ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΓΕΝΙΚΟ ΤΜΗΜΑ

**ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**
ΕΦΑΡΜΟΣΜΕΝΕΣ ΕΠΙΣΤΗΜΕΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ

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

# ΤΕΧΝΙΚΕΣ ΠΟΛΥΠΛΕΓΜΑΤΟΣ ΓΙΑ ΠΑΡΑΛΛΗΛΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΥΠΟΛΟΓΙΣΜΩΝ ΜΕ ΕΠΙΤΑΧΥΝΤΕΣ

**ΝΙΚΗ ΕΜ. ΧΑΡΑΛΑΜΠΑΚΗ**

Επιβλέπων : Επικ. Καθηγητής **Εμμανουήλ Μαθιουδάκης**

**ΧΑΝΙΑ , 2014**

Η διατριβή αυτή εξετάστηκε επιτυχώς από τη παρακάτω Τριμελή Επιτροπή

- Επικ. Καθηγητή Εμμανουήλ Μαθιουδάκη (επιβλέπων)
- Καθηγητή Ιωάννη Σαριδάκη
- Καθηγήτρια Ελενα Παπαδοπούλου

η οποία ορίστηκε κατά τη 13η /6-12-2012 συνεδρίαση της Γενικής Συνέλευσης Ειδικής Σύνθεσης του Γενικού Τμήματος του Πολυτεχνείου Κρήτης.

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους εκείνους που με συνέδραμαν για την εκπόνηση αυτής της διατριβής. Πρώτα απ' όλους ευχαριστώ τον επιβλέποντα των μεταπτυχιακών σπουδών μου Επίκουρο Καθηγητή Εμμανουήλ Μαθιουδάκη, ο οποίος εμπλούτισε τη φαρέτρα μου με τα βέλη της ορθής αναζήτησης και έρευνας με τελικό στόχο την ολοκλήρωση της διατριβής μου και την επιστημονική μου ανάταση.

Ιδιαίτερα ευχαριστώ τον Διευθυντή του Εργαστηρίου Εφαρμοσμένων Μαθηματικών και Ηλεκτρονικών Υπολογιστών (ΕΕΜΗΥ) Καθηγητή Ιωάννη Σαριδάκη και την Καθηγήτρια Έλενα Παπαδοπούλου για τις εύστοχες και γόνιμες παρατηρήσεις που μου παρείχαν ως μέλη της Επιτροπής.

Τον υποψήφιο διδάκτορα Βασίλειο Μάνδικα ευχαριστώ για τις επιστημονικές γνώσεις που μου παρείχε στην τεχνική πολυπλέγματος.

Τέλος ευχαριστώ τους συμφοιτητές μου, Νικόλαο Βιλανάκη και Δημήτριο Μπομπολάκη,για τον πολύτιμο και ανεκτίμητο χρόνο που μοιραστήκαμε και φυσικά την οικογένειά μου που με υποστήριξε ώστε να ολοκληρώσω τις σπουδές μου.

# Περίληψη

Τα σημερινά υπερυπολογιστικά συστήματα για την ενίσχυση της υπολογιστικής τους ισχύος διαθέτουν υποσυστήματα διεξαγωγής επιστημονικών υπολογισμών, τα οποία ουσιαστικά λειτουργούν ως επιταχυντές υπολογισμών. Η βασικότερη κατηγορία επιταχυντών ανήκει στα γραφικά υποσυστήματα (GPUs), τα οποία σήμερα διαθέτουν επεξεργαστές με χιλιάδες υπολογιστικούς πυρήνες και σημαντικά μεγέθη τοπικής μνήμης. Στη διατριβή αυτή παρουσιάζεται η εφαρμογή της Τεχνικής Πολυπλέγματος για την επαναληπτική επίλυση γενικών, δομημένων και αραιών γραμμικών συστημάτων, που προκύπτουν από την αριθμητική μέθοδο επίλυσης Προβλημάτων Συνοριακών Τιμών (ΠΣΤ) με χρήση της μεθόδου των πεπερασμένων διαφορών συμπαγών σχημάτων για ορθογώνια χωρία. Η χωρική διακριτοποίηση ανά κατεύθυνση μπορεί να είναι διαφορετική. Η υλοποίηση πραγματοποιείται σε υπολογιστικά περιβάλλοντα τα οποία διαθέτουν επιταχυντές με πολυεπεξεργαστικά γραφικά υποσυστήματα και γίνεται μελέτη της συμπεριφοράς απόδοσης ενός ειδικά σχεδιασμένου παράλληλου αλγορίθμου της μεθόδου για αυτού του είδους τις υπολογιστικές αρχιτεκτονικές.

Η κατασκευή του παράλληλου αλγορίθμου βασίστηκε σε ειδικές παράλληλες διαδικασίες γραμμικής άλγεβρας, για τις οποίες σχεδιάστηκαν εξειδικευμένοι αλγόριθμοι για αρχιτεκτονικές κοινής μνήμης. Αυτό ήταν αναγκαίο για την αποδοτική υλοποίηση της μεθόδου, διότι ο πίνακας συντελεστών των αγνώστων του γραμμικού συστήματος είναι συγκεκριμένης δομής, διαθέτει συγκεκριμένα χαρακτηριστικά και ιδιότητες, οι οποίες χρειάστηκε να ληφθούν υπόψη στη διεξαγωγή των υπολογισμών. Έτσι είναι εφικτή η καλύτερη εκμετάλλευση των δυνατοτήτων επιτάχυνσης της υπολογιστικής διαδικασίας από

τα υποσυστήματα επιτάχυνσης. Ο σχεδιασμός του αλγορίθμου βασίστηκε στην αρχή ότι η τοπική μνήμη των επιταχυντών είναι σημαντικά μικρότερη αυτής του υπολογιστικού συστήματος. Δηλαδή είναι προτιμότερο να κατασκευάζονται τα βασικά στοιχεία σύνθεσης του πίνακα συντελεστών των αγνώστων, ο οποίος αντιστοιχεί για κάθε πρόβλημα ανάλογα με το μέγεθος του κάθε πλέγματος, αντί αυτά να αποθηκεύονται ή να μεταφέρονται διαρκώς από τη κεντρική μνήμη του συστήματος σε αυτή του επιταχυντή. Οπότε και για αυτού του είδους τις διαδικασίες χρειάστηκε να κατασκευαστούν αποδοτικοί παράλληλοι αλγόριθμοι, ώστε να είναι εφικτή η επίλυση μεγάλων διακριτοποιημένων προβλημάτων.

Η διατριβή αυτή είναι δομημένη σε πέντε κεφάλαια: Στο πρώτο κεφάλαιο παρουσιάζεται συνοπτικά η εφαρμογή της αριθμητικής μεθόδου πεπερασμένων διαφορών συμπαγών σχημάτων κατά την επίλυση προβλημάτων τύπου anisotropic Poisson. Γίνεται χρήση μιας διχρωματικής αρίθμησης αγνώστων και εξισώσεων, ώστε να είναι εφικτή η διεξαγωγή ανεξάρτητων υπολογισμών κατά την επίλυση του γραμμικού συστήματος. Στο δεύτερο κεφάλαιο παρουσιάζονται οι βασικές αρχές της Τεχνικής Πολυπλέγματος, οι οποίες χρησιμοποιήθηκαν σε αυτή τη διατριβή. Στο τρίτο κεφάλαιο παρουσιάζεται η κατασκευή παράλληλου αλγορίθμου της μεθόδου για υλοποίηση σε υπολογιστικές αρχιτεκτονικές με επιταχυντές υπολογισμών. Το τελευταίο κεφάλαιο εμφανίζει τις μετρήσεις απόδοσης του αλγορίθμου της αριθμητικής επίλυσης, στην οποία έχει ενσωματωθεί η παράλληλη υλοποίηση της Τεχνική Πολυπλέγματος. Τα πειραματικά αποτελέσματα καθώς και τα συμπεράσματα απο την υλοποίηση του αλγορίθμου με τη χρήση γραφικών υποσυστημάτων συμπληρώνουν την τελευταία ενότητα. Τέλος, οι κώδικες προγραμμάτων των εφαρμογών που αναπτύχθηκαν σε αυτή τη διατριβή με τη χρήση της γλώσσας προγραμματισμού Fortran και του προτύπου OpenACC περιέχονται στο παράρτημα.

Τα ερευνητικά αποτελέσματα αυτής της εργασίας παρουσιάστηκαν [8] στα πλαίσια του διεθνούς συνεδρίου NumAn2014 *(http://numan2014.amcl.tuc.gr)*.

# Περιεχόμενα

# Κατάλογος Σχημάτων

# Κεφάλαιο 1

# Αριθμητικά σχήματα Συμπαγών Πεπερασμένων Διαφορών

Η μοντελοποίηση φυσικών φαινομένων και καταστάσεων οδηγεί στη παραγωγή Μερικών Διαφορικών Εξισώσεων (ΜΔΕ), οι οποίες επιλύονται αριθμητικά κι έτσι είναι εφικτή η μελέτη αυτών των φυσικών προβλημάτων. Μια αρκετά συνηθισμένη ΜΔΕ είναι γνωστή ως Poisson, με την οποία μπορεί να μοντελοποιηθούν για παράδειγμα φαινόμενα όπως αυτά του ηλεκτρισμού (υπολογισμός τάσης ρεύματος σε συγκεκριμένες περιοχές φόρτισης) και μηχανικής (φαινόμενα ομοιόμορφης ή ανομοιόμορφης διάχυσης σε υλικά). Στη διατριβή αυτή θα επιλυθούν με χρήση αριθμητικών μεθόδων προβλήματα που μοντελοποιούνται μέσω ΜΔΕ τύπου Poisson κάνοντας χρήση της αριθμητικής μεθόδου των πεπερασμένων διαφορών συμπαγών σχημάτων [10,13,40]. Για την επίλυση του παραγόμενου γραμμικού συστήματος εξισώσεων θα εφαρμοστεί η τεχνική του γεωμετρικού πολυπλέγματος ως προς μία και δύο χωρικές κατευθύνσεις. Η επιλογή αυτής της τεχνικής έγινε διότι τα αριθμητικά σχήματα πολυπλέγματος θεωρούνται από τους πιο

αποδοτικούς επιλυτές γραμμικών συστημάτων ακόμα και σε παράλληλα περιβάλλοντα υπολογισμών [8,12]. Η επίλυση προβλημάτων τα οποία μοντελοποιούνται μέσω Poisson ΜΔΕ, συνήθως αποτελούν επιμέρους διαδικασίες σε χρονικά εξελισσόμενα φαινόμενα. Αυτό έχει ως αποτέλεσμα την ανάγκη επίλυσης αρκετών Poisson προβλημάτων σε κάθε χρονικό βήμα της διακριτοποίησης. Για παράδειγμα κατά την μοντελοποίηση ασυμπίεστων ροών μέσω των εξισώσεων Navier-Stokes, χρειάζεται η διόρθωση της πίεσης αρκετές φορές σε κάθε χρονικό βήμα της αριθμητικής μεθόδου. Η διόρθωση της πίεσης η οποία περιγράφει το φαινόμενο της ασυμπιεστότητας της ροής μοντελοποιείται μέσω της επίλυσης μιας Poisson ΜΔΕ. Έτσι η επίλυση αυτή θα πρέπει να πραγματοποιείται όσο γίνεται ταχύτερα, ώστε η συνολική μέθοδος να μπορεί να θεωρηθεί ως αποδοτική.

## 1.1 Πεπερασμένες διαφορές συμπαγών σχημάτων υψηλής ακρίβειας

Σε αυτή τη διατριβή θα χρησιμοποιηθεί ως πρόβλημα μοντέλο η anisotropic-Poisson ΜΔΕ, η οποία στις δύο διαστάσεις έχει τη παρακάτω μορφή :

$$\varepsilon u_{xx}(x,y) + u_{yy}(x,y) = f(x,y)$$

ή

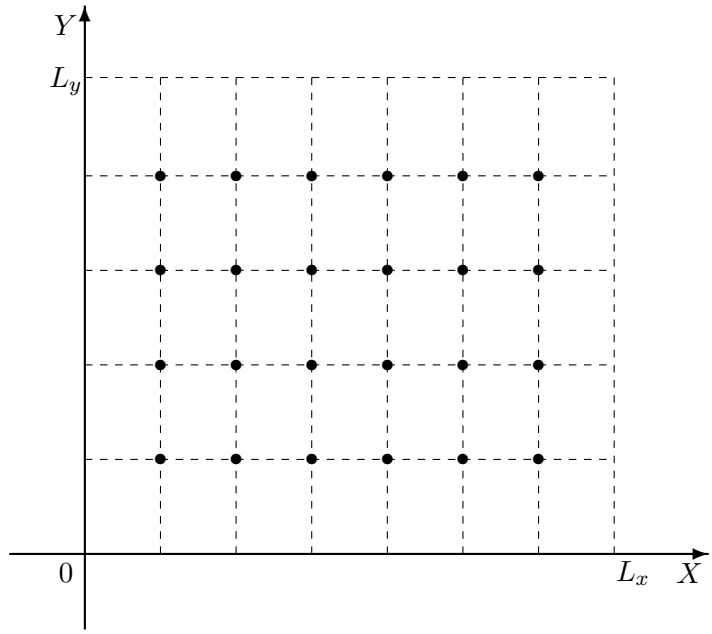
$$u_{xx}(x,y) + \varepsilon u_{yy}(x,y) = f(x,y)$$

ανάλογα με τη κατευθυνόμενη διάχυση, με κατάλληλες συνοριακές συνθήκες και $\varepsilon \ll 1$. Αν θεωρήσουμε ότι το πεδίο ορισμού της παραπάνω ΜΔΕ είναι το χωρίο $[0, L_x] \times [0, \overline{L}_y]$ και έστω ότι ισχύει $\varepsilon = \gamma^2$, τότε για το anisotropic πρόβλημα

$$u_{xx}(x,\overline{y}) + \varepsilon u_{\overline{yy}}(x,\overline{y}) = f(x,\overline{y})$$

με $\overline{y} = \gamma y$ για $(x,y) \in [0, L_x] \times [0, L_y] \equiv \Omega$ με $\overline{L}_y = \varepsilon L_y$ θα ορίζεται το παρακάτω μετασχηματισμένο isotropic Poisson πρόβλημα ως

$$u_{xx}(x,y) + u_{yy}(x,y) = f(x,y) \tag{1.1}$$




**Σχήμα 1.1:** Διακριτοποίηση του χωρίου $\Omega$ με $N_x = 7$ και $N_y = 5$.

με τις κατάλληλες συνοριακές συνθήκες. Η πραγματική λύση $u(x, y)$ και η συνάρτηση $f(x, y)$ θεωρούμε ότι είναι αρκετά ομαλές με συνεχείς μερικές παραγώγους. Έστω το ορθογώνιο χωρίο $\Omega \equiv [0, L_x] \times [0, L_y]$ στο οποίο ορίζεται το Poisson Πρόβλημα Συνοριακών Τιμών. Διακριτοποιούμε το πεδίο ορισμού $\Omega$ με ομοιόμορφη διαμέριση $\Delta x$ και $\Delta y$ ως προς κάθε χωρική κατεύθυνση με το πλήθος των υπολογιστικών κελιών να είναι $N_x = L_x/\Delta x$ και $N_y = L_y/\Delta y$ αντίστοιχα. Το σχήμα 1.1 εμφανίζει την περίπτωση αυτής της διακριτοποίησης σε επτά υπολογιστικά κελιά ως προς τη χωρική διάσταση $X$ και σε πέντε υπολογιστικά κελιά ως προς τη χωρική διάσταση $Y$.

Τα σημεία του πλέγματος (υπολογιστικοί κόμβοι) θα είναι $(x_i, y_i)$ όπου $0 \leq i \leq N_x$ και $0 \leq j \leq N_y$. Ο τύπος προσέγγισης της κεντρικής διαφοράς δεύτερης τάξης εκφράζεται ως :

$$\delta_x^2 u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}, \quad \delta_y^2 u_{i,j} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}$$

Η παραπάνω εξίσωση (1.1) μπορεί να διακριτοποιηθεί σε κάποιο σημείο $(x_i, y_i)$ του πλέγματος ως εξής:

$$\delta_x^2 u_{i,j} + \delta_y^2 u_{i,j} = f_{i,j} + O(\Delta^2), \tag{1.2}$$

όπου η έκφραση $O(\Delta^2)$ περιλαμβάνει όρους δεύτερης τάξης της μορφής $O(\Delta x^2 + \Delta y^2)$.

Για την κατασκευή υψηλότερης τάξης αριθμητικών συμπαγών σχημάτων πεπερασμένων διαφορών θα θεωρήσουμε το μονοδιάστατο Poisson πρόβλημα

$$u_{xx}(x) = f(x), x \in [0, L_x] \tag{1.3}$$

με κατάλληλες συνοριακές συνθήκες. Χρησιμοποιώντας ομοιόμορφη διαμέριση μήκους $\Delta x$ του χωρίου $\Omega$ , προκύπτουν $N_x = L_x/\Delta x$ υποδιαστήματα με κόμβους $x_i = i\Delta x$ και $0 \leq i \leq N_x$ . Η δεύτερη παράγωγος $u_{xx}$ σε κάθε κόμβο $x_i$ του πλέγματος μπορεί να προσεγγιστεί σύμφωνα με τον τελεστή κεντρικών διαφορών

$$u_{xx} = \delta_x^2 u_i - \frac{\Delta x^2}{12} u_{xxxx} + O(\Delta x^4). \tag{1.4}$$

όπου $u_{xxxx}$ θεωρούμε την τέταρτη παράγωγο στο τυχαίο σημείο διακριτοποίησης $x_i$.

Μη λαμβάνοντας υπόψη τους δύο τελευταίους όρους του δεξιού μέλους της παραπάνω εξίσωσης, παράγεται ο τύπος της κεντρικής διαφοράς δεύτερης τάξης:

$$\delta_x^2 u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta_x^2}$$

Η ιδέα για την παραγωγή αριθμητικών τύπων υψηλότερης ακρίβειας είναι να προσεγγίσουμε τον όρο $u_{xxxx}$ με τάξη δύο, επιτυγχάνοντας έτσι συνολική ακρίβεια τέταρτης τάξης. Παραγωγίζοντας δύο φορές την σχέση (1.3) προκύπτει ότι:

$$u_{xxxx} = f_{xx}. \tag{1.5}$$

Εφαρμόζοντας στη παραπάνω σχέση τη κεντρική διαφορά για τους όρους $u_{xx}$ και $f_{xx}$ έχουμε :

$$u_{xx} = \delta_x^2 u_i + O(\Delta x^2) \ , \ f_{xx} = \delta_x^2 f_i + O(\Delta x^2) \tag{1.6}$$

οπότε με αντικατάσταση στην (1.5) προκύπτει :

$$u_{xxxx} = \delta_x^2 f_i + O(\Delta x^2). \tag{1.7}$$

Αν αντικαστήσουμε στην (1.4) θα έχουμε :

$$u_{xx} = \delta_x^2 u_i - \frac{\Delta x^2}{12}\delta_x^2 f_i + O(\Delta x^4)$$

δηλαδή,

$$u_{xx} = \delta_x^2 u_i - \frac{\Delta x^2}{12}\delta_x^2 f_i + O(\Delta x^4).$$

οπότε για το αριθμητικό τελικό σχήμα ακρίβειας τέταρτης τάξης θα ισχύει για το πρόβλημα (1.3) η παρακάτω σχέση :

$$\delta_x^2 u_i - \frac{\Delta x^2}{12}\delta_x^2 f_i = f_i + O(\Delta x^4). \tag{1.8}$$

Παρατηρούμε ότι η μοναδική διαφορά από το αντίστοιχο σχήμα της δεύτερης τάξης είναι η εμπλοκή υπολογισμού προσέγγισης παραγώγων του δεξιού μέλους της διαφορικής εξίσωσης . Έτσι εξίσωση (1.8) μπορεί να έχει την μορφή :

$$\left(1 + \frac{\Delta x^2}{12}\delta_x^2\right)^{-1} \delta_x^2 u_i = f_i + O(\Delta x^4), \tag{1.9}$$

όπου ο τελεστής $T_x^{-1} \equiv \left(1 + \frac{\Delta x^2}{12}\delta_x^2\right)^{-1}$ εκφράζεται μόνο ως μια συμβολική μορφή αναπαράστασης. Χρησιμοποιώντας τον συμβολισμό κατά αντιστοιχία με το μονοδιάστατο πρόβλημα προκύπτει η παρακάτω συμβολική σχέση για τη περίπτωση του διδιάστατου προβλήματος:

$$\left(1 + \frac{\Delta x^2}{12}\delta_x^2\right)^{-1} \delta_x^2 u_{i,j} + \left(1 + \frac{\Delta y^2}{12}\delta_y^2\right)^{-1} \delta_y^2 u_{i,j} = f + O(\Delta^4) \tag{1.10}$$

όπου $O(\Delta^4)$ περιλαμβάνει όρους τάξης $O(\Delta x^4 + \Delta y^4)$. Εφαρμόζοντας τους συμβολικούς τελεστές $T_x^{-1}$ και $T_y^{-1}$ θα έχουμε:

$$\left(1 + \frac{\Delta y^2}{12}\delta_y^2\right)\delta_x^2 u_{i,j} + \left(1 + \frac{\Delta x^2}{12}\delta_x^2\right)\delta_y^2 u_{i,j} = \left(1 + \frac{\Delta y^2}{12}\delta_y^2\right)\left(1 + \frac{\Delta x^2}{12}\delta_x^2\right) f_{i,j} + O(\Delta^4)$$

ισοδύναμα

$$T_y\delta_x^2 u_{i,j} + T_x\delta_y^2 u_{i,j} = T_x T_y f_{i,j} + O(\Delta^4) \tag{1.11}$$

Σημειώνουμε ότι στον όρο $O(\Delta^4)$ περιέχονται όροι της μορφής $O(\Delta x^2 \cdot \Delta y^2)$. Άρα το τελικό σχήμα τέταρτης τάξης που θα προκύψει είναι:

$$(\delta_x^2 + \delta_y^2)u_{i,j} + \frac{1}{12}(\Delta x^2 + \Delta y^2)\delta_x^2\delta_y^2 u_{i,j} = f_{i,j} + \frac{1}{12}(\Delta x^2\delta_x^2 + \Delta y^2\delta_y^2)f_{i,j}. \tag{1.12}$$

Αν ισχύει ότι $\gamma = \Delta_x/\Delta_y$ τότε η παραπάνω σχέση γράφεται σε διακριτοποιημένη μορφή σε όλους τους κόμβους ως:

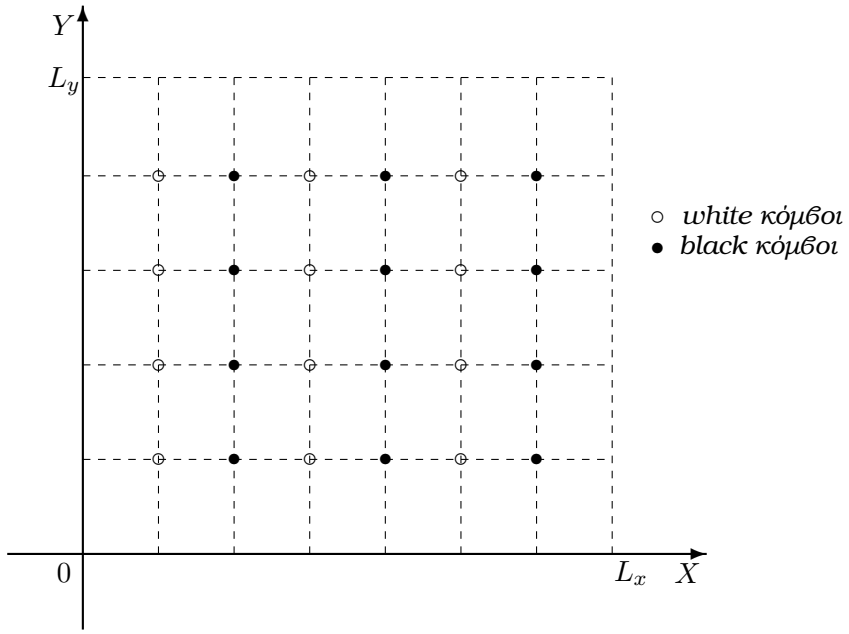
$$d(u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i-1,j-1}) + b(u_{i+1,j} + u_{i-1,j}) + c(u_{i,j+1} + u_{i,j-1}) - a_{i,j}u_{i,j} = \frac{\delta x^2}{2}(8f_{i,j} + f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}), \tag{1.13}$$

όπου οι συντελεστές θα είναι:

$$a = 10(1+\gamma^2), \quad b = 5 - \gamma^2, \quad c = 5\gamma^2 - 1, \quad d = (1+\gamma^2)/2.$$

## 1.2 Αλγεβρικό σύστημα εξισώσεων

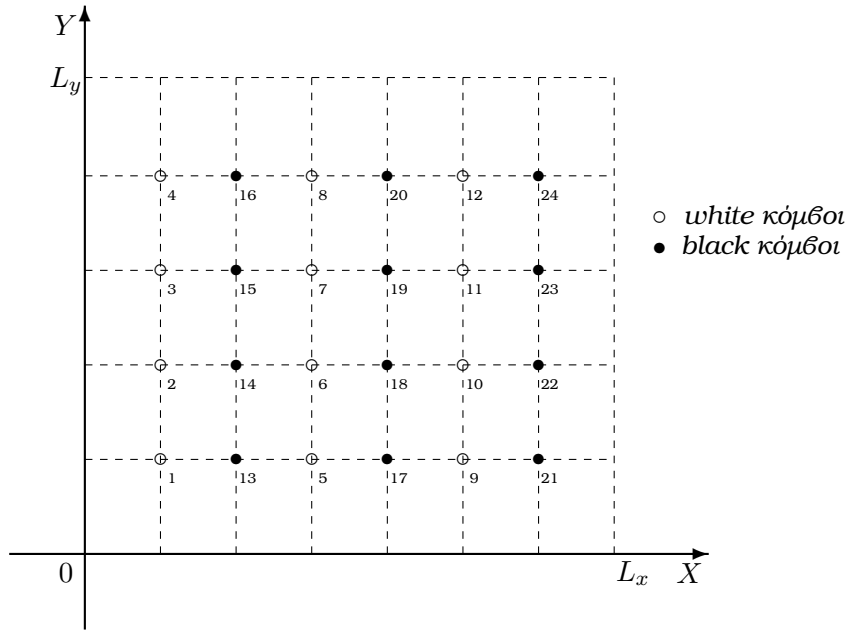
Για τον υπολογισμό των προσεγγιστικών τιμών της λύσης της ΜΔΕ, με το προηγούμενο αριθμητικό σχήμα πεπερασμένων διαφορών συμπαγών σχημάτων, χρειάζεται μία διαδικασία αρίθμησης αγνώστων και εξισώσεων, η οποία θα εφαρμοστεί για τη κατασκευή του




**Σχήμα 1.2:** Διχρωματική μέθοδος για $N_x = 7$ και $N_y = 5$.

γραμμικού συστήματος. Επειδή η δομή του πίνακα συντελεστών είναι άμεσα συνδεδεμένη με τη διαδικασία αρίθμησης, επιλέγουμε μια διχρωματική μέθοδο (zebra), ώστε ο πίνακας να διαθέτει αυξημένες παράλληλες ιδιότητες. Σύμφωνα με αυτήν, άγνωστοι οι οποίοι ανήκουν σε μια κάθετη στήλη του πλέγματος διακριτοποίησης ανήκουν και στην ίδια ομάδα χρωματισμού. Γειτονικές ομάδες αγνώστων δεν θα πρέπει να ανήκουν στην ίδια ομάδα χρώματος.

Η εικόνα 1.2 εμφανίζει την περίπτωση διακριτοποίησης όπου υπάρχουν επτά υπολογιστικά κελιά κατά την x κατεύθυνση και πέντε κατά την y κατεύθυνση. Η κατασκευή του πίνακα συντελεστών σύμφωνα με τη διχρωματική μέθοδο αρίθμησης επιτρέπει την απεξάρτηση ομάδων αγνώστων μεταξύ τους, με αποτέλεσμα την αύξηση των παράλληλων χαρακτηριστικών του πίνακα συντελεστών. Κατά την περίπτωση ύπαρξης Dirichlet συνοριακών συνθηκών στην ΜΔΕ, η κατασκευή του πίνακα ξεκινάει λαμβάνοντας υπόψιν την αρίθμηση των εξισώσεων του πρώτου χρώματος. Ξεκινώντας από κάτω προς τα πάνω και συνεχίζοντας τη διαδικασία αυτή με τις επόμενες στήλες του ίδιου χρώματος. Η κατασκευή του γραμμικού συστήματος ολοκληρώνεται με την εισαγωγή σε αυτό, των

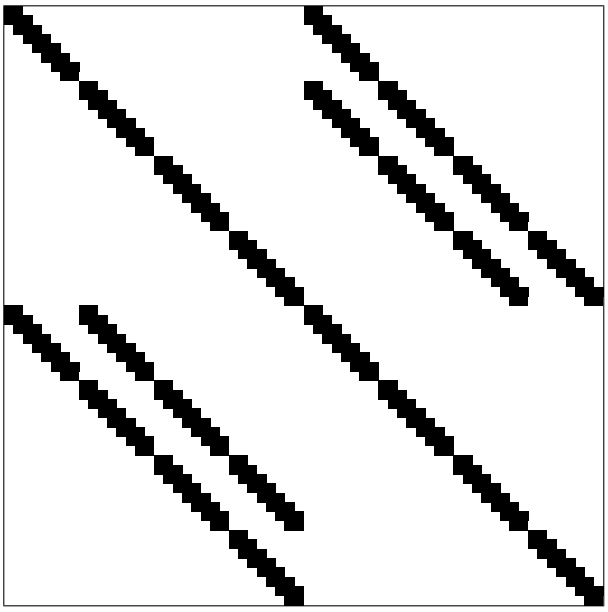



**Σχήμα 1.3:** Αρίθμηση διχρωματικής μεθόδου για $N_x = 7$ και $N_y = 5$.

εξισώσεων του επόμενου χρώματος με όμοιο τρόπο. Έτσι ο πίνακας των συντελεστών των αγνώστων θα προκύψει σε block black-white μορφή.

Το σχήμα 1.3 εμφανίζει τη διχρωματική Black-White αρίθμηση του προηγούμενο παραδείγματος. Ο πίνακας των συντελεστών των αγνώστων $A$ έχει την παρακάτω γενική block δομή

$$A = \begin{bmatrix}
A_0 & O & O & \dots & O & O & O & A_1 & O & O & \dots & O & O & O \\
O & A_0 & O & \dots & O & O & O & A_1 & A_1 & O & \dots & O & O & O \\
O & O & A_0 & \dots & O & O & O & O & A_1 & A_1 & \dots & O & O & O \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
O & O & O & \dots & A_0 & O & O & O & O & O & \dots & A_1 & O & O \\
O & O & O & \dots & O & A_0 & O & O & O & O & \dots & A_1 & A_1 & O \\
O & O & O & \dots & O & O & A_0 & O & O & O & \dots & O & A_1 & A_1 \\
A_1 & A_1 & O & \dots & O & O & O & A_0 & O & O & \dots & O & O & O \\
O & A_1 & A_1 & \dots & O & O & O & O & A_0 & O & \dots & O & O & O \\
O & O & A_1 & \dots & O & O & O & O & O & A_0 & \dots & O & O & O \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
O & O & O & \dots & A_1 & A_1 & O & O & O & O & \dots & A_0 & O & O \\
O & O & O & \dots & O & A_1 & A_1 & O & O & O & \dots & O & A_0 & O \\
O & O & O & \dots & O & O & A_1 & O & O & O & \dots & O & O & A_0
\end{bmatrix},$$



**Σχήμα 1.4:** Black-White δομή του πίνακα συντελεστών

Το παραπάνω σχήμα εμφανίζει τη δομή του πίνακα για διακριτοποίηση με $N_x = N_y = 9$. Ο πίνακας κατασκευάζεται από τους βασικούς πίνακες $A_0$ και $A_1$, οι οποίοι αναφέρονται στην μονοδιάστατη περίπτωση. Αυτοί οι πίνακες είναι συμμετρικοί και τριδιαγώνιοι με τα παρακάτω στοιχεία

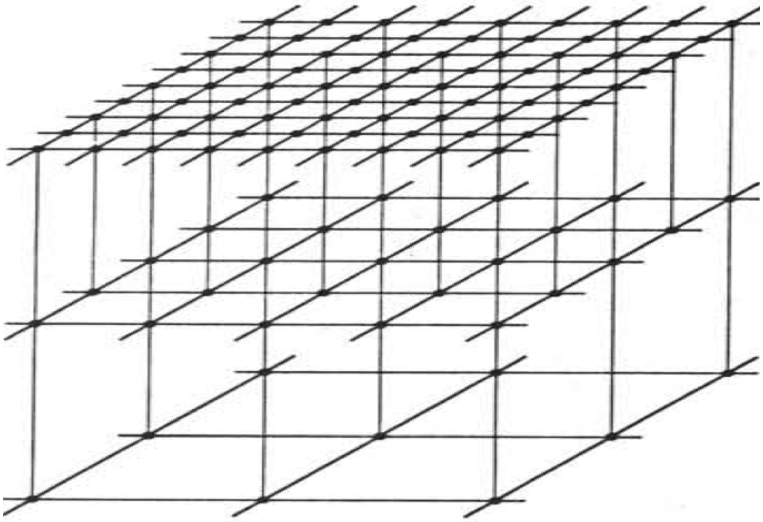
$A_0 = diag[5 - \gamma^2, -10(1 + \gamma^2), 5 - \gamma^2]$ και $A_1 = diag[\frac{1+\gamma^2}{2}, 5\gamma^2 - 1, \frac{1+\gamma^2}{2}]$.

Επειδή ο πίνακας των συντελεστών των αγνώστων του γραμμικού συστήματος είναι αραιός, μεγάλης διάστασης και block μορφής, για την αποδοτική επίλυσή του θα πρέπει να χρησιμοποιηθεί μια επαναληπτική μέθοδος σε αρχιτεκτονικές παράλληλων υπολογισμών. Η Τεχνική Πολυπλέγματος αποτελεί μία καλή επιλογή για αυτό το σκοπό και περιγράφεται στο επόμενο κεφάλαιο.

# Κεφάλαιο 2

# Τεχνική Πολυπλέγματος

Στην Αριθμητική Ανάλυση μια πολύ διαδεδομένη τεχνική επίλυσης γραμμικών συστημάτων τα οποία παράγονται από μια αριθμητική μέθοδο διακριτοποίησης Μερικών Διαφορικών Εξισώσεων (ΜΔΕ) είναι η Γεωμετρική Τεχνική Πολυπλέγματος (ΤΠΠ) [3-7,9,14,15,23,27,31]. Η τεχνική αυτή χρησιμοποιεί ένα αριθμητικό σχήμα διακριτοποίησης μέσω του οποίου παράγονται προσεγγίσεις της λύσης της ΜΔΕ υλοποιώντας μια ιεραρχία διακριτοποίησης. Η ΤΠΠ μπορεί να χρησιμοποιηθεί σαν επιλυτής το ίδιο καλά με τους προρυθμιστές (preconditioners) σε μια επαναληπτική διαδικασία. Η βασική ιδέα της ΤΠΠ είναι να επιταχύνει σύγκλιση η επαναληπτική μέθοδος, με διορθώσεις σε κάθε βήμα. Λύνοντας το πρόβλημα σε αραιότερο πλέγμα επιτυγχάνεται η επιτάχυνση της μεθόδου. Αυτή η αρχή βασίζεται στην διαδικασία κατά την οποία κατασκευάζονται καινούριοι κόμβοι δεδομένων ανάμεσα στους ήδη γνωστούς κόμβους, μεταξύ των αραιότερων και πυκνότερων πλεγμάτων. Η ΤΠΠ μπορεί να εφαρμοστεί σε συνδυασμό με οποιαδήποτε μέθοδο διακριτοποίησης, όπως είναι οι μέθοδοι πεπερασμένων στοιχείων



**Σχήμα 2.1:** Παράδειγμα μετάβασης σε τρία πλέγματα.

και διαφορών, ενώ μπορεί να επικρατήσει σε αυθαίρετα χωρία και συνοριακές συνθήκες. Το βασικότερο χαρακτηριστικό της ΤΠΠ είναι η ακολουθία ιεραρχικών αραιών και πυκνών πλεγμάτων που κατασκευάζονται. Το είδος των πλεγμάτων αυτών καθορίζεται από την αριθμητική μέθοδο επίλυσης της ΜΔΕ και αυτά τα πλέγματα μπορεί να είναι δομημένα ή μη. Σε αυτή την εργασία θα γίνει χρήση ομαδοποιημένων (block) δομημένων πλεγμάτων, αφού θα εφαρμοστεί η μέθοδος πεπερασμένων διαφορών σε ορθογώνια χωρία.

## 2.1 Περιγραφή της μεθόδου

Κατά την εφαρμογή της ΤΠΠ σε μια διάσταση αρχικά θεωρούμε ομοιόμορφο διαμερισμό του διαστήματος $\Omega \equiv [0, L_x]$ σε $n$ υποδιαστήματα, όπου $n = 2^p$ για θετικό ακέραιο $p$. Η ακολουθία των πλεγμάτων θα έχει τη μορφή $\Omega_h, \Omega_{2h}, \Omega_{4h}, \ldots, \Omega_{h_0}$ όπου $h = \frac{1}{n}$ είναι το βήμα διακριτοποίησης του πλέγματος. Θεωρούμε ότι αυτή η ακολουθία πλεγμάτων ολοκληρώνεται στο αραιότερο πλέγμα $\Omega_{h_0}$, το οποίο μπορεί να αποτελείται από μερικούς εσωτερικούς κόμβους. Η εικόνα 2.1 εμφανίζει ένα παράδειγμα εναλλαγής τριών πλεγμάτων. Επομένως για βήμα της διακριτοποίησης h κάθε κόμβος θα είναι στη θέση πλέγματος $x_i = (i-1)h$ με $i = 1, 2, ..., (n+1)$. Η διακριτοποίηση αυτή οδηγεί στη

κατασκευή ενός γραμμικού συστήματος

$$Au = f, \tag{2.1}$$

όπου $A$ είναι ο πίνακας των συντελεστών των αγνώστων $u_i$ και $f$ το δεξί μέλος του συστήματος με στοιχεία $f_i$, τα οποία αποτελούν τη διακριτοποίηση της συνάρτησης $f(x)$ σε κάθε κόμβο.

Εφαρμόζοντας τη ΤΠΠ σε δύο διαστάσεις για το παραλληλόγραμμο χωρίο $\Omega \equiv [0, L_x] \times [0, L_y]$, το οποίο θα διακριτοποιηθεί ομοιόμορφα σε κάθε κατεύθυνση σε $n$ υποδιαστήματα, με $n = 2^p$. Το βήμα διακριτοποίησης για κάθε κατεύθυνση είναι $h = 1/n$ και αν οι συντεταγμένες των σημείων σε κάθε κόμβο είναι $(x_i, y_j) = ((i-1)h, (j-1)h)$ με $i, j = 1, 2, ..., (n+1)$, η εφαρμογή μιας αριθμητικής μεθόδου διακριτοποίησης θα οδηγήσει στη κατασκευή του συστήματος $Au = f$, όπου $A$, $u$ και $f$ ανάλογα όπως τη μία διάσταση.

Υπάρχουν δύο αριθμητικά μεγέθη με τα οποία μπορεί να ελεγχθεί κατά πόσο η αριθμητική λύση του γραμμικού συστήματος προσεγγίζει την ακριβή λύση. Το ένα είναι το αλγεβρικό σφάλμα (algebraic error) που προκύπτει υπολογίζοντας τη νόρμα της διαφοράς της αριθμητικής λύσης από την ακριβή και το άλλο είναι το υπόλοιπο (residual) του γραμμικού συστήματος, το οποίο εκφράζει το κατά πόσο η αριθμητική λύση κατάφερε να προσεγγίσει τη πραγματική λύση του συστήματος $Au = f$.

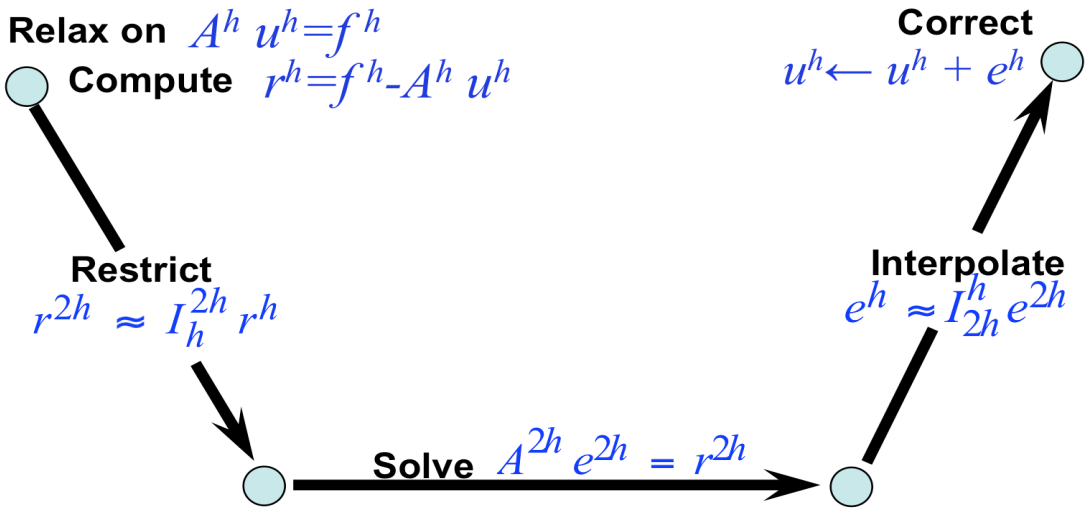
Από τα βασικότερα πλεονεκτήματα της ΤΠΠ είναι η απομείωση των υψίσυχνων σφαλμάτων κατά την μετάβασή τους σε διαφορετικά μεγέθους πλέγματα. Αυτό συμβαίνει διότι οι κλασσικές επαναληπτικές μέθοδοι, όπως είναι η Jacobi και η Gauss-Seidel [16,24,28,29], οι οποίες μπορεί να έχουν πολύ αργή σύγκλιση κατά την επίλυση του γραμμικού συστήματος, έχουν όμως τη δυνατότητα να μειώνουν πολύ γρήγορα τα στοιχεία του σφάλματος (ή υπολοίπου) προς τη κατεύθυνση των ιδιοδιανυσμάτων τα οποία αντιστοιχούν στις μεγαλύτερες ιδιοτιμές του επαναληπτικού πίνακα της μεθόδου. Αυτά τα στοιχεία του σφάλματος που σχετίζονται με τέτοιου τύπου ιδιοδιανύσματα ονομάζονται

υψίσυχνα μέλη, ενώ τα υπόλοιπα στοιχεία των σφαλμάτων συσχετίζονται με τα υπόλοιπα ιδιοδιανύσματα που ανήκουν στα χαμηλής συχνότητας μέλη του σφάλματος, τα οποία δε μπορούν να μειωθούν ικανοποιητικά από την επαναληπτική διαδικασία της μεθόδου και είναι αυτά τα οποία προκαλούν την αργή σύγκλιση της. Όμως πολλά στοιχεία των σφαλμάτων (περίπου τα μισά) απεικονίζονται με φυσικό τρόπο σε υψηλής συχνότητας μέλη στα αραιά πλέγματα, οπότε η κύρια ιδέα της μετάβασης σε αραιό πλέγμα είναι η εξαφάνιση στοιχείων του σφάλματος. Αυτό επιτυγχάνεται με την επαναληπτική εφαρμογή της μετάβασης σε διαρκώς αραιότερα ιεραρχικά πλέγματα. Η χρήση της μεθόδου Gauss-Seidel ως εξομαλυντή του σφάλματος του γραμμικού συστήματος, κατά την οποία γίνονται ένα ή δύο επαναληπτικά βήματα, ώστε να εξαλειφθούν οι υψίσυχνοι όροι του υπολοίπου και του σφάλματος είναι μια καλή επιλογή. Θα πρέπει να σημειωθεί ότι σε αυτή την υλοποίηση έγινε αρχικά χρήση της μεθόδου Jacobi ως εξομαλυντή σφάλματος, η οποία είναι πλήρως παραλληλοποιήσιμη όμως απέτυχε να το εξομαλύνει ικανοποιητικά. Αυτή η μέθοδος εξομάλυνσης βασίζεται στη διάσπαση του πίνακα των συντελεστών πίνακα $A = D - L - U$, όπου ο πίνακας αυτός παραμένει ο ίδιος για κάθε επαναληπτικό βήμα και αυτό οφείλεται στο ότι η μέθοδος ανήκει στη κατηγορία των Στατικών Επαναληπτικών Μεθόδων. Τα στοιχεία που συνθέτουν τη ΤΠΠ είναι η διαδικασία εξομάλυνσης των σφαλμάτων, οι τελεστές μεταφοράς από τα πυκνά πλέγματα στα αραιά και αντίστροφα και ο αλγόριθμος του επαναληπτικού κύκλου. Η επόμενη ενότητα παρουσιάζει τη πιο απλή έκδοση της ΤΠΠ η οποία περιλαμβάνει τη χρήση μόνο δύο πλεγμάτων.

## 2.2 Αλγόριθμος διαδικασίας των δύο πλεγμάτων

Κάθε επαναληπτικό βήμα (κύκλος) της μεθόδου των δύο πλεγμάτων αποτελείται από τη διαδικασία της προεξομάλυνσης (presmoothing) στην οποία επιτυγχάνεται η εξομάλυνση του υπολοίπου, της μεταφοράς από το πυκνό στο αραιό πλέγμα και αντίστροφα και της εξομάλυνσης του σφάλματος. Το βήμα αυτής της επαναληπτικής διαδικασίας των δύο πλεγμάτων, στην οποία υπολογίζεται η προσεγγιστική λύση $u_h^{m+1}$ από την $u_h^m$, εμφανίζει

η παρακάτω εικόνα




**Σχήμα 2.2:** Αλγόριθμος της Τεχνικής Πολυπλέγματος των δυο πλεγμάτων.

Ο αλγόριθμος των δύο πλεγμάτων περιλαμβάνει τις διαδικασίες εξομάλυνσης, το τελεστή της παρεκβολής από το πυκνό στο αραιό πλέγμα ($I_h^{2h}$), τη μέθοδο επίλυσης στο αραιό πλέγμα και τον τελεστή της παρεμβολής από το αραιό στο πυκνό πλέγμα ($I_{2h}^h$). Υλοποιήσεις της ΤΠΠ και θεωρία πολυπλέγματος δείχνουν ότι η επιλογή στοιχείων αυτών των διαδικασιών μπορεί να έχουν μεγάλη επιρροή στην αποδοτικότητα των αποτελεσμάτων του αλγορίθμου. Παρόλα αυτά δεν υπάρχουν γενικοί απλοί κανόνες για το πως να γίνει η επιλογή των στοιχείων αυτών ώστε να κατασκευαστούν βέλτιστοι αλγόριθμοι σε περίπλοκες εφαρμογές. Τα μεγέθη του πλέγματος επιλέγονται σύμφωνα με την βασική στρατηγική coarsening στην οποία το μέγεθος $h$ διπλασιάζεται σε κάθε εναλλαγή πλέγματος. Στις δύο διαστάσεις αυτό συμβαίνει μόνο ως προς τη μία κατεύθυνση έχουμε τη τεχνική του semi-coarsening, ενώ αν διπλασιάζεται ως προς και τις δύο κατευθύνσεις έχουμε την τεχνική του full-coarsening. Το μέγεθος του πλέγματος μπορεί να αλλάξει με διάφορους άλλους τρόπους όπως είναι το διχρωματικό coarsening (κόκκινο-μαύρο) ή να τετραπλασιάζεται το μέγεθος $h$, δηλαδή κάνοντας χρήση του πλέγματος $\Omega_{4h}$.

## 2.3 Τελεστές μετάβασης πληροφοριών μεταξύ διαδοχικών πλεγμάτων

**Παρεκβολή-Restriction**

Η μεταφορά των πληροφοριών που υπάρχουν στους κόμβους του πλέγματος από ένα πυκνό σε ένα αραιότερο περιγράφεται μέσω της εφαρμογής κατάλληλων τελεστών. Η διαδικασία αυτή ονομάζεται παρεκβολή. Έτσι ο τελεστής της παρεκβολής $I_h^{2h}$ απεικονίζει μία συνάρτηση που μεταφέρει τις πληροφορίες των κόμβων από το πλέγμα διάστασης $h$ στους κόμβους του πλέγματος διάστασης $2h$, με άλλα λόγια ο τελεστής αυτός μετασχηματίζει ένα διάνυσμα από το πυκνό πλέγμα σε ένα αραιότερο

$$I_h^{2h} : \Omega_h \to \Omega_{2h}.$$

Ένας τελεστής παρεκβολής που εμπλέκει τις πληροφορίες των γειτονικών κόμβων είναι ο περιοριστής πλήρους στάθμισης (full weighting restriction), ο οποίος στη μία κατεύθυνση έχει τη παρακάτω κατάλληλη μορφή

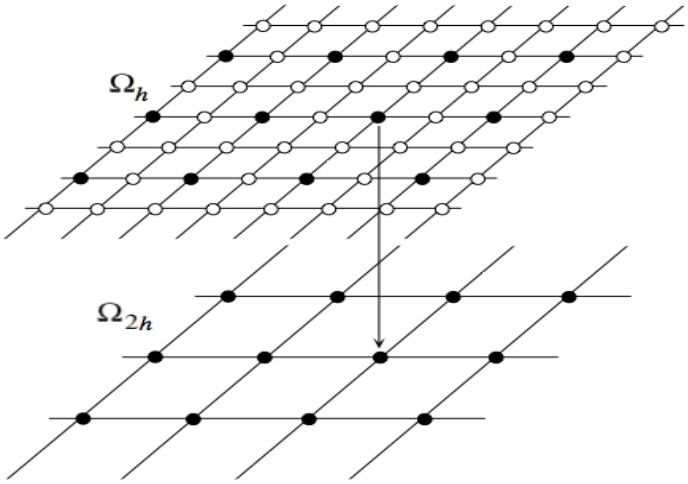
$$I_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \qquad \text{ή} \qquad I_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

ανάλογα με τη χωρική κατεύθυνση εφαρμογής του. Ο τελεστής αυτός σε μια διακριτοποιημένη συνάρτηση $r_{i,j}^h$ και για το κόμβο του πλέγματος με $r_{i,j}^{2h} \in \Omega_{2h}$ θα ισούται με

$$\begin{aligned} r_{i,j}^{2h} &= I_h^{2h} r_{i,j}^h \\ &= \tfrac{1}{4}(r_{i,2j-1}^h + 2r_{i,2j}^h + r_{i,2j+1}^h) \end{aligned} .$$

Αντίστοιχα ορίζεται ο τελεστής full weighting restriction ως προς τις δυο χωρικές κατευθύνσεις

$$I_h^{2h} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h^{2h}$$




**Σχήμα 2.3:** Σχηματική παράσταση διαδικασίας παρεκβολής.

ο οποίος αν εφαρμοστεί στην διακτιτοποιημένη συνάρτηση $r^h_{i,j}$ θα έχουμε

$$\begin{aligned} r^{2h}_{i,j} &= I^{2h}_h r^h_{i,j} \\ &= \frac{1}{16}[r^h_{2i-1,2j-1} + r^h_{2i-1,2j+1} + r^h_{2i+1,2j-1} + r^h_{2i+1,2j+1} \\ &+ 2(r^h_{2i,2j-1} + r^h_{2i,2j+1} + r^h_{2i-1,2j} + r^h_{2i+1,2j}) \\ &+ 4r^h_{2i,2j}]. \end{aligned}$$

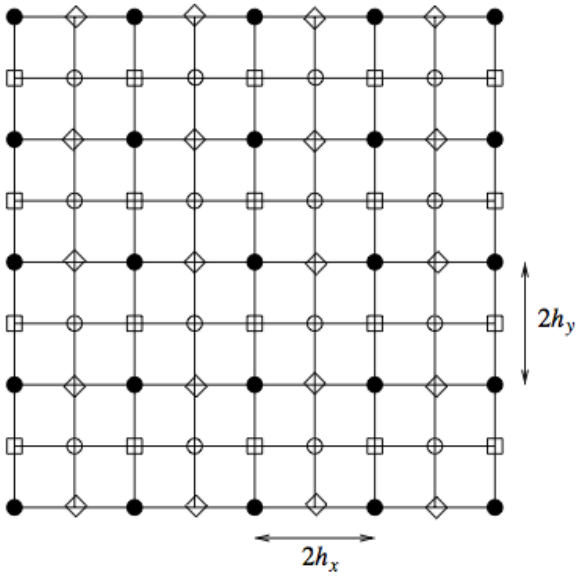
Είναι φανερό ότι υπολογίζεται ένας σταθμισμένος μέσος όρος από εννιά γειτονικούς κόμβους του πυκνού πλέγματος. Η εικόνα 2.3 εμφανίζει σχηματικά τη μετάβαση των πληροφορικών για κάθε κόμβο του πλέγματος από το πυκνό στο αραιό πλέγμα.

**Παρεμβολή-Interpolation**

Η παρεμβολή είναι η αντίστροφη διαδικασία της παρεκβολής όπου ο τελεστής της μετασχηματίζει ένα διάνυσμα από το αραιό πλέγμα σε ένα πυκνότερο, μεταφέρει δηλαδή τις πληροφορίες των κόμβων ανάμεσα σε δύο πλέγματα ως

$$I^h_{2h} : \Omega_{2h} \to \Omega_h.$$

Ένας τελεστής παρεμβολής που χρησιμοποιείται συχνά είναι ο γραμμικός (bilinear interpolation), ο οποίος στη μία κατεύθυνση έχει τη παρακάτω κατάλληλη μορφή




**Σχήμα 2.4:** Σχηματική παράσταση διαδικασίας παρεμβολής.

$$I_{2h}^{h} = \frac{1}{2}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \qquad \text{ή} \qquad I_{2h}^{h} = \frac{1}{2}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}.$$

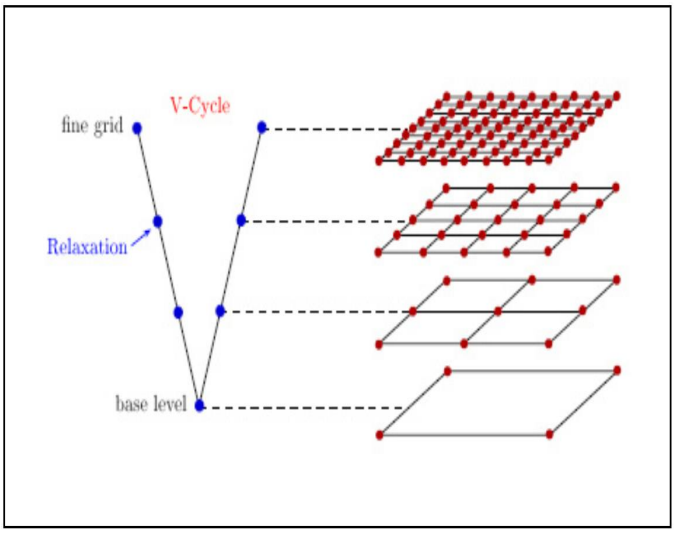
Εφαρμόζοντας το τελεστή αυτό σε μια διακριτοποιημένη συνάρτηση $r_{i,j}^{2h}$ σε ένα κόμβο του πυκνού πλέγματος $\Omega_h$ θα υπολογίζεται ως

$$\begin{cases} r_{i,2j}^{h} &= r_{i,j}^{2h} \\ r_{i,2j+1}^{h} &= \frac{1}{2}(r_{i,j}^{2h} + r_{i,j+1}^{2h}) \end{cases}.$$

Με όμοιο τρόπο ορίζεται ο τελεστής bilinear interpolation ως προς τις δυο χωρικές κατευθύνσεις

$$I_{2h}^{h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h}^{h}$$

ο οποίος αν εφαρμοστεί στη διακριτοποιημένη συνάρτηση $r_{i,j}^{2h}$ θα έχουμε για τους κόμβους του πυκνού πλέγματος




**Σχήμα 2.5:** Σχηματική παράσταση αλγορίθμου V-κύκλου τριών επιπέδων.

$$\begin{cases} r^h_{2i,2j} &= r^{2h}_{i,j} \\ r^h_{2i+1,2j} &= \frac{1}{2}(r^{2h}_{i,j} + r^{2h}_{i+1,j}) \\ r^h_{2i,2j+1} &= \frac{1}{2}(r^{2h}_{i,j} + r^{2h}_{i,j+1}) \\ r^h_{2i+1,2j+1} &= \frac{1}{4}(r^{2h}_{i,j} + r^{2h}_{i+1,j} + r^{2h}_{i,j+1} + r^{2h}_{i+1,j+1}) \end{cases} . \tag{2.2}$$

Η εικόνα 2.4 εμφανίζει τη διαδικασία παρεμβολής για τη μετάβαση από το αραιό πλέγμα στο πυκνότερο. Οι κόμβοι του αραιού πλέγματος παρουσιάζονται με τους μαύρους κύκλους, ενώ αυτοί του πυκνού εμφανίζονται και με τα υπόλοιπα σύμβολα ανάλογα με το αριθμητικό σχήμα του καθενός σύμφωνα με το παραπάνω τελεστή παρεμβολής.

Οι τελεστές που περιγράφηκαν είναι αυτοί που χρησιμοποιήθηκαν για την υλοποίηση της ΤΠΠ σε αυτή την εργασία σύμφωνα με τα ερευνητικά αποτελέσματα που παρουσιάζονται στην εργασία [19]. Σε αυτή τη μελέτη αποδοτικότερος αλγόριθμος που υλοποιεί την ΤΠΠ προέκυψε αυτός του V-κύκλου. Η εικόνα 2.5 εμφανίζει μια σχηματική αναπαράσταση του αλγορίθμου για τη περίπτωση μετάβασης σε τρία πλέγματα εκτός του βασικού πλέγματος στο οποίο θα προσεγγιστεί η λύση. Σύμφωνα με τον αλγόριθμο αυτό υπολογίζεται αρχικά το υπόλοιπο $r = b - Au$ στο πλέγμα στο οποίο προσεγγίζεται

η λύση του γραμμικού συστήματος $Au = b$ με αρχική προσέγγιση την $u^0$. Στην συνέχεια υπάρχουν διαδοχικές μεταβάσεις σε αραιότερα κάθε φορά πλέγματα στα οποία αφού υπολογιστεί το αντίστοιχο υπόλοιπο $r^h$, επιτυγχάνεται εξομάλυνση του σφάλματος με τη χρήση ενός βήματος της επαναληπτικής μεθόδου για την επίλυση του γραμμικού συστήματος σφάλματος $A^h e^h = r^h$. τη συνέχεια αφού εφαρμοστεί ο τελεστής παρεμβολής στο υπόλοιπο $r^h$ υπολογίζεται το $r^{2h}$ στο αραιότερο πλέγμα κι έτσι είναι εφικτή η αναδρομική εφαρμογή αυτής της διαδικασίας μέχρι το πιο αραιό πλέγμα το οποίο έχει επιλεγεί. Το πλήθος υλοποίησης αυτής της διαδικασίας καθορίζει τα επίπεδα τα οποία διαθέτει ο αλγόριθμος του V-κύκλου. Στο πιο αραιό πλέγμα επιλύεται ευκολότερα το γραμμικό σύστημα του σφάλματος. Η λύση του στη συνέχεια παρεμβάλλεται στο αμέσως επόμενο πυκνότερο πλέγμα στο οποίο επιτυγχάνεται η διόρθωση του σφάλματος το οποίο έχει εξομαλυνθεί κατά τη φάση της εξομάλυνσης πριν τη διαδικασία της παρεκβολής. Το διορθωμένο σφάλμα εξομαλύνεται ξανά με την εφαρμογή ενός επαναληπτικού βήματος της μεθόδου εξομάλυνσης, το οποίο στη συνέχεια παρεμβάλλεται στο πιο πυκνό πλέγμα. Η διαδικασία αυτή υλοποιείται σε ίδιο πλήθος με τα επίπεδα του V-κύκλου, όπου στο τέλος επιτυγχάνεται η διόρθωση της προσεγγιστικής λύσης του γραμμικού συστήματος. Εξετάζεται αν είναι ικανοποιητική η νέα προσέγγιση κι αν αυτό δε συμβαίνει τότε η διαδικασία του V-κύκλου επαναλαμβάνεται. Ο παρακάτω αλγόριθμος περιγράφει τη διαδικασία αυτή για τη περίπτωση $l_v$ επιπέδων ή εναλλαγές πλεγμάτων.

**Αλγόριθμος V-κύκλου**

*while* $||\boldsymbol{f}^0 - A^0\boldsymbol{u}^0|| < tol$ *do*
  *for* $l = 0$ *to* $l_v - 1$ *do*
    *Evaluate residual* $\boldsymbol{r}^l$
    *Smooth with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
    *Restrict residual* $\boldsymbol{r}^l$ *to* $\boldsymbol{r}^{l+1}$
  *enddo*
  *Solve with Gauss-Seidel* $A^{l_v}\, e^{l_v} = \boldsymbol{r}^{l_v}$
  *for* $l = l_v - 1$ *to* $0$ *do*
    *Interpolate error* $\boldsymbol{e}^{l+1}$ *to* $\boldsymbol{e}^l$
    *Update error* $\boldsymbol{e}^l$
    *Smooth with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
  *enddo*
  *Update* $\boldsymbol{u}^0 = \boldsymbol{u}^0 + \boldsymbol{e}^0$
*enddo*

# Κεφάλαιο 3

# Παράλληλος αλγόριθμος ΤΠΠ

## 3.1 Γραφικά υποσυστήματα επιτάχυνσης υπολογισμών

Τα τελευταία χρόνια στον κατάλογο των 500 υπερυπολογιστικών συστημάτων [39] εμφανίζονται διαρκώς και περισσότερα μηχανήματα τα οποία περιλαμβάνουν επιταχυντές υπολογισμών. Τα περισσότερα τέτοια συστήματα διαθέτουν πολυεπεξεργαστικούς υπολογιστικούς κόμβους, οι οποίοι είναι συνδεδεμένοι σε ένα γρήγορο δίκτυο και περιλαμβάνουν γραφικά υποσυστήματα(GPUs), τα οποία συμμετέχουν στην διεξαγωγή των αριθμητικών πράξεων του κάθε κόμβου. Τα GPUs διαθέτουν αυτόνομη μνήμη, μικρότερου μεγέθους από του υπολογιστικού κόμβου, ενώ συνδέονται με αυτόν μέσω διαύλου τύπου PCI-Express. Η βασικότερη διαφορά μεταξύ των παράλληλων αρχιτεκτονικών του πολυεπεξεργαστικού κόμβου και του γραφικού υποσυστήματος του είναι στο πλήθος των υπολογιστικών πυρήνων που διαθέτουν. Έτσι ενώ το GPU διαθέτει μερικές εκατοντάδες

οργανωμένους σε ομάδες, ο υπολογιστικός κόμβος διαθέτει μερικές δεκάδες μόνο, αλλά σημαντικά ισχυρότερους. Αυτό έχει ως αποτέλεσμα διαφορετικού τύπου υπολογισμοί να διεξάγονται αποδοτικότερα σε κάθε μία πλατφόρμα επιστημονικών υπολογισμών. Τα διαθέσιμα εργαλεία ανάπτυξης εφαρμογών σε αυτού του είδους των αρχιτεκτονικών είναι οι μεταγλωττιστές C και Fortran, οι οποίοι διαθέτουν κατάλληλα πρότυπα (APIs) για τη διεξαγωγή των παράλληλων υπολογισμών. Έτσι για την υλοποίηση των παράλληλων υπολογισμών σε ένα υπολογιστικό κόμβο γίνεται χρήση του OpenMP[36] και για τη διεξαγωγή υπολογισμών μεταξύ τους το MPI[37] κυρίως. Αντίθετα για την ανάπτυξη εφαρμογών με χρήση GPUs [11,17,18,25] γίνεται χρήση των OpenCL και Cuda. Τα τελευταία χρόνια έχει εξελιχθεί αρκετά το πρότυπο OpenACC [35], το οποίο επιτρέπει τη διαχείριση της λειτουργίας (μεταφορά δεδομένων και διεξαγωγή παράλληλων υπολογισμών) των GPUs από κώδικες προγραμμάτων σε C και Fortran. Αυτό οφείλεται στο γεγονός ότι με τη χρήση συγκεκριμένων οδηγιών σε κώδικες C και Fortran είναι δυνατή η κατασκευή από το μεταγλωττιστή κώδικα Cuda. Σε αυτή την διατριβή θα χρησιμοποιηθεί το πρότυπο OpenACC σε συνδυασμό με κώδικες της γλώσσας προγραμματισμού Fortran για την υλοποίηση των αριθμητικών μεθόδων σε παράλληλες αρχιτεκτονικές υπολογισμών, οι οποίες διαθέτουν γραφικά υποσυστήματα διεξαγωγής επιστημονικών υπολογισμών.

## 3.2 Κατασκευή παράλληλου αλγορίθμου

Η χρήση της διχρωματικής αρίθμησης αγνώστων και εξισώσεων, η οποία παρουσιάστηκε στο πρώτο κεφάλαιο, αυξάνει τις παράλληλες ιδιότητες του παραγόμενου γραμμικού συστήματος της αριθμητικής μεθόδου [20-22,30]. Αυτό συνέβη, διότι ομάδες αγνώστων ανεξαρτητοποιήθηκαν με αποτέλεσμα να είναι εφικτή η διεξαγωγή μεγάλου μέρους των υπολογισμών επίλυσης σε παράλληλες αρχιτεκτονικές. Για την αποδοτική υλοποίηση της μεθόδου σε τέτοιου είδους αρχιτεκτονικές υπολογισμών χρειάζεται η κατασκευή ενός αλγορίθμου, στον οποίο διαδικασίες οι οποίες συγκεντρώνουν το μεγαλύτερο υπολογι-

στικό κόστος θα υλοποιούνται αποδοτικά μέσω παράλληλων διαδικασιών. Ειδικότερα οι διαδικασίες υλοποίησης κάθε V-κύκλου οι οποίες συνθέτουν μία σειριακή διαδικασία, εξαιτίας της εναλλαγής του μεγέθους των προβλημάτων διακριτοποίησης σε ιεραρχικά εξαρτόμενα πλέγματα, είναι αυτές οι οποίες θα χρειαστεί να υλοποιηθούν μέσω κατάλληλων παράλληλων αλγορίθμων [1,2]. Οι διαδικασίες των παρεκβολών και παρεμβολών περιλαμβάνουν υπολογισμούς οι οποίοι έχουν τη δυνατότητα να υλοποιηθούν ανεξάρτητα μεταξύ τους σύμφωνα με τους κατάλληλους τελεστές που έχουν επιλεγεί. Οι πράξεις αυτές αναφέρονται σε αγνώστους οι οποίοι αντιστοιχούν σε ίδια ομάδα χρώματος, αλλά σε διαφορετική κάθετη στήλη του πλέγματος διακριτοποίησης. Οι παράλληλοι αλγόριθμοι στη συνέχεια περιγράφουν τις τέσσερις αυτές παράλληλες διαδικασίες, δηλαδή τις διαδικασίες παρεκβολής και παρεμβολής ως προς τη μία διεύθυνση και ως προς και δύο χωρικές κατευθύνσεις για τη συνάρτηση $r^h$, η οποία έχει τιμές στους κόμβους πλέγματος με βήμα διακριτοποίησης $h$. Ειδικότερα ο αλγόριθμος της παρεκβολής σε μια διεύθυνση αποτελείται από δυο παράλληλες επαναληπτικές διαδικασίες και έχει τη μορφή :

**Παράλληλος αλγόριθμος παρεκβολής ως προς μία κατεύθυνση**

$r^h \longrightarrow r^{2h}$

$N_w = \frac{N_x}{2} \qquad N_b = \frac{N_x}{2} - 1$

$N_y^{2h} = \frac{N_y^h}{2} - 1$

*for* $i = 1$ *to* $N_w + N_b$ *do in parallel*

*for* $j = 1$ *to* $N_y^{2h}$ *do in parallel*

$r_{i,j}^{2h} = r_{i,2j+1}^h + 2r_{i,2j}^h + r_{i,2j-1}^h$

*endfor*

*endfor*

Ο αλγόριθμος κάθε κύκλου στη διαδικασία ανόδου περιλαμβάνει την εφαρμογή του τελεστή παρεμβολής, η οποία περιγράφεται από δυο παράλληλες επαναληπτικές διαδικασίες. Η εξωτερική διαδικασία διαχειρίζεται τις τιμές πλέγματος που αναφέρονται στις κάθετες γραμμές του, ενώ η εσωτερική σε τιμές κάθε στήλης του ίδιου χρωματισμού. Ο παρακάτω αλγόριθμος περιγράφει αυτή τη διαδικασία ως

**Παράλληλος αλγόριθμος παρεμβολής σε μία κατεύθυνση**

$r^{2h} \longrightarrow r^h$

$N_w = \frac{N_x}{2} \qquad N_b = \frac{N_x}{2} - 1$

$N_y^h = 2N_y^{2h} + 1$

*for* $i = 1$ *to* $N_w + N_b$ *do in parallel*

$r_{i,1}^h = \frac{r_{i,1}^{2h}}{2}$

$r_{i,2}^h = r_{i,1}^{2h}$

*for* $j = 1$ *to* $N_y^{2h} - 2$ *do in parallel*

$r_{i,2j+1}^h = \frac{r_{i,j}^{2h} + r_{i,j+1}^{2h}}{2}$

$r_{i,2j+2}^h = r_{i,j+1}^{2h}$

*endfor*

$r_{i,N_y^h-1}^h = \frac{r_{i,N_y^{2h}-1}^{2h}}{2}$

*endfor*

Οι επόμενοι παράλληλοι αλγόριθμοι περιγράφουν τις διαδικασίες της παρεκβολής και παρεμβολής για τη περίπτωση εφαρμογής τελεστών σε δυο διαστάσεις. Αποτελούνται από δύο παράλληλες διαδικασίες, οι οποίες αναφέρονται σε τιμές πλέγματος του κάθε χρώματος, ενώ καθεμία περιλαμβάνει εσωτερικά μια παράλληλη κατασκευή των τιμών του αραιότερου ή πυκνότερου πλέγματος.

**Παράλληλος αλγόριθμος παρεκβολής σε δύο κατευθύνσεις**

$r^h \longrightarrow r^{2h}$

$$N_x^{2h} = \frac{N_x^h}{2} - 1 \qquad N_y^{2h} = \frac{N_y^h}{2} - 1$$

$$N_b^{2h} = \frac{N_x^{2h}+1}{2} - 1 \qquad N_w^{2h} = \frac{N_x^{2h}+1}{2}$$

*for* $i = 1$ *to* $N_w^{2h}$ *do in parallel*

*for* $j = 1$ *to* $N_y^{2h}$ *do in parallel*

$$\begin{aligned} r^{2h}_{2i-1,j} = & \frac{1}{4}\left(r^h_{2i-1,2j-1} + 2r^h_{2i,2j-1} + r^h_{2i+1,2j-1}\right) \\ & + \frac{1}{4}\left(2r^h_{2i-1,2j} + 4r^h_{2i,2j} + 2r^h_{2i+1,2j}\right) \\ & + \frac{1}{4}\left(r^h_{2i-1,2j+1} + 2r^h_{2i,2j+1} + r^h_{2i+1,2j+1}\right) \end{aligned}$$

*endfor*

*endfor*

*for* $i = 1$ *to* $N_b^{2h}$ *do in parallel*

*for* $j = 1$ *to* $N_y^{2h}$ *do in parallel*

$$\begin{aligned} r^{2h}_{2i,j} = & \frac{1}{4}\left(r^h_{4i-1,2j-1} + 2r^h_{4i,2j-1} + r^h_{4i+1,2j-1}\right) \\ & + \frac{1}{4}\left(2r^h_{4i-1,2j} + 4r^h_{4i,2j} + 2r^h_{4i+1,2j}\right) \\ & + \frac{1}{4}\left(r^h_{4i-1,2j+1} + 2r^h_{4i,2j+1} + r^h_{4i+1,2j+1}\right) \end{aligned}$$

*endfor*

*endfor*

**Παράλληλος αλγόριθμος παρεμβολής σε δύο κατευθύνσεις**

$r^{2h} \longrightarrow r^h$

$N_x^h = 2N_x^{2h} + 1 \qquad N_y^h = 2N_y^{2h} + 1$

$N_b^h = \frac{N_x^h + 1}{2} - 1 \qquad N_w^h = \frac{N_x^h + 1}{2}$

*for* $i = 1$ *to* $N_w^h$ *do in parallel*

*for* $j = 1$ *to* $N_y^{2h} + 1$ *do in parallel*

*if* $i = 1$ *then*

*if* $j = 1$ *then*

$r_{1,1}^h = \frac{1}{4} r_{1,1}^{2h}$

*elseif* $j = N_y^{2h} + 1$ *then*

$r_{1,2(j-1)}^h = \frac{1}{2} r_{1,j-1}^{2h}$

$r_{1,2j-1}^h = \frac{1}{4} r_{1,j-1}^{2h}$

*else*

$r_{1,2(j-1)}^h = \frac{1}{2} r_{1,j-1}^{2h}$

$r_{1,2j-1}^h = \frac{1}{4} \left( r_{1,j-1}^{2h} + r_{1,j}^{2h} \right)$

*endif*

*elseif* $i = N_w^h$ *then*

*if* $j = 1$ *then*

$r_{2i-1,1}^h = \frac{1}{4} r_{i-1,1}^{2h}$

*elseif* $j = N_y^{2h} + 1$ *then*

$r_{2i-1,2(j-1)}^h = \frac{1}{2} r_{i-1,j-1}^{2h}$

$r_{2i-1,2j-1}^h = \frac{1}{4} r_{i-1,j-1}^{2h}$

*else*

$$\boldsymbol{r}^{h}_{2i-1,2(j-1)} = \tfrac{1}{2}\boldsymbol{r}^{2h}_{i-1,j-1}$$

$$\boldsymbol{r}^{h}_{2i-1,2j-1} = \tfrac{1}{4}\left(\boldsymbol{r}^{2h}_{i-1,j-1} + \boldsymbol{r}^{2h}_{i-1,j}\right)$$

*endif*

*else*

*if* $j = 1$ *then*

$$\boldsymbol{r}^{h}_{2i-1,1} = \tfrac{1}{4}\left(\boldsymbol{r}^{2h}_{i-1,1} + \boldsymbol{r}^{2h}_{i,1}\right)$$

*elseif* $j = N_y^{2h} + 1$ *then*

$$\boldsymbol{r}^{h}_{2i-1,2(j-1)} = \tfrac{1}{2}\left(\boldsymbol{r}^{2h}_{i-1,j-1} + \boldsymbol{r}^{2h}_{i,j-1}\right)$$

$$\boldsymbol{r}^{h}_{2i-1,2j-1} = \tfrac{1}{4}\left(\boldsymbol{r}^{2h}_{i-1,j-1} + \boldsymbol{r}^{2h}_{i,j-1}\right)$$

*else*

$$\boldsymbol{r}^{h}_{2i-1,2(j-1)} = \tfrac{1}{2}\left(\boldsymbol{r}^{2h}_{i-1,j-1} + \boldsymbol{r}^{2h}_{i,j-1}\right)$$

$$\boldsymbol{r}^{h}_{2i-1,2j-1} = \tfrac{1}{4}\left(\boldsymbol{r}^{2h}_{i-1,j-1} + \boldsymbol{r}^{2h}_{i-1,j} + \boldsymbol{r}^{2h}_{i,j-1} + \boldsymbol{r}^{2h}_{i,j}\right)$$

*endif*

*endif*

*endfor*

*endfor*

*for* $i = 1$ *to* $N_b^h$ *do* in parallel

*for* $j = 1$ *to* $N_y^{2h} + 1$ *do* in parallel

*if* $j = 1$ *then*

$$\boldsymbol{r}^{h}_{2i,1} = \tfrac{1}{2}\boldsymbol{r}^{2h}_{i,1}$$

*elseif* $j = N_y^{2h} + 1$ *then*

$$r^{h}_{2i,2(j-1)} = r^{2h}_{i,j-1}$$

$$r^{h}_{2i,2j-1} = \frac{1}{2} r^{2h}_{i,j-1}$$

*else*

$$r^{h}_{2i,2(j-1)} = r^{2h}_{i,j-1}$$

$$r^{h}_{2i,2j-1} = \frac{1}{2}\left(r^{2h}_{i,j-1} + r^{2h}_{i,j}\right)$$

*endif*

*endfor*

*endfor*

Ο αλγόριθμος V-κύκλου της ΤΠΠ περιλαμβάνει τη διαδικασία υπολογισμού του υπολοίπου $r = b - Au$ για κάθε πλέγμα. Η διαδικασία αυτή εμπλέκει τον πολλαπλασιασμό του πίνακα $A$ με ένα διάνυσμα και η πράξη αυτή μπορεί να εκτελεστεί παράλληλα σύμφωνα με τον παρακάτω αλγόριθμο. Στο πίνακα $A$ εμπλέκονται μόνο τα μη μηδενικά στοιχεία των συμμετρικών τριδιαγώνιων πινάκων $A_0$ και $A_1$. Λαμβάνοντας υπόψη και τις αριθμητικές τιμές των στοιχείων των πινάκων είναι δυνατός ο υπολογισμός του υπολοίπου χωρίς τη διεξαγωγή πράξεων με δομές πινάκων. Έτσι χρησιμοποιώντας μόνο τη θέση και την αριθμητική τιμή κάθε μη μηδενικού στοιχείου τους μπορεί να διεξαχθούν οι κατάλληλοι υπολογισμοί. Ο παράλληλος αλγόριθμος που ακολουθεί έχει κατασκευαστεί χωρίς τη χρήση των πινάκων με συνέπεια τη σημαντική μείωση των απαιτούμενων θέσεων μνήμης για την υλοποίηση αυτής της διαδικασίας.

**Παράλληλος αλγόριθμος υπολογισμού υπολοίπου**

$\boldsymbol{r = b - Au}$

*for* $i = 1$ *to* $N_y - 1$ *do in* parallel

*if* $i = 1$ *then*

$\boldsymbol{r_1^w = cu_1^b + du_2^b}$

*elseif* $i = N_y - 1$ *then*

$\boldsymbol{r_i^w = du_{i-1}^b + cu_i^b}$

*else*

$\boldsymbol{r_i^w = du_{i-1}^b + cu_i^b + du_{i+1}^b}$

*endif*

*endfor*

*for* $k = 1$ *to* $\frac{N_x}{2} - 2$ *do in* parallel

$j = k(N_y - 1)$

$t = (k-1)(N_y - 1)$

*for* $i = 1$ *to* $N_y - 1$ *do in* parallel

*if* $i = 1$ *then*

$\boldsymbol{r_{j+1}^w = cu_{t+1}^b + du_{t+2}^b + cu_{j+1}^b + du_{j+2}^b}$

*elseif* $i = N_y - 1$ *then*

$\boldsymbol{r_{j+i}^w = du_{t+i-1}^b + cu_{t+i}^b + du_{j+i-1}^b + cu_{j+i}^b}$

*else*

$\boldsymbol{r_{j+i}^w = du_{t+i-1}^b + cu_{t+i}^b + du_{t+i+1}^b + du_{j+i-1}^b + cu_{j+i}^b + du_{j+i+1}^b}$

*endif*

*endfor*

*endfor*

$j = (\frac{N_x}{2} - 1)(N_y - 1)$

$j = (\frac{N_x}{2} - 2)(N_y - 1)$

*for* $i = 1$ *to* $N_y - 1$ *do in parallel*

*if* $i = 1$ *then*

$$\boldsymbol{r^w_{j+1} = cu^b_{t+1} + du^b_{t+2}}$$

*elseif* $i = N_y - 1$ *then*

$$\boldsymbol{r^w_{j+i} = du^b_{t+i-1} + cu^b_{t+i}}$$

*else*

$$\boldsymbol{r^w_{j+i} = du^b_{t+i-1} + cu^b_{t+i} + du^b_{t+i+1}}$$

*endif*

*endfor*

*for* $k = 1$ *to* $\frac{N_x}{2} - 1$ *do in parallel*

$j = k(N_y - 1)$

$t = (k - 1)(N_y - 1)$

*for* $i = 1$ *to* $N_y - 1$ *do in parallel*

*if* $i = 1$ *then*

$$\boldsymbol{r^b_{t+1} = cu^w_{t+1} + du^w_{t+2} + cu^w_{j+1} + du^w_{j+2}}$$

*elseif* $i = N_y - 1$ *then*

$$\boldsymbol{r^b_{t+i} = du^w_{t+i-1} + cu^w_{t+i} + du^w_{j+i-1} + cu^w_{j+i}}$$

*else*

$$\boldsymbol{r^b_{t+i} = du^w_{t+i-1} + cu^w_{t+i} + du^w_{t+i+1} + du^w_{j+i-1} + cu^w_{j+i} + du^w_{j+i+1}}$$

*endif*

*endfor*

*endfor*

.

*for* $k = 1$ *to* $\frac{N_x}{2}$ *do in parallel*

$t = (k-1)(N_y - 1)$

*for* $i = 1$ *to* $N_y - 1$ *do in parallel*

*if* $i = 1$ *then*

$$\boldsymbol{r^w_{t+1}} = \boldsymbol{r^w_{t+1}} - \boldsymbol{a u^w_{t+1}} + \boldsymbol{b u^w_{t+2}}$$

*elseif* $i = N_y - 1$ *then*

$$\boldsymbol{r^w_{t+i}} = \boldsymbol{r^w_{t+1}} + \boldsymbol{b u^w_{t+i-1}} - \boldsymbol{a u^w_{t+i}}$$

*else*

$$\boldsymbol{r^w_{t+i}} = \boldsymbol{r^w_{t+1}} + \boldsymbol{b u^w_{t+i-1}} - \boldsymbol{a u^w_{t+i}} + \boldsymbol{b u^w_{t+i+1}}$$

*endif*

*endfor*

*endfor*

*for* $k = 1$ *to* $\frac{N_x}{2} - 1$ *do in parallel*

$t = (k-1)(N_y - 1)$

*for* $i = 1$ *to* $N_y - 1$ *do in parallel*

*if* $i = 1$ *then*

$$\boldsymbol{r^b_{t+1}} = \boldsymbol{r^b_{t+1}} - \boldsymbol{a u^b_{t+1}} + \boldsymbol{b u^b_{t+2}}$$

*elseif* $i = N_y - 1$ *then*

$$\boldsymbol{r^b_{t+i}} = \boldsymbol{r^b_{t+1}} + \boldsymbol{b u^b_{t+i-1}} - \boldsymbol{a u^b_{t+i}}$$

*else*

$$\boldsymbol{r^b_{t+i}} = \boldsymbol{r^b_{t+1}} + \boldsymbol{b u^b_{t+i-1}} - \boldsymbol{a u^b_{t+i}} + \boldsymbol{b u^b_{t+i+1}}$$

*endif*

*endfor*

*endfor*

*for* $i = 1$ *to* $(N_x - 1)(N_y - 1)$ *do in parallel*

$\boldsymbol{r_i} = \boldsymbol{b_i} - \boldsymbol{r_i}$

*endfor*

Η διαδικασία της εξομάλυνσης με την χρήση της επαναληπτικής μεθόδου Gauss-Seidel αποτελεί την πιο απαιτητική υπολογιστικά διαδικασία της ΤΠΠ, αν και σε κάθε επίπεδο εναλλαγής πλέγματος εκτελείται μόνο ένα επαναληπτικό βήμα. Μόνο στο αραιότερο πλέγμα εκτελούνται περισσότερα από ένα επαναληπτικά βήματα, μέχρι η μέθοδος να επιτύχει σύγκλιση. Σύμφωνα με τη διάσπαση $A = D_A - L_A - U_A$ του πίνακα συντελεστών

$$A = \begin{bmatrix} D_w & H_b \\ H_w & D_b \end{bmatrix}$$

κάθε επαναληπτικό βήμα θα έχει τη μορφή

$$(D_A - L_A)u^{(m+1)} = U_A u^{(m)} + f \ , \quad m = 0, 1, \ldots$$

Λαμβάνοντας υπόψη τη παρακάτω δομή των πινάκων

$$D_A = \begin{bmatrix} D_w & O \\ O & D_b \end{bmatrix} \ , \ L_A = \begin{bmatrix} O & O \\ -H_w & O \end{bmatrix} \text{ και } U_A = \begin{bmatrix} O & -H_b \\ O & O \end{bmatrix} \ ,$$

αλλά και τη παρακάτω διαμέριση του διανύσματος των αγνώστων $u$ καθώς και του δεξιού μέλους $f$ ως

$$u = \begin{bmatrix} u_w \\ u_b \end{bmatrix} \quad , \quad f = \begin{bmatrix} f_w \\ f_b \end{bmatrix}$$

τότε κάθε νέα προσέγγιση των αγνώστων περιγράφεται ως

$$\begin{bmatrix} D_w & O \\ H_w & D_b \end{bmatrix} \begin{bmatrix} u_w^{(m+1)} \\ u_b^{(m+1)} \end{bmatrix} = \begin{bmatrix} O & -H_b \\ O & O \end{bmatrix} \begin{bmatrix} u_w^{(m)} \\ u_b^{(m)} \end{bmatrix} + \begin{bmatrix} f_w \\ f_b \end{bmatrix} \quad ,$$

η οποία μπορεί να θεωρηθεί ως δύο ξεχωριστές προσεγγίσεις αρχικά των white αγνώστων $u_w^{m+1}$ και στη συνέχεια των black $u_b^{m+1}$. Οι δύο νέες αυτές διαδικασίες προσέγγισης περιγράφονται από τους μαθηματικούς τύπους

$$D_w u_w^{(m+1)} = -H_b u_b^{(m)} + f_w$$
$$D_b u_b^{(m+1)} = -H_w u_w^{(m+1)} + f_b.$$

Οι block διαγώνιοι πίνακες συντελεστών $D_w$ και $D_b$ των νέων προσεγγίσεων των αγνώστων, περιλαμβάνουν το συμμετρικό τριδιαγώνιο πίνακα $A_0$. Έτσι είναι εφικτή η παράλληλη προσέγγιση κάθε ομάδας αγνώστων που αντιστοιχούν σε μία κάθετη γραμμή του πλέγματος με την επίλυση των υποσυστημάτων που έχουν ως πίνακα συντελεστών τον πίνακα $A_0$. Η επίλυση κάθε τέτοιου υποσυστήματος μπορεί να γίνει άμεσα με τη χρήση της $L\,U$ διάσπασης ως $A_0 = L_0 U_0$. Επειδή ο $A_0$ είναι συμμετρικός και τριδιαγώνιος πίνακας, οι πίνακες $L_0$ και $U_0$ είναι διαθέσιμοι μέσω του TDMA (Thomas algorithm)[26] και θα έχουν τα παρακάτω στοιχεία

$$L_0 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\frac{b}{a} & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\frac{b}{a-\frac{b^2}{t_1}} & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & -\frac{b}{a-\frac{b^2}{t_{i-1}}} & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -\frac{b}{a-\frac{b^2}{t_i}} & 1 \end{bmatrix}$$

και

$$U_0 = \begin{bmatrix} -a & b & 0 & \cdots & 0 & 0 & 0 \\ 0 & \frac{b^2}{a} - a & b & \cdots & 0 & 0 & 0 \\ 0 & 0 & \frac{b^2}{p_1} - a & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{b^2}{p_{i-2}} - a & b & 0 \\ 0 & 0 & 0 & \cdots & 0 & \frac{b^2}{p_{i-1}} - a & b \\ 0 & 0 & 0 & \cdots & 0 & 0 & \frac{b^2}{p_i} - a \end{bmatrix}.$$

Τα στοιχεία αυτά μπορούν να υπολογιστούν από τους αναδρομικούς τύπους

$t_1 = a$, $t_i = a - \frac{b^2}{t_{i-1}}$ και $p_1 = -a$, $p_i = \frac{b^2}{p_{i-1}} - a$ για $i = 2, \ldots, N_y - 3$.

Είναι φανερό ότι κατά τη διαδικασία επίλυσης των υποσυστημάτων και πιο συγκεκριμένα στη φάση της μπρος-πίσω αντικατάστασης απαιτείται μόνο η αποθήκευση της υποδιαγωνίου του $L_0$ και της διαγωνίου του $U_0$ σε δύο διανύσματα διάστασης $N_y - 2$ και $N_y - 1$ αντίστοιχα. Για την κατασκευή του δεύτερου μέλους των υποσυστημάτων υπάρχει ο πολλαπλασιασμός του block πίνακα $H_b$ για το πρώτο και του block πίνακα $H_w$ για το δεύτερο με ένα διάνυσμα. Οι block αυτοί πολλαπλασιασμοί εμπλέκουν το συμμετρικό τριδιαγώνιο πίνακα $A_1$, με συνέπεια οι πράξεις αυτές να είναι πλήρως παραλληλοποιήσιμες. Ο αλγόριθμος κάθε επαναληπτικού βήματος της Gauss-Seidel μπορεί να περιγραφεί ως

**Παράλληλος αλγόριθμος Gauss-Seidel**

$\boldsymbol{Au = b}$

*for* $m = 0, 1, 2, \ldots$

*compute* $\boldsymbol{f_w = b_w - H_b u_b^{(m)}}$ *in parallel*

*solve* $\boldsymbol{D_w u_w^{(m+1)} = f_w}$ *in parallel*

*compute* $\boldsymbol{f_b = b_b - H_w u_w^{(m+1)}}$ *in parallel*

*solve* $\boldsymbol{D_b u_b^{(m+1)} = f_b}$ *in parallel*

*check for convergence*

*endfor*

**Παράλληλος αλγόριθμος επίλυσης** $D_w u^w = f^w$

*for* $k = 0$ *to* $\frac{N_x}{2} - 1$ *do in* parallel

$j = k(N_y - 1) + 1$

*solve* $\boldsymbol{L_0 y = f_j^w}$

*solve* $\boldsymbol{U_0 u_j^w = y}$

*endfor*

**Παράλληλος αλγόριθμος επίλυσης** $D_b u^b = f^b$

*for* $k = 0$ *to* $\frac{N_x}{2} - 2$ *do in* parallel

$j = k(N_y - 1) + 1$

*solve* $\boldsymbol{L_0 y = f_j^b}$

*solve* $\boldsymbol{U_0 u_j^b = y}$

*endfor*

Οι παράλληλες διαδικασίες οι οποίες συνθέτουν τον αλγόριθμο κάθε V-κύκλου περιλαμβάνουν υπολογισμούς οι οποίοι μπορούν να διεξαχθούν αποκλειστικά σε παράλληλες αρχιτεκτονικές. Έτσι στην αρχή κάθε V-κύκλου τα δεδομένα μπορούν να αποσταλούν στο υπολογιστικό υποσύστημα (GPU) από τον κεντρικό επεξεργαστή (CPU) πριν την έναρξη της διαδικασίας. Στο τέλος της διαδικασίας τα δεδομένα των αποτελεσμάτων θα πρέπει να επιστραφούν πίσω στο CPU από το GPU. Ο παράλληλος αλγόριθμος κάθε V-κύκλου, περιλαμβάνει μεταφορά δεδομένων μόνο στην αρχή και στο τέλος του, αφού

όλοι οι υπολογισμοί διεξάγονται αποκλειστικά στο GPU. Ο παράλληλος αλγόριθμος κάθε V-κύκλου μπορεί να περιγραφεί ως

*Send* $\boldsymbol{f}^0(1:n)$ *from* CPU *to* GPU
*while* $||\boldsymbol{f}^0 - A^0\boldsymbol{u}^0|| < tol$ *do*
  *for* $l = 0$ *to* $l_s - 1$ *do*
    *Evaluate in parallel residual* $\boldsymbol{r}^l$
    *Smooth in parallel with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
    *Restrict in parallel residual* $\boldsymbol{r}^l$ *to* $\boldsymbol{r}^{l+1}$ *with semi-coarsening*
  *enddo*
  *for* $l = l_s$ *to* $l_v - 1$ *do*
    *Evaluate in parallel residual* $\boldsymbol{r}^l$
    *Smooth in parallel with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
    *Restrict in parallel residual* $\boldsymbol{r}^l$ *to* $\boldsymbol{r}^{l+1}$ *with full-coarsening*
  *enddo*
  *Solve in parallel with Gauss-Seidel* $A^{l_v}\, e^{l_v} = \boldsymbol{r}^{l_v}$
  *for* $l = l_v - 1$ *to* $l_s$ *do*
    *Interpolate in parallel error* $\boldsymbol{e}^{l+1}$ *to* $\boldsymbol{e}^l$ *with full-coarsening*
    *Update in parallel error* $\boldsymbol{e}^l$
    *Smooth in parallel with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
  *enddo*
  *for* $l = l_s - 1$ *to* $0$ *do*
    *Interpolate in parallel error* $\boldsymbol{e}^{l+1}$ *to* $\boldsymbol{e}^l$ *with semi-coarsening*
    *Update in parallel error* $\boldsymbol{e}^l$
    *Smooth in parallel with Gauss-Seidel* $A^l\, e^l = \boldsymbol{r}^l$
  *enddo*
*Update in parallel solution vector* $\boldsymbol{u}^0 = \boldsymbol{u}^0 + \boldsymbol{e}^0$
*enddo*
*Send* $\boldsymbol{u}^0(1:n)$ *from* GPU *to* CPU

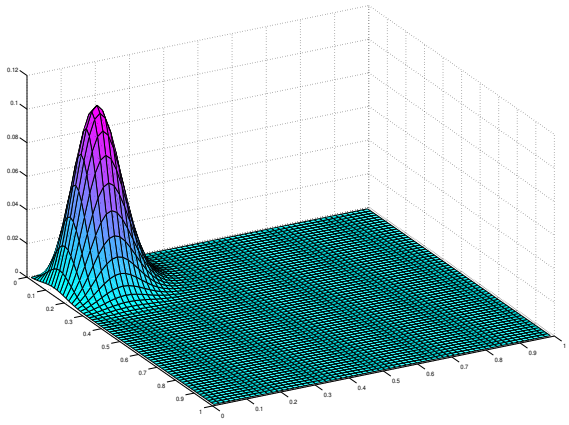
# Κεφάλαιο 4

# Αριθμητικά αποτελέσματα

## 4.1 Το πρόβλημα δοκιμής

Η υλοποίηση του παράλληλου αλγορίθμου της αριθμητικής μεθόδου έγινε σε ένα πολυεπεξεργαστικό σύστημα τύπου HP SL390s G7, το οποίο διαθέτει δύο επεξεργαστές 6 πυρήνων τύπου Xeon X5600@2.8 Ghz με μνήμη Cache 12MB Level 3 για τον καθένα. Η συνολική μνήμη του υπολογιστικού συστήματος είναι 24 GB και το λειτουργικό σύστημα είναι το Oracle Linux έκδοσης 6.2. Είναι εξοπλισμένο με GPU επιταχυντή υπολογισμών τύπου Fermi και ειδικότερα Tesla M2070 [34], η οποία διαθέτει 6GB μνήμη και συνολικά 448 υπολογιστικούς πυρήνες, οι οποίοι είναι οργανωμένοι σε 14 μικροεπεξεργαστές. Η ανάπτυξη της εφαρμογής έγινε με χρήση των μεταγλωττιστών της εταιρίας PGI έκδοσης 14.5 σε Fortran [38], κάνοντας χρήση υπολογισμών διπλής ακρίβειας καθώς και του προτύπου OpenACC [35] όπως αυτό έχει υλοποιηθεί στο συγκεκριμένο λογισμικό μεταγλωττιστών. Οι βασικές διαδικασίες γραμμικής άλγεβρας στο CPU επιτυγχάνονται με χρήση των βιβλιοθηκών υποπρογραμμάτων BLAS [32] και Lapack [33].
Το πρόβλημα μοντέλο που χρησιμοποιήθηκε επιδέχεται την παρακάτω ακριβή λύση

$$u(x,y) = 10\phi(x)\phi(y) \quad , \quad \phi(x) = \exp^{-100(x-0,1)^2}(x^2 - x)$$



**Σχήμα 4.1:** Η ακριβής λύση για το πρόβλημα δοκιμής.

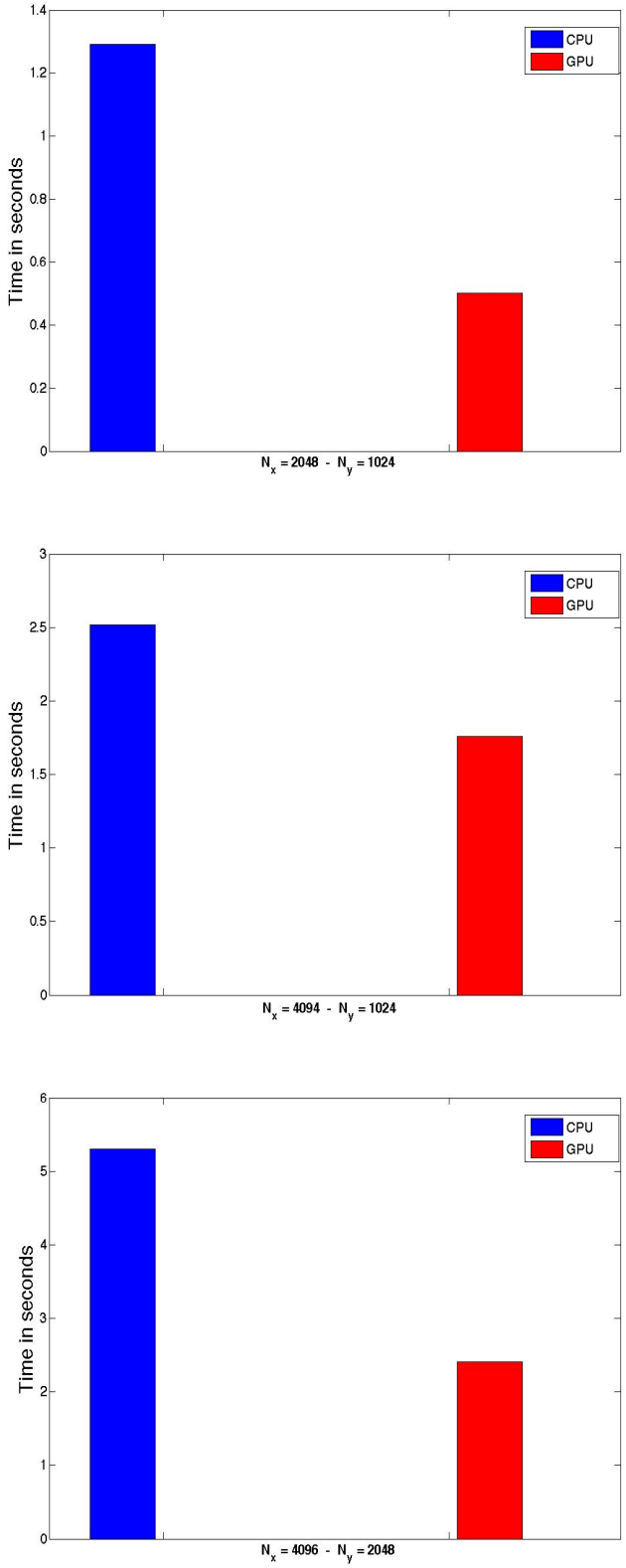
με πεδίο ορισμού το μοναδιαίο τετράγωνο και η εικόνα 4.1 εμφανίζει το αντίστοιχο γράφημα της.

## 4.2 Μελέτη απόδοσης παράλληλου αλγορίθμου

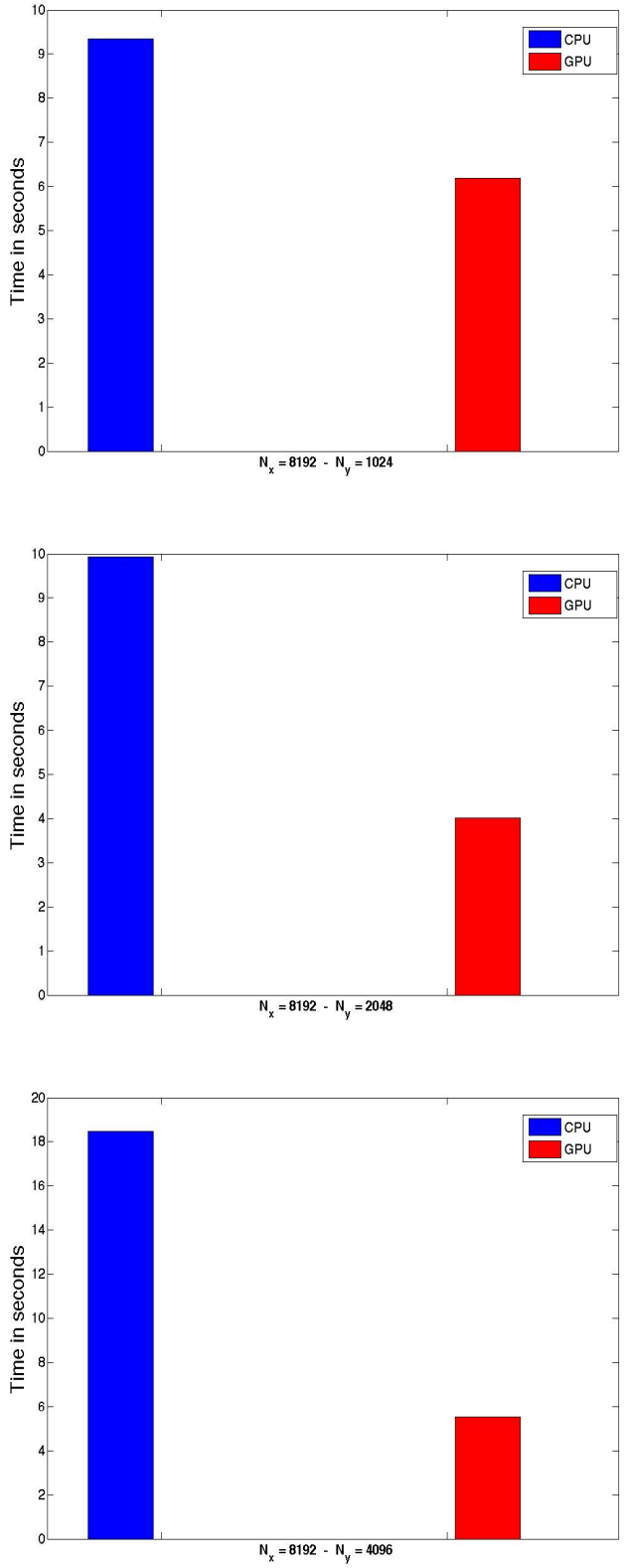
Η διερεύνηση της απόδοσης του αλγορίθμου στηρίχθηκε σε μετρήσεις του χρόνου εκτέλεσης της CPU υλοποίησης και ως προς αυτή της CPU-GPU. Επιλύθηκαν προβλήματα με διαφορετικές διακριτοποιήσεις ως προς την κάθε κατεύθυνση. Ο παρακάτω πίνακας περιέχει τις διακριτοποιήσεις για τις οποίες υπήρξε επιτάχυνση των υπολογισμών από τη χρήση μιας GPU, τη παράμετρο $\varepsilon$, το πλήθος των V-κύκλων καθώς και τις άπειρες νόρμες των σφαλμάτων επίλυσης του γραμμικού συστήματος και των λύσεων της ΜΔΕ.

| $N_x$ | $N_y$ | $\epsilon$ | V-cycles | $\|\|b - Au_n\|\|_\infty$ | $\|\|u - u_n\|\|_\infty$ |
|---|---|---|---|---|---|
| 2048 | 1024 | $2.5000e-1$ | 7 | $4.36e-14$ | $1.61e-10$ |
| 4096 | 1024 | $6.2500e-2$ | 6 | $4.43e-13$ | $1.09e-11$ |
| 4096 | 2048 | $2.5000e-1$ | 7 | $1.77e-14$ | $1.01e-11$ |
| 8192 | 1024 | $1.5625e-2$ | 11 | $1.99e-13$ | $4.16e-12$ |
| 8192 | 2048 | $6.2500e-2$ | 6 | $1.91e-13$ | $1.01e-11$ |
| 8192 | 4096 | $2.5000e-1$ | 6 | $2.62e-13$ | $9.64e-12$ |
| 16384 | 1024 | $3.9063e-3$ | 13 | $1.92e-13$ | $1.91e-11$ |
| 16384 | 2048 | $1.5625e-2$ | 11 | $1.99.e-13$ | $4.25e-12$ |
| 32768 | 1024 | $9.7656e-4$ | 14 | $1.63e-13$ | $7.44e-11$ |

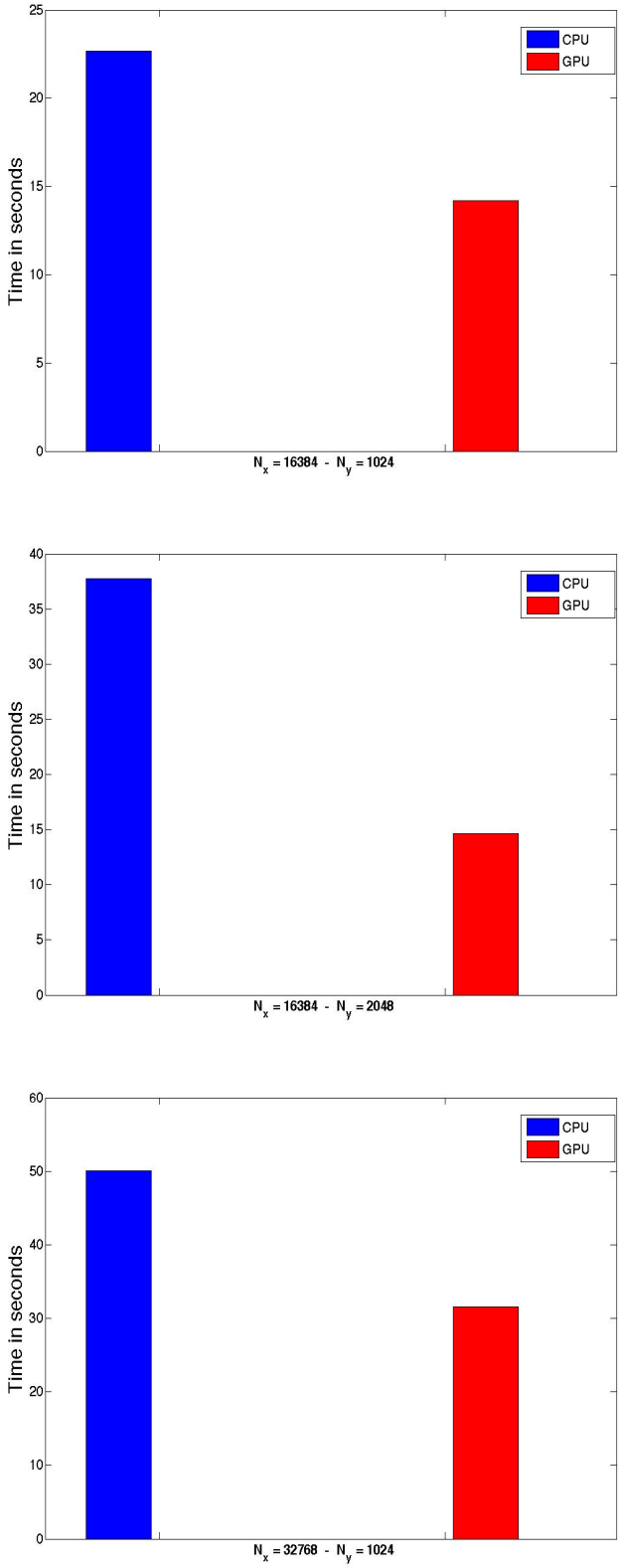
Τα επόμενα γραφήματα εμφανίζουν τους χρόνους στις δυο υπολογιστικές αρχιτεκτονικές.



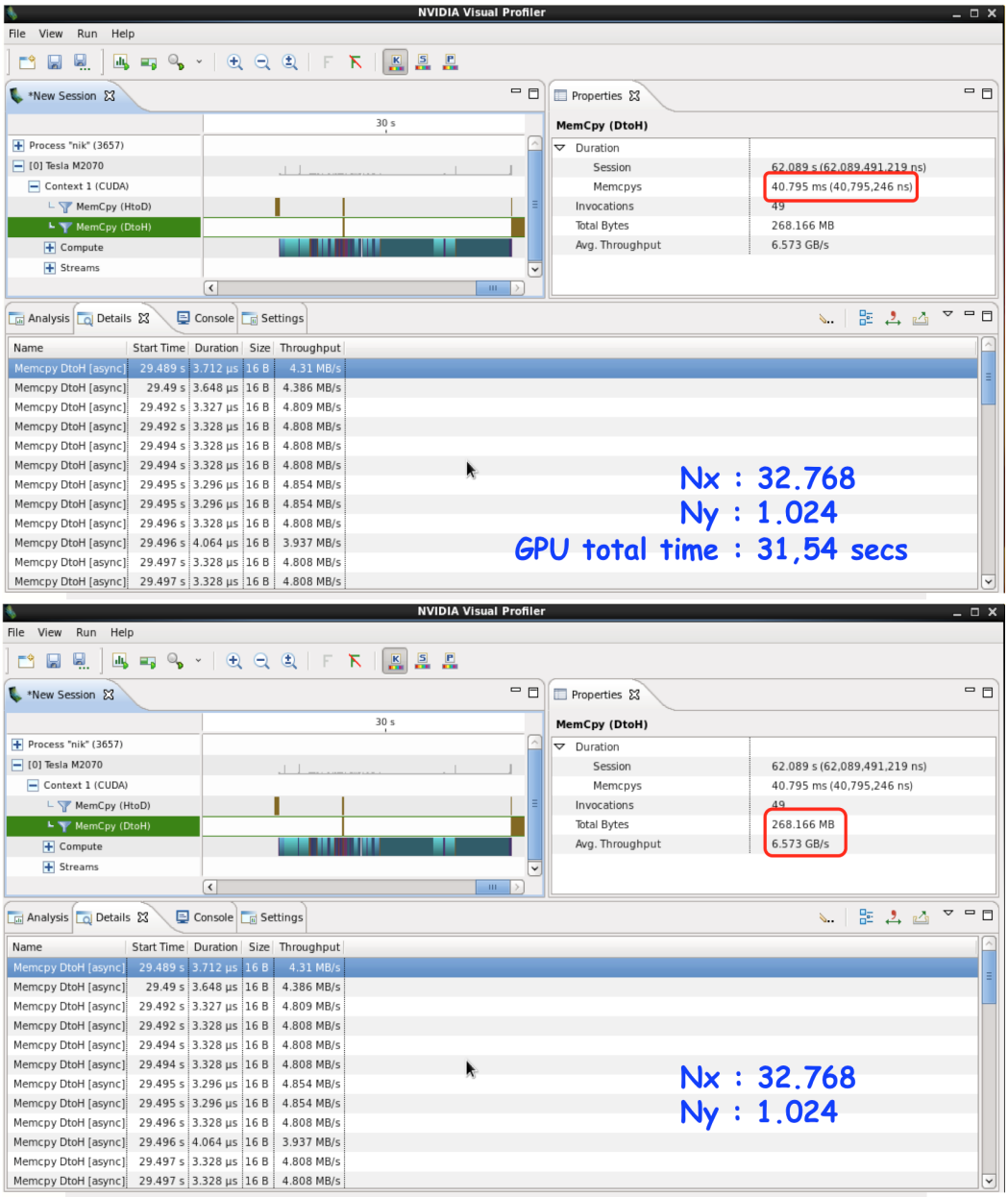

**Σχήμα 4.2:** Χρόνοι επίλυσης σε CPU και GPU για επιλεγμένες διακριτοποιήσεις.




**Σχήμα 4.3:** Χρόνοι επίλυσης σε CPU και GPU για επιλεγμένες διακριτοποιήσεις.




**Σχήμα 4.4:** Χρόνοι επίλυσης σε CPU και GPU για επιλεγμένες διακριτοποιήσεις.




**Σχήμα 4.5:** Χρόνοι μεταφοράς και όγκος δεδομένων μεταξύ CPU και GPU.

Ο παρακάτω πίνακας εμφανίζει την επιτάχυνση (speedup) που σημειώθηκε για καθένα από τα προβλήματα δοκιμών.

| $N_x$ | $N_y$ | $\epsilon$ | speedup |
|---|---|---|---|
| 2048 | 1024 | $2.5000e-1$ | 1.19 |
| 4096 | 1024 | $6.2500e-2$ | 1.43 |
| 4096 | 2048 | $2.5000e-1$ | 2.20 |
| 8192 | 1024 | $1.5625e-2$ | 1.51 |
| 8192 | 2048 | $6.2500e-2$ | 2.47 |
| 8192 | 4096 | $2.5000e-1$ | 3.34 |
| 16384 | 1024 | $3.9063e-3$ | 1.60 |
| 16384 | 2048 | $1.5625e-2$ | 2.58 |
| 32768 | 1024 | $9.7656e-4$ | 1.59 |

Ο παράλληλος αλγόριθμος έχει σχεδιαστεί ώστε το σύνολο των πράξεων να διεξάγονται στο GPU με αποτέλεσμα να απαιτείται η μεταφορά των δεδομένων από το CPU στο GPU στην αρχή, ενώ των αποτελεσμάτων με αντίστροφη πορεία στο τέλος. Το υπολογιστικό κόστος της μεταφοράς, ανά κατεύθυνση, για το πρόβλημα διακριτοποίησης $N_x = 32.768$ και $N_y = 1.024$ μετρήθηκε στα 0,041 seconds για συνολικό χρόνο εκτέλεσης 31,54 seconds. Σε αυτή την περίπτωση μεταφέρθηκαν περίπου 268 MB δεδομένα, με ρυθμό μεταφοράς 6,573 GB ανά δευτερόλεπτο, μέσω του διαύλου PCI-Express. Για την μέτρηση του υπολογιστικού κόστους της εφαρμογής χρησιμοποιήθηκε το λογισμικό NVIDIA Visual Profiler, το οποίο έχει τη δυνατότητα της γραφικής απόδοσης της υλοποίησης κάθε προβλήματος δοκιμής. Οι μετρήσεις στην εικόνα 4.5 εμφανίζουν τη χρήση αυτού του λογισμικού για αυτές τις διακριτοποιήσεις.

Η καλύτερη απόδοση με επιτάχυνση πάνω από 3 εμφανίστηκε για διακριτοποίηση $N_x = 8.192$ και $N_y = 4.096$. Γενικότερα μεγαλύτερες αποδόσεις εμφανίζονται για τις περιπτώσεις που υπάρχουν πολλά επίπεδα εναλλαγής πλεγμάτων τα οποία απαιτούν την εφαρμογή τελεστών πολυπλέγματος ως προς και δύο χωρικές κατευθύνσεις. Δηλαδή, για προβλήματα τα οποία υπάρχει το ίδιο πλήθος υπολογιστικών κελιών ανά κατεύθυνση σε πολλά επίπεδα της ΤΠΠ. Αυτό συμβαίνει γιατί στις περιπτώσεις αυτές υπάρχει αυξημένο κόστος υπολογισμών από τη χρήση διδιάστατων τελεστών της ΤΠΠ.

# Κεφάλαιο 5

# Συμπεράσματα

Η Τεχνική Πολυπλέγματος είναι μια πολύ αποδοτική διαδικασία επίλυσης γραμμικών συστημάτων που προκύπτουν από μεθόδους διακριτοποίησης προβλημάτων συνοριακών τιμών. Το βασικό μειονέκτημα της αριθμητικής αυτής μεθόδου είναι η αδυναμία αποδοτικής υλοποίησης της σε σύγχρονες αρχιτεκτονικές διεξαγωγής παράλληλων υπολογισμών. Αυτό συμβαίνει για δύο κύριους λόγους, οι οποίοι είναι ανεξάρτητοι από τον αλγόριθμο εφαρμογής της επαναληπτικής αυτής διαδικασίας. Ο πρώτος οφείλεται στο γεγονός της συνεχούς μεταβολής του μεγέθους του προς επίλυση προβλήματος, ενώ η πλήρη επίλυση του επιτυγχάνεται για ένα πολύ μικρού μεγέθος το οποίο αναφερεται σε ένα πολύ αραιό πλέγμα με αποτέλεσμα το υπολογιστικό κόστος να είναι μικρό. Ο δεύτερος λόγος είναι η σειριακότητα της τεχνικής αυτής εξαιτίας της κατασκευής των ιεραρχικών πλεγμάτων και της μετάβασης από το ένα στο άλλο.

Η εξέλιξη της τεχνολογίας από την άλλη έχει επιτρέψει στα παράλληλα μηχανήματα τη διεξαγωγή επιστημονικών υπολογισμών σε υποσυστήματα τα οποία αναλαμβάνουν το ρόλο των υπολογιστικών επιταχυντών. Τα υποσυστήματα αυτά μέχρι πριν λίγο καιρό απαιτούσαν εξιδεικευμένες γνώσεις προγραμματισμού για την πλήρη αξιοποίηση των παρεχόμενων δυνατοτήτων τους. Όμως σήμερα είναι εφικτή η ανάπτυξη εφαρμογών με τη χρήση πιο εύχρηστων εργαλείων όπως του προτύπου OpenACC. Σε κάθε περίπτωση απαιτείται η κατασκευή κατάλληλου αλγορίθμου και η οργάνωση των υπολογισμών ώστε να είναι αποδοτική η διεξαγωγή τους σε αυτά τα υποσυστήματα, τα οποία αποτελούνται

από εκατοντάδες υπολογιστικούς πυρήνες οργανομένους σε δεκάδες ομάδες, ενώ έχουν αυτόνομη κοινής χρήσης μνήμη διαφορετική από αυτή του υπολογιστικού συστήματος. Η επικοινωνία μεταξύ των δυο τύπων μνήμης είναι σχετικά αργή κι αυτό θα πρέπει να λαμβάνεται σοβαρά υπόψη στη κατασκευή του παράλληλου αλγορίθμου.

Σε αυτή τη διατριβή κατασκευάστηκε ένας παράλληλος αλγόριθμος της τεχνικής πολυπλέγματος για υπολογιστικά συστήματα με γραφικά υποσυστήματα πολλαπλών πυρήνων για την επίλυση γραμμικών συτημάτων που προκύπτουν από τη διακριτοποίηση τύπου Poisson προβλημάτων με διαφορετικού μεγέθους βήματα διακριτοποίησης ως προς κάθε χωρική κατεύθυνση. Η μέθοδος διακριτοποίησης βασίστηκε στην εφαρμογή συμπαγών πεπερασμένων διαφορών με αποτέλεσμα τη δημιουργία ενός καλά block δομημένου πίνακα συντελεστών των αγνώστων.

Τα αποτελέσματα που προκύπτουν από τη μελέτη της συμπεριφοράς υλοποίησης του αλγορίθμου μας οδήγησαν στο συμπέρασμα ότι ακόμα και τεχνικές με πολύ λίγα χαρακτηριστικά παραλληλοποίησης, όπως η τεχνική πολυπλέγματος, μπορούν να έχουν αποδοτικές υλοποιήσεις. Αυτό μπορεί να επιτευχθεί με κατάλληλες εφαρμογές τεχνικών ανεξαρτητοποίησης διεξαγωγής υπολογισμών σε συγκεκριμένες διαδικασίες, αλλά κυρίως οργανώνοντας κατάλληλα τη διεξαγωγή τους λαμβάνοντας υπόψη την αρχιτεκτονική των επιταχυντών. Για το πρώτο μπορεί να χρησιμοποιηθούν κλασσικές τεχνικές, όπως διχρωματικές μέθοδοι αρίθμησης αγνώστων και εξισώσεων, ενώ για το δεύτερο συχνά χρειάζεται να γίνει οργάνωση των υπολογισμών στο χαμηλότερο δυνατό επίπεδο με ταυτόχρονη διαχείριση των μεταφερόμενων δεδομένων μεταξύ της κύριας μνήμης του υπολογιστικού συστήματος και του επιταχυντή. Το πρότυπο OpenACC είναι σήμερα σε θέση να προσφέρει αποδοτικές υλοποιήσεις παράλληλων αλγορίθμων αυτού του είδους.

# Παράρτημα Α΄

# Κώδικας προγράμματος σε Fortran με χρήση του προτύπου OpenACC

## Α΄.1 Κύριο πρόγραμμα

```
      use openacc
      use omp_lib
      parameter(nx=32768,ny=1024,n=(nx-1)*(ny-1),
     +        lev=5,lev2d=8,lk=3*nx*ny)
      implicit real*8 (a-h,o-z)
      real*8 a0(2,nx-1),a1(2,nx-1),b(n),s(n),e(lk),r(lk),a,c,bp,tt,d,
     +     x(0:nx),y(0:ny),xo(n),xn(n),t(n),a0f(4,nx-1),l(nx-1),UL(nx-1)
      integer ipvt(nx-1),step,st,nx1,ny2,nw,nb,i

      iter=1000
      tol=1.0d-16
      vtol=1.0d-10
      rnorm=1.0d0
      step=1
      nx1=nx-1
      g=dfloat(ny)/nx
      print*,'                                                '
      print*,'________________________________________________'
      print*,'                                                '
      print*,'Nx= ',nx,'  Ny= ',ny

      call acc_init(acc_device_nvidia)
      call makeb(nx,ny,xo,x,y)
      call zebra(nx,ny,xo,b)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      GPU Region
C
```

```
       tm0=omp_get_wtime()
!$acc data copyin(b(1:n)),copyout(xn(1:n)),
!$acc& create(s(1:n),e(1:lk),l(1:nx-1),ul(1:nx-1),xo(1:n),
!$acc&        t(1:n),r(1:lk))


c      start V-cycle
c      down
!$acc kernels
!$acc loop gang (1024) vector(32)
        do i=1,n
          xo(i)=0.0d0
        enddo
!$acc loop vector(32)
        do i=1,n
          xn(i)=0.0d0
        enddo
!$acc end kernels

       st=0
        do while ((st.lt.step).and.(rnorm.gt.vtol))
       st=st+1
        iter=1
        tol=5.0d-7

c     Smooth with Gauss-Seidel
       nx1=nx-1
       ny2=int(ny/2)
       nw=int(ny/2)*nx1
       nb=(int(ny/2)-1)*nx1
       c=5.0d0*g*g-1.0d0
       d=(1.0d0+g*g)*0.5d0
       bp=5.0d0-g*g
       a=10.0d0*(1.0d0+(g*g))
!$acc   kernels

       tt=(-bp*bp/a)+a
!$acc loop vector(32) seq
      do i=1,nx1-1
       if (i.eq.1) then
        l(i)=-bp/a
       elseif (i.eq.2) then
        l(i)=-bp/tt
       else
        tt=a-bp*bp/tt
        l(i)=-bp/tt
       endif
```

```
      enddo

       tt=(-bp**2.0d0/a)+a
!$acc loop vector(32) seq
      do i=1,nx1
       if (i.eq.1) then
        ul(i)=-a
       elseif (i.eq.2) then
        ul(i)=(bp*bp/a)-a
       else
        ul(i)=(bp*bp/tt)-a
        tt=a-bp**2/tt
       endif
      enddo

!$acc loop vector(32)
       do i=1,nx1
        if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
        elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
        else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
        endif
       enddo

!$acc loop gang(1024) independent
       do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +               d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(64)
        do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +               d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +               +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
        enddo
         t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +               d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
       enddo

         k=ny2*nx1-nx1
         j=(ny2-1)*nx1-nx1
!$acc loop independent
       do i=1,nx1
        if (i.eq.1) then
         t((ny2-1)*nx1+i)=c*xo(nw+(ny2-2)*nx1+i)+d*xo(nw+(ny2-2)*nx1+i+1)
        elseif (i.eq.nx1) then
```

```
          t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
         else
          t((ny2-1)*nx1+i)=c*xo(nw+(ny2-2)*nx1+i)+
     +             d*(xo(nw+(ny2-2)*nx1+i+1)+xo(nw+(ny2-2)*nx1+i-1))
         endif
        enddo

!$acc loop independent
        do i=1,nw
         t(i)=b(i)-t(i)
        enddo
!$acc  end kernels

!$acc   kernels
!$acc loop independent
        do k=1,ny2
          do i=1,nx1
           if (i.eq.1) then
             s((k-1)*nx1+i)=t((k-1)*nx1+i)
           else
             s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
           endif
          enddo
          do i=nx1,1,-1
           if (i.eq.nx1) then
            xn(k*i)=s(k*i)/ul(i)
           else
            xn((k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*xn((k-1)*nx1+i+1))/ul(i)
           endif
          enddo
       enddo
!$acc  end kernels

!$acc   kernels
!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          t((k-1)*nx1+1)=c*(xn(k*nx1+1)+xn((k-1)*nx1+1))+
     +                 d*((xn(k*nx1+2)+xn((k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t((k-1)*nx1+i)=c*(xn(k*nx1+i)+xn((k-1)*nx1+i))+
     +                 d*(xn(k*nx1+i+1)+xn((k-1)*nx1+i+1)
     +                 +xn(k*nx1+i-1)+xn((k-1)*nx1+i-1))
         enddo
          t(k*nx1)=c*(xn((k+1)*nx1)+xn(k*nx1))+
     +                 d*(xn((k+1)*nx1-1)+xn(k*nx1-1))
        enddo
```

```
!$acc loop independent
       do i=1,nb
        t(i)=b(nw+i)-t(i)
       enddo

!$acc loop vector(32) independent
       do k=1,ny2-1
         s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop seq
        do i=2,nx1
         s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
        enddo
         xn(nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop seq
        do i=nx1-1,1,-1
         xn(nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*xn(nw+(k-1)
     &                      *nx1+i+1))/ul(i)
        enddo
      enddo
!$acc  end kernels

!$acc   kernels
       iter=it
       tol=res

c    Evaluate Residual
!$acc loop vector(32)
       do i=1,nx1
        if (i.eq.1) then
         r(i)=c*xn(nw+i)+d*xn(nw+1+i)
        elseif (i.eq.nx1) then
         r(i)=c*xn(nw+i)+d*xn(nw+i-1)
        else
         r(i)=c*xn(nw+i)+d*(xn(nw+i+1)+xn(nw+i-1))
        endif
       enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          r(k*nx1+1)=c*(xn(nw+k*nx1+1)+xn(nw+(k-1)*nx1+1))+
     +                d*((xn(nw+k*nx1+2)+xn(nw+(k-1)*nx1+2)))
!$acc loop vector(32) independent
         do i=2,nx1-1
          r(k*nx1+i)=c*(xn(nw+k*nx1+i)+xn(nw+(k-1)*nx1+i))+
     +                d*(xn(nw+k*nx1+i+1)+xn(nw+(k-1)*nx1+i+1)
     +                +xn(nw+k*nx1+i-1)+xn(nw+(k-1)*nx1+i-1))
```

```
          enddo
           r((k+1)*nx1)=c*(xn(nw+(k+1)*nx1)+xn(nw+k*nx1))+
     +                  d*(xn(nw+(k+1)*nx1-1)+xn(nw+k*nx1-1))
        enddo

!$acc loop vector(32) independent
         do i=1,nx1
          if (i.eq.1) then
           r(nw-nx1+i)=c*xn(2*nw-2*nx1+i)+d*xn(2*nw-2*nx1+i+1)
          elseif (i.eq.nx1) then
           r(ny2*i)=c*xn(nw+(ny2-1)*i)+d*xn(nw+(ny2-1)*i-1)
          else
           r(nw-nx1+i)=c*xn(2*nw-2*nx1+i)+d*(xn(2*nw-2*nx1+i+1)+
     +                 xn(2*nw-2*nx1+i-1))
          endif
         enddo

!$acc loop vector(32) independent
         do k=ny2-1,1,-1
           r(nw+(k-1)*nx1+1)=c*(xn(k*nx1+1)+xn((k-1)*nx1+1))+
     +                  d*((xn(k*nx1+2)+xn((k-1)*nx1+2)))
!$acc loop vector(32)
          do i=2,nx1-1
           r(nw+(k-1)*nx1+i)=c*(xn(k*nx1+i)+xn((k-1)*nx1+i))+
     +                  d*(xn(k*nx1+i+1)+xn((k-1)*nx1+i+1)
     +                  +xn(k*nx1+i-1)+xn((k-1)*nx1+i-1))
          enddo
           r(nw+k*nx1)=c*(xn((k+1)*nx1)+xn(k*nx1))+
     +                  d*(xn((k+1)*nx1-1)+xn(k*nx1-1))
         enddo

!$acc loop vector(32) independent
        do k=1,ny2
          r((k-1)*nx1+1)=r((k-1)*nx1+1)+bp*xn((k-1)*nx1+2)-a*xn((k-1)*nx1+1)
!$acc loop vector(32)
         do i=2,nx1-1
          r((k-1)*nx1+i)=r((k-1)*nx1+i)+bp*(xn((k-1)*nx1+i-1)+
     +                 xn((k-1)*nx1+i+1))-a*xn((k-1)*nx1+i)
         enddo
          r(k*nx1)=r(k*nx1)+bp*xn(k*nx1-1)-a*xn(k*nx1)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-1
          r(nw+(k-1)*nx1+1)=r(nw+(k-1)*nx1+1)+bp*xn(nw+(k-1)*nx1+2)
     +                     -a*xn(nw+(k-1)*nx1+1)
!$acc loop vector(32)
```

```
         do i=2,nx1-1
          r(nw+(k-1)*nx1+i)=r(nw+(k-1)*nx1+i)+bp*(xn(nw+(k-1)*nx1+i-1)+
     +                xn(nw+(k-1)*nx1+i+1))-a*xn(nw+(k-1)*nx1+i)
         enddo
          r(nw+k*nx1)=r(nw+k*nx1)+bp*xn(nw+k*nx1-1)-a*xn(nw+k*nx1)
        enddo

!$acc loop vector(32)
       do i=1,nw+nb
        r(i)=b(i)-r(i)
       enddo

!$acc loop vector(32)
        do i=1,lk
         e(i)=0.0d0
        enddo

         nxl=nx
         levk=1
!$acc loop vector(32)
         do i=1,n
           xo(i)=r(i)
         enddo
!$acc  end kernels


c    Multigrid (down)
c    Restrict Residual with semi-coarsening
         ilev=0
        do while (ilev.le.lev-1)
!$acc  kernels
         ileng=(int((nx/(2**ilev)))-1)*(ny-1)
         nx1=nxl-1
         ny1=ny-1
         nx12=int((nx1-1)/2)
         le=levk+ileng

!$acc loop vector(32) independent
         do i=1,ny1
!$acc loop vector(32) independent
          do j=1,nx12
            m=nx1*(i-1)+2*j
            r(le+(i-1)*nx12+j)=2.0d0*xo(m)+xo(m-1)+xo(m+1)
          enddo
         enddo

         nxl=int(nxl/2)
```

```
        iter=1
        if ((lev2d.eq.0).and.(ilev.eq.lev-1)) iter=100000
        tol=5.0d-15
        ileng2=(int((nx/(2**(ilev+1))))-1)*(ny-1)

!$acc loop vector(32)
         do i=1,ileng2
            xo(i)=e(levk+ileng+i)
         enddo

!$acc end  kernels
c     Smooth with Gauss-Seidel
       nx1=nxl-1
!$acc kernels
       na=(nxl-1)*(ny-1)
       dx=1.0d0/dfloat(nxl)
       ny2=int(ny/2)
       nw=int(ny/2)*nx1
       nb=(int(ny/2)-1)*nx1
       dy=1.0d0/dfloat(ny)
       gg=dx/dy
       c=5.0d0*gg*gg-1.0d0
       d=(1.0d0+gg*gg)*0.5d0
       bp=5.0d0-gg*gg
       a=10.0d0*(1.0d0+(gg*gg))
       li=levk+ileng

!$acc loop seq
       do i=1,nx1-1
        if (i.eq.1) then
         l(i)=-bp/a
        elseif (i.eq.2) then
         tt=(-bp*bp/a)+a
         l(i)=-bp/tt
        else
         tt=a-bp*bp/tt
         l(i)=-bp/tt
        endif
       enddo

!$acc loop vector(32) seq
       do i=1,nx1
        if (i.eq.1) then
         ul(i)=-a
        elseif (i.eq.2) then
         ul(i)=(bp*bp/a)-a
         tt=(-bp*bp/a)+a
```

```
         else
          ul(i)=(bp*bp/tt)-a
          tt=a-bp*bp/tt
         endif
        enddo

        it=0
101     it=it+1
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
         elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
         else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +                d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +                d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +                +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
         enddo
          t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +                d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
        enddo
          k=ny2*nx1-nx1
          j=(ny2-1)*nx1-nx1

!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
         elseif (i.eq.nx1) then
          t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
         else
          t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
         endif
        enddo

!$acc loop vector(32)
```

```
       do i=1,nw
        t(i)=r(li+i)-t(i)
       enddo

!$acc loop vector(32) independent
       do k=1,ny2
         s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
        do i=2,nx1
         s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
        enddo
         e(li+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
        do i=nx1-1,1,-1
         e(li+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+(k-1)*
     &                      nx1+i+1))/ul(i)
        enddo
       enddo

!$acc loop vector(32) independent
       do k=ny2-1,1,-1
         t((k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                  d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32)
        do i=2,nx1-1
         t((k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                  d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                  +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
        enddo
         t(k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                  d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
       enddo

!$acc loop vector(32)
       do i=1,nb
        t(i)=r(li+nw+i)-t(i)
       enddo

!$acc loop vector(128) independent
       do k=1,ny2-1
         s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop seq
        do i=2,nx1
         s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
        enddo
         e(li+nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop seq
```

```
         do i=nx1-1,1,-1
          e(li+nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+nw+(k-1)*
     &                          nx1+i+1))/ul(i)
         enddo
        enddo

         iter=it
         tol=res
c       Evaluate Residual
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          xo(i)=c*e(li+nw+i)+d*e(li+nw+i+1)
         elseif (i.eq.nx1) then
          xo(i)=c*e(li+nw+i)+d*e(li+nw+i-1)
         else
          xo(i)=c*e(li+nw+i)+d*(e(li+nw+i+1)+e(li+nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          xo(k*nx1+1)=c*(e(li+nw+k*nx1+1)+e(li+nw+(k-1)*nx1+1))+
     +                 d*((e(li+nw+k*nx1+2)+e(li+nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          xo(k*nx1+i)=c*(e(li+nw+k*nx1+i)+e(li+nw+(k-1)*nx1+i))+
     +                 d*(e(li+nw+k*nx1+i+1)+e(li+nw+(k-1)*nx1+i+1)
     +                 +e(li+nw+k*nx1+i-1)+e(li+nw+(k-1)*nx1+i-1))
         enddo
          xo((k+1)*nx1)=c*(e(li+nw+(k+1)*nx1)+e(li+nw+k*nx1))+
     +                 d*(e(li+nw+(k+1)*nx1-1)+e(li+nw+k*nx1-1))
        enddo

          k=nw-nx1
          k1=k-nx1
!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          xo(k+i)=c*e(li+nw+k1+i)+d*e(li+nw+k1+i+1)
         elseif (i.eq.nx1) then
          xo(ny2*i)=c*e(li+nw+(ny2-1)*i)+d*e(li+nw+(ny2-1)*i-1)
         else
          xo(k+i)=c*e(li+nw+k1+i)+d*(e(li+nw+k1+i+1)+e(li+nw+k1+i-1))
         endif
        enddo
```

```
!$acc loop vector(32) independent
       do k=ny2-1,1,-1
         xo(nw+(k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                 d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32) independent
        do i=2,nx1-1
         xo(nw+(k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                 d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                 +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
        enddo
         xo(nw+k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                 d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
       enddo

!$acc loop vector(32) independent
       do k=1,ny2
         xo((k-1)*nx1+1)=xo((k-1)*nx1+1)+bp*e(li+(k-1)*nx1+2)-
     +               a*e(li+(k-1)*nx1+1)
!$acc loop vector(32)
        do i=2,nx1-1
         xo((k-1)*nx1+i)=xo((k-1)*nx1+i)+bp*(e(li+(k-1)*nx1+i-1)+
     +                e(li+(k-1)*nx1+i+1))-a*e(li+(k-1)*nx1+i)
        enddo
         xo(k*nx1)=xo(k*nx1)+bp*e(li+k*nx1-1)-a*e(li+k*nx1)
       enddo

!$acc loop vector(32) independent
       do k=1,ny2-1
         xo(nw+(k-1)*nx1+1)=xo(nw+(k-1)*nx1+1)+bp*e(li+nw+(k-1)*nx1+2)
     +                    -a*e(li+nw+(k-1)*nx1+1)
!$acc loop vector(32) independent
        do i=2,nx1-1
         xo(nw+(k-1)*nx1+i)=xo(nw+(k-1)*nx1+i)+bp*(e(li+nw+(k-1)*nx1+i-1)+
     +                e(li+nw+(k-1)*nx1+i+1))-a*e(li+nw+(k-1)*nx1+i)
        enddo
         xo(nw+k*nx1)=xo(nw+k*nx1)+bp*e(li+nw+k*nx1-1)-a*e(li+nw+k*nx1)
       enddo
!$acc end kernels

!$acc kernels
        k=nw+nb
!$acc  loop
       do i=1,k
        ii=li+i
        xo(i)=r(ii)-xo(i)
       enddo
```

```
        levk=levk+ileng
        ilev=ilev+1
!$acc end kernels
       enddo

        nyl=ny
        nx2d=nxl
        ny2d=ny
c    Multigrid (down)
c    Restrict Residual with full-coarsening
       do ilev=0,lev2d-1
        nx2=int(nx2d/2)-1
        ny2=int(ny2d/2)-1
        nx1=nx2d-1
        d4=1.0d0/4.0d0
        li=levk+ileng
        nbb=int(ny2d/2)*nx1
        ny22=int((ny2+1)/2)
!$acc kernels

!$acc loop vector(32) independent
        do i=1,ny22
!$acc loop vector(32)
         do j=1,nx2
          nw=(i-1)*nx1+2*(j-1)+(i-1)*(nx2d-1)+1
          r(li+(i-1)*nx2+j)=d4*(xo(nw)+2.0d0*xo(nw+1)+xo(nw+2)+xo(nx1+nw)
     &                +2.0d0*xo(nx1+nw+1)+xo(nx1+nw+2)+2.0d0*xo(nbb+nw)
     &                +4.0d0*xo(nbb+nw+1)+2.0d0*xo(nbb+nw+2))
         enddo
        enddo
!$acc end kernels

          nwww=ny22*nx2
          nw=nx2d-1
          nwk=nx2d
          nb=int(ny2d/2)*nx1+nx2d
          nww=int(ny2d/2)*nx1+nx2d-1
!$acc kernels
!$acc loop vector(32) independent
        do i=1,ny22-1
!$acc loop vector(32)
         do j=1,nx2
           nbb=(i-1)*nx1+2*(j-1)+(i-1)*(nx2d-1)+1
           r(li+nwww+(i-1)*nx2+j)=d4*(xo(nbb+nw)+2.0d0*xo(nbb+nw+1)+
     &        xo(nbb+nw+2)+xo(nx1+nbb+nw)+2.0d0*xo(nx1+nbb+nw+1)
     &        +xo(nx1+nbb+nw+2)+2.0d0*xo(nbb+nww)
     &                +4.0d0*xo(nww+nbb+1)+2.0d0*xo(nww+nbb+2))
```

```
         enddo
        enddo
!$acc end kernels

         ny2d=int(ny2d/2)
         nx2d=int(nx2d/2)
         iter=1
         if (ilev.eq.lev2d-1) iter=100000
         tol=1.0d-8
         ileng2=(int((nxl/(2**(ilev+1))))-1)*(int((nyl/(2**(ilev+1)))-1))
!$acc kernels
!$acc loop vector(32)
        do i=1,ileng2
         xo(i)=e(levk+ileng+i)
        enddo

c     Smooth with Gauss-Seidel
         nx1=nx2d-1
         na=(nx2d-1)*(ny2d-1)
         ny2=int(ny2d/2)
         nw=int(ny2d/2)*nx1
         nb=(int(ny2d/2)-1)*nx1
         dx=1.0d0/dfloat(nx2d)
         dy=1.0d0/dfloat(ny2d)
         gg=dx/dy
         c=5.0d0*gg*gg-1.0d0
         d=(1.0d0+gg*gg)*0.5d0
         bp=5.0d0-gg*gg
         a=10.0d0*(1.0d0+(gg*gg))
         li=levk+ileng

!$acc loop vector(32) seq
        do i=1,nx1-1
         if (i.eq.1) then
          l(i)=-bp/a
         elseif (i.eq.2) then
          tt=(-bp**2/a)+a
          l(i)=-bp/tt
         else
          tt=a-bp**2/tt
          l(i)=-bp/tt
         endif
        enddo

!$acc loop vector(32) seq
        do i=1,nx1
         if (i.eq.1) then
```

```
          ul(i)=-a
         elseif (i.eq.2) then
          ul(i)=(bp**2/a)-a
          tt=(-bp**2/a)+a
         else
          ul(i)=(bp**2/tt)-a
          tt=a-bp**2/tt
         endif
        enddo
!$acc end kernels

        it=0
        res=1.0d0
        if (ilev.eq.lev2d-1) iter=100000
       do while ((it.lt.iter).and.(tol.lt.res))
        it=it+1
!$acc kernels

!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
         elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
         else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +                d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +                d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +                +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
         enddo
          t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +                d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
        enddo

         k=ny2*nx1-nx1
         j=(ny2-1)*nx1-nx1
!$acc loop vector(32) independent
        do i=1,nx1
```

```
          if (i.eq.1) then
           t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
          elseif (i.eq.nx1) then
           t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
          else
           t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
          endif
         enddo

!$acc loop vector(32)
        do i=1,nw
         t(i)=r(li+i)-t(i)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          e(li+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          e(li+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+(k-1)*
     &                      nx1+i+1))/ul(i)
         enddo
        enddo

!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          t((k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                 d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t((k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                 d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                 +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
         enddo
          t(k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                 d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
        enddo

!$acc loop vector(32)
        do i=1,nb
         t(i)=r(li+nw+i)-t(i)
        enddo
```

```
!$acc loop vector(32) independent
        do k=1,ny2-1
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          e(li+nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          e(li+nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+nw+(k-1)*
     &                          nx1+i+1))/ul(i)
         enddo
        enddo

!$acc loop vector(32)
        do i=1,na
          ii=li+i
          xo(i)=e(ii)-xo(i)
        enddo
!$acc end kernels

!$acc kernels
         xm=0.0d0
!$acc loop reduction(max:xm)
        do i=1,na
         xm=dmax1(xm,dabs(e(li+i)))
        enddo

         xma=0.0d0
!$acc loop reduction(max:xma)
        do i=1,na
         xma=dmax1(xma,dabs(xo(i)))
        enddo
         res=xma/xm
!$acc end kernels

!$acc  kernels
        if((it.lt.iter).and.(tol.lt.res)) then
!$acc loop vector(32)
         do i=1,na
          xo(i)=e(li+i)
         enddo
        endif
!$acc end kernels
       enddo
```

```
         iter=it
         tol=res
c      Smooth with Gauss-Seidel
!$acc  kernels
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          xo(i)=c*e(li+nw+i)+d*e(li+nw+i+1)
         elseif (i.eq.nx1) then
          xo(i)=c*e(li+nw+i)+d*e(li+nw+i-1)
         else
          xo(i)=c*e(li+nw+i)+d*(e(li+nw+i+1)+e(li+nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          xo(k*nx1+1)=c*(e(li+nw+k*nx1+1)+e(li+nw+(k-1)*nx1+1))+
     +                d*((e(li+nw+k*nx1+2)+e(li+nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          xo(k*nx1+i)=c*(e(li+nw+k*nx1+i)+e(li+nw+(k-1)*nx1+i))+
     +                d*(e(li+nw+k*nx1+i+1)+e(li+nw+(k-1)*nx1+i+1)
     +                +e(li+nw+k*nx1+i-1)+e(li+nw+(k-1)*nx1+i-1))
         enddo
          xo((k+1)*nx1)=c*(e(li+nw+(k+1)*nx1)+e(li+nw+k*nx1))+
     +                d*(e(li+nw+(k+1)*nx1-1)+e(li+nw+k*nx1-1))
        enddo

         k=nw-nx1
         k1=k-nx1
!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          xo(k+i)=c*e(li+nw+k1+i)+d*e(li+nw+k1+i+1)
         elseif (i.eq.nx1) then
          xo(ny2*i)=c*e(li+nw+(ny2-1)*i)+d*e(li+nw+(ny2-1)*i-1)
         else
          xo(k+i)=c*e(li+nw+k1+i)+d*(e(li+nw+k1+i+1)+e(li+nw+k1+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          xo(nw+(k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
```

```
!$acc loop vector(32) independent
         do i=2,nx1-1
          xo(nw+(k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
         enddo
          xo(nw+k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
        enddo

!$acc loop vector(32) independent
        do k=1,ny2
          xo((k-1)*nx1+1)=xo((k-1)*nx1+1)+bp*e(li+(k-1)*nx1+2)-
     +                a*e(li+(k-1)*nx1+1)
!$acc loop vector(32)
         do i=2,nx1-1
          xo((k-1)*nx1+i)=xo((k-1)*nx1+i)+bp*(e(li+(k-1)*nx1+i-1)+
     +               e(li+(k-1)*nx1+i+1))-a*e(li+(k-1)*nx1+i)
         enddo
          xo(k*nx1)=xo(k*nx1)+bp*e(li+k*nx1-1)-a*e(li+k*nx1)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-1
          xo(nw+(k-1)*nx1+1)=xo(nw+(k-1)*nx1+1)+bp*e(li+nw+(k-1)*nx1+2)-
     +                    a*e(li+nw+(k-1)*nx1+1)
!$acc loop vector(32) independent
         do i=2,nx1-1
          xo(nw+(k-1)*nx1+i)=xo(nw+(k-1)*nx1+i)+bp*(e(li+nw+(k-1)*nx1+i-1)+
     +               e(li+nw+(k-1)*nx1+i+1))-a*e(li+nw+(k-1)*nx1+i)
         enddo
          xo(nw+k*nx1)=xo(nw+k*nx1)+bp*e(li+nw+k*nx1-1)-a*e(li+nw+k*nx1)
        enddo
!$acc end kernels

!$acc kernels
         k=nw+nb
!$acc loop vector(32)
        do i=1,k
         ii=li+i
         xo(i)=r(ii)-xo(i)
        enddo
!$acc end kernels

         levk=levk+ileng
       enddo
```

```
c     Multigrid 2D (up)
c     Interpolate Error with full-coarsening
         nx2=int(2*nx2d)
         ny2=int(2*ny2d)
         nw=int(ny2d/2)*(nx2d-1)
         nb=(int(ny2d/2)-1)*(nx2d-1)
         d2=0.5d0
         d4=0.25d0
         nw2=int((nx2d-1)*(ny2-1))

!$acc kernels
!$acc loop vector(32) independent
        do i=1,nx2d-1
         if (i.lt.nx2d-1) then
          if (i.eq.1) xo(i)=d4*e(levk+i)
           xo(i*2)=d2*e(levk+i)
           xo(i*2+1)=d4*(e(levk+i)+e(levk+i+1))
         else
           xo(i*2)=d2*e(levk+i)
           xo(i*2+1)=d4*e(levk+i)
         endif
        enddo

!$acc loop vector(32) independent
        do iw=1,ny2d-2
          km=int(iw/2)
         if (mod(iw,2).eq.0) then
          kl=km-1
         else
          kl=km
         endif
          xo(nx2-1+(iw-1)*(nx2-1)+1)=d4*(e(levk+(iw-1)*(nx2d-1)+1-(kl*
     &       (nx2d-1)))+e(levk+nw+(iw-1)*(nx2d-1)+1-km*(nx2d-1)))
!$acc loop vector(32)
         do i=1,nx2d-2
          xo(nx2-1+(iw-1)*(nx2-1)+2*i)=d2*(e(levk+nw+(iw-1)*
     &       (nx2d-1)+i-km*(nx2d-1))+e(levk+(iw-1)*(nx2d-1)+i-(kl*
     &       (nx2d-1))))
          xo(nx2-1+(iw-1)*(nx2-1)+2*i+1)=d4*(e(levk+(iw-1)*
     &       (nx2d-1)+i-(kl*(nx2d-1)))+e(levk+(iw-1)*(nx2d-1)+i+1-(kl*
     &       (nx2d-1)))+e(levk+nw+(iw-1)*(nx2d-1)+i-km*(nx2d-1))+
     &       e(levk+nw+(iw-1)*(nx2d-1)+i+1-km*(nx2d-1)))
         enddo
          xo(nx2-1+(iw-1)*(nx2-1)+2*(nx2d-1))=d2*(e(levk+(iw-1)
     &       *(nx2d-1)+nx2d-1-(kl*(nx2d-1)))+e(levk+nw+(iw-1)*(nx2d-1)+
     &         nx2d-1-km*(nx2d-1)))
          xo(nx2-1+(iw-1)*(nx2-1)+2*(nx2d-1)+1)=d4*(e(levk+(iw-1)
```

```
     &       *(nx2d-1)+nx2d-1-(kl*(nx2d-1)))+e(levk+nw+(iw-1)*(nx2d-1)+
     &         nx2d-1-km*(nx2d-1)))
        enddo
!$acc end kernels

!$acc kernels
!$acc loop independent
        do i=1,nx2d-1
         if (i.lt.nx2d-1) then
          if (i.eq.1) xo(nw2+i)=d4*e(levk+nw-nx2d+i+1)
           xo(nw2+2*i)=d2*e(levk+nw-nx2d+1+i)
           xo(nw2+2*i+1)=d4*(e(levk+nw-nx2d+1+i)+e(levk+nw-nx2d+2+i))
         else
           xo(nw2+2*i)=d2*e(levk+nw)
           xo(nw2+2*i+1)=d4*e(levk+nw)
         endif
        enddo
!$acc end kernels

!$acc kernels
!$acc loop vector(32) independent
        do i=1,ny2d-1
           if (mod(i,2).eq.0) then
             km=(int(i/2)-1)*(nx2d-1)+nw
           else
             km=int(i/2)*(nx2d-1)
           endif
            xo(nw2+nx2+(i-1)*(nx2-1))=d2*e(levk+1+km)
!$acc loop vector(32)
         do iw=1,nx2d-2
            xo(nw2+nx2+(i-1)*(nx2-1)+iw+(iw-1))=e(levk+iw+km)
            xo(nw2+nx2+(i-1)*(nx2-1)+iw+(iw-1)+1)=d2*(e(levk+iw+km)+
     &          e(levk+iw+1+km))
         enddo
            xo(nw2+nx2+(i-1)*(nx2-1)+nx2d-1+nx2d-2)=e(levk+km+nx2d-1)
            xo(nw2+nx2+(i-1)*(nx2-1)+nx2d-1+nx2d-1)=d2*e(levk+km+nx2d-1)
        enddo

!$acc loop vector(32) independent
        do i=1,ileng
         xo(i)=xo(i)+e(levk-ileng+i)
        enddo
!$acc end kernels

       do ilev=lev2d,2,-1
         nx2d=int(nx2d*2)
         ny2d=int(ny2d*2)
```

```
          nx2d2=int(nx2d*2)
          ny2d2=int(ny2d*2)
          ileng2=(ny2d2-1)*(nx2d2-1)
          iter=1
          ileng=(ny2d-1)*(nx2d-1)
          tol=1.0d-17

c     Smooth with Gauss-Seidel
          nx1=nx2d-1
          na=(nx2d-1)*(ny2d-1)
          ny2=int(ny2d/2)
          nw=int(ny2d/2)*nx1
          nb=(int(ny2d/2)-1)*nx1
          dx=1.0d0/dfloat(nx2d)
          dy=1.0d0/dfloat(ny2d)
          gg=dx/dy
          c=5.0d0*gg*gg-1.0d0
          d=(1.0d0+gg*gg)*0.5d0
          bp=5.0d0-gg*gg
          a=10.0d0*(1.0d0+(gg*gg))
          li=levk-ileng

!$acc kernels
!$acc loop vector(32) seq
         do i=1,nx1-1
          if (i.eq.1) then
           l(i)=-bp/a
          elseif (i.eq.2) then
           tt=(-bp**2/a)+a
           l(i)=-bp/tt
          else
           tt=a-bp**2/tt
           l(i)=-bp/tt
          endif
         enddo

!$acc loop vector(32) seq
         do i=1,nx1
          if (i.eq.1) then
           ul(i)=-a
          elseif (i.eq.2) then
           ul(i)=(bp**2/a)-a
           tt=(-bp**2/a)+a
          else
           ul(i)=(bp**2/tt)-a
           tt=a-bp**2/tt
          endif
```

```
        enddo

         it=0
!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
         elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
         else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +               d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +               d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +               +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
         enddo
          t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +               d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
        enddo

          k=ny2*nx1-nx1
          j=(ny2-1)*nx1-nx1
!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
         elseif (i.eq.nx1) then
          t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
         else
          t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do i=1,nw
         ii=li+i
         t(i)=r(ii)-t(i)
        enddo
```

```
!$acc loop vector(32) independent
        do k=1,ny2
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          e(li+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          e(li+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+(k-1)*
     &                      nx1+i+1))/ul(i)
         enddo
        enddo

!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          t((k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                 d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t((k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                 d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                 +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
         enddo
          t(k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                 d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
        enddo

!$acc loop vector(32)
        do i=1,nb
         t(i)=r(li+nw+i)-t(i)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-1
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          e(li+nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          e(li+nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+nw+(k-1)*
     &                         nx1+i+1))/ul(i)
         enddo
```

```
        enddo
!$acc end kernels

         iter=it
         tol=res
!$acc kernels
         levk=levk-ileng
!$acc loop vector(32)
        do i=1,ileng2
         xo(i)=e(levk-ileng2+i)
        enddo

c      Interpolate Error with full-coarsening
         nx2=int(2*nx2d)
         ny2=int(2*ny2d)
         nw=int(ny2d/2)*(nx2d-1)
         nb=(int(ny2d/2)-1)*(nx2d-1)
         nw2=int((nx2d-1)*(ny2-1))

!$acc loop vector(32)
        do i=1,nx2d-1
         if (i.eq.1) then
          s(i)=d4*e(li+i)
          s(i*2)=d2*e(li+i)
          s(i*2+1)=d4*(e(li+i)+e(li+i+1))
         elseif (i.eq.nx2d-1) then
          s(i*2)=d2*e(li+i)
          s(i*2+1)=d4*e(li+i)
         else
          s(i*2)=d2*e(li+i)
          s(i*2+1)=d4*(e(li+i)+e(li+i+1))
         endif
        enddo

!$acc loop vector(32) independent
        do iw=1,ny2d-2
          km=int(iw/2)
         if (mod(iw,2).eq.0) then
          kl=km-1
         else
          kl=km
         endif
          s(nx2-1+(iw-1)*(nx2-1)+1)=d4*(e(li+(iw-1)*(nx2d-1)+1-(kl*
     &      (nx2d-1)))+e(li+nw+(iw-1)*(nx2d-1)+1-km*(nx2d-1)))
!$acc loop vector(32) independent
         do i=1,nx2d-2
          s(nx2-1+(iw-1)*(nx2-1)+2*i)=d2*(e(li+nw+(iw-1)*
```

```
     &      (nx2d-1)+i-km*(nx2d-1))+e(li+(iw-1)*(nx2d-1)+i-(kl*
     &      (nx2d-1))))
           s(nx2-1+(iw-1)*(nx2-1)+2*i+1)=d4*(e(li+nw+(iw-1)*
     &      (nx2d-1)+i-km*(nx2d-1))+e(li+nw+(iw-1)*(nx2d-1)+i+1-km*
     &      (nx2d-1))+e(li+(iw-1)*(nx2d-1)+i-(kl*(nx2d-1)))+e(li+(iw-1)*
     &      (nx2d-1)+i+1-(kl*(nx2d-1))))
          enddo
           s(nx2-1+(iw-1)*(nx2-1)+2*(nx2d-1))=d2*(e(li+(iw-1)*
     &      (nx2d-1)+nx2d-1-(kl*(nx2d-1)))+e(li+nw+(iw-1)*(nx2d-1)+nx2d-
     &      1-km*(nx2d-1)))
           s(nx2-1+(iw-1)*(nx2-1)+2*(nx2d-1)+1)=d4*(e(li+(iw-1)*
     &      (nx2d-1)+nx2d-1-(kl*(nx2d-1)))+e(li+nw+(iw-1)*(nx2d-1)+nx2d-
     &      1-km*(nx2d-1)))
        enddo

!$acc loop vector(32) independent
        do i=1,nx2d-1
         if (i.eq.1) then
          s(nw2+i)=d4*e(li+nw-nx2d+i+1)
          s(nw2+2*i)=d2*e(li+nw-nx2d+1+i)
          s(nw2+2*i+1)=d4*(e(li+nw-nx2d+1+i)+e(li+nw-nx2d+2+i))
         elseif (i.eq.nx2d-1) then
          s(nw2+2*i)=d2*e(li+nw)
          s(nw2+2*i+1)=d4*e(li+nw)
         else
          s(nw2+2*i)=d2*e(li+nw-nx2d+1+i)
          s(nw2+2*i+1)=d4*(e(li+nw-nx2d+1+i)+e(li+nw-nx2d+2+i))
         endif
        enddo

!$acc loop vector(32) independent
        do i=1,ny2d-1
          if (mod(i,2).eq.0) then
           km=(int(i/2)-1)*(nx2d-1)+nw
          else
           km=int(i/2)*(nx2d-1)
          endif
           s(nw2+nx2+(i-1)*(nx2-1))=d2*e(li+1+km)
!$acc loop vector(32) independent
         do iw=1,nx2d-2
           s(nw2+nx2+(i-1)*(nx2-1)+iw+(iw-1))=e(li+iw+km)
           s(nw2+nx2+(i-1)*(nx2-1)+iw+(iw-1)+1)=d2*(e(li+iw+km)+
     &        e(li+iw+1+km))
         enddo
           s(nw2+nx2+(i-1)*(nx2-1)+nx2d-1+nx2d-2)=e(li+km+nx2d-1)
           s(nw2+nx2+(i-1)*(nx2-1)+nx2d-1+nx2d-1)=d2*e(li+km+nx2d-1)
        enddo
```

```
!$acc loop vector(32)
        do i=1,ileng2
         xo(i)=xo(i)+s(i)
        enddo
!$acc end kernels
       enddo

        ilev=1
        nx2d=int(nx2d*2)
        ny2d=int(ny2d*2)
        nx2d2=int(nx2d*2)
        ny2d2=int(ny2d*2)
        ileng2=(ny2d2-1)*(nx2d2-1)
        iter=1
        ileng=(ny2d-1)*(nx2d-1)
        tol=1.0d-17
c    Smooth with Gauss-Seidel
        nx1=nx2d-1
        na=(nx2d-1)*(ny2d-1)
        ny2=int(ny2d/2)
        nw=int(ny2d/2)*nx1
        nb=(int(ny2d/2)-1)*nx1
        dx=1.0d0/dfloat(nx2d)
        dy=1.0d0/dfloat(ny2d)
        gg=dx/dy
        c=5.0d0*gg*gg-1.0d0
        d=(1.0d0+gg*gg)*0.5d0
        bp=5.0d0-gg*gg
        a=10.0d0*(1.0d0+(gg*gg))
        li=levk-ileng

!$acc kernels
!$acc loop vector(32) seq
       do i=1,nx1-1
        if (i.eq.1) then
         l(i)=-bp/a
        elseif (i.eq.2) then
         tt=(-bp**2/a)+a
         l(i)=-bp/tt
        else
         tt=a-bp**2/tt
         l(i)=-bp/tt
        endif
       enddo

!$acc loop vector(32) seq
```

```
       do i=1,nx1
        if (i.eq.1) then
         ul(i)=-a
        elseif (i.eq.2) then
         ul(i)=(bp**2/a)-a
         tt=(-bp**2/a)+a
        else
         ul(i)=(bp**2/tt)-a
         tt=a-bp**2/tt
        endif
       enddo

        it=0
!$acc loop vector(32) independent
       do i=1,nx1
        if (i.eq.1) then
         t(i)=c*xo(nw+i)+d*xo(nw+i+1)
        elseif(i.eq.nx1) then
         t(i)=c*xo(nw+i)+d*xo(nw+i-1)
        else
         t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
        endif
       enddo

!$acc loop vector(32) independent
       do k=1,ny2-2
         t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +                d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
        do i=2,nx1-1
         t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +                d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +                +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
        enddo
         t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +                d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
       enddo

         k=ny2*nx1-nx1
         j=(ny2-1)*nx1-nx1
!$acc loop vector(32) independent
       do i=1,nx1
        if (i.eq.1) then
         t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
        elseif (i.eq.nx1) then
         t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
        else
```

```
          t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do i=1,nw
         ii=li+i
         t(i)=r(ii)-t(i)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          e(li+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          e(li+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+(k-1)*
     &                         nx1+i+1))/ul(i)
         enddo
        enddo

!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          t((k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                  d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t((k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                  d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                  +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
         enddo
          t(k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                  d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
        enddo

!$acc loop vector(32)
        do i=1,nb
         t(i)=r(li+nw+i)-t(i)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-1
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
```

```
!$acc loop vector(32) seq
        do i=2,nx1
         s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
        enddo
         e(li+nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
        do i=nx1-1,1,-1
         e(li+nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+nw+(k-1)*
     &                          nx1+i+1))/ul(i)
        enddo
       enddo
!$acc end kernels

c       Multigrid (up)
c       Interopolate Error with semi-coarsening
!$acc kernels
         k=1
         m=1
         nx1=nxl-1
         ny1=ny-1
         li=levk-ileng
!$acc loop vector(32) independent
       do i=1,ny1
         s(i+(i-1)*nx1*2)=e(li+(i-1)*nx1+1)*0.5d0
         s(i+1+(i-1)*nx1*2)=e(li+(i-1)*nx1+1)
!$acc loop vector(32)
        do j=2,nx1
         s((i-1)*nx1*2+2*j-2+i)=0.5d0*(e(li+(i-1)*nx1+1+j-2)+e(li+
     &       (i-1)*nx1+1+j-1))
         s((i-1)*nx1*2+2*j-1+i)=e(li+(i-1)*nx1+1+j-1)
        enddo
         s(nx1*2*i+i)=e(li+(i-1)*nx1+nx1)*0.5d0
       enddo

         k=(2*nxl-1)*(ny-1)
!$acc loop vector(32)
        do i=1,k
         xo(i)=s(i)
        enddo
         levk=levk-ileng
!$acc end kernels

       do ilev=lev-1,1,-1
         nxl=int(nxl*2)
         nxl2=int(nxl*2)
         iter=1
         tol=5.0d-15
```

```
          ileng=(ny-1)*(nxl-1)
          ileng2=(ny-1)*(nxl2-1)
          li=levk-ileng

!$acc kernels
!$acc loop vector(32)
        do i=1,ileng
         xo(i)=xo(i)+e(li+i)
        enddo
!$acc end kernels

c     Smooth with Gauss-Seidel
          nx1=nxl-1
          na=(nxl-1)*(ny-1)
          ny2=int(ny/2)
          nw=int(ny/2)*nx1
          nb=(int(ny/2)-1)*nx1
          dx=1.0d0/dfloat(nxl)
          dy=1.0d0/dfloat(ny)
          gg=dx/dy
          c=5.0d0*gg*gg-1.0d0
          d=(1.0d0+gg*gg)*0.5d0
          bp=5.0d0-gg*gg
          a=10.0d0*(1.0d0+(gg*gg))

!$acc kernels
!$acc loop vector(32) seq
        do i=1,nx1-1
         if (i.eq.1) then
          l(i)=-bp/a
         elseif (i.eq.2) then
          tt=(-bp**2/a)+a
          l(i)=-bp/tt
         else
          tt=a-bp**2/tt
          l(i)=-bp/tt
         endif
        enddo

!$acc loop vector(32) seq
        do i=1,nx1
         if (i.eq.1) then
          ul(i)=-a
         elseif (i.eq.2) then
          ul(i)=(bp**2/a)-a
          tt=(-bp**2/a)+a
         else
```

```
          ul(i)=(bp**2/tt)-a
          tt=a-bp**2/tt
         endif
        enddo

      it=0
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
         elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
         else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +               d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +               d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +               +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
         enddo
          t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +               d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
        enddo

          k=ny2*nx1-nx1
          j=(ny2-1)*nx1-nx1
!$acc loop vector(32) independent
        do i=1,nx1
         if (i.eq.1) then
          t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
         elseif (i.eq.nx1) then
          t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
         else
          t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
         endif
        enddo

!$acc loop vector(32)
        do i=1,nw
         ii=li+i
```

```
          t(i)=r(ii)-t(i)
         enddo

!$acc loop vector(32) independent
         do k=1,ny2
           s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
          do i=2,nx1
           s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
          enddo
           e(li+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
          do i=nx1-1,1,-1
           e(li+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+(k-1)*
     &                       nx1+i+1))/ul(i)
          enddo
         enddo

!$acc loop vector(32) independent
         do k=ny2-1,1,-1
           t((k-1)*nx1+1)=c*(e(li+k*nx1+1)+e(li+(k-1)*nx1+1))+
     +                  d*((e(li+k*nx1+2)+e(li+(k-1)*nx1+2)))
!$acc loop vector(32)
          do i=2,nx1-1
           t((k-1)*nx1+i)=c*(e(li+k*nx1+i)+e(li+(k-1)*nx1+i))+
     +                  d*(e(li+k*nx1+i+1)+e(li+(k-1)*nx1+i+1)
     +                  +e(li+k*nx1+i-1)+e(li+(k-1)*nx1+i-1))
          enddo
           t(k*nx1)=c*(e(li+(k+1)*nx1)+e(li+k*nx1))+
     +                  d*(e(li+(k+1)*nx1-1)+e(li+k*nx1-1))
         enddo

!$acc loop vector(32)
         do i=1,nb
          ii=li+nw+i
          t(i)=r(ii)-t(i)
         enddo

!$acc loop vector(32) independent
         do k=1,ny2-1
           s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
          do i=2,nx1
           s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
          enddo
           e(li+nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
```

```
          do i=nx1-1,1,-1
           e(li+nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*e(li+nw+(k-1)*
     &                          nx1+i+1))/ul(i)
          enddo
         enddo
!$acc end kernels

          iter=it
          tol=res
          levk=levk-ileng

!$acc  kernels
!$acc loop vector(32)
         do i=1,ileng2
          ii=levk-ileng2+i
          xo(i)=e(ii)
         enddo
!$acc end  kernels

c     Interpolate Error with semi-coarsening
          k=1
          m=1
          nx1=nxl-1
          ny1=ny-1
!$acc  kernels
!$acc loop vector(32) independent
         do i=1,ny1
           s(i+(i-1)*nx1*2)=e(li+((i-1)*nx1+1))*0.5d0
           s(i+(i-1)*nx1*2+1)=e(li+((i-1)*nx1+1))
!$acc loop vector(32)
          do j=2,nx1
           s((i-1)*nx1*2+2*j-2+i)=0.5d0*(e(li+(i-1)*nx1+1+j-2)+e(li+
     &        (i-1)*nx1+1+j-1))
           s((i-1)*nx1*2+2*j-1+i)=e(li+(i-1)*nx1+1+j-1)
          enddo
           s(nx1*2*i+i)=e(li+(i-1)*nx1+nx1)*0.5d0
         enddo

         if (ilev.gt.1) then
!$acc loop vector(32)
         do i=1,ileng2
          xo(i)=xo(i)+s(i)
         enddo
         endif
!$acc end kernels
       enddo
211     continue
```

```
!$acc kernels
!$acc loop vector(32)
        do i=1,n
         xo(i)=xn(i)
        enddo

!$acc loop vector(32)
        do i=1,n
         xo(i)=xo(i)+s(i)
        enddo
!$acc end kernels

         iter=1
         tol=5.0d-15
c     Smooth with Gauss-Seidel
         nx1=nx-1
         ny2=int(ny/2)
         nw=int(ny/2)*nx1
         nb=(int(ny/2)-1)*nx1
         c=5.0d0*g*g-1.0d0
         d=(1.0d0+g*g)*0.5d0
         bp=5.0d0-g*g
         a=10.0d0*(1.0d0+(g*g))

!$acc kernels
         tt=(-bp**2/a)+a
!$acc loop vector(32) seq
        do i=1,nx1-1
         if (i.eq.1) then
          l(i)=-bp/a
         elseif (i.eq.2) then
          tt=(-bp**2/a)+a
          l(i)=-bp/tt
         else
          tt=a-bp*bp/tt
          l(i)=-bp/tt
         endif
        enddo

         tt=(-bp*bp/a)+a
!$acc loop vector(32) seq
        do i=1,nx1
         if (i.eq.1) then
          ul(i)=-a
         elseif (i.eq.2) then
          ul(i)=(bp*bp/a)-a
```

```
          tt=(-bp**2/a)+a
         else
          ul(i)=(bp*bp/tt)-a
          tt=a-bp*bp/tt
         endif
        enddo

         it=0
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i+1)
         elseif (i.eq.nx1) then
          t(i)=c*xo(nw+i)+d*xo(nw+i-1)
         else
          t(i)=c*xo(nw+i)+d*(xo(nw+i+1)+xo(nw+i-1))
         endif
        enddo

!$acc loop vector(32) independent
        do k=1,ny2-2
          t(k*nx1+1)=c*(xo(nw+k*nx1+1)+xo(nw+(k-1)*nx1+1))+
     +                d*((xo(nw+k*nx1+2)+xo(nw+(k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t(k*nx1+i)=c*(xo(nw+k*nx1+i)+xo(nw+(k-1)*nx1+i))+
     +                d*(xo(nw+k*nx1+i+1)+xo(nw+(k-1)*nx1+i+1)
     +                +xo(nw+k*nx1+i-1)+xo(nw+(k-1)*nx1+i-1))
         enddo
          t((k+1)*nx1)=c*(xo(nw+(k+1)*nx1)+xo(nw+k*nx1))+
     +                d*(xo(nw+(k+1)*nx1-1)+xo(nw+k*nx1-1))
        enddo

         k=ny2*nx1-nx1
         j=(ny2-1)*nx1-nx1
!$acc loop vector(32)
        do i=1,nx1
         if (i.eq.1) then
          t(k+i)=c*xo(nw+j+i)+d*xo(nw+j+i+1)
         elseif (i.eq.nx1) then
          t(ny2*i)=c*xo(nw+(ny2-1)*i)+d*xo(nw+(ny2-1)*i-1)
         else
          t(k+i)=c*xo(nw+j+i)+d*(xo(nw+j+i+1)+xo(nw+j+i-1))
         endif
        enddo
!$acc end kernels
```

```
!$acc kernels
!$acc loop vector(32)
        do i=1,nw
         t(i)=b(i)-t(i)
        enddo

!$acc loop vector(32) independent
        do k=1,ny2
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
         do i=2,nx1
          s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
         enddo
          xn(k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
         do i=nx1-1,1,-1
          xn((k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*xn((k-1)*nx1+i+1))/ul(i)
         enddo
        enddo
!$acc end kernels

!$acc kernels
!$acc loop vector(32) independent
        do k=ny2-1,1,-1
          t((k-1)*nx1+1)=c*(xn(k*nx1+1)+xn((k-1)*nx1+1))+
     +                   d*((xn(k*nx1+2)+xn((k-1)*nx1+2)))
!$acc loop vector(32)
         do i=2,nx1-1
          t((k-1)*nx1+i)=c*(xn(k*nx1+i)+xn((k-1)*nx1+i))+
     +                   d*(xn(k*nx1+i+1)+xn((k-1)*nx1+i+1)
     +                   +xn(k*nx1+i-1)+xn((k-1)*nx1+i-1))
         enddo
          t(k*nx1)=c*(xn((k+1)*nx1)+xn(k*nx1))+
     +                   d*(xn((k+1)*nx1-1)+xn(k*nx1-1))
        enddo

!$acc loop vector(32)
        do i=1,nb
         t(i)=b(nw+i)-t(i)
        enddo
!$acc end kernels

!$acc kernels
!$acc loop vector(32) independent
        do k=1,ny2-1
          s((k-1)*nx1+1)=t((k-1)*nx1+1)
!$acc loop vector(32) seq
```

```
          do i=2,nx1
           s((k-1)*nx1+i)=t((k-1)*nx1+i)-l(i-1)*s((k-1)*nx1+i-1)
          enddo
           xn(nw+k*nx1)=s(k*nx1)/ul(nx1)
!$acc loop vector(32) seq
          do i=nx1-1,1,-1
           xn(nw+(k-1)*nx1+i)=(s((k-1)*nx1+i)-bp*xn(nw+(k-1)*
     &                        nx1+i+1))/ul(i)
          enddo
         enddo
!$acc end kernels

!$acc  kernels
!$acc loop vector independent
        do i=1,nx1
         if (i.eq.1) then
          r(i)=c*xn(nw+i)+d*xn(nw+i+1)
         elseif (i.eq.nx1) then
          r(i)=c*xn(nw+i)+d*xn(nw+i-1)
         else
          k=nw+i
          r(i)=c*xn(k)+d*xn(k+1)+d*xn(k-1)
         endif
        enddo
!$acc  end kernels

!$acc  kernels
!$acc loop vector independent
        do k=1,ny2-2
          r(k*nx1+1)=c*(xn(nw+k*nx1+1)+xn(nw+(k-1)*nx1+1))+
     +                d*((xn(nw+k*nx1+2)+xn(nw+(k-1)*nx1+2)))
!$acc loop vector independent
          do i=2,nx1-1
           r(k*nx1+i)=c*(xn(nw+k*nx1+i)+xn(nw+(k-1)*nx1+i))+
     +                d*(xn(nw+k*nx1+i+1)+xn(nw+(k-1)*nx1+i+1)
     +                +xn(nw+k*nx1+i-1)+xn(nw+(k-1)*nx1+i-1))
          enddo
           r((k+1)*nx1)=c*(xn(nw+(k+1)*nx1)+xn(nw+k*nx1))+
     +                d*(xn(nw+(k+1)*nx1-1)+xn(nw+k*nx1-1))
        enddo
!$acc  end kernels

!$acc  kernels
          k=nw-nx1
          k1=k-nx1
!$acc loop vector
        do i=1,nx1
```

```
          if (i.eq.1) then
           r(k+i)=c*xn(nw+k1+i)+d*xn(nw+k1+i+1)
          elseif (i.eq.nx1) then
           r(ny2*i)=c*xn(nw+(ny2-1)*i)+d*xn(nw+(ny2-1)*i-1)
          else
           r(k+i)=c*xn(nw+k1+i)+d*(xn(nw+k1+i+1)+xn(nw+k1+i-1))
          endif
         enddo
!$acc  end kernels

!$acc  kernels
!$acc loop vector independent
        do k=ny2-1,1,-1
          r(nw+(k-1)*nx1+1)=c*(xn(k*nx1+1)+xn((k-1)*nx1+1))+
     +                 d*((xn(k*nx1+2)+xn((k-1)*nx1+2)))
!$acc loop vector independent
          do i=2,nx1-1
           r(nw+(k-1)*nx1+i)=c*(xn(k*nx1+i)+xn((k-1)*nx1+i))+
     +                 d*(xn(k*nx1+i+1)+xn((k-1)*nx1+i+1)
     +                 +xn(k*nx1+i-1)+xn((k-1)*nx1+i-1))
          enddo
           r(nw+k*nx1)=c*(xn((k+1)*nx1)+xn(k*nx1))+
     +                 d*(xn((k+1)*nx1-1)+xn(k*nx1-1))
        enddo
!$acc  end kernels

!$acc  kernels
!$acc loop vector independent
        do k=1,ny2
          r((k-1)*nx1+1)=r((k-1)*nx1+1)+bp*xn((k-1)*nx1+2)-a*xn((k-1)*nx1+1)
!$acc loop vector(32)
          do i=2,nx1-1
           r((k-1)*nx1+i)=r((k-1)*nx1+i)+bp*(xn((k-1)*nx1+i-1)+
     +                xn((k-1)*nx1+i+1))-a*xn((k-1)*nx1+i)
          enddo
           r(k*nx1)=r(k*nx1)+bp*xn(k*nx1-1)-a*xn(k*nx1)
        enddo

!$acc loop vector independent
        do k=1,ny2-1
          r(nw+(k-1)*nx1+1)=r(nw+(k-1)*nx1+1)+bp*xn(nw+(k-1)*nx1+2)
     +                    -a*xn(nw+(k-1)*nx1+1)
!$acc loop vector
          do i=2,nx1-1
           r(nw+(k-1)*nx1+i)=r(nw+(k-1)*nx1+i)+bp*(xn(nw+(k-1)*nx1+i-1)+
     +                xn(nw+(k-1)*nx1+i+1))-a*xn(nw+(k-1)*nx1+i)
          enddo
```

```
          r(nw+k*nx1)=r(nw+k*nx1)+bp*xn(nw+k*nx1-1)-a*xn(nw+k*nx1)
         enddo
!$acc  end kernels

!$acc  kernels
!$acc loop vector
         do i=1,nw+nb
          r(i)=b(i)-r(i)
         enddo

!$acc loop vector
         do i=1,n
          xo(i)=xn(i)
         enddo
!$acc end kernels

!$acc kernels
          rnorm=0.0d0
!$acc loop reduction(+:rnorm)
         do i=1,n
          rnorm=rnorm+r(i)*r(i)
         enddo
!$acc end kernels

         rnorm=dsqrt(rnorm)
c        print*,'---- Vcycle No',st,'  ------',rnorm
         enddo
!$acc end data
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       END  GPU Region
C
        tm=omp_get_wtime()-tm0
        print*,'Time for  V-cycles =',tm,'  secs'
        print*,'Number of V-cycles =',st

c       End V-cycle
888     open(1,file='du')
        call makeA(a0,a0f,a1,nx,ny,ipvt)
        call matA(xn,xo,nx,ny,a0,a1,s)
        call daxpy(n,-1.0d0,b,1,xo,1)
        dm=0.0d0

       do i=1,n
        dm=dmax1(xo(i),dm)
       enddo
```

```
      print*,'||b-A*xe||oo=',dm
      call dscal(n,-1.0d0,xo,1)
      print*,'||b-A*xe||2=',dnrm2(n,xo,1)
      call zebrainv(nx,ny,xn,xo)
      k=1

     do j=1,ny-1
       do i=1,nx-1
        xn(k)=u(x(i),y(j))
        k=k+1
       enddo
      enddo

111   format(f12.6)
      open(2,file='x')
      open(3,file='y')

      do i=1,ny-1
       write(3,111) y(i)
      enddo
      close(3)

      do i=1,nx-1
       write(2,111) x(i)
      enddo
      close(2)

      dm=0.0d0
     do i=1,n
      xn(i)=dabs(xn(i)-xo(i))
      dm=dmax1(xn(i),dm)
      write(1,111) xn(i)
     enddo
     print*,'||x-xe||2=',dnrm2(n,xn,1)
     print*,'||x-xe||oo=',dm
     close(1)
     stop
     end
```

## Α΄.2 Κώδικες υποπρογραμμάτων σε Fortran

```
C  This subroutine creates matrix factorizations of basic matrices A0, A1
      subroutine makeA(a0,a0f,a1,nx,ny,ipvt)
      implicit real*8 (a-h,o-z)
      real*8 a0(2,*),a1(2,*),g,a,c,bp,d,dx,dy,
     +       a0f(4,*)
      integer ipvt(*)

      dx=1.0d0/dfloat(nx)
      dy=1.0d0/dfloat(ny)
      g=dx/dy
      a=10.0d0*(1.0d0+(g*g))
      c=5.0d0*g*g-1.0d0
      bp=5.0d0-g*g
      d=(1.0d0+g*g)*0.5d0
      do i=1,nx-1
       a0(2,i)=-a
       a0f(3,i)=-a
       a1(2,i)=c
      enddo
      do i=2,nx-1
       a0(1,i)=bp
       a1(1,i)=d
       a0f(2,i)=bp
       a0f(4,i-1)=bp
      enddo
      call dgbtrf(nx-1,nx-1,1,1,a0f,4,ipvt,info)
      if (info.ne.0) then
       print*,' Ao is not invertable',info
       stop
      endif
      return
      end subroutine

C  This subroutine performs the zebra coloring scheme
      subroutine zebra(nx,ny,a,b)
      implicit real*8 (a-h,o-z)
      real*8 a(*),b(*)
      nx1=nx-1
      ny1=ny-1
      k=0
      do i=0,ny1-1,2
       call dcopy(nx1,a(i*nx1+1),1,b(k*nx1+1),1)
       k=k+1
      enddo
      do i=1,ny1-2,2
```

```
       call dcopy(nx,a(i*nx1+1),1,b(k*nx1+1),1)
       k=k+1
      enddo
      return
      end

C  This subroutine performs the inverse zebra coloring scheme
      subroutine zebrainv(nx,ny,a,b)
      implicit real*8 (a-h,o-z)
      real*8 a(*),b(*)

      nx1=nx-1
      ny1=ny-1
      k=0
      do i=0,ny-2,2
       call dcopy(nx1,a(k*nx1+1),1,b(i*nx1+1),1)
       k=k+1
      enddo
      do i=1,ny-3,2
       call dcopy(nx1,a(k*nx1+1),1,b(i*nx1+1),1)
       k=k+1
      enddo
      return
      end

C  This subroutine performs the matrix vector multiplication x=Hb t
      subroutine matblack1(nx,ny,a1,x,t)
      implicit real*8 (a-h,o-z)
      real*8 a1(2,*),x(*),t(*)
      n=int(ny/2)
      nx1=nx-1
      call dsbmv('U',nx1,1,1.0d0,a1,2,t,1,0.0d0,x,1)
      do k=1,n-2
         call daxpy(nx1,1.0d0,t(k*nx1+1),1,t((k-1)*nx1+1),1)
         call dsbmv('U',nx1,1,1.0d0,a1,2,t((k-1)*nx1+1),1,0.0d0,
     +              x(k*nx1+1),1)
       enddo
         call dsbmv('U',nx1,1,1.0d0,a1,2,t((n-2)*nx1+1),1,0.0d0,
     +              x((n-1)*nx1+1),1)
       return
       end

C  This subroutine performs the matrix vector multiplication x=Hw t
      subroutine matwhite1(nx,ny,a1,x,t)
      implicit real*8 (a-h,o-z)
      real*8 a1(2,*),x(*),t(*)
      n=int(ny/2)
```

```
      nx1=nx-1
      do k=n-1,1,-1
         call daxpy(nx1,1.0d0,t((k-1)*nx1+1),1,t(k*nx1+1),1)
         call dsbmv('U',nx1,1,1.0d0,a1,2,t(k*nx1+1),1,0.0d0,
     +              x((k-1)*nx1+1),1)
       enddo
       return
       end

C  This subroutine performs the block diagonal solution of Dw x=x
      subroutine solvewhite1(nx,ny,a0f,ipvt,x)
      implicit real*8 (a-h,o-z)
      real*8 a0f(4,*),x(*)
      integer ipvt(*)
      n=int(ny/2)
      nx1=nx-1

      do k=1,n
         call dgbtrs('N',nx1,1,1,1,a0f,4,ipvt,
     +               x((k-1)*nx1+1),nx1,info)
       enddo
       return
       end

C  This subroutine performs the block diagonal solution of Db x=x
      subroutine solveblack1(nx,ny,a0f,ipvt,x)
      implicit real*8 (a-h,o-z)
      real*8 a0f(4,*),x(*)
      integer ipvt(*)
      n=int(ny/2)
      nx1=nx-1
      do k=1,n-1
         call dgbtrs('N',nx1,1,1,1,a0f,4,ipvt,
     +               x((k-1)*nx1+1),nx1,info)
       enddo
       return
       end

C  This subroutine performs the matrix vector multiplication x=Dw t+x
      subroutine matdw1(nx,ny,a0,x,t)
      implicit real*8 (a-h,o-z)
      real*8 a0(2,*),x(*),t(*)
      n=int(ny/2)
      nx1=nx-1
       do k=1,n
      call dsbmv('U',nx1,1,1.0d0,a0,2,t((k-1)*nx1+1),1,1.0d0,
     +           x((k-1)*nx1+1),1)
```

```
      enddo
      return
      end

C  This subroutine performs the matrix vector multiplication x=Db t
      subroutine matdb1(nx,ny,a0,x,t)
      implicit real*8 (a-h,o-z)
      real*8 a0(2,*),x(*),t(*)
      n=int(ny/2)
      nx1=nx-1
      do k=1,n-1
      call dsbmv('U',nx1,1,1.0d0,a0,2,t((k-1)*nx1+1),1,1.0d0,
     +           x((k-1)*nx1+1),1)
      enddo
      return
      end

C  This subroutine performs the matrix vector multiplication x=A t
      subroutine matA(t,x,nx,ny,a0,a1,s)
      implicit real*8 (a-h,o-z)
      real*8 a0(2,*),x(*),t(*),a1(2,*),s(*)
      nx1=nx-1
      nw=int(ny/2)*nx1
      nb=(int(ny/2)-1)*nx1
      call dcopy(nb,t(nw+1),1,s,1)
      call matblack1(nx,ny,a1,x,s)
      call dcopy(nw,t,1,s,1)
      call matwhite1(nx,ny,a1,x(nw+1),s)
      call matdw1(nx,ny,a0,x,t)
      call matdb1(nx,ny,a0,x(nw+1),t(nw+1))
      return
      end

C  This subroutine creates the right hand side vector of the linear system
      subroutine makeb(nx,ny,b,x,y)
      implicit real*8 (a-h,o-z)
      real*8 b(*),x(0:nx),y(0:ny),dx,dy,g,a,c,bp,d,u,f
      external u,f

      dx=1.0d0/dfloat(nx)
      dy=1.0d0/dfloat(ny)
      g=dx/dy
      c=5.0d0*g*g-1.0d0
      bp=5.0d0-g*g
      d=(1.0d0+g*g)*0.5d0

      x(0)=0.0d0
```

```
      do i=1,nx
       x(i)=x(i-1)+dx
      enddo
      y(0)=0.0d0
      do j=1,ny
       y(j)=y(j-1)+dy
      enddo
      k=1
      do j=1,ny-1
       do i=1,nx-1
        b(k)=(dx*dx)*0.5d0*(8.0d0*f(x(i),y(j))+f(x(i),y(j+1))+f(x(i),
     +        y(j-1))+f(x(i+1),y(j))+f(x(i-1),y(j)))
        k=k+1
       enddo
      enddo

       b(1)=b(1)-d*u(x(2),y(0))-c*u(x(1),y(0))-d*u(x(0),y(0))-
     +      bp*u(x(0),y(1))-d*u(x(0),y(2))
      do i=2,nx-2
       b(i)=b(i)-c*u(x(i),y(0))-d*u(x(i-1),y(0))-d*u(x(i+1),y(0))
      enddo
       b(nx-1)=b(nx-1)-d*u(x(nx-2),y(0))-c*u(x(nx-1),y(0))-d*u(x(nx),
     +         y(0))-bp*u(x(nx),y(1))-d*u(x(nx),y(2))

      do k=1,ny-3
       b(k*(nx-1)+1)=b(k*(nx-1)+1)-d*u(x(0),y(k))-bp*u(x(0),y(k+1))-
     +               d*u(x(0),y(k+2))
       b((k+1)*(nx-1))=b((k+1)*(nx-1))-d*u(x(nx),y(k))-bp*u(x(nx),
     +                 y(k+1))-d*u(x(nx),y(k+2))
      enddo

       b(((ny-2)*(nx-1))+1)=b(((ny-2)*(nx-1))+1)-d*u(x(0),y(ny-2))-
     +           bp*u(x(0),y(ny-1))-d*u(x(0),y(ny))-c*u(x(1),y(ny))-
     +           d*u(x(2),y(ny))
      do i=2,nx-2
       b(((ny-2)*(nx-1))+i)=b(((ny-2)*(nx-1))+i)-d*u(x(i-1),y(ny))-
     +                      c*u(x(i),y(ny))-d*u(x(i+1),y(ny))
      enddo
       b((ny-1)*(nx-1))=b((ny-1)*(nx-1))-d*u(x(nx-2),y(ny))-
     +                  c*u(x(nx-1),y(ny))-d*u(x(nx),y(ny))-
     +                  bp*u(x(nx),y(ny-1))-d*u(x(nx),y(ny-2))
      return
      end subroutine

C  This function is the right hand discretized function of the PDE
       real*8 function f(x,y)
    implicit real*8 (a-h,o-z)
```

```
            e1=dexp(-100.0d0*(y-0.1d0)*(y-0.1d0))
            e2=dexp(-100.0d0*(x-0.1d0)*(x-0.1d0))
             f=(-4000.0d0*e2*(x*x-x)*e1*(y*y-y)+10.0d0
     +         *(-200.0d0*x+20.0d0)*(-200.0d0*x+20.0d0)*e2*
     +         (x*x-x)*e1*(y*y-y)+20.0d0*(-200.0d0*x+20.0d0)*e2*
     +         (2.0d0*x-1.0d0)*e1*(y*y-y)+20.0d0*e2*e1*(y*y-y)+
     +         10.0d0*e2*(x*x-x)*(-200.0d0*y+20.0d0)*(-200.0d0*y+
     +         20.0d0)*e1*(y*y-y)+20.0d0*e2*(x*x-x)*(-200.0d0*y+
     +         20.0d0)*e1*(2.0d0*y-1.0d0)+20.0d0*e2*(x*x-x)*e1)
        return
        end

C  This function is the  discretized actual solution function of the PDE
      real*8 function u(x,y)
      implicit real*8 (a-h,o-z)
      a=0.1d0
      b=0.1d0
      u=10.0d0*dexp(-100.0d0*((x-a)**2+(y-b)**2))*(x**2-x)*(y**2-y)
      return
      end
```

## Α΄.3 Αρχείο παραμέτρων μεταγλώττισης

```
# This is the code for the Makefile

FORTRAN  =  pgf90
OPTS     =
LOADER   =  pgf90
LOADOPTS =  -mp -O4 -acc -Minfo=accel -Mcuda=cc2x,6.0 -mcmodel=medium
            -Mlarge_arrays -o nik

FILES  =   main.o makeA.o iparmq.o ieeeck.o ilaenv.o dlaswp.o dgemv.o
           dtbsv.o makeb.o lsame.o dgemm.o dtrsm.o dger.o dswap.o idamax.o
           dgbtf2.o dgbtrf.o dgbtrs.o xerbla.o dsbmv.o zebra.o u.o fb.o
           matblack.o dscal.o daxpy.o dnrm2.o dcopy.o

OBJS   =

nik : Makefile $(FILES)
        $(LOADER) $(LOADOPTS) $(FILES) $(OBJS) $(LIBRARY)
.f.o:
        pgf90  -mp -acc -O4 -Minfo=accel -Mcuda=cc2x,6.0 -mcmodel=medium
               -Mlarge_arrays -c $*.f
```

# Βιβλιογραφία


[1] M. Adams, M. Brezina, J. Hu and R. Tuminaro "Parallel Multigrid smoothing: Polynomial versus Gauss-Seidel", *J. Comp. Physics* 188, pp. 593-610, 2003.

[2] H. Anzt, S. Tomov, M. Gates, J. Dongarra and V. Heuveline "Block-asynchronous Multigrid Smoothers for GPU-accelerated systems", University of Tennessee Computer Science Technical Report UT-CS-11-690 , 2011.

[3] J. H. Bramble, "Multigrid Methods", vol. 294, *Pitman Research Notes in Mathematical Sciences*, Longman Scientific and Technical, Essex, England, 1993.

[4] A. Brandt, "Multi-level adaptive solutions to boundary value problems", *Mathematics of Computation*, 31 (1977),pp.333-390.

[5] A. Brandt, "A guide to multigrid development", *in Multigrid Methods*, W.Hackbusch and U. Trottenberg, eds.,Springer Verlag, Berlin, 1982,pp. 220-312.

[6] A. Brandt, S.F.McCormilk, and J.Ruge,"Algebraic multigrid (AMG) for sparse matrix equations", *in Sparsity and Its Applications*, D.J.Evans, ed.,Cambridge University Press,Cambridge, 1984, pp. 257-284.

[7] W. L. Briggs, V. E. Henson, and S. F. McCormilk, "A Multigrid Tutorial", SIAM, Philadelphia, 2000 . Second edition.

[8] N.E. Charalampaki and E.N. Mathioudakis, "CPU-GPU computations for MultiGrid techniques coupled with Fourth-Order Compact Discretizations for I-

sotropic and Anisotropic Poisson problems", *Procs of NumAn2014 Int. Conf*, http://lib.amcl.tuc.gr/handle/triton/70, 2014.

[9] C. C. Christara and B. Smith, "Multigrid and Multilevel Methods for Quadratic Spline Collocation", *BIT*, vol. 37 (4), pp.781-803, 1997.

[10] L. Collatz, "The numerical Treatment of Differential equations", *Springer Verlag*, 1960

[11] S. Cook, "CUDA Programming", *Morgan Kaufmann*, 2013.

[12] C. Feng, S. Shu, J. Xu and C.-S. Zhang, "Numerical Study of Geometric Multigrid Methods on CPU-GPU Heterogeneous Computers", *Adv. Appl. Math. Mech.*, 6(1), pp 1-23, 2014

[13] M. Gupta, J. Kouatchou and J. Zhang, "Comparison of Second- and Fourth-Order Discretizations for Multigrid Poisson Solvers", *JCP*, 132, pp 226-232, 1997

[14] W. Hackbusch and U. Trottenberg, "Multigrid Methods", vol. 960, *Lecture Notes in Mathematics*, Spinger-Verlag, Berlin, 1982.

[15] W. Hackbusch, "Multi-Grid Methods and Applications",Vol. 4, Springer Series *in Computational Mathematics*, Springer-Verlag, Berlin,1985.

[16] L. A. Hageman and D. M. Young, " Applied Iterative Methods", *Academic Press*, New York, 1981.

[17] *icl.cs.utk.edu/magma/*.

[18] D. Kirk and W. Hwu, "Programming Massivelly Parallel Processors", Morgan Kaufmann, 2010.

[19] V. Mandikas, E. N. Mathioudakis and N. Kampanis, "MultiGrid techniques and high-order cell-centered compact scheme discretizations for elliptic PDEs", *submitted* , 2013.

[20] 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.

[21] 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.

[22] 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.

[23] S.F McCormick, ed., "Multigrid Methods", SIAM Philadelphia, 1987.

[24] Y. Saad, "Iterative Methods for sparse linear systems", SIAM, 2003.

[25] J. Sanders and E. Kandrot "CUDA by example: An Itroduction to General-Purpose GPU Programming", Addison-Wesley, 2011.

[26] L. H. Thomas, "Elliptic problems in linear difference equations over a network", *Watson Sci. Comput. Lab. Rept.*, Columbia University, NY, 1949

[27] U. Trottenberg, C. Oosterlee, and A. Schüller, "Multigrid", Academic Press, New-York, 2001.

[28] D.M. Young, "Iterative Solution of Large Linear Systems", *Academic Press*, New York, 1981

[29] R. Varga, "Matrix Iterative Analysis", *Springer Verlag*, 2000

[30] N. Vilanakis and E. Mathioudakis, "Parallel iterative solution of the Hermite Collocation equations on GPUs II", *Journal of Physics: Conference Series*, 490(1), 2014

[31] P. Wesseling, "An Itroduction to Multigrid Methods", J. Wiley and Sons, Chichester, U.K., 1992.

[32] *www.netlib.org/blas*.

[33] *www.netlib.org/lapack*.

[34] *www.nvidia.com/cuda*.

[35] *www.openacc-standard.org*.

[36] *www.openmp.org*.

[37] *www.open-mpi.org*.

[38] *www.pgroup.com*.

[39] *www.top500.org*.

[40] J. Zhang, "Multigrid method and fourth-order compact scheme for 2D Poisson equation with unequal mesh-size discretization", *JCP*, 179, pp 170-179, 2002