endif
         else
   call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
     +                   111,MPI_COMM_WORLD,status,ierr)
            call MPI_SEND(xhr(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
     +                   myrank+1,11,MPI_COMM_WORLD,ierr)
         if (myrank.gt.0) then
     call MPI_SEND(xhr,inh,MPI_DOUBLE_PRECISION,
     +                   myrank-1,11,MPI_COMM_WORLD,ierr)
            call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +                   12,MPI_COMM_WORLD,status,ierr)
         endif
         endif

         if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
```

```
        endif

        call matred(xhr,xer2,ns,a3,a4,temp,myrank,ip,np,v1,v2)
        call matdblack(ns,xhb,xeb2,a1,a2,ip,np)
        call daxpy(ip,1.0d0,xer2,1,xeb2,1)
        call daxpy(ip,-1.0d0,xer1,1,rnr,1)
        call daxpy(ip,-1.0d0,xeb2,1,rnb,1)

c-------------------------------------------------
c       Krithrio Termatismou
c-------------------------------------------------

        dnrn=0.0d0

        do i=1,ip
         dnrn=dnrn+rnr(i)**2.0d0+rnb(i)**2.0d0
        enddo

        if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
        endif

        call MPI_REDUCE(dnrn,dnrnup,1,MPI_DOUBLE_PRECISION,
     +                  MPI_SUM,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()

         dnrnup=dsqrt(dnrnup)
        if (k.eq.1) then
            wf=1.0d0
            dnrninit=dnrnup
        else
            wf=dnrnup/dnrninit
        endif
        if (wf.lt.tolex) then
            id=1
        else
            id=0
        endif
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
```

```
        endif

        call MPI_BCAST(id,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

         if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
         endif

         if (id.eq.1) goto 888
c_______________________________________________________________

        call dcopy(ip,rnr,1,brpart,1)
        call dcopy(ip,rnb,1,bbpart,1)
        call dcopy(ip,brpart,1,w,1)
        call drsolve(ns,w,M1,ipvt1,M2,ipvt2,myrank,ip,np)
        call dcopy(ip,w,1,brpart,1)

       if (myrank.eq.0) then
        tim2=MPI_Wtime()
        tcomp=tim2-tim1+tcomp
        tmc1=MPI_Wtime()
       endif

        if (mod(myrank,2).eq.1) then
  call MPI_SEND(w,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +                  111,MPI_COMM_WORLD,ierr)
          call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
     +                  MPI_COMM_WORLD,status,ierr)
        if (myrank.lt.np-2) then
          call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,
     +                  myrank+1,11,MPI_COMM_WORLD,status,ierr)
          call MPI_SEND(w(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
     +                  myrank+1,12,MPI_COMM_WORLD,ierr)
        endif
        else
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
     +                  111,MPI_COMM_WORLD,status,ierr)
          call MPI_SEND(w(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
     +                 myrank+1,11,MPI_COMM_WORLD,ierr)
        if (myrank.gt.0) then
    call MPI_SEND(w,inh,MPI_DOUBLE_PRECISION,
     +                  myrank-1,11,MPI_COMM_WORLD,ierr)
          call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +                  12,MPI_COMM_WORLD,status,ierr)
        endif
```

```
         endif

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()
        endif

        call matred(w,tt,ns,a3,a4,t,myrank,ip,np,v1,v2)
        call dcopy(ip,bbpart,1,w(1),1)
        call daxpy(ip,-1.0d0,tt,1,w,1)
        call dbsolve(ns,w,M1,ipvt1,M2,ipvt2,ip,np)
        call dcopy(ip,w,1,bbpart,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

        call MPI_Gather(brpart,ip,MPI_DOUBLE_PRECISION,b,ip,
     +                  MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
        call MPI_Gather(bbpart,ip,MPI_DOUBLE_PRECISION,b(n2+1),ip,
     +                  MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()
        endif

        call dcopy(n2,b(n2+1),1,bb,1)
        call dcopy(n2,bb,1,xb,1)
        iter=40
        resid=1.0d-3
c-------------------------------------------------------------------------
c-------------------------------------------------------------------------
c                  Bi-CGSTAB  BEGINS

c   BiCGSTAB without precondition of Ax=b

        if (myrank.eq.0) then
         imaxstep=iter
         tol=resid
         iter=0
         dnrmb=dnrm2(n2,xb,1)
         tim2=MPI_Wtime()
```

```
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  call MPI_Scatter(xb,ip,MPI_DOUBLE_PRECISION,brpart,ip,
+                  MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call dcopy(ip,brpart,1,bbpart,1)
  call dcopy(ip,brpart,1,w,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  if (mod(myrank,2).eq.1) then
   call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                111,MPI_COMM_WORLD,ierr)
   call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                11,MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
   call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+                11,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,myrank+1,
+                12,MPI_COMM_WORLD,ierr)
  endif
  else
   call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+                111,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
+                myrank+1,11,MPI_COMM_WORLD,ierr)
   if (myrank.gt.0) then
   call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,
+                myrank-1,11,MPI_COMM_WORLD,ierr)
   call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
```

```
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matblack(w,tt,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
  call drsolve(ns,tt,M1,ipvt1,M2,ipvt2,myrank,ip,np)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  if (mod(myrank,2).eq.1) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               111,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
+               MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
+               11,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,myrank+1,
+               12,MPI_COMM_WORLD,ierr)
  endif
  else
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
+               111,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
+               myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
  endif
  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matred(tt,w,ns,a3,a4,temp,myrank,ip,np,v1,v2)
  call dbsolve(ns,w,M1,ipvt1,M2,ipvt2,ip,np)
  call daxpy(ip,-1.0d0,w,1,brpart,1)
  call dcopy(ip,brpart,1,rpart,1)
```

```
      call dscal (ip,-1.0d0,rpart,1)
      call daxpy(ip,1.0d0,bbpart,1,rpart,1)
      call dcopy(ip,rpart,1,rhpart,1)
      call dcopy(ip,rpart,1,pi,1)
      roip1pt=ddot(ip,rhpart,1,rpart,1)

      if (myrank.eq.0) then
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
      endif

      call MPI_REDUCE(roip1pt,roip1,1,MPI_DOUBLE_PRECISION,
     +                MPI_SUM,0,MPI_COMM_WORLD,ierr)

      if (myrank.eq.0) then
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
       tim1=MPI_Wtime()
      endif

999   continue

      call dcopy(ip,pi,1,w,1)
      call dcopy(ip,pi,1,brpart,1)

      if (myrank.eq.0) then
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
      endif

      if (mod(myrank,2).eq.1) then
      call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +              111,MPI_COMM_WORLD,ierr)
      call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +              11,MPI_COMM_WORLD,status,ierr)
      if (myrank.lt.np-2) then
      call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
     +              11,MPI_COMM_WORLD,status,ierr)
      call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
     +              myrank+1,12,MPI_COMM_WORLD,ierr)
      endif
      else
      call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
     +              111,MPI_COMM_WORLD,status,ierr)
```

```
  call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
+               myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
  call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matblack(w,tt,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
  call drsolve(ns,tt,M1,ipvt1,M2,ipvt2,myrank,ip,np)

  if (myrank.eq.0) then
  tim2=MPI_Wtime()
  tcomp=tim2-tim1+tcomp
  tmc1=MPI_Wtime()
  endif

  if (mod(myrank,2).eq.1) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               111,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
+               MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,11,
+               MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,myrank+1,
+               12,MPI_COMM_WORLD,ierr)
  endif
  else
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
+               111,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
+               myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
```

```
      endif

      if (myrank.eq.0) then
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
       tim1=MPI_Wtime()
      endif

      call matred(tt,w,ns,a3,a4,temp,myrank,ip,np,v1,v2)
      call dbsolve(ns,w,M1,ipvt1,M2,ipvt2,ip,np)
      call daxpy(ip,-1.0d0,w,1,brpart,1)
      call dcopy(ip,brpart,1,ui,1)

      aip=ddot(ip,rhpart,1,ui,1)

      if (myrank.eq.0) then
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
      endif

      call MPI_REDUCE(aip,aii,1,MPI_DOUBLE_PRECISION,
     +                MPI_SUM,0,MPI_COMM_WORLD,ierr)

      if (myrank.eq.0)  ai=roip1/aii

      call MPI_BCAST(ai,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

      if (myrank.eq.0) then
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
       tim1=MPI_Wtime()
      endif

      call dcopy(ip,rpart,1,s,1)
      call daxpy(ip,-ai,ui,1,s,1)
      call dcopy(ip,s,1,brpart,1)
      call dcopy(ip,brpart,1,w,1)

      if (myrank.eq.0) then
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
      endif

      if (mod(myrank,2).eq.1) then
      call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,myrank-1,
```

```
+               111,MPI_COMM_WORLD,ierr)
  call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+               11,MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
  call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+               11,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,myrank+1,
+               12,MPI_COMM_WORLD,ierr)
  endif
  else
  call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+               111,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(w(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
+               myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(w,inh2,MPI_DOUBLE_PRECISION,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
  call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matblack(w,tt,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
  call drsolve(ns,tt,M1,ipvt1,M2,ipvt2,myrank,ip,np)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  if (mod(myrank,2).eq.1) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               111,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
+               MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
+               11,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
+               myrank+1,12,MPI_COMM_WORLD,ierr)
```

```
  endif
  else
  call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
+               111,MPI_COMM_WORLD,status,ierr)
  call MPI_SEND(tt(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
+               myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(tt,inh,MPI_DOUBLE_PRECISION,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
  call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matred(tt,w,ns,a3,a4,temp,myrank,ip,np,v1,v2)
  call dbsolve(ns,w,M1,ipvt1,M2,ipvt2,ip,np)
  call daxpy(ip,-1.0d0,w,1,brpart,1)
  call dcopy(ip,brpart,1,tpart,1)
    wia=ddot(ip,tpart,1,s,1)
    wib=ddot(ip,tpart,1,tpart,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  call MPI_REDUCE(wia,wa,1,MPI_DOUBLE_PRECISION,
+                 MPI_SUM,0,MPI_COMM_WORLD,ierr)
  call MPI_REDUCE(wib,wb,1,MPI_DOUBLE_PRECISION,
+                 MPI_SUM,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
   iter=iter+1
   wi=wa/wb
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
```

```
  endif

  call MPI_BCAST(wi,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call daxpy(ip,ai,pi,1,bbpart,1)
  call daxpy(ip,wi,s,1,bbpart,1)
  call daxpy(ip,-wi,tpart,1,s,1)
  call dcopy(ip,s,1,rpart,1)
  roip1p=ddot(ip,rhpart,1,rpart,1)

  if (myrank.eq.0) then
   roip2=roip1
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  call MPI_REDUCE(roip1p,roip1,1,MPI_DOUBLE_PRECISION,
+                 MPI_SUM,0,MPI_COMM_WORLD,ierr)
  if (myrank.eq.0)  bi=(roip1/roip2)*(ai/wi)
  call MPI_BCAST(bi,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

   call daxpy(ip,-wi,ui,1,pi,1)
   call dcopy(ip,rpart,1,tpart,1)
   call daxpy(ip,bi,pi,1,tpart,1)
   call dcopy(ip,tpart,1,pi,1)

   dnr=0.0d0
    do i=1,ip
     dnr=dnr+tpart(i)**2.0d0
    enddo

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
```

```
         tmc1=MPI_Wtime()
        endif

        call MPI_REDUCE(dnr,dnrmr,1,MPI_DOUBLE_PRECISION,
     +                  MPI_SUM,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()

         dnrmr=dsqrt(dnrmr)
         resid=dnrmr/dnrmb

        if (wi.ne.0.0d0.and.resid.gt.tol.and.iter.lt.imaxstep)
     + then
          id=1
        else
          id=0
        endif
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

        call MPI_BCAST(id,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
         if (id.eq.1) goto 999
         if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
         endif

c--------------BiCGSTAB END-----------------------------------

        call dcopy(n2,b,1,x,1)

        if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
        endif

        call MPI_Scatter(x,ip,MPI_DOUBLE_PRECISION,brpart,ip,
     +                   MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
```

```
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call dcopy(ip,bbpart,1,tt,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  if (mod(myrank,2).eq.1) then
   call MPI_SEND(tt,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                111,MPI_COMM_WORLD,ierr)
   call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                11,MPI_COMM_WORLD,status,ierr)
  if (myrank.lt.np-2) then
   call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+                11,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(tt(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
+                myrank+1,12,MPI_COMM_WORLD,ierr)
  endif
  else
   call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
+                111,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(tt(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
+                myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
   call MPI_SEND(tt,inh2,MPI_DOUBLE_PRECISION,
+                myrank-1,11,MPI_COMM_WORLD,ierr)
   call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
+                12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
    tmc2=MPI_Wtime()
    tcomm=tmc2-tmc1+tcomm
    tim1=MPI_Wtime()
  endif

  call matblack(tt,w,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
  call drsolve(ns,w,M1,ipvt1,M2,ipvt2,myrank,ip,np)
  call dcopy(ip,brpart,1,tt(1),1)
  call daxpy(ip,-1.0d0,w,1,tt,1)
```

```
        call dcopy(ip,tt,1,brpart,1)


        call dcopy(ip,brpart,1,cr,1)
        call dcopy(ip,bbpart,1,cb,1)
        call daxpy(ip,1.0d0,cr,1,xhr,1)
        call daxpy(ip,1.0d0,cb,1,xhb,1)
        call dcopy(ip,qr,1,rnr,1)
        call dcopy(ip,qb,1,rnb,1)

        enddo

c-----------------------------------------------------------------------
        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif
888     continue


c-----------------------------------------------------------------------
        call MPI_Gather(xhb,ip,MPI_DOUBLE_PRECISION,x(n2+1),ip,
     +                  MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
        call MPI_Gather(xhr,ip,MPI_DOUBLE_PRECISION,x,ip,
     +                  MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
c-----------------------------------------------------------------------
        if (myrank.eq.0) then
           print*,'||x||2=',dnrm2(n,x,1)
           print*,'||b-Ax||2=',dnrnup
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm

       print*,'External Steps    =',k
       print*,'BiCGSTAB Steps    = ',iter
       print*,'Total Communication Time=',tcomm
       print*,'Total Computation Time  =',tcomp
       print*,'Total Time  =',tcomp+tcomm
       print*,'-----------------------------------'
       endif
c-----------------------------------------------------------------------
       call MPI_FINALIZE(ierr)
       stop
       end
```

Υποπρογράμματα

```
      subroutine matblack(x,y,ns,a3,a4,t,myrank,ip,np,tf,tl)
      real*8 a3(5,*),a4(5,*),x(*),y(*),t(*),tf(*),tl(*)

c     Y=Hb*X  where Hb block part of collocation matrix

      inh=2*ns
      inh2=2*inh
      nsi=ns*inh
      ns2=ns/np

      if (myrank.eq.0) then
      call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(1),1,0.0d0,y,1)
      call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(inh+1),
     +           1,1.0d0,y,1)
      do k=1,ns2-3,2
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
     +           1,0.0d0,y(k*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
     +           1,1.0d0,y(k*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
     +           1,0.0d0,y((k+1)*inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
     +           1,1.0d0,y((k+1)*inh+1),1)
      call dcopy(inh,y((k+1)*inh+1),1,t,1)
      call daxpy(inh,-1.0d0,y(k*inh+1),1,y((k+1)*inh+1),1)
      call daxpy(inh,1.0d0,t,1,y(k*inh+1),1)
      enddo
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-2*inh+1),
     +           1,0.0d0,y(ip-inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-inh+1),
     +           1,1.0d0,y(ip-inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,tl,
     +           1,1.0d0,y(ip-inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tl(inh+1),
     +           1,1.0d0,y(ip-inh+1),1)
      elseif (myrank.eq.np-1) then
       call dgbmv('n',inh,inh,2,2,-1.0d0,a3,5,tf,
     +           1,0.0d0,y,1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tf(inh+1),
     +           1,1.0d0,y,1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x,
     +           1,1.0d0,y,1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(inh+1),
     +           1,1.0d0,y,1)
      do k=1,ns2-3,2
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
     +           1,0.0d0,y(k*inh+1),1)
```

```
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
+            1,1.0d0,y(k*inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
+            1,0.0d0,y((k+1)*inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
+            1,1.0d0,y((k+1)*inh+1),1)
 call dcopy(inh,y((k+1)*inh+1),1,t,1)
 call daxpy(inh,-1.0d0,y(k*inh+1),1,y((k+1)*inh+1),1)
 call daxpy(inh,1.0d0,t,1,y(k*inh+1),1)
 enddo
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-2*inh+1),
+            1,0.0d0,y(ip-inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-inh+1),
+            1,1.0d0,y(ip-inh+1),1)
 else
  call dgbmv('n',inh,inh,2,2,-1.0d0,a3,5,tf,
+            1,0.0d0,y,1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tf(inh+1),
+            1,1.0d0,y,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x,
+            1,1.0d0,y,1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(inh+1),
+            1,1.0d0,y,1)
 do k=1,ns2-3,2
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
+            1,0.0d0,y(k*inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
+            1,1.0d0,y(k*inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
+            1,0.0d0,y((k+1)*inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
+            1,1.0d0,y((k+1)*inh+1),1)
  call dcopy(inh,y((k+1)*inh+1),1,t,1)
  call daxpy(inh,-1.0d0,y(k*inh+1),1,y((k+1)*inh+1),1)
  call daxpy(inh,1.0d0,t,1,y(k*inh+1),1)
 enddo
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-2*inh+1),
+            1,0.0d0,y(ip-inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-inh+1),
+            1,1.0d0,y(ip-inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,tl,
+            1,1.0d0,y(ip-inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tl(inh+1),
+            1,1.0d0,y(ip-inh+1),1)
 endif
 return
 end
```

```
*****************************************************************
       subroutine matred(x,y,ns,a3,a4,t,myrank,ip,np,tf,tl)
       implicit real*8 (a-h,o-z)
       real*8 a3(5,*),a4(5,*),x(*),y(*),t(*),tf(*),tl(*)

c      Y=Hr*X  where Hr block part of collocation matrix

      inh=2*ns
      inh2=2*inh
      nsi=ns*inh
      ns2=ns/np

      if (myrank.eq.0) then
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x,1,0.0d0,
     +             y(1),1)
       call dcopy(inh,y(1),1,t,1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(inh2+1),1,0.0d0,
     +             y(inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(inh+1),1,1.0d0,
     +             y(inh+1),1)
       call daxpy(inh,1.0d0,y(inh+1),1,y(1),1)
       call daxpy(inh,-1.0d0,t,1,y(inh+1),1)

      do k=2,ns2-4,2
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
     +            1,0.0d0,y(k*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
     +            1,1.0d0,y(k*inh+1),1)
       call dcopy(inh,y(k*inh+1),1,t,1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
     +            1,0.0d0,y((k+1)*inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
     +            1,1.0d0,y((k+1)*inh+1),1)
       call daxpy(inh,1.0d0,y((k+1)*inh+1),1,y(k*inh+1),1)
       call daxpy(inh,-1.0d0,t,1,y((k+1)*inh+1),1)
      enddo
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-3*inh+1),
     +            1,0.0d0,y(ip-2*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-2*inh+1),
     +            1,1.0d0,y(ip-2*inh+1),1)
       call dcopy(inh,y(ip-2*inh+1),1,t,1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-inh+1),
     +            1,0.0d0,y(ip-inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tl,
     +             1,1.0d0,y(ip-inh+1),1)
       call daxpy(inh,1.0d0,y(ip-inh+1),1,y(ip-2*inh+1),1)
```

```
  call daxpy(inh,-1.0d0,t,1,y(ip-inh+1),1)
 elseif (myrank.eq.np-1) then
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,tf,
+             1,0.0d0,y,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x,
+             1,1.0d0,y,1)
  call dcopy(inh,y,1,t,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(inh+1),
+             1,0.0d0,y(inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(2*inh+1),
+             1,1.0d0,y(inh+1),1)
  call daxpy(inh,1.0d0,y(inh+1),1,y,1)
  call daxpy(inh,-1.0d0,t,1,y(inh+1),1)
 do k=2,ns2-4,2
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
+             1,0.0d0,y(k*inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
+             1,1.0d0,y(k*inh+1),1)
  call dcopy(inh,y(k*inh+1),1,t,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
+             1,0.0d0,y((k+1)*inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
+             1,1.0d0,y((k+1)*inh+1),1)
  call daxpy(inh,1.0d0,y((k+1)*inh+1),1,y(k*inh+1),1)
  call daxpy(inh,-1.0d0,t,1,y((k+1)*inh+1),1)
 enddo
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(ip-inh+1),1,0.0d0,
+             y(ip-inh+1),1)
  call dcopy(inh,y(ip-inh+1),1,t,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-3*inh+1),1,0.0d0,
+             y(ip-2*inh+1),1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-2*inh+1),1,1.0d0,
+             y(ip-2*inh+1),1)
  call daxpy(inh,-1.0d0,y(ip-2*inh+1),1,y(ip-inh+1),1)
  call daxpy(inh,1.0d0,t,1,y(ip-2*inh+1),1)
 else
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,tf,
+             1,0.0d0,y,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x,
+             1,1.0d0,y,1)
  call dcopy(inh,y,1,t,1)
  call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(inh+1),
+             1,0.0d0,y(inh+1),1)
  call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x(2*inh+1),
+             1,1.0d0,y(inh+1),1)
  call daxpy(inh,1.0d0,y(inh+1),1,y,1)
  call daxpy(inh,-1.0d0,t,1,y(inh+1),1)
```

```
      do k=2,ns2-4,2
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k-1)*inh+1),
     +            1,0.0d0,y(k*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(k*inh+1),
     +             1,1.0d0,y(k*inh+1),1)
       call dcopy(inh,y(k*inh+1),1,t,1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x((k+1)*inh+1),
     +             1,0.0d0,y((k+1)*inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,x((k+2)*inh+1),
     +             1,1.0d0,y((k+1)*inh+1),1)
       call daxpy(inh,1.0d0,y((k+1)*inh+1),1,y(k*inh+1),1)
       call daxpy(inh,-1.0d0,t,1,y((k+1)*inh+1),1)
      enddo
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-3*inh+1),
     +             1,0.0d0,y(ip-2*inh+1),1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,x(ip-2*inh+1),
     +             1,1.0d0,y(ip-2*inh+1),1)
       call dcopy(inh,y(ip-2*inh+1),1,t,1)
       call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,x(ip-inh+1),
     +             1,0.0d0,y(ip-inh+1),1)
       call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,tl,
     +             1,1.0d0,y(ip-inh+1),1)
       call daxpy(inh,1.0d0,y(ip-inh+1),1,y(ip-2*inh+1),1)
       call daxpy(inh,-1.0d0,t,1,y(ip-inh+1),1)
      endif
      return
      end
*********************************************************************

      subroutine matdblack(ns,x,y,a1,a2,ip,np)
      implicit real*8 (a-h,o-z)

      real*8 x(*),a1(5,*),a2(5,*),y(*)
      integer ip

c     Db*x = y where Db:the Black diagonal block of Collocation matrix

      inh=ns*2
      ns2=ns/np
!$ OMP PARALLEL DO
      do k=0,ns2-2,2
call dgbmv('n',inh,inh,2,2,2.0d0,a1,5,x(k*inh+1),1,0.0d0,
     +          y(k*inh+1),1)
call dgbmv('n',inh,inh,2,2,2.0d0,a2,5,x((k+1)*inh+1),1,0.0d0,
     +          y((k+1)*inh+1),1)
      enddo
```

```
!$ OMP END PARALLEL DO
      return
      end

**********************************************************************

      subroutine matdred(ns,x,y,a1,a2,myrank,ip,np)
      implicit real*8 (a-h,o-z)
      real*8 x(*),a1(5,*),a2(5,*),y(*)
      integer myrank,ip,np

c     Dr*x = y where Dr:the Red diagonal block of Collocation matrix

      inh=ns*2
      ns2=ns/np
      if (myrank.eq.0) then
    call dgbmv('n',inh,inh,2,2,1.0d0,a2,5,x,
     +              1,0.0d0,y,1)
    call dgbmv('n',inh,inh,2,2,2.0d0,a1,5,x(inh+1),
     +              1,0.0d0,y(inh+1),1)
!$OMP PARALLEL DO
      do k=2,ns2-2,2
    call dgbmv('n',inh,inh,2,2,2.0d0,a2,5,x(k*inh+1),
     +              1,0.0d0,y(k*inh+1),1)
    call dgbmv('n',inh,inh,2,2,2.0d0,a1,5,x((k+1)*inh+1),
     +              1,0.0d0,y((k+1)*inh+1),1)
      enddo
!$OMP END PARALLEL DO

      elseif (myrank.eq.np-1) then
      do k=0,ns2-4,2
    call dgbmv('n',inh,inh,2,2,2.0d0,a2,5,x(k*inh+1),
     +              1,0.0d0,y(k*inh+1),1)
    call dgbmv('n',inh,inh,2,2,2.0d0,a1,5,x((k+1)*inh+1),
     +              1,0.0d0,y((k+1)*inh+1),1)
      enddo
    call dgbmv('n',inh,inh,2,2,2.0d0,a2,5,x(ip-2*inh+1),
     +              1,0.0d0,y(ip-2*inh+1),1)
    call dgbmv('n',inh,inh,2,2,-1.0d0,a2,5,x(ip-inh+1),
     +              1,0.0d0,y(ip-inh+1),1)
     else
      do k=0,ns2-2,2
    call dgbmv('n',inh,inh,2,2,2.0d0,a2,5,x(k*inh+1),
     +              1,0.0d0,y(k*inh+1),1)
    call dgbmv('n',inh,inh,2,2,2.0d0,a1,5,x((k+1)*inh+1),
     +              1,0.0d0,y((k+1)*inh+1),1)
      enddo
```

```
      endif
      return
      end

*********************************************************************

      subroutine drsolve(ns,X,M1,ipvt1,M2,ipvt2,myrank,ip,np)

      implicit real*8 (a-h,o-z)
      real*8 X(*),M1(7,*),M2(7,*)
      integer ipvt1(*),ipvt2(*),myrank,ip,np

c     Solve   Dr*x = x where Dr:the Red diagonal block of Collocation matrix

      inh=ns*2
      ns2=ns/np

      if (myrank.eq.0) then
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(1),inh,info)
      call dscal(ip-inh,0.5d0,x(inh+1),1)
      call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x(inh+1),inh,info)

!$OMP PARALLEL DO
      do k=2,ns2-2,2
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x((k+1)*inh+1),inh,info)
      enddo
!$OMP END PARALLEL DO

      elseif (myrank.eq.np-1) then
      call dscal(ip-inh,0.5d0,x,1)
      do k=0,ns2-4,2
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x((k+1)*inh+1),inh,info)
      enddo

      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(ip-2*inh+1),inh,info)
      call dscal(inh,-1.0d0,x(ip-inh+1),1)
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(ip-inh+1),inh,info)
      else
      call dscal(ip,0.5d0,x,1)
      do k=0,ns2-2,2
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x((k+1)*inh+1),inh,info)
      enddo
      endif
      return
```

```
      end

*****************************************************************

      subroutine dbsolve(ns,X,M1,ipvt1,M2,ipvt2,ip,np)
      implicit real*8 (a-h,o-z)

      real*8 X(*),M1(7,*),M2(7,*)
      integer ipvt1(*),ipvt2(*),ip

c     Solve   Db*x = x where Db:the Black diagonal block of Collocation matrix

      inh=ns*2
      ns2=ns/np
      call dscal(ip,0.5d0,x(1),1)

!$ OMP PARALLEL DO
      do k=0,ns2-2,2
       call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x(k*inh+1),
     +              inh,info)
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x((k+1)*inh+1),
     +               inh,info)
      enddo
!$ OMP END PARALLEL DO

      return
      end
```

Α΄.1.2 Χρήση μικτής ακρίβειας υπολογισμών

Κυρίως πρόγραμμα

```
      parameter (ns=2048,np=128,inh=2*ns,inh2=4*ns,
     +           n=inh*inh,n2=n/2,ip=n2/np)
      implicit real*8 (a-h,o-z)
      include 'mpif.h'
      real*8  b(n),x(n),a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),
     +  bm(inh),temp(inh),v1(inh),v2(inh),tmp1(2*inh),tmp2(2*inh),
     +  cr(ip),cb(ip),xhr(ip),xhb(ip),rnr(ip),rnb(ip),
     +  qr(ip),qb(ip),xeb1(ip),xeb2(ip),xer1(ip),xer2(ip),
     +  bbpart(ip),brpart(ip),MPI_Wtime,dnrn,dnrnup,dnrninit,dsqrt,
     +  tolex,wf,tmc1,tmc2,tcomp,tcomm,tim1,tim2

      real  a11(5,inh),a22(5,inh),a33(5,inh),a44(5,inh),
     +  M11(7,inh),M22(7,inh),bbparts(ip),brparts(ip),bs(n),
```

```
     +  tmp11(2*inh),tmp22(2*inh),v11(inh),v22(inh),t(n2),tt(ip),
     +  rpart(ip),tpart(ip),w(ip),s(ip),bb(n2),ui(ip),
     +  xb(n2),rhpart(ip),pi(ip),temps(inh),dnr,dnrmr,dnrmb,aip,
     +  r(n2),wia,wib,wi,aii,bi,sngl,roip1pt,roip1,roip2,
     +  xe(n),wa,wb,resid,sqrt,sdot,snrm2,roip1p,ai,tol

      integer ipvt1(inh),ipvt2(inh),p,status(MPI_STATUS_SIZE),ierr


      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD,p,ierr)
      call start(ns,a1,a2,a3,a4,M11,M22,ipvt1,ipvt2,a11,a22,a33,a44)
      maxstep=280

      if (myrank.eq.0) then
       tolex=1.0d-9
       print*,'-----------------------------------------------------'
       print*,'Error Correction Mixed Precision Iterative Refinenement'
       print*,'-----------------------------------------------------'
       print*,'   Ns    = ',ns
       print*,'   Procs    = ',np
       print*,'   External Steps    =',maxstep
       tcomp=0.0d0
       tcomm=0.0d0

       call makeb(ns,b,bm)
       call redblack(ns,b,x)
       call makeredblack(ns,x,b)

       tmc1=MPI_Wtime()
      endif

c------To 2o melos xwrizetai se np kommatia---------------------

        call MPI_Scatter(b,ip,MPI_DOUBLE_PRECISION,brpart,ip,
     +                   MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
        call MPI_Scatter(b(n2+1),ip,MPI_DOUBLE_PRECISION,bbpart,ip,
     +                   MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) then
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
       endif

c-------Dhmiourgia rn kai xh se Red & Black----------------------
```

```
        if (myrank.eq.0) then
         tim1=MPI_Wtime()
        endif

        call dcopy(ip,brpart,1,qr,1)
        call dcopy(ip,bbpart,1,qb,1)
        call dcopy(ip,brpart,1,xhr,1)
        call dcopy(ip,bbpart,1,xhb,1)
        call dcopy(ip,qr,1,rnr,1)
        call dcopy(ip,qb,1,rnb,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
        endif

c---------------------------------------
c       Ekkinhsh epanalhptikhs diadikasias
c---------------------------------------

        do k=1,maxstep

        if (myrank.eq.0) then
         tim1=MPI_Wtime()
        endif

        call matdred(ns,xhr,xer1,a1,a2,myrank,ip,np)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

c---------------------------------------
c             Epikoinwnia gia thn Matblack
c---------------------------------------

         if (mod(myrank,2).eq.1) then
  call MPI_SEND(xhb,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  111,MPI_COMM_WORLD,ierr)
          call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  11,MPI_COMM_WORLD,status,ierr)
         if (myrank.lt.np-2) then
          call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,
     +                  myrank+1,11,MPI_COMM_WORLD,status,ierr)
          call MPI_SEND(xhb(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
```

```
     +                  myrank+1,12,MPI_COMM_WORLD,ierr)
         endif
        else
  call MPI_RECV(tmp2,inh2,MPI_DOUBLE_PRECISION,myrank+1,
     +                  111,MPI_COMM_WORLD,status,ierr)
          call MPI_SEND(xhb(ip-inh2+1),inh2,MPI_DOUBLE_PRECISION,
     +                  myrank+1,11,MPI_COMM_WORLD,ierr)
         if (myrank.gt.0) then
    call MPI_SEND(xhb,inh2,MPI_DOUBLE_PRECISION,
     +                  myrank-1,11,MPI_COMM_WORLD,ierr)
          call MPI_RECV(tmp1,inh2,MPI_DOUBLE_PRECISION,myrank-1,
     +                  12,MPI_COMM_WORLD,status,ierr)
         endif
         endif
         if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
         endif

          call matblackdb(xhb,xeb1,ns,a3,a4,v1,myrank,ip,np,tmp1,tmp2)
          call daxpy(ip,1.0d0,xeb1,1,xer1,1)
         if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
         endif

c---------------------------------------
c             Epikoinwnia gia thn Matblack
c---------------------------------------

         if (mod(myrank,2).eq.1) then
   call MPI_SEND(xhr,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +               111,MPI_COMM_WORLD,ierr)
       call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,11,
     +               MPI_COMM_WORLD,status,ierr)
         if (myrank.lt.np-2) then
       call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,
     +               myrank+1,11,MPI_COMM_WORLD,status,ierr)
       call MPI_SEND(xhr(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
     +               myrank+1,12,MPI_COMM_WORLD,ierr)
         endif
         else
   call MPI_RECV(v2,inh,MPI_DOUBLE_PRECISION,myrank+1,
     +               111,MPI_COMM_WORLD,status,ierr)
       call MPI_SEND(xhr(ip-inh+1),inh,MPI_DOUBLE_PRECISION,
```

```
     +               myrank+1,11,MPI_COMM_WORLD,ierr)
        if (myrank.gt.0) then
     call MPI_SEND(xhr,inh,MPI_DOUBLE_PRECISION,
     +               myrank-1,11,MPI_COMM_WORLD,ierr)
       call MPI_RECV(v1,inh,MPI_DOUBLE_PRECISION,myrank-1,
     +               12,MPI_COMM_WORLD,status,ierr)
        endif
        endif

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()
        endif

        call matreddb(xhr,xer2,ns,a3,a4,temp,myrank,ip,np,v1,v2)
        call matdblack(ns,xhb,xeb2,a1,a2,ip,np)
        call daxpy(ip,1.0d0,xer2,1,xeb2,1)
c-----------------------------------------------------------------
        call daxpy(ip,-1.0d0,xer1,1,rnr,1)
        call daxpy(ip,-1.0d0,xeb2,1,rnb,1)
c-------------------------------------------------
c               Krithrio Termatismou
c-------------------------------------------------

        dnrn=0.0d0
        do i=1,ip
         dnrn=dnrn+rnr(i)**2.0d0+rnb(i)**2.0d0
        enddo
        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

        call MPI_REDUCE(dnrn,dnrnup,1,MPI_DOUBLE_PRECISION,
     +                  MPI_SUM,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()

        dnrnup=dsqrt(dnrnup)
         if (k.eq.1) then
            wf=1.0d0
            dnrninit=dnrnup
```

```
          else
             wf=dnrnup/dnrninit
          endif
          if (wf.lt.tolex) then
             id=1
          else
             id=0
          endif
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
         endif

         call MPI_BCAST(id,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

          if (myrank.eq.0) then
           tmc2=MPI_Wtime()
           tcomm=tmc2-tmc1+tcomm
           tim1=MPI_Wtime()
          endif

          if (id.eq.1) goto 888

c-------------------Single-Precision-----------------------

         brparts=sngl(rnr)
         bbparts=sngl(rnb)

         call scopy(ip,brparts,1,w,1)
         call drsolve(ns,w,M11,ipvt1,M22,ipvt2,myrank,ip,np)
         call scopy(ip,w,1,brparts,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

        if (mod(myrank,2).eq.1) then
         call MPI_SEND(w,inh,MPI_REAL4,myrank-1,111,MPI_COMM_WORLD,ierr)
         call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,11,
     +                 MPI_COMM_WORLD,status,ierr)
        if (myrank.lt.np-2) then
         call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,11,
     +                 MPI_COMM_WORLD,status,ierr)
         call MPI_SEND(w(ip-inh+1),inh,MPI_REAL4,myrank+1,
     +                 12,MPI_COMM_WORLD,ierr)
```

```
   endif
   else
   call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                111,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(w(ip-inh+1),inh,MPI_REAL4,
+                myrank+1,11,MPI_COMM_WORLD,ierr)
   if (myrank.gt.0) then
   call MPI_SEND(w,inh,MPI_REAL4,myrank-1,
+                11,MPI_COMM_WORLD,ierr)
   call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,
+                12,MPI_COMM_WORLD,status,ierr)
   endif
   endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matred(w,tt,ns,a33,a44,t,myrank,ip,np,v11,v22)
  call scopy(ip,bbparts,1,w(1),1)
  call saxpy(ip,-1.0e0,tt,1,w,1)
  call dbsolve(ns,w,M11,ipvt1,M22,ipvt2,ip,np)
  call scopy(ip,w,1,bbparts,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  call MPI_Gather(brparts,ip,MPI_REAL4,bs,ip,
+                 MPI_REAL4,0,MPI_COMM_WORLD,ierr)
  call MPI_Gather(bbparts,ip,MPI_REAL4,bs(n2+1),ip,
+                 MPI_REAL4,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

   call scopy(n2,bs(n2+1),1,bb,1)
   call scopy(n2,bb,1,xb,1)
   iter=40
   resid=1.0e-3
```

```
c-----------------------------------------------------------------------
c                  Bi-CGSTAB  BEGINS

C      BiCGSTAB without precondition of Ax=b

       if (myrank.eq.0) then
        imaxstep=iter
        tol=resid
        iter=0
        dnrmb=snrm2(n2,xb,1)
        tim2=MPI_Wtime()
        tcomp=tim2-tim1+tcomp
        tmc1=MPI_Wtime()
       endif

       call MPI_Scatter(xb,ip,MPI_REAL4,brparts,ip,
     +                  MPI_REAL4,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) then
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
        tim1=MPI_Wtime()
       endif

        call scopy(ip,brparts,1,bbparts,1)
        call scopy(ip,brparts,1,w,1)

       if (myrank.eq.0) then
        tim2=MPI_Wtime()
        tcomp=tim2-tim1+tcomp
        tmc1=MPI_Wtime()
       endif

       if (mod(myrank,2).eq.1) then
        call MPI_SEND(w,inh2,MPI_REAL4,myrank-1,
     +                111,MPI_COMM_WORLD,ierr)
        call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
     +                11,MPI_COMM_WORLD,status,ierr)
       if (myrank.lt.np-2) then
        call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,11,
     +                MPI_COMM_WORLD,status,ierr)
        call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
     +                myrank+1,12,MPI_COMM_WORLD,ierr)
       endif
       else
        call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
```

```
+                111,MPI_COMM_WORLD,status,ierr)
   call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
+                myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
   call MPI_SEND(w,inh2,MPI_REAL4,
+                myrank-1,11,MPI_COMM_WORLD,ierr)
   call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
+                12,MPI_COMM_WORLD,status,ierr)
  endif
  endif
  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

  call matblack(w,tt,ns,a33,a44,v11,myrank,ip,np,tmp11,tmp22)
  call drsolve(ns,tt,M11,ipvt1,M22,ipvt2,myrank,ip,np)

   if (myrank.eq.0) then
    tim2=MPI_Wtime()
    tcomp=tim2-tim1+tcomp
    tmc1=MPI_Wtime()
  endif

    if (mod(myrank,2).eq.1) then
 call MPI_SEND(tt,inh,MPI_REAL4,myrank-1,
+                 111,MPI_COMM_WORLD,ierr)
     call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,11,
+                 MPI_COMM_WORLD,status,ierr)
    if (myrank.lt.np-2) then
     call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                 11,MPI_COMM_WORLD,status,ierr)
     call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
+                 myrank+1,12,MPI_COMM_WORLD,ierr)
    endif
   else
 call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                 111,MPI_COMM_WORLD,status,ierr)
     call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
+                 myrank+1,11,MPI_COMM_WORLD,ierr)
   if (myrank.gt.0) then
   call MPI_SEND(tt,inh,MPI_REAL4,
+                 myrank-1,11,MPI_COMM_WORLD,ierr)
     call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,
+                 12,MPI_COMM_WORLD,status,ierr)
   endif
```

```
         endif

        if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
        endif

         call matred(tt,w,ns,a33,a44,temps,myrank,ip,np,v11,v22)
         call dbsolve(ns,w,M11,ipvt1,M22,ipvt2,ip,np)
         call saxpy(ip,-1.0e0,w,1,brparts,1)
         call scopy(ip,brparts,1,rpart,1)


         call sscal(ip,-1.0e0,rpart,1)
         call saxpy(ip,1.0e0,bbparts,1,rpart,1)
         call scopy(ip,rpart,1,rhpart,1)
         call scopy(ip,rpart,1,pi,1)
      roip1pt=sdot(ip,rhpart,1,rpart,1)

         if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
         endif

         call MPI_REDUCE(roip1pt,roip1,1,MPI_REAL4,
       +                 MPI_SUM,0,MPI_COMM_WORLD,ierr)

         if (myrank.eq.0) then
          tmc2=MPI_Wtime()
          tcomm=tmc2-tmc1+tcomm
          tim1=MPI_Wtime()
         endif

  999    continue

         call scopy(ip,pi,1,w,1)
         call scopy(ip,pi,1,brparts,1)

         if (myrank.eq.0) then
          tim2=MPI_Wtime()
          tcomp=tim2-tim1+tcomp
          tmc1=MPI_Wtime()
         endif

        if (mod(myrank,2).eq.1) then
```

```
call MPI_SEND(w,inh2,MPI_REAL4,myrank-1,
 +                111,MPI_COMM_WORLD,ierr)
    call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
 +                11,MPI_COMM_WORLD,status,ierr)
   if (myrank.lt.np-2) then
    call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
 +                11,MPI_COMM_WORLD,status,ierr)
    call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
 +                myrank+1,12,MPI_COMM_WORLD,ierr)
   endif
   else
call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
 +                111,MPI_COMM_WORLD,status,ierr)
    call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
 +                myrank+1,11,MPI_COMM_WORLD,ierr)
   if (myrank.gt.0) then
  call MPI_SEND(w,inh2,MPI_REAL4,
 +                myrank-1,11,MPI_COMM_WORLD,ierr)
    call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
 +                12,MPI_COMM_WORLD,status,ierr)
   endif
  endif

   if (myrank.eq.0) then
    tmc2=MPI_Wtime()
    tcomm=tmc2-tmc1+tcomm
    tim1=MPI_Wtime()
   endif

    call matblack(w,tt,ns,a33,a44,v11,myrank,ip,np,tmp11,tmp22)
    call drsolve(ns,tt,M11,ipvt1,M22,ipvt2,myrank,ip,np)

    if (myrank.eq.0) then
     tim2=MPI_Wtime()
     tcomp=tim2-tim1+tcomp
     tmc1=MPI_Wtime()
    endif

    if (mod(myrank,2).eq.1) then
     call MPI_SEND(tt,inh,MPI_REAL4,myrank-1,
 +                 111,MPI_COMM_WORLD,ierr)
     call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,11,
 +                 MPI_COMM_WORLD,status,ierr)
    if (myrank.lt.np-2) then
     call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
 +                 11,MPI_COMM_WORLD,status,ierr)
     call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
```

```
+                 myrank+1,12,MPI_COMM_WORLD,ierr)
   endif
   else
call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                 111,MPI_COMM_WORLD,status,ierr)
    call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
+                 myrank+1,11,MPI_COMM_WORLD,ierr)
 if (myrank.gt.0) then
    call MPI_SEND(tt,inh,MPI_REAL4,myrank-1,11,MPI_COMM_WORLD,ierr)
    call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
 endif
 endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

   call matred(tt,w,ns,a33,a44,temps,myrank,ip,np,v11,v22)
   call dbsolve(ns,w,M11,ipvt1,M22,ipvt2,ip,np)
   call saxpy(ip,-1.0e0,w,1,brparts,1)
   call scopy(ip,brparts,1,ui,1)

  aip=sdot(ip,rhpart,1,ui,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

  call MPI_REDUCE(aip,aii,1,MPI_REAL4,MPI_SUM,0,
+                 MPI_COMM_WORLD,ierr)
  if (myrank.eq.0)  ai=roip1/aii

  call MPI_BCAST(ai,1,MPI_REAL4,0,MPI_COMM_WORLD,ierr)

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

   call scopy(ip,rpart,1,s,1)
   call saxpy(ip,-ai,ui,1,s,1)
```

```
     call scopy(ip,s,1,brparts,1)
     call scopy(ip,brparts,1,w,1)

    if (myrank.eq.0) then
     tim2=MPI_Wtime()
     tcomp=tim2-tim1+tcomp
     tmc1=MPI_Wtime()
    endif

    if (mod(myrank,2).eq.1) then
call MPI_SEND(w,inh2,MPI_REAL4,myrank-1,
 +                111,MPI_COMM_WORLD,ierr)
     call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
 +                11,MPI_COMM_WORLD,status,ierr)
    if (myrank.lt.np-2) then
     call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
 +                11,MPI_COMM_WORLD,status,ierr)
     call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
 +                myrank+1,12,MPI_COMM_WORLD,ierr)
    endif
    else
call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
 +                111,MPI_COMM_WORLD,status,ierr)
     call MPI_SEND(w(ip-inh2+1),inh2,MPI_REAL4,
 +                myrank+1,11,MPI_COMM_WORLD,ierr)
    if (myrank.gt.0) then
   call MPI_SEND(w,inh2,MPI_REAL4,
 +                myrank-1,11,MPI_COMM_WORLD,ierr)
     call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
 +                12,MPI_COMM_WORLD,status,ierr)
    endif
    endif

    if (myrank.eq.0) then
     tmc2=MPI_Wtime()
     tcomm=tmc2-tmc1+tcomm
     tim1=MPI_Wtime()
    endif

     call matblack(w,tt,ns,a33,a44,v11,myrank,ip,np,tmp11,tmp22)
     call drsolve(ns,tt,M11,ipvt1,M22,ipvt2,myrank,ip,np)

    if (myrank.eq.0) then
     tim2=MPI_Wtime()
     tcomp=tim2-tim1+tcomp
     tmc1=MPI_Wtime()
    endif
```

```
  if (mod(myrank,2).eq.1) then
call MPI_SEND(tt,inh,MPI_REAL4,myrank-1,
+                111,MPI_COMM_WORLD,ierr)
    call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,11,
+                MPI_COMM_WORLD,status,ierr)
   if (myrank.lt.np-2) then
    call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                11,MPI_COMM_WORLD,status,ierr)
    call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
+                myrank+1,12,MPI_COMM_WORLD,ierr)
   endif
  else
call MPI_RECV(v22,inh,MPI_REAL4,myrank+1,
+                111,MPI_COMM_WORLD,status,ierr)
    call MPI_SEND(tt(ip-inh+1),inh,MPI_REAL4,
+                myrank+1,11,MPI_COMM_WORLD,ierr)
  if (myrank.gt.0) then
  call MPI_SEND(tt,inh,MPI_REAL4,
+               myrank-1,11,MPI_COMM_WORLD,ierr)
    call MPI_RECV(v11,inh,MPI_REAL4,myrank-1,
+               12,MPI_COMM_WORLD,status,ierr)
  endif
  endif

  if (myrank.eq.0) then
   tmc2=MPI_Wtime()
   tcomm=tmc2-tmc1+tcomm
   tim1=MPI_Wtime()
  endif

   call matred(tt,w,ns,a33,a44,temps,myrank,ip,np,v11,v22)
   call dbsolve(ns,w,M11,ipvt1,M22,ipvt2,ip,np)
   call saxpy(ip,-1.0e0,w,1,brparts,1)

   call scopy(ip,brparts,1,tpart,1)
wia=sdot(ip,tpart,1,s,1)
    wib=sdot(ip,tpart,1,tpart,1)

  if (myrank.eq.0) then
   tim2=MPI_Wtime()
   tcomp=tim2-tim1+tcomp
   tmc1=MPI_Wtime()
  endif

   call MPI_REDUCE(wia,wa,1,MPI_REAL4,
+                 MPI_SUM,0,MPI_COMM_WORLD,ierr)
```

```
        call MPI_REDUCE(wib,wb,1,MPI_REAL4,
     +                  MPI_SUM,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) then
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
        tim1=MPI_Wtime()

        iter=iter+1

    wi=wa/wb

        tim2=MPI_Wtime()
        tcomp=tim2-tim1+tcomp
        tmc1=MPI_Wtime()
       endif

        call MPI_BCAST(wi,1,MPI_REAL4,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) then
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
        tim1=MPI_Wtime()
       endif

call saxpy(ip,ai,pi,1,bbparts,1)
call saxpy(ip,wi,s,1,bbparts,1)
call saxpy(ip,-wi,tpart,1,s,1)
call scopy(ip,s,1,rpart,1)
roip1p=sdot(ip,rhpart,1,rpart,1)


       if (myrank.eq.0) then
    roip2=roip1
        tim2=MPI_Wtime()
        tcomp=tim2-tim1+tcomp
        tmc1=MPI_Wtime()
       endif

       call MPI_REDUCE(roip1p,roip1,1,MPI_REAL4,
     +                 MPI_SUM,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) bi=(roip1/roip2)*(ai/wi)

       call MPI_BCAST(bi,1,MPI_REAL4,0,MPI_COMM_WORLD,ierr)

       if (myrank.eq.0) then
```

```
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
        tim1=MPI_Wtime()
       endif
call saxpy(ip,-wi,ui,1,pi,1)
call scopy(ip,rpart,1,tpart,1)
call saxpy(ip,bi,pi,1,tpart,1)
call scopy(ip,tpart,1,pi,1)

        dnr=0.0e0
       do i=1,ip
        dnr=dnr+tpart(i)**2.0e0
       enddo

      if (myrank.eq.0) then
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
      endif

      call MPI_REDUCE(dnr,dnrmr,1,MPI_REAL4,
     +                MPI_SUM,0,MPI_COMM_WORLD,ierr)

      if (myrank.eq.0) then
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
       tim1=MPI_Wtime()

       dnrmr=sqrt(dnrmr)
       resid=dnrmr/dnrmb
       if (wi.ne.(0.0e0).and.resid.gt.tol.and.iter.lt.imaxstep)
     +  then
        id=1
       else
        id=0
       endif
       tim2=MPI_Wtime()
       tcomp=tim2-tim1+tcomp
       tmc1=MPI_Wtime()
       endif

       call MPI_BCAST(id,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
       if (id.eq.1) goto 999
       if (myrank.eq.0) then
        tmc2=MPI_Wtime()
        tcomm=tmc2-tmc1+tcomm
        tim1=MPI_Wtime()
```

```
        endif

c---------------------BiCGSTAB END-----------------------------

        call scopy(n2,bs,1,xe,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

        call MPI_Scatter(xe,ip,MPI_REAL4,brparts,ip,
     +                   MPI_REAL4,0,MPI_COMM_WORLD,ierr)

        if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()
        endif

        call scopy(ip,bbparts,1,tt,1)

        if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
        endif

       if (mod(myrank,2).eq.1) then
    call MPI_SEND(tt,inh2,MPI_REAL4,myrank-1,
     +                111,MPI_COMM_WORLD,ierr)
        call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
     +                11,MPI_COMM_WORLD,status,ierr)
       if (myrank.lt.np-2) then
        call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
     +                11,MPI_COMM_WORLD,status,ierr)
        call MPI_SEND(tt(ip-inh2+1),inh2,MPI_REAL4,
     +                myrank+1,12,MPI_COMM_WORLD,ierr)
       endif
       else
    call MPI_RECV(tmp22,inh2,MPI_REAL4,myrank+1,
     +                111,MPI_COMM_WORLD,status,ierr)
        call MPI_SEND(tt(ip-inh2+1),inh2,MPI_REAL4,
     +                myrank+1,11,MPI_COMM_WORLD,ierr)
       if (myrank.gt.0) then
      call MPI_SEND(tt,inh2,MPI_REAL4,
```

```
     +                myrank-1,11,MPI_COMM_WORLD,ierr)
         call MPI_RECV(tmp11,inh2,MPI_REAL4,myrank-1,
     +                12,MPI_COMM_WORLD,status,ierr)
       endif
       endif
       if (myrank.eq.0) then
         tmc2=MPI_Wtime()
         tcomm=tmc2-tmc1+tcomm
         tim1=MPI_Wtime()
       endif
       call matblack(tt,w,ns,a33,a44,v11,myrank,ip,np,tmp11,tmp22)
       call drsolve(ns,w,M11,ipvt1,M22,ipvt2,myrank,ip,np)
       call scopy(ip,brparts,1,tt(1),1)
       call saxpy(ip,-1.0e0,w,1,tt,1)
       call scopy(ip,tt,1,brparts,1)



C-----------------Single END-------------------------

       cr=dble(brparts)
       cb=dble(bbparts)
       call daxpy(ip,1.0d0,cr,1,xhr,1)
       call daxpy(ip,1.0d0,cb,1,xhb,1)
       call dcopy(ip,qr,1,rnr,1)
       call dcopy(ip,qb,1,rnb,1)

       enddo
c-----------------------------------------------------------------------
       if (myrank.eq.0) then
         tim2=MPI_Wtime()
         tcomp=tim2-tim1+tcomp
         tmc1=MPI_Wtime()
       endif

888    continue
c-----------------------------------------------------------------------
       call MPI_Gather(xhb,ip,MPI_DOUBLE_PRECISION,x(n2+1),ip,
     +                 MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
       call MPI_Gather(xhr,ip,MPI_DOUBLE_PRECISION,x,ip,
     +                 MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
c-----------------------------------------------------------------------
       if (myrank.eq.0) then
          print*,'||x||2=',dnrm2(n,x,1)
          print*,'||b-Ax||2=',dnrnup
       tmc2=MPI_Wtime()
       tcomm=tmc2-tmc1+tcomm
```

```
      print*,'BiCGSTAB Steps    = ',iter
      print*,'Total Communication Time=',tcomm
      print*,'Total Computation Time  =',tcomp
      print*,'Total Time  =',tcomp+tcomm
      print*,'-----------------------------------'
      endif
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
      call MPI_FINALIZE(ierr)
      stop
      end
```

Α΄.2 Επίλυση του Schur Complement με τη μέθοδο Newton σε μονοεπεξεργαστικό σύστημα κοινής μνήμης

Α΄.2.1 Χρήση διπλής ακρίβειας υπολογισμών

Κυρίως πρόγραμμα

```
      parameter (ns=2048,inh=2*ns,n=4*ns*ns,n2=n/2,maxstep=280)
      implicit real*8 (a-h,o-z)
      include 'mpif.h'

      integer i,j,ipvt1(inh),ipvt2(inh),info,k,
     +        status(MPI_STATUS_SIZE),ierr,p
      real*8 b(n),x(n),xh(n),xe(n),rn(n),c(n),q(n),
     +       xb(n2),t(n2),r(n2),rh(n2),pi(n2),ph(n2),s(n2),
     +       sh(n2),w(n2),tt(n2),bb(n2),ui(n2),
     +       a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),
     +       M1(7,inh),M2(7,inh),bm(inh),bmm(inh),temp(inh),
     +       normupd,norminit,wf,resid,tim1,tim2,MPI_Wtime

      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD,p,ierr)

      tol=1.0d-9

      print*,'------------------------------------------------------'
      print*,'Error Correction Double Precision Iterative Refinenement'
      print*,'------------------------------------------------------'
      print*,'   Ns    = ',ns
      print*,'   Total External Steps    =',maxstep

      call makeb(ns,b,bm)
      call redblack(ns,b,x)
      call makeredblack(ns,x,b)
      call start(ns,a1,a2,a3,a4,M1,M2,ipvt1,ipvt2)
      call dcopy(n,b,1,q,1)
      call dcopy(n,b,1,xh,1)
      call dcopy(n,q,1,rn,1)

      wf=1.0d0
      tim1=MPI_Wtime()

      do k=1,maxstep
```

```
      call matvec(xh,xe,ns,a1,a2,a3,a4,temp)
      call daxpy(n,-1.0d0,xe,1,rn,1)

      if (k.eq.1)then
       norminit=dnrm2(n,rn,1)
      else
       normupd=dnrm2(n,rn,1)
       wf=normupd/norminit
      endif

      if (wf.lt.tol) then
        goto 10
      endif

c----------------------Inner Part Double Precision---------------

      call dcopy(n,rn,1,b,1)
      call makebhat(b,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt)
      call dcopy(n2,b(n2+1),1,bb,1)
      call dcopy(n2,bb,1,xb,1)
      resid=1.0d-3
      iter=40
      call bicgstab(ns,xb,bb,a3,a4,M1,M2,ipvt1,ipvt2,resid,
     +              bm,bmm,r,rh,pi,ph,t,s,sh,ui,iter,temp,w,tt)
      call dcopy(n2,b,1,x,1)
      call dcopy(n2,xb,1,x(n2+1),1)
      call makexhat(x,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt)
      call dcopy(n,x,1,c,1)

c-------------------------------------------------------------

      call daxpy(n,1.0d0,c,1,xh,1)
      call dcopy(n,q,1,rn,1)
      enddo

10    continue
      tim2=MPI_Wtime()
      print*,'External Steps    =',k-1
      print*,'Ns =',ns
      print*,'Time = ',tim2-tim1,'  secs'
      print*,'||X||2=',dnrm2(n,xh,1)
      print*,'||b-Ax||=',normupd

      call MPI_FINALIZE(ierr)
      stop
      end
```

Α΄.2.2 Χρήση μικτής ακρίβειας υπολογισμών

Κυρίως πρόγραμμα

```
      parameter (ns=2048,inh=2*ns,n=4*ns*ns,n2=n/2,maxstep=280)
      implicit real*8 (a-h,o-z)
      include 'mpif.h'

      integer i,j,ipvt1(inh),ipvt2(inh),info,k,iter
     +        status(MPI_STATUS_SIZE),ierr,p
      real*8 b(n),xe(n),rn(n),c(n),xh(n),
     +       a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),
     +       bbm(inh),temp(inh),tim1,tim2,
     +       normupd,norminit,wf,tolq,MPI_Wtime

      real*4 a11(5,inh),a22(5,inh),a33(5,inh),a44(5,inh),
     +       M11(7,inh),M22(7,inh),tt(n2),bs(n),ws(n2),
     +       temps(inh),bb(n2),xb(n2),x(n),ui(n2),
     +       bm(inh),bmm(inh),r(n2),rh(n2),pi(n2),ph(n2),
     +       t(n2),s(n2),sh(n2),sngl,snrm2,resid

      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD,p,ierr)

      print*,'-------------------------------------------------------'
      print*,'Error Correction Single Precision Iterative Refinenement'
      print*,'-------------------------------------------------------'
      print*,'   Ns    = ',ns
      print*,'   Total External Steps    =',maxstep

      tolq=1.0d-9
      tim1=0.0d0
      tim2=0.0d0

      call makeb(ns,b,bbm)
      call redblack(ns,b,xe)
      call makeredblack(ns,xe,b)
      call start(ns,a1,a2,a3,a4,M11,M22,ipvt1,ipvt2,a11,a22,a33,a44)
      call dcopy(n,b,1,xh,1)
      call dcopy(n,b,1,rn,1)

      wf=1.0d0

      tim1=MPI_Wtime()

      do k=1,maxstep
```

```
      call matvecdb(xh,xe,ns,a1,a2,a3,a4,temp)
      call daxpy(n,-1.0d0,xe,1,rn,1)

      if (k.eq.1)then
       norminit=dnrm2(n,rn,1)
      else
       normupd=dnrm2(n,rn,1)
       wf=normupd/norminit
      endif

      if (wf.lt.tolq) then
        goto 10
      endif

c----------------------Inner Part Single Precision---------------

      bs=sngl(rn)
      call makebhat(bs,ns,ws,M11,ipvt1,M22,ipvt2,a33,a44,temps,tt)
      call scopy(n2,bs(n2+1),1,bb,1)
      call scopy(n2,bb,1,xb,1)
      resid=1.0e-3
      iter=40
      call bicgstab(ns,xb,bb,a33,a44,M11,M22,ipvt1,ipvt2,resid,
     &              bm,bmm,r,rh,pi,ph,t,s,sh,ui,iter,temps,ws,tt)
      call scopy(n2,bs,1,x,1)
      call scopy(n2,xb,1,x(n2+1),1)
      call makexhat(x,ns,ws,M11,ipvt1,M22,ipvt2,a33,a44,temps,tt)
c-----------------------------------------------------------------

      c=dble(x)
      call daxpy(n,1.0d0,c,1,xh,1)
      call dcopy(n,b,1,rn,1)

      enddo

10    continue

      tim2=MPI_Wtime()

      print*,'External Steps    =',k-1
      print*,'Ns =',ns
      print*,'Time =',tim2-tim1,'  secs'
      print*,'||X||2=',dnrm2(n,xh,1)
      print*,'||b-Ax||=',normupd

      call MPI_FINALIZE(ierr)
```

```
stop
end
```

Α΄.3 Επίλυση του Schur Complement με τη μέθοδο Newton σε αρχιτεκτονικές πολλαπλών υπολογιστικών πυρήνων με χρήση GPU

Α΄.3.1 Κυρίως πρόγραμμα

```
      use openacc
      use omp_lib
      use bcgs
      use bmat
      use bcgs
      parameter (ns=2048,inh=2*ns,n=4*ns*ns,nsp1=ns+1,
     &           n2=n/2,maxstep=280)

      implicit real*8 (a-h,o-z)
      integer i,j,ipvt1(inh),ipvt2(inh),info,k,iter
      real*8 b(n),xe(n),rn(n),c(n),xh(n),
     +       a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),
     +       bbm(inh),temp(inh),y(nsp1),
     +       normupd,norminit,wf,tolq,t1(n2)

      real*4 a11(5,inh),a22(5,inh),a33(5,inh),a44(5,inh),
     +       M11(7,inh),M22(7,inh),tt(n2),bs(n),ws(n2),
     +       temps(inh),bb(n2),xb(n2),x(n),ui(n2),
     +       bm(inh),bmm(inh),r(n2),rh(n2),pi(n2),ph(n2),
     +       t(n2),s(n2),sh(n2),sngl,snrm2,resid,tm,dtime,ta(2),tmm

      call acc_init(acc_device_nvidia)

      tolq=1.0d-09

      call makeb(ns,b,bbm)
      call redblack(ns,b,xe)
      call makeredblack(ns,xe,b)
      call start(ns,a1,a2,a3,a4,M11,M22,ipvt1,ipvt2,a11,a22,a33,a44)
      call dcopy(n,b,1,xh,1)
      call dcopy(n,b,1,rn,1)

      wf=1.0d0
!$acc data copyin(a33(1:5,1:2*ns),a44(1:5,1:2*ns))
      tmm=omp_get_wtime()


      do k=1,maxstep
```

```
      call matvecdb(xh,xe,ns,a1,a2,a3,a4,temp)
      call daxpy(n,-1.0d0,xe,1,rn,1)

      if (k.eq.1)then
       norminit=dnrm2(n,rn,1)
      else
       normupd=dnrm2(n,rn,1)
       wf=normupd/norminit
      endif

      if (wf.lt.tolq) then
        goto 10
      endif

c-----------------------------------------------------------------
      bs=sngl(rn)
      call makebhatGPU(bs,ns,ws,M11,ipvt1,M22,ipvt2,a33,a44,temps,tt,t1)
      call scopy(n2,bs(n2+1),1,bb,1)
      call scopy(n2,bb,1,xb,1)
      resid=1.0e-4
      iter=40
      call bicgstab(ns,xb,bb,a33,a44,M11,M22,ipvt1,ipvt2,resid,
     &bm,bmm,r,rh,pi,ph,t,s,sh,ui,iter,temps,ws,tt)
      call scopy(n2,bs,1,x,1)
      call scopy(n2,xb,1,x(n2+1),1)
      call makexhat(x,ns,ws,M11,ipvt1,M22,ipvt2,a33,a44,temps,tt)
c-----------------------------------------------------------------

      c=dble(x)
      call daxpy(n,1.0d0,c,1,xh,1)
      call dcopy(n,b,1,rn,1)
      enddo

10    continue
!$acc end data

      tm=omp_get_wtime()-tmm
      print*,k-1,'iterations'
      print*,'Ns =',ns
      print*,'Time =',tm,'  secs'
      print*,'||X||2=',dnrm2(n,xh,1)
      print*,'||b-Ax||=',normupd


      stop
      end
```

Α΄.3.2 Υποπρογράμματα

Ακολουθούν τα κυριότερα υποπρογράμματα που χρησιμοποιούνται κατά την εκτέλεση του κυρίως προγράμματος, και συγκεκριμένα τα υποπρογράμματα `bicgstab` που υλοποιεί την επαναληπτική μέθοδο bicgstab, `matvecGPU` που υλοποιεί τους πολλαπλασιασμούς με τους πίνακες H_R και H_B με χρήση GPU και `makebhatGPU` που δημιουργεί το διάνυσμα $\hat{b}$. Τα υπόλοιπα υποπρογράμματα που χρησιμοποιούνται είναι αντίστοιχα με αυτά που έχουν παρουσιαστεί παραπάνω.

```
      module bcgs
      use bmat
      contains
      subroutine bicgstab(ns,x,b,a33,a44,M11,M22,ipvt1,ipvt2,
     +         error,t3,t4,r,rh,pi,ph,t,s,sh,ui,istep,temp,w,tt)
      implicit real*4 (a-h,o-z)
      real*4 x(*),b(*),a33(5,2*ns),a44(5,2*ns),M11(1:7,1:2*ns)
     +       M22(7,2*ns),r(*),t3(*),t4(*),rh(*),pi(*),ph(*),t(*)
     +       s(*),sh(*),ui(*),error,temp(*),w(*),tt(*)
      integer ipvt1(*),ipvt2(*)

C   BiCGSTAB without precondition of Ax=b

      imaxstep=istep
      tol=error
      istep=0

      n=2*ns*ns
      n2=n/2
      dnrmb=snrm2(n,b,1)
      call matvecGPU(x,r,ns,M11,M22,a33,a44,ipvt1,ipvt2,temp,w,tt)
      call sscal (n,-1.0e0,r,1)
      call saxpy(n,1.0e0,b,1,r,1)
      dnrm=snrm2(n,r,1)
      call scopy(n,r,1,rh,1)
      call scopy(n,r,1,pi,1)
      roip1=sdot(n,rh,1,r,1)

999   continue
       istep=istep+1
       if (roip1.eq.0.0e0) then
         print*,' BiCGSTAB Fails - Pi-1=0 '
         return
         stop
       endif

       call matvecGPU(pi,ui,ns,M11,M22,a33,a44,ipvt1,ipvt2,temp,w,tt)
       ai=roip1/sdot(n,rh,1,ui,1)
```

```
         call scopy(n,r,1,s,1)
         call saxpy(n,-ai,ui,1,s,1)
         if (snrm2(n,s,1).lt.1.0e-19) then
         call saxpy(n,ai,ph,1,x,1)
          print*,' ||s||2 is small enough after ',istep,' steps'
         else
          call matvecGPU(s,t,ns,M11,M22,a33,a44,ipvt1,ipvt2,temp,w,tt)
          wi=sdot(n,t,1,s,1)/sdot(n,t,1,t,1)
          call saxpy(n,ai,pi,1,x,1)
          call saxpy(n,wi,s,1,x,1)
          call saxpy(n,-wi,t,1,s,1)
          call scopy(n,s,1,r,1)
          dnrmr=snrm2(n,r,1)
          roip2=roip1
          roip1=sdot(n,rh,1,r,1)

          bi=(roip1/roip2)*(ai/wi)

          call saxpy(n,-wi,ui,1,pi,1)
          call scopy(n,r,1,t,1)
          call saxpy(n,bi,pi,1,t,1)
          call scopy(n,t,1,pi,1)
          error=dnrmr/dnrmb
          print*,'Error =',error,istep,snrm2(n,x,1)
          if (wi.ne.0.0e0.and.error.gt.tol.and.istep.lt.imaxstep)
     +   goto 999
          print*,' BiCGSTAB exit after ',istep,'  steps.',snrm2(n,x,1)
         endif
         return
         end subroutine
         end module

**********************************************

       module bmat
       contains
       subroutine matvecGPU(x,y,ns,M11,M22,a33,a44,ipvt1,
     +                    ipvt2,t,t1,t12)
        implicit real*4 (a-h,o-z)

        real*4 M11(1:7,1:2*ns),M22(1:7,1:2*ns),a33(1:5,1:2*ns),
     +      y(1:2*ns*ns),t(1:2*ns),t1(1:2*ns*ns),t12(1:2*ns*ns),snrm2,
     +      x(1:2*ns*ns),a44(1:5,2*ns)

        integer ipvt1(*),ipvt2(*),ns,k0,k1,k2,k3,i

c       Y=C*X  where C : reduced collocation matrix
```

```
       nsi=2*ns*ns
       inh=2*ns
       inh2=2*inh

!$acc data copyin(x(1:nsi)),
!$acc&     pcreate(t12(1:nsi),t1(1:nsi),t(1:inh)),
!$acc&     copyout(y(1:nsi)),
!$acc&     present(a33(1:5,1:2*ns),a44(1:5,1:2*ns))

!$acc kernels
!$acc loop vector(64)
        do i=1,inh
         t1(i)=a33(3,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         t1(i-1)=t1(i-1)+a33(2,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         t1(i+1)=t1(i+1)+a33(4,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i)=t1(i)+a33(1,i+2)*x(i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i+2)=t1(i+2)+a33(5,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh
         t1(i)=-a44(3,i)*x(inh+i)+t1(i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         t1(i-1)=t1(i-1)-a44(2,i)*x(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         t1(i+1)=t1(i+1)-a44(4,i)*x(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i)=t1(i)-a44(1,i+2)*x(inh+i+2)
        enddo
```

```
!$acc loop vector(64)
        do i=1,inh-2
         t1(i+2)=t1(i+2)-a44(5,i)*x(inh+i)
        enddo
!$acc end kernels
c//////////////////////////////
!$acc  kernels

!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh
         t1(k2+i)=a33(3,i)*x(k1+i)+a44(3,i)*x(k2+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=2,inh
         t1(k2+i-1)=t1(k2+i-1)+a33(2,i)*x(k1+i)+a44(2,i)*x(k2+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-1
         t1(k2+i+1)=t1(k2+i+1)+a33(4,i)*x(k1+i)+a44(4,i)*x(k2+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-2
         t1(k2+i)=t1(k2+i)+a33(1,i+2)*x(k1+i+2)+a44(1,i+2)*x(k2+i+2)
        enddo
        enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
```

```
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-2
         t1(k2+i+2)=t1(k2+i+2)+a33(5,i)*x(k1+i)+a44(5,i)*x(k2+i)
        enddo
        enddo
c/////////////////////////////////////////////////////
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh
         t1(k3+i)=a33(3,i)*x(k3+i)-a44(3,i)*x(k4+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=2,inh
         t1(k3+i-1)=t1(k3+i-1)+a33(2,i)*x(k3+i)-a44(2,i)*x(k4+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-1
         t1(k3+i+1)=t1(k3+i+1)+a33(4,i)*x(k3+i)-a44(4,i)*x(k4+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-2
         t1(k3+i)=t1(k3+i)+a33(1,i+2)*x(k3+i+2)-a44(1,i+2)*x(k4+i+2)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
```

```
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-2
         t1(k3+i+2)=t1(k3+i+2)+a33(5,i)*x(k3+i)-a44(5,i)*x(k4+i)
        enddo
      enddo
!$acc end kernels
c-----------------------------------------
!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
         t12(k3+i)=t1(k3+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
         t1(k3+i)=t1(k3+i)-t1(k2+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
         t1(k2+i)=t1(k2+i)+t12(k3+i)
       enddo
      enddo
!$acc end kernels
!$acc kernels
      k3=nsi-inh
      k2=nsi-2*inh
!$acc loop vector(64)
        do i=1,inh
         t1(k3+i)=a33(3,i)*x(k2+i)+a44(3,i)*x(k3+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         t1(k3+i-1)=t1(k3+i-1)+a33(2,i)*x(k2+i)+a44(2,i)*x(k3+i)
        enddo
!$acc loop vector(64)
```

```
        do i=1,inh-1
         t1(k3+i+1)=t1(k3+i+1)+a33(4,i)*x(k2+i)+a44(4,i)*x(k3+i)
       enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(k3+i)=t1(k3+i)+a33(1,i+2)*x(k2+i+2)+a44(1,i+2)*x(k3+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(k3+i+2)=t1(k3+i+2)+a33(5,i)*x(k2+i)+a44(5,i)*x(k3+i)
        enddo
!$acc end kernels

!$acc update host(t1(1:2*ns*ns))
c----------------------------------------
       call drsolve(ns,t1,M11,ipvt1,M22,ipvt2)
c----------------------------------------
!$acc update device(t1(1:2*ns*ns))

!$acc kernels
!$acc loop vector(64)
        do i=1,inh
         y(i)=a44(3,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(i-1)=y(i-1)+a44(2,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(i+1)=y(i+1)+a44(4,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(i)=y(i)+a44(1,i+2)*t1(i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(i+2)=y(i+2)+a44(5,i)*t1(i)
        enddo

!$acc loop vector(64)
        do i=1,inh
         y(inh+i)=-a44(3,i)*t1(inh2+i)+a33(3,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
```

```
         y(inh+i-1)=y(inh+i-1)-a44(2,i)*t1(inh2+i)+a33(2,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(inh+1+i)=y(inh+1+i)-a44(4,i)*t1(inh2+i)+a33(4,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
        y(inh+i)=y(inh+i)-a44(1,i+2)*t1(inh2+i+2)+a33(1,i+2)*t1(inh+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(inh+i+2)=y(inh+i+2)-a44(5,i)*t1(inh2+i)+a33(5,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh
         t(i)=y(i)
        enddo
!$acc loop independent vector(64)
       do i=1,inh
         y(i)=y(inh+i)+y(i)
       enddo
!$acc loop independent vector(64)
       do i=1,inh
         y(inh+i)=y(inh+i)-t(i)
       enddo
!$acc end kernels

!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh
         y(k1+i)=a33(3,i)*t1(k0+i)+a44(3,i)*t1(k1+i)
        enddo
       enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=2,inh
         y(k1+i-1)=y(k1+i-1)+a33(2,i)*t1(k0+i)+a44(2,i)*t1(k1+i)
        enddo
      enddo
```

```
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-1
         y(k1+i+1)=y(k1+i+1)+a33(4,i)*t1(k0+i)+a44(4,i)*t1(k1+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-2
         y(k1+i)=y(k1+i)+a33(1,i+2)*t1(k0+i+2)+a44(1,i+2)*t1(k1+i+2)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-2
         y(k1+i+2)=y(k1+i+2)+a33(5,i)*t1(k0+i)+a44(5,i)*t1(k1+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh
         y(k2+i)=a33(3,i)*t1(k2+i)-a44(3,i)*t1(k3+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=2,inh
         y(k2+i-1)=y(k2+i-1)+a33(2,i)*t1(k2+i)-a44(2,i)*t1(k3+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
```

```
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-1
         y(k2+i+1)=y(k2+i+1)+a33(4,i)*t1(k2+i)-a44(4,i)*t1(k3+i)
        enddo
       enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-2
         y(k2+i)=y(k2+i)+a33(1,i+2)*t1(k2+i+2)-a44(1,i+2)*t1(k3+i+2)
        enddo
       enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-2
         y(k2+i+2)=y(k2+i+2)+a33(5,i)*t1(k2+i)-a44(5,i)*t1(k3+i)
        enddo
      enddo
!$acc end kernels
!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
!$acc loop independent private(k1) gang(512) vector(64)
       do i=1,inh
         t12(k1+i)=y(k1+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
       do i=1,inh
         y(k1+i)=y(k1+i)+y(k2+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
```

```
       k2=(k+1)*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
       do i=1,inh
         y(k2+i)=y(k2+i)-t12(k1+i)
       enddo
      enddo

      k1=nsi-inh
      k2=nsi-2*inh
      k3=nsi-3*inh
!$acc loop vector(64)
        do i=1,inh
         y(k1+i)=-a44(3,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(k1+i-1)=y(k1+i-1)-a44(2,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(k1+i+1)=y(k1+i+1)-a44(4,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k1+i)=y(k1+i)-a44(1,i+2)*t1(k1+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k1+i+2)=y(k1+i+2)-a44(5,i)*t1(k1+i)
        enddo
      k1=nsi-inh
!$acc loop vector(64)
       do i=1,inh
        t(i)=y(k1+i)
       enddo
!$acc loop vector(64)
        do i=1,inh
         y(k2+i)=a33(3,i)*t1(k3+i)+a44(3,i)*t1(k2+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(k2+i-1)=y(k2+i-1)+a33(2,i)*t1(k3+i)+a44(2,i)*t1(k2+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(k2+i+1)=y(k2+i+1)+a33(4,i)*t1(k3+i)+a44(4,i)*t1(k2+i)
        enddo
```

```
!$acc loop vector(64)
        do i=1,inh-2
         y(k2+i)=y(k2+i)+a33(1,i+2)*t1(k3+i+2)+a44(1,i+2)*t1(k2+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k2+i+2)=y(k2+i+2)+a33(5,i)*t1(k3+i)+a44(5,i)*t1(k2+i)
        enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         y(k1+i)=y(k1+i)-y(k2+i)
       enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         y(k2+i)=t(i)+y(k2+i)
       enddo
!$acc end kernels
!$acc end data
c-----------------------------------------------------
       call dbsolve(ns,y,M11,ipvt1,M22,ipvt2)
       call sscal(nsi,-1.0e0,y,1)
       call saxpy(nsi,1.0e0,x,1,y,1)
      return
      end subroutine
      end module bmat
************************************************

       subroutine makebhatGPU(b,ns,tb,M11,ipvt1,M22,ipvt2,a33,
     +                        a44,t,tt,t12)
       real*4 a33(1:5,1:2*ns),a44(1:5,1:2*ns),b(*),M11(7,*),M22(7,*),
     +        t(1:inh),tb(1:nsi),tt(1:nsi),t12(1:2*ns*ns),
     +        snrm2
       integer ipvt1(*),ipvt2(*)

       nsi=2*ns*ns
       inh=2*ns
       inh2=2*inh
       call scopy(nsi,b(1),1,tb(1),1)
       call drsolve(ns,tb,M11,ipvt1,M22,ipvt2)
       call scopy(nsi,tb,1,b(1),1)
c-----------------------------------------------------------
!$acc data copyin(tb(1:nsi)),create(t(1:inh),t12(1:2*ns*ns)),
!$acc&copyout(tt(1:nsi)),present(a33(1:5,1:2*ns),a44(1:5,1:2*ns))
!$acc kernels
!$acc loop vector(64)
        do i=1,inh
         tt(i)=a44(3,i)*tb(i)
```

```
        enddo
!$acc loop vector(64)
        do i=2,inh
         tt(i-1)=tt(i-1)+a44(2,i)*tb(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         tt(i+1)=tt(i+1)+a44(4,i)*tb(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         tt(i)=tt(i)+a44(1,i+2)*tb(i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         tt(i+2)=tt(i+2)+a44(5,i)*tb(i)
        enddo

!$acc loop gang vector(64)
        do i=1,inh
         tt(inh+i)=-a44(3,i)*tb(inh2+i)+a33(3,i)*tb(inh+i)
        enddo
!$acc loop gang vector(64)
        do i=2,inh
         tt(inh+i-1)=tt(inh+i-1)-a44(2,i)*tb(inh2+i)+a33(2,i)*tb(inh+i)
        enddo
!$acc loop gang vector(64)
        do i=1,inh-1
         tt(inh+1+i)=tt(inh+1+i)-a44(4,i)*tb(inh2+i)+a33(4,i)*tb(inh+i)
        enddo
!$acc loop gang vector(64)
      do i=1,inh-2
      tt(inh+i)=tt(inh+i)-a44(1,i+2)*tb(inh2+i+2)+a33(1,i+2)*tb(inh+i+2)
      enddo
!$acc loop gang vector(64)
        do i=1,inh-2
        tt(inh+i+2)=tt(inh+i+2)-a44(5,i)*tb(inh2+i)+a33(5,i)*tb(inh+i)
        enddo

!$acc loop gang vector(64)
        do i=1,inh
         t(i)=tt(i)
        enddo
!$acc loop independent vector(64)
       do i=1,inh
         tt(i)=tt(inh+i)+tt(i)
       enddo
```

```
!$acc loop independent vector(64)
       do i=1,inh
         tt(inh+i)=tt(inh+i)-t(i)
       enddo
!$acc end kernels

!$acc kernels
!$acc loop independent gang
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(64)
        do i=1,inh
         tt(k1+i)=a33(3,i)*tb(k0+i)+a44(3,i)*tb(k1+i)
        enddo
!$acc loop independent vector(64)
        do i=2,inh
         tt(k1+i-1)=tt(k1+i-1)+a33(2,i)*tb(k0+i)+a44(2,i)*tb(k1+i)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-1
         tt(k1+i+1)=tt(k1+i+1)+a33(4,i)*tb(k0+i)+a44(4,i)*tb(k1+i)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-2
         tt(k1+i)=tt(k1+i)+a33(1,i+2)*tb(k0+i+2)+a44(1,i+2)*tb(k1+i+2)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-2
         tt(k1+i+2)=tt(k1+i+2)+a33(5,i)*tb(k0+i)+a44(5,i)*tb(k1+i)
        enddo
!$acc loop independent vector(64)
        do i=1,inh
         tt(k2+i)=a33(3,i)*tb(k2+i)-a44(3,i)*tb(k3+i)
        enddo
!$acc loop independent vector(64)
        do i=2,inh
         tt(k2+i-1)=tt(k2+i-1)+a33(2,i)*tb(k2+i)-a44(2,i)*tb(k3+i)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-1
         tt(k2+i+1)=tt(k2+i+1)+a33(4,i)*tb(k2+i)-a44(4,i)*tb(k3+i)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-2
```

```
         tt(k2+i)=tt(k2+i)+a33(1,i+2)*tb(k2+i+2)-a44(1,i+2)*tb(k3+i+2)
        enddo
!$acc loop independent vector(64)
        do i=1,inh-2
         tt(k2+i+2)=tt(k2+i+2)+a33(5,i)*tb(k2+i)-a44(5,i)*tb(k3+i)
        enddo
       enddo
!$acc end kernels

!$acc kernels
!$acc loop gang independent
      do k=2,ns-4,2
       k1=k*inh
!$acc loop independent vector(64)
       do i=1,inh
         t12(k1+i)=tt(k1+i)
        enddo
       enddo
!$acc loop gang independent
      do k=2,ns-4,2
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent vector(64)
       do i=1,inh
         tt(k1+i)=tt(k1+i)+tt(k2+i)
       enddo
!$acc loop independent vector(64)
       do i=1,inh
         tt(k2+i)=tt(k2+i)-t12(k1+i)
       enddo
      enddo
!$acc end kernels
!$acc kernels


      k1=nsi-inh
      k2=nsi-2*inh
      k3=nsi-3*inh
!$acc loop vector(64)
        do i=1,inh
         tt(k1+i)=-a44(3,i)*tb(k1+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         tt(k1+i-1)=tt(k1+i-1)-a44(2,i)*tb(k1+i)
        enddo
!$acc loop vector(64)
```

```
        do i=1,inh-1
         tt(k1+i+1)=tt(k1+i+1)-a44(4,i)*tb(k1+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         tt(k1+i)=tt(k1+i)-a44(1,i+2)*tb(k1+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         tt(k1+i+2)=tt(k1+i+2)-a44(5,i)*tb(k1+i)
        enddo
      k1=nsi-inh
!$acc loop gang vector(64)
       do i=1,inh
        t(i)=tt(k1+i)
       enddo
!$acc loop gang vector(64)
        do i=1,inh
         tt(k2+i)=a33(3,i)*tb(k3+i)+a44(3,i)*tb(k2+i)
        enddo
!$acc loop gang vector(64)
        do i=2,inh
         tt(k2+i-1)=tt(k2+i-1)+a33(2,i)*tb(k3+i)+a44(2,i)*tb(k2+i)
        enddo
!$acc loop gang vector(64)
        do i=1,inh-1
         tt(k2+i+1)=tt(k2+i+1)+a33(4,i)*tb(k3+i)+a44(4,i)*tb(k2+i)
        enddo
!$acc loop gang vector(64)
        do i=1,inh-2
         tt(k2+i)=tt(k2+i)+a33(1,i+2)*tb(k3+i+2)+a44(1,i+2)*tb(k2+i+2)
        enddo
!$acc loop gang vector(64)
        do i=1,inh-2
         tt(k2+i+2)=tt(k2+i+2)+a33(5,i)*tb(k3+i)+a44(5,i)*tb(k2+i)
        enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         tt(k1+i)=tt(k1+i)-tt(k2+i)
       enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         tt(k2+i)=t(i)+tt(k2+i)
       enddo
!$acc end kernels
!$acc end data
c--------------------------------------------------------------
```

```
       call scopy(nsi,b(nsi+1),1,tb(1),1)
       call saxpy(nsi,-1.0e0,tt,1,tb,1)
       call dbsolve(ns,tb,M11,ipvt1,M22,ipvt2)
       call scopy(nsi,tb,1,b(nsi+1),1)
      return
      end
*************************************************
```

Α΄.4 Επίλυση του Schur Complement σε αρχιτεκτονικές πολλαπλών υπολογιστικών πυρήνων με χρήση GPU

Α΄.4.1 Κυρίως πρόγραμμα

```
      use openacc
      use omp_lib
      use bmat
      use bicgs

      parameter (ns=256,inh=2*ns,n=4*ns*ns,nsp1=ns+1,n2=n/2)

      implicit real*8 (a-h,o-z)

      real*8 b(n),x(n),M1(7,inh),M2(7,inh),xb(n2),
     +  a1(5,inh),a2(5,inh),a3(5,inh),a4(5,inh),t(n2),
     +  bm(inh),xe(n),bmm(inh),y(nsp1),r(n2),rh(n2),pi(n2),
     +  ph(n2),s(n2),sh(n2),resid,temp(inh),w(n2),tt(n2),
     +  bb(n2),ui(n2),tm0,tm

      integer ipvt1(inh),ipvt2(inh)

      call acc_init(acc_device_nvidia)

      iter=294
       print*,'_____________________________________________________'
       print*,'           Ns = ',ns
       call makeb(ns,b,bm)
       call redblack(ns,b,x)
       call makeredblack(ns,x,b)
       call start(ns,a1,a2,a3,a4,M1,M2,ipvt1,ipvt2)

       tm0=omp_get_wtime()

       call makebhat(b,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt)

       resid=1.0d-11

       call dcopy(n2,b(n2+1),1,bb,1)
       call dcopy(n2,bb,1,xb,1)

!$acc data copyin(a3(1:5,1:inh),a4(1:5,1:inh))
       call bicgstab(ns,xb,bb,a3,a4,M1,M2,ipvt1,ipvt2,resid,
     +               bm,bmm,r,rh,pi,ph,t,s,sh,ui,iter,w,tt)
```

```
!$acc end data

       call dcopy(n2,b,1,x,1)
       call dcopy(n2,xb,1,x(n2+1),1)
       call makexhat(x,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt)

       tm=omp_get_wtime()-tm0

       print*,'Total Time  =',tm,'  secs'
       print*,'Iterations = ',iter
       print*,'||x||2=', dnrm2(n,x,1)


       call exact(ns,b,y)
       call redblack(ns,b,xe)
       call matvec(x,xe,ns,a1,a2,a3,a4,temp)

       call makeb(ns,b,bm)
       call redblack(ns,b,x)
       call makeredblack(ns,x,b)
       call daxpy(n,-1.0d0,xe,1,b,1)
       print*,'||b - Ax||2 =',dnrm2(n,b,1)
       print*,'_____________________________________________'
       call acc_shutdown(acc_device_nvidia)
       stop

       end
```

Α΄.4.2 Υποπρογράμματα

Παρακάτω ακολουθούν τα κυριότερα υποπρογράμματα που χρησιμοποιούνται κατά την εκτέλεση του κυρίως προγράμματος. Συγκεκριμένα, τα υποπρογράμματα `bicgstab` που υλοποιεί την ομώνυμη επαναληπτική μέθοδο, `bmatvec` που υλοποιεί τους πολλαπλασιασμούς με τους πίνακες H_R και H_B με χρήση GPU, `makebhat` και `makexhat` που δημιουργούν τα διανύσματα $\hat{b}$ και $\hat{x}$, και τέλος τα `drsolve` και `dbsolve` που επιλύουν τα γραμμικά συστήματα όπου συμμετέχουν οι πίνακες D_R και D_B.

```
      module bicgs
      contains
      subroutine bicgstab(ns,x,b,a3,a4,M1,M2,ipvt1,ipvt2,error,
     +                    t3,t4,r,rh,pi,ph,t,s,sh,ui,istep,w,tt)
      use bmat
```

```
      implicit real*8 (a-h,o-z)
      real*8 x(*),b(*),a3(1:5,1:2*ns),a4(1:5,1:2*ns),M1(7,*)
     +       M2(7,*),r(*),t3(*),t4(*),rh(*),pi(*),ph(*),t(*)
     +       s(*),sh(*),ui(*),error,w(*),tt(*)
      integer ipvt1(*),ipvt2(*)

c    BiCGSTAB without precondition of Ax=b

      imaxstep=istep
      tol=error
      istep=0
      n=2*ns*ns
      inh=2*ns
      dnrmb=dnrm2(n,b,1)

      call bmatvec(x,r,ns,M1,M2,a3,a4,ipvt1,ipvt2,w,tt,ph)
      call dscal (n,-1.0d0,r,1)
      call daxpy(n,1.0d0,b,1,r,1)
      dnrm=dnrm2(n,r,1)
      call dcopy(n,r,1,rh,1)
      call dcopy(n,r,1,pi,1)
      roip1=ddot(n,rh,1,r,1)

999   continue
       istep=istep+1
       if (roip1.eq.0.0d0) then
         print*,' BiCGSTAB Fails - Pi-1=0 '
         return
         stop
       endif

       call bmatvec(pi,ui,ns,M1,M2,a3,a4,ipvt1,ipvt2,w,tt,ph)
       ai=roip1/ddot(n,rh,1,ui,1)
       call dcopy(n,r,1,s,1)
       call daxpy(n,-ai,ui,1,s,1)
       if (dnrm2(n,s,1).lt.1.0d-19) then
        call daxpy(n,ai,ph,1,x,1)
c        print*,' ||s||2 is small enough after ',istep,' steps'
       else
        call bmatvec(s,t,ns,M1,M2,a3,a4,ipvt1,ipvt2,w,tt,ph)
        wi=ddot(n,t,1,s,1)/ddot(n,t,1,t,1)
        call daxpy(n,ai,pi,1,x,1)
        call daxpy(n,wi,s,1,x,1)
        call daxpy(n,-wi,t,1,s,1)
        call dcopy(n,s,1,r,1)
        dnrmr=dnrm2(n,r,1)
```

```
          roip2=roip1
          roip1=ddot(n,rh,1,r,1)
          bi=(roip1/roip2)*(ai/wi)
          call daxpy(n,-wi,ui,1,pi,1)
          call dcopy(n,r,1,t,1)
          call daxpy(n,bi,pi,1,t,1)
          call dcopy(n,t,1,pi,1)
          error=dnrmr/dnrmb
           print*,'Error =',error,dnrm2(n,x,1),istep

          if (wi.ne.0.0d0.and.error.gt.tol.and.istep.lt.imaxstep)
     +    goto 999
           print*,' BiCGSTAB exit after ',istep,'  steps.'
         endif

         return
         end subroutine
         end module
*****************************************************************

       module bmat
       contains
       subroutine bmatvec(x,y,ns,M1,M2,a3,a4,ipvt1,ipvt2,ar,t1,t12)

        implicit real*8 (a-h,o-z)
        real*8 M1(7,*),M2(7,*),a3(1:5,1:2*ns),a4(1:5,2*ns),
     +         x(1:4*ns*ns),y(1:2*ns*ns),dnrm2,
     +         ar(1:2*ns),t1(1:2*ns*ns),t12(1:2*ns*ns),
        integer ipvt1(*),ipvt2(*)

C       Y=C*X  where C : reduced collocation matrix

        nsi=2*ns*ns
        inh=2*ns
        inh2=2*inh

!$acc data copyin(x(1:nsi))
!$acc&     pcreate(t12(1:nsi),t1(1:nsi),ar(1:inh))
!$acc&     copyout(y(1:nsi))
!$acc&     present(a3(1:5,1:inh),a4(1:5,1:inh))

!$acc kernels
!$acc loop vector(64)
         do i=1,inh
          t1(i)=a3(3,i)*x(i)
         enddo
!$acc loop vector(64)
```

```
        do i=2,inh
         t1(i-1)=t1(i-1)+a3(2,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         t1(i+1)=t1(i+1)+a3(4,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i)=t1(i)+a3(1,i+2)*x(i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i+2)=t1(i+2)+a3(5,i)*x(i)
        enddo
!$acc loop vector(64)
        do i=1,inh
         t1(i)=-a4(3,i)*x(inh+i)+t1(i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         t1(i-1)=t1(i-1)-a4(2,i)*x(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         t1(i+1)=t1(i+1)-a4(4,i)*x(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i)=t1(i)-a4(1,i+2)*x(inh+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(i+2)=t1(i+2)-a4(5,i)*x(inh+i)
        enddo
!$acc end kernels
c-------------------------------------------------------
!$acc  kernels

!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh
         t1(k2+i)=a3(3,i)*x(k1+i)+a4(3,i)*x(k2+i)
        enddo
```

```
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=2,inh
         t1(k2+i-1)=t1(k2+i-1)+a3(2,i)*x(k1+i)+a4(2,i)*x(k2+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-1
         t1(k2+i+1)=t1(k2+i+1)+a3(4,i)*x(k1+i)+a4(4,i)*x(k2+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-2
         t1(k2+i)=t1(k2+i)+a3(1,i+2)*x(k1+i+2)+a4(1,i+2)*x(k2+i+2)
        enddo
        enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
        do i=1,inh-2
         t1(k2+i+2)=t1(k2+i+2)+a3(5,i)*x(k1+i)+a4(5,i)*x(k2+i)
        enddo
        enddo
c-------------------------------------------------------
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh
         t1(k3+i)=a3(3,i)*x(k3+i)-a4(3,i)*x(k4+i)
        enddo
      enddo
```

```
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=2,inh
         t1(k3+i-1)=t1(k3+i-1)+a3(2,i)*x(k3+i)-a4(2,i)*x(k4+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-1
         t1(k3+i+1)=t1(k3+i+1)+a3(4,i)*x(k3+i)-a4(4,i)*x(k4+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-2
         t1(k3+i)=t1(k3+i)+a3(1,i+2)*x(k3+i+2)-a4(1,i+2)*x(k4+i+2)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent private(k3,k4) gang(512) vector(64)
        do i=1,inh-2
         t1(k3+i+2)=t1(k3+i+2)+a3(5,i)*x(k3+i)-a4(5,i)*x(k4+i)
        enddo
      enddo
!$acc end kernels
c-----------------------------------------------------
!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
        t12(k3+i)=t1(k3+i)
       enddo
      enddo
```

```
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
         t1(k3+i)=t1(k3+i)-t1(k2+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=1,ns-3,2
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
       do i=1,inh
         t1(k2+i)=t1(k2+i)+t12(k3+i)
       enddo
      enddo
!$acc end kernels
!$acc kernels
      k3=nsi-inh
      k2=nsi-2*inh
!$acc loop vector(64)
        do i=1,inh
         t1(k3+i)=a3(3,i)*x(k2+i)+a4(3,i)*x(k3+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         t1(k3+i-1)=t1(k3+i-1)+a3(2,i)*x(k2+i)+a4(2,i)*x(k3+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         t1(k3+i+1)=t1(k3+i+1)+a3(4,i)*x(k2+i)+a4(4,i)*x(k3+i)
       enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(k3+i)=t1(k3+i)+a3(1,i+2)*x(k2+i+2)+a4(1,i+2)*x(k3+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         t1(k3+i+2)=t1(k3+i+2)+a3(5,i)*x(k2+i)+a4(5,i)*x(k3+i)
        enddo
!$acc end kernels
c-----------------------------------------------------
!$acc update host(t1(1:2*ns*ns))

       call drsolve(ns,t1,M1,ipvt1,M2,ipvt2)
```

```
!$acc update device(t1(1:2*ns*ns))
c-----------------------------------------------------
!$acc kernels
!$acc loop vector(64)
        do i=1,inh
         y(i)=a4(3,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(i-1)=y(i-1)+a4(2,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(i+1)=y(i+1)+a4(4,i)*t1(i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(i)=y(i)+a4(1,i+2)*t1(i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(i+2)=y(i+2)+a4(5,i)*t1(i)
        enddo

!$acc loop vector(64)
        do i=1,inh
         y(inh+i)=-a4(3,i)*t1(inh2+i)+a3(3,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(inh+i-1)=y(inh+i-1)-a4(2,i)*t1(inh2+i)+a3(2,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(inh+1+i)=y(inh+1+i)-a4(4,i)*t1(inh2+i)+a3(4,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(inh+i)=y(inh+i)-a4(1,i+2)*t1(inh2+i+2)+a3(1,i+2)*t1(inh+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(inh+i+2)=y(inh+i+2)-a4(5,i)*t1(inh2+i)+a3(5,i)*t1(inh+i)
        enddo
!$acc loop vector(64)
        do i=1,inh
```

```
         ar(i)=y(i)
        enddo
!$acc loop independent vector(64)
       do i=1,inh
         y(i)=y(inh+i)+y(i)
       enddo
!$acc loop independent vector(64)
       do i=1,inh
         y(inh+i)=y(inh+i)-ar(i)
       enddo
!$acc end kernels

!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh
         y(k1+i)=a3(3,i)*t1(k0+i)+a4(3,i)*t1(k1+i)
        enddo
       enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=2,inh
         y(k1+i-1)=y(k1+i-1)+a3(2,i)*t1(k0+i)+a4(2,i)*t1(k1+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-1
         y(k1+i+1)=y(k1+i+1)+a3(4,i)*t1(k0+i)+a4(4,i)*t1(k1+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-2
         y(k1+i)=y(k1+i)+a3(1,i+2)*t1(k0+i+2)+a4(1,i+2)*t1(k1+i+2)
```

```
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent private(k0,k1) gang(512) vector(64)
        do i=1,inh-2
         y(k1+i+2)=y(k1+i+2)+a3(5,i)*t1(k0+i)+a4(5,i)*t1(k1+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh
         y(k2+i)=a3(3,i)*t1(k2+i)-a4(3,i)*t1(k3+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=2,inh
         y(k2+i-1)=y(k2+i-1)+a3(2,i)*t1(k2+i)-a4(2,i)*t1(k3+i)
        enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-1
         y(k2+i+1)=y(k2+i+1)+a3(4,i)*t1(k2+i)-a4(4,i)*t1(k3+i)
        enddo
       enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-2
         y(k2+i)=y(k2+i)+a3(1,i+2)*t1(k2+i+2)-a4(1,i+2)*t1(k3+i+2)
        enddo
       enddo
```

```
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent private(k2,k3) gang(512) vector(64)
        do i=1,inh-2
         y(k2+i+2)=y(k2+i+2)+a3(5,i)*t1(k2+i)-a4(5,i)*t1(k3+i)
        enddo
      enddo
!$acc end kernels
!$acc kernels
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
!$acc loop independent private(k1) gang(512) vector(64)
       do i=1,inh
         t12(k1+i)=y(k1+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
       do i=1,inh
         y(k1+i)=y(k1+i)+y(k2+i)
       enddo
      enddo
!$acc loop independent gang(1024) vector(64)
      do k=2,ns-4,2
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent private(k1,k2) gang(512) vector(64)
       do i=1,inh
         y(k2+i)=y(k2+i)-t12(k1+i)
       enddo
      enddo

      k1=nsi-inh
      k2=nsi-2*inh
      k3=nsi-3*inh
!$acc loop vector(64)
        do i=1,inh
         y(k1+i)=-a4(3,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
```

```
         y(k1+i-1)=y(k1+i-1)-a4(2,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(k1+i+1)=y(k1+i+1)-a4(4,i)*t1(k1+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k1+i)=y(k1+i)-a4(1,i+2)*t1(k1+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k1+i+2)=y(k1+i+2)-a4(5,i)*t1(k1+i)
        enddo
      k1=nsi-inh
!$acc loop vector(64)
       do i=1,inh
        ar(i)=y(k1+i)
       enddo
!$acc loop vector(64)
        do i=1,inh
         y(k2+i)=a3(3,i)*t1(k3+i)+a4(3,i)*t1(k2+i)
        enddo
!$acc loop vector(64)
        do i=2,inh
         y(k2+i-1)=y(k2+i-1)+a3(2,i)*t1(k3+i)+a4(2,i)*t1(k2+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-1
         y(k2+i+1)=y(k2+i+1)+a3(4,i)*t1(k3+i)+a4(4,i)*t1(k2+i)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k2+i)=y(k2+i)+a3(1,i+2)*t1(k3+i+2)+a4(1,i+2)*t1(k2+i+2)
        enddo
!$acc loop vector(64)
        do i=1,inh-2
         y(k2+i+2)=y(k2+i+2)+a3(5,i)*t1(k3+i)+a4(5,i)*t1(k2+i)
        enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         y(k1+i)=y(k1+i)-y(k2+i)
       enddo
!$acc loop gang independent vector(64)
       do i=1,inh
         y(k2+i)=ar(i)+y(k2+i)
       enddo
```

```
!$acc end kernels
!$acc end data
c--------------------------------------------------------
       call dbsolve(ns,y,M1,ipvt1,M2,ipvt2)
       call dscal(nsi,-1.0d0,y,1)
       call daxpy(nsi,1.0d0,x,1,y,1)
      return
      end subroutine
      end module bmat

*****************************************************************

      subroutine makebhat(b,ns,tb,M1,ipvt1,M2,ipvt2,a3,a4,t,tt)

       real*8 a3(5,*),a4(5,*),b(*),M1(7,*),M2(7,*),t(*),tb(*),tt(*)
       integer ipvt1(*),ipvt2(*)

       nsi=2*ns*ns
       call dcopy(nsi,b(1),1,tb(1),1)
       call drsolve(ns,tb,M1,ipvt1,M2,ipvt2)
       call dcopy(nsi,tb,1,b(1),1)
       call matred(tb,tt,ns,a3,a4,t)
       call dcopy(nsi,b(nsi+1),1,tb(1),1)
       call daxpy(nsi,-1.0d0,tt,1,tb,1)
       call dbsolve(ns,tb,M1,ipvt1,M2,ipvt2)
       call dcopy(nsi,tb,1,b(nsi+1),1)

      return
      end
*****************************************************************

       subroutine makexhat(x,ns,tb,M1,ipvt1,M2,ipvt2,a3,a4,t,tt)

       real*8 a3(1:5,1:2*ns),a4(1:5,1:2*ns),x(1:4*ns*ns),t(1:2*ns),
     +        tb(1:2*ns*ns),tt(1:2*ns*ns), M1(7,*),M2(7,*),dnrm2
       integer ipvt1(*),ipvt2(*)

       nsi=2*ns*ns
       inh=2*ns
       call dcopy(2*ns*ns,x(nsi+1:),1,tt,1)
       call matblack(tt,tb,ns,a3,a4,t)
       call drsolve(ns,tb,M1,ipvt1,M2,ipvt2)
       call dcopy(nsi,x,1,tt,1)
       call daxpy(nsi,-1.0d0,tb,1,tt,1)
       call dcopy(nsi,tt,1,x,1)

      return
```

```
      end subroutine
*****************************************************************

      subroutine drsolve(ns,X,M1,ipvt1,M2,ipvt2)
      implicit real*8 (a-h,o-z)
      real*8 X(*),M1(7,*),M2(7,*)
      integer ipvt1(*),ipvt2(*)

c      Solve   Dr*x = x where Dr:the Red diagonal block of Collocation matrix

      inh=ns*2
      nsi=ns*inh
      call dscal(nsi-2*inh,0.5d0,x(inh+1),1)
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(1),inh,info)

c    Begin  Parallel Loop

!$OMP PARALLEL DO
      do k=1,ns-2,2
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x((k+1)*inh+1),inh,info)
      enddo
!$OMP END PARALLEL DO

c    End  Parallel Loop

      call dscal(inh,-1.0d0,x(nsi-inh+1),1)
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x(nsi-inh+1),inh,info)

      return
      end
*****************************************************************

      subroutine dbsolve(ns,X,M1,ipvt1,M2,ipvt2)
      implicit real*8 (a-h,o-z)

      real*8 X(*),M1(7,*),M2(7,*)
      integer ipvt1(*),ipvt2(*)

c    Solve   Db*x = x where Db:the Black diagonal block of Collocation matrix

      inh=ns*2
      call dscal(ns*inh,0.5d0,x(1),1)

!$OMP PARALLEL DO
      do k=0,ns-1,2
       call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,x(k*inh+1),inh,info)
```

```
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,x((k+1)*inh+1),inh,info)
      enddo
!$OMP END PARALLEL DO

      return
      end
```

Α΄.5 Επίλυση του Schur Complement σε αρχιτεκτονικές πολλαπλών υπολογιστικών πυρήνων με χρήση πολλαπλών GPUs

```
 use openacc
 use omp_lib
 use blackm

 parameter (ns=2048,inh=2*ns,n=4*ns*ns,nsf=ns/2,
+           nsp1=ns+1,n2=n/2,nsi2=n2/2,inh2=2*inh)

 implicit real*8 (a-h,o-z)
 real*8 b(n),x(n),M1(7,inh),M2(7,inh),xb(n2),t(n2),
+  a1(5,inh),a2(5,inh),a3(1:5,1:inh),a4(1:5,1:inh),
+  bm(inh),xe(n),yy(nsp1),r(n2),rh(n2),pi(n2),y(n2),
+  ph(n2),s(n2),sh(n2),resid,temp(inh),w(n2),tt(n2),
+  bb(n2),ui(n2),t1(n2),t12(n2),
+  roip1,roip2,tm0,tm1,tm2,tm

 integer ipvt1(inh),ipvt2(inh),devicecount,iGPU,is,
+        ipiv1(inh),ipiv2(inh)

  devicecount=acc_get_num_devices(acc_device_nvidia)

  print*,'_____________________________________________________'
  print*,'           #GPUs Total :',devicecount
  nsi=int(n2/devicecount)
  call start(ns,a1,a2,a3,a4,M1,M2,ipvt1,ipvt2)
  call makeb(ns,b,bm)
  call redblack(ns,b,x)
  call makeredblack(ns,x,b)
  print*,'_____________________________________________________'
  print*,'           Ns : ',ns

  iGPU=0
  if (iGPU.eq.0) tm0=omp_get_wtime()

   call makebhat(b,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt,iGPU)

  resid=1.0d-10

  call dcopy(n2,b(n2+1),1,bb,1)
  call dcopy(n2,bb,1,xb,1)
  if (iGPU.eq.0) tm1=omp_get_wtime()-tm0
```

```
!$OMP PARALLEL private(iGPU,iq1,iq2,temp,a3,a4,M1,M2,ipiv1,ipiv2,
!$OMP+                 ipvt1,ipvt2,a1,a2)
!$OMP+         shared(iter,istep,error,resid)

       call start(ns,a1,a2,a3,a4,M1,M2,ipvt1,ipvt2)
       iGPU=omp_get_thread_num()
       iq1=nsi*iGPU+1
       iq2=nsi*(iGPU+1)
       call acc_set_device_num(iGPU,acc_device_nvidia)

!$acc data copyin(a3(1:5,1:inh),a4(1:5,1:inh))
!$acc&     create(t12(iq1:iq2))

       call acc_deviceptr(a3,a4)
       if (iGPU.eq.0) tm2=omp_get_wtime()
c---------------------------------------
c   BiCGSTAB without precondition of Ax=b
c---------------------------------------
       iter=5
       dnrmb=dnrm2(n2,bb,1)

!$acc kernels copyin(xb(iq1:iq2)),copyout(t1(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))
      if (iGPU.eq.0) then
!$acc loop vector(32)
        do is=1,inh
         t1(is)=a3(3,is)*xb(is)
        enddo
!$acc loop vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)+a3(2,is)*xb(is)
        enddo
!$acc loop vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)+a3(4,is)*xb(is)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
         t1(is)=t1(is)+a3(1,is+2)*xb(is+2)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)+a3(5,is)*xb(is)
        enddo
!$acc loop vector(32)
        do is=1,inh
         t1(is)=-a4(3,is)*xb(inh+is)+t1(is)
```

```
        enddo
!$acc loop vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)-a4(2,is)*xb(inh+is)
        enddo
!$acc loop vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)-a4(4,is)*xb(inh+is)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
         t1(is)=t1(is)-a4(1,is+2)*xb(inh+is+2)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)-a4(5,is)*xb(inh+is)
        enddo
       endif
c--------------------------------------------------

!$acc loop independent gang private(k1,k2,k)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k2+is)=a3(3,is)*xb(k1+is)+a4(3,is)*xb(k2+is)
        enddo
       enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         t1(k2+is-1)=t1(k2+is-1)+a3(2,is)*xb(k1+is)+a4(2,is)*xb(k2+is)
        enddo
      enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-1
```

```
          t1(k2+is+1)=t1(k2+is+1)+a3(4,is)*xb(k1+is)+a4(4,is)*xb(k2+is)
         enddo
        enddo
!$acc loop independent gang private(k1,k2,k)
       do kp=1,nsf-3,2
        k=kp+iGPU*nsf
        k1=(k-1)*inh
        k2=k*inh
!$acc loop independent vector(32)
         do is=1,inh-2
        t1(k2+is)=t1(k2+is)+a3(1,is+2)*xb(k1+is+2)+a4(1,is+2)*xb(k2+is+2)
         enddo
       enddo
!$acc loop independent gang private(k1,k2,k)
       do kp=1,nsf-3,2
        k=kp+iGPU*nsf
        k1=(k-1)*inh
        k2=k*inh
!$acc loop independent vector(32)
         do is=1,inh-2
          t1(k2+is+2)=t1(k2+is+2)+a3(5,is)*xb(k1+is)+a4(5,is)*xb(k2+is)
         enddo
       enddo
c-------------------------------------------------
!$acc loop independent gang private(k3,k4,k)
       do kp=1,nsf-3,2
        k=kp+iGPU*nsf
        k3=(k+1)*inh
        k4=(k+2)*inh
!$acc loop independent vector(32)
         do is=1,inh
          t1(k3+is)=a3(3,is)*xb(k3+is)-a4(3,is)*xb(k4+is)
         enddo
       enddo
!$acc loop independent gang private(k3,k4,k)
       do kp=1,nsf-3,2
        k=kp+iGPU*nsf
        k3=(k+1)*inh
        k4=(k+2)*inh
!$acc loop independent vector(32)
         do is=2,inh
          t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*xb(k3+is)-a4(2,is)*xb(k4+is)
         enddo
       enddo
!$acc loop independent gang private(k3,k4,k)
       do kp=1,nsf-3,2
        k=kp+iGPU*nsf
```

```
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*xb(k3+is)-a4(4,is)*xb(k4+is)
        enddo
      enddo
!$acc loop independent gang private(k3,k4,k)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
      t1(k3+is)=t1(k3+is)+a3(1,is+2)*xb(k3+is+2)-a4(1,is+2)*xb(k4+is+2)
        enddo
      enddo
!$acc loop independent gang  private(k3,k4,k)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*xb(k3+is)-a4(5,is)*xb(k4+is)
        enddo
      enddo
c-----------------------------------------------------
!$acc loop gang private(k,k3)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t12(k3+is)=t1(k3+is)
       enddo
      enddo
!$acc loop gang private(k,k2,k3)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k3+is)=t1(k3+is)-t1(k2+is)
       enddo
      enddo
```

```
!$acc loop gang  private(k,k2,k3)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k2+is)=t1(k2+is)+t12(k3+is)
       enddo
      enddo

      if (iGPU.eq.1) then
      k3=n2-inh
      k2=n2-2*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k3+is)=a3(3,is)*xb(k2+is)+a4(3,is)*xb(k3+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
         t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*xb(k2+is)+a4(2,is)*xb(k3+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*xb(k2+is)+a4(4,is)*xb(k3+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
       t1(k3+is)=t1(k3+is)+a3(1,is+2)*xb(k2+is+2)+a4(1,is+2)*xb(k3+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*xb(k2+is)+a4(5,is)*xb(k3+is)
        enddo
       endif
!$acc end kernels
c      if (iGPU.eq.0) then
!$OMP SECTIONS
!$OMP SECTION
         call dcopy(inh,xb(nsf*inh+1),1,temp,1)
         call daxpy(inh,1.0d0,xb((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +            1,0.0d0,t1((nsf-1)*inh+1),1)
         call dcopy(inh,xb((nsf-1)*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,xb((nsf+1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,temp,
     +            1,1.0d0,t1((nsf-1)*inh+1),1)
```

```
c        else
!$OMP SECTION
         call dcopy(inh,xb(nsf*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,xb((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +           1,0.0d0,t1(nsf*inh+1),1)
         call dcopy(inh,xb((nsf+1)*inh+1),1,temp,1)
         call daxpy(inh,1.0d0,xb((nsf-1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,temp,
     +           1,1.0d0,t1(nsf*inh+1),1)
c         endif
!$OMP END SECTIONS
c-----------------------------------------
      if (iGPU.eq.0) then
      call dscal(ns*inh-2*inh,0.5d0,t1(inh+1),1)
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(1),inh,info)
      endif
!$OMP DO
      do k=1,ns-2,2
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,t1(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1((k+1)*inh+1),inh,info)
      enddo
!$OMP END  DO
      if (iGPU.eq.1) then
      call dscal(inh,-1.0d0,t1(ns*inh-inh+1),1)
      call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(ns*inh-inh+1),inh,info)
      endif
c-----------------------------------------

c-----------------------------------------------
!$acc kernels copyin(t1(iq1:iq2)),copyout(y(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))
        if (iGPU.eq.0) then
!$acc loop vector(32)
        do is=1,inh
         y(is)=a4(3,is)*t1(is)
        enddo
!$acc loop vector(32)
        do is=2,inh
         y(is-1)=y(is-1)+a4(2,is)*t1(is)
        enddo
!$acc loop vector(32)
        do is=1,inh-1
         y(is+1)=y(is+1)+a4(4,is)*t1(is)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
```

```
         y(is)=y(is)+a4(1,is+2)*t1(is+2)
        enddo
!$acc loop vector(32)
        do is=1,inh-2
         y(is+2)=y(is+2)+a4(5,is)*t1(is)
        enddo

!$acc loop independent vector(32)
        do is=1,inh
         y(inh+is)=-a4(3,is)*t1(inh2+is)+a3(3,is)*t1(inh+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
        y(inh+is-1)=y(inh+is-1)-a4(2,is)*t1(inh2+is)+a3(2,is)*t1(inh+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
        y(inh+1+is)=y(inh+1+is)-a4(4,is)*t1(inh2+is)+a3(4,is)*t1(inh+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         y(inh+is)=y(inh+is)-a4(1,is+2)*t1(inh2+is+2)
     +             +a3(1,is+2)*t1(inh+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
        y(inh+is+2)=y(inh+is+2)-a4(5,is)*t1(inh2+is)+a3(5,is)*t1(inh+is)
        enddo

!$acc loop gang vector(32)
        do is=1,inh
         t12(is)=y(is)
        enddo
!$acc loop gang independent vector(32)
       do is=1,inh
         y(is)=y(inh+is)+y(is)
       enddo

!$acc loop gang independent vector(32)
       do is=1,inh
         y(inh+is)=y(inh+is)-t12(is)
       enddo
      endif
c-------------------------------------------------------------
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
```

```
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh
         y(k1+is)=a3(3,is)*t1(k0+is)+a4(3,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         y(k1+is-1)=y(k1+is-1)+a3(2,is)*t1(k0+is)+a4(2,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         y(k1+is+1)=y(k1+is+1)+a3(4,is)*t1(k0+is)+a4(4,is)*t1(k1+is)
        enddo
       enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k1+is)=y(k1+is)+a3(1,is+2)*t1(k0+is+2)+a4(1,is+2)*t1(k1+is+2)
        enddo
       enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k1+is+2)=y(k1+is+2)+a3(5,is)*t1(k0+is)+a4(5,is)*t1(k1+is)
        enddo
       enddo
```

```
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh
         y(k2+is)=a3(3,is)*t1(k2+is)-a4(3,is)*t1(k3+is)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=2,inh
         y(k2+is-1)=y(k2+is-1)+a3(2,is)*t1(k2+is)-a4(2,is)*t1(k3+is)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         y(k2+is+1)=y(k2+is+1)+a3(4,is)*t1(k2+is)-a4(4,is)*t1(k3+is)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k2+is)=y(k2+is)+a3(1,is+2)*t1(k2+is+2)-a4(1,is+2)*t1(k3+is+2)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
```

```
          y(k2+is+2)=y(k2+is+2)+a3(5,is)*t1(k2+is)-a4(5,is)*t1(k3+is)
         enddo
        enddo
c-----------------------------------------------------------------------
!$acc loop independent gang private(k1,k)
       do kp=2,nsf-4,2
        k=kp+iGPU*nsf
         k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh
          t12(k1+is)=y(k1+is)
        enddo
       enddo
!$acc loop independent private(k1,k2,k)
       do kp=2,nsf-4,2
        k=kp+iGPU*nsf
        k1=k*inh
        k2=(k+1)*inh
!$acc loop independent vector(32)
        do is=1,inh
          y(k1+is)=y(k1+is)+y(k2+is)
        enddo
       enddo
!$acc loop independent private(k1,k2,k)
       do kp=2,nsf-4,2
        k=kp+iGPU*nsf
        k1=k*inh
        k2=(k+1)*inh
!$acc loop independent vector(32)
        do is=1,inh
          y(k2+is)=y(k2+is)-t12(k1+is)
        enddo
       enddo

       if (iGPU.eq.1) then
       k1=n2-inh
       k2=n2-2*inh
       k3=n2-3*inh
!$acc loop independent vector(32)
         do is=1,inh
          y(k1+is)=-a4(3,is)*t1(k1+is)
         enddo
!$acc loop independent vector(32)
         do is=2,inh
          y(k1+is-1)=y(k1+is-1)-a4(2,is)*t1(k1+is)
         enddo
!$acc loop independent vector(32)
```

```
        do is=1,inh-1
         y(k1+is+1)=y(k1+is+1)-a4(4,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k1+is)=y(k1+is)-a4(1,is+2)*t1(k1+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k1+is+2)=y(k1+is+2)-a4(5,is)*t1(k1+is)
        enddo

!$acc loop vector(32)
       do is=1,inh
        t12(k1+is)=y(k1+is)
       enddo

!$acc loop independent vector(32)
        do is=1,inh
         y(k2+is)=a3(3,is)*t1(k3+is)+a4(3,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
         y(k2+is-1)=y(k2+is-1)+a3(2,is)*t1(k3+is)+a4(2,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         y(k2+is+1)=y(k2+is+1)+a3(4,is)*t1(k3+is)+a4(4,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k2+is)=y(k2+is)+a3(1,is+2)*t1(k3+is+2)+a4(1,is+2)*t1(k2+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         y(k2+is+2)=y(k2+is+2)+a3(5,is)*t1(k3+is)+a4(5,is)*t1(k2+is)
        enddo

!$acc loop independent vector(32)
       do is=1,inh
         y(k1+is)=y(k1+is)-y(k2+is)
       enddo

!$acc loop independent vector(32)
       do is=1,inh
         y(k2+is)=t12(k1+is)+y(k2+is)
       enddo
```

```
      endif
!$acc end kernels
c      if (iGPU.eq.0) then
!$OMP SECTIONS
!$OMP SECTION
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,y((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1(nsf*inh+1),
     +          1,1.0d0,y((nsf-1)*inh+1),1)
        call dcopy(inh,y((nsf-1)*inh+1),1,y((nsf-2)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1((nsf-2)*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,y((nsf-2)*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,y((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-3)*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,y((nsf-2)*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,y((nsf-1)*inh+1),1)
c      else
!$OMP SECTION
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf+1)*inh+1),
     +          1,0.0d0,y(nsf*inh+1),1)
        call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1((nsf+2)*inh+1),
     +          1,1.0d0,y(nsf*inh+1),1)
        call dcopy(inh,y(nsf*inh+1),1,y((nsf+1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1(nsf*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,y(nsf*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,y((nsf+1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,y(nsf*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,y((nsf+1)*inh+1),1)
!$OMP END SECTIONS
c      endif

c-----------------------------------------------------
c      call dbsolve(ns,y,M1,ipvt1,M2,ipvt2,iGPU)
       if (iGPU.eq.0) call dscal(ns*inh,0.5d0,y(1),1)
!$OMP DO
      do k=0,ns-1,2
       call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,y(k*inh+1),
     +              inh,info)
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,y((k+1)*inh+1),
     +               inh,info)
      enddo
!$OMP ENDDO
```

```
       if (iGPU.eq.1) then
       call dscal(n2,-1.0d0,y,1)
       call daxpy(n2,1.0d0,xb,1,y,1)
       call dcopy(n2,y,1,r,1)
c-------------------------------------
       call dscal(n2,-1.0d0,r,1)
       call daxpy(n2,1.0d0,bb,1,r,1)
       dnrm=dnrm2(n2,r,1)
       call dcopy(n2,r,1,rh,1)
       call dcopy(n2,r,1,pi,1)
       roip1=ddot(n2,rh,1,r,1)
      endif
!$OMP BARRIER
        istep=0
999   continue
!$OMP BARRIER
       if (iGPU.eq.0) istep=istep+1
       if (roip1.eq.0.0d0) then
         print*,' BiCGSTAB Fails - Pi-1=0 '
       stop
       endif

!$acc kernels copyin(pi(iq1:iq2)),copyout(t1(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))
      if (iGPU.eq.0) then
!$acc loop gang vector(32)
        do is=1,inh
         t1(is)=a3(3,is)*pi(is)
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)+a3(2,is)*pi(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)+a3(4,is)*pi(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is)=t1(is)+a3(1,is+2)*pi(is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)+a3(5,is)*pi(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh
```

```
         t1(is)=-a4(3,is)*pi(inh+is)+t1(is)
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)-a4(2,is)*pi(inh+is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)-a4(4,is)*pi(inh+is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is)=t1(is)-a4(1,is+2)*pi(inh+is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)-a4(5,is)*pi(inh+is)
        enddo
        endif
! acc loop gang private(k1,k2,k3,k4,k) independent
c     do kp=1,nsf-3,2
c      k=kp+iGPU*nsf
c      k1=(k-1)*inh
c      k2=k*inh
c      k3=(k+1)*inh
c      k4=(k+2)*inh


!$acc loop independent gang private(k,k1,k2)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k2+is)=a3(3,is)*pi(k1+is)+a4(3,is)*pi(k2+is)
        enddo
       enddo
!$acc loop independent gang private(k,k1,k2)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         t1(k2+is-1)=t1(k2+is-1)+a3(2,is)*pi(k1+is)+a4(2,is)*pi(k2+is)
        enddo
```

```
      enddo
!$acc loop independent gang private(k,k1,k2)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independentvector(32)
        do is=1,inh-1
         t1(k2+is+1)=t1(k2+is+1)+a3(4,is)*pi(k1+is)+a4(4,is)*pi(k2+is)
        enddo
      enddo
!$acc loop independent gang private(k,k1,k2)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       t1(k2+is)=t1(k2+is)+a3(1,is+2)*pi(k1+is+2)+a4(1,is+2)*pi(k2+is+2)
        enddo
      enddo
!$acc loop independent gang private(k,k1,k2)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k2+is+2)=t1(k2+is+2)+a3(5,is)*pi(k1+is)+a4(5,is)*pi(k2+is)
        enddo
      enddo
c-----------------------------------------------------------------------
!$acc loop independent gang private(k,k3,k4)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k3+is)=a3(3,is)*pi(k3+is)-a4(3,is)*pi(k4+is)
        enddo
      enddo
!$acc loop independent gang private(k,k3,k4)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
```

```
!$acc loop independent vector(32)
        do is=2,inh
         t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*pi(k3+is)-a4(2,is)*pi(k4+is)
        enddo
      enddo
!$acc loop independent gang private(k,k3,k4)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*pi(k3+is)-a4(4,is)*pi(k4+is)
        enddo
      enddo
!$acc loop independent gang private(k,k3,k4)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
      t1(k3+is)=t1(k3+is)+a3(1,is+2)*pi(k3+is+2)-a4(1,is+2)*pi(k4+is+2)
        enddo
      enddo
!$acc loop independent gang private(k,k3,k4)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*pi(k3+is)-a4(5,is)*pi(k4+is)
        enddo
      enddo
c*************************************************
!$acc loop gang private(k3,k) independent
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t12(k3+is)=t1(k3+is)
       enddo
      enddo
!$acc loop independent gang private(k,k2,k3)
      do kp=1,nsf-3,2
```

```
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k3+is)=t1(k3+is)-t1(k2+is)
       enddo
      enddo
!$acc loop independent gang private(k,k2,k3)
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k2+is)=t1(k2+is)+t12(k3+is)
       enddo
      enddo
      if (iGPU.eq.1) then
      k3=n2-inh
      k2=n2-2*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k3+is)=a3(3,is)*pi(k2+is)+a4(3,is)*pi(k3+is)
        enddo
!$acc loop gang independent vector(32)
        do is=2,inh
         t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*pi(k2+is)+a4(2,is)*pi(k3+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*pi(k2+is)+a4(4,is)*pi(k3+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
       t1(k3+is)=t1(k3+is)+a3(1,is+2)*pi(k2+is+2)+a4(1,is+2)*pi(k3+is+2)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*pi(k2+is)+a4(5,is)*pi(k3+is)
        enddo
       endif
!$acc end kernels
c      if (iGPU.eq.0) then
!$OMP SECTIONS
!$OMP SECTION
         call dcopy(inh,pi(nsf*inh+1),1,temp,1)
```

```
         call daxpy(inh,1.0d0,pi((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +           1,0.0d0,t1((nsf-1)*inh+1),1)
         call dcopy(inh,pi((nsf-1)*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,pi((nsf+1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,temp,
     +           1,1.0d0,t1((nsf-1)*inh+1),1)
c        else
!$OMP SECTION
         call dcopy(inh,pi(nsf*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,pi((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +           1,0.0d0,t1(nsf*inh+1),1)
         call dcopy(inh,pi((nsf+1)*inh+1),1,temp,1)
         call daxpy(inh,1.0d0,pi((nsf-1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,temp,
     +           1,1.0d0,t1(nsf*inh+1),1)
c      endif
!$OMP END SECTIONS

c-----------------------------------------
c      call drsolve(ns,t1,M1,ipvt1,M2,ipvt2,iGPU)
       if (iGPU.eq.0) then
        call dscal(ns*inh-2*inh,0.5d0,t1(inh+1),1)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(1),inh,info)
       endif
!$OMP BARRIER
!$OMP DO
       do k=1,ns-2,2
         call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,t1(k*inh+1),inh,info)
         call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1((k+1)*inh+1),inh,info)
       enddo
C$OMP END  DO
       if (iGPU.eq.1) then
        call dscal(inh,-1.0d0,t1(ns*inh-inh+1),1)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(ns*inh-inh+1),inh,info)
       endif
!$OMP BARRIER
c-----------------------------------------
c-----------------------------------------
!$acc kernels copyin(t1(iq1:iq2)), copyout(ui(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))

        if (iGPU.eq.0) then
!$acc loop gang vector(32)
        do is=1,inh
         ui(is)=a4(3,is)*t1(is)
```

```
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         ui(is-1)=ui(is-1)+a4(2,is)*t1(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-1
         ui(is+1)=ui(is+1)+a4(4,is)*t1(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         ui(is)=ui(is)+a4(1,is+2)*t1(is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         ui(is+2)=ui(is+2)+a4(5,is)*t1(is)
        enddo

!$acc loop gang independent vector(32)
        do is=1,inh
         ui(inh+is)=-a4(3,is)*t1(inh2+is)+a3(3,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=2,inh
      ui(inh+is-1)=ui(inh+is-1)-a4(2,is)*t1(inh2+is)+a3(2,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-1
      ui(inh+1+is)=ui(inh+1+is)-a4(4,is)*t1(inh2+is)+a3(4,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
         ui(inh+is)=ui(inh+is)-a4(1,is+2)*t1(inh2+is+2)
     +             +a3(1,is+2)*t1(inh+is+2)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
      ui(inh+is+2)=ui(inh+is+2)-a4(5,is)*t1(inh2+is)+a3(5,is)*t1(inh+is)
        enddo

!$acc loop gang vector(32)
        do is=1,inh
         t12(is)=ui(is)
        enddo
!$acc loop gang independent vector(32)
       do is=1,inh
         ui(is)=ui(inh+is)+ui(is)
```

```
       enddo

!$acc loop gang independent vector(32)
       do is=1,inh
         ui(inh+is)=ui(inh+is)-t12(is)
       enddo
      endif
c-----------------------------------------------------------------------
! acc loop gang private(k0,k1,k2,k3,k) independent
c     do kp=2,nsf-4,2
c     k=kp+iGPU*nsf
c     k0=(k-1)*inh
c     k1=k*inh
c     k2=(k+1)*inh
c     k3=(k+2)*inh

!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh
         ui(k1+is)=a3(3,is)*t1(k0+is)+a4(3,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         ui(k1+is-1)=ui(k1+is-1)+a3(2,is)*t1(k0+is)+a4(2,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         ui(k1+is+1)=ui(k1+is+1)+a3(4,is)*t1(k0+is)+a4(4,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
```

```
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       ui(k1+is)=ui(k1+is)+a3(1,is+2)*t1(k0+is+2)+a4(1,is+2)*t1(k1+is+2)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
        ui(k1+is+2)=ui(k1+is+2)+a3(5,is)*t1(k0+is)+a4(5,is)*t1(k1+is)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh
         ui(k2+is)=a3(3,is)*t1(k2+is)-a4(3,is)*t1(k3+is)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=2,inh
         ui(k2+is-1)=ui(k2+is-1)+a3(2,is)*t1(k2+is)-a4(2,is)*t1(k3+is)
        enddo
       enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         ui(k2+is+1)=ui(k2+is+1)+a3(4,is)*t1(k2+is)-a4(4,is)*t1(k3+is)
```

```
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       ui(k2+is)=ui(k2+is)+a3(1,is+2)*t1(k2+is+2)-a4(1,is+2)*t1(k3+is+2)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         ui(k2+is+2)=ui(k2+is+2)+a3(5,is)*t1(k2+is)-a4(5,is)*t1(k3+is)
        enddo
       enddo
c-------------------------------------------------------------------------
!$acc loop independent gang private(k1,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
        k1=k*inh
!$acc loop independent vector(32)
       do is=1,inh
         t12(k1+is)=ui(k1+is)
       enddo
      enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         ui(k1+is)=ui(k1+is)+ui(k2+is)
       enddo
      enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
       k1=k*inh
       k2=(k+1)*inh
```

```
!$acc loop independent vector(32)
       do is=1,inh
         ui(k2+is)=ui(k2+is)-t12(k1+is)
       enddo
      enddo

      if (iGPU.eq.1) then
      k1=n2-inh
      k2=n2-2*inh
      k3=n2-3*inh
!$acc loop independent vector(32)
        do is=1,inh
         ui(k1+is)=-a4(3,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
         ui(k1+is-1)=ui(k1+is-1)-a4(2,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         ui(k1+is+1)=ui(k1+is+1)-a4(4,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         ui(k1+is)=ui(k1+is)-a4(1,is+2)*t1(k1+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         ui(k1+is+2)=ui(k1+is+2)-a4(5,is)*t1(k1+is)
        enddo

!$acc loop vector(32)
       do is=1,inh
        t12(k1+is)=ui(k1+is)
       enddo

!$acc loop independent vector(32)
        do is=1,inh
         ui(k2+is)=a3(3,is)*t1(k3+is)+a4(3,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
         ui(k2+is-1)=ui(k2+is-1)+a3(2,is)*t1(k3+is)+a4(2,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         ui(k2+is+1)=ui(k2+is+1)+a3(4,is)*t1(k3+is)+a4(4,is)*t1(k2+is)
```

```
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
       ui(k2+is)=ui(k2+is)+a3(1,is+2)*t1(k3+is+2)+a4(1,is+2)*t1(k2+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         ui(k2+is+2)=ui(k2+is+2)+a3(5,is)*t1(k3+is)+a4(5,is)*t1(k2+is)
        enddo

!$acc loop independent vector(32)
       do is=1,inh
         ui(k1+is)=ui(k1+is)-ui(k2+is)
       enddo

!$acc loop independent vector(32)
       do is=1,inh
         ui(k2+is)=t12(k1+is)+ui(k2+is)
       enddo
      endif
!$acc end kernels
!$OMP SECTIONS
!$OMP SECTION
c      if (iGPU.eq.0) then
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,ui((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1(nsf*inh+1),
     +          1,1.0d0,ui((nsf-1)*inh+1),1)
        call dcopy(inh,ui((nsf-1)*inh+1),1,ui((nsf-2)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1((nsf-2)*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,ui((nsf-2)*inh+1),1)
       call daxpy(inh,-1.0d0,temp,1,ui((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-3)*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,ui((nsf-2)*inh+1),1)
       call daxpy(inh,-1.0d0,temp,1,ui((nsf-1)*inh+1),1)
!$OMP SECTION
c      else
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf+1)*inh+1),
     +          1,0.0d0,ui(nsf*inh+1),1)
        call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1((nsf+2)*inh+1),
     +          1,1.0d0,ui(nsf*inh+1),1)
        call dcopy(inh,ui(nsf*inh+1),1,ui((nsf+1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1(nsf*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,ui(nsf*inh+1),1)
```

```
       call daxpy(inh,-1.0d0,temp,1,ui((nsf+1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,ui(nsf*inh+1),1)
       call daxpy(inh,-1.0d0,temp,1,ui((nsf+1)*inh+1),1)

!$OMP END SECTIONS
c      endif
! OMP BARRIER
c-------------------------------------------------
c      call dbsolve(ns,ui,M1,ipvt1,M2,ipvt2,iGPU)
      if (iGPU.eq.0) call dscal(ns*inh,0.5d0,ui(1),1)
!$OMP BARRIER
!$OMP DO
      do k=0,ns-1,2
       call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,ui(k*inh+1),
     +              inh,info)
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,ui((k+1)*inh+1),
     +               inh,info)
      enddo
!$OMP END DO
c-------------------------------------------------
      if (iGPU.eq.0) then
        call dscal(n2,-1.0d0,ui,1)
        call daxpy(n2,1.0d0,pi,1,ui,1)
        ai=roip1/ddot(n2,rh,1,ui,1)
        call dcopy(n2,r,1,s,1)
        call daxpy(n2,-ai,ui,1,s,1)
      endif
!$OMP BARRIER
         if (dnrm2(n2,s,1).lt.1.0d-19) then
          call daxpy(n2,ai,ph,1,xb,1)
         print*,' ||s||2 is small enough after ',istep,' steps'
         else
c         call bmatvec(s,t,ns,M1,M2,a3,a4,ipvt1,ipvt2,w,tt,ph)

!$acc kernels copyin(s(iq1:iq2)),copyout(t1(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))
      if (iGPU.eq.0) then
!$acc loop gang vector(32)
        do is=1,inh
         t1(is)=a3(3,is)*s(is)
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)+a3(2,is)*s(is)
        enddo
```

```
!$acc loop gang vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)+a3(4,is)*s(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is)=t1(is)+a3(1,is+2)*s(is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)+a3(5,is)*s(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh
         t1(is)=-a4(3,is)*s(inh+is)+t1(is)
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         t1(is-1)=t1(is-1)-a4(2,is)*s(inh+is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-1
         t1(is+1)=t1(is+1)-a4(4,is)*s(inh+is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is)=t1(is)-a4(1,is+2)*s(inh+is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t1(is+2)=t1(is+2)-a4(5,is)*s(inh+is)
        enddo
        endif
! acc loop  private(k1,k2,k3,k4,k) independent
c     do kp=1,nsf-3,2
c      k=kp+iGPU*nsf
c      k1=(k-1)*inh
c      k2=k*inh
c      k3=(k+1)*inh
c      k4=(k+2)*inh

!$acc loop private(k1,k2,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
```

```
        do is=1,inh
         t1(k2+is)=a3(3,is)*s(k1+is)+a4(3,is)*s(k2+is)
        enddo
      enddo
!$acc loop private(k1,k2,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         t1(k2+is-1)=t1(k2+is-1)+a3(2,is)*s(k1+is)+a4(2,is)*s(k2+is)
        enddo
      enddo
!$acc loop private(k1,k2,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         t1(k2+is+1)=t1(k2+is+1)+a3(4,is)*s(k1+is)+a4(4,is)*s(k2+is)
        enddo
      enddo
!$acc loop private(k1,k2,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       t1(k2+is)=t1(k2+is)+a3(1,is+2)*s(k1+is+2)+a4(1,is+2)*s(k2+is+2)
        enddo
      enddo
!$acc loop private(k1,k2,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k1=(k-1)*inh
       k2=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k2+is+2)=t1(k2+is+2)+a3(5,is)*s(k1+is)+a4(5,is)*s(k2+is)
        enddo
      enddo
c-------------------------------------------------------
!$acc loop private(k3,k4,k) independent gang
      do kp=1,nsf-3,2
```

```
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k3+is)=a3(3,is)*s(k3+is)-a4(3,is)*s(k4+is)
        enddo
      enddo
!$acc loop private(k3,k4,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=2,inh
         t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*s(k3+is)-a4(2,is)*s(k4+is)
        enddo
      enddo
!$acc loop private(k3,k4,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*s(k3+is)-a4(4,is)*s(k4+is)
        enddo
      enddo
!$acc loop private(k3,k4,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
      t1(k3+is)=t1(k3+is)+a3(1,is+2)*s(k3+is+2)-a4(1,is+2)*s(k4+is+2)
        enddo
       enddo
!$acc loop private(k3,k4,k) independent gang
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
       k3=(k+1)*inh
       k4=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*s(k3+is)-a4(5,is)*s(k4+is)
        enddo
```

```
      enddo
c*************************************************
!$acc loop gang private(k3,k) independent
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t12(k3+is)=t1(k3+is)
       enddo
       enddo
!$acc loop gang private(k2,k3,k) independent
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k3+is)=t1(k3+is)-t1(k2+is)
       enddo
      enddo
!$acc loop gang private(k2,k3,k) independent
      do kp=1,nsf-3,2
       k=kp+iGPU*nsf
         k2=k*inh
         k3=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t1(k2+is)=t1(k2+is)+t12(k3+is)
       enddo
      enddo

      if (iGPU.eq.1) then
      k3=n2-inh
      k2=n2-2*inh
!$acc loop independent vector(32)
        do is=1,inh
         t1(k3+is)=a3(3,is)*s(k2+is)+a4(3,is)*s(k3+is)
        enddo
!$acc loop gang independent vector(32)
        do is=2,inh
         t1(k3+is-1)=t1(k3+is-1)+a3(2,is)*s(k2+is)+a4(2,is)*s(k3+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-1
         t1(k3+is+1)=t1(k3+is+1)+a3(4,is)*s(k2+is)+a4(4,is)*s(k3+is)
        enddo
```

```
!$acc loop gang independent vector(32)
        do is=1,inh-2
       t1(k3+is)=t1(k3+is)+a3(1,is+2)*s(k2+is+2)+a4(1,is+2)*s(k3+is+2)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
         t1(k3+is+2)=t1(k3+is+2)+a3(5,is)*s(k2+is)+a4(5,is)*s(k3+is)
        enddo
       endif
!$acc end kernels

!$OMP SECTIONS
!$OMP SECTION
         call dcopy(inh,s(nsf*inh+1),1,temp,1)
         call daxpy(inh,1.0d0,s((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +           1,0.0d0,t1((nsf-1)*inh+1),1)
         call dcopy(inh,s((nsf-1)*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,s((nsf+1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,temp,
     +           1,1.0d0,t1((nsf-1)*inh+1),1)

!$OMP SECTION
         call dcopy(inh,s(nsf*inh+1),1,temp,1)
         call daxpy(inh,-1.0d0,s((nsf-2)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,temp,
     +           1,0.0d0,t1(nsf*inh+1),1)
         call dcopy(inh,s((nsf+1)*inh+1),1,temp,1)
         call daxpy(inh,1.0d0,s((nsf-1)*inh+1),1,temp,1)
         call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,temp,
     +           1,1.0d0,t1(nsf*inh+1),1)

!$OMP END SECTIONS

!$OMP BARRIER
c-----------------------------------------
      if (iGPU.eq.0) then
       call dscal(ns*inh-2*inh,0.5d0,t1(inh+1),1)
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(1),inh,info)
      endif
!$OMP BARRIER
!$OMP DO
      do k=1,ns-2,2
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,t1(k*inh+1),inh,info)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1((k+1)*inh+1),inh,info)
      enddo
!$OMP ENDDO
```

```
      if (iGPU.eq.1) then
       call dscal(inh,-1.0d0,t1(ns*inh-inh+1),1)
       call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t1(ns*inh-inh+1),inh,info)
      endif
!$OMP BARRIER
c---------------------------------------------
c---------------------------------------------
!$acc kernels copyin(t1(iq1:iq2)), copyout(t(iq1:iq2)),
!$acc&        present(a3(1:5,1:inh),a4(1:5,1:inh),t12(iq1:iq2))

        if (iGPU.eq.0) then
!$acc loop gang vector(32)
        do is=1,inh
         t(is)=a4(3,is)*t1(is)
        enddo
!$acc loop gang vector(32)
        do is=2,inh
         t(is-1)=t(is-1)+a4(2,is)*t1(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-1
         t(is+1)=t(is+1)+a4(4,is)*t1(is)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t(is)=t(is)+a4(1,is+2)*t1(is+2)
        enddo
!$acc loop gang vector(32)
        do is=1,inh-2
         t(is+2)=t(is+2)+a4(5,is)*t1(is)
        enddo

!$acc loop gang independent vector(32)
        do is=1,inh
         t(inh+is)=-a4(3,is)*t1(inh2+is)+a3(3,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=2,inh
      t(inh+is-1)=t(inh+is-1)-a4(2,is)*t1(inh2+is)+a3(2,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-1
      t(inh+1+is)=t(inh+1+is)-a4(4,is)*t1(inh2+is)+a3(4,is)*t1(inh+is)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
         t(inh+is)=t(inh+is)-a4(1,is+2)*t1(inh2+is+2)
```

```
     +             +a3(1,is+2)*t1(inh+is+2)
        enddo
!$acc loop gang independent vector(32)
        do is=1,inh-2
      t(inh+is+2)=t(inh+is+2)-a4(5,is)*t1(inh2+is)+a3(5,is)*t1(inh+is)
        enddo

!$acc loop gang vector(32)
        do is=1,inh
         t12(is)=t(is)
        enddo
!$acc loop gang independent vector(32)
       do is=1,inh
         t(is)=t(inh+is)+t(is)
       enddo

!$acc loop gang independent vector(32)
       do is=1,inh
         t(inh+is)=t(inh+is)-t12(is)
       enddo
       endif
c-------------------------------------------------------
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh
         t(k1+is)=a3(3,is)*t1(k0+is)+a4(3,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=2,inh
         t(k1+is-1)=t(k1+is-1)+a3(2,is)*t1(k0+is)+a4(2,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
```

```
!$acc loop independent vector(32)
        do is=1,inh-1
         t(k1+is+1)=t(k1+is+1)+a3(4,is)*t1(k0+is)+a4(4,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       t(k1+is)=t(k1+is)+a3(1,is+2)*t1(k0+is+2)+a4(1,is+2)*t1(k1+is+2)
        enddo
      enddo
!$acc loop independent gang private(k0,k1,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k0=(k-1)*inh
      k1=k*inh
!$acc loop independent vector(32)
        do is=1,inh-2
        t(k1+is+2)=t(k1+is+2)+a3(5,is)*t1(k0+is)+a4(5,is)*t1(k1+is)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh
         t(k2+is)=a3(3,is)*t1(k2+is)-a4(3,is)*t1(k3+is)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=2,inh
         t(k2+is-1)=t(k2+is-1)+a3(2,is)*t1(k2+is)-a4(2,is)*t1(k3+is)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
```

```
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-1
         t(k2+is+1)=t(k2+is+1)+a3(4,is)*t1(k2+is)-a4(4,is)*t1(k3+is)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
       t(k2+is)=t(k2+is)+a3(1,is+2)*t1(k2+is+2)-a4(1,is+2)*t1(k3+is+2)
        enddo
      enddo
!$acc loop independent gang private(k2,k3,k)
      do kp=2,nsf-4,2
      k=kp+iGPU*nsf
      k2=(k+1)*inh
      k3=(k+2)*inh
!$acc loop independent vector(32)
        do is=1,inh-2
         t(k2+is+2)=t(k2+is+2)+a3(5,is)*t1(k2+is)-a4(5,is)*t1(k3+is)
        enddo
       enddo
c-----------------------------------------------------------------------
!$acc loop independent gang private(k1,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
        k1=k*inh
!$acc loop independent vector(32)
       do is=1,inh
         t12(k1+is)=t(k1+is)
       enddo
      enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t(k1+is)=t(k1+is)+t(k2+is)
       enddo
```

```
      enddo
!$acc loop independent gang private(k1,k2,k)
      do kp=2,nsf-4,2
       k=kp+iGPU*nsf
       k1=k*inh
       k2=(k+1)*inh
!$acc loop independent vector(32)
       do is=1,inh
         t(k2+is)=t(k2+is)-t12(k1+is)
       enddo
      enddo

      if (iGPU.eq.1) then
      k1=n2-inh
      k2=n2-2*inh
      k3=n2-3*inh
!$acc loop independent vector(32)
        do is=1,inh
         t(k1+is)=-a4(3,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=2,inh
         t(k1+is-1)=t(k1+is-1)-a4(2,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         t(k1+is+1)=t(k1+is+1)-a4(4,is)*t1(k1+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         t(k1+is)=t(k1+is)-a4(1,is+2)*t1(k1+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         t(k1+is+2)=t(k1+is+2)-a4(5,is)*t1(k1+is)
        enddo

!$acc loop vector(32)
       do is=1,inh
        t12(k1+is)=t(k1+is)
       enddo

!$acc loop independent vector(32)
        do is=1,inh
         t(k2+is)=a3(3,is)*t1(k3+is)+a4(3,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
```

```
        do is=2,inh
         t(k2+is-1)=t(k2+is-1)+a3(2,is)*t1(k3+is)+a4(2,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-1
         t(k2+is+1)=t(k2+is+1)+a3(4,is)*t1(k3+is)+a4(4,is)*t1(k2+is)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
       t(k2+is)=t(k2+is)+a3(1,is+2)*t1(k3+is+2)+a4(1,is+2)*t1(k2+is+2)
        enddo
!$acc loop independent vector(32)
        do is=1,inh-2
         t(k2+is+2)=t(k2+is+2)+a3(5,is)*t1(k3+is)+a4(5,is)*t1(k2+is)
        enddo

!$acc loop independent vector(32)
       do is=1,inh
         t(k1+is)=t(k1+is)-t(k2+is)
       enddo

!$acc loop independent vector(32)
       do is=1,inh
         t(k2+is)=t12(k1+is)+t(k2+is)
       enddo
      endif
!$acc end kernels
c      if (iGPU.eq.0) then
!$OMP SECTIONS
!$OMP SECTION
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,t((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1(nsf*inh+1),
     +          1,1.0d0,t((nsf-1)*inh+1),1)
        call dcopy(inh,t((nsf-1)*inh+1),1,t((nsf-2)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1((nsf-2)*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,t((nsf-2)*inh+1),1)
       call daxpy(inh,-1.0d0,temp,1,t((nsf-1)*inh+1),1)
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-3)*inh+1),
     +          1,0.0d0,temp,1)
       call daxpy(inh,1.0d0,temp,1,t((nsf-2)*inh+1),1)
       call daxpy(inh,-1.0d0,temp,1,t((nsf-1)*inh+1),1)
c      else
!$OMP SECTION
        call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf+1)*inh+1),
     +          1,0.0d0,t(nsf*inh+1),1)
```

```
         call dgbmv('n',inh,inh,2,2,-1.0d0,a4,5,t1((nsf+2)*inh+1),
     +          1,1.0d0,t(nsf*inh+1),1)
         call dcopy(inh,t(nsf*inh+1),1,t((nsf+1)*inh+1),1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a4,5,t1(nsf*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,t(nsf*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,t((nsf+1)*inh+1),1)
         call dgbmv('n',inh,inh,2,2,1.0d0,a3,5,t1((nsf-1)*inh+1),
     +          1,0.0d0,temp,1)
        call daxpy(inh,1.0d0,temp,1,t(nsf*inh+1),1)
        call daxpy(inh,-1.0d0,temp,1,t((nsf+1)*inh+1),1)
!$OMP END SECTIONS
c      endif

c-------------------------------------------------
c      call dbsolve(ns,t,M1,ipvt1,M2,ipvt2,iGPU)
      if (iGPU.eq.0) call dscal(ns*inh,0.5d0,t(1),1)
!$OMP BARRIER
!$OMP DO
      do k=0,ns-1,2
        call dgbtrs('N',inh,2,2,1,M1,7,ipvt1,t(k*inh+1),
     +              inh,info)
        call dgbtrs('N',inh,2,2,1,M2,7,ipvt2,t((k+1)*inh+1),
     +               inh,info)
      enddo
!$OMP ENDDO

       if(iGPU.eq.0) then
        call dscal(n2,-1.0d0,t,1)
        call daxpy(n2,1.0d0,s,1,t,1)
         wi=ddot(n2,t,1,s,1)/ddot(n2,t,1,t,1)
c       print*,dnrm2(n2,t(1),1),'T ',wi
         call daxpy(n2,ai,pi,1,xb,1)
         call daxpy(n2,wi,s,1,xb,1)
         call daxpy(n2,-wi,t,1,s,1)
         call dcopy(n2,s,1,r,1)
         dnrmr=dnrm2(n2,r,1)
         roip2=roip1
         roip1=ddot(n2,rh,1,r,1)
         bi=(roip1/roip2)*(ai/wi)
         call daxpy(n2,-wi,ui,1,pi,1)
         call dcopy(n2,r,1,t,1)
         call daxpy(n2,bi,pi,1,t,1)
         call dcopy(n2,t,1,pi,1)
        endif
!$OMP BARRIER
         error=dnrmr/dnrmb
```

```
         if (wi.ne.0.0d0.and.error.gt.resid.and.istep.lt.iter)
     +   goto 999
         print*,' BiCGSTAB exit after ',istep,'  steps.'
         endif
        if (iGPU.eq.0) iter=istep
!$acc end data
!$OMP END PARALLEL


        if(iGPU.eq.0) then
        call dcopy(n2,b,1,x,1)
        call dcopy(n2,xb,1,x(n2+1),1)
        call makexhat(x,ns,w,M1,ipvt1,M2,ipvt2,a3,a4,temp,tt,iGPU)
        tm=omp_get_wtime()-tm2

        print*,'Total Time  =',tm+tm1
        print*,'Iterations = ',iter
        print*,'||x||2=', dnrm2(n,x,1)


        call exact(ns,b,yy)
        call redblack(ns,b,xe)
        call matvec(x,xe,ns,a1,a2,a3,a4,temp)

        call makeb(ns,b,bm)
        call redblack(ns,b,x)
        call makeredblack(ns,x,b)
        call daxpy(n,-1.0d0,xe,1,b,1)
        print*,'||b - Ax||2 =',dnrm2(n,b,1)
        print*,'_____________________________________________________'
        endif
        stop
       end
```

Βιβλιογραφία

[1] H. Anzt, P. Luszczek, J. Dongarra and V. Heuveline, "GPU-Accelerated Asynchronous Error Correction for Mixed Precision Iterative Refinement", Technical Report, URL: http://link.springer.com/chapter/10.1007%2F978-3-642-32820-6_89

[2] O. Axelsson and A. Barker, "Finite element solution of boundary value problems". *Theory and computation*, Academic Press, Orlando, Fl., 1984.

[3] J. Beyer, "The use of OpenACC and OpenMP Accelerator directives with the CCE", Technical Paper, 2012,
URL: http://www.openacc.org/sites/default/files/S3084_jamescbeyer_0.pdf

[4] G. Birkiff, M. H. Schultz and R. S. Varga, "Piecewise Hermite in one and two Variables with Applications to Partial Differential Equations", *Numer. Math.* 11, 232-256, 1968.

[5] C. de Boor and Swartz, "Collocation at Gaussian Points", *SIAM J. Numer. Anal.* 10, 582-606, 1973.

[6] S.H.Brill and G.F.Pinder, "Parallel implementation of the bi-cgstab method with block red-black gauss-seidel preconditioner applied to the hermite collocation discretization of partial differetial equations", *Parallel Computing*, vol. 28, pp. 399–414, 2002.

[7] I. Buck, "The Evolution of GPUs for General Purpose Computing", San Jose Convention Center, 2010, URL: http://www.nvidia.com

[8] S. Cook, "CUDA Programming: A Developer's Guide to Parallel Computing with GPUs", 2013, Elsevier Inc.

[9] J. Douglas and T. Dupont, "Collocation Methods for Parabolic Equations in a Single Space Variable", *Springer-Verlag Lecture Notes 385*, Berlin/New York,1974.

[10] D. Goddeke, R. Strzodka, and S. Turek: Accelerating double precision FEM simulations with GPUs. In 18th Symposium on Simulation Technique, ASIM, Erlangen, September 12-15 2005.

[11] A. Hadjidimos, T.S. Papatheodorou and Y. G. Saridakis, "Optimal Block Iterative Schemes for Certain Large Sparse and Non-symmetric Linear Systems", *Linear Algebra Appl.*, 110, 285-318, 1988.

[12] L. A. Hageman and D. M. Young, " Applied Iterative Methods", *Academic Press*, New York, 1981.

[13] E. N. Houstis, "Application of the method of Collocation on Lines for Solving Nonlinear Hyperbolic Problems",*Math. Comp.* 31, 443 - 456, 1977.

[14] E. N. Houstis, W. Mitchell, J. R. Rice, "Collocation Software for Second Order Elliptic Partial Differential Equations", *ACM Trans. Math. Software* 11, 379-412, 1985.

[15] *http://computing.llnl.gov/tutorials/openMP/*

[16] *http://developer.nvidia.com/cublas*

[17] *http://developer.nvidia.com/cusparse*

[18] *http://developer.nvidia.com/openacc*

[19] *http://icl.utk.edu/magma/*

[20] *http://www.cs.virginia.edu/ mwb7w/cuda_support/pinned_tradeoff.html*

[21] *http://www.culatools.com/*

[22] *http://www.gpgpu.org*

[23] *http://www.intel.com*

[24] *http://www.karlrupp.net*

[25] *http://www.khronos.org/opencl/*

[26] *http://www.mcs.anl.gov/research/projects/mpi/*

[27] *http://www.mpi-forum.org*

[28] *http://www.netlib.org/blas/*

[29] *www.netlib.org/lapack/*

[30] *http://www.nvidia.com/cuda*

[31] *http://www.nvidia.com/object/tesla-supercomputing-solutions.html*

[32] *http://www.nvidia.com/docs/IO/123576/nv-applications-catalog-lowres.pdf*

[33] *http://www.openacc-standard.org*

[34] *http://www.openmp.org*

[35] *http://www.open-mpi.org*

[36] *http://www.pgroup.com*

[37] *http://www.pgroup.com/lit/articles/insider/v4n2a1.htm*

[38] *http://www.top500.org*

[39] *http://www.roguewave.com/products/imsl-numerical-libraries/fortran-library.aspx*

[40] D. Kirk and W. Hwu, "Programming Massivelly Parallel Processors", Morgan Kaufmann, 2010.

[41] Yu-Lin Lai, A. Hadjidimos, E. N. Houstis and J. R. Rice, "On the iterative solution of Hermite Collocation Equations", *SIAM J Mat. Analysics*, vol. 16 (1), pp. 254-277, 1995.

[42] A. Logg, K. A. Mardal, G. N. Wells, "Automated Solution of Differential Equations by the Finite Element Method", Springer, 2012

[43] V. G. Mandikas, " Network Scientific Computations Using Multigrid Technique For The Hermite Collocation Numerical Method", MSc Thesis, Technical University of Crete, URL: http://www.library.tuc.gr/artemis/MT2009-0023/MT2009-0023.pdf, 2008.

[44] E. N. Mathioudakis, N. D. Vilanakis, E. P. Papadopoulou and Y. G. Saridakis, " Parallel iterative solution of the Hermite Collocation equations on GPUs ", Proc. of the World Congress on Engineering 2013 (WCE2013, Imperial College - London, U.K.), *Best Paper Award of The 2013 International Conference of Parallel and Distributed Computing*, vol 2, pp. 1281-1286, URL: http://www.iaeng.org/publication/WCE2013/WCE2013_pp1281-1286.pdf

[45] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, " Mapping Parallel Iterative Algorithms for PDE Computations on a Distributed Memory Computers", *Parallel Algorithms and Applications* , 8, pp. 141-154,1996.

[46] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, "Bi-CGSTAB for collocation equations on distributed memory parallel computers", *Numerical Mathematics and advanced applications - ENUMATH 2001*, pp. 957-966, 2003.

[47] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, "Iterative Solution of Elliptic Collocation Systems on a Cognitive Parallel Computer", *Computers and Mathematics with applications*, vol. 48, pp. 951-970, 2004.

[48] E. N. Mathioudakis, E. P. Papadopoulou, and Y. G. Saridakis, "Preconditioning for solving hermite collocation by the bi-cgstab," *WSEAS Trans. on Mathematics*, vol. (5),7, pp. 811–816, 2006.

[49] E. N. Mathioudakis and E. P. Papadopoulou, "Grid computing for Bi-CGSTAB applied to the solution of modified Helmholtz equation", *INt. J. Applied Maths and comp. sciences*, vol. 4, no. 3, pp. 179-184, 2007.

[50] NVIDIA's Next Generation CUDA Compute Architecture: Fermi, Whitepaper, 2009

[51] J. Palacios and J. Triska, "A Comparison of Modern GPU and CPU Architectures: And the Common Convergence of Both", 2011

[52] T.S. Papatheodorou, "Inverses for a Class of Banded Matrices and Applications to Piecewise Cubic Approximation", *J. Comp. Appl. Math.* 8 (4), 285-288, 1982.

[53] Y. Saad, "Iterative Methods for sparse linear systems", SIAM, 2003.

[54] J. Sanders and E. Kandrot "CUDA by example: An Itroduction to General-Purpose GPU Programming", Addison-Wesley, 2011.

[55] X. Trompoukis, "Numerical Solution of Aerodynamic-Aeroelastic Problems on Graphics Processing Units", NTUA, Athens, URL: dspace.lib.ntua.gr/bitstream/123456789/6542/1/trompoukisx_gpus.pdf 2012

[56] R.S. Varga, "Matrix Iterative Analysis",*Prentice Hall*, Englewood Cliffs, NJ, 1962

[57] N. D. Vilanakis, E. N. Mathioudakis, "Parallel iterative solution of the Hermite Collocation equations on GPUs II", Procs of IC-MSQUARE, To appear in Journal of Physics: Conference Series

[58] V. Volkov, "Use registers and multiple outputs per thread on GPU", Whitepaper, UC Berkeley, 2010

[59] M. Wolfe, "GPU Programming with OpenACC Tutorials", SC12, 2012

[60] M. Wolfe, "Programming Heterogeneous X64+GPU Systems Using OpenACC", The Portland Group in Cooperation with IEEE Computer Society, 2013

[61] D.M. Young, "Iterative Solution of Large Linear Systems", *Academic Press*, New York, 1981

[62] M. Zahran, "Graphics Processing Units: Architecture and Programming, History of GPU Computing", New York University, 2012

[63] Ν. Χατζηγεωργίου, ¨Επιτάχυνση της εύρεσης της κυκλικής αναπαράστασης Συνόλων Μερικής Διάταξης POSET με χρήση GPU", Διπλωματική εργασία, ΕΜΠ, 2011.