ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΓΕΝΙΚΟ ΤΜΗΜΑ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΦΑΡΜΟΣΜΕΝΕΣ ΕΠΙΣΤΗΜΕΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ

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

ΕΠΙΣΤΗΜΟΝΙΚΟΙ ΥΠΟΛΟΓΙΣΜΟΙ ΣΕ ΕΠΕΞΕΡΓΑΣΤΕΣ ΓΡΑΦΙΚΩΝ ΥΠΟΣΥΣΤΗΜΑΤΩΝ ΠΟΛΛΑΠΛΩΝ ΠΥΡΗΝΩΝ ΓΙΑ ΤΗΝ ΑΡΙΘΜΗΤΙΚΗ ΜΕΘΟΔΟ ΠΕΠΕΡΑΣΜΕΝΩΝ ΣΤΟΙΧΕΙΩΝ HERMITE COLLOCATION

ΙΩΑΝΝΗΣ ΑΘΑΝΑΣΑΚΗΣ

Επιβλέπων : Επίκ. Καθηγητής **Εμμανουήλ Μαθιουδάκης**

ΧΑΝΙΑ , 2012

Η διατριβή αυτή εξετάστηκε επιτυχώς από τη παρακάτω Τριμελή Επιτροπή

- Επίκ. Καθηγητή Εμμανουήλ Μαθιουδάκη
- Καθηγήτρια Έλενα Παπαδοπούλου
- Καθηγητή Ιωάννη Σαριδάκη

η οποία ορίστηκε κατά τη 28^{η}/19-7-2012 συνεδρίαση της Γενικής Συνέλευσης Ειδικής Σύνθεσης του Γενικού Τμήματος του Πολυτεχνείου Κρήτης.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους εκείνους που μου παρείχαν την υποστήριξη για την εκπόνηση αυτής της διατριβής. Πρώτα απ' όλους ευχαριστώ τον επιβλέποντα των μεταπτυχιακών σπουδών μου Επίκουρο Καθηγητή Εμμανουήλ Μαθιουδάκη για την επιστημονική καθοδήγηση και την ηθική υποστήριξη που μου παρείχε τα δύο τελευταία χρόνια.

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

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

Τέλος ευχαριστώ τη συμφοιτήτρια μου Μαρία Γαϊτάνη και την οικογένεια μου που με την δική τους υποστήριξη κατάφερα να ολοκληρώσω τις σπουδές μου.

Περίληψη

Η εξέλιξη της τεχνολογίας των υπερυπολογιστικών στυστημάτων έχει επιτρέψει την επικουρική χρήση των επεξεργαστών διαφόρων υποσυστημάτων τους για την ενίσχυση των υπολογιστικών δυνατοτήτων τους. Για παράδειγμα είναι εφικτή η διεξαγωγή μέρους των επιστημονικών υπολογισμών απο τους επεξεργαστές των γραφικών υποσυστημάτων. Αυτό έχει ωθήσει την παραγωγή των γραφικών επεξεργαστών στη κατηγορία των παράλληλων αρχιτεκτονικών, με αποτέλεσμα να υπάρχουν σήμερα γραφικά υποσυστήματα τα οποία διαθέτουν επεξεργαστές με χιλιάδες υπολογιστικούς πυρήνες και σημαντικά μεγέθη τοπικής μνήμης. Στη διατριβή αυτή παρουσιάζεται η εφαρμογή της Τεχνική Πολυπλέγματος για την επαναληπτική επίλυση αραιών γραμμικών συστημάτων που προκύπτουν από την αριθμητική μέθοδο επίλυσης Προβλημάτων Συνοριακών Τιμών (ΠΣΤ) με χρήση της μεθόδου πεπερασμένων στοιχείων Hermite Collocation σε παραλληλόγραμμα χωρία. Η υλοποίηση πραγματοποιείται σε υπολογιστικά περιβάλλοντα τα οποία διαθέτουν πολυεπεξεργαστικά γραφικά υποσυστήματα και γίνεται μελέτη της συμπεριφοράς απόδοσης των κλασσικών αλγορίθμων της μεθόδου σε αυτού του είδους υπολογιστικές αρχιτεκτονικές.

Η εργασία αυτή είναι δομημένη σε πέντε κεφάλαια: Στο πρώτο κεφάλαιο παρουσιάζονται τεχνικές αποθήκευσης αραιών πινάκων καθώς και αλγόριθμοι για τις αντίστοιχες βασικές πράξεις Γραμμικής Άλγεβρας. Στη συνέχεια παρουσιάζεται το τμήμα της βιβλιοθήκης υποπρογραμμάτων Γραμμικής Άλγεβρας SPARSKIT καθώς και παρόμοιου τύπου υποπρογράμματα που κατασκευάστηκαν για την ολοκλήρωση της εφαρμογής. Στο δεύτερο κεφάλαιο παρουσιάζονται οι βασικές αρχές της Τεχνικής Πολυπλέγματος. Στο τρίτο

κεφάλαιο εφαρμόζεται η αριθμητική μέθοδος επίλυσης ΠΣΤ Collocation με χρήση των πολυωνύμων βάσης Hermite. Κατασκευάζεται το γραμμικό σύστημα που προκύπτει από την εφαρμογή της μεθόδου για την περίπτωση γραμμικού τελεστή δεύτερης τάξης. Στο τέταρτο κεφάλαιο γίνεται αναφορά στη φιλοσοφία του παράλληλου προγραμματισμού για την ανάπτυξη εφαρμογών των οποίων οι επιστημονικοί υπολογισμοί διεξάγονται επικουρικά και σε επεξεργαστές των γραφικών υποσυστημάτων. Παρουσιάζονται οι διαθέσιμες γλώσσες προγραμματισμού για παράλληλη επεξεργασία σε κάρτες γραφικών και οι βιβλιοθήκες υποπρογραμμάτων πράξεων Γραμμικής Άλγεβρας. Στο τέλος του κεφαλαίου παρουσιάζονται τα αποτελέσματα απο τη μελέτη της συμπεριφοράς απόδοσης της υλοποίησης ενός προβλήματος δοκιμής σε ένα γραφικό υποσύστημα. Το τελευταίο κεφάλαιο εμφανίζει τις μετρήσεις απόδοσης του αλγορίθμου της αριθμητικής επίλυσης προβλημάτων συνοριακών τιμών με τη μέθοδο πεπερασμένων στοιχείων Hermite Collocation, στην οποία έχει ενσωματωθεί η Τεχνική Πολυπλέγματος. Τα πειραματικά αποτελέσματα καθώς και τα συμπεράσματα απο την υλοποίηση του αλγορίθμου με τη χρήση γραφικών υποσυστημάτων συμπληρώνουν αυτή την ενότητα. Τέλος, οι κώδικες των εφαρμογών που αναπτύχθηκαν σε αυτή τη διατριβή με τη χρήση των γλωσσών προγραμματισμού Fortran και CUDA παρατίθενται στα δύο παραρτήματα.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Τεχνικές αποθήκευσης αραιών πινάκων και βασικές πράξεις Γραμμικής Άλγεβρας

1.1 Δομές αποθήκευσης

1.1.1 Εισαγωγή

Κατά την επίλυση μιας ΜΔΕ με χρήση μιας αριθμητική μεθόδου διακριτοποίησης του χωρίου που ορίζεται ένα ΠΣΤ παράγεται ένα γραμμικό σύστημα της μορφής $Ax = b$, όπου ο πίνακας των συντελεστών των αγνώστων Α είναι αραιός. Γενικά ένας πίνακας θεωρείται αραιός εάν ένα πολύ μεγάλο ποσοστό των στοιχείων του είναι μηδενικά. Σε αυτές τις περιπτώσεις μπορούμε να εκμεταλευτούμε την δομή του πίνακα ώστε να μην αποθηκευτούν τα μηδενικά στοιχεία του κι έτσι να αποφύγουμε χώρο τις άσκοπες πράξεις μεταξύ τους αλλά και να επιτευχθεί δραματική μείωση του χώρου αποθηκεύοντας μόνο τα μη μηδενικά στοιχεία του πίνακα.

Το όφελος με την επιλογή μιας αραιής μορφής αποθήκευσης , είναι φυσικά η βελτιστο-

ποίηση μνήμης και διεξαγωγής των πράξεων γραμμικής άλγεβρας. Στο κεφάλαιο αυτό παρουσιάζονται διάφορες τεχνικές αραιής αποθήκευσης (π.χ. COO,CSR,CSC,...) αλλά και διάφοροι αλγόριθμοι για αραιούς πίνακες οι οποίοι σήμερα χρησιμοποιούνται για τη πραγματοποίηση επιστημονικών υπολογισμών.

1.1.2 Η μέθοδος αραιής αποθήκευσης COO

Ο κύριος σκοπός σε αυτήν την μορφή αποθήκευσης, η οποία είναι απο τις πιο διαδεδομένες, είναι να αποθηκεύσουμε μόνο τα μη μηδενικά στοιχεία ενός πίνακα εξετάζοντας το πίνακα κατά γραμμές. Έστω ότι συμβολίζουμε το πλήθος των μη μηδενικών στοιχείων με nz για ένα πραγματικό πίνακα διάστασης $n \times n$.

Σύμφωνα με τη μορφή αραιής αποθήκευσης coordinate format (COO) αποθηκεύουμε κάθε μη μηδενικό στοιχείο του πίνακα σε ένα διάνυσμα AA nz θέσεων, διατρέχοντας όλες τις γραμμές του πίνακα και έπειτα χρησιμοποιούμε δύο διανύσματα ja, ia μεγέθους nz τα οποία δείχνουν, την στήλη και την γραμμή που βρίσκεται αντίστοιχα κάθε στοιχείο του διανύσματος AA.

Για παράδειγμα αν

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

τότε τα διανύσματα AA, ja, και ia θα είναι αντίστοιχα:

$$\begin{array}{rcl} AA & = & (\ 1. \ \ 2. \ \ 3. \ \ 4. \ \ 5. \ \ 6. \ \ 7. \ \ 8. \ \ 9. \ \ 10. \ \ 11. \ \ 12. \ ) \\ ja & = & (\ 1 \ \ 4 \ \ 1 \ \ 2 \ \ 4 \ \ 1 \ \ 3 \ \ 4 \ \ 5 \ \ 3 \ \ 4 \ \ 5 \ ) \\ ia & = & (\ 1 \ \ 1 \ \ 2 \ \ 2 \ \ 2 \ \ 3 \ \ 3 \ \ 3 \ \ 3 \ \ 4 \ \ 4 \ \ 5 \ ) \end{array}$$

1.1.3 Η μέθοδος αραιής αποθήκευσης πινάκων CSR

Μια παραλλαγή της παραπάνω μεθόδου αποθήκευσης COO είναι η τεχνική CSR (compressed sparse row). Η διαφοροποίηση έγκειται στο γεγονός ότι στη θέση του διανύσματος δεικτών γραμμής ia υπάρχει ένα διάνυσμα με δείκτες που δείχνουν σε ποιά θέση του διανύσματος AA πραγματοποιείται αλλαγή της γραμμής. Αυτό είναι αρκετά πιο αποδοτικό σε σύγκριση με την COO δομή διότι το διάνυσμα ia θα είναι το πολύ n+1 θέσεων. Στην τελευταία θέση του διανύσματος ia αποθηκεύεται η τιμή nz+ia(1).

Η παραπάνω δομή αποθήκευσης είναι μία απο τις πιο διαδεδομένες και εύχρηστες για βασικές πράξεις γραμμικής άλγεβρας. Παρακάτω εμφανίζεται η τροποποίηση σε σχέση με την προηγούμενη αποθήκευση:

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

Τα διανύσματα AA, ja, και ia θα είναι αντίστοιχα:

$$\begin{array}{rcl} AA & = & (\ 1.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.\ ) \\ ja & = & (\ 1\ \ 4\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\ ) \\ ia & = & (\ 1\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\ ) \end{array}$$

1.1.4 Η μέθοδος αραιής αποθήκευσης πινάκων CSC

Η μέθοδος CSC (compressed sparse collumn) είναι μία δομή αποθήκευσης αραιών πινάκων που ανήκει στην οικογένεια των προηγούμενων δύο. Η διαφοροποίηση από την CSR δομή είναι η εναλλαγή αποθήκευσης των χαρακτηριστικών γραμμών και στηλών.

Παρακάτω εμφανίζεται στο ίδιο παράδειγμα η τροποποίηση αυτής της δομής.

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

Τα διανύσματα AA, ja, και ia θα είναι αντίστοιχα:

$$\begin{array}{rcl} AA & = & (\ 1.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.\ ) \\ ja & = & (\ 1\ \ 4\ \ 5\ \ 7\ \ 11\ \ 13\ ) \\ ia & = & (\ 1\ \ 2\ \ 3\ \ 2\ \ 3\ \ 4\ \ 1\ \ 2\ \ 3\ \ 4\ \ 3\ \ 5\ ) \end{array}$$

1.1.5 Η μέθοδος αραιής αποθήκευσης πινάκων MSR

Η τεχνική της δομής αποθήκευσης MSR (modified sparse row) αποτελεί μια παραλλαγή της CSR. Εκμεταλλεύεται το γεγονός ότι συνήθως στους πίνακες συντελεστών των αγνώστων απο τια προβλήματα που προκύπτουν απο τις διακριτοποιήσεις των ΜΔΕ τα διαγώνια στοιχεία είναι μη μηδενικά. Έτσι δεν χρειάζεται να αποθηκεύεται για αυτά κάποια πληροφορία. Σε αυτήν την δομή υπάρχουν μόνο δύο διανύσματα , το AA που αποθηκεύει τα στοιχεία του πίνακα και το διάνυσμα ja που είναι ένα διάνυσμα με δείκτες και άλλες σχετικές πληροφορίες για τα μη μηδενικά στοιχεία του πίνακα. Παρακάτω υπάρχει ένα παράδειγμα για την αποθήκευση τύπου MSR.

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

Στις πρώτες n θέσεις του AA θα αποθηκευτούν τα διαγώνια στοιχεία του πίνακα, ενώ η n+1 θέση μένει κενή (για κάποια μελλοντική πληροφορία του πίνακα). Έπειτα ξεκινώντας απο την θέση n+2 αποθηκεύονται όλα τα μη μηδενικά στοιχεία του πίνακα (ανα γραμμή) εκτός της διαγωνίου. Για κάθε μη διαγώνιο στοιχείο στην θέση $AA(k)$ αποθηκεύεται στο $ja(k)$ πληροφορία για την στήλη του στοιχείου. Στις n+1 θέσεις του διανύσματος ja έχουν αποθηκευτεί δείκτες που δείχνουν σε ποια θέση του διανύσματος AA υπάρχει στοιχείο που αλλάζει γραμμή στην δομή του πίνακα.

Τα διανύσματα AA ja θα είναι αντίστοιχα :

$$\begin{array}{rcccccccccccccc} AA & = & (& 1. & 4. & 7. & 11. & 12. & * & 2. & 3. & 5. & 6. & 8. & 8. & 10 &) \\ ja & = & (& 7 & 8 & 10 & 13 & 14 & 14 & 4 & 1 & 4 & 1 & 4 & 5 & 3 &) \end{array}$$

1.1.6 Η μέθοδος αραιής αποθήκευσης πινάκων DIAG

Εάν τα μη μηδενικά στοιχεία ενός πίνακα, βρίσκονται στις κύριες διαγώνιους του πίνακα τότε μία μορφή αποθήκευσης που μπορεί να χρησιμοποιηθεί είναι σύμφωνα με την τενχική DIAG. Η βασική ιδέα είναι να αποθηκευτούν ολόκληροι οι διαγώνιοι που περιέχουν μη μηδενικά στοιχεία σε έναν μικρότερο πίνακα. Θα χρειαστεί ένα διάνυσμα με ακέραιους, μήκους ίσο με το πλήθος των διαγωνίων που θα αποθηκευτούν οι δείκτες των στοιχείων κάθε στήλης του AA πίνακα σε σχέση με τη θέση τους στη διαγώνιο του πίνακα A .

Για παράδειγμα, ο πίνακας

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

θα έχει αποθήκευση σε μορφή DIAG ως εξής:

$$AA = \begin{pmatrix} * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11. & 12. & * \end{pmatrix} \quad ioff = (-1\ 0\ 2)$$

1.1.7 Η μέθοδος αραιής αποθήκευση πινάκων Ellpack-Itpack

Μια απο τις πρώτες σε εμφάνιση γενικές τεχνικές αποθήκευσης για αραιούς πίνακες είναι η μορφή Ellpack-Itpack.Η παραπάνω μορφή χρησινοποιείται κύρια στην υλοποίηση των αριθμητικών μεθόδων στις μαθηματικές βιβλιοθήκες Itpack και Ellpack. Σε αυτήν την δομή, μετράμε τα μη μηδενικα στοιχεία του πίνακα ανα γραμμή και κρατάμε τα περισσότερα στο πλήθος. Έστω ns. Για την αποθήκευση χρειάζονται 2 πίνακες μεγέθους $n \cdot ns$ ένας πραγματικός και ένας ακέραιος. Τα μη μηδενικά στοιχεία του A θα αποθηκευτούν στον πίνακα AA ανα γραμμή και στην αντίστοιχη γραμμή του ja θα αποθηκευτεί η πληροφορία για την στήλη του κάθε στοιχείου. Παρακάτω εμφανίζεται η δομή των πινάκων AA και ja σε αποθήκευση τύπου Ellpack-Itpack :

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

τότε οι πίνακες AA και ja θα είναι:

$$AA = \begin{pmatrix} 1. & 2. & 0. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & 0. \\ 11. & 12. & 0. \end{pmatrix} \quad ja = \begin{pmatrix} 1 & 3 & 1 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & 4 \\ 4 & 5 & 5 \end{pmatrix}$$

1.2 Αλγόριθμοι-βιβλιοθήκες υποπρογραμμάτων Γραμμικής Άλγεβρας για Αραιούς πίνακες

1.2.1 Βιβλιοθήκη υποπρογραμμάτων SPARSKIT

Εισαγωγή

Στην υλοποίηση των αλγορίθμων της παρούσας διατριβής έχει χρησιμοποιηθεί η βιβλιοθήκη $SPARSKIT2$,που περιλαμβάνει υποπρογράμματα αλγορίθμων βασικών πράξεων γραμμικής άλγεβρας για διάφορους τύπους αποθήκευσης αραιών πινάκων καθώς και επαναληπτικές μεθόδους για την επίλυση αραιών γραμμικών συστημάτων. Οι αλγόριθμοι έχουν αναπτυχθεί στη γλώσσα προγραμματισμού $Fortran\ 90$. Στη συνέχεια παρουσιάζονται οι αλγόριθμοι των υποπρογραμμάτων που χρησιμοποιήθηκαν απο αυτή τη βιβλιοθήκη, αλλα και αυτοί που χρειάστηκε να τροποποιηθούν ή να κατασκευαστούν στη παρούσα διατριβή.

Το υποπρόγραμμα AMUX

Το υποπρόγραμμα AMUX υλοποιεί τον αλγόριθμο πολλαπλασιασμού πίνακα με διάνυσμα σε δομή αποθήκευσης CSR. (Y=A*X) . Δηλαδή υπολογίζεται το διάνυσμα Y ως γινόμενο του πίνακα A με το διάνυσμα κατάλληλης διάστασης X. Η κλήση του υποπρογράμματος γίνεται ως

call AMUX (N , X , Y , A , JA , IA)

Με τα παρακάτω ορίσματα :

Ορίσματα εισόδου :

N : ακέραιος, διάσταση του πραγματικού πίνακα A διάστασης $N \times N$.

X : πραγματικό διάνυσμα N θέσεων.

A : πραγματικό διάνυσμα nz θέσεων που περιέχει τα μη μηδενικά στοιχεία του πίνακα A.

JA : ακέραιο διάνυσμα nz θέσεων που περιέχει τις πληροφορίες για τις στήλες των μη μηδενικών στοιχείων του A.

IA : ακέραιο διάνυσμα N+1 θέσεων που περιέχει τους δείκτες γραμμής των στοιχείων του A σύμφωνα με την τεχνική αποθήκευσης CSR.

Ορίσματα εξόδου :

Y : πραγματικό διάνυσμα N θέσεων, που περιέχει το αποτέλεσμα της πράξης A*X.

Αλγόριθμος AMUX:

Για i =1,N

t=0

Για k=$ia(i)$, $ia(i+1)-1$

$t = t + a(k) * x(ja(k))$

Τέλος Για

$y(i) = t$

Τέλος Για

Το υποπρόγραμμα ατελούς διάσπασης πίνακα ILU(k)

Για τη διάσπαση LU σε ένα γραμμικό σύστημα, χρησιμοποιήθηκε η διαδικασία της ατελούς διάσπασης $ILU(k)$. Η $ILU(k)$ είναι μια ατελής παραγοντοποίηση του πίνακα

Α με κ επίπεδο γεμίσματος του πίνακα $(level\ of\ filling)$. Η διαδικασία αυτή είναι αποδοτικότερη σε σχέση με μία πλήρη διάσπαση του πίνακα, επειδή πραγματοποιείται εξοικονόμηση μνήμης και πράξεων, προσθέτοντας ένα ελεγχόμενο σφάλμα, το οποίο σε συνδυασμό με αυτό της τεχνικής πολυπλέγματος διορθώνεται επαναληπτικά. Ειδικότερα η κλήση του υποπρογράμματος γίνεται ως :

call ILUk (N , A , JA , IA , LFIL , ALU , JLU , JU , LEVS , IWK , W , JW , IERR)

Με τα παρακάτω ορίσματα :

Ορίσματα εισόδου :

N : ακέραιος, διάσταση του πραγματικού πίνακα A, διάστασης $N \times N$

A : πραγματικό διάνυσμα nz θέσεων που περιέχει τα μη μηδενικά στοιχεία του πίνακα A

JA : ακέραιο διάνυσμα nz θέσεων που περιέχει τις πληροφορίες για τις στήλες των μη μηδενικών στοιχείων του A

IA : ακέραιο διάνυσμα N+1 θέσεων που περιέχει τους δείκτες γραμμής των στοιχείων του A σύμφωνα με την τεχνική αποθήκευσης CSR.

$LFIL$: ακέραιος , κάθε στοιχείο του οποίου το επίπεδο γεμίσματος του, υπερβαίνει το $LFIL$ απορίπτεται

IWK : ακέραιος, το ελάχιστο μήκος των διανυσμάτων ALU, JLU και $LEVS$

Ορίσματα εξόδου :

ALU : πραγματικό διάνυσμα που περιέχει τους πίνακες L και U, όπου $A = L \cdot U$ αποθηκευμένους σε δομή MSR . Σε κάθε i γραμμή του πίνακα είναι αποθηκευμένα πρώτα τα στοιχεία του L (χωρίς την διαγώνιο με την μονάδα) και έπειτα ακολουθεί ο πίνακας U

JLU : ακέραιο διάνυσμα, βοηθητικό διάνυσμα του ALU απαραίτητο για την MSR αποθήκευση

JU : ακέραιο διάνυσμα το οποίο περιέχει πληροφορίες δεικτών του πίνακα U

$LEVS$: ακέραιο διάνυσμα μεγέθους iwk το οποίο περιέχει πληροφορίες του επίπεδου

γεμίσματος κάθε στοιχείου του πίνακα A

$IERR$: ακέραιος,περιλαμβάνει πληροφορία για την σωστή λειτουργία του προγράμματος, όπου εάν:

$IERR = 0$ το πρόγραμμα τερματίστηκε επιτυχώς

$IERR > 0$ δεν έγινε οδήγηση στο βήμα $IERR$

$IERR = -1$ ο πίνακας που δόθηκε σαν όρισμα πιθανόν είναι λάθος

$IERR = -2$ ο πίνακας L δεν χωράει στο διάνυσμα ALU

$IERR = -3$ ο πίνακας U δεν χωράει στο διάνυσμα ALU

$IERR = -4$ μή έγκυρη τιμή του $LFIL$

$IERR = -5$ μηδενική γραμμή στον πίνακα A ή U

Χώρος εργασίας :

JW : ακέραιο διάνυσμα εργασίας διάστασης $3 * n$

W : πραγματικό διάνυσμα εργασίας διάστασης n

Αλγόριθμος ILU(k) :

Για κάθε μη μηδενικό στοιχείο $a_{i,j}$ ορίζουμε $lev(a_{i,j}) = 0$

Για $i = 1, N$

Για $k = 1, i-1$ και για $lev(a_{i,k}) \leq LFIL$

$a_{i,k} = a_{k,k}/a_{k,k}$

$a_{i,*} = a_{i,*} - a_{i,k} * a_{k,*}$

$lev(a_{i,*}) = min(lev(a_{i,*}), lev(a_{i,k}) + lev(a_{k,*}) + 1)$

Τέλος Για

Εάν $lev(a_{i,*}) > LFIL$ τότε $a_{i,j} = 0$

Τέλος Για

Το υποπρόγραμμα ατελούς επίλυσης LUSOL

Το υποπρόγραμμα $LUSOL$ υλοποιεί τον αλγόριθμο της προς τα εμπρός και πίσω αντικατάστασης της απαλοιφής Gauss για την άμεση επίλυση ενός γραμμικού συστήματος. Δέχεται σαν όρισμα τα ορίσματα εξόδου του υποπρογράμματος $ILU(k)$. Ειδικότερα, η κλήση του πραγματοποιείται με την εντολή :

call LUSOL (N , Y , X , ALU , JLU , JU)

Με τα παρακάτω ορίσματα :

Ορίσματα εισόδου :

N : ακέραιος, διάσταση του πραγματικού πίνακα A,διάστασης $N \times N$

Y : πραγματικό διάνυσμα , το δεξί μέλος του γραμμικού συστήματος $Ax = y$

ALU : ο πίνακας LU όπως προέκυψε από την κλήση της $ILU(k)$

JLU : ακέραιο διάνυσμα των δεικτών του LU

JU : ακέραιο διάνυσμα των δεικτών του U

Ορίσματα εξόδου :

X : πραγματικό διάνυσμα, διάστασης N το οποίο περιλαμβάνει τη λύση του γραμμικού συστήματος

Αλγόριθμος LUSOL :

Για $i = 1, N$

$x(i) = y(i)$

Για $k = jlu(i), ju(i) - 1$

$x(i) = x(i) - alu(k) * x(jlu(k))$

Τέλος Για

Τέλος Για

Για $i = N, 1, -1$

Για $k = ju(i), jlu(i+1) - 1$

$x(i) = x(i) - alu(k) * x(jlu(k))$

Τέλος Για

$x(i) = alu(i) * x(i)$

Τέλος Για

1.2.2 Το υποπρόγραμμα εξωτερικού γινομένου πινάκων KRONS

Το $KRONS$ κατασκευάστηκε από την αρχή και υλοποιεί το γινόμενο $kronecker$ ή εξωτερικό γινόμενο μεταξύ δύο αραιών πινάκων οι οποίοι έχουν αποθηκευτεί σύμφωνα με την τεχνική CSR. Εάν A και B δύο πραγματικοί πίνακες διάστασης $ma \times na$ και $mb \times nb$ αντίστοιχα τότε το αποτέλεσμα του εξωτερικού τους γινομένου είναι ο πίνακας C διάστασης $ma \cdot mb \times na \cdot nb$. Η κλήση του προγράμματος γίνεται ως :

call KRONS (MA , NA , A , JA , IA , MB , NB , B , JB , IB , C , JC , IC)

με τα ορίσματα εισόδου :

Ορίσματα εισόδου :

MA : ακέραιος, οι γραμμές του πίνακα A

NA : ακέραιος, οι στήλες του πίνακα A

A : πραγματικό διάνυσμα διάστασης nzA, το οποίο περιέχει τον πίνακα A

JA : ακέραιο διάνυσμα διάστασης nzA, το οποίο περιέχει πληροφορία για τις στήλες των στοιχείων του A

IA : ακέραιο διάνυσμα διάστασης $ma + 1$, το οποίο περιέχει τους δείκτες γραμμής των στοιχείων του A σύμφωνα με την τεχνική αποθήκευσης CSR.

MB : ακέραιος, οι γραμμές του B NB : ακέραιος, οι στήλες του πίνακα B

B : πραγματικό διάνυσμα διάστασης nzB, το οποίο περιέχει τον πίνακα B

JB : ακέραιο διάνυσμα διάστασης nzB, το οποίο περιέχει πληροφορία για τις στήλες των στοιχείων του B

IB : ακέραιο διάνυσμα διάστασης $mb + 1$, το οποίο περιέχει τους δείκτες γραμμής των στοιχείων του B σύμφωνα με την τεχνική αποθήκευσης CSR.

Ορίσματα εξόδου :

C : πραγματικό διάνυσμα διάστασης $nzA * nzB$, που περιέχει τον πίνακα γινομένου JC

: ακέραιο διάνυσμα διάστασης $nzA * nzB$, το οποίο περιέχει πληροφορία για τις στήλες των στοιχείων του C

IC : ακέραιο διάνυσμα διάστασης $ma * mb + 1$, το οποίο περιέχει τους δείκτες γραμμής των στοιχείων του C σύμφωνα με την τεχνική αποθήκευσης CSR.

Αλγόριθμος KRONS:

$q = 0$ Για $k = 1, ma$

Για $i = 1, mb$

Για $l = ia(k), ia(k+1) - 1$

Για $j = ib(i), ib(i+1) - 1$

$q = q + 1$

$c(q) = a(l) * b(j)$

$iac(q) = (k-1) * mb + i$

$jc(q) = (ja(l) - 1) * nb + jb(j)$

Τέλος Για

Τέλος Για

Τέλος Για

Τέλος Για

Ο παραπάνω αλγόριθμος αποθηκεύει το γινόμενο με τεχνική COO, οπότε χρειάζεται η κατάλληλη μετατροπή του iac απο COO σε CSR και αποθήκευση στο ic.

1.2.3 Το υποπρόγραμμα DSMSM

Το υποπρόγραμμα $DSMSM$ υλοποιεί τον αλγόριθμο της πράξης: $C = alpha \cdot A \cdot B + C$, όπου ο A είναι διαγώνιος πίνακας και οι B και C είναι αραιοί πίνακες σε δομή αποθή-

κευσης CSR Η κλήση του γίνεται ως :

call DSMSM (M , N , ALPHA , A , B , BETA , C , JC , IC)

Με τα παρακάτω ορίσματα :

Ορίσματα εισόδου :

M : ακέραιος, οι γραμμές των πινάκων A, B, C

N : ακέραιος, οι στήλες των πινάκων A, B, C

$ALPHA$: πραγματικό, βαθμωτό

A : πραγματικό διάνυσμα, το οποίο περιέχει την διαγώνιο του A

B : πραγματικό διάνυσμα, το οποίο περιέχει τον πίνακα B

$BETA$: πραγματικό, βαθμωτό

C : πραγματικό διάνυσμα το οποίο περιέχει τον πίνακα C

JC : ακέραιο διάνυσμα, το οποίο περιέχει πληροφορία για τις στήλες των πινάκων

IC : ακέραιο διάνυσμα, το οποίο περεέχει τους δείκτες γραμμής των στοιχείων του A σύμφωνα με την τεχνική αποθήκευσης CSR.

Ορίσματα εξόδου :

C : πραγματικό διάνυσμα, το οποίο ανανενώνεται και περιλαμβάνει το αποτέλεσμα της πράξης

Αλγόριθμος DSMSM:

Για $i = 1, M$

Για $j = ic(i), ic(i+1) - 1$

$C(j) = alpha * A(i) * B(j) + C(j)$

Τέλος Για

Τέλος Για

Κεφάλαιο 2

Τεχνικές Πολυπλέγματος

2.1 Εισαγωγή

Στην αριθμητική ανάλυση, οι τεχνικές πολυπλέγματος αποτελούν ένα σύνολο αλγορίθμων στις αριθμητικές μεθόδους επίλυσης διαφορικών εξισώσεων χρησιμοποιώντας μια ακολουθία απο κλιμακούμενες διακριτοποιήσεις [1, 6, 7, 8, 9, 10, 11, 14, 15, 28, 33, 36]. Η κύρια ιδέα των τεχνικών πολυπλέγματος είναι η αύξηση της σύγκλισης μιας βασικής επαναληπτικής διαδικασίας χρησιμοποιώντας σχήματα διόρθωσης από βήμα σε βήμα. Αυτό επιτυγχάνεται λύνοντας ένα αραιότερο πρόβλημα και η κύρια ιδέα ταυτίζεται με την ιδέα της παρεμβολής μεταξύ αραιών και πυκνών διακριτοποιήσεων.

Η πιο συνηθισμένη εφαρμογή των τεχνικών πολυπλέγματος είναι η αριθμητική επίλυση μερικών διαφορικών εξισώσεων ελλειπτικού τύπου σε δύο ή τρεις διαστάσεις. Οι τεχνικές πολυπλέγματος μπορούν να συνδυαστούν με οποιαδήποτε απο τις συνηθισμένες τεχνικές διακριτοποίησης. Έτσι για παράδειγμα, μπορούν να συνδυσαστούν με μεθόδους πεπερασμένων στοιχείων , όπου συνολικά κατασκευάζουν μια απο τις ταχύτερες τεχνικές επίλυσης διαφορικών εξισώσεων εώς και σήμερα. Σε σύγκριση με άλλες μεθόδους, οι τεχνικές πολυπλέγματος είναι εύκολα εφαρμόσιμες σε διάφορους τύπους προβλημάτων και συνοριακών συνθηκών, χωρίς να εξαρτώνται απο ειδικές ιδιότητες του ΠΣΤ. Επίσης, μπορεί να γίνει η εφαρμογή τους σε πιο σύνθετα προβλήματα, μη συμμετρικά και μη γραμμικά συστήματα εξισώσεων όπως το σύστημα $Lam\hat{e}$ για την ελαστικότητα ή τις εξισώσεις $Navier - Stokes$.

2.2 Διακριτοποίηση Προβλημάτων Συνοριακών Τιμών

2.2.1 Εισαγωγή

Σε αυτή την παράγραφο, περιγράφεται σύντομα η πιο απλή και εύχρηστη αριθμητική μέθοδος επίλυσης ΠΣΤ. Αυτή είναι η μέθοδος των πεπερασμένων διαφορών σε μια και δύο διαστάσεις, ενώ εφαρμόζεται στο μοντέλο πρόβλημα Poisson , για την παρουσίαση των απαραίτητων βασικών εννοιών της τεχνικής πολυπλέγματος.

2.2.2 Πρόβλημα Poisson μιας διάστασης

Θεωρούμε την μονοδιάστατη εξίσωση $Poisson$ στη μορφή :

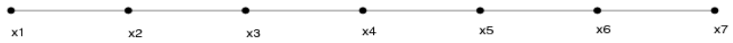
$$u_{xx} = f(x) \quad , \quad x \in [0,1]$$
$$u(0) = u(1) = 0$$

και την ομοιόμορφη διαμέριση του χωρίου [0,1] σε n υποδιαστήματα μήκους $h = \frac{1}{n}$ με κόμβους τα σημεία $x_i = (i-1) \cdot h$ για $i = 1, \ldots, n+1$. Χρησιμοποιώντας το ανάπτυγμα της σειράς $Taylor$ προκύπτει ο παρακάτω προσεγγιστικός τύπος για την δεύτερη παράγωγο θεωρώντας ότι $u_i = u(x_i)$

$$u_i^{''} = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2)$$

Αντικαθιστώντας την δεύτερη παράγωγο στο ΠΣΤ, προκύπτει ότι :

$$u_{i-1} - 2u_i + u_{i+1} = h^2 \cdot f(x_i)$$



Σχήμα 2.1: Ομοιόμορφη διαμέριση στο $[0,1]$ για $n = 6$.

κι έτσι απο τις έξι εξισώσεις θα προκύψει το γραμμικό σύστημα $Au = f$ με πίνακα συντελεστών των αγνώστων της μορφής :

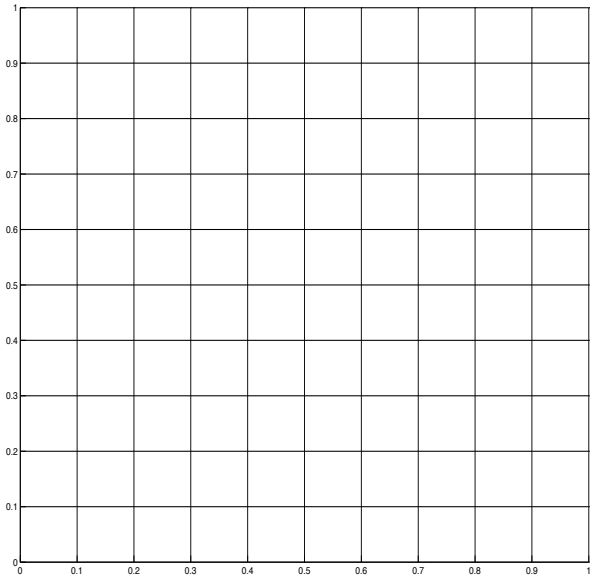
$$A = \begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{bmatrix}$$

2.2.3 Πρόβλημα Poisson δύο διαστάσεων

Έστω η εξίσωση $Poisson$ δύο διαστάσεων :

$$u_{xx} + u_{yy} = f(x,y) \ , \ (x,y) \in \Omega = (0,1) \times (0,1)$$
$$u = 0 \quad , \quad (x,y) \in \partial\Omega$$

Θεωρούμε ομοιόμορφη διαμέριση για κάθε χωρική κατεύθυνση με βήμα $h = \frac{1}{n}$ με τους $x_i = (i-1) \cdot h$ και $y_j = (j-1) \cdot h$,όπου $i, j = 1, \ldots, n+1$. Χρησιμοποιώντας τα αναπτύγματα $Taylor$ της μιας διάστασης για κάθε όρο ξεχωριστά στο ΠΣΤ θα προκύψει η εξίσωση :



Σχήμα 2.2: Ομοιόμορφη διαμέριση του $[0,1] \times [0,1]$ για $n = 10$.

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = h^2 f(x_i, y_j)$$

Οπότε εάν γραφούν οι εξισώσεις σε μορφή συστήματος θα σχηματιστεί ο πίνακας :

$$A = \begin{bmatrix} D & I & & \\ I & D & I & \\ & I & D & I \\ & & I & D \end{bmatrix} \text{ όπου } D = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & 1 & -4 & 1 \\ & & 1 & -4 \end{bmatrix} \text{ και } I = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

Αντίστοιχα με το πρόβλημα της μιας διάστασης, η λύση της διαφορικής εξίσωσης προσεγγίζεται από τη λύση ενός γραμμικού συστήματος : $Au = f$.

2.3 Βασικές έννοιες Τεχνικών Πολυπλέγματος

Ας υποθέσουμε ότι u είναι η πραγματική λύση μιας διαφορικής εξίσωσης και v είναι η αριθμητική της προσέγγιση. Τότε ορίζουμε ως αριθμητικό σφάλμα της εξίσωσης την διαφορά:

$$e = u - v \tag{2.1}$$

Στα αριθμητικά προβλήματα συνηθίζεται το σφάλμα να μετριέται με κάποια νόρμα , όπως αυτή του άπειρου ή η νόρμα 2, οι οποίες ορίζονται αντίστοιχα:

$$\|e\|_\infty = max_{1\leq i\leq n}|e_i| \quad \text{και} \quad \|e\|_2 = (\sum_{i=1}^{n} e_i^2)^{1/2} \tag{2.2}$$

Στην πραγματικότητα όμως, το διάνυσμα του σφάλματος είναι δύσκολο να υπολογιστεί αφού είναι αναγκαία η γνώση της πραγματικής λύσης της εξίσωσης. Εάν όμως αντικατασταθεί στο αρχικό σύστημα η αριθμητική λύση v, τότε μπορεί εύκολα να οριστεί η έννοια του διανύσματος υπολοίπου r που προκύπτει απο την διαφορα:

$$r = f - Av \tag{2.3}$$

Με συνδυασμό αυτών των δύο εννοιών , μπορεί να παραχθεί μία νέα εξίσωση, αυτή του υπολοίπου, η οποία προκύπτει ως εξής :

$$Au = f \Leftrightarrow A(v + e) = f \Leftrightarrow Av + Ae = f \Leftrightarrow Ae = f - Av \Leftrightarrow Ae = r \tag{2.4}$$

Λύνοντας το σύστημα $Ae = r$ θα προκύψει το διάνυσμα του σφάλματος e με το οποίο θα βελτιωθεί η αριθμητική λύση του συστήματος

$$u = v + e \tag{2.5}$$

Ο συνδυασμός αυτών των εξισώσεων ονομάζεται σχήμα διόρθωσης , γιατί είναι μια βασική τεχνική για να διορθωθεί η αριθμητική λύση v επαναληπτικά. Στις τεχνικές πολυπλέγ-

ματος το σχήμα διόρθωσης κατέχει πολύ σημαντικό ρόλο.

Στη συνέχεια θα παραχθεί η επαναληπτική μέθοδος Gauss-Seidel [16, 17, 31, 34, 35], η οποία ανήκει στις βασικές επαναληπτικές μεθόδους. Γενικά, είναι μια επαναληπτική μέθοδος με αργό ρυθμό σύγκλισης, αλλά για τις τεχνικές πολυπλέγματος αποτελεί ένα πολύτιμο εργαλείο εξαιτίας της ιδιότητας που έχει να εξομαλύνει γρήγορα το σφάλμα.
Ας υποθέσουμε ότι έχουμε έναν πίνακα συντελεστών A, σε ένα γραμμικό σύστημα εξισώσεων. Θεωρούμε την διάσπαση του:

$$A = D - L - U \tag{2.6}$$

Όπου D διαγώνιος πίνακας και L και U κάτω και άνω τριγωνικοί αντίστοιχα.
Εάν αντικαταστήσουμε την σχέση (2.6) στο γραμικό σύστημα $Au = f$ τότε:

$$(D - L - U)u = f \tag{2.7}$$

Εάν κρατήσουμε το κάτω τριγωνικό κομμάτι του A ξεχωριστά η σχέση θα γίνει ως εξής:

$$\begin{aligned} (D - L)u - Uu &= f \\ (D - L)u &= Uu + f \end{aligned} \tag{2.8}$$

κι έτσι η εξίσωση μπορεί να γραφεί:

$$u = (D - L)^{-1}Uu + (D - L)^{-1}f \tag{2.9}$$

Επομένως θεωρούμε τον επαναληπτικό πίνακα της μεθόδου $Gauss - Seidel$ $R_{GS} = (D - L)^{-1}U$ και η παραπάνω σχέση ξαναγράφεται ως εξής:

$$u = R_{GS}u + (D - L)^{-1}f \tag{2.10}$$

Η τελική εξίσωση (2.10) εάν χωριστεί σε επαναληπτικά χρονικά βήματα m τότε η εξίσωση $Gauss - Seidel$ γράφεται :

$$v^{(m+1)} = R_{GS}v^m + (D - L)^{-1}f \tag{2.11}$$

Αφαιρώντας τις σχέσεις (2.11) και (2.10) κατα μέλη και λαμβάνοντας υπόψη την σχέση $e = u - v$ τότε θα προκύψει μια νέα σχέση για το σφάλμα της εξίσωσης :

$$e^{(m+1)} = R_{GS} e^{(m)} \tag{2.12}$$

ή

$$(D - L)e^{(m+1)} = Ue^{(m)} \tag{2.13}$$

Το γεγονός ότι η επαναληπτική μέθοδος $Gauss - Seidel$ μπορεί και εξομαλύνει γρήγορα το σφάλμα της εξίσωσης, μπορεί να εξηγηθεί με την χρήση ιδιοσυναρτήσεων . Εαν υποθέσουμε ότι το σφάλμα $e = e_h(x, y)$, είναι μία συνάρτηση των διακριτών μεταβλητών x, y σε μια διακριτοποίηση h τότε μπορεί να γραφεί στην μορφή :

$$e_h(x, y) = \sum_{k,l=1}^{n-1} a_{k,l} \sin(k\pi x) \sin(l\pi y) \tag{2.14}$$

Εάν L είναι ο τελεστής της διαφορικής εξίσωσης, τότε Δ_h θα είναι ο διακριτός τελεστής της διαφορικής εξίσωσης. Άρα $\forall (x, y) \in \Omega_h$ οι

$$\phi_h^{k,l}(x, y) = \sin(k\pi x) \sin(l\pi y) \quad (k, l = 1, \ldots, n - 1) \tag{2.15}$$

είναι οι διακριτές ιδιοσυναρτήσεις του διακριτού τελεστή Δ_h.
Το γεγονός ότι το σφάλμα γίνεται πιο ομαλό μετά από κάποια επαναληπτικά βήματα σημαίνει ότι οι υψηλής συχνότητας όροι, δηλαδή

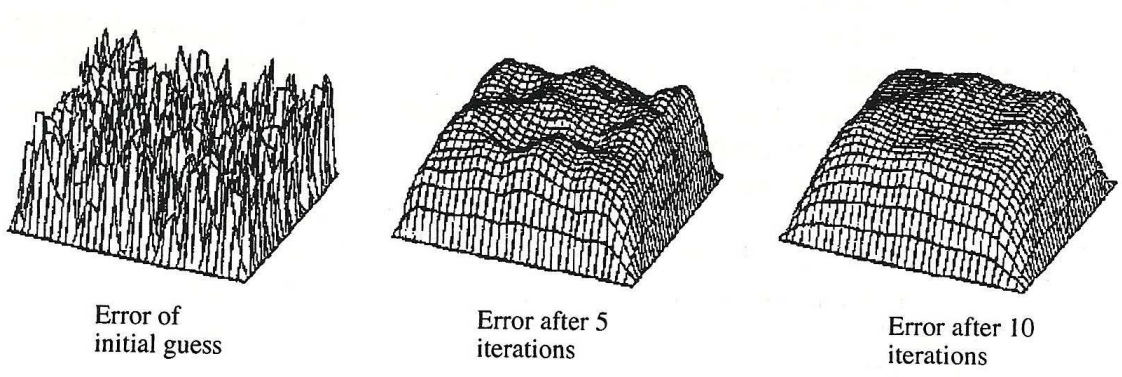
$$a_{k,l} \sin(k\pi x) \sin(l\pi y) \text{ για μεγάλα } k, l \tag{2.16}$$

γίνονται μικροί μετά από κάποιες επαναλήψεις, ενώ οι χαμηλής συχνότητας όροι

$$a_{k,l} \sin(k\pi x) \sin(l\pi y) \text{ για μικρά } k, l \tag{2.17}$$

δύσκολα μεταβάλλονται, προκύπτει απο την αντικατάσταση των αναπτυγμάτων των σφαλμάτων στην εξίσωση 2.13. Η διάκριση μεταξύ των υψηλών και χαμηλών συχνοτήτων

είναι σημαντική για τις τεχνικές πολυπλέγματος.



Σχήμα 2.3: Η ιδιότητα του εξομαλυντή , να διορθώνει το σφάλμα σε λίγες επαναλήψεις

Ας επανέλθουμε πάλι στο πρόβλημα μοντέλο $Poisson$ δύο διαστάσεων σε ένα πλέγμα Ω_h με βήμα διακριτοποίησης $h = 1/n$. Αντίστοιχα ας θεωρήσουμε το ίδιο πρόβλημα σε ένα αραιότερο πλέγμα Ω_H, όπου το $H > h$. Χωρίς βλάβη της γενικότητας εάν θεωρήσουμε το h να είναι ζυγός αριθμός, τότε μπορούμε να επιλέξουμε $H = 2h$, που είναι η φυσική επιλογή για τις τεχνικές πολυπλέγματος. Αυτή η επιλογή αραιότερου πλέγματος ονομάζεται συνηθισμένη αραίωση πλέγματος.

Για τον σαφή ορισμό των όρων υψηλής και χαμηλής συχνότητας , επιστρέφουμε στις ιδιοσυναρτήσεις (2.15) με $\phi^{k,l} = \phi_h^{k,l}$. Για δοσμένα (k, l) θεωρούμε τις τέσσερις ιδιοσυναρτήσεις:

$$\phi^{k,l}, \quad \phi^{n-k,n-l}, \quad \phi^{n-k,l}, \quad \phi^{k,n-l}$$

και παρατηρούμε ότι ως ιδιοσυναρτήσεις του Ω_{2h} ισχύει η σχέση:

$$\phi^{k,l}(x,y) = -\phi^{n-k,l}(x,y) = -\phi^{k,n-l}(x,y) = \phi^{n-k,n-l}(x,y) \quad \text{για} \quad (x,y) \in \Omega_{2h}$$

Αυτό σημαίνει ότι αυτές οι ιδιοσυναρτήσεις δεν μπορούν να εξαφανιστούν στο Ω_{2h}, ενώ π.χ. για $l = n/2$ η $\phi^{k,l}$ θα απαλοιφθεί στο Ω_{2h}. Με βάση τα παραπάνω μπορούμε να ορίζουμε για $k, l \in \{1, \ldots, n-1\}$ ότι μια ιδιοσυνάρτηση $\phi^{k,l}$ είναι :

χαμηλής συχνότητας εάν $max(k,l) < n/2$

υψηλής συχνότητας εάν $n/2 \leq max(k,l) < n$

Προφανώς , μόνο οι όροι χαμηλής συχνότητας θα είναι ορατοί στο Ω_{2h}, αφού όλοι οι όροι υψηλής συχνότητας μετατρέπονται σε όρους χαμηλής συχνότητας ή εξαφανίζονται στο Ω_{2h}.

Εάν εφαρμόσουμε τον ορισμό της ταξινόμησης των ιδιοσυχνοτήτων στην σχέση (2.14) τότε το άθροισμα θα χωριστεί σε δύο αντίστοιχα αθροίσματα :

$$\sum_{k,l=1}^{n-1} a_{k,l}\phi^{k,l} = \sum^{high} a_{k,l}\phi^{k,l} + \sum^{low} a_{k,l}\phi^{k,l} \tag{2.18}$$

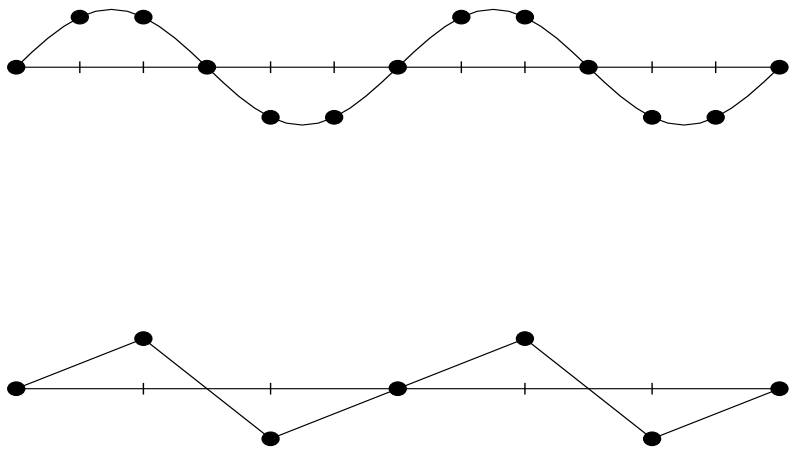
όπου

$$\sum^{low} a_{k,l}\phi^{k,l} = \sum_{k,l=1}^{n/2-1} a_{k,l}\phi^{k,l} \tag{2.19}$$

και

$$\sum^{high} a_{k,l}\phi^{k,l} = \sum_{k,l=n/2 \leq max(k,l)}^{n-1} a_{k,l}\phi^{k,l} \tag{2.20}$$

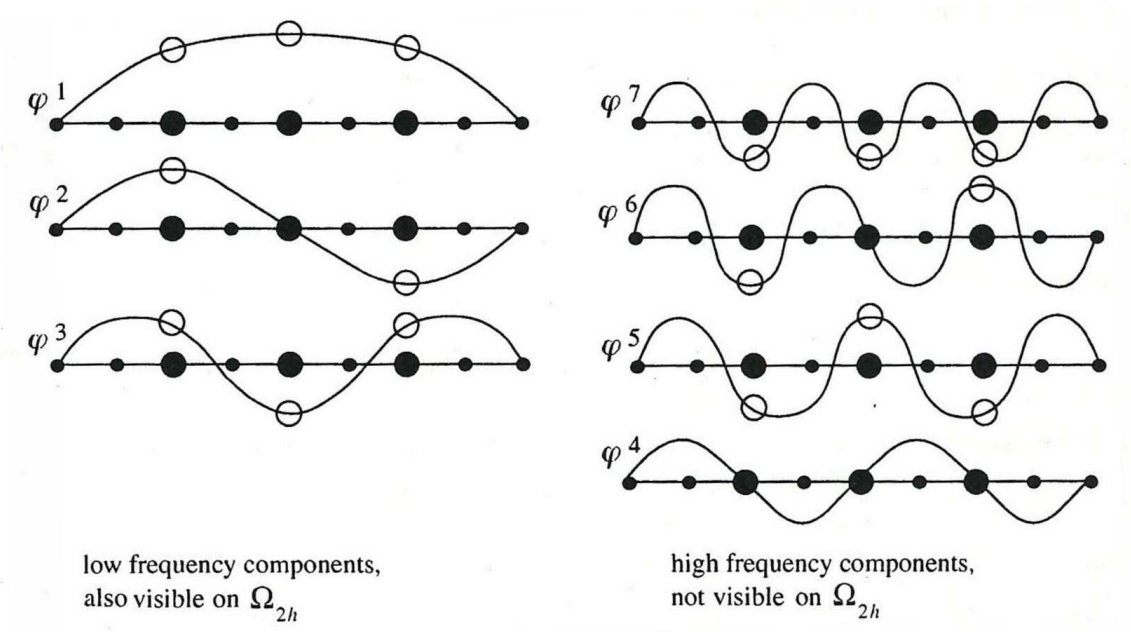
Το σχήμα εξομάλυνσης της μεθόδου $Gauss - Seidel$, όπως και τα υπόλοιπα σχήματα εξομάλυνσης, έχει την ιδιότητα να διορθώνει γρήγορα τους όρους του σφάλματος με υψηλή συχνότητα, ενώ να δυσκολεύται να εξομαλύνει αυτούς με την χαμηλή και αυτό



Σχήμα 2.4: Το αραιό πλέγμα (κάτω) αναγνωρίζει μια συνάρτηση ως λιγότερο ομαλή από ότι το πιο πυκνό πλέγμα(πάνω).

αποτελεί ένα μεγάλο μειονέκτημα της μεθόδου, αφού μειώνει την τάξη σύγκλισης της. Κάνοντας χρήση των βασικών ιδιοτήτων των τεχνικών πολυπλέγματος που ήδη έχουν αναφερθεί , παρακάτω αναλύεται ο τρόπος εκμετάλευσης αυτού του μειονεκτήματος, ώστε να αυξηθεί η τάξη σύγκλισης της μεθόδου.

Η κύρια ιδέα είναι να υπολογιστεί μια καλύτερη αρχική προσέγγιση της λύσης, ώστε να αυξηθεί η τάξη σύγκλισης. Αυτό επιτυγχάνεται πραγματοποιώντας ένα μικρό αριθμό επαναλήψεων σε αραιότερο πλέγμα και κάνοντας παρεμβολή της λύσης αυτής στο πυκνότερο. Αναλυτικά, η λύση του συστήματος σε ένα μικρότερο πλέγμα θα είναι μια ταχύτερη υπολογιστικά διαδικασία λόγω του νέου μεγέθους του προβλήματος. Όμως το πιο αποτελεσματικό σε αυτή τη διαδικασία είναι ότι σε ένα αραιότερο πλέγμα, οι ιδιοσυναρτήσεις υψηλών συχνοτήτων θα εξαφανιστούν και αυτές των χαμηλών συχνοτήτων θα μετατραπούν σε υψηλές. Λόγω της ιδιότητας ότι η $Gauss - Seidel$ μπορεί να εξομαλύνει γρήγορα τους όρους υψηλών συχνοτήτων, τότε στο αραιό πλέγμα σε λίγα μόνο βήματα θα εξομαλύνει τις ιδιοσυναρτήσεις που για το πυκνό πλέγμα θεωρούνται χαμηλών συ-



Σχήμα 2.5: Οι όροι με χαμηλές συχνότητες είναι ορατοί στο Ω_H (αριστερά), ενώ οι όροι υψηλών συχνοτήτων δεν είναι ορατοί (δεξιά)

χνοτήτων. Αυτό φαίνεται στο σχήμα (2.4) , όπου μια ιδιοσυνάρτηση χαμηλής συχνότητας μετατρέπεται σε μια λιγότερη ομαλή ιδιοσυνάρτηση υψηλής συχνότητας. Επομένως είναι κατανοητό ότι μόλις εξομαλυνθούν όλοι οι όροι υψηλής συχνότητας σε ένα πλέγμα, τότε μεταφερόμαστε σε ένα αραιότερο πλέγμα, όπου και συνεχίζουμε εκεί την επίλυση του προβλήματος. Αυτή η διαδικασία μπορεί να επαναληφθεί αρκετές φορές μειώνοντας συνεχώς το πλέγμα. Όμως ο αρχικός μας στόχος είναι μια καλύτερη προσέγγιση της αρχικής λύσης που θα λάβει το σύστημα, άρα όλα τα παραπάνω αφορούν την εξίσωση υπολοίπου και το σχήμα διόρθωσης.

2.4 Η τεχνική των δύο πλεγμάτων

Όπως παρουσιάστηκε στη προηγούμενη ενότητα το αριθμητικό σχήμα διόρθωσης της προσεγγιστικής λύσης $e_h = u_h - v_h$ βασισμένη στο πλέγμα Ω_h θα περιγράφεται από τη διαδικασία:

Υπολογισμός διανύσματος υπολοίπου:

$$r_h = f_h - A_h v_h \tag{2.21}$$

Επίλυση γραμμικού συστήματος υπολοίπου:

$$A_h e_h = r_h \tag{2.22}$$

Άρα, καταλήγουμε ότι

$$u_h = v_h + e_h \tag{2.23}$$

Για να προσπαθήσουμε να επεκτείνουμε αυτή την τεχνική διόρθωσης ανάμεσα σε δύο πλέγματα Ω_h και Ω_H με βήματα διακριτοποίησης $h = 1/n$ και $H = 2h$ αντίστοιχα, υπάρχει η ανάγκη να οριστούν δύο τελεστές I_h^H και I_H^h. Οι τελεστές αυτοί ονομάζονται τελεστές μεταφοράς πλεγμάτων ή τελεστές παρεμβολής και παρεκβολής αντίστοιχα. Στην επόμενη παράγραφο θα παρουσιαστούν αναλυτικότερα αυτοί οι τελεστές μεταφοράς.

Θεωρούμε ότι $I_h^H : \mathcal{G}(\Omega_h) \to \mathcal{G}(\Omega_H)$ και $\mathcal{G}(\Omega_H) \to \mathcal{G}(\Omega_h)$

τότε ο τελεστής I_h^H χρησιμοποιείται για να παρεκβάλουμε τη τιμή r_h στο πλέγμα Ω_H ως

$$r_H = I_h^H r_h \tag{2.24}$$

ενώ ο τελεστής I_H^h χρησιμοποιείται για να παρεμβάλουμε τις τιμές της διόρθωσης e_H στο πλέγμα Ω_h

$$e_h = I_H^h e_H \tag{2.25}$$

Αλγοριθμικά αυτή η διαδικασία διόρθωσης θα είναι:

Υπολογισμός του διανύσματος υπολοίπου:

$$r_h = f_h - A_h v_h \tag{2.26}$$

Παρεκβολή του διανύσματος υπολοίπου (πυκνό σε αραιό):

$$r_H = I_h^H r_h \tag{2.27}$$

Επίλυση στο Ω_H:

$$A_H e_H = r_H \tag{2.28}$$

Παρεμβολή την διόρθωση (αραιό σε πυκνό) :

$$e_h = I_H^h e_H \tag{2.29}$$

Υπολογισμός νέας προσέγγισης:

$$v_h = v_h + e_h \tag{2.30}$$

Εάν συνδυάσουμε αυτό το σχήμα διόρθωσης ανάμεσα στα δύο πλέγματα με τη διαδικασία της εξομάλυνσης $Gauss - Seidel$, τότε θα προκύψει η τεχνική των δύο πλεγμάτων όπου και περιγράφεται με τον παρακάτω αλγόριθμο (η δύναμη του m μας δείχνει το χρονικό βήμα της μεθόδου) :

Κύκλος δύο πλεγμάτων $v_h^{m+1} = TGCYC(v_h^m, A_h, f_h, \nu_1, \nu_2)$

(1) Προ-εξομάλυνση

-Υπολογισμός του $\bar{v}_h^m$ εφαρμόζοντας ν_1 φορές την διαδικασία εξομάλυνσης στο v_h^m

$$\bar{v}_h^m = SMOOTH^{\nu_1}(v_h^m, A_h, f_h) \tag{2.31}$$

(2) Διαδικασία διόρθωσης σε αραιό πλέγμα

-Υπολογισμός του διανύσματος υπολοίπου: $\bar{r}_h^m = f_h - A_h \bar{v}_h^m$

-Παρεκβολή του διανύσματος υπολοίπου (πυκνό σε αραιό): $\bar{r}_H^m = I_h^H \bar{r}_h^m$

-Επίλυση στο Ω_H: $A_H e_H^m = \bar{r}_H^m$

-Παρεμβολή την διόρθωση (αραιό σε πυκνό) : $e_h^m = I_H^h e_H^m$

-Υπολογισμός νέας προσέγγισης: $v_h^m = \bar{v}_h^m + e_h^m$

(3) Μετα-εξομάλυνση

-Υπολογισμός του u_h^{m+1} εφαρμόζοντας ν_2 φορές την διαδικασία εξομάλυνσης στο ανανεωμένο πλέον u_h^m

$$u_h^{m+1} = SMOOTH^{\nu_2}(u_h^m, A_h, f_h) \tag{2.32}$$

2.5 Τελεστές μεταφοράς πλέγματος

2.5.1 Παρεκβολή

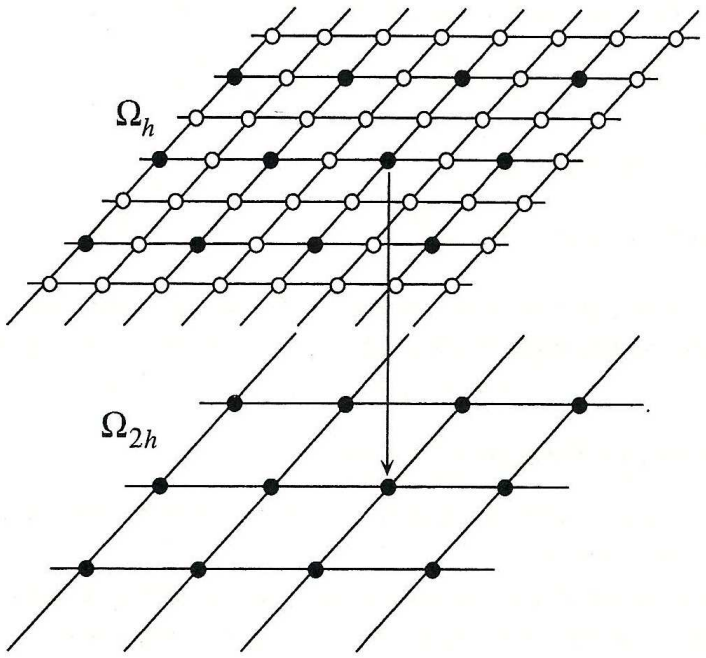
Παρεκβολή στις τεχνικές πολυπλέγματος ονομάζεται η διαδικασία κατά την οποία μεταφέρεται το διάνυσμα σφάλματος r_h στο r_H. Η επιλογή του τελεστή παρεμβολής ή παρεκβολής είναι στενά συνδεδεμένη με την επιλογή του αραιότερου πλέγματος. Σε αυτή την ενότητα επιλέγεται το αραίοτερο πλέγμα να διαθέτει βήμα διακριτοποίησης $H = 2h$ όπου $h = 1/n$. Ο τελεστής ορίζεται : $I_h^H : \mathcal{G}(\Omega_h) \rightarrow \mathcal{G}(\Omega_H)$

Ο απλούστερος τελεστής παρεκβολής για δύο πλέγματα Ω_h και Ω_H είναι ο γνωστός τελεστής $injection$. Εάν σε δύο πλέγματα (πυκνό και αραιό) κάποιοι κόμβοι ταυτίζονται, τότε ο τελεστής $injection$ μεταφέρει απλά την πληροφορία απο το στοιχείο του πυκνού πλέγματος στο αντίστοιχο του αραιού:

$$r_H(P) = I_h^H r_h(P) = r_h(P) \;\; \text{όπου} \;\; P \in \Omega_H \subset \Omega_h \tag{2.33}$$

Στην πράξη αυτός ο τελεστής χρησιμοποιείται μόνο σε πλέγματα στα οποία υπάρχει άμεση ταύτιση κόμβων. Εάν η διακριτοποίηση του χωρίου έχει πραγματοποιηθεί για την εφαρμογή κάποιας μεθόδου πεπερασμένων στοιχείων , τότε το πιο πιθανό είναι ότι οι κόμβοι ανάμεσα στα πλέγματα Ω_h και Ω_H να μην ταυτίζονται άμεσα. Το γεγονός αυτό οδηγεί στον ορισμό ενός διαφορετικού τελεστή παρεκβολής, τον $full\ weighting\ (FW)$. Ο FW είναι ένας απο τους τελεστές, που χρησιμοποιείται συχνά στις τεχνικές πολυπλέγματος. Μπορεί να γραφεί σε κομβική μορφή ως:

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



Σχήμα 2.6: Ο τελεστής παρεκβολής $injection$.

Εφαρμόζοντας αυτόν τον τελεστή σε ένα σημείο $d_h(x,y)$ σε ένα ζεύγος $(x,y) \in \Omega_H$ προκύπτει:

$$d_H(x,y) = I_h^H d_h(x,y) = \tfrac{1}{16}[4d_h(x,y) + 2d_h(x+h,y) + 2d_h(x-h,y) + 2d_h(x,y+h)$$
$$+2d_h(x,y-h) + d_h(x+h,y+h) + d_h(x+h,y-h) + d_h(x-y,y+h) + d_h(x-h,y-h)]$$

Αξίζει να αναφέρουμε ότι ο τελεστής FW, είναι το αποτέλεσμα ενός εξωτερικού γινομένου των μονοδιάστατων FW τελεστών:

$$I_h^H = \tfrac{1}{4}\,[1\ 2\ 1] \qquad I_h^H = \tfrac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

Ένας τρίτος δημοφιλής τελεστής παρεκβολής είναι ο $half\ weighting\ (HW)$:

$$I_h^H = \tfrac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}_h^H$$

ο οποίος αποτελεί μία απλούστερη έκδοση του FW, αλλά προσφέρει λιγότερη ακρίβεια προσέγγισης.

2.5.2 Παρεμβολή

Οι τελεστές παρεμβολής χρησιμοποιούνται για την μεταφορά των συναρτήσεων του πλέγματος Ω_H στις αντίστοιχες συναρτήσεις του Ω_h. Ένας πολύ γνωστός τελεστής παρεμβολής είναι ο $bilinear\ interpolation$, ο οποίος ορίζεται:

$$I_H^h : \mathcal{G}(\Omega_H) \to \mathcal{G}(\Omega_h) \tag{2.34}$$

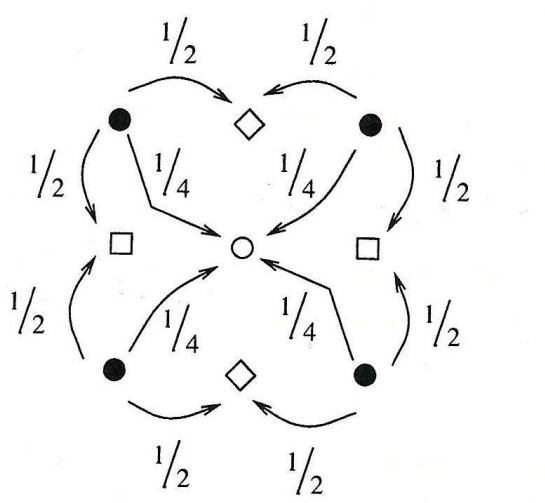
και μπορεί να γραφεί σε κομβική μορφή ως:

$$I_H^h = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_H^h$$

Σε αυτή την περίπτωση οι αγκύλες έχουν αντιστραφεί για να γίνει ευκολότερος ο διαχωρισμός της παρεκβολής με την παρεμβολή.

Εφαρμόζοντας αυτόν τον τελεστή σε ένα σημείο $u_H(x, y)$ για $(x, y) \in \Omega_H$ προκύπτει το αριθμητικό σχήμα:

$$u_h(x,y) = I_H^h u_H(x,y) = \begin{cases} u_H(x,y) \\ +\frac{1}{2}[u_H(x,y+h) + u_H(x,y-h)] \\ +\frac{1}{2}[u_H(x+h,y) + u_H(x-h,y)] \\ +\frac{1}{4}[u_H(x+h,y+h) + u_H(x+h,y-h) \\ +u_H(x-h,y+h) + u_H(x-h,y-h)] \end{cases}$$



Σχήμα 2.7: Ο τελεστής παρεμβολής *bilinear interpolation* σχηματικά.

Θα πρέπει να σημειωθεί ότι ο τελεστής *bilinear interpolation* είναι αποτέλσμα ενός εξωτερικού γινομένου των μονοδιάστατων τελεστών:

$$I_H^h = \tfrac{1}{2}\,]1\ 2\ 1[\qquad I_H^h = \tfrac{1}{2} \left] \begin{matrix} 1 \\ 2 \\ 1 \end{matrix} \right[$$

2.6 Τεχνική πολυπλέγματος

Στην προηγούμενη ενότητα παρουσιάστηκε μια τεχνική δύο πλεγμάτων και οι αντίστοιχοι τελεστές μεταφοράς των συναρτήσεων ανάμεσα στα δύο πλέγματα. Έτσι μπορούμε να επεκτείνουμε αυτήν την ιδεά και να μεταβούμε στη τεχνική πολυπλέγματος. Η μετάβαση από το πλέγμα δύο διαστάσεων στο πολυπλέγμα προέκυψε απο την παρατήρηση, ¨ ότι σε μια τεχνική δύο πλεγμάτων που συγκλίνει ομαλά είναι άσκοπο να λυθεί ακριβώς η εξίσωση υπολοίπου στο αραιό πλέγμα¨. Αντί για αυτό , και χωρίς καμία επιρροή στην ταχύτητα σύγκλισης, το σφάλμα e_H θα μπορούσε να αντικατασταθεί με μια προσέγγιση. Ένας φυσικός τρόπος να βρούμε μια νέα προσέγγιση είναι να εφαρμόσουμε ξανά την τεχνική των δύο πλεγμάτων στο Ω_H και σε ένα πιο αραιό απο αυτό.
Αυτό είναι εφικτό, έαν προφανώς η εξίσωση υπολοίπου στο αραιό πλέγμα είναι της ίδιας μορφής με αυτή της αρχικής. Εάν η τάξη σύγκλισης της τεχνικής δύο πλεγμάτων είναι αρκετά μικρή, τότε είναι αποδοτικό να επαναλάβουμε γ φορές την τεχνική δύο πλεγμάτων. Η ιδέα αυτή μπορεί να εφαρμοστεί συνολικά χρησιμοποιώντας αραιότερα και στην συνέχεια πιο αραιά πλέγματα μέχρι να καταλήξουμε στο πιο αραιό.
Στο πιο αραιό πλέγμα μπορούμε να λύσουμε το γραμμικό σύστημα με οποιαδήποτε άμεση μέθοδο ή επαναληπτική διότι συνήθως το αραιότερο πλέγμα εμπλέκει πολύ μικρό αριθμό κόμβων και κατά συνέπεια, η τάξη του γραμμικού συστήματος είναι μικρή.

Ας θεωρήσουμε την ακολουθία πλεγμάτων απο το πιο πυκνό στο πιο αραιό:

$$\Omega_{h_l}, \Omega_{h_{l-1}}, \ldots, \Omega_{h_2}, \Omega_{h_1}, \text{ όπου } h_l = 2h_{l-1}$$

τότε ο αλγόριθμος του πολυπλέγματος θα είναι ως εξής:

Κύκλος πολυπλέγματος $v_k^{m+1} = MGCYC(k, \gamma, v_k^m, A_k, f_k, \nu_1, \nu_2)$

(1) Προ-εξομάλυνση

-Υπολογισμός του $\bar{v}_k^m$ εφαρμόζοντας ν_1 φορές την διαδικασία εξομάλυνσης στο v_k^m

$$\bar{v}_k^m = SMOOTH^{\nu_1}(v_k^m, A_k, f_k) \tag{2.35}$$

(2) Διαδικασία διόρθωσης σε αραιό πλέγμα

-Υπολογισμός του διανύσματος υπολοίπου: $\bar{r}_k^m = f_k - A_k \bar{v}_k^m$

-Παρεκβολή του διανύσματος υπολοίπου (πυκνό σε αραιό): $\bar{r}_{k-1}^m = I_k^H \bar{r}_k^m$

-Επίλυση στο Ω_{k-1}: $A_{k-1} e_{k-1}^m = \bar{r}_{k-1}^m$,όπου

εάν $k = 1$, άμεση επίλυση του συστήματος

εάν $k > 1$, τότε κάνε γ φορές τον κ-κύκλο με αρχική προσέγγιση το 0

$$v_{k-1}^m = MGCYC^{\gamma}(k-1, \gamma, 0, A_{k-1}, \bar{r}_{k-1}^m, \nu_1, \nu_2)$$

-Παρεμβολή την διόρθωση (αραιό σε πυκνό) : $e_k^m = I_{k-1}^k e_{k-1}^m$

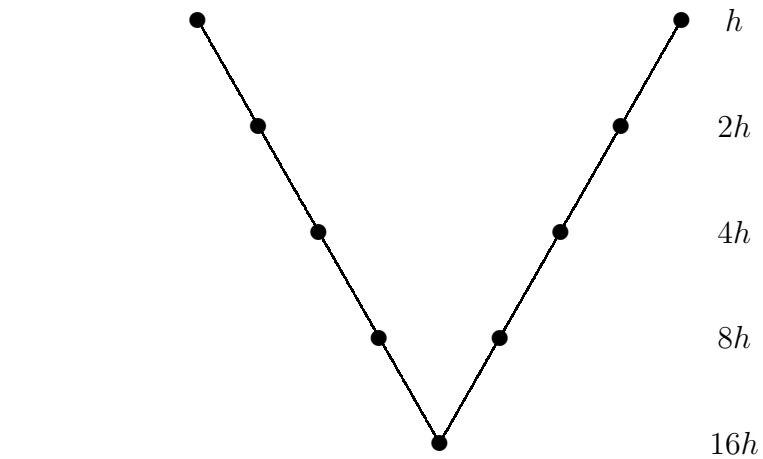
-Υπολογισμός νέας προσέγγισης: $v_k^m = \bar{v}_k^m + e_k^m$

(3) Μετα-εξομάλυνση

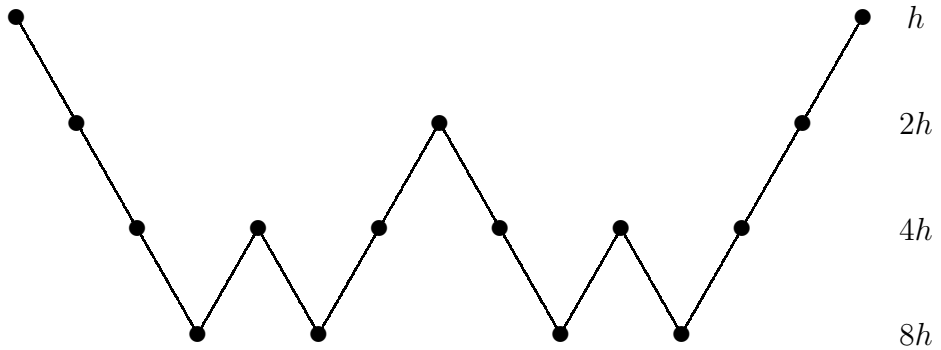
-Υπολογισμός του u_k^{m+1} εφαρμόζοντας ν_2 φορές την διαδικασία εξομάλυνσης στο ανανεωμένο πλέον u_k^m

$$u_k^{m+1} = SMOOTH^{\nu_2}(u_k^m, A_k, f_k) \tag{2.36}$$

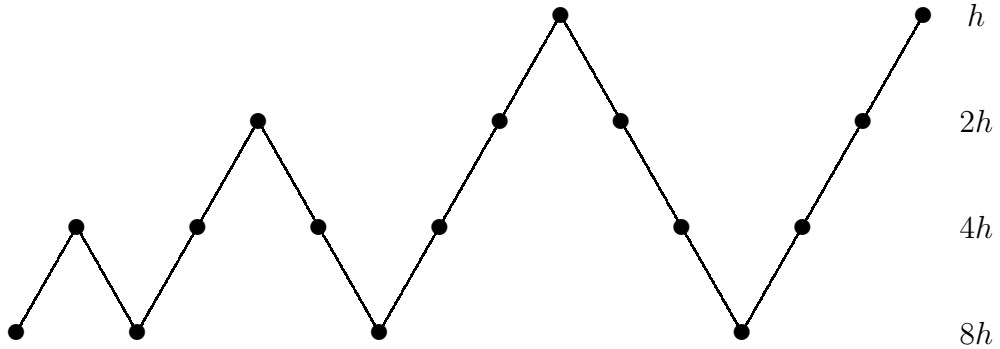
Κατά την επαναληπτική εφαρμογή της παραπάνω γενικής τεχνικής πολυπλέγματος είναι εφικτή η δημιουργία διάφορων σχημάτων συνολικής μετάβασης μεταξύ των πλεγμάτων. Τα πιο διαδεδομένα τέτοια σχήματα είναι το σχήμα V-κύκλος, W-κύκλος και FMG (πλήρες κύκλος πολυπλέγματος). Στο σχήμα (2.8 (α)) εμφανίζεται γραφικά η διαδικασία ενός V-κύκλου. Προκύπτει απο την διαρκή εφαρμογή μεταβάσεων σε πιο αραιό πλέγμα και στη συνέχεια στα πυκνότερα. Μια παραλλαγή του αποτελεί ο W-κύκλος (2.8 (β)) κατά το οποίο μεταφέρονται οι τιμές σε αραιότερα και πυκνότερα πλέγματα με κάποια συχνότητα. Τέλος, το σχήμα του FMG (2.8 (γ)) ξεκινά προσεγγίσεις οι οποίες βασίζονται σε ένα αραιό πλέγμα και με διαδοχικές μεταβάσεις σε πυκνότερα και αραιότερα πλέγματα δημιουργείται η προσέγγιση της λύσης στο πυκνότερο πλέγμα, συχνά χρησιμοποιόντας υψηλότερης τάξης παρεμβολή στο τελικό στάδιο.



(α)



(β)



(γ)

Σχήμα 2.8: Σχηματική περιγραφή των κύκλων (α) V , (β) W (γ) FMG. Στη βάση των σχημάτων βρίσκεται το πιο αραιό πλέγμα, στην κορυφή το αρχικό (πυκνό) και ενδιάμεσα τα διαδοχικά αραιότερα πλέγματα.

Κεφάλαιο 3

Αριθμητική επίλυση ΠΣΤ με τη μέθοδο πεπερασμένων στοιχείων Hermite Collocation

3.1 Εισαγωγή

Για την αριθμητική επίλυση Προβλημάτων Συνοριακών Τιμών (ΠΣΤ) μπορεί να χρησιμοποιηθούν τεχνικές αριθμητικών σχημάτων που ανήκουν στις κατηγορίες των πεπερασμένων Στοιχείων, Όγκων, Διαφορών ή Φασματικών μεθόδων [3, 13, 18, 19, 27]. Ένα ΠΣΤ μπορεί να περιγραφεί μέσω των παρακάτω σχέσεων :

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

όπου ο τελεστής $\mathcal{L}$ ισχύει εντός του χωρίου Ω, ενώ ο τελεστής $\mathcal{B}$ περιγράφει τις συνοριακές συνθήκες του προβλήματος και εφαρμόζεται πάνω στο σύνορο $\partial\Omega$. Απο τις πιο επιτυχείς επιλογές για την επίλυση ΠΣΤ αποτελεί η εφαρμογή μιας μεθόδου Πεπερασμένων

Στοιχείων η οποία μπορεί να περιγραφεί σχηματικά :

- **Βήμα 0:** Γεωμετρική Διαμέριση του πεδίου Ω σε πεπερασμένο πλήθος στοιχείων.

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

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

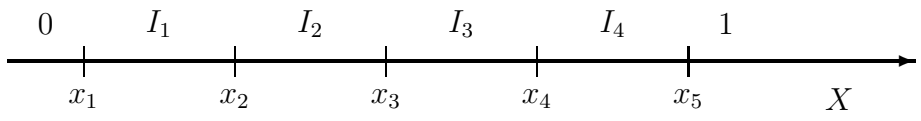
- **Βήμα 2:** Επιλογή μεθόδου διακριτοποίησης (Galerkin, Rayleigh-Ritz, Least Squares, Collocation) [3, 13] για μετάβαση από το **συνεχή** χώρο στο **διακριτό** δηλαδή μετατροπή του ΠΣΤ σε ένα πρόβλημα επίλυσης του γραμμικού συστήματος

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

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

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

Διαφορετικές επιλογές σε κάθε ένα από τα παραπάνω βήματα συνεπάγονται και διαφορετικές μεθόδους για την επίλυση ΠΣΤ.



Σχήμα 3.1: Γεωμετρική απεικόνιση της διαμέρισης του πεδίου Ω.

3.2 Αριθμητική μονοδιάστατη μέθοδος Collocation για πεπερασμένα στοιχεία τύπου Hermite

Σε μία διάσταση οι διαφορικοί τελεστές $\mathcal{L}$ και $\mathcal{B}$ ενός ΠΣΤ ορίζονται ως

$$\begin{cases} \mathcal{L} \equiv a(x)u''(x) + b(x)u'(x) + c(x)u(x) \quad , \quad x \in \Omega \equiv [a,b] \\ \mathcal{B}u(a) \equiv \gamma_0 u(a) + \gamma_1 u'(a) \ , \ \mathcal{B}u(b) \equiv \delta_0 u(b) + \delta_1 u'(b) \end{cases} . \tag{3.4}$$

Για την αριθμητική επίλυση του με τη μέθοδο πεπερασμένων στοιχείων Collocation ακολουθούμε τα παρακάτω βήματα

- **Βήμα 0:** Θεωρούμε ομοιόμορφη διαμέριση του διαστήματος $\Omega = [0,1]$ σε n_s υποδιαστήματα (πεπερασμένα στοιχεία) $I_m \ , \ m = 1 \, , \, \ldots \, , \, n_s$ με βήμα διακριτοποίησης $h = \frac{1}{n_s}$ και συντεταγμένες κόμβων $(x_i$, όπου $x_i = (i-1)h$ με $i = 1, \ldots, (n_s+1)$. Το Σχήμα 3.1 εμφανίζει την διαμέριση του Ω για την περίπτωση όπου $n_s = 4$.

- **Βήμα 1:** Ως συναρτήσεις βάσης επιλέγονται τα τμηματικά κυβικά πολυώνυμα Hermite [30], με γενική μορφή $\Phi_k(x)$, και η συνάρτηση $u(x)$ προσεγγίζεται από την

$$u(x) \simeq u_n(x) = \sum_{i=1}^{n} a_i \Phi_i(x) \quad , \tag{3.5}$$

όπου $n = 2(n_s + 1)$.

- **Βήμα 2:** Ως μέθοδος διακριτοποίησης επιλέγεται η μέθοδος της **Collocation** [5, 18, 19], η οποία κατασκευάζει το γραμμικό σύστημα $C\vec{\mathbf{a}} = \vec{\mathbf{b}}$ απαιτώντας οι συνθήκες $\mathcal{L}u_n - f = 0$ και $\mathcal{B}u_n - g = 0$ να ισχύουν για n καθορισμένα εσωτερικά και συνοριακά collocation σημεία και ως τέτοια επιλέγονται τα σημεία Gauss.

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

Στις επόμενες παραγράφους παρουσιάζονται αναλυτικά οι παραπάνω επιλογές της μεθόδου.

3.2.1 Πολυώνυμα Hermite ως συναρτήσεις βάσης

Τα τμηματικά κυβικά πολυώνυμα Hermite ορίζονται ως εξής [4, 24] :

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

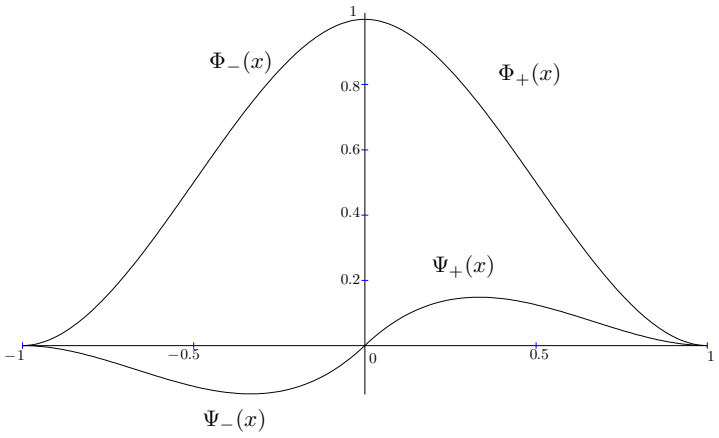
$$\Psi(x) \doteq \begin{cases} \Psi_+(x) & , \quad x \in [0,1] \\ \Psi_-(x) & , \quad x \in [-1,0] \\ 0 & , \quad x \notin [-1,1] \end{cases} , \tag{3.7}$$

όπου

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

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

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



Σχήμα 3.2: Κυβικά πολυώνυμα Hermite.

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

Το Σχήμα 3.2 εμφανίζει τα κυβικά πολυώνυμα Hermite, όπως αυτά ορίζονται στο $[-1,1]$.

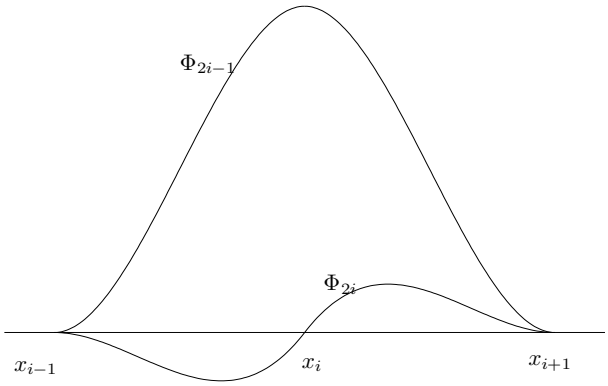
Σε κάθε κόμβο x_m αντιστοιχούν δύο συναρτήσεις και ορίζονται ως εξής:

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

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

όπου $m = 1, \ldots, (n_s+1)$, $I_i \equiv [x_i, x_{i+1}]$, $i = 1, \ldots, n_s$.

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



Σχήμα 3.3: Πολυώνυμα Hermite ορισμένα στον κόμβο x_i.

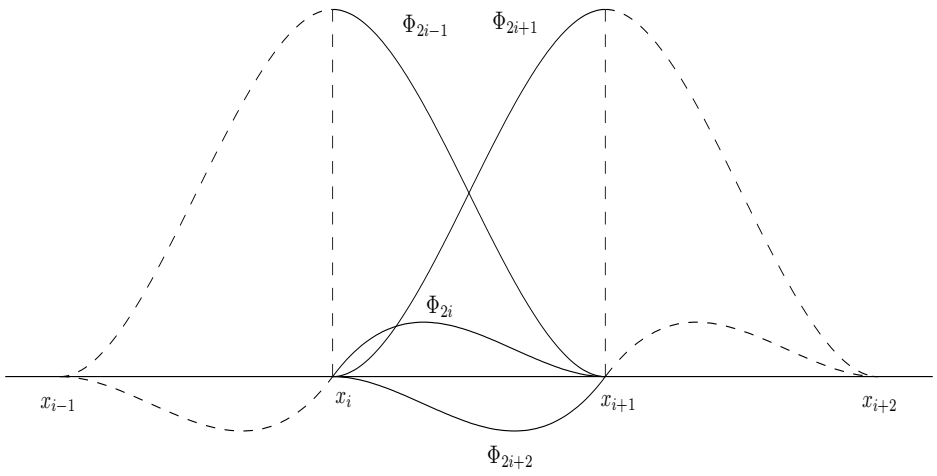
Στο Σχήμα 3.3 εμφανίζεται ένας τυχαίος κόμβος x_i και παρουσιάζονται οι αντίστοιχες συναρτήσεις, όπως αυτές ορίζονται σε αυτό το κόμβο και θα ισχύουν οι εξής ιδιότητες

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

για όλα τα $m \;=\; 1,\ldots,(n_s+1)$ και όπου δ^i_m : Δέλτα του Kronecker.

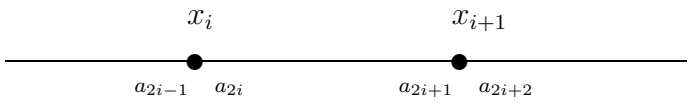
Σε κάθε υποδιάστημα I_i διέρχονται τέσσερα μόνο μη μηδενικά πολυώνυμα Hermite με δείκτες : Φ_{2i-1} , Φ_{2i} , Φ_{2i+1} και Φ_{2i+2} όπου εμφανίζονται στο Σχήμα 3.4. Σαν άμεση συνέπεια των προηγουμένων σχέσεων προκύπτει ότι

$$\begin{cases} u_n(x_i) &= a_{2i-1} \quad , \quad h\frac{d}{dx}u_n(x_i) &= a_{2i} \\ u_n(x_{i+1}) &= a_{2i+1} \quad , \quad h\frac{d}{dx}u_n(x_{i+1}) &= a_{2i+2} \end{cases} , \tag{3.15}$$



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

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



Σε κάθε πεπερασμένο στοιχείο I_i αντιστοιχούν 4 μη μηδενικές συναρτήσεις βάσης και έτσι για $x \in I_i$ ισχύει ότι :

$$u_n(x) = \sum_{k=2i-1}^{2i+2} \alpha_k \Phi_k(x) \ , \tag{3.16}$$

οπότε κάθε στοιχείο I_i από τα n_s είναι στοιχείο με 4 βαθμούς ελευθερίας και άρα οι άγνωστοι οι οποίοι πρέπει να υπολογιστούν έχουν πλήθος $4n_s$. Παρατηρούμε ότι σε

διαδοχικά στοιχεία I_i , I_{i+1} οι αγνώστοι που αντιστοιχούν στο δεύτερο κόμβο του I_i, ταυτίζονται με τους αγνώστους που αντιστοιχούν στον πρώτο κόμβο του I_{i+1}, με αποτέλεσμα οι άγνωστοι να είναι συνολικά $n = 2(n_s + 1)$.

Οι εξισώσεις θα κατασκευάστουν απαιτώντας το υπόλοιπο $\mathcal{L}u_n - f$ να μηδενίζεται σε $n_I = 2n_s$ **εσωτερικά** collocation σημεία και το υπόλοιπο $\mathcal{B}u_n - g$ να μηδενίζεται σε $n_b = 2$ **συνοριακά** collocation σημεία. Το πλήθος $n_I + n_b$ των collocation εξισώσεων ισούται με τον αριθμό των αγνώστων, ο οποίος προκύπτει από τη χρήση των κυβικών πολυωνύμων Hermite [4]. Κάνοντας χρήση των σχέσεων που προκύπτουν στα άκρα από τις συνοριακές συνθήκες καθώς και τη σχέση (3.15) των ιδιοτήτων των πολυωνύμων βάσης, είναι εφικτός ο άμεσος υπολογισμός του πρώτου ή δεύτερου μαζί με του πρωτελευταίου ή του τελευταίου αγνώστου.

Οπότε μετά την εφαρμογή των συνοριακών συνθηκών το πλήθος των αγνώστων και συνεπώς των collocation σημείων ισούται με $n = 2n_s$, που είναι το πλήθος των εσωτερικών σημείων.

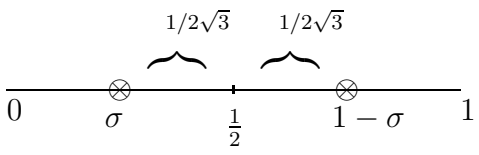
3.2.2 Σημεία Collocation - Βασικοί πίνακες

Τα σημεία Gauss [5] στο διάστημα $[-1, 1]$ είναι η απεικόνιση των ρίζων του Legendre πολυωνύμου δευτέρου βαθμού $\frac{1}{2}(3x^2-1) = 0$, δηλαδή τα σημεία $\mp\frac{\sqrt{3}}{3}$. Ο μετασχηματισμός των σημείων Gauss στο πεπερασμένο στοιχείο $I_i = [x_i, x_{i+1}]$ οδηγεί στις σχέσεις

$$\sigma_{2i} \;=\; \tfrac{x_i+x_{i+1}}{2} - \tfrac{1}{\sqrt{3}}\tfrac{h_i}{2} \quad , \quad \sigma_{2i+1} \;=\; \tfrac{x_i+x_{i+1}}{2} + \tfrac{1}{\sqrt{3}}\tfrac{h_i}{2} \;. \tag{3.17}$$

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

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



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

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

$$\begin{cases} D^k\Phi_{2i-1}(\sigma_{2i}) = \frac{1}{h^k}D^k\Phi(\sigma) \quad , \quad D^k\Phi_{2i-1}(\sigma_{2i+1}) = \frac{1}{h^k}D^k\Phi(1-\sigma) \\ D^k\Phi_{2i}(\sigma_{2i}) = \frac{1}{h^{k-1}}D^k\Psi(\sigma) \quad , \quad D^k\Psi_{2i-1}(\sigma_{2i+1}) = \frac{1}{h^{k-1}}D^k\Psi(1-\sigma) \\ D^k\Phi(1-\sigma) = D^k(1-\Phi(\sigma)) = (-1)^k D^k\Phi(\sigma) \end{cases} \tag{3.18}$$

για κάθε $i = 1, \ldots, n_s$. Αν επιλέξουμε τα σημεία Gauss ως εσωτερικά collocation σημεία, ορίζονται οι παρακάτω πίνακες, για $l = 0, 1, 2$

$$k_i^l = D^l\Phi_k(\sigma_j)_{k=2i-1,j=2i}^{2i+2\quad 2i+1} \tag{3.19}$$

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

$$k^0 = \begin{bmatrix} a & hb & 1-a & -h\bar{b} \\ 1-a & h\bar{b} & a & -hb \end{bmatrix} \quad , \tag{3.20}$$

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

$$k^1 = \frac{1}{h}\begin{bmatrix} -1 & h\hat{b} & 1 & -h\hat{b} \\ -1 & -h\hat{b} & 1 & h\hat{b} \end{bmatrix} \quad , \tag{3.21}$$

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

$$k^2 = \frac{1}{h^2}\begin{bmatrix} -a' & -hb' & a' & -h\bar{b}' \\ a' & h\bar{b}' & -a' & hb \end{bmatrix} \quad , \tag{3.22}$$

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

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

$$K^l = \begin{bmatrix} \kappa_2^l & \kappa_3^l & \kappa_4^l & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & \kappa_1^l & \kappa_2^l & \kappa_3^l & \kappa_4^l & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & \kappa_1^l & \kappa_2^l & \kappa_3^l & \kappa_4^l & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \kappa_1^l & \kappa_2^l & \kappa_3^l \end{bmatrix} . \tag{3.23}$$

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

3.3 Μέθοδος πεπερασμένων στοιχείων Hermite Collocation σε δύο διαστάσεις

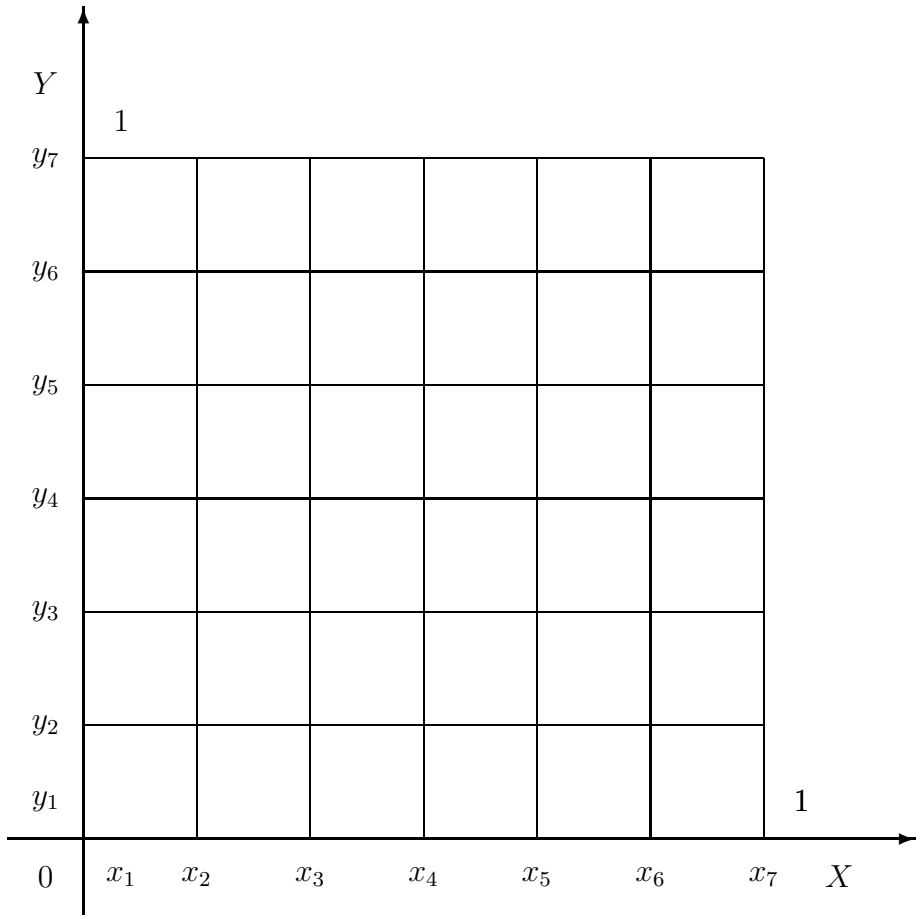
Για την περίπτωση όπου το ΠΣΤ ορίζεται σε ένα χωρίο Ω που είναι μια ορθογώνια περιοχή $\Omega \equiv [a, b] \times [c, d]$, οι τελεστές $\mathcal{L}$ και $\mathcal{B}$ του ΠΣΤ (3.1) θα είναι γενικά οι :

$$\begin{cases} \mathcal{L} \equiv a(x,y)\frac{\partial^2}{\partial x^2} + 2b(x,y)\frac{\partial^2}{\partial x \partial y} + c(x,y)\frac{\partial^2}{\partial y^2} + d(x,y)\frac{\partial}{\partial x} + e(x,y)\frac{\partial}{\partial y} + h(x,y) \\ \mathcal{B} \equiv \alpha(x,y) + \beta(x,y)\frac{\partial}{\partial \vec{n}} \end{cases} \tag{3.24}$$

Η συνθήκη $a(x,y)c(x,y) > b^2(x,y)$ χαρακτηρίζει τον **ελλειπτικό** τύπο του προβλήματος και συνεπάγεται ότι οι συναρτήσεις $a(x,y)$ και $c(x,y)$ είναι ομόσημες και μη μηδενικές.

Για την αριθμητική επίλυση του ΠΣΤ με τη μέθοδο πεπερασμένων στοιχείων Collocation ακολουθούμε τα παρακάτω βήματα θεωρώντας ότι το πεδίο ορισμού του ΠΣΤ έχει μετασχηματιστεί στο μοναδιαίο τετράγωνο.

- **Βήμα 0:** Θεωρούμε ομοιόμορφο διαμερισμό των διαστημάτων $I^x = I^y = [0, 1]$ σε n_s υποδιαστήματα $I_m^x = I_m^y$, $m = 1$, $\ldots$, n_s τα οποία παράγουν ένα ομοιόμορφο



Σχήμα 3.5: Γεωμετρική απεικόνιση της διαμέρισης του πεδίου Ω

πλέγμα με βήμα διακριτοποίησης $h = \frac{1}{n_s}$ και συντεταγμένες κόμβων (x_i, y_j), όπου $x_i = (i-1)h$ και $y_j = (j-1)h$, με $i, j = 1\,,\,\ldots\,,\,(n_s+1)$. Το Σχήμα 3.5 εμφανίζει την διαμέριση του Ω για $n_s = 6$.

- **Βήμα 1:** Ως συναρτήσεις βάσης επιλέγονται τα Hermite Bicubic πολυώνυμα, με γενική μορφή $\Phi_k(x,y) = \Phi_i(x)\Phi_j(y)$, και η συνάρτηση $u(x,y)$ προσεγγίζεται από την

$$u(x,y) \;\simeq\; u_n(x,y) \;=\; \sum_{i=1}^{\tilde{n}} \sum_{j=1}^{\tilde{n}} a_{i,j} \Phi_i(x) \Phi_j(y) \;\;, \tag{3.25}$$

όπου $\tilde{n} = 2(n_s+1)$.

- **Βήμα 2:** Ως μέθοδος διακριτοποίησης επιλέγεται η μέθοδος της **Collocation**,

η οποία κατασκευάζει το γραμμικό σύστημα $C\tilde{\mathbf{a}} = \tilde{\mathbf{b}}$ απαιτώντας οι συνθήκες $\mathcal{L}u_n - f = 0$ και $\mathcal{B}u_n - g = 0$ να ισχύουν για n καθορισμένα εσωτερικά και συνοριακά collocation σημεία.

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

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

$$\begin{cases} \Phi_{2i-1,2j-1}(x,y) &= \Phi_{2i-1}(x)\Phi_{2j-1}(y) \\ \Phi_{2i-1,2j}(x,y) &= \Phi_{2i-1}(x)\Phi_{2j}(y) \\ \Phi_{2i,2j-1}(x,y) &= \Phi_{2i}(x)\Phi_{2j-1}(y) \\ \Phi_{2i,2j}(x,y) &= \Phi_{2i}(x)\Phi_{2j}(y) \end{cases} \tag{3.26}$$

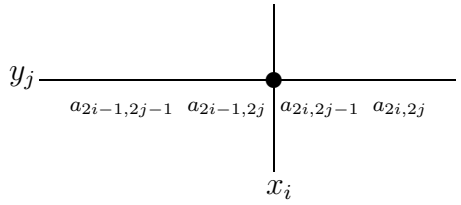
με τις εξής ιδιότητες :

$$\begin{cases} \Phi_{2i-1,2j-1}(x_i,y_j) = 1 \quad , \quad h\frac{\partial}{\partial y}\Phi_{2i-1,2j}(x_i,y_j) = 1 \\ h\frac{\partial}{\partial x}\Phi_{2i,2j-1}(x_i,y_j) = 1 \quad , \quad h^2\frac{\partial^2}{\partial x \partial y}\Phi_{2i,2j}(x_i,y_j) = 1 \end{cases} . \tag{3.27}$$

Σαν άμεση συνέπεια των προηγουμένων σχέσεων προκύπτει ότι

$$\begin{cases} u_n(x_i,y_j) = a_{2i-1,2j-1} \quad , \quad h\frac{\partial}{\partial y}u_n(x_i,y_j) = a_{2i-1,2j} \\ h\frac{\partial}{\partial x}u_n(x_i,y_j) = a_{2i,2j-1} \quad , \quad h^2\frac{\partial^2}{\partial x \partial y}u_n(x_i,y_j) = a_{2i,2j} \end{cases} , \tag{3.28}$$

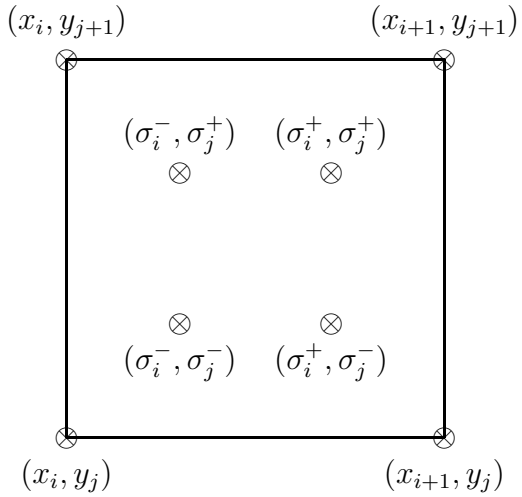
όπου $i\,,\, j = 1, \ldots, (n_s+1)$ και στο παρακάτω σχήμα εμφανίζονται οι τέσσερις άγνωστοι όπως αυτοί αντιστοιχούν στον κόμβο (x_i, y_i).



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

$$u_n(x, y) \ = \ \sum_{k=2i-1}^{2i+2} \sum_{l=2j-1}^{2j+2} \alpha_{k,l} \Phi_k(x) \Phi_l(y) \ . \tag{3.29}$$

Γι'αυτό το λόγο κάθε πεπερασμένο στοιχείο I_{ij}^{xy} είναι στοιχείο με 16 βαθμούς ελευθερίας.

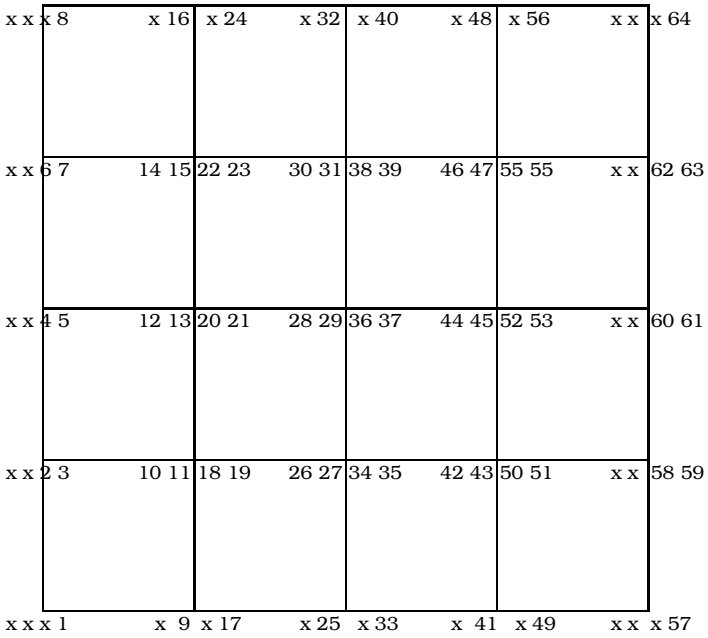


Σχήμα 3.6: Τα τέσσερα σημεία Gauss στο πεπερασμένο στοιχείο I_{ij}^{xy}.

Στην περίπτωση του διδιάστατου προβλήματος χρειαζόμαστε $n_I = 4n_s^2$ **εσωτερικά** collocation σημεία και $n_b = 4(2n_s + 1)$ **συνοριακά** collocation σημεία. Όμοια με το μονοδιάστατο πρόβλημα το πλήθος $n_I + n_b$ των collocation εξισώσεων θα ισούται με τον αριθμό των αγνώστων. Εφαρμόζοντας τις σχέσεις των συνοριακών συνθηκών με τη ταυτόχρονη χρήση τέταρτης τάξης σφάλματος προσεγγιστικών σχέσεων πεπερασμένων διαφορών των παραγώγων (κεντρικών, προς τα εμπρός και πίσω) είναι εφικτός ο άμεσος προσδιορισμός της προσέγγισης όλων των τιμών της λύσης του ΠΣΤ καθώς και της κατα κατεύθυνση τιμές των παραγώγων πάνω στα σημεία διακριτοποίησης του πλέγματος του συνόρου $\partial\Omega$. Από τις συνοριακές συνθήκες δηλαδή προσδιορίζονται άμεσα $4(2n_s + 1)$ το πλήθος άγνωστοι και απαλοίφονται απο το γραμμικό σύστημα. Έτσι έχουμε να υπολογίσουμε $n = 4n_s^2$ αγνώστους, όσο δηλαδή και το πλήθος των εσωτερικών collocation σημείων.

Στην περίπτωση ελλειπτικών ΠΣΤ η κλασσική επιλογή εσωτερικών collocation σημείων είναι αυτή των σημείων Gauss [5] με συντεταγμένες $(\sigma_i^{\pm}, \sigma_j^{\pm})$, όπου για κάθε $i = 1, \ldots, n_s$ ισχύει ότι $\sigma_i^{\pm} = \frac{h}{2}(2i - 1 \pm \frac{\sqrt{3}}{3})$. Στο Σχήμα 3.6 εμφανίζονται τα τέσσερα σημεία Gauss του στοιχείου I_{ij}^{xy}.

Η αρίθμηση αγνώστων και εξισώσεων καθορίζει την δομή του Collocation πίνακα, και



Σχήμα 3.7: Block Τριδιαγώνια Αρίθμηση αγνώστων για $n_s = 4$.

κατά συνέπεια την επιλογή της μεθόδου επίλυσης του παραγόμενου γραμμικού συστήματος. Στην εργασία [30] ο Θ. Σ. Παπαθεοδώρου πρότεινε την αρίθμηση των αγνώστων, η οποία παρουσιάζεται στο Σχήμα 3.7, και των εξισώσεων, η οποία παρουσιάζεται στο Σχήμα 3.8. Εξαιτίας της αρίθμησης αυτής προκύπτει ο Collocation πίνακας σε block τριδιαγώνια δομή η οποία εμφανίζεται στο Σχήμα 3.9. Οι άγνωστοι, οι οποίοι έχουν απαλοιφθεί λόγω των συνοριακών συνθηκών σημειώνονται με "x" [25, 26, 29].

Κάνοντας χρήση της $block$ τριδιαγώνιας μεθόδου αρίθμησης παράγεται το ακόλουθο γραμμικό σύστημα

$$A\boldsymbol{x} \;=\; \boldsymbol{b} \;\;, \tag{3.30}$$

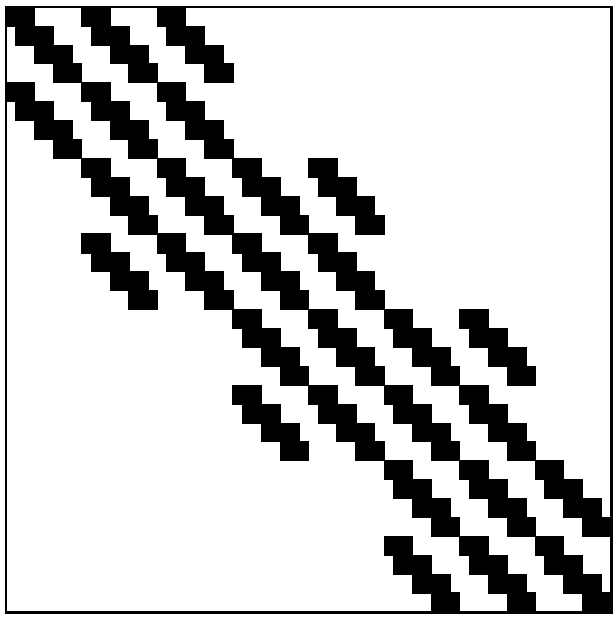
όπου $A \in \mathbb{R}^{n,n} \;\; (n = 4{n_s}^2)$ είναι ο Collocation πίνακας συντελεστών και

$\boldsymbol{x} \;=\; [x_1 \;\; x_2 \;\; \cdots \;\; x_n]^T \;=\; [\alpha_{1,1} \;\; \cdots \;\; \alpha_{2n_s,2n_s}]^T$ είναι το διάνυσμα των αγνώστων.

Ο πίνακας A με την βοήθεια των συνολικών βασικών πινάκων $K^l \equiv K^l_x \equiv K^l_y$ μπορεί να

8	16	24	32	40	48	56	64
7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57

Σχήμα 3.8: Block Τριδιαγώνια Αρίθμηση εξισώσεων για $n_s = 4$.



Σχήμα 3.9: Δομή του Block Τριδιαγώνιου Collocation Πίνακα.

κατασκευαστεί σύμφωνα με την παρακάτω σχέση για τη γενική μορφή του ΠΣΤ [19, 20, 21]

$$A = F_a(K_x^2 \otimes K_y^0) + F_b(K_x^1 \otimes K_y^1) + F_c(K_x^0 \otimes K_y^2) + F_d(K_x^1 \otimes K_y^0) + F_e(K_x^0 \otimes K_y^1) + F_h(K_x^0 \otimes K_y^0)$$

Οι πίνακες F_* προκύπτουν από το γινόμενο δυο διαγωνίων πινάκων. Ειδικότερα:

$$F_a = H^{(2)}G_a \ , \ F_b = 2H^{(1)}H^{(1)}G_b \ , \ F_c = H^{(2)}G_c,$$
$$F_d = H^{(1)}G_d \ , \ F_e = H^{(1)}G_e \text{ και } F_h = G_h \text{ όπου}$$

$$H^{(1)} = diag[\tfrac{1}{h}, \tfrac{1}{h}, \ldots, \tfrac{1}{h}] \in \Re^{4ns^2, 4ns^2}$$
$$H^{(2)} = diag[\tfrac{1}{h^2}, \tfrac{1}{h^2}, \ldots, \tfrac{1}{h^2}] \in \Re^{4ns^2, 4ns^2}$$

$$G_a = diag[G_1, G_2, \ldots, G_{2ns-1}, G_{2ns}] \in \Re^{4ns^2, 4ns^2} \text{ όπου}$$
$$G_{2k-1} = diag[a(\sigma_{2k-1}, \sigma_1), \ldots, a(\sigma_{2k-1}, \sigma_{2ns})] \text{ και}$$
$$G_{2k} = diag[a(\sigma_{2k}, \sigma_1), \ldots, a(\sigma_{2k}, \sigma_{2ns})] \text{ για } k = 1, \ldots, ns$$

Εάν θεωρήσουμε το γινόμενο $h^2 F_*(K_x^i \otimes K_y^j) = \Pi_*^{i,j}$ και πολλαπλασιάζοντας τα δύο μέλη του γραμμικού συστήματος με h^2 αυτό μπορεί να γραφεί στην μορφή:

$$(\Pi_a^{2,0} + \Pi_b^{1,1} + \Pi_c^{0,2} + \Pi_d^{1,0} + \Pi_e^{0,1} + \Pi_h^{0,0})x = h^2 \, \mathbf{f}$$

Όπου το διάνυσμα $\mathbf{f} = [f_1, f_2, \ldots, f_{2ns}]$ με :

$f_{2k-1} = [f_{2k-1}(\sigma_{2k-1}, \sigma_1), \ldots, f_{2k-1}(\sigma_{2k-1}, \sigma_2 ns)]$ και

$f_{2k} = [f_{2k}(\sigma_{2k}, \sigma_1), \ldots, f_{2k}(\sigma_{2k}, \sigma_{2ns})]$

Καθένας από τους παραπάνω πίνακες Π_* μπορεί να θεωρηθεί ως γινόμενο ενός πίνακα $\Lambda = diag(\Lambda_1, \Lambda_2, \ldots, \Lambda_{2ns})$ με:

$\Lambda_i = diag(h^2, h, h^2, h, \ldots, h^2, h^2)_{2n}$ για i περιττό και $\Lambda_i = diag(h, 1, h, 1, \ldots, h, h)_{2n}$ για i άρτιο με έναν αντίστοιχο πίνακα P_*, οπότε το γραμμικό σύστημα θα έχει την παρακάτω μορφή:

$$(\Pi_a^{2,0} + \Pi_b^{1,1} + \Pi_c^{0,2} + \Pi_d^{1,0} + \Pi_e^{0,1} + \Pi_h^{0,0})x = h^2 f \tag{3.31}$$

ή

$$(P_a^{2,0}\Lambda + P_b^{1,1}\Lambda + P_c^{0,2}\Lambda + P_d^{1,0}\Lambda + P_e^{0,1}\Lambda + P_h^{0,0}\Lambda)x = h^2 f \tag{3.32}$$

ή

$$(P_a^{2,0} + P_b^{1,1} + P_c^{0,2} + P_d^{1,0} + P_e^{0,1} + P_h^{0,0})\Lambda x = h^2 f \tag{3.33}$$

ή

$$(P_a^{2,0} + P_b^{1,1} + P_c^{0,2} + P_d^{1,0} + P_e^{0,1} + P_h^{0,0})y = h^2 f \tag{3.34}$$

ή

$$Cy = b \tag{3.35}$$

Η δομή του $Collocation$ πίνακα συντελεστών των αγνώστων είναι τύπου block. Όμως εξαιτίας των συντελεστών συναρτήσεων των γραμμικών όρων του ΠΣΤ δεν προκύπτει συσχέτιση ανάμεσα στα block των συντελεστών κι επομένως η επιλογή της αρτιότερης δομής αποθήκευσης του πίνακα θα πρέπει να γίνει ανάμεσα σε αυτές της αραιής αποθήκευσης.

Κεφάλαιο 4

Παράλληλος προγραμματισμός σε κάρτες γραφικών-GPUs

4.1 Εισαγωγή

Οι κάρτες γραφικών (GPU) είναι εξιδεικευμένα ηλεκτρονικά κυκλώματα σχεδιασμένα να διεξάγουν τους απαραίτητους υπολογισμούς για τη κατασκευή και επεξεργασία με μεγάλη ταχύτητα εικόνων σε μια οθόνη. Χρησιμοποιούνται σε κινητά τηλέφωνα, προσωπικούς ηλεκτρονικούς υπολογιστές, εξειδικευμένα συστήματα όπως παιχνιδομηχανές και συστήματα πλοήγησης. Οι σύγχρονες κάρτες γραφικών για να ικανοποιήσουν τις πολύ αυξημένες απαιτήσεις της τεχνολογίας, όπως για παράδειγμα τρισδιάστατες απεικονίσεις σε πραγματικό χρόνο, χρησιμοποιούν τη τεχνολογία massively parallel για τη πραγματοποίηση των επιστημονικών υπολογισμών. Έτσι μέσω κατάλληλων παράλληλων αλγορίθμων έχουν καταφέρει να είναι αποδοτικότερες από τους κεντρικούς επεξεργαστές των υπολογιστικών συστημάτων [2, 23, 32]. Αν και συνήθως οι κάρτες γραφικών εκτελούν υπολογισμούς για γραφικά, μπορούμε να εκμεταλευτούμε την αρχιτεκτονική τους και να διεξάγουμε υπολογισμούς εφαρμογών που συνήθως εκτελεί ο κεντρικός επεξεργαστής. Η παραπάνω χρήση μιας κάρτας γραφικών ονομάζεται προγραμματισμός

γενικής χρήσης σε μονάδες γραφικών (GPGPU). Η ικανότητα μιας κάρτας γραφικών να εκτελεί παράλληλους αλγορίθμους μπορεί να αυξηθεί κάνοντας χρήση πολλαπλών GPU ή μιας κάρτας γραφικών με μεγάλο πλήθος επεξεργαστών (cores). Τα εργαλεία ελεύθερης χρήσης που έχε στη διάθεση του ο προγραμματιστής για αυτή τη δουλειά είναι εξειδικευμένες γλώσσες για κάρτες γραφικών, δηλαδή επεκτάσεις της C με το πρότυπο (API) CUDA [39] ή της OpenCL [38].

4.2 Παράλληλος και επιστημονικός προγραμματισμός σε κάρτες γραφικών

4.2.1 Η αρχή του παράλληλου προγραμματισμού

Για 30 περίπου χρόνια, ο μόνος τρόπος για να βελτιωθεί η υπολογιστική ταχύτητα και η επίδοση ενός υπολογιστή ήταν η αύξηση της ταχύτητας εκτέλεσης εργασιών του κεντρικού επεξεργαστή. Οι πρώτοι προσωπικοί υπολογιστές ξεκίνησαν με ταχύτητες εκτέλεσης εργασιών 1MHz , ενώ σήμερα οι περισσότεροι προσωπικοί υπολογιστές έχουν ταχύτητα από 1GHz εώς 4GHz, σχεδόν 1,000 φορές γρηγορότεροι απο τον πρώτο προσωπικό υπολογιστή. Ωστόσο, η εξέλιξη των κεντρικών επεξεργαστών δεν ήταν ο μόνος λόγος που οι υπολογιστές έγιναν ταχύτεροι, ένας ακόμα λόγος ήταν η συνεχής βελτίωση στην προγραμματιστική σχεδίαση των λογισμικών που τα υποστηρίζουν.

Τα τελευταία χρόνια, οι κατασκευαστές κεντρικών επεξεργαστών ερευνούσαν εναλλακτικές μεθόδους βελτίωσης της επίδοσης του υπολογιστή, διότι η ενέργεια που χρειάζεται ένας επεξεργαστής αυξάνεται εκθετικά, σε σχέση με την υπολογιστική του ισχύ. Η αύξηση της υπολογιστικής ισχύς ενός επεξεργαστή, θα είχε αντίκτυπο στο μέγεθος του κυκλώματος αλλά και στην θερμοκρασία λειτουργίας του. Έτσι ερευνητές και κατασκευαστές

έπρεπε να βρούν κάτι καινούργιο , αποδοτικότερο όσο αφορά την αύξηση της υπολογιστικής ισχύος.

Τελικά, η λύση σε αυτό το πρόβλημα δόθηκε με τον παρακάτω τρόπο. Αντί να αυξηθεί η υπολογιστική ισχύς σε έναν επεξεργαστή, αυξήθηκε το πλήθος των επεξεργαστών σε έναν υπολογιστή. Σήμερα, τα πολυεπεξεργαστικά συστήματα αποτελούνται απο 2, 3, 4, 6, 8 ή 16 επεξεργαστές και πλέον υπάρχουν σε κάθε προσωπικό υπολογιστή , ακόμα και στα κινητά τηλέφωνα.

4.2.2 Από τους κεντρικούς επεξεργαστές στις κάρτες γραφικών

Ο γενικής χρήσης προγραμματισμός σε κάρτα γραφικών GPU είναι κάτι διαφορετικό σε σύγκριση με τον αντίστοιχο για τον κεντρικό επεξεργαστή (CPU) του υπολογιστικού συστήματος. Η ανάγκη για την δημιουργία ενός νέου είδους περιεφεριακής συσκευής με δικό της επεξεργαστή για την αποκλειστική κατασκευή γραφικών προκλήθηκε αρχικά από την αύξηση της δημοσιότητας του γραφικού λειτουργικού συστήματος Windows στις αρχές της δεκαετίας του 90. Οι πρώτες κάρτες γραφικών παρήγαγαν γραφικά δύο διαστάσεων (2D). Αργότερα, η εταιρεία Silicon Graphics καθιέρωσε ως πλατφόρμα ανάπτυξης τρισδιάστατων γραφικών (3D) το προγραμματιστικό περιβάλλον που βασίστηκε στην βιβλιοθήκη OpenGL.

Στις αρχές του 2000, σχεδιάστηκαν κάρτες γραφικών οι οποίες μπορούσαν να παράγουν ένα χρώμα για κάθε pixel οθόνης χρησιμοποιώντας μεταβλητές που λέγονταν pixel shaders, τα οποία εκμεταλευόταν την θέση (x, y) στην οθόνη και με κατάλληλα ορίσματα ορίσματα δημιουργούσαν το χρώμα. Μερικά απο αυτά τα ειδικά ορίσματα ήταν η επιλογή χρώματος, θέσης και σκίασης. Οι προγραμματιστές παρατήρησαν ότι στην θέση εισόδου του χρώματος θα μπορούσε να μπει οποιοδήποτε άλλο δεδομένο, έτσι η είσοδος και η έξοδος των δεδομένων μπορούσαν να είναι αριθμητικές μεταβλητές. Ωστόσο, υπήρχαν κάποιοι φυσικοί περιορισμοί στην εξέλιξη του προγραμματισμού σε κάρτες γραφικών,

διότι τα δεδομένα έπρεπε να δίνονται μόνο μέσα απο μεταβλητές χρώματος ή χαρακτήρων και υπήρχαν δυσκολίες στον τρόπο και το μέρος που θα αποθηκεύονταν τα αποτελέσματα στην μνήμη υπολογιστή. Επίσης, λόγω της αδυναμίας να προβλεφθεί εάν μια κάρτα γραφικών θα χειρίζοταν ικανοποιητικά αριθμούς κινητής υποδιαστολής, οι περισσότεροι επιστημονικοί υπολογισμοί θα ήταν αδύνατο να πραγματοποιηθούν στην κάρτα γραφικών. Εκτός από τους παραπάνω περιορισμούς που είχε ο προγραμματισμός στις κάρτα γραφικών, θα έπρεπε κάποιος που ήθελε να τη χρησιμοποιήσει για προγραμματισμό γενικών υπολογισμών να γνωρίζει τις γλώσσες OpenGL ή DirectX. Αυτό σημαίνει ότι τα δεδομένα θα αποθηκεύονταν σε μεταβλητές γραφικών και οι υπολογισμοί θα γινόταν από ειδικές συναρτήσεις για γραφικά. Λόγω αυτών των μειονεκτημάτων δημιουργήθηκε η ανάγκη για μια ειδική γλώσσα για κάρτες γραφικών που θα μπορεί να εκτελεί γενικούς υπολογισμούς υψηλής ακρίβειας. Το πρότυπο αυτό ονομάστηκε CUDA (Compute Unified Device Architecture) και υλοποιήθηκε από την κατασκευάστρια καρτών γραφικών Nvidia.

4.3 Υπολογιστικά εργαλεία για κάρτες γραφικών

4.3.1 Το πρότυπο CUDA

Οι σύγχρονες κάρτες γραφικών σε αντίθεση με τις προηγούμενες γενιές τους υποστηρίζουν την αρχιτεκτονική CUDA και επιτρέπουν την χρήση αριθμητικών ή λογικών μεταβλητών για την διεξαγωγή υπολογισμών παράλληλα με την ιδιότητα τους να κατασκευάζουν γραφικά. Υπάρχουν εξιδεικευμένες κάρτες γραφικών μεγαλύτερης υπολογιστικής ισχύος , χωρητικότητας μνήμης και είναι σε θέση να διεξάγουν υπολογισμούς πραγματικών αριθμών διπλής ακρίβειας. Οι κάρτες αυτές είναι σχεδιασμένες αποκλειστικά για παράλληλο προγραμματισμό και δεν υποστηρίζουν γραφικά. Οι κάρτες γραφικών έχουν σχεδιαστεί να είναι πολυεπεξεργαστικά συστήματα κοινής μνήμης, τις περισσότερες φορές ξεχωριστή απο αυτή του κεντρικού επεξεργαστή. Στην συνέχεια αναφέρονται ενδεικτικά δυο ερευνητικές περιοχές όπου υπάρχουν αξιόλογες εφαρμογές.

Εφαρμογές CUDA

Εφαρμογή στην Ιατρική

Ο αριθμός των ανθρώπων που προσβάλονται απο τον καρκίνο του μαστού έχει αυξηθεί δραματικά τα τελευταία 20 χρόνια. Κάθε περίπτωση καρκίνου του μαστού πρέπει να διαγιγνώσκεται έγκαιρα, ώστε να μπορέσει να θεραπευτεί με την χρήση φαρμάκων, χημειοθεραπείας ή χειρουργικής επέμβασης. Για τους παραπάνω λόγους, οι ερευνητές χρειάζονται γρήγορους και ακριβείς μεθόδους εντοπισμού της ασθένειας. Η μαστογραφία αποτελεί μια απο τις καλύτερες τεχνικές εντοπισμού του καρκίνου του μαστού. Ωστόσο, είναι μια ραδιενεργή και επιβλαβής για τον οργανισμό εξέταση γι΄ αυτό πολλές φορές συνδυάζεται με τον υπέρηχο που είναι ασφαλέστερος αλλά όχι απόλυτα ακριβής. Ως αποτέλεσμα όλων αυτών δημιουργήθηκε το TechiScan, μία μέθοδος τρισδιάστατου υπέρηχου. Το αρνητικό του TechiScan ήταν ότι η ανάλυση των δεδομένων ενός τρισδιάστατου προβλήματος απαιτούσε υψηλό υπολογιστικό κόστος αλλά και χρόνο αντίστοιχα. Η λύση αυτού του προβλήματος δόθηκε με τον προγραμματισμό σε CUDA C, δύο κάρτες γραφικών τύπου TESLA C1060 οι οποίες μπορούν και αναλύουν 35GB δεδομένα σε 15 μόνο λεπτά.

Εφαρμογή σε Υπολογιστική Ρευστοδυναμική

Για πολλά χρόνια, ήταν δύσκολο να μελετηθούν αριθμητικά συσκευές αεροδυναμικής ή της ρευστοδυναμικής, όπως για παράδειγμα οι ανεμογεννήτριες. Αυτό οφείλεται στο γεγονός ότι οι συσκευές αυτές δεν μπορούν να προσομοιωθούν αποτελεσματικά απο απλούς μαθηματικούς τύπους, και έτσι το υπολογιστικό κόστος είναι τεράστιο για ρεαλιστικές εφαρμογές. Μία τέτοιου τύπου έρευνα μπορεί να γίνει μόνο με τη χρήση υπερυπολογιστών, οι οποίοι είναι σε θέση να διαχειριστούν το υπολογιστικό φορτίο. Η υπολογιστική ομάδα "many-core group" του πανεπιστήμιου Cambridge προγραμματί-

ζοντας κάρτες γραφικών για υπολογιστική ρευστοδυναμική εξέλιξε την έρευνα της σε πρωτοφανή επίπεδα χρησιμοποιώντας προσωπικούς υπολογιστές με κάρτες γραφικών για προγραμματισμό γενικής χρήσης, αντί για δαπανηρούς υπερυπολογιστές.

4.3.2 Πρότυπο OpenCL

Το πρότυπο OpenCL είναι μία πρόσφατα αναπτυγμένη προγραμματιστική βιβλιοθήκη που εκτός απο τις κάρτες γραφικών, υποστηρίζει προγραμματισμό σε κεντρικούς επεξεργαστές και άλλου τύπου κάρτες. Το OpenCL αναπτύχθηκε απο την ομάδα Khronos, η οποία είχε αναπτύξει και το πρότυπο OpenGL για τα γραφικά σε συνεργασία με πολλές κατασκευάστριες εταιρείες καρτών. Η αποστολή και η λήψη των ορισμάτων μιας συνάρτησης για κάρτα γραφικών kernel διαχειρίζεται απο τον προγραμματιστή. Μια ακόμα σημαντική διαφορά μεταξύ αυτών των δύο προτύπων είναι ότι στο OpenCL οι συναρτήσεις της κάρτας δεν μεταγλωτίζονται μαζί με τον υπόλοιπο κώδικα, αλλά μεταγλωτίζονται την ώρα που η εφαρμογή εκτελείται απο την ίδια τη βιβλιοθήκη. Αυτό το είδος μεταγλώτισης επιτυγχάνεται με την αποστολή των kernel OpenCL ως μια ακολουθία συμβόλων. Στην πραγματικότητα αυτή η διαδικασία εμπεριέχει πολύ περισσότερο κώδικα σε σύγκριση με μια εφαρμογή ανεπτυγμένη με CUDA όπου για αυτές τις διαδικασίες ο προγραμματιστής δεν ασχολείται.

4.3.3 Μεταγλωτιστές PGI x64+GPU Fortran και C99

Ο εμπορικός μεταγλωτιστής PGI x86+GPU [40] είναι βασισμένος σε ιδέες που χρησιμοποιούνται από το πρότυπο παράλληλου προγραμματισμού OpenMP. Συγκεκριμένα, εισάγει ένα σύνολο απο οδηγίες (directives) ή pragmas (στην γλώσσα C) που ορίζουν ποιο τμήμα του κώδικα θα εκτελεστεί παράλληλα απο την κάρτα γραφικών. Αυτές οι οδηγίες οριοθετούν και περιγράφουν μια δομή επαναληπτικών εντολών και την επιλογή

των GPU block και νήματος(thread). Μια ακόμα λειτουργία αυτών των οδηγιών είναι ότι ο προγραμματιστής μπορεί να αποφασίσει ποια δεδομένα θα αντιγραφούν απο την μνήμη του κεντρικού επεξεργαστή (CPU) στην μνήμη της κάρτας γραφικών (GPU) και αντίστροφα. Οι οδηγίες για κάρτες γραφικών είναι παρόμοιες στη χρήση με αυτές του προτύπου OpenMP και έχουν περίπου το ίδιο επίπεδο πολυπλοκότητας στη κατασκευή ενός παράλληλου κώδικα. Το API (Application Programming Interface) υποστηρίζει τις γλώσσες C και Fortran

Ο μεταγλωτιστής PGI x64+GPU αναλύει την δομή του προγράμματος και των δεδομένων, και έπειτα χωρίζει το εκτελέσιμο πρόγραμμα σε τμήματα που εκτελούνται στη κάρτα γραφικών ή στο κεντρικό επεξεργαστή αντίστοιχα. Οι ανταλλαγές των δεδομένων ανάμεσα στις μνήμες της κάρτας γραφικών και του επεξεργαστή γίνεται αυτόματα χωρίς ο προγραμματιστής να χρειάζεται να οργανώνει και να σχεδιάζει τις ανταλλαγές, όπως στα πρότυπα CUDA και OpenCL. Αυτή η διαδικασία μπορεί να ελαχιστοποιεί τη πολυπλοκότητα της ανάπτυξης ενός παράλληλου αλγορίθμου στην κάρτα γραφικών, και ο προγραμματιστής είναι μόνο υπεύθυνος για όλες τις σημαντικές αποφάσεις μιας τέτοιας διαδικασίας όπως, η επιλόγη των υπολογιστικών πυρήνων σε μία επαναληπτική εντολή.

4.3.4 Συνδυασμός προτύπων MPI και CUDA

Μέχρι σήμερα πολλές εφαρμογές υπολογισμών υψηλής απόδοσης (High Performance Computing) έχουν αναπτυχθεί με το πρότυπο παράλληλου προγραμματισμού MPI. Το MPI επιτρέπει τον παράλληλο προγραμματισμό μεταξύ επεξεργαστών αποκλειστικής μνήμης μέσω ανταλλαγής μηνυμάτων. Ο απλούστερος τρόπος την υκοποίηση μιας εφαρμογής MPI που περιέχει kernels κάρτας γραφικών είναι να χρησιμοποιηθεί ο μεταγλωτιστής nvcc της Nvidia, ο οποίος διαθέτει και τις βιβλιοθήκες του προτύπου MPI.

Μία τεχνική για να συνδυαστεί το πρότυπο MPI με το CUDA είναι κάθε επεξεργαστής του MPI να έχει πρόσβαση αποκλειστικά σε μία μόνο κάρτα γραφικών. Αυτό όμως στα πε-

ρισσότερα πολυεπεξεργαστικά υπολογιστικά συστήματα θα είχε ως αποτέλεσμα αρκετοί επεξεργαστές να μην έχουν εργασία στην εκτέλεση της εφαρμογής, αφού ο αριθμός των επεξεργαστών (CPUs) είναι μεγαλύτερος από των αριθμό των καρτών γραφικών (GPUs). Έτσι η βέλτιστη επίδοση επιτυγχάνεται με την χρήση μιας κάρτα γραφικών για κάθε επεξεργαστή.

4.4 Βιβλιοθήκες Γραμμικής Άλγεβρας για πράξεις σε κάρτες γραφικών

Τα τελευταία χρόνια έχει γίνει μεγάλη προσπάθεια απο την ερευνητική κοινότητα να κατασκευαστούν εκδόσεις βιβλιοθηκών υποπρογραμμάτων τα οποία υλοποιούν βασικές διαδικασίες γραμμικής άλγεβρας σε γραφικά υποσυστήματα. Από τις πιο πετυχημένες νέες εκδόσεις των δημοφιλών βιβλιοθηκών LAPACK και PLASMA και απο τους ίδιους κατασκευαστές είναι η βιβλιοθήκη MAGMA [22]. Η βιβλιοθήκη αυτή περιλαμβάνει όλες τις αποδοτικές εκδόσεις των υποπρογραμμάτων των δύο παραπάνω βιβλιοθηκών για τους κατάλληλους γραφικούς επεξεργαστές. Απευθύνεται σε υλοποιήσεις γλωσσών προγραμματισμού C, C++ και Fortran για επιστημονικούς υπολογισμούς που σχετίζονται με πυκνούς πίνακες και είναι ελεύθερα διαθέσιμη.

Μια εμπορική έκδοση των σημαντικότερων υποπρογραμμάτων των βιβλιοθηκών BLAS και LAPACK περιλαμβάνει η CULA [37], η οποία διαθέτει δύο τμήματα. Το πρώτο διαχειρίζεται πράξεις με πυκνούς πίνακες ενώ το δεύτερο με αραιούς. Το δεύτερο τμήμα υλοποιεί τις επαναληπτικές μεθόδους CG, MINRES, BiCG, GMRES και BiCG-STAB με προρυθμίσεις Jacobi, Block-Jacobi και ατελούς παραγοντοιποίησης με μηδενικά γεμίσματα. Υποστηρίζονται οι γλώσσες προγραμματισμού C, C++, Fortran, Matlab και

Python. Μόνο η έκδοση για αριθμητική απλής ακρίβειας είναι ελεύθερα διαθέσιμη.
Μια άλλη ελεύθερα διαθέσιμα βιβλιοθήκη των παραπάνω επαναληπτικών μεθόδων για αραιούς πίνακες είναι η CUSP [12]. Υποστηρίζει προρυθμίσεις τύπου Jacobi, προσέγγισης αντιστρόφου και αλγεβρικής Τεχνικής Πολυπλέγματος.
Από την ερευνετική ομάδα του Yousef Saad αναπτύχθηκε το 2011 το ελεύθερο λογισμικό CUDA ITSOL [41]. Περιλαμβάνει βασικές πράξεις Γραμμικής Άλγεβρας καθώς και επιλύσεις αραιών γραμμικών συστημάτων οι οποίες βασίζονται στην ατελή παραγοντοποίηση. Παρακάτω περγράφονται σύντομα οι διαδικασίες απο τις βιβλιοθήκες υποπρογραμμάτων που χρησιμοποιήθηκαν σε αυτή την υλοποίηση.

4.4.1 Η βιβλιοθήκη CUBLAS

Στο πρότυπο CUDA, εκτός από τις εντολές που προστέθηκαν επιπλέον στην γλώσσα C, συμπεριλαμβάνονται υποπρογράμματα για πράξεις Γραμμικής Άλγεβρας σε κάρτες γραφικών, παραπλήσια με αυτά της γνωστής βιβλιοθήκης BLAS (Basic Linear Algebra Subroutines). Η βιβλιοθήκη αυτή ονομάζεται CUBLAS, αποτελεί λογισμικό ελεύθερης χρήσης και καλύπτει ένα μεγάλο πλήθος υποπρογραμμάτων που είναι αντίστοιχα με την BLAS. Η βιβλιοθήκη υποστηρίζει υποπρογράμματα πραγματικών αριθμών απλής και διπλής ακρίβειας καθώς και για τους αντίστοιχους μιγαδικούς αριθμούς. Επίσης, παράλληλα με την βιβλιοθήκη CUBLAS δίνονται κάποιες ακόμα βοηθητικές συναρτήσεις απαραίτητες για την σωστή λειτουργία της βιβλιοθήκης. Παρακάτω παρουσιάζονται οι υπορουτίνες της CUBLAS που χρησιμοποιήθηκαν σε αυτή τη διατριβή.

Το υποπρόγραμμα cublasDaxpy

Το υποπρόγραμμα cublasDaxpy υλοποιεί τον αλγόριθμο αθροίσματος δύο διανυσμάτων διάστασης N, $(Y = \alpha X + Y)$ όπου το α είναι βαθμωτό. Το υποπρόγραμμα καλείται με τον εξής τρόπο:

call cublasDaxpy (Handle , N , Alpha , X , incX , Y , incY)

με τα ορίσματα:

Ορίσματα εισόδου:

$Handle\ (H)$: μεταβλητή τύπου cublasHandle_t, αρχικοποιείται με την απαραίτητη κλήση της βοηθητικής συνάρτησης cublasCreate(Handle)

$N\ (H)$: ακέραιος, η διάσταση των διανυσμάτων X,Y

$Alpha\ (H/D)$: πραγματική μεταβλητή τύπου δείκτη, το βαθμωτό που πλολλαπλασιάζεται με το διάνυσμα X

$X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος X

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incY\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος Y

Ορίσματα εξόδου:

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

Το υποπρόγραμμα cublasDscal

Το υποπρόγραμμα cublasDscal υλοποιεί τον αλγόριθμο πολλαπλασιασμού ενός διανύσματος X διάστασης N με ένα βαθμωτό α. $(X = \alpha X)$ Το υποπρόγραμμα καλείται με τον εξής τρόπο:

call cublasDscal (Handle , N , Alpha , X , incX)

με τα ορίσματα:

Ορίσματα εισόδου:

$Handle\ (H)$: μεταβλητή τύπου cublasHandle_t, αρχικοποιείται με την απαραίτητη κλήση της βοηθητικής συνάρτησης cublasCreate(Handle)

$N\ (H)$: ακέραιος, η διάσταση του διανυσμάτος X

$Alpha\ (H/D)$: πραγματική μεταβλητή τύπου δείκτη, το βαθμωτό που πλολλαπλασιάζεται με το διάνυσμα X

$X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος X

Ορίσματα εξόδου:

$X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

Το υποπρόγραμμα cublasDcopy

Το υποπρόγραμμα cublasDcopy υλοποιεί τον αλγόριθμο αντιγραφής ενός διανύσματος X διάστασης N σε ένα διάνυσμα Y αντίστοιχης διάστασης. Το υποπρόγραμμα καλείται με τον εξής τρόπο:

call cublasDcopy (Handle , N , X , incX , Y , incY)

με τα ορίσματα:

Ορίσματα εισόδου:

$Handle\ (H)$: μεταβλητή τύπου cublasHandle_t, αρχικοποιείται με την απαραίτητη κλήση της βοηθητικής συνάρτησης cublasCreate(Handle)

$N\ (H)$: ακέραιος, η διάσταση του διανυσμάτος X

$X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος Y

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incY\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος Y

Ορίσματα εξόδου:

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

Το υποπρόγραμμα cublasSetVector

Το υποπρόγραμμα cublasSetVector υλοποιεί τον αλγόριθμο αντιγραφής ενός διανύσματος απο την μνήμη του κεντρικού επεξεργαστή στην μνήμη της κάρτας γραφικών. Το υποπρόγραμμα καλείται με τον εξής τρόπο:

call cublasSetVector (N , ElemSize , X , incX , DEV_X , incDEV_X)

με τα ορίσματα:

Ορίσματα εισόδου:

$N\ (H)$: ακέραιος, η διάσταση των διανυσμάτων X,DEV_X

$ElemSize\ (H)$: μεταβλητή τύπου sizeof(double)

$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος Ξ

$DEV\_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incDEV\_X\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος DEV_X

Ορίσματα εξόδου:

$DEV_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

Το υποπρόγραμμα cublasGetVector

Το υποπρόγραμμα cublasSetVector υλοποιεί τον αλγόριθμο αντιγραφής ενός διανύσματος απο την μνήμη της κάρτας γραφικών στην μνήμη του κεντρικού επεξεργαστή. Το υποπρόγραμμα καλείται με τον εξής τρόπο:

call cublasGetVector (N , ElemSize , DEV_X , incDEV_X , X , incX)

με τα ορίσματα:

Ορίσματα εισόδου:

$N\ (H)$: ακέραιος, η διάσταση των διανυσμάτων X,DEV_X

$ElemSize\ (H)$: μεταβλητή τύπου sizeof(double)

$DEV\_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incDEV\_X\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος DEV_X

$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος Ξ

Ορίσματα εξόδου:

$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

4.4.2 Η βιβλιοθήκη CUSPARSE

Στην προηγούμενη ενότητα παρουσιάστηκε η βιβλιοθήκη CUBLAS η οποία υποστηρίζει πίνακες πυκνής αποθήκευσης. Για την περίπτωση πινάκων αραιής αποθήκευσης υπάρχει η βιβλιοθήκη CUSPARSE. Στην συνέχεια παρουσιάζονται οι συναρτήσεις που χρησιμοποιήθηκαν για την εκπόνηση αυτής της διατριβής.

Το υποπρόγραμμα cusparseDcsrmv

Το υποπρόγραμμα cusparseDcsrmv υλοποιεί τον αλγόριθμο του πολλαπλασιασμού ενός αραιού πίνακα δομής CSR με διαστάσεις $M \times N$ με ένα διάνυσμα N θέσεων αντίστοιχα $(y = \alpha \cdot Ax + \beta y)$. Παρακάτω περιγράφεται ο τρόπος κλήσης του υποπρογράμματος:

call cusparseDcsrmv (Handle , transA , M , N , NNZ , Alpha , descrA , A , IA , JA , X , BETA , Y)

με τα ορίσματα:

Ορίσματα εισόδου:

$Handle\ (H)$: μεταβλητή τύπου cusparseHandle_t, αρχικοποιείται με την απαραίτητη κλήση της βοηθητικής συνάρτησης cusparseCreate(Handle)

$transA\ (H)$: μεταβλητή τύπου cusparseOperation_t, ορίζει τον τύπο του πίνακα μεταξύ των επιλογών CUSPARSE_OPERATION_NON_TRANSPOSE , CUSPARSE_OPERATION_TRANSPOSE και CUSPARSE_OPERATION_CONJUGATE_TRANSPOSE

$M\ (H)$: ακέραιος, οι γραμμές του πίνακα A

$N\ (H)$: ακέραιος, οι στήλες του πίνακα A

$NNZ\ (H)$: ακέραιος, το πλήθος των μη μηδενικών στοιχείων του πίνακα A

$Alpha\ (H)$: πραγματική μεταβλητή τύπου δείκτη, το βαθμωτό που πολλαπλασιάζεται με τον πίνακα A

$descrA\ (H)$: μεταβλητή τύπου cusparseMatDescr_t, ορίζει μια περιγραφή του πίνακα μεταξύ των επιλογών CUSPARSE_MATRIX_TYPE_GENERAL , CUSPARSE_MATRIX_TYPE_SYMMETRIC και CUSPARSE_MATRIX_TYPE_HERMITIAN

$A\ (D)$: πραγματικό διάνυσμα NNZ θέσεων που περιέχει τα μη μηδενικά στοιχεία του πίνακα A

$IA\ (D)$: ακέραιο διάνυσμα M+1 θέσεων που περιέχει τους δείκτες γραμμής των στοιχείων του A σύμφωνα με την τεχνική αποθήκευσης CSR

$JA\ (D)$: ακέραιο διάνυσμα NNZ θέσεων που περιέχει τις πληροφορίες για τις στήλες των μη μηδενικών στοιχείων του A

$X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$BETA\ (H)$: πραγματική μεταβλητή τύπου δείκτη, το βαθμωτό που πολλαπλασιάζεται με το διάνυσμα Y

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

Ορίσματα εξόδου:

$Y\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

4.5 CUDA και Streams

Σε μία γλώσσα προγραμματισμού ο όρος stream μπορεί να έχει διαφορετικές ερμηνείες. Όμως όλες αναφέρονται σε μια συχνότητα δεδομένων τα οποία είναι διαθέσιμα σε όλη την διάρκεια εκτέλεσης της εφαρμογής. Σε λειτουργικά συστήματα όπως το UNIX ή άλλα σχετικά συστήματα που βασίζονται στην γλώσσα C , το stream αποτελεί μία πηγή δεδομένων μετρημένα bytes. Σε μία παράλληλη διαδικασία, ειδικά για κάρτες γραφικών, ο όρος stream μπορεί να αναφερθεί στο υλικό ή στο λογισμικό αντίστοιχα και ορίζει μια ημισυνεχής ροή δεδομένων η οποία παράγεται σε ένα πρόγραμμα ροής την στιγμή όπου συναντά τη συνθήκη της αφετηρίας του stream. Στο προγραμματισμό κάρτας γραφικών για γενικούς υπολογισμούς, μπορούν να χρησιμοποιηθούν streams για την αύξηση της παραλληλίας του αλγορίθμου, αφού τα δεδόμενα μπορούν να στέλνονται και να λαμβάνονται ασύγχρονα. Παρακάτω παρουσιάζεται ο τρόπος αντικατάστασης των υποπρογραμμάτων cublasSetVector και cublasGetVector με την αντίστοιχη τους ασύγχρονη έκδοση.

Το υποπρόγραμμα cublasSetVectorAsync

Το υποπρόγραμμα cublasSetVectorAsync υλοποιεί τον αλγόριθμο ασύγχρονης αποστολής ενός διανύσματος απο τη μνήμη του κεντρικού επεξεργαστή στην μνήμη της κάρτας γραφικών. Το υποπρόγραμμα καλείται ως εξής:

call cublasSetVectorAsync (N , ElemSize , X , incX , DEV_X , incDEV_X , stream)

με τα ορίσματα:

Ορίσματα εισόδου:

$N\ (H)$: ακέραιος, η διάσταση των διανυσμάτων X,DEV_X

$ElemSize\ (H)$: μεταβλητή τύπου sizeof(double)

$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος X

$DEV\_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incDEV\_X\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος DEV_X

$stream\ (H)$: μεταβλητή τύπου cudaStream_t, αρχικοποιείται με την αναγκαία κλήση της συνάρτησης cublasSetStream(handle, stream)

Ορίσματα εξόδου:

$DEV\_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

Το υποπρόγραμμα cublasGetVectorAsync

Το υποπρόγραμμα cublasGetVectorAsync υλοποιεί τον αλγόριθμο ασύγχρονης αποστολής ενός διανύσματος απο την μνήμη της κάρτας γραφικών στην μνήμη του κεντρικού επεξεργαστή. Το υποπρόγραμμα καλείται ως εξής:

call cublasGetVectorAsync (N , ElemSize , DEV_X , incDEV_X , X , incX , stream)

με τα ορίσματα:

Ορίσματα εισόδου:

$N\ (H)$: ακέραιος, η διάσταση των διανυσμάτων X,DEV_X

$ElemSize\ (H)$: μεταβλητή τύπου sizeof(double)

$DEV\_X\ (D)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incDEV\_X\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος DEV_X

$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, διάστασης N

$incX\ (H)$: ακέραιος, το βήμα προσπέλασης του διανύσματος X

$stream\ (H)$: μεταβλητή τύπου cudaStream_t, αρχικοποιείται με την αναγκαία κλήση της συνάρτησης cublasSetStream(handle, stream)

Ορίσματα εξόδου:

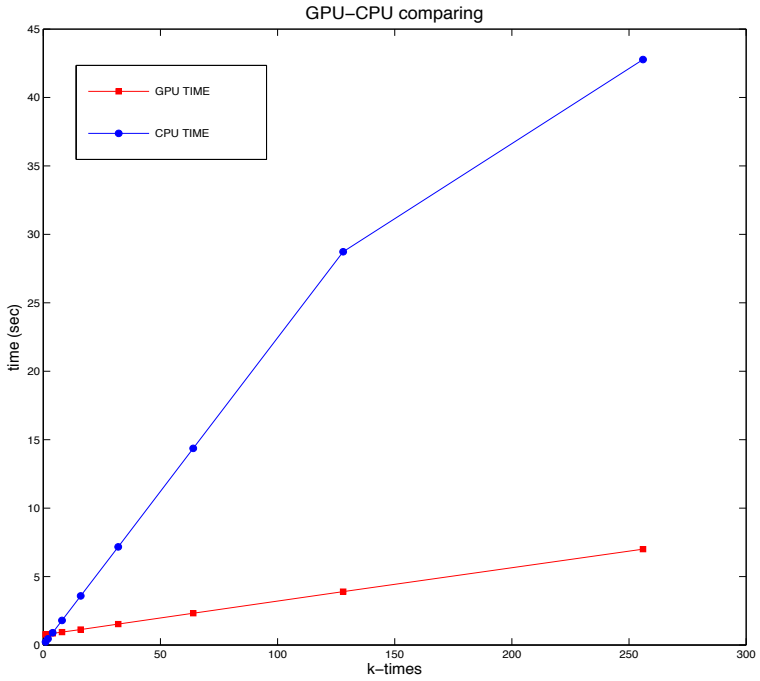
$X\ (H)$: πραγματικό διάνυσμα τύπου δείκτη, αποθηκεύεται το αποτέλεσμα του υποπρογράμματος

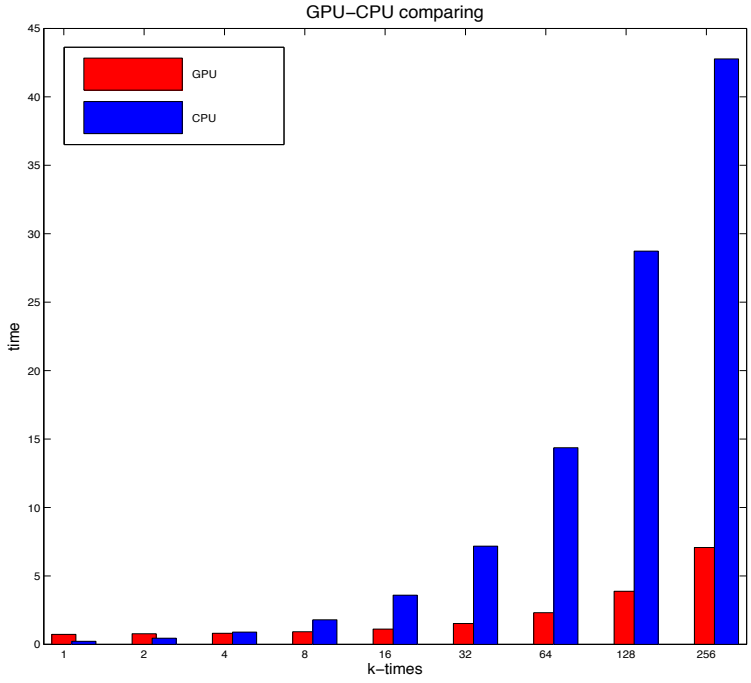
Όπου (H) ή (D) στο όνομα των μεταβλητών σημαίνει ότι η μεταβλητή πρέπει να είναι αποθηκευμένη στην μνήμη του κεντρικού επεξεργαστή ή της κάρτας γραφικών αντίστοιχα.

4.6 Παράδειγμα προγραμματισμού σε κάρτα γραφικών

Ο παράλληλος προγραμματισμός με χρήση κάρτας γραφικών έχει το μειονέκτημα της διαχείρησης του κόστους επικοινωνίας μεταξύ των επεξεργαστών απο τον προγραμματιστή. Η επικοινωνία των επεξεργαστών μπορεί να περιέχει από την ανταλαγή απλών αλφαριθμητικών μεταβλητών μέχρι διανύσματα εκατομμυρίων θέσεων. Σε κάθε περίπτωση η λανθασμένη διαχείρηση της επικοινωνίας μεταξύ των επεξεργαστών μπορεί να έχει καταστροφικά αποτελέσματα για την επίδοση, αφου επηρεάζει την διαδικασία συγχρονισμού τους. Γι' αυτό το λόγο ο προγραμματιστής πρέπει να υλοποιήσει την κατασκευή του αλγορίθμου έτσι ώστε ο χρόνος της επικοινωνίας μεταξύ των επεξεργαστών να ελαχιστοποιείται. Στην συνέχεια χρησιμοποιείται το παράδειγμα της πρόσθεσης δύο διανυσμάτων μεγέθους 10^8 και 10^7 θέσεων πραγματικών διπλής ακρίβειας.Συγκρίνεται η απόδοση διεξαγωγής των πράξεων στην κάρτα γραφικών τύπου Nvidia Tesla M2070 448 πυρήνων, 6 Gb μνήμης, εύρους 150 Gb/s και στον επεξεργαστή τύπου Xeon@2.8 GHz, 6 πυρήνων και 24 Gb μνήμης τα οποία διαθέτει το υπολογιστικό σύστημα. Η πρόσθεση των διανυσμάτων πραγματοποιείται κ-φορές, ενώ η αποστολή και η λήψη μόνο μία φορά. Ο παρακάτω πίνακας εμφανίζει τους χρόνους εκτέλεσης των επί μέρους διαδικασίων μετρημένους σε δευτερόλεπτα για τη περιπτώση μεγέθους διανυσμάτων 10^8 θέσεων.

κ	Αποστολή σε GPU	Πράξη σε GPU	Λήψη από GPU	Συνολικός GPU	Συνολικός CPU
1	0.439411	0.024930	0.268556	0.731888	0.224965
2	0.459107	0.049823	0.268641	0.775881	0.450931
4	0.444712	0.099518	0.268898	0.810876	0.895863
8	0.452815	0.199511	0.269376	0.919859	1.793727
16	0.448237	0.398812	0.268636	1.113831	3.593453
32	0.461959	0.797441	0.269210	1.524769	7.173910
64	0.454794	1.593737	0.268466	2.312649	14.364816
128	0.433322	3.187270	0.268681	3.881409	28.730632
256	0.441659	6.375949	0.268188	7.071925	42.772497





Σχήμα 4.1: Χρόνος εκτέλεσης σε CPU και GPU για διανύσματα μεγέθους 10^8.

Το παραπάνω εμφανίζει το συνολικό χρόνο εκτέλεσης στον επεξεργαστή και την κάρτα γραφικών του προβλήματος δοκιμής μέχρι 256 φορές. Παρατηρούμε ότι για πάνω απο 4 προσθέσεις ο χρόνος εκτέλεσης στην κάρτα γραφικών είναι μικρότερος απ΄ ότι στον επεξεργαστή, ενώ για 256 προσθέσεις διανυσμάτων ο χρόνος στην κάρτα γραφικών είναι πάνω απο 6 φορές ταχύτερος. Για πλήθος προσθέσεων μικρότερο απο 4 παρατηρούμε αυξημένο χρόνο εκτέλεσης στην κάρτα γραφικών σε σχέση με τον αντίστοιχο του επεξεργαστή, Αυτό συμβαίνει διότι ο χρόνος αποστολής και λήψης των διανυσμάτων απο την μνήμη του επεξεργαστή σε αυτή της κάρτας γραφικών και αντίστροφα αποτελεί τη πιο χρονοβόρα διαδικασία του αλγορίθμου σε σχέση με το χρόνο υπολογισμών στη κάρτα γραφικών.

Στη συνέχεια εμφανίζονται οι χρονομετρήσεις για μέγεθος διανυσμάτων 10^7.

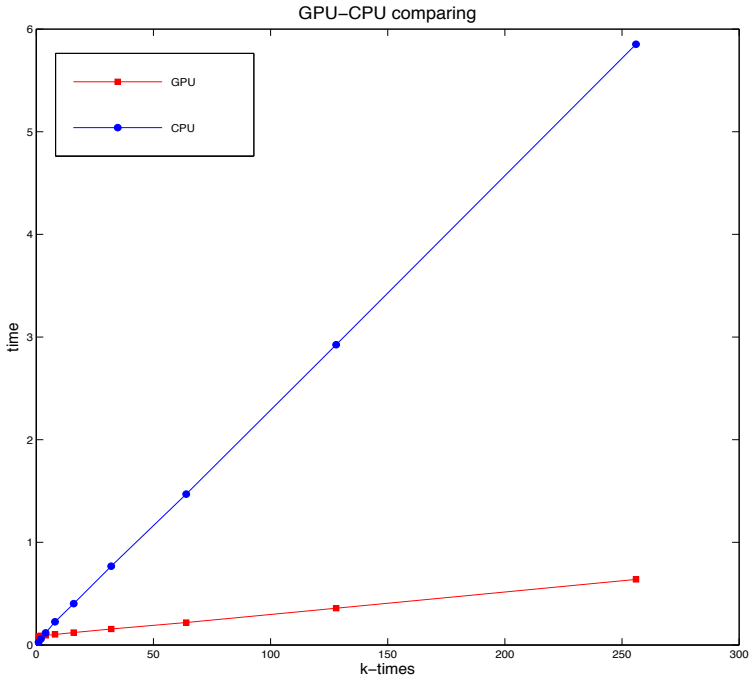
κ	Αποστολή GPU	Πράξη GPU	Λήψη GPU	Συνολικός GPU	Συνολικός CPU
1	0.049298	0.002254	0.032553	8.4988E-002	2.6995E-2
2	0.049367	0.004456	0.035642	8.9986E-002	5.8990E-2
4	0.049906	0.008859	0.035649	9.3984E-002	0.117982
8	0.049647	0.017663	0.035737	0.102983	0.225965
16	0.049522	0.035269	0.035609	0.120982	0.402938
32	0.049377	0.070551	0.035664	0.155976	0.767882
64	0.045987	0.140930	0.032005	0.218967	1.469777
128	0.049623	0.281900	0.027424	0.357944	2.924555
256	0.049143	0.563866	0.027392	0.638901	5.852110

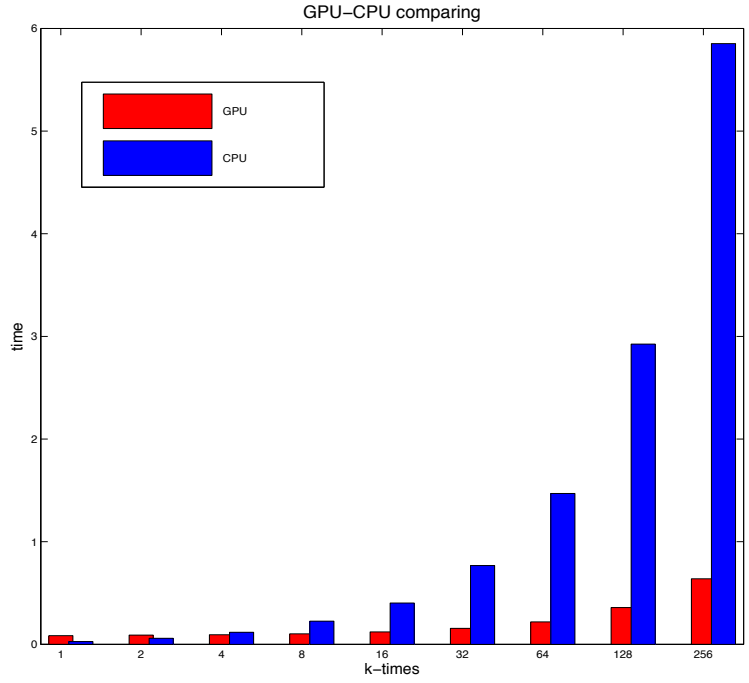
Στη πρώτη περίπτωση όπου τα διανύσματα είχαν διάσταση 10^8 η χωρητικότητα κάθενος στη μνήμη της κάρτας γραφικών ήταν :

$$100,000,000 \cdot 8\ bytes = 800,000,000\ bytes$$

ή

$$800,000,000/1024^3 \simeq 0.7450\ GBytes$$





Σχήμα 4.2: Χρόνος εκτέλεσης σε CPU και GPU για διανύσματα μεγέθους 10^7.

Ο μέσος χρόνος αποστολής/λήψης ενός τέτοιου διανύσματος τέτοιου μεγέθους είναι περίπου 0.2688 δευτερόλεπτα, οπότε η ταχύτητα αποστολής/λήψης θα είναι περίπου ίση με $\frac{0.7450}{0.2685} = 2.7746\ Gbyte/Sec$.

Επειδή ο χρόνος εκτέλεσης της διαδικασίας στη κάρτα γραφικών είναι το άθροισμα των χρόνων επικοινωνίας και υπολογισμών, παρατηρούμε ότι είναι αποδοτικότερη η εκτέλεση της διαδικασίας στην κάρτα μόνο εάν ο χρόνος υπολογισμών είναι σημαντικά αυξημένος σε σχέση με το χρόνο μεταφοράς των δεδομένων στην μνήμη της. Έτσι για τη διεξαγωγή υπολογισμών που εμπλέκουν πάνω απο 4 προσθέσεις διανυσμάτων ο χρόνος εκτέλεσης στον επεξεργαστή είναι μεγαλύτερος και αυτό ισχύει ανεξάρτητα απο το μέγεθος των δεδομένων.

Στη συνέχεια παρουσιάζονται τεχνικές αποστολής και λήψης διανυσμάτων ανάμεσα στις μνήμες της κάρτας γραφικών και του κεντρικού επεξεργαστή με σκοπό την μείωση του χρόνου εκτέλεσης.

4.6.1 Αποστολή και Λήψη με ένα STREAM

Όπως παρουσιάστηκε στη προηγούμενη παράγραφο (4.5), είναι εφικτή η αντικατάσταση μιας συγχρονισμένης διαδικασίας αποστολής και λήψης δεδομένων με μια ασύγχρονη με τη χρήση ενός stream. Στη συνέχεια παρουσιάζεται η αύξηση του ρυθμού μεταφοράς των δεδομένων για την περίπτωση χρήσης διανυσμάτων 10^8 θέσεων. Έγινε ομοιόμορφη διαμέριση κάθε διανύσματος σε 100 υποδιανύσματα. Εκτελέστηκαν 16 επιτρεπτές απο τα τεχνικά χαρακτηριστικά της κάρτας ταυτόχρονες συναρτήσεις kernel. Στη συνέχεια με τη χρήση μιας επαναληπτικής διαδικασίας πραγματοποιείται η αποστολή των υποδιανυσμάτων με ασύγχρονο ρυθμό με τη χρήση των υποπρογραμμάτων cublasSetVectorAsync και cublasGetVectorAsync.

Ο συνολικός χρόνος υλοποίησης αυτής της διαδικασίας ήταν 0.520449 δευτερόλεπτα

και είναι κατα 28% ταχύτερος από τη συγχρονισμένη έκδοση της όπου ο χρόνος ήταν 0.731888. Επίσης, η ταχύτητα λήψης του διανύσματος έγινε σε 0.184469 δευτερόλεπτα και ο ρυθμός αποστολής/λήψης ήταν $4.04\ Gbyte/Sec$, δηλαδή περίπου δύο φορές γρηγορότερα από την αντίστοιχη συγχρονισμένη αποστολή/λήψη.

4.6.2 Αποστολή και Λήψη με πολλαπλών STREAMS

Η χρήση ενός stream για κάθε υποδιάνυσμα θα μπορούσε να πολλαπλασιάσει τη μείωση του χρόνου μεταφοράς των δεδομένων. Έτσι άν χρησιμοποιηθούν 10 stream ταυτόχρονα για να εκτελέσουν το υποπρόγραμμα τότε κάθε stream θα πρέπει αντιπροσωπεύει ένα τμήμα διανύσματος 10,000,000 θέσεων. Ο συνολικός χρόνος εκτέλεσης για αυτή τη περίπτωση ήταν 0.499358, δηλαδή 31.7% και 4% ταχύτερος απο τον αντίστοιχο της συγχρονισμένης και της ασύγχρονης με ένα stream. Ο χρόνος αποστολής/λήψης ήταν 0.156760 δευτερόλεπτα, δηλαδή ο ρυθμός μεταφοράς έγινε $4.7\ Gbyte/Sec$.

4.6.3 Αποστολή και Λήψη μηδενικής αντιγραφής

Η τεχνική αυτή καταργεί τη διαδικασία αποστολής και λήψης δεδομένων, αφού δημιουργούνται δείκτες μεταβλητές οι οποίες έχουν άμεση πρόσβαση στην κεντρική μνήμη του συστήματος. Αξίζει να σημειωθεί ότι η δυνατότητα αυτή παρέχεται μόνο απο συγκεκριμένους τύπους καρτών. Οι προϋποθέσεις αυτής της τεχνικής είναι τα αποθηκευμένα διανύσματα στην κεντρική μνήμη να έχουν δηλωθεί με την εντολή cudaHostAlloc με τα flags : cudaHostAllocWriteCombined και cudaHostAllocMapped και αντίστοιχα για την κάρτα γραφικών να έχει οριστεί το flag : cudaDeviceMapHost καθώς και να έχει γίνει κλήση της βοηθητικής συνάρτησης cudaHostGetDevicePointer για κάθε διάνυσμα. Η διαδικασία της αντιστοίχησης ενός διανύσματος απο τη κεντρική μνήμη του κεντρικού επεξεργαστή στη μνήμη της κάρτας γραφικών είναι αρκετά χρονοβόρα. Για παράδειγμα

για δύο διανύσματα 100,000,000 θέσεων απαιτούνται 6.046116 δευτερόλεπτα. Η τεχνική δεν είναι συμφέρουσα όταν πρέπει να εκτελεστούν λίγες συναρτήσεις kernel, εξαιτίας του αυξημένου χρόνου αντιστοίχησης των δεδομένων.

Κεφάλαιο 5

Υλοποίηση και μελέτη της συμπεριφοράς του αλγορίθμου

5.1 Εισαγωγή

Το κεφάλαιο αυτό παρουσιάζει τις μετρήσεις απόδοσης από την υλοποίηση της μεθόδου Πεπερασμένων Στοιχείων Hermite Collocation με τη χρήση της Τεχνικής Πολυπλέγματος σε μηχανές οι οποίες διαθέτουν γραφικά υποσυστήματα για τη διεξαγωγή των επιστημονικών υπολογισμών. Το μηχάνημα που χρησιμοποιήθηκε είναι τύπου HP SL390, το οποίο διαθέτει δύο επεξεργαστές Xeon@2.8 GHz 6 υπολογιστικών πυρήνων για τον καθένα και 24GB συνολικής μνήμης. Το γραφικό υποσύστημα αποτελείται απο δύο κάρτες γραφικών τύπου Nvidia Tesla M2070 των 448 πυρήνων, 6GB μνήμης με εύρος 150GB/sec στη κάθε μία. Η επικοινωνία μεταξύ υπολογιστικού συστήματος και κάρτας γραφικών πραγματοποιείται μέσω ανεξάρτητων διαύλων τύπου x16 PCI-e Gen2. Το λειτουργικό σύστημα είναι η έκδοση Oracle Linux 6.2 x64, ο Fortran μεταγλωτιστής είναι η έκδοση 4.4.6 της GNU υλοποίησης, ενώ η έκδοση του CUDA Toolkit είναι η 5.0.
Στον αλγόριθμο της αριθμητικής μεθόδου η Τεχνική Πολυπλέγματος χρησιμοποίησε το σχήμα εξομάλυνσης της επαναληπτικής μεθόδου Gauss Seidel σε συνδυασμό με την ατελή διάσπαση πίνακα ILU(k). Ο επαναληπτικός πίνακας $G = M^{-1}N$ του επαναληπτικού σχήματος

$$x^{(k+1)} = Gx^k + b \quad k = 1, 2, \ldots$$

της μεθόδου Gauss Seidel κατασκευάστηκε σύμφωνα με τη διάσπαση του πίνακα συντελεστών που περιγράφεται στα σχήματα (5.1) και (5.2) . Το επίπεδο γεμίσματος (LFIL) της ατελούς παραγοντοποίησης του πίνακα M καθορίζεται από μία συνάρτηση $lfil(l)$, όπου l είναι το επίπεδο του πλέγματος της Τεχνικής Πολυπλέγματος. Τα επίπεδα πλέγματος καθορίζονται ώς εξής :

ns	8	16	32	64	128	256	512	1024	2048
l	1	2	3	4	5	6	7	8	9

Για να είναι εφικτή η μελέτη και η σύγκριση ανάμεσα σε διαφορετικά προβλήματα έγινε χρήση της νόρμα 2 του σφάλματος $e = u - v$, της νόρμα 2 του διανύσματος υπολοίπου $r = b - Ax$, η τάξη σύγκλισης της μεθόδου και του χρόνου υλοποίησης της αριθμητικής μεθόδου.

Η Τεχνική Πολυπλέγματος εφαρμόστηκε στη πιο αποδοτική της μορφή σύμφωνα με τη () η οποία περιλαμβάνει το τελεστή παρεκβολής Full Weighting και το τελεστή παρεμβολής bilinear injection.

5.2 Πρώτο πρόβλημα δοκιμής

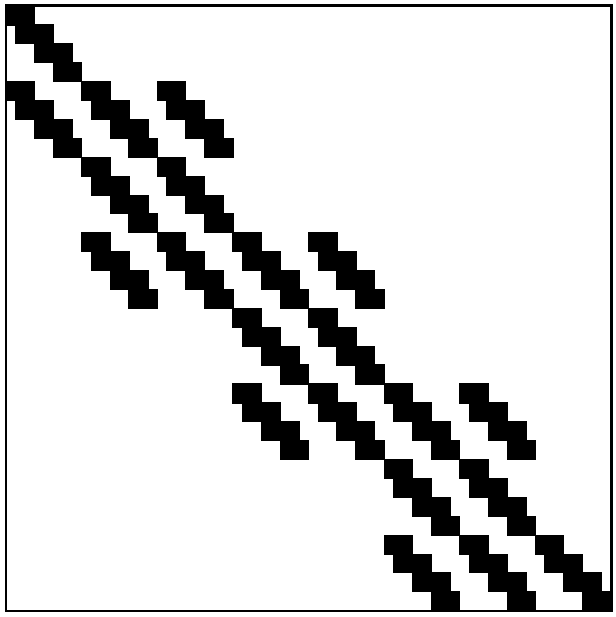
Αρχικά επιλύθηκε το παρακάτω ΠΣΤ :

$$\begin{cases} u_{xx}(x,y) + u_{yy}(x,y) + P(x,y)u_x(x,y) + Q(x,y)u_y(x,y) = f(x,y) \quad , \quad (x,y) \in \Omega \equiv [0,1] \times [0,1] \\ \\ u(x,y) = 0 \quad , \quad \in \partial\Omega \\ \\ P(x,y) = 10x(1-y) \quad , \quad Q(x,y) = 10y(1-x) \end{cases} \tag{5.1}$$

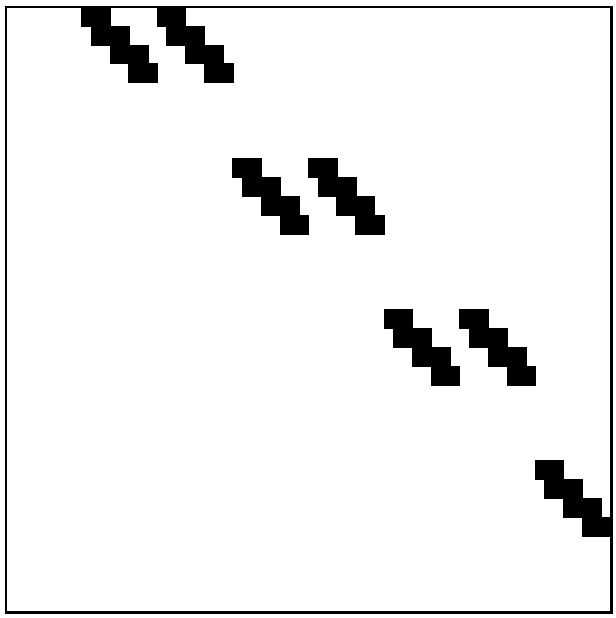
το οποίο έχει την αναλυτική λύση

$$u(x,y) = x^2y^2(1-x)(1-y)$$

η οποία εμφανίζεται στο σχήμα 5.3 . Στη συνέχεια οι πίνακες T1 και T2 περιλαμβάνουν τις μετρήσεις χρόνου και σφαλμάτων για διακριτοποιήσεις μεγέθους $n_s = 16, 32, 64, 128, 256, 512$

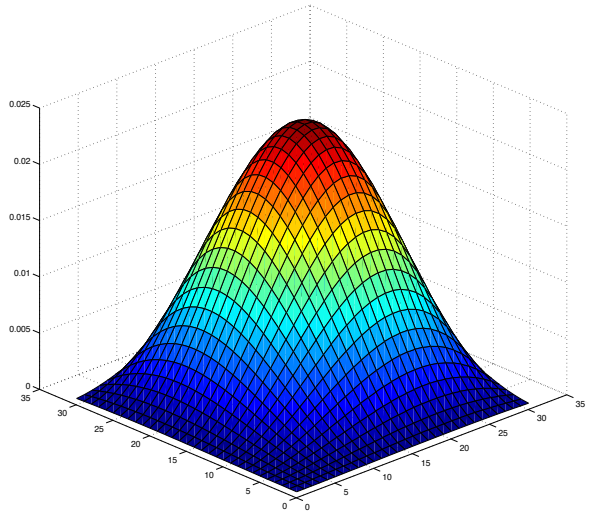


Σχήμα 5.1: Δομή του M Collocation Πίνακα.



Σχήμα 5.2: Δομή του N Collocation Πίνακα.

και 1024. Το κατώτερο επίπεδο της τεχνικής πολυπλέγματος σε όλες τις περιπτώσεις είναι το πλέγμα $n_s = 8$.



Σχήμα 5.3: Η ακριβής λύση $u = u(x, y)$ για το πρόβλημα δοκιμής 1.

T1	Multigrid technique with ILU(l)				
ns	$v - cycles$	$MG\ Time$	$\|\|u - x\|\|_2$	$\|\|u - x_c\|\|_2$	$\|\|b - Ax\|\|_2$
16	6	1.5998E-002	5.9114E-004	5.8204E-004	3.3066E-003
32	10	4.8992E-002	2.0986E-004	2.0868E-004	8.3536E-004
64	13	0.1439	8.4162E-005	8.3999E-005	2.1517E-004
128	13	0.4639	4.0570E-005	4.0548E-005	5.6899E-005
256	14	2.4146	2.1548E-005	2.1545E-005	1.5354E-005
512	14	10.5653	1.1827E-005	1.1826E-005	4.1993E-006
1024	15	50.0843	6.5575E-006	6.5575E-006	1.1582E-006

Ο πίνακας T1 παρουσιάζει τα αριθμητικά αποτελέσματα με επίπεδο γεμίσματος της ατελούς παραγοντοποίησης $LFIL = l$. Όπως αναμενόταν η χρήση περιορισμένου επιπέδου γεμίσματος της ατελούς διάσπασης πίνακα αύξησε τα σφάλματα με συνέπεια τον περιορισμό της τάξης σύγκλισης της μεθόδου. Το μόνο όφελος από αυτή την επιλογή

είναι η ταχύτερη παραγοντοποίηση του πίνακα, με περιορισμό του μεγέθους του και τον ταχύτερο τερματισμό της τεχνικής πολυπλέγματος. Στον πίνακα T2 παρουσιάζονται τα αριθμητικά αποτελέσματα της τεχνικής πολυπλέγματος με επίπεδο γεμίσματος ατελούς διάσπασης πίνακα $LFIL = 2l$.

T2	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	6	1.7996E-002	1.1279E-005	1.1128E-005	7.5309E-005
32	7	3.9993E-002	6.2896E-007	6.2569E-007	2.6437E-006
64	12	0.12398	3.5204E-008	3.5137E-008	9.0879E-008
128	15	0.7118	2.2701E-009	2.2688E-009	3.2344E-009
256	20	4.4613	1.6851E-010	1.6848E-010	1.1884E-010
512	27	25.6451	2.3317E-011	2.3317E-011	4.4437E-012
1024	35	156.4312	1.2239E-010	1.2239E-010	1.7398E-013

Όπως προκύπτει ο διπλασιασμός του επίπεδου γεμίσματος ανά επίπεδο πλέγματος βελτίωσε τα σφάλματα και αύξησε τη τάξη σύγκλισης της μεθόδου. Αυτό είχε ώς αποτέλεσμα τη σημαντική αύξηση του χρόνου εκτέλεσης, την αύξηση των θέσεων μνήμης αποθήκευσης του παραγοντοποιημένου πίνακα.

T3	Τάξη σύγκλισης			
ns	$\|\|u-x\|\|_2$	Τάξη σύγκλισης	$\|\|u-x_c\|\|_2$	Τάξη σύγκλισης
16	1.1279E-005	-	1.1128E-005	-
32	6.2896E-007	4.1645	6.2569E-007	4.1526
64	3.5204E-008	4.1592	3.5137E-008	4.1544
128	2.2701E-009	3.9549	2.2688E-009	3.9530
256	1.6851E-010	3.7518	1.6848E-010	3.7513

Ο πίνακας T3 εμφανίζει τη τάξη σύγκλισης της μεθόδου ως προς τα σφάλματα υπολογισμένα στους κόμβους και στα κέντρα των υπολογιστικών κελιών.

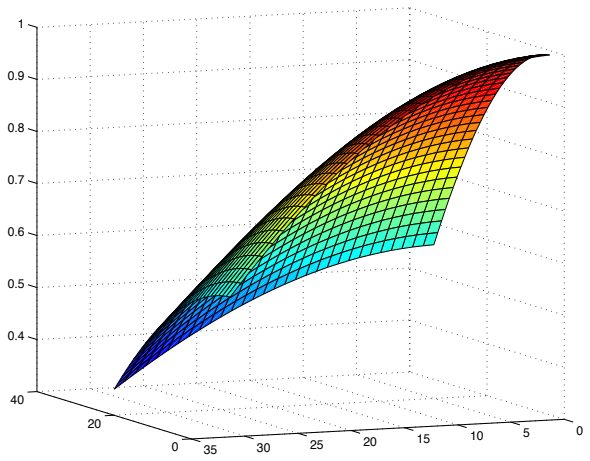
5.3 Δεύτερο πρόβλημα δοκιμής

Θεωρούμε το ΠΣΤ :

$$\begin{cases} u_{xx}(x,y) + u_{xy}(x,y) + u_{yy}(x,y) + u_x(x,y) + u_y(x,y) = f(x,y) \quad , \quad (x,y) \in \Omega \equiv [0,1] \times [0,1] \\ \\ u(x,y) = cos(x) \cdot cos(y) \quad , \quad \in \partial\Omega \end{cases} \tag{5.2}$$

το οποίο έχει αναλυτική λύση

$$u(x,y) = cos(x) \cdot cos(y)$$



Σχήμα 5.4: Η ακριβής λύση $u = u(x,y)$ για το πρόβλημα δοκιμής 2.

η οποία εμφανίζεται στο σχήμα 5.4. Στη συνέχεια οι πίνακες Τ4 και Τ5 περιλαμβάνουν τις μετρήσεις χρόνου και σφαλμάτων για διακριτοποιήσεις μεγέθους $n_s = 16, 32, 64, 128, 256, 512$ και 1024 με την διαφορά ότι αντί για $lfil(l) = l$ θα χρησιμοποιηθεί η $lfil(l) = l + 4$.

T4	Multigrid technique with ILU(l+4)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	7	2.6995E-002	1.0405E-003	3.8603E-003	1.1178E-003
32	10	6.7989E-002	5.8670E-004	1.9393E-003	5.5189E-004
64	11	0.1249	3.1673E-004	9.7544E-004	2.5496E-004
128	12	0.5859	1.6884E-004	4.9060E-004	1.1567E-004
256	13	2.9605	9.1403E-005	2.4758E-004	5.2215E-005
512	13	12.6370	5.2410E-005	1.2636E-004	2.3545E-005
1024	16	62.6794	3.3589E-005	6.6570E-005	1.0576E-005

T5	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	7	2.6995E-002	3.4989E-003	5.0093E-003	2.3667E-002
32	8	5.3991E-002	5.6255E-004	1.9242E-003	2.4861E-003
64	11	0.1249	3.1673E-004	9.7544E-004	2.5496E-004
128	12	0.6279	1.7322E-004	4.9223E-004	2.5958E-005
256	15	3.8314	8.9227E-005	2.4682E-004	2.6378E-006
512	20	20.64386	4.5159E-005	1.2354E-004	2.5965E-007
1024	26	136.8781	2.2701E-005	6.1797E-005	2.6283E-008

Από τους πίνακες Τ4 και Τ5 προκύπτει ότι η συμπεριφορά του αλγορίθμου σε αυτό το πρόβλημα δοκιμής έχει ανάλογη συμπεριφορά με αυτή του πρώτου προβλήματος, με τη διαφορά ότι η τάξη σύγκλισης για αυτό το πρόβλημα έχει μειωθεί.

5.4 Τρίτο πρόβλημα δοκιμής

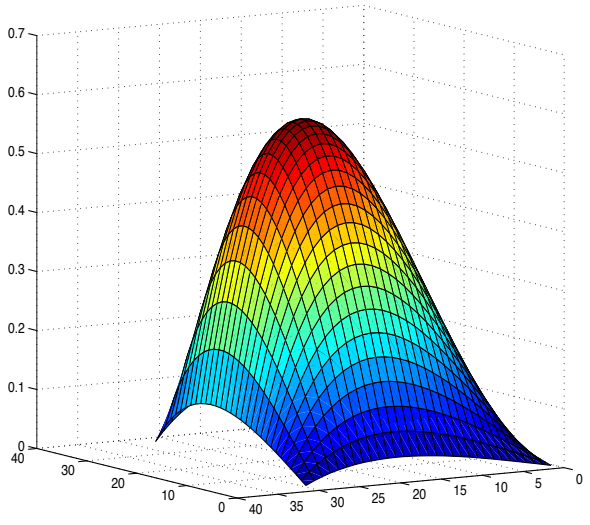
Θεωρούμε το ΠΣΤ :

$$\begin{cases} u_{xx}(x,y) + u_{yy}(x,y) = 6xye^{x+y}(xy + x + y - 3) \quad , \quad (x,y) \in \Omega \equiv [0,1] \times [0,1] \\ \\ u(x,y) = 0 \quad , \quad \in \partial\Omega \end{cases} \tag{5.3}$$

το οποίο έχει αναλυτική λύση

$$u(x,y) = 3e^{x+y}(x - x^2)(y - y^2)$$

η οποία εμφανίζεται στο σχήμα 5.5. Στο τρίτο πρόβλημα δοκιμής εξετάζεται ένα ΠΣΤ με διαφορικό τελεστή Laplace και μηδενικές συνοριακές συνθήκες. Στους πίνακες Τ6 και Τ7 παρουσιάζονται τα αριθμητικά αποτελέσματα της μεθόδου για τα επίπεδα πλέγματος και το επίπεδο γεμίσματος που χρησιμοποιήθηκαν στα προηγούμενα προβλήματα δοκιμής.



Σχήμα 5.5: Η ακριβής λύση $u = u(x,y)$ για το πρόβλημα δοκιμής 3.

T6	Multigrid technique with ILU(l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	5	1.6997E-002	1.4888E-002	1.4649E-002	7.6278E-002
32	7	3.5994E-002	4.9344E-003	4.9009E-003	2.0870E-002
64	7	6.6989E-002	2.0991E-003	2.0944E-003	5.7817E-003
128	9	0.3609	1.1190E-003	1.1183E-003	1.6096E-003
256	12	1.9027	6.3539E-004	6.3531E-004	4.4887E-004
512	12	9.1476	3.6154E-004	3.6153E-004	1.2524E-004
1024	14	46.5649	2.0431E-004	2.0431E-004	3.4950E-005

T7	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	6	2.2995E-002	2.5564E-004	2.3852E-004	1.3808E-003
32	8	5.3991E-002	1.3188E-005	1.1480E-005	5.5526E-005
64	11	0.12398	8.0833E-007	7.1942E-007	2.1803E-006
128	16	0.7858	6.0752E-008	7.3746E-008	8.5266E-008
256	22	4.8432	4.7970E-009	7.4678E-009	3.3198E-009
512	27	25.5741	8.1339E-010	1.4503E-009	1.2930E-010
1024	36	153.7326	5.4461E-009	5.5322E-009	5.0682E-012

5.5 Υλοποίηση σε γραφικά υποσυστήματα

Η ενότητα αυτή παρουσιάζει τα αποτελέσματα από την υλοποίηση του τρίτου προβλήματος δοκιμής με την επικουρική χρήση του γραφικού υποσυστήματος της μηχανής. Πιο συγκεκριμένα το γραφικό υποσύστημα ανέλαβε τη διεξαγωγή των υπολογισμών του σχήματος εξομάλυνσης της Τεχνικής Πολυπλέγματος. Η υλοποίηση στο πολυεπεξεργαστικό γραφικό υποσύστημα στηρίχτηκε σε 3 διαφορετικές προσεγγίσεις παραλληλοποίησης του αλγορίθμου της διαδικασίας αυτής. Ο βασικότερος στόχος ήταν η ελαχιστοποίηση της επικοινωνίας μεταξύ του κεντρικού επεξεργαστή και της κάρτας γραφικών. Παρακάτω παρουσιάζονται και συγκρίνονται αυτές οι 3 διαφορετικές προσεγγίσεις.

Η πρώτη υλοποίηση έγινε σύμφωνα με τον παρακάτω αλγόριθμο :

Αρχικοποίηση παραμέτρων της κάρτας γραφικών

Δημιουργία πινάκων Collocation C_l , όπου l το επίπεδο πλέγματος

Διάσπαση των πινάκων C_l σε κάτω και άνω τριγωνικούς πίνακες DL_l και U_l

Παραγοντοποίηση των πινάκων DL_l

Αποστολή απο CPU σε GPU τους πίνακες U_l

Αρχή Τεχνικής Πολυπλέγματος με χρήση του σχήματος Gauss Seidel

Gauss Seidel:

Αποστολή απο CPU σε GPU τα διανύσματα xo και rhs

Για $i = 1, iter$

$dclonex = dxo$ (στη κάρτα)

$dyo = dU_l \cdot dxo$ (στη κάρτα)

$dyo = -dyo$ (στη κάρτα)

$dyo = drhs + dyo$ (στη κάρτα)

Αποστολή από GPU σε CPU $xo = dyo$

$DL_l x = xo$ (στο κεντρικό επεξεργαστή)

Αποστολή απο CPU σε GPU $dx = x$

$dclonex = -dx + dclonex$ (στη κάρτα)

$dxo = dx$ (στη κάρτα)

Έλεγχος $\frac{||dclonex||_2}{||dx||_2} < 10^{-8}$

Τέλος Για

Ο πίνακας T8 περιλαμβάνει τις μετρήσεις χρόνου και σφάλματος για όλες τις διακριτοποιήσεις :

T8	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	6	0.2709	2.5564E-004	2.3852E-004	1.3808E-003
32	8	0.3529	1.3188E-005	1.1480E-005	5.5526E-005
64	11	0.5599	8.0833E-007	7.1942E-007	2.1803E-006
128	16	1.4497	6.0871E-008	7.3904E-008	8.5266E-008
256	22	5.6671	4.7973E-009	7.4678E-009	3.3198E-009
512	27	26.7189	8.1179E-010	1.4481E-009	1.2930E-010
1024	36	174.2864	5.4473E-009	5.5334E-009	5.0687E-012

Η δεύτερη υλοποίηση περιλαμβάνει τη διεξαγωγή των πράξεων πολλαπλασιασμού πίνακα με διάνυσμα στο κεντρικό επεξεργαστή του υπολογιστικού συστήματος σύμφωνα με το παρακάτω αλγόριθμο :

Αρχικοποίηση παραμέτρων της κάρτας γραφικών

Δημιουργία πινάκων Collocation C_l , όπου l το επίπεδο πλέγματος

Διάσπαση των πινάκων C_l σε κάτω και άνω τριγωνικούς πίνακες DL_l και U_l

Παραγοντοποίηση των πινάκων DL_l

Αρχή Τεχνικής Πολυπλέγματος με χρήση του σχήματος Gauss Seidel

Gauss Seidel:

Αποστολή απο CPU σε GPU τα διανύσματα xo και rhs

Για $i = 1, iter$

$dclonex = dxo$ (στη κάρτα)

$xo = U_l \cdot xo$ (στο κεντρικό επεξεργαστή)

Αποστολή απο CPU σε GPU $dyo = xo$

$dyo = -dyo$ (στη κάρτα)

$dyo = drhs + dyo$ (στη κάρτα)

Αποστολή από GPU σε CPU $xo = dyo$

$DL_l x = xo$ (στο κεντρικό επεξεργαστή)

Αποστολή απο CPU σε GPU $dx = x$

$dclonex = -dx + dclonex$ (στη κάρτα)

$dxo = dx$ (στη κάρτα)

Εάν $\frac{||dclonex||_2}{||dx||_2} < 10^{-8}$

Τερματισμός

Αλλιώς

Αποστολή από GPU σε CPU $xo = dxo$

Τέλος Εάν

Τέλος Για

Ο πίνακας T9 περιέχει τις μετρήσεις αυτής της υλοποίησης.

T9	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	6	0.2789	2.5564E-004	2.3852E-004	1.3808E-003
32	8	0.3949	1.3188E-005	1.1480E-005	5.5526E-005
64	11	0.6379	8.0833E-007	7.1942E-007	2.1803E-006
128	16	1.5857	6.0871E-008	7.3904E-008	8.5266E-008
256	22	5.6561	4.7973E-009	7.4678E-009	3.3198E-009
512	27	26.6309	8.1179E-010	1.4481E-009	1.2930E-010
1024	36	174.2864	5.4473E-009	5.5334E-009	5.0687E-012

Όπως προκύπτει για αραιές διακριτοποιήσεις υπάρχει μία μικρή χρονική επιβάρυνση, ενώ για πυκνές δεν παρουσιάστηκε μεταβολή. Αυτό οφείλεται στο γεγονός της επιπρόσθετης αποστολής δύο διανυσμάτων μεγέθους $m = 4n_s^2$ μεταξύ κεντρικού επεξεργαστή και κάρτας γραφικών.

Η τελευταία δοκιμαστική υλοποίηση στηρίχτηκε στην αρχή της διεξαγωγής του συνόλου των υπολογισμών στο γραφικό υποσύστημα για τη διαδικασία εξομάλυνσης της Τεχνικής Πολυπλέγματος. Αυτό έχει ως αποτέλεσμα τη μηδενική επικοινωνία μεταξύ επεξεργαστών. Ο παρακάτω πίνακας εμφανίζει 3 μετρήσεις για αραιές διακριτοποιήσεις.

T9	Multigrid technique with ILU(2l)				
ns	$v-cycles$	$MG\ Time$	$\|\|u-x\|\|_2$	$\|\|u-x_c\|\|_2$	$\|\|b-Ax\|\|_2$
16	6	1.12582	2.5564E-004	2.3852E-004	1.3808E-003
32	8	2.3066	1.3188E-005	1.1480E-005	5.5526E-005
64	11	7.2928	8.0833E-007	7.1942E-007	2.1803E-006

Παρατηρούμε μια σημαντική χρονική επιβάρυνση η οποία οφείλεται στη διαδικασία της μπρός πίσω αντικατάστασης. Ο συγχρονισμένος αλγόριθμος αυτής της διαδικασίας ευθύνεται για την καθυστέρηση του χρόνου εκτέλεσης της διαδικασίας εξομάλυνσης της Τεχνικής Πολυπλέγματος στη κάρτα γραφικών.

5.6 Συμπεράσματα

Σε αυτή τη διατριβή εφαρμόστηκε η αριθμητική μέθοδος Πεπερασμένων Στοιχείων Hermite Collocation στην επίλυση Προβλημάτων Συνοριακών Τιμών ελλειπτικού τύπου. Η κατασκευή αλγορίθμου και η υλοποίηση του πραγματοποιήθηκε στο γενικό τελεστή δεύτερης τάξης για οποιοδήποτε τύπο συνοριακών συνθηκών. Ο αλγόριθμος περιλαμβάνει ένα αρχικό στάδιο άμεσου υπολογισμού συνοριακών αγνώστων με αριθμητικές προσεγγίσεις υψηλής τάξης σφαλμάτων. Αυτό έχει ως αποτέλεσμα τον περιορισμό του πλήθους των αγνώστων σε $4n_s^2$, όπου n_s το πλήθος των πεπερασμένων στοιχείων ανά κατεύθυνση. Το παραγόμενο γραμμικό σύστημα αλγεβρικών εξισώσεων διαθέτει πίνακα συντελεστών block τριδιαγώνιας αραιής μορφής. Η επίλυση του Collocation γραμμικού συστήματος πραγματοποιειείται αποδοτικά με τη χρήση Τεχνικών Πολυπλέγματος και της επαναληπτικής μεθόδου Gauss Seidel για την εξομάλυνση του σφάλματος. Τα αποτελέσματα των μετρήσεων επαλήθευσαν την υψηλή τάξη ακρίβειας της μεθόδου τόσο στους κόμβους του πλέγματος, όσο και σε οποιοδήποτε άλλο σημείο του. Η επαναληπτική διαδικασία εξομάλυνσης του σφάλματος διαθέτει το μεγαλύτερο υπολογιστικό κόστος του αλγορίθμου, η μείωση του οποίου μπορεί να επιτευχθεί με την επικουρική διεξαγωγή μέρους των υπολογισμών σε γραφικά υποσυστήματα. Επειδή τα ελεύθερα διαθέσιμα εργαλεία για την υλοποίηση ενός τέτοιου στόχου είναι σε πολύ αρχική μορφή και απαιτούν εξειδικευμένες προγραμματιστικές γνώσεις δεν έγινε εφικτή η σημαντική μείωση του χρόνου εκτέλεσης της μεθόδου σε ένα απο τα πιο αντιπροσωπευτικά υπολογιστικά συστήματα που είναι διαθέσιμα σήμερα για αυτού του είδους τις υλοποιήσεις. Σύμφωνα με τα αποτελέσματα των πειραματικών μετρήσεων σφαλμάτων και χρόνου εκτέλεσης της εφαρμογής της αριθμητικής μεθόδου προέκυψαν τα παρακάτω συμπεράσματα

- Η αριθμητική μέθοδος πεπερασμένων στοιχείων Collocation βασισμένη στα Hermite πολυώνυμα βάσης επιλύει με υψηλής τάξης ακρίβεια σφάλματος ελλειπτικά προβλήματα συνοριακών τιμών και παρέχει προσεγγίσεις σε οποιοδήποτε σημείο

του χωρίου.

- Η Τεχνική Πολυπλέγματος επιλύει ικανοποιητικά το Collocation γραμμικό σύστημα διατηρώντας τη τάξη ακρίβειας της μεθόδου των πεπερασμένων στοιχείων.

- Ο υπολογισμός σημαντικού πλήθους συνοριακών αγνώστων με χρήση υψηλής τάξης ακρίβειας προσεγγιστικούς τύπους πριν απο την εφαρμογή της μεθόδου μειώνει το πλήθος των αγνώστων σε $4n_s^2$

- Η επικουρική χρήση γραφικών υποσυστημάτων για τη διεξαγωγή μέρους των επιστημονικών υπολογισμών της μεθόδου δεν μειώνει το υπολογιστικό κόστος χωρίς τη κατασκευή κατάλληλου παράλληλου αλγορίθμου

- Η Τεχνική Πολυπλέγματος είναι γνωστό ότι έχει μη αποδοτική συμπεριφορά σε παράλληλες αρχιτεκτονικές υπολογισμών. Η κύρια αιτία για αυτό είναι η συνεχής μεταβολή του μεγέθους του προβλήματος σε κάθε εναλλαγή του πλέγματος. Το φαινόμενο αυτό εμποδίζει την αποδοτική συμπεριφορά της μεθόδου και στα υπολογιστικά περιβάλλοντα των γραφικών υποσυστημάτων. Τα διαθέσιμα ελεύθερα εργαλεία ανάπτυξης των εφαρμογών δεν είναι σε θέση να αντιμετοπίσουν αυτό το πρόβλημα.

Ειδικότερα από τη μελέτη της συμπεριφοράς του κλασσικού αλγορίθμου της μεθόδου προέκυψε ότι η κατασκευή και αποθήκευση του πίνακα συντελεστών των αγνώστων σε αραιή μορφή δεν είναι κατάλληλη για την αποδοτική διεξαγωγή επιστημονικών υπολογισμών σε γραφικά υποσυστήματα. Ίσως η αρίθμηση αγνώστων και εξισώσεων με ένα διχρωματικό σχήμα βοηθούσε στην απεξάρτηση ομάδων αγνώστων και εξισώσεων έτσι, ώστε να είναι εφικτή η προσέγγιση τους σε ανεξάρτητες διαδικασίες υπολογισμών. Τότε θα γινόταν δυνατή η κατασκευή παράλληλου αλγορίθμου μέσω του οποίου θα μπορούσε να γίνει σταδιακή μεταφορά μέρους των δεδομένων μεταξύ μνήμης κεντρικού επεξεργαστή και γραφικού υποσυστήματος. Η μείωση του επικοινωνιακού κόστους μπορεί να

επιτευχθεί με τη χρήση της τεχνικής της μηδενικής αντιγραφής διανυσμάτων. Η εξέλιξη του λογισμικού ανάπτυξης τέτοιου τύπου εφαρμογών θα ήταν σημαντική βοήθεια προς τους προγραμματιστές.

Παράρτημα Α΄

Κώδικας προγραμματισμού σε Fortran

Α΄.1 Κυρίως πρόγραμμα

```
C ================================================
C |---------------Instructions---------------------|
C |                                                |
C | This program creates the basic matrices for the|
C | one dimensional collocation method and with    |
C | the subroutines krons and kronb finally creates|
C | the matrix C for the two dimensional problem   |
C |                                                |
C | N := the number of the elements for both axis  |
C | M := the size of the final matrices            |
C |________________________________________________|
C
C Date  :: 14/5/2012

       module array
       real*8, dimension(:), allocatable :: k0,k1,k2,c,mbuff,mbuff2,
     +  k0f,k1f,k2f,k0e,k1e,k2e,initb,initu,initl,initr,rhs,u,uc,
```

```
+   r,tmp,a1u,a1b,a3u,a3b,lly,rry,ubx,bbx,uux,dl,wk
    real*8, dimension(:), allocatable :: initbx,initux,initry,
+   initly,hu,x,a1u2,a3u2,a1b2,a3b2,ubx2,clonex,xo,
+   lu1,lu2,lu3,lu4,lu5,lu6,lu7,lu8,lu9
    integer, dimension(:), allocatable :: ju,iu,jdl,idl,ik,jc,ic,
+   levs,jw,ikc,jk,ikf,jkf,jke,ike,jm,im,it,ipiv,ikf2,jkf2
+   ,jlu1,jlu2,jlu3,jlu4,jlu5,jlu6,jlu7,jlu8,jlu9,ilu1,ilu2,ilu3,
+   ilu4,ilu5,ilu6,ilu7,ilu8,ilu9
    end module

    module carray
    real*8, dimension(:), allocatable :: c1,c2,c3,c4,c5,c6,c7,c8,c9
    integer, dimension(:), allocatable :: jc1,jc2,jc3,jc4,jc5,jc6,jc7
+   ,jc8,jc9,ic1,ic2,ic3,ic4,ic5,ic6,ic7,ic8,ic9
    end module

    module sarray
    real*8, dimension(:), allocatable ::u1,u2,u3,u4,u5,u6,u7,u8,u9,
+   dl1,dl2,dl3,dl4,dl5,dl6,dl7,dl8,dl9
    integer, dimension(:), allocatable :: ju1,ju2,ju3,ju4,ju5,
+   ju6,ju7,ju8,ju9,iu1,iu2,iu3,iu4,iu5,iu6,iu7,iu8,iu9,jdl1,jdl2,
+   jdl3,jdl4,jdl5,jdl6,jdl7,jdl8,jdl9,idl1,idl2,idl3,idl4,idl5,
+   idl6,idl7,idl8,idl9
    end module

    module erarray
    real*8, dimension(:), allocatable :: e1,e2,e3,e4,e5,e6,e7,e8,e9,
```

```
     +  r1,r2,r3,r4,r5,r6,r7,r8,r9,zeros
        end module


        program collocation
        use array
        use carray
        use sarray
        use erarray


        real*8 at,bbart,bt,b,bbar,a,bhat,h,x0,x1,dnrm2
        integer flag,counter,info,q,l,pof2,tm,tn,tnz
        integer ku,kl,tmpi,w,z,zz,ldim(9),ldimc(9),cyc
        character*64, funca,funcb,funcc,funcd,funce,funcf
integer iwk,tiwk

real*8 t1,t2,t3,t4,t5,t6,t7,matread,rhsread

c       real*8 fpar(16),uc(n**2)
        integer ipar(16),ierr

         call cpu_time(t1)
lfil=10
call system('clear')
668       print*,'Give the number of the elements'
read*, n
```

```
if (n.le.0) then
print*, 'Collocation can not work with zero or negative elements'
print*, 'Try again'
goto 668
endif
       m=4*n**2
       nz=8*n-4
       iwk=8*nz**2


c       external bcgstab,runrc

       l=pof2(n)-2
       ldim(:)=0
       ldimc(:)=0

       z=n
       q=4*n**2
       do k=1,l
       ldim(k)=z
       ldimc(k)=q
       z=z/2
       q=4*(z)**2
       enddo


       at=2.0d0*sqrt(3.0d0);
```

```
      bbart=sqrt(3.0d0)-1.0d0;
      bt=sqrt(3.0d0)+1.0d0;
      b=(3.0d0+sqrt(3.0d0))/36.0d0;
      bbar=(3.0d0-sqrt(3.0d0))/36.0d0;
      a=(9.0d0+4.0d0*sqrt(3.0d0))/18.0d0;
      bhat=sqrt(3.0d0)/6.0d0;
      ku=2
      kl=2

c      print*, 'Give the number of the discretization of x'
c      read*, n
c
c      if (n.le.0) then
c      print*, 'Wrong N'
c      goto 1
c      endif


      call system('clear')
      print*
      print*, '*********************************************'
      print*
      print*, 'Make your equation by typing the function or 0'
      print*, 'for the existing coefficients and non-existing'
      print*, '  respectively'
      print*
      print*, '   a Uxx + b Uxy + c Uyy + d Ux + e Uy +f U = g '
```

```
      print*
      print*
      print*, '*******************************************'
      print*
c      print*, 'NOTE: use x and y as variables and the fortran'
c      print*, 'symbols i.e. x**2+5*y+3 (**=power)'

call cpu_time(t6)
      print*, 'a='
      read*,funca
      print*, 'b='
      read*,funcb
      print*, 'c='
      read*,funcc
      print*, 'd='
      read*,funcd
      print*, 'e='
      read*, funce
      print*, 'f='
      read*, funcf
call cpu_time(matread)
matread=matread-t6


      x0=0.0d0
      x1=1.0d0
      k=1
```

```
10      continue

        n=ldim(k)
        m=ldimc(k)
        nz=8*ldim(k)-4

        h=(x1-x0)/dble(n)
        allocate(k2(nz))
        allocate(k1(nz))
        allocate(k0(nz))
        allocate(jk(nz))
        allocate(ikc(nz))
        allocate(ik(2*n+1))


        q=0
        do i=1,2*n
        do j=max(i-kl,1),min(ku+i,2*n)

         if (mod(i,2).ne.0) then

         if (i.eq.j) then
         q=q+1
                 k2(q)=-bt
                 k1(q)=bhat
                 k0(q)=b
```

```
            ikc(q)=i
            jk(q)=j
        elseif (i.eq.j-1) then
        q=q+1
            k2(q)=at
            k1(q)=1.0d0
            k0(q)=1.0d0-a
            ikc(q)=i
            jk(q)=j
        elseif (i.eq.j+1) then
        q=q+1
            k2(q)=-at
            k1(q)=-1.0d0
            k0(q)=a
            ikc(q)=i
            jk(q)=j
        endif

    else

        if (i.eq.j) then
        q=q+1
            k2(q)=-at
            k1(q)=1.0d0
            k0(q)=a
            ikc(q)=i
            jk(q)=j
```

```
    elseif (i.eq.j-1) then
    q=q+1
        k2(q)=bt
        k1(q)=bhat
        k0(q)=-b
        ikc(q)=i
        jk(q)=j
    elseif (i.eq.j+1) then
    q=q+1
        k2(q)=bbart
        k1(q)=-bhat
        k0(q)=bbar
        ikc(q)=i
        jk(q)=j
    endif

endif

if ((j.ne.1).and.(j.ne.2*n)) then

    if (mod(j,2).eq.0) then

        if (i.eq.j+2) then
        q=q+1
        k2(q)=at
        k1(q)=-1.0d0
        k0(q)=1.0d0-a
```

```
          ikc(q)=i
          jk(q)=j
          endif

     else

          if (i.eq.j-2) then
          q=q+1
          k2(q)=-bbart
          k1(q)=-bhat
          k0(q)=-bbar
          ikc(q)=i
          jk(q)=j
          endif

     endif
 endif

enddo
enddo


ik(1)=ikc(1)
w=1
do i=1,nz-1
tmpi=ikc(i)
if (tmpi.ne.ikc(i+1)) then
```

```
        w=w+1
        ik(w)=i+1
endif
enddo
ik(w+1)=ik(1)+nz
deallocate(ikc)

k2(nz)=bt
        k2(nz-3)=-bbart

        k1(nz)=bhat
        k1(nz-3)=-bhat

        k0(nz)=-b
        k0(nz-3)=-bbar

        allocate(k2f(8*n))
        allocate(k1f(8*n))
        allocate(k0f(8*n))
        allocate(jkf(8*n))
        allocate(ikf(2*n+1))

        q=1
        do i=1,2*n,2

        k2f(q)=-at
        k2f(q+1)=-bt
```

```
k2f(q+2)=at
k2f(q+3)=-bbart
k2f(q+4)=at
k2f(q+5)=bbart
k2f(q+6)=-at
k2f(q+7)=bt

k0f(q)=a
k0f(q+1)=b
k0f(q+2)=1-a
k0f(q+3)=-bbar
k0f(q+4)=1-a
k0f(q+5)=bbar
k0f(q+6)=a
k0f(q+7)=-b

k1f(q)=-1
k1f(q+1)=bhat
k1f(q+2)=1
k1f(q+3)=-bhat
k1f(q+4)=-1
k1f(q+5)=-bhat
k1f(q+6)=1
k1f(q+7)=bhat

jkf(q)=i
jkf(q+1)=i+1
```

```
jkf(q+2)=i+2
jkf(q+3)=i+3
jkf(q+4)=i
jkf(q+5)=i+1
jkf(q+6)=i+2
jkf(q+7)=i+3
q=q+8

enddo
do i=1,2*n
ikf(i)=(i-1)*4+1
enddo
ikf(2*n+1)=2*n*4+1

allocate(k2e(4))
allocate(k1e(4))
allocate(k0e(4))
allocate(jke(4))
allocate(ike(2*n+1))



k0e(1)=a
k0e(2)=1.0d0-a
k0e(3)=1.0d0-a
k0e(4)=a
k1e(1)=-1.0d0
k1e(2)=-1.0d0
```

```
k1e(3)=1.0d0
k1e(4)=1.0d0
k2e(1)=-at
k2e(2)=at
k2e(3)=at
k2e(4)=-at

jke(1)=1
jke(2)=1
jke(3)=2*n+1
jke(4)=2*n+1

ike(1)=1
ike(2)=2
do i=3,2*n-2
ike(i)=3
enddo
ike(2*n-1)=3
ike(2*n)=4
ike(2*n+1)=5




allocate(C(nz*nz))
allocate(a1u(8*n))
allocate(a3u(8*n))
```

```
allocate(a1b(8*n))
allocate(a3b(8*n))
allocate(ubx(4*nz))
allocate(jc(nz*nz))
allocate(ic(m+1))
allocate(mbuff(m))
allocate(mbuff2(nz*nz))
allocate(it(2))
allocate(jm(nz*4))
allocate(im(m+1))
allocate(u((2*n-1)*nz))
allocate(ju((2*n-1)*nz))
allocate(iu(m+1))
allocate(dl(nz**2-(2*n-1)*nz))
allocate(jdl(nz**2-(2*n-1)*nz))
allocate(idl(m+1))


do i=1,nz*nz
c(i)=0.0d0
enddo
do i=1,m+1
ic(i)=0
enddo

do i=1,8*n
a1u(i)=0.0d0
```

```
      a3u(i)=0.0d0
      a3b(i)=0.0d0
      a1b(i)=0.0d0
      enddo
      do i=1,2
      it(i)=i
      enddo

      do i=1,nz*4
      ubX(i)=0.0d0
      enddo


      flag=1
      call getfunctionbmg(n,n,mbuff,flag,funca)
      if (flag.eq.1) then
      call krons(2*n,2*n,k2,jk,ik,2*n,2*n,k0,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,1.0d0,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,-at,1,it,2*n,2*n+2,k0f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-2*n+1),mbuff2,1.0d0,a1b,
     +  jkf,ikf)
      call krons(1,1,at,1,it,2*n,2*n+2,k0f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-4*n+1),mbuff2,1.0d0,a3b,
     +  jkf,ikf)
```

```
      call krons(2*n,2*n,k2,jk,ik,2*n,2*n+2,k0e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,1.0d0,mbuff,mbuff2,1.0d0,ubX,jm,im)

      endif

      flag=1
      call getfunctionbmg(n,n,mbuff,flag,funcb)
      if (flag.eq.1) then
      call krons(2*n,2*n,k1,jk,ik,2*n,2*n,k1,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,1.0d0,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,-1.0d0,1,it,2*n,2*n+2,k1f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call krons(1,1,1.0d0,1,it,2*n,2*n+2,k1f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-2*n+1),mbuff2,1.0d0,a1b,
     +  jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-4*n+1),mbuff2,1.0d0,a3b,
     +  jkf,ikf)

      call krons(2*n,2*n,k1,jk,ik,2*n,2*n+2,k1e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,1.0d0,mbuff,mbuff2,1.0d0,ubX,jm,im)

      endif

      flag=1
```

```
      call getfunctionbmg(n,n,mbuff,flag,funcc)
      if (flag.eq.1) then
      call krons(2*n,2*n,k0,jk,ik,2*n,2*n,k2,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,1.0d0,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,a,1,it,2*n,2*n+2,k2f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-2*n+1),mbuff2,1.0d0,a1b,
     +  jkf,ikf)
      call krons(1,1,1.0d0-a,1,it,2*n,2*n+2,k2f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call dsmsm(2*n,2*n+2,1.0d0,mbuff(m-4*n+1),mbuff2,1.0d0,a3b,
     +  jkf,ikf)

      call krons(2*n,2*n,k0,jk,ik,2*n,2*n+2,k2e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,1.0d0,mbuff,mbuff2,1.0d0,ubX,jm,im)

      endif

      flag=1
      call getfunctionbmg(n,n,mbuff,flag,funcd)
      if (flag.eq.1) then
      call krons(2*n,2*n,k1,jk,ik,2*n,2*n,k0,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,h,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,-1.0d0,1,it,2*n,2*n+2,k0,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
```

```
      call dsmsm(2*n,2*n+2,h,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call krons(1,1,1.0d0,1,it,2*n,2*n+2,k0f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff(m-2*n+1),mbuff2,1.0d0,a1b,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff(m-4*n+1),mbuff2,1.0d0,a3b,jkf,ikf)

      call krons(2*n,2*n,k1,jk,ik,2*n,2*n+2,k0e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,h,mbuff,mbuff2,1.0d0,ubX,jm,im)

      endif
C This matrix multiplied with the scalar h cause of the common coefficient

      flag=1
      call getfunctionbmg(n,n,mbuff,flag,funce)
      if (flag.eq.1) then
      call krons(2*n,2*n,k0,jk,ik,2*n,2*n,k1,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,h,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,a,1,it,2*n,2*n+2,k1f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff(m-2*n+1),mbuff2,1.0d0,a1b,jkf,ikf)
      call krons(1,1,1.0d0-a,1,it,2*n,2*n+2,k1f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call dsmsm(2*n,2*n+2,h,mbuff(m-4*n+1),mbuff2,1.0d0,a3b,jkf,ikf)

      call krons(2*n,2*n,k0,jk,ik,2*n,2*n+2,k1e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,h,mbuff,mbuff2,1.0d0,ubX,jm,im)
```

```
      endif
C This matrix multiplied with the scalar h cause of the common coefficient

      flag=1
      call getfunctionbmg(n,n,mbuff,flag,funcf)
      if (flag.eq.1) then
      call krons(2*n,2*n,k0,jk,ik,2*n,2*n,k0,jk,ik,mbuff2,jc,ic)
      call dsmsm(m,m,h**2,mbuff,mbuff2,1.0d0,c,jc,ic)

      call krons(1,1,a,1,it,2*n,2*n+2,k0f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h**2,mbuff,mbuff2,1.0d0,a1u,jkf,ikf)
      call dsmsm(2*n,2*n+2,h**2,mbuff(m-2*n+1),mbuff2,1.0d0,
     +  a1b,jkf,ikf)
      call krons(1,1,1.0d0-a,1,it,2*n,2*n+2,k0f,jkf,ikf,mbuff2,jkf,ikf)
      call dsmsm(2*n,2*n+2,h**2,mbuff(2*n+1),mbuff2,1.0d0,a3u,jkf,ikf)
      call dsmsm(2*n,2*n+2,h**2,mbuff(m-4*n+1),mbuff2,1.0d0,
     +  a3b,jkf,ikf)

      call krons(2*n,2*n,k0,jk,ik,2*n,2*n+2,k0e,jke,ike,mbuff2,jm,im)
      call dsmsm(m,m+4*n,h**2,mbuff,mbuff2,1.0d0,ubX,jm,im)

      endif
C This matrix multiplied with the scalar h**2 cause of the common coefficie


C split the matrix into lower triangular dl and upper u
```

```
      q=0
      w=0
      z=0
      zz=1
      iu(1)=1
      idl(1)=1
      do i=1,2*n
       do kk=1,2*n
        do j=ik(i),ik(i+1)-1
        do ll=ik(kk),ik(kk+1)-1
        z=z+1
        if (((i+1.eq.jk(j)).or.(i+2.eq.jk(j)))
     + .and.(mod(i,2).ne.0)) then
        q=q+1
        U(q)=C(z)
        ju(q)=jc(z)
        else
        w=w+1
        DL(w)=C(z)
        jdl(w)=jc(z)
        endif
        enddo
        enddo
        zz=zz+1
        iu(zz)=q+1
        idl(zz)=w+1
```

```
 enddo
enddo



if (k.eq.1) then
allocate(c1(nz*nz))
allocate(jc1(nz*nz))
allocate(ic1(m+1))
allocate(a1u2(8*n))
allocate(a3u2(8*n))
allocate(a1b2(8*n))
allocate(a3b2(8*n))
allocate(ubx2(4*nz))
allocate(jkf2(8*n))
allocate(ikf2(2*n+1))
allocate(u1((2*n-1)*nz))
allocate(ju1((2*n-1)*nz))
allocate(iu1(m+1))
allocate(dl1(nz**2-(2*n-1)*nz))
allocate(jdl1(nz**2-(2*n-1)*nz))
allocate(idl1(m+1))



u1(:)=u(:)
ju1(:)=ju(:)
iu1(:)=iu(:)
dl1(:)=dl(:)
```

```
jdl1(:)=jdl(:)
idl1(:)=idl(:)
c1(:)=c(:)
jc1(:)=jc(:)
ic1(:)=ic(:)
a1u2(:)=a1u(:)
a3u2(:)=a3u(:)
a1b2(:)=a1b(:)
a3b2(:)=a3b(:)
ubx2(:)=ubx(:)
jkf2(:)=jkf(:)
ikf2(:)=ikf(:)

elseif (k.eq.2) then
allocate(c2(nz*nz))
allocate(jc2(nz*nz))
allocate(ic2(m+1))
allocate(u2((2*n-1)*nz))
allocate(ju2((2*n-1)*nz))
allocate(iu2(m+1))
allocate(dl2(nz**2-(2*n-1)*nz))
allocate(jdl2(nz**2-(2*n-1)*nz))
allocate(idl2(m+1))



u2(:)=u(:)
ju2(:)=ju(:)
```

```
iu2(:)=iu(:)
dl2(:)=dl(:)
jdl2(:)=jdl(:)
idl2(:)=idl(:)

c2(:)=c(:)
jc2(:)=jc(:)
ic2(:)=ic(:)

elseif (k.eq.3) then
allocate(c3(nz*nz))
allocate(jc3(nz*nz))
allocate(ic3(m+1))
allocate(u3((2*n-1)*nz))
allocate(ju3((2*n-1)*nz))
allocate(iu3(m+1))
allocate(dl3(nz**2-(2*n-1)*nz))
allocate(jdl3(nz**2-(2*n-1)*nz))
allocate(idl3(m+1))



u3(:)=u(:)
ju3(:)=ju(:)
iu3(:)=iu(:)
dl3(:)=dl(:)
jdl3(:)=jdl(:)
idl3(:)=idl(:)
```

```
c3(:)=c(:)
jc3(:)=jc(:)
ic3(:)=ic(:)

elseif (k.eq.4) then
allocate(c4(nz*nz))
allocate(jc4(nz*nz))
allocate(ic4(m+1))
allocate(u4((2*n-1)*nz))
allocate(ju4((2*n-1)*nz))
allocate(iu4(m+1))
allocate(dl4(nz**2-(2*n-1)*nz))
allocate(jdl4(nz**2-(2*n-1)*nz))
allocate(idl4(m+1))


u4(:)=u(:)
ju4(:)=ju(:)
iu4(:)=iu(:)
dl4(:)=dl(:)
jdl4(:)=jdl(:)
idl4(:)=idl(:)

c4(:)=c(:)
jc4(:)=jc(:)
ic4(:)=ic(:)
```

```
elseif (k.eq.5) then
allocate(c5(nz*nz))
allocate(jc5(nz*nz))
allocate(ic5(m+1))
allocate(u5((2*n-1)*nz))
allocate(ju5((2*n-1)*nz))
allocate(iu5(m+1))
allocate(dl5(nz**2-(2*n-1)*nz))
allocate(jdl5(nz**2-(2*n-1)*nz))
allocate(idl5(m+1))


u5(:)=u(:)
ju5(:)=ju(:)
iu5(:)=iu(:)
dl5(:)=dl(:)
jdl5(:)=jdl(:)
idl5(:)=idl(:)

c5(:)=c(:)
jc5(:)=jc(:)
ic5(:)=ic(:)

elseif (k.eq.6) then
allocate(c6(nz*nz))
allocate(jc6(nz*nz))
```

```
allocate(ic6(m+1))
allocate(u6((2*n-1)*nz))
allocate(ju6((2*n-1)*nz))
allocate(iu6(m+1))
allocate(dl6(nz**2-(2*n-1)*nz))
allocate(jdl6(nz**2-(2*n-1)*nz))
allocate(idl6(m+1))


u6(:)=u(:)
ju6(:)=ju(:)
iu6(:)=iu(:)
dl6(:)=dl(:)
jdl6(:)=jdl(:)
idl6(:)=idl(:)

c6(:)=c(:)
jc6(:)=jc(:)
ic6(:)=ic(:)

elseif (k.eq.7) then
allocate(c7(nz*nz))
allocate(jc7(nz*nz))
allocate(ic7(m+1))
allocate(u7((2*n-1)*nz))
allocate(ju7((2*n-1)*nz))
allocate(iu7(m+1))
```

```
allocate(dl7(nz**2-(2*n-1)*nz))
allocate(jdl7(nz**2-(2*n-1)*nz))
allocate(idl7(m+1))



u7(:)=u(:)
ju7(:)=ju(:)
iu7(:)=iu(:)
dl7(:)=dl(:)
jdl7(:)=jdl(:)
idl7(:)=idl(:)

c7(:)=c(:)
jc7(:)=jc(:)
ic7(:)=ic(:)

elseif (k.eq.8) then
allocate(c8(nz*nz))
allocate(jc8(nz*nz))
allocate(ic8(m+1))
allocate(u8((2*n-1)*nz))
allocate(ju8((2*n-1)*nz))
allocate(iu8(m+1))
allocate(dl8(nz**2-(2*n-1)*nz))
allocate(jdl8(nz**2-(2*n-1)*nz))
allocate(idl8(m+1))
```

```
u8(:)=u(:)
ju8(:)=ju(:)
iu8(:)=iu(:)
dl8(:)=dl(:)
jdl8(:)=jdl(:)
idl8(:)=idl(:)

c8(:)=c(:)
jc8(:)=jc(:)
ic8(:)=ic(:)

elseif (k.eq.9) then
allocate(c9(nz*nz))
allocate(jc9(nz*nz))
allocate(ic9(m+1))
allocate(u9((2*n-1)*nz))
allocate(ju9((2*n-1)*nz))
allocate(iu9(m+1))
allocate(dl9(nz**2-(2*n-1)*nz))
allocate(jdl9(nz**2-(2*n-1)*nz))
allocate(idl9(m+1))


u9(:)=u(:)
ju9(:)=ju(:)
iu9(:)=iu(:)
```

```
dl9(:)=dl(:)
jdl9(:)=jdl(:)
idl9(:)=idl(:)

c9(:)=c(:)
jc9(:)=jc(:)
ic9(:)=ic(:)



endif



deallocate(k2)
deallocate(k1)
deallocate(k0)
deallocate(k2f)
deallocate(k1f)
deallocate(k0f)
deallocate(k2e)
deallocate(k1e)
deallocate(k0e)
deallocate(mbuff)
deallocate(mbuff2)
deallocate(jk)
deallocate(ik)
deallocate(it)
```

```
deallocate(jke)
deallocate(ike)
deallocate(jm)
deallocate(im)
deallocate(c)
deallocate(jc)
deallocate(ic)
deallocate(a1u)
deallocate(a1b)
deallocate(a3u)
deallocate(a3b)
deallocate(jkf)
deallocate(ikf)
deallocate(ubx)
deallocate(u)
deallocate(ju)
deallocate(iu)
deallocate(dl)
deallocate(jdl)
deallocate(idl)


if (k.lt.l) then
k=k+1
goto 10
endif
```

```
call cpu_time(t3)



C This is a testing procedure
c       allocate(r1(ldimc(1)))
c       allocate(r2(ldimc(1)))
c       allocate(r3(ldimc(1)))
c       r1(:)=1.0d0

c       call amux(ldimc(1),r1,r2,c1,jc1,ic1)
c       print*, dnrm2(ldimc(1),r2,1)

c       call amux(ldimc(1),r1,r2,u1,ju1,iu1)
c       call amux(ldimc(1),r1,r3,dl1,jdl1,idl1)
c       call daxpy(ldimc(1),1.0d0,r2,1,r3,1)
c       print*, dnrm2(ldimc(1),r3,1)



       n=ldim(1)
       m=ldimc(1)
       nz=8*n-4
       allocate(rhs(m))

       flag=1
       print*, 'g='
```

```
       call getfunctionb2(n,n,rhs,flag,t6)

rhsread=t6
C The vector rhs multiplied with the scalar h**2




       allocate(initb(n+1))
       allocate(initu(n+1))
       allocate(initr(n+1))
       allocate(initl(n+1))
       allocate(initbx(n+1))
       allocate(initux(n+1))
       allocate(initry(n+1))
       allocate(initly(n+1))




       print*
       print*, '*******************************************'
       print*
       print*, '      Type your initial conditions'
       print*
       print*, '   U(0,x) , U(1,x) , U(0,y) & U(1,y) '
       print*
       print*
       print*, '*******************************************'
```

```
       print*



       flag=1
       print*, 'U(x,0)='
       call getfunctionxy(n,initb,flag,1,t6)
rhsread=rhsread+t6

       flag=1
       print*, 'U(x,1)='
       call getfunctionxy(n,initu,flag,2,t6)
rhsread=rhsread+t6

       flag=1
       print*, 'U(0,y)='
       call getfunctionxy(n,initl,flag,3,t6)
rhsread=rhsread+t6

       flag=1
       print*, 'U(1,y)='
       call getfunctionxy(n,initr,flag,4,t6)
rhsread=rhsread+t6

       do i=3,n-1
       initbX(i)=(initb(i-2)-8.0d0*initb(i-1)
     +  +8.0d0*initb(i+1)-initb(i+2))/(12.0d0*h)
       enddo
```

```
      initbX(1)=(-25.0d0*initb(1)+48.0d0*initb(2)-36.0d0*initb(3)
     +  +16.0d0*initb(4)-3.0d0*initb(5))/(12.0d0*h)
      initbX(2)=(-25.0d0*initb(2)+48.0d0*initb(3)-36.0d0*initb(4)
     +  +16.0d0*initb(5)-3.0d0*initb(6))/(12.0d0*h)
      initbX(n+1)=(25.0d0*initb(n+1)-48.0d0*initb(n)+36.0d0*
     +  initb(n-1)-16.0d0*initb(n-2)+3.0d0*initb(n-3))/(12.0d0*h)
      initbX(n)=(25.0d0*initb(n)-48.0d0*initb(n-1)+36.0d0*
     +  initb(n-2)-16.0d0*initb(n-3)+3.0d0*initb(n-4))/(12.0d0*h)

      do i=3,n-1
      inituX(i)=(initu(i-2)-8.0d0*initu(i-1)
     +  +8.0d0*initu(i+1)-initu(i+2))/(12.0d0*h)
      enddo

      inituX(1)=(-25.0d0*initu(1)+48.0d0*initu(2)-36.0d0*initu(3)
     +  +16.0d0*initu(4)-3.0d0*initu(5))/(12.0d0*h)
      inituX(2)=(-25.0d0*initu(2)+48.0d0*initu(3)-36.0d0*initu(4)
     +  +16.0d0*initu(5)-3.0d0*initu(6))/(12.0d0*h)
      inituX(n+1)=(25.0d0*initu(n+1)-48.0d0*initu(n)+36.0d0*
     +  initu(n-1)-16.0d0*initu(n-2)+3.0d0*initu(n-3))/(12.0d0*h)
      inituX(n)=(25.0d0*initu(n)-48.0d0*initu(n-1)+36.0d0*
     +  initu(n-2)-16.0d0*initu(n-3)+3.0d0*initu(n-4))/(12.0d0*h)

      do i=3,n-1
      initlY(i)=(initl(i-2)-8.0d0*initl(i-1)
     +  +8.0d0*initl(i+1)-initl(i+2))/(12.0d0*h)
```

```
      enddo

      initlY(1)=(-25.0d0*initl(1)+48.0d0*initl(2)-36.0d0*initl(3)
     +  +16.0d0*initl(4)-3.0d0*initl(5))/(12.0d0*h)
      initlY(2)=(-25.0d0*initl(2)+48.0d0*initl(3)-36.0d0*initl(4)
     +  +16.0d0*initl(5)-3.0d0*initl(6))/(12.0d0*h)
      initlY(n+1)=(25.0d0*initl(n+1)-48.0d0*initl(n)+36.0d0*
     +  initl(n-1)-16.0d0*initl(n-2)+3.0d0*initl(n-3))/(12.0d0*h)
      initlY(n)=(25.0d0*initl(n)-48.0d0*initl(n-1)+36.0d0*
     +  initl(n-2)-16.0d0*initl(n-3)+3.0d0*initl(n-4))/(12.0d0*h)

      do i=3,n-1
      initrY(i)=(initr(i-2)-8.0d0*initr(i-1)
     +  +8.0d0*initr(i+1)-initr(i+2))/(12.0d0*h)
      enddo

      initrY(1)=(-25.0d0*initr(1)+48.0d0*initr(2)-36.0d0*initr(3)
     +  +16.0d0*initr(4)-3.0d0*initr(5))/(12.0d0*h)
      initrY(2)=(-25.0d0*initr(2)+48.0d0*initr(3)-36.0d0*initr(4)
     +  +16.0d0*initr(5)-3.0d0*initr(6))/(12.0d0*h)
      initrY(n+1)=(25.0d0*initr(n+1)-48.0d0*initr(n)+36.0d0*
     +  initr(n-1)-16.0d0*initr(n-2)+3.0d0*initr(n-3))/(12.0d0*h)
      initrY(n)=(25.0d0*initr(n)-48.0d0*initr(n-1)+36.0d0*
     +  initr(n-2)-16.0d0*initr(n-3)+3.0d0*initr(n-4))/(12.0d0*h)


      allocate(lly(2*n+2))
```

```
allocate(rry(2*n+2))

q=0
do i=1,n+1
q=q+1
lly(q)=initl(i)
q=q+1
lly(q)=h*initly(i)
enddo
q=0
do i=1,n+1
q=q+1
rry(q)=initr(i)
q=q+1
rry(q)=h*initry(i)
enddo

allocate(tmp(2*n))


call amux(2*n,lly,tmp,a1u2,jkf2,ikf2)
call daxpy(2*n,-1.0d0,tmp,1,rhs,1)
call amux(2*n,lly,tmp,a3u2,jkf2,ikf2)
call daxpy(2*n,-1.0d0,tmp,1,rhs(2*n+1),1)
call amux(2*n,rry,tmp,a1b2,jkf2,ikf2)
call daxpy(2*n,-1.0d0,tmp,1,rhs(m-2*n+1),1)
call amux(2*n,rry,tmp,a3b2,jkf2,ikf2)
```

```
call daxpy(2*n,-1.0d0,tmp,1,rhs(m-4*n+1),1)

deallocate(lly)
deallocate(rry)
deallocate(tmp)
deallocate(a1u2)
deallocate(a3u2)
deallocate(a1b2)
deallocate(a3b2)
deallocate(ikf2)
deallocate(jkf2)



allocate(uux(2*n))
allocate(bbx(2*n))

q=0
do i=1,n
q=q+1
bbX(q)=h*initbX(i)
if (i.ne.n) then
q=q+1
bbx(q)=initb(i+1)
else
q=q+1
bbx(q)=h*initbX(i+1)
endif
```

```
enddo



q=0
do i=1,n
q=q+1
uuX(q)=h*inituX(i)
if (i.ne.n) then
q=q+1
uux(q)=initu(i+1)
else
q=q+1
uuX(q)=h*inituX(i+1)
endif
enddo



do i=1,3
rhs(1)=rhs(1)-bbx(i)*ubx2(i)
rhs(2)=rhs(2)-bbx(i)*ubx2(3+i)
rhs(2*n-1)=rhs(2*n-1)-uux(i)*ubx2(6+i)
rhs(2*n)=rhs(2*n)-uux(i)*ubx2(9+i)
rhs(2*n+1)=rhs(2*n+1)-bbx(i)*ubx2(12+i)
rhs(2*n+2)=rhs(2*n+2)-bbx(i)*ubx2(15+i)
rhs(4*n-1)=rhs(4*n-1)-uux(i)*ubx2(18+i)
rhs(4*n)=rhs(4*n)-uux(i)*ubx2(21+i)
enddo
```

```
w=24
q=2
do k=2,n-1
z=1
do i=q,q+3
rhs((k-1)*4*n+1)=rhs((k-1)*4*n+1)-bbx(i)*ubx2(w+z)
rhs((k-1)*4*n+2)=rhs((k-1)*4*n+2)-bbx(i)*ubx2(w+4+z)
rhs((k-1)*4*n+2*n-1)=rhs((k-1)*4*n+2*n-1)-uux(i)*ubx2(w+8+z)
rhs((k-1)*4*n+2*n)=rhs((k-1)*4*n+2*n)-uux(i)*ubx2(w+12+z)
rhs((k-1)*4*n+2*n+1)=rhs((k-1)*4*n+2*n+1)-bbx(i)*ubx2(w+16+z)
rhs((k-1)*4*n+2*n+2)=rhs((k-1)*4*n+2*n+2)-bbx(i)*ubx2(w+20+z)
rhs((k-1)*4*n+4*n-1)=rhs((k-1)*4*n+4*n-1)-uux(i)*ubx2(w+24+z)
rhs((k-1)*4*n+4*n)=rhs((k-1)*4*n+4*n)-uux(i)*ubx2(w+28+z)
z=z+1
enddo
w=w+32
q=q+2
enddo

z=1
do i=2*n-2,2*n
rhs((n-1)*4*n+1)=rhs((n-1)*4*n+1)-bbx(i)*ubx2(w+z)
rhs((n-1)*4*n+2)=rhs((n-1)*4*n+2)-bbx(i)*ubx2(w+3+z)
rhs((n-1)*4*n+2*n-1)=rhs((n-1)*4*n+2*n-1)-uux(i)*ubx2(w+6+z)
rhs((n-1)*4*n+2*n)=rhs((n-1)*4*n+2*n)-uux(i)*ubx2(w+9+z)
rhs((n-1)*4*n+2*n+1)=rhs((n-1)*4*n+2*n+1)-bbx(i)*ubx2(w+12+z)
```

```
        rhs((n-1)*4*n+2*n+2)=rhs((n-1)*4*n+2*n+2)-bbx(i)*ubx2(w+15+z)
        rhs((n-1)*4*n+4*n-1)=rhs((n-1)*4*n+4*n-1)-uux(i)*ubx2(w+18+z)
        rhs((n-1)*4*n+4*n)=rhs((n-1)*4*n+4*n)-uux(i)*ubx2(w+21+z)
        z=z+1
        enddo

        deallocate(bbx)
        deallocate(uux)
        deallocate(ubx2)

call cpu_time(t4)

        allocate(levs(iwk))
        allocate(jw(3*m))
        allocate(wk(m))
        allocate(x(m))
        allocate(tmp(m))



print*
print*, 'ILUK starting. . .'
        do k=1,l
        tnz=8*ldim(k)-4
        tiwk=8*tnz**2
        m=ldimc(k)
print*, 'LEVEL ::', k ,'(',ldim(k),')'
        if (k.eq.1) then
```

```
      allocate(lu1(tiwk))
      allocate(jlu1(tiwk))
      allocate(ilu1(tiwk))
      call iluk(m,dl1,jdl1,idl1,lfil,lu1,jlu1,ilu1,levs,tiwk,
     +  wk,jw,ierr)
      elseif (k.eq.2) then
      allocate(lu2(tiwk))
      allocate(jlu2(tiwk))
      allocate(ilu2(tiwk))
      call iluk(m,dl2,jdl2,idl2,lfil,lu2,jlu2,ilu2,levs,tiwk,
     +  wk,jw,ierr)
      elseif (k.eq.3) then
      allocate(lu3(tiwk))
      allocate(jlu3(tiwk))
      allocate(ilu3(tiwk))
      call iluk(m,dl3,jdl3,idl3,lfil,lu3,jlu3,ilu3,levs,tiwk,
     +  wk,jw,ierr)
      elseif (k.eq.4) then
      allocate(lu4(tiwk))
      allocate(jlu4(tiwk))
      allocate(ilu4(tiwk))
      call iluk(m,dl4,jdl4,idl4,lfil,lu4,jlu4,ilu4,levs,tiwk,
     +  wk,jw,ierr)
      elseif (k.eq.5) then
      allocate(lu5(tiwk))
      allocate(jlu5(tiwk))
      allocate(ilu5(tiwk))
```

```
  call iluk(m,dl5,jdl5,idl5,lfil,lu5,jlu5,ilu5,levs,tiwk,
+  wk,jw,ierr)
  elseif (k.eq.6) then
  allocate(lu6(tiwk))
  allocate(jlu6(tiwk))
 allocate(ilu6(tiwk))
 call iluk(m,dl6,jdl6,idl6,lfil,lu6,jlu6,ilu6,levs,tiwk,
+  wk,jw,ierr)
  elseif (k.eq.7) then
  allocate(lu7(tiwk))
  allocate(jlu7(tiwk))
  allocate(ilu7(tiwk))
  call iluk(m,dl7,jdl7,idl7,lfil,lu7,jlu7,ilu7,levs,tiwk,
+  wk,jw,ierr)
  elseif (k.eq.8) then
  allocate(lu8(tiwk))
  allocate(jlu8(tiwk))
  allocate(ilu8(tiwk))
  call iluk(m,dl8,jdl8,idl8,lfil,lu8,jlu8,ilu8,levs,tiwk,
+  wk,jw,ierr)
  elseif (k.eq.9) then
  allocate(lu9(tiwk))
  allocate(jlu9(tiwk))
  allocate(ilu9(tiwk))
  call iluk(m,dl9,jdl9,idl9,lfil,lu9,jlu9,ilu9,levs,tiwk,
+  wk,jw,ierr)
  endif
```

```
if (ierr.ne.0) then
print*, '*------------------------*'
print*, 'ILUK failure in',k,'level'
print*, 'ERROR FLAG =',ierr
print*, 'Shuting down'
print*, '*------------------------*'
stop
endif
       enddo
print*, 'ILUK finished'
print*

call cpu_time(t5)



C-----MULTIGRID----
print*, 'MULTIGRID starting. . .'
       cyc=0

       x(:)=rhs(:)

20      continue
       cyc=cyc+1

        do k=1,l
        tn=ldim(k)
        tm=ldimc(k)
```

```
          if (k.ne.l) then
          iter=2
          else
          iter=10000
          endif
         if (k.eq.1) then
if (cyc.eq.1) then
       call gs(tm,iter,c1,jc1,ic1,u1,ju1,iu1,lu1,jlu1,ilu1,rhs,x,x)
endif
       if (cyc.eq.1) then
       allocate(r1(tm))
       allocate(r2(ldimc(2)))
       endif
       call amux(tm,x,r1,c1,jc1,ic1)
       call dscal(tm,-1.0d0,r1,1)
       call daxpy(tm,1.0d0,rhs,1,r1,1)
       call restriction(tn,r1,r2)
       elseif (k.eq.2) then
       if (cyc.eq.1) then
       allocate(e2(tm))
       allocate(zeros(tm))
       endif
       zeros(:)=0.0d0
       call gs(tm,iter,c2,jc2,ic2,u2,ju2,iu2,lu2,jlu2,ilu2,r2,e2,zeros)
       if (k.eq.l) goto 40
       call amux(tm,e2,tmp,c2,jc2,ic2)
       call dscal(tm,-1.0d0,tmp,1)
```

```
 call daxpy(tm,1.0d0,r2,1,tmp,1)
 if (cyc.eq.1) allocate(r3(ldimc(3)))
 call restriction(tn,tmp,r3)
 elseif (k.eq.3) then
 if (cyc.eq.1) allocate(e3(tm))
 call gs(tm,iter,c3,jc3,ic3,u3,ju3,iu3,lu3,jlu3,ilu3,r3,e3,zeros)
 if (k.eq.l) goto 40
 call amux(tm,e3,tmp,c3,jc3,ic3)
 call dscal(tm,-1.0d0,tmp,1)
 call daxpy(tm,1.0d0,r3,1,tmp,1)
 if (cyc.eq.1) allocate(r4(ldimc(4)))
 call restriction(tn,tmp,r4)
 elseif (k.eq.4) then

if (cyc.eq.1) allocate(e4(tm))
 call gs(tm,iter,c4,jc4,ic4,u4,ju4,iu4,lu4,jlu4,ilu4,r4,e4,zeros)
 if (k.eq.l) goto 40
 call amux(tm,e4,tmp,c4,jc4,ic4)
 call dscal(tm,-1.0d0,tmp,1)
 call daxpy(tm,1.0d0,r4,1,tmp,1)
 if (cyc.eq.1) allocate(r5(ldimc(5)))
 call restriction(tn,tmp,r5)
 elseif (k.eq.5) then
 if (cyc.eq.1) allocate(e5(tm))
 call gs(tm,iter,c5,jc5,ic5,u5,ju5,iu5,lu5,jlu5,ilu5,r5,e5,zeros)
 if (k.eq.l) goto 40
 call amux(tm,e5,tmp,c5,jc5,ic5)
```

```
call dscal(tm,-1.0d0,tmp,1)
call daxpy(tm,1.0d0,r5,1,tmp,1)
if (cyc.eq.1) allocate(r6(ldimc(6)))
call restriction(tn,tmp,r6)
elseif (k.eq.6) then
if (cyc.eq.1) allocate(e6(tm))
call gs(tm,iter,c6,jc6,ic6,u6,ju6,iu6,lu6,jlu6,ilu6,r6,e6,zeros)
if (k.eq.l) goto 40
call amux(tm,e6,tmp,c6,jc6,ic6)
call dscal(tm,-1.0d0,tmp,1)
call daxpy(tm,1.0d0,r6,1,tmp,1)
if (cyc.eq.1) allocate(r7(ldimc(7)))
call restriction(tn,tmp,r7)
elseif (k.eq.7) then
if (cyc.eq.1) allocate(e7(tm))
call gs(tm,iter,c7,jc7,ic7,u7,ju7,iu7,lu7,jlu7,ilu7,r7,e7,zeros)
if (k.eq.l) goto 40
call amux(tm,e7,tmp,c7,jc7,ic7)
call dscal(tm,-1.0d0,tmp,1)
call daxpy(tm,1.0d0,r7,1,tmp,1)
if (cyc.eq.1) allocate(r8(ldimc(8)))
call restriction(tn,tmp,r8)
elseif (k.eq.8) then
if (cyc.eq.1) allocate(e8(tm))
call gs(tm,iter,c8,jc8,ic8,u8,ju8,iu8,lu8,jlu8,ilu8,r8,e8,zeros)
if (k.eq.l) goto 40
call amux(tm,e8,tmp,c8,jc8,ic8)
```

```
        call dscal(tm,-1.0d0,tmp,1)
        call daxpy(tm,1.0d0,r8,1,tmp,1)
        if (cyc.eq.1) allocate(r9(ldimc(9)))
        call restriction(tn,tmp,r9)
        elseif (k.eq.9) then
        if (cyc.eq.1) allocate(e9(tm))
        call gs(tm,iter,c9,jc9,ic9,u9,ju9,iu9,lu9,jlu9,ilu9,r9,e9,zeros)
        endif
        enddo
40       continue



        do k=l-1,1,-1

        tn=ldim(k)
        tm=ldimc(k)
        iter=2
        if (k.eq.8) then
        call prolongation(ldim(9),e9,tmp)
        call daxpy(tm,1.0d0,tmp,1,e8,1)
        call gs(tm,iter,c8,jc8,ic8,u8,ju8,iu8,lu8,jlu8,ilu8,r8,e8,e8)
        elseif (k.eq.7) then
        call prolongation(ldim(8),e8,tmp)
        call daxpy(tm,1.0d0,tmp,1,e7,1)
        call gs(tm,iter,c7,jc7,ic7,u7,ju7,iu7,lu7,jlu7,ilu7,r7,e7,e7)
        elseif (k.eq.6) then
```

```
call prolongation(ldim(7),e7,tmp)
call daxpy(tm,1.0d0,tmp,1,e6,1)
call gs(tm,iter,c6,jc6,ic6,u6,ju6,iu6,lu6,jlu6,ilu6,r6,e6,e6)
elseif (k.eq.5) then
call prolongation(ldim(6),e6,tmp)
call daxpy(tm,1.0d0,tmp,1,e5,1)
call gs(tm,iter,c5,jc5,ic5,u5,ju5,iu5,lu5,jlu5,ilu5,r5,e5,e5)
elseif (k.eq.4) then
call prolongation(ldim(5),e5,tmp)
call daxpy(tm,1.0d0,tmp,1,e4,1)
call gs(tm,iter,c4,jc4,ic4,u4,ju4,iu4,lu4,jlu4,ilu4,r4,e4,e4)
elseif (k.eq.3) then
call prolongation(ldim(4),e4,tmp)
call daxpy(tm,1.0d0,tmp,1,e3,1)
call gs(tm,iter,c3,jc3,ic3,u3,ju3,iu3,lu3,jlu3,ilu3,r3,e3,e3)
elseif (k.eq.2) then
call prolongation(ldim(3),e3,tmp)
call daxpy(tm,1.0d0,tmp,1,e2,1)
call gs(tm,iter,c2,jc2,ic2,u2,ju2,iu2,lu2,jlu2,ilu2,r2,e2,e2)
elseif (k.eq.1) then
call prolongation(ldim(2),e2,tmp)
call daxpy(tm,1.0d0,tmp,1,x,1)
call gs(tm,iter,c1,jc1,ic1,u1,ju1,iu1,lu1,jlu1,ilu1,rhs,x,x)
endif
enddo
```

```
       call amux(ldimc(1),x,tmp,c1,jc1,ic1)
       call daxpy(ldimc(1),-1.0d0,rhs,1,tmp,1)
       print*, dnrm2(ldimc(1),tmp,1),cyc
print*, 'END OF CYCLE', cyc
       if (dnrm2(ldimc(1),tmp,1).gt. 1.0d-06) goto 20


        call cpu_time(t2)
print*, 'The matrix generation time is:', t3-t1-matread
print*
print*, 'The time for rhs is:', t4-t3-rhsread
print*
print*, 'The time for the iluk is:', t5-t4
print*
print*, 'The multigrid time is:', t2-t5
print*
        print*, 'The total time is :', t2-t1-matread-rhsread

C  SKIP THE REST


c       allocate(uc(n**2))

c       call dexa(n,initb,initbx,initu,initux,initl,initly,initr
c     +  ,initry,x,uc)
```

```
      deallocate(levs)
      deallocate(jw)
      deallocate(wk)
      deallocate(rhs)

      deallocate(initu)
      deallocate(initux)
      deallocate(initb)
      deallocate(initbx)
      deallocate(initl)
      deallocate(initly)
      deallocate(initr)
      deallocate(initry)



c2      print*, 'SUCCESS in',i,'steps and error norm2',
c     +  dnrm2(m,clonex,1)/dnrm2(m,x,1)

c       open(1,file='collocation256.txt')
c       do i=1,ldimc(1)
c       do j=ic1(i),ic1(i+1)-1
c       write(1,*) i,jc1(j),c1(j)
c       enddo
c       enddo
c       close(1)
```

```
50      continue
        open(1,file='c.txt')
        do i=1,ldimc(1)
        write(1,*) x(i)
        enddo
        close(1)
        deallocate(x)




        stop
        end
```

Α΄.2 Υποπρογράμματα

```
C------------SUBROUTINES-----------------------------
C
C------------INSTRUCTIONS-----------------------------
C ================================================
C |                                              |
C | This subroutine performs a C=alpha*A*B+beta*C    |
C | where A is a diagonal matrix, B and C are sparse |
C | matrices with the same structure.              |
C |                                              |
```

```
C | INPUT ARGUMENTS=..                                    |
C |                                                       |
C | m  := INTEGER , arrays number of A,B,C                |
C | n  := INTEGER , columns number of A,B,C               |
C | alpha:=REAL*8 , scalar                                |
C | A  := REAL*8  , matrix A                              |
C | beta:=REAL*8  , scalar                                |
C | B  := REAL*8  , matrix B                              |
C | jc :: INTEGER array-pointer for collumns of C         |
C | ic :: INTEGER array-pointer for elements that         |
C |change line of matrix C                                |
C |                                                       |
C | OUTPUT ARGUMENTS=..                                   |
C |                                                       |
C | C  := REAL*8  , matrix C                              |
C |                                                       |
C |_______________________________________________________|

      subroutine dsmsm(m,n,alpha,A,B,beta,C,jc,ic)

      real*8 alpha,A(*),B(*),beta,C(*)
      integer m,n,jc(*),ic(*)

      do i=1,m
      do j=ic(i),ic(i+1)-1
         C(j)=alpha*A(i)*B(j)+beta*C(j)
      enddo
```

```
      enddo

      return
      end


C-----------INSTRUCTIONS-----------------------------
C ================================================
C |                                              |
C | This subroutine computes the approximation of the|
C | U in the center of the cells.                |
C |                                              |
C | INPUTS ::                                    |
C |                                              |
C | n :: integer, number of elements             |
C | inb :: real*8, dimension n+1 the U (bottom)  |
C | inbx:: real*8, dimension n+1 the Ux  (bottom)|
C | inu :: real*8, dimension n+1 the U (up)      |
C | inux:: real*8, dimension n+1 the Ux (up)     |
C | inl :: real*8, dimension n+1 the U (left)    |
C | inlx:: real*8, dimension n+1 the Ux (left)   |
C | inr :: real*8, dimension n+1 the U (right)   |
C | inrx:: real*8, dimension n+1 the Ux (right)  |
C | sol :: real*8, dimension 4*n^2 the solution  |
C |                                              |
C | OUTPUTS ::                                   |
C |                                              |
```

```
C | u :: real*8, dimension n^2                                    |
C |                                                               |
C |_______________________________________________________________|

      subroutine dexa(n,inb,inbx,inu,inux,inl,inly,inr,inry,sol,u)

      integer n,q,z
      real*8 inb(*),inbx(*),inu(*),inux(*),inl(*),inly(*)
      real*8 inr(*),inry(*),sol(*),u(*),h,f1,f2,f3,f4

      h=1.0d0/dble(n)
      f1=1.0d0/2.0d0
      f3=f1
      f2=1.0d0/8.0d0
      f4=-f2

c      do i=1,n
c      x(i)=dble(i)*h-h/2.0d0
c      enddo

      q=1
      do i=1,n
      u(i)=inl(q)*f1*f1+h*inly(q)*f1*f2+inl(q+1)*f1*f3+
     +  h*inly(q+1)*f1*f4
      q=q+1
      enddo
```

```
  u(1)=u(1)+h*inbx(1)*f2*f1+sol(1)*f2*f2+sol(2)*f2*f3+
+  sol(3)*f2*f4
  u(1)=u(1)+inb(2)*f3*f1+sol(2*n+1)*f3*f2+sol(2*n+2)*f3*f3+
+  sol(2*n+3)*f3*f4
  u(1)=u(1)+h*inbx(2)*f4*f1+sol(4*n+1)*f4*f2+sol(4*n+2)*f4*f3+
+  sol(4*n+3)*f4*f4

  q=2
  do i=2,n-1
  u(i)=u(i)+sol(q)*f2*f1+sol(q+1)*f2*f2+sol(q+2)*f2*f3+sol(q+3)*
+  f2*f4
  u(i)=u(i)+sol(2*n+q)*f3*f1+sol(2*n+q+1)*f3*f2+sol(2*n+q+2)*
+  f3*f3+sol(2*n+q+3)*f3*f4
  u(i)=u(i)+sol(4*n+q)*f4*f1+sol(4*n+q+1)*f4*f2+sol(4*n+q+2)*
+  f4*f3+sol(4*n+q+3)*f4*f4
  q=q+2
  enddo

  u(n)=u(n)+sol(2*n-2)*f2*f1+sol(2*n-1)*f2*f2+h*inux(1)*f2*f3+
+  sol(2*n)*f2*f4
  u(n)=u(n)+sol(4*n-2)*f3*f1+sol(4*n-1)*f3*f2+inu(2)*f3*f3+
+  sol(4*n)*f3*f4
  u(n)=u(n)+sol(6*n-2)*f4*f1+sol(6*n-1)*f4*f2+h*inux(2)*f4*f3+
+  sol(6*n)*f4*f4

  do i=2,n-1
  z=(i-2)*4*n+2*n
```

```
   u((i-1)*n+1)=inb(i)*f1*f1+sol(z+1)*f1*f2+sol(z+2)*f1*f3+
+    sol(z+3)*f1*f4
   u((i-1)*n+1)=u((i-1)*n+1)+h*inbx(i)*f2*f1+sol(z+2*n+1)*f2*f2+
+    sol(z+2*n+2)*f2*f3+sol(z+2*n+3)*f2*f4
   u((i-1)*n+1)=u((i-1)*n+1)+inb(i+1)*f3*f1+sol(z+4*n+1)*f3*f2+
+    sol(z+4*n+2)*f3*f3+sol(z+4*n+3)*f3*f4
   u((i-1)*n+1)=u((i-1)*n+1)+h*inbx(i+1)*f4*f1+sol(z+6*n+1)*f4*f2+
+    sol(z+6*n+2)*f4*f3+sol(z+6*n+3)*f4*f4
  q=(i-2)*4*n+2*n+2
  do k=2,n-1
   u((i-1)*n+k)=sol(q)*f1*f1+sol(q+1)*f1*f2+sol(q+2)*f1*f3+
+    sol(q+3)*f1*f4
   u((i-1)*n+k)=u((i-1)*n+k)+sol(q+2*n)*f2*f1+sol(q+2*n+1)*f2*f2+
+    sol(q+2*n+2)*f2*f3+sol(q+2*n+3)*f2*f4
   u((i-1)*n+k)=u((i-1)*n+k)+sol(q+4*n)*f3*f1+sol(q+4*n+1)*f3*f2+
+    sol(q+4*n+2)*f3*f3+sol(q+4*n+3)*f3*f4
   u((i-1)*n+k)=u((i-1)*n+k)+sol(q+6*n)*f4*f1+sol(q+6*n+1)*f4*f2+
+    sol(q+6*n+2)*f4*f3+sol(q+6*n+3)*f4*f4
  q=q+2
  enddo
  z=(i-2)*4*n+4*n-2
   u(i*n)=sol(z)*f1*f1+sol(z+1)*f1*f2+inu(i)*f1*f3+sol(z+2)*f1*f4
   u(i*n)=u(i*n)+sol(z+2*n)*f2*f1+sol(z+2*n+1)*f2*f2+h*inux(i)
+    *f2*f3+sol(z+2*n+2)*f2*f4
   u(i*n)=u(i*n)+sol(z+4*n)*f3*f1+sol(z+4*n+1)*f3*f2+inu(i+1)*f3*
+    f3+sol(z+4*n+2)*f3*f4
   u(i*n)=u(i*n)+sol(z+6*n)*f4*f1+sol(z+6*n+1)*f4*f2+h*inux(i+1)*
```

```
+    f4*f3+sol(z+6*n+2)*f4*f4
     enddo

     do i=1,n
     u(n*(n-1)+i)=inr(i)*f3*f1+h*inry(i)*f3*f2+inr(i+1)*f3*f3+
+    h*inry(i+1)*f3*f4
     enddo

   z=4*n**2-6*n+1
     u(n*(n-1)+1)=u(n*(n-1)+1)+inb(n)*f1*f1+sol(z)*f1*f2+sol(z+1)*
+    f1*f3+sol(z+2)*f1*f4
     u(n*(n-1)+1)=u(n*(n-1)+1)+h*inbx(n)*f2*f1+sol(z+2*n)*f2*f2+
+    sol(z+2*n+1)*f2*f3+sol(z+2*n+2)*f2*f4
     u(n*(n-1)+1)=u(n*(n-1)+1)+h*inbx(n+1)*f4*f1+sol(z+4*n)*f4*f2+
+    sol(z+4*n+1)*f4*f3+sol(z+4*n+2)*f4*f4


   z=z+1
   do k=2,n-1
     u(n*(n-1)+k)=u(n*(n-1)+k)+sol(z)*f1*f1+sol(z+1)*f1*f2+sol(z+2)*
+    f1*f3+sol(z+3)*f1*f4
     u(n*(n-1)+k)=u(n*(n-1)+k)+sol(z+2*n)*f2*f1+sol(z+2*n+1)*f2*f2+
+    sol(z+2*n+2)*f2*f3+sol(z+2*n+3)*f2*f4
     u(n*(n-1)+k)=u(n*(n-1)+k)+sol(z+4*n)*f4*f1+sol(z+4*n+1)*f4*f2+
+    sol(z+4*n+2)*f4*f3+sol(z+4*n+3)*f4*f4
   z=z+2
   enddo
```

```
        z=4*n**2-4*n-2
         u(n**2)=u(n**2)+sol(z)*f1*f1+sol(z+1)*f1*f2+inu(n)*f1*f3
     +    +sol(z+2)*f1*f4
         u(n**2)=u(n**2)+sol(z+2*n)*f2*f1+sol(z+2*n+1)*f2*f2+h*inux(n)*
     +    f2*f3+sol(z+2*n+2)*f2*f4
         u(n**2)=u(n**2)+sol(z+4*n)*f4*f1+sol(z+4*n+1)*f4*f2+h*inux(n+1)
     +    *f4*f3+sol(z+4*n+2)*f4*f4


        return
        end

C-------------------------------------------------------------
C-------------------------------------------------------------
C-------------------------------------------------------------

        integer function pof2(x)

        integer x,i,k

        k=0
        i=x
        if (i.eq.2) then
        k=1
        goto 2
        endif
```

```
1       if (mod(i,2).eq.0) then
       k=k+1
       if (i.eq.2) then
       goto 2
       endif
       i=i/2
       goto 1
       else
       k=-1
       endif

2       pof2=k
       return
       end



C-----------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------------------------------------------------------

        subroutine gs(m,iter,c,jc,ic,u,ju,iu,lu,jlu,ilu,rhs,x,x0)

       real*8 c(*),u(*),lu(*),rhs(*),x(*),hu(m),r(m),clonex(m)
       real*8 dnrm2,xo(m),x0(*)
       integer n,jc(*),ic(*),ju(*),iu(*),jlu(*),ilu(*),m,iter,z
```

```
      call dcopy(m,x0,1,xo,1)

      z=0
      do i=1,iter
      z=z+1
      call dcopy(m,xo,1,clonex,1)
      call amux(m,xo,xo,u,ju,iu)
      call dscal(m,-1.0d0,xo,1)
      call daxpy(m,1.0d0,rhs,1,xo,1)
      call lusol(m,xo,x,lu,jlu,ilu)
      call daxpy(m,-1.0d0,x,1,clonex,1)
c      print*, i,dnrm2(m,clonex,1)/dnrm2(m,x,1)
      call dcopy(m,x,1,xo,1)
      if (dnrm2(m,clonex,1)/dnrm2(m,x,1).lt. 1.0d-08) then
      goto 2
      endif
      enddo

2     continue
      print*, 'End of gs in', z,'steps'

      return
      end
```

```
C ================================================
C |---------------Instructions---------------------|
C |                                                |
C | THIS IS A SPARSE VERSION OF getfunctionb.f     |
C | for the right hand side                        |
C |                                                |
C | This subroutine helps user to choose the coe-  |
C | -fficients of his choice.                      |
C |                                                |
C | INPUT ARGUMENTS :                              |
C |                                                |
C | m  := INTEGER , number of x'x elements         |
C | n  := INTEGER , number of y'y elements         |
C | A  := REAL*8  , diagonal matrix A              |
C | flag:=INTEGER , information for the coefficient|
C |                                                |
C |________________________________________________|
      subroutine getfunctionbmg(m,n,A,flag,func)
      real*8 A(*),pi,e
      integer m,n,lda,flag
      character*64 func


c      compiler='gfortran'
      pi=4.0d0*datan(1.0d0)
      e=dexp(1.0d0)
```

```
      open(1,file='temp.f')
      write(1,*) '       program temp'
      write(1,*) '       real*8 f,x,y,hx,hy,pi,e '
      write(1,*) '       integer m,n,i,j,k,l '

c      print*, 'Give the function'
      if (func.eq.'0') then
      flag=0
      call system('rm temp.f')
      return
      endif


      write(1,*) '       pi=4.0d0*datan(1.0d0)'
      write(1,*) '       e=dexp(1.0d0)'
      write(1,*) '       hx=1.0d0/',dble(m)
      write(1,*) '       hy=1.0d0/',dble(n)

      write(1,*) '       open(2,file="buff.txt")'
      write(1,*) '       do i=1,',m
      write(1,*) '       do k=1,2'
      write(1,*) '       if (k.eq.2) then'
      write(1,*) '       x=hx/2.0d0*(2.0d0*dble(i)-1.0d0+sqrt(3.0d0)
     &  /3.0d0)'
      write(1,*) '       else'
      write(1,*) '       x=hx/2.0d0*(2.0d0*dble(i)-1.0d0-sqrt(3.0d0)
     &  /3.0d0)'
```

```
      write(1,*) '       endif'

      write(1,*) '       do j=1,',n
      write(1,*) '       do l=1,2'
      write(1,*) '       if (l.eq.2) then'
      write(1,*) '       y=hy/2.0d0*(2.0d0*dble(j)-1.0d0+sqrt(3.0d0)
     &  /3.0d0)'
      write(1,*) '       else'
      write(1,*) '       y=hy/2.0d0*(2.0d0*dble(j)-1.0d0-sqrt(3.0d0)
     &  /3.0d0)'
      write(1,*) '       endif'

      write(1,*) '       f=',func

      write(1,*) '       write(2,*) f'

      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       close(2)'
      write(1,*) '       end'
      close(1)
```

```
C You can change the compiler below
       call system('gfortran -o t temp.f')
       call system('./t')


       open(3,file='buff.txt')
       do i=1,4*m*n
       read(3,*) A(i)
       enddo
       close(3)


c       call system('rm temp.f')
c       call system('rm t')
c       call system('rm buff.txt')

       return
       end



C ================================================
C |---------------Instructions---------------------|
C |                                                |
C | THIS IS A SPARSE VERSION OF getfunctionb2.f    |
C | for the right hand side                        |
C |                                                |
```

```
C | This subroutine helps user to choose the coe-  |
C | -fficients of his choice.                       |
C |                                                 |
C | INPUT ARGUMENTS :                               |
C |                                                 |
C | m  := INTEGER , number of x'x elements          |
C | n  := INTEGER , number of y'y elements          |
C | A  := REAL*8  , diagonal matrix A               |
C | flag:=INTEGER , information for the coefficient|
C |                                                 |
C |_________________________________________________|
      subroutine getfunctionb2mg(m,n,A,flag,time,func)
      real*8 A(*),pi,e,x,y,t1,t2,time
      integer m,n,lda,flag
      character*64 func


c      compiler='gfortran'
      pi=4.0d0*datan(1.0d0)
      e=dexp(1.0d0)

      open(1,file='temp.f')
      write(1,*) '       program temp'
      write(1,*) '       real*8 f,x,y,hx,hy,pi,e '
      write(1,*) '       integer m,n,i,j,k,l '

c      print*, 'Give the function'
```

```
call cpu_time(t1)


call cpu_time(t2)
time=(t2-t1)
       if (func.eq.'0') then
       flag=0
       do i=1,4*m*n
       A(i)=0.0d0
       enddo
       call system('rm temp.f')
       return
       endif


       write(1,*) '       pi=4.0d0*datan(1.0d0)'
       write(1,*) '       e=dexp(1.0d0)'
       write(1,*) '       hx=1.0d0/',dble(m)
       write(1,*) '       hy=1.0d0/',dble(n)
       write(1,*) '       open(2,file="buff.txt")'
       write(1,*) '       do i=1,',m
       write(1,*) '       do k=1,2'
       write(1,*) '       if (k.eq.2) then'
       write(1,*) '       x=hx/2.0d0*(2.0d0*dble(i)-1.0d0+sqrt(3.0d0)
     &  /3.0d0)'
       write(1,*) '       else'
       write(1,*) '       x=hx/2.0d0*(2.0d0*dble(i)-1.0d0-sqrt(3.0d0)
     &  /3.0d0)'
       write(1,*) '       endif'
```

```
      write(1,*) '       do j=1,',n
      write(1,*) '       do l=1,2'
      write(1,*) '       if (l.eq.2) then'
      write(1,*) '       y=hy/2.0d0*(2.0d0*dble(j)-1.0d0+sqrt(3.0d0)
     &  /3.0d0)'
      write(1,*) '       else'
      write(1,*) '       y=hy/2.0d0*(2.0d0*dble(j)-1.0d0-sqrt(3.0d0)
     &  /3.0d0)'
      write(1,*) '       endif'

      write(1,*) '       f=',func
      write(1,*) '       f=hx*hy*f'

      write(1,*) '       write(2,*) f'

      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       enddo'
      write(1,*) '       close(2)'
      write(1,*) '       end'
      close(1)


C You can change the compiler below
      call system('gfortran -o t temp.f')
```

```
      call system('./t')



      open(3,file='buff.txt')
      do i=1,4*m*n
      read(3,*) A(i)
      enddo
      close(3)



      call system('rm temp.f')
      call system('rm t')
      call system('rm buff.txt')

      return
      end

C ================================================
C |-------Instructions-----------------------------|
C |                                                |
C | This subroutine performs a kronecker product   |
C | between two sparse matrices of dimension ,     |
C | (nza) and (nzb). The result finally stored     |
C | at matrix c with dimension (nza*nzb).          |
C |                                                |
C | THIS IS THE SPARSE VERSION OF kron.f           |
C | format used :: CSR                             |
```

```
C |                                                    |
C | INPUT ARGUMENTS:..                                 |
C |                                                    |
C | ma :: INTEGER arrays of A                          |
C | na :: INTEGER columns of A                         |
C | a  :: REAL*8  sparse matrix A (array)              |
C | ja :: INTEGER array-pointer for collumns of A      |
C | ia :: INTEGER array-pointer for elements that      |
C |change line of matrix A                             |
C | mb :: INTEGER arrays of B                          |
C | nb :: INTEGER columns of B                         |
C | b  :: REAL*8  sparse matrix B (array)              |
C | jb :: INTEGER array-pointer for collumns of B      |
C | ib :: INTEGER array-pointer for elements that      |
C |change line of matrix B                             |
C |                                                    |
C | OUTPUT ARGUMENTS:..                                |
C |                                                    |
C | c  :: REAL*8  sparse matrix C (array)              |
C | jc :: INTEGER array-pointer for collumns of C      |
C | ic :: INTEGER array-pointer for elements that      |
C |change line of matrix C                             |
C |____________________________________________________|
C
C
C Date   :: 11/5/2012.
```

```
subroutine krons(ma,na,a,ja,ia,mb,nb,b,jb,ib,c,jc,ic)

real*8 a(*),b(*),c(*)
integer ja(*),jb(*),ia(*),ib(*),jc(*),ic(*)
integer tmpi,w,q,point,iac((8*mb-4)**2+1)



q=0
do k=1,ma
       do i=1,mb
do l=ia(k),ia(k+1)-1

       do j=ib(i),ib(i+1)-1
       q=q+1
       c(q)=a(l)*b(j)
       iac(q)=(k-1)*mb+i
       jc(q)=(ja(l)-1)*nb+jb(j)

       enddo
       enddo
 enddo
 enddo

ic(1)=iac(1)
w=1
do i=1,q-1
tmpi=iac(i)
```

```
        if (iac(i+1)-tmpi.eq.1) then
        w=w+1
        ic(w)=i+1
        elseif (iac(i+1)-tmpi.gt.1) then
        do j=1,iac(i+1)-tmpi
        w=w+1
        ic(w)=i+1
        enddo
endif
enddo
ic(w+1)=ic(1)+q



         return
         end

C-------------------------------------------------------------
C-------------------------------------------------------------
C-------------------------------------------------------------
C The prolongation subroutine



        subroutine prolongation(ns,xold,xnew)
        implicit real*8 (a-h,o-z)

        real*8 xnew(*),xold(*)
```

```
      integer ns

         nold=4*ns*ns
         nnew=16*ns*ns
         d=1.0d0/2.0d0
         dd=1.0d0/4.0d0
         h=1.0d0/dfloat(ns)

C        U2i,2j(h)=Ui,j(2h)

        xnew(1)=xold(1)
        do i=1,ns-1
          do k=0,1
            xnew(4*i+k)=xold(2*i+k)
          end do
        end do
        xnew(4*ns)=xold(2*ns)

        do j=1,ns-1
          do i=1,ns-1
            do l=0,1
              do k=0,1

                xnew(12*ns+16*ns*(j-1)+4*i+k+l*4*ns)=
     +               xold((l+1)*2*ns+4*ns*(j-1)+2*i+k)
            end do
          end do
```

```
        end do

       xnew(12*ns+16*ns*(j-1)+1)=xold(2*ns*(2*j-1)+1)
       xnew(16*ns+16*ns*(j-1))=xold(2*ns*2*j)
       xnew(16*ns+16*ns*(j-1)+1)=xold(2*ns*2*j+1)
       xnew(20*ns+16*ns*(j-1))=xold(2*ns*(2*j+1))
      end do

      xnew(nnew-4*ns+1)=xold(nold-2*ns+1)
      do i=1,ns-1
        do k=0,1
          xnew(nnew-4*ns+4*i+k)=xold(nold-2*ns+2*i+k)
        end do
      end do
      xnew(nnew)=xold(nold)

C     U2i+1,2j(h)=1/2{Ui,j(2h)+Ui+1,j(2h)}

       xnew(4*ns+1)=d*(xold(2*ns+1))
       xnew(8*ns+1)=d*(xold(1)+xold(4*ns+1))
       do i=1,ns-1
        xnew(4*ns+4*i)=d*(xold(2*ns+2*i))
        xnew(4*ns+4*i+1)=d*(xold(2*ns+2*i+1))
        xnew(8*ns+4*i)=d*(xold(2*i)+xold(4*ns+2*i))
        xnew(8*ns+4*i+1)=d*(xold(2*i+1)+xold(4*ns+2*i+1))
       end do
       xnew(8*ns)=d*(xold(4*ns))
```

```
      xnew(12*ns)=d*(xold(2*ns)+xold(6*ns))

      do j=1,ns-2
        do i=1,ns-1
          do l=0,1
            do k=0,1

       xnew(20*ns+16*ns*(j-1)+4*i+k+l*4*ns)=d*(xold(2*ns+4*ns*(j-1)+
     +             2*i+k+l*2*ns)+ xold(6*ns+4*ns*(j-1)+2*i+k+l*2*ns))
              end do
            end do
          end do
      xnew(20*ns+16*ns*(j-1)+1)=d*(xold(2*ns+4*ns*(j-1)+1)+
     +               xold(6*ns+4*ns*(j-1)+1))
      xnew(24*ns+16*ns*(j-1)+1)=d*(xold(4*ns+4*ns*(j-1)+1)+
     +               xold(8*ns+4*ns*(j-1)+1))
      xnew(24*ns+16*ns*(j-1))=d*(xold(4*ns+4*ns*(j-1))+
     +              xold(8*ns+4*ns*(j-1)))
      xnew(28*ns+16*ns*(j-1))=d*(xold(6*ns+4*ns*(j-1))+
     +              xold(10*ns+4*ns*(j-1)))
      end do

    xnew(nnew-12*ns+1)=d*(xold(nold-6*ns+1))
    xnew(nnew-8*ns+1)=d*(xold(nold-4*ns+1)+xold(nold-2*ns+1))
    do i=1,ns-1
    xnew(nnew-12*ns+4*i)=d*(xold(nold-6*ns+2*i))
    xnew(nnew-12*ns+4*i+1)=d*(xold(nold-6*ns+2*i+1))
```

```
      xnew(nnew-8*ns+4*i)=d*(xold(nold-4*ns+2*i)+xold(nold-2*ns+2*i))
      xnew(nnew-8*ns+4*i+1)=d*(xold(nold-4*ns+2*i+1)+
     +       xold(nold-2*ns+2*i+1))
      end do
      xnew(nnew-8*ns)=d*(xold(nold-4*ns))
      xnew(nnew-4*ns)=d*(xold(nold-2*ns)+xold(nold))



C         U2i,2j+1(h)=1/2{Ui,j(2h)+Ui,j+1(2h)}

      xnew(2)=d*(xold(2))
      xnew(3)=d*(xold(1)+xold(3))
      do i=1,ns-2
        do k=0,1
          xnew(2+4*i+k)=d*(xold(2*i+k)+xold(2*(i+1)+k))
        end do
      end do
      xnew(4*ns-2)=d*(xold(2*ns-2))
      xnew(4*ns-1)=d*(xold(2*ns-1)+xold(2*ns))

      do j=1,ns-1
        do i=1,ns-2
          do l=0,1
            do k=0,1

           xnew(12*ns+16*ns*(j-1)+2+4*i+k+l*4*ns)=d*
```

```
+              (xold(2*ns+4*ns*(j-1)+2*i+k+l*2*ns)+
+              xold(2*ns+4*ns*(j-1)+2*(i+1)+k+l*2*ns))
        end do
       end do
      end do
    xnew(12*ns+16*ns*(j-1)+2)=d*(xold(2*ns+4*ns*(j-1)+2))
    xnew(12*ns+16*ns*(j-1)+3)=d*(xold(2*ns+4*ns*(j-1)+1)+
+                     xold(2*ns+4*ns*(j-1)+3))
    xnew(16*ns+16*ns*(j-1)+2)=d*(xold(4*ns+4*ns*(j-1)+2))
    xnew(16*ns+16*ns*(j-1)+3)=d*(xold(4*ns+4*ns*(j-1)+1)+
+                    xold(4*ns+4*ns*(j-1)+3))

    xnew(16*ns+16*ns*(j-1)-2)=d*(xold(4*ns+4*ns*(j-1)-2))
    xnew(16*ns+16*ns*(j-1)-1)=d*(xold(4*ns+4*ns*(j-1)-1)+
+                      xold(4*ns+4*ns*(j-1)))
    xnew(20*ns+16*ns*(j-1)-2)=d*(xold(6*ns+4*ns*(j-1)-2))
    xnew(20*ns+16*ns*(j-1)-1)=d*(xold(6*ns+4*ns*(j-1)-1)+
+                      xold(6*ns+4*ns*(j-1)))

  end do

  xnew(nnew-4*ns+2)=d*(xold(nold-2*ns+2))
  xnew(nnew-4*ns+3)=d*(xold(nold-2*ns+1)+xold(nold-2*ns+3))
  do i=1,ns-2
    do k=0,1
      xnew(nnew-4*ns+2+4*i+k)=d*(xold(nold-2*ns+2*i+k)+
+                          xold(nold-2*ns+2*(i+1)+k))
```

```
          end do
        end do
        xnew(nnew-2)=d*(xold(nold-2))
        xnew(nnew-1)=d*(xold(nold-1)+xold(nold))

C       U2i+1,2j+1(h)=1/4{Ui,j(2h)+Ui,j+1(2h)+Ui+1,j(2h)+Ui+1,j+1(2h)}

        xnew(4*ns+2)=dd*(xold(2*ns+2))
        xnew(4*ns+3)=dd*(xold(2*ns+1)+xold(2*ns+3))
        xnew(8*ns+2)=dd*(xold(2)+xold(4*ns+2))
        xnew(8*ns+3)=dd*(xold(1)+xold(3)+xold(4*ns+1)+xold(4*ns+3))
           do i=1,ns-2
            xnew(4*ns+2+4*i)=dd*(xold(2*ns+2*i)+
     +        xold(2*ns+2*(i+1)))
            xnew(4*ns+3+4*i)=dd*(xold(2*ns+2*i+1)+
     +         xold(2*ns+2*(i+1)+1))
            xnew(8*ns+2+4*i)=dd*(xold(2*i)+xold(2*(i+1))+
     +                xold(4*ns+2*i)+xold(4*ns+2*(i+1)))
            xnew(8*ns+3+4*i)=dd*(xold(1+2*i)+xold(1+2*(i+1))+
     +                xold(4*ns+1+2*i)+xold(4*ns+1+2*(i+1)))
          end do
         xnew(8*ns-2)=dd*(xold(4*ns-2))
         xnew(8*ns-1)=dd*(xold(4*ns-1)+xold(4*ns))
         xnew(12*ns-2)=dd*(xold(2*ns-2)+xold(6*ns-2))
         xnew(12*ns-1)=dd*(xold(2*ns-1)+xold(2*ns)+
     +      xold(6*ns-1)+xold(6*ns))
```

```
      do j=1,ns-2
        do i=1,ns-2
          do l=0,1
            do k=0,1

          xnew(20*ns+16*ns*(j-1)+2+k+4*i+l*4*ns)=dd*(
     +           xold(2*ns+4*ns*(j-1)+k+2*i+l*2*ns)+
     +           xold(2*ns+4*ns*(j-1)+k+2*(i+1)+l*2*ns)+
     +           xold(6*ns+4*ns*(j-1)+k+2*i+l*2*ns)+
     +           xold(6*ns+4*ns*(j-1)+k+2*(i+1)+l*2*ns))
            end do
          end do
        end do

        xnew(20*ns+16*ns*(j-1)+2)=dd*(xold(2*ns+4*ns*(j-1)+2)+
     +          xold(6*ns+4*ns*(j-1)+2))
        xnew(20*ns+16*ns*(j-1)+3)=dd*(xold(2*ns+4*ns*(j-1)+1)+
     +           xold(2*ns+4*ns*(j-1)+3)+xold(6*ns+4*ns*(j-1)+1)+
     +           xold(6*ns+4*ns*(j-1)+3))
        xnew(24*ns+16*ns*(j-1)+2)=dd*(
     +          xold(4*ns+4*ns*(j-1)+2)+xold(8*ns+4*ns*(j-1)+2))
        xnew(24*ns+16*ns*(j-1)+3)=dd*(xold(4*ns+4*ns*(j-1)+1)+
     +           xold(4*ns+4*ns*(j-1)+3)+xold(8*ns+4*ns*(j-1)+1)+
     +           xold(8*ns+4*ns*(j-1)+3))
        xnew(24*ns+16*ns*(j-1)-2)=dd*(xold(4*ns+4*ns*(j-1)-2)+
     +           xold(8*ns+4*ns*(j-1)-2))
```

```
        xnew(24*ns+16*ns*(j-1)-1)=dd*(xold(4*ns+4*ns*(j-1)-1)+
     +           xold(4*ns+4*ns*(j-1))+xold(8*ns+4*ns*(j-1)-1)+
     +           xold(8*ns+4*ns*(j-1)))
        xnew(28*ns+16*ns*(j-1)-2)=dd*(xold(6*ns+4*ns*(j-1)-2)+
     +         xold(10*ns+4*ns*(j-1)-2))
        xnew(28*ns+16*ns*(j-1)-1)=dd*(xold(6*ns+4*ns*(j-1)-1)+
     +           xold(6*ns+4*ns*(j-1))+xold(10*ns+4*ns*(j-1)-1)+
     +           xold(10*ns+4*ns*(j-1)))

      end do


      xnew(nnew-12*ns+2)=dd*(xold(nold-6*ns+2))
      xnew(nnew-12*ns+3)=dd*(xold(nold-6*ns+1)+xold(nold-6*ns+3))
      xnew(nnew-8*ns+2)=dd*(xold(nold-4*ns+2)+xold(nold-2*ns+2))
      xnew(nnew-8*ns+3)=dd*(xold(nold-4*ns+1)+xold(nold-4*ns+3)+
     +        xold(nold-2*ns+1)+xold(nold-2*ns+3))

      do i=1,ns-2
       xnew(nnew-12*ns+2+4*i)=dd*(xold(nold-6*ns+2*i)+
     +           xold(nold-6*ns+2*(i+1)))
       xnew(nnew-12*ns+3+4*i)=dd*(xold(nold-6*ns+1+2*i)+
     +         xold(nold-6*ns+1+2*(i+1)))
       xnew(nnew-8*ns+2+4*i)=dd*(xold(nold-4*ns+2*i)+
     +           xold(nold-4*ns+2*(i+1))+xold(nold-2*ns+2*i)+
     +           xold(nold-2*ns+2*(i+1)))
       xnew(nnew-8*ns+3+4*i)=dd*(xold(nold-4*ns+1+2*i)+
```

```
     +           xold(nold-4*ns+1+2*(i+1))+xold(nold-2*ns+1+2*i)+
     +           xold(nold-2*ns+1+2*(i+1)))
            end do

           xnew(nnew-8*ns-2)=dd*(xold(nold-4*ns-2))
           xnew(nnew-8*ns-1)=dd*(xold(nold-4*ns-1)+xold(nold-4*ns))
           xnew(nnew-4*ns-2)=dd*(xold(nold-2*ns-2)+
     +      xold(nold-2))
           xnew(nnew-4*ns-1)=dd*(xold(nold-2*ns-1)+
     +       xold(nold-2*ns)+xold(nold-1)+xold(nold))

c       prosoxi u, hux , huy ,h^2uxy

c       do j=1,2*ns-1
c       do i=1,4*ns,2
c        xnew(4*ns+8*ns*(j-1)+i)=xnew(4*ns+8*ns*(j-1)+i)/2.0d0
c       enddo
c         xnew(8*ns*j)=xnew(8*ns*j)/2.0d0
c       enddo

c       do j=1,2*ns+1
c       do i=2,4*ns-2,2
c        xnew(8*ns*(j-1)+i)=xnew(8*ns*(j-1)+i)/2.0d0
c       enddo
c       enddo

c       do j=1,2*ns+1
```

```
c       do i=1,4*ns,2
c        xnew(8*ns*(j-1)+i)=xnew(8*ns*(j-1)+i)/4.0d0
c       enddo
c         xnew(4*ns+8*ns*(j-1))=xnew(4*ns+8*ns*(j-1))/4.0d0
c       enddo


c          call dscal(nnew,4.0d0,xnew,1)
       return
       end

C-------------------------------------------------------------
C-------------------------------------------------------------
C-------------------------------------------------------------
C The restriction subroutine


       subroutine restriction(ns,xold,xnew)
       implicit real*8 (a-h,o-z)

       real*8 xnew(*),xold(*)
       integer ns

          nold=4*ns*ns
          nnew=ns*ns
          ns2=ns/2
          d=1.0d0/16.0d0
```

```
      d4=1.0d0/4.0d0
      h=1.0d0/dfloat(ns)


      xnew(1)=xold(1)

    do  i=1,(ns2-1)

     xnew(2*i)=d4*(xold(4*i-2)+2.0d0*xold(4*i)+xold(4*i+2))
     xnew(2*i+1)=d4*(xold(4*i-1)+2.0d0*xold(4*i+1)+xold(4*i+3))

    enddo

     xnew(ns)=xold(2*ns)

   do j=1,(ns2-1)
     do i=1,(ns2-1)
       do l=0,1
         do k=0,1

     xnew((l+1)*ns+2*ns*(j-1)+2*i+k)=d*(xold((l+1)*2*ns+8*ns*(j-1)+
+           4*(i-1)+2+k)+xold((l+1)*2*ns+8*ns*(j-1)+4*(i-1)+6+k)+
+           xold((l+5)*2*ns+8*ns*(j-1)+4*(i-1)+2+k)+
+           xold((l+5)*2*ns+8*ns*(j-1)+4*(i-1)+6+k)+
+           2.0d0*(xold((l+3)*2*ns+8*ns*(j-1)+4*(i-1)+2+k)+
+           xold((l+3)*2*ns+8*ns*(j-1)+4*(i-1)+6+k)+
+           xold((l+1)*2*ns+8*ns*(j-1)+4*(i-1)+4+k)+
```

```
+          xold((l+5)*2*ns+8*ns*(j-1)+4*(i-1)+4+k))+
+          4.0d0*xold((l+3)*2*ns+8*ns*(j-1)+4*(i-1)+4+k))

         end do
       end do
     end do
        xnew(ns+2*ns*(j-1)+1)=d4*(xold(2*ns+8*ns*(j-1)+1)+
+     xold(10*ns+8*ns*(j-1)+1)+2.0d0*xold(6*ns+8*ns*(j-1)+1))

         xnew(2*ns*j+1)=d4*(xold(4*ns+8*ns*(j-1)+1)+
+    xold(12*ns+8*ns*(j-1)+1)+2.0d0*xold(8*ns+8*ns*(j-1)+1))

        xnew(2*ns*j)=d4*(xold(4*ns+8*ns*(j-1))+
+      xold(12*ns+8*ns*(j-1))+2.0d0*xold(8*ns+8*ns*(j-1)))

        xnew(2*ns*j+ns)=d4*(xold(6*ns+8*ns*(j-1))+
+    xold(14*ns+8*ns*(j-1))+2.0d0*xold(10*ns+8*ns*(j-1)))
  end do

     xnew(nnew-ns+1)=xold(nold-2*ns+1)

  do i=1,(ns2-1)
     xnew(nnew-ns+2*i)=d4*(xold(nold-2*ns+4*i-2)+
+        xold(nold-2*ns+4*i+2)+2.0d0*xold(nold-2*ns+4*i))

     xnew(nnew-ns+2*i+1)=d4*(xold(nold-2*ns+4*i-1)+
+     xold(nold-2*ns+4*i+3)+2.0d0*xold(nold-2*ns+4*i+1))
```

```
 end do

xnew(nnew)=xold(nold)


   call dscal(nnew,4.0d0,xnew,1)


 return
 end
```

Παράρτημα Β΄

Κώδικας προγραμματισμού σε CUDA

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cublas.h>
#include <cublas_v2.h>
#include <time.h>
#include <sys/time.h>

double getTime(void)
{
    struct timeval tod;

    gettimeofday(&tod, NULL);
    double time_seconds = (double) tod.tv_sec +
      ((double) tod.tv_usec / 1000000.0);
    return time_seconds;
}
```

```
int main(){

  double *x, *y, a=1.0;
  double t1,t2;
  float time;
  double *dx,*dy;
  int n=1000*1000,ns=n*100;
  cublasHandle_t handleId;
  cudaEvent_t start,stop;

  cudaEventCreate( &start );
  cudaEventCreate( &stop );

  t1=getTime();
  cublasCreate( &handleId );
  t2=getTime();

  printf("Initialization time is = %f\n", t2-t1 );

  cudaEventRecord( start, 0);
  cudaHostAlloc( (void**)&x, ns*sizeof(double),
    cudaHostAllocWriteCombined | cudaHostAllocMapped );
  cudaHostAlloc( (void**)&y, ns*sizeof(double),
    cudaHostAllocWriteCombined | cudaHostAllocMapped );
  cudaEventRecord( stop, 0);
  cudaEventSynchronize( stop );
```

```
cudaEventElapsedTime( &time, start, stop );

printf("Allocating time is = %f\n", time/1000.0 );

t1=getTime();
for (int i=0;i<ns;i++){
  x[i]=1.0;
  y[i]=1.0;
}
t2=getTime();

printf("Setting vectors time is = %f\n", t2-t1 );

cudaEventRecord( start, 0);
cudaSetDeviceFlags( cudaDeviceMapHost );

cudaHostGetDevicePointer( &dx, x, 0 );
cudaHostGetDevicePointer( &dy, y, 0 );

cublasDaxpy( handleId, n, &a, dx, 1, dy, 1);
cudaThreadSynchronize();
cudaEventRecord( stop, 0);
cudaEventSynchronize( stop );
cudaEventElapsedTime( &time, start, stop );

printf("Total time is = %f, %f\n", time/1000.0, y[0]);
```

```
  cudaFreeHost( x );
  cudaFreeHost( y );
  cublasDestroy( handleId );

}


%----------------------------------------


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cublas.h>
#include <cublas_v2.h>
#include <time.h>
#include <sys/time.h>


double getTime(void)
{
    struct timeval tod;

    gettimeofday(&tod, NULL);
    double time_seconds = (double) tod.tv_sec +
      ((double) tod.tv_usec / 1000000.0);
    return time_seconds;
}
```

```
int main(){

  double *x, *y, a=1.0;
  float t1,t2,t3,t4,ts,td,tr;
  double *dx,*dy;
  int n=1000*1000,ns=n*100,devname,i;
  int nstreams=10;
  cudaStream_t stream,streams[nstreams];
  cublasHandle_t handleId;
  cudaDeviceProp prop;
  cublasStatus_t error;
  cudaEvent_t start,stop;


  cudaEventCreate( &start );
  cudaEventCreate( &stop );

  cudaGetDevice( &devname );
  cudaGetDeviceProperties( &prop, devname );
  if (!prop.deviceOverlap) {
    printf("This device will not handle overlaps\n");
  }

    cudaStreamCreate( &stream );
```

```
  t1=getTime();
  cudaHostAlloc( &x , ns*sizeof(double) , cudaHostAllocDefault );
  cudaHostAlloc( &y , ns*sizeof(double) , cudaHostAllocDefault );

  cudaMallocHost( &dx , n*sizeof(double) );
  cudaMallocHost( &dy , n*sizeof(double) );
  t2=getTime();
  printf("Allocating time is = %f\n", t2-t1);

  cublasCreate(&handleId);

  for (int i=0; i<ns; i++){
    x[i]=2.0;
    y[i]=1.0;
  }

  ts=0.0;
  td=0.0;
  tr=0.0;

  // ONE STREAM ASYNC PERFORMANCE FOR DAXPY

cudaEventRecord( start, 0);
  for (int i=0;i<ns;i+=n){
    cublasSetVectorAsync( n, sizeof(double), x+i, 1, dx, 1, stream );
    cublasSetVectorAsync( n, sizeof(double), y+i, 1, dy, 1, stream );
     if (stream == NULL){
```

```
        printf("Stream is NULL exiting the program\n");
        exit(-1);
      }
    cublasDaxpy( handleId, n, &a, dx, 1, dy, 1);

    error=cublasGetVectorAsync( n, sizeof(double), dy, 1, y+i
     , 1, stream );

    if (error!=CUBLAS_STATUS_SUCCESS){
      printf("Error reiciving y vector\n");
      exit(-1);
    }
  }

cudaEventRecord( stop, 0);
    cudaEventSynchronize( stop );
    cudaEventElapsedTime( &t1, start, stop );

    cudaStreamDestroy( stream );
    cudaFree( dx );
    cudaFree( dy );

  printf("\nGPU ONE STREAM Async Performance:\n");
  printf("Sending Time = %f \n", ts/1000.0);
  printf("Daxpy Time = %f \n", td/1000.0);
  printf("Recieving Time = %f \n", tr/1000.0);
  printf("Total Time = %f \n", t1/1000.0);
```

```
  // 10 STREAMS PERFORMANCE FOR DAXPY

  int sz=ns/nstreams;
  for (i=0;i<nstreams;i++){
  cudaStreamCreate( &streams[i] );
  }

  cudaMallocHost( &dx, ns*sizeof(double) );
  cudaMallocHost( &dy, ns*sizeof(double) );

  ts=0.0;
  td=0.0;
  tr=0.0;

cudaEventRecord( start, 0);
  for (i=0;i<nstreams;i++){
    cublasSetVectorAsync( sz, sizeof(double), x+i*sz, 1, dx+i*sz,
      1, streams[i] );
    cublasSetVectorAsync( sz, sizeof(double), y+i*sz, 1, dy+i*sz,
      1, streams[i] );

    cublasDaxpy( handleId, sz, &a, dx+i*sz, 1, dy+i*sz, 1);

    cublasGetVectorAsync( sz, sizeof(double), dy+i*sz, 1, y+i*sz,
     1, streams[i]);
```

```
  }
cudaEventRecord( stop, 0);
    cudaEventSynchronize( stop );
    cudaEventElapsedTime( &t1, start, stop );

  printf("\n\nGPU %d streams Async\n", nstreams);
  printf("Sending Time = %f \n", ts/1000.0);
  printf("Daxpy Time = %f \n", td/1000.0);
  printf("Recieving Time = %f \n", tr/1000.0);
  printf("Total Time = %f \n", t1/1000.0);

  for (i=0;i<nstreams;i++){
  cudaStreamDestroy( streams[i] );
  }

  cudaFree( dx);
  cudaFree( dy);

 }


%----------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cublas.h>
```

```
#include <cublas_v2.h>
#include <time.h>
#include <sys/time.h>

double getTime(void)
{
    struct timeval tod;

    gettimeofday(&tod, NULL);
    double time_seconds = (double) tod.tv_sec +
      ((double) tod.tv_usec / 1000000.0);
    return time_seconds;
}

extern "C" {


double *dev_x, *dev_y;
cublasHandle_t handleId;

void init_(int *N)
{
cublasStatus_t cublas_error;
cudaError_t cuda_error;
int n;

n = *N  ;
```

```
cublas_error = cublasCreate(&handleId);
if ( cublas_error != CUBLAS_STATUS_SUCCESS ) {
printf("ERROR initializing cublas, N= %d, n=%d\n", *N, n);
exit(-1);
} else {
printf("Initializing cublas: N= %d, n=%d\n", *N, n);
cuda_error = cudaMalloc( (void**)&dev_x, n * sizeof(double) );
//cuda_error = cudaMallocHost( (void**)&dev_x, n * sizeof(double) );
if ( cuda_error != cudaSuccess) {
printf("ERROR allocating dev_x of size: N= %d, n=%d\n", *N, n);
exit(-1);
}
cuda_error = cudaMalloc( (void**)&dev_y, n * sizeof(double) );
//cuda_error = cudaMallocHost( (void**)&dev_y, n * sizeof(double) );
if ( cuda_error != cudaSuccess) {
printf("ERROR allocating dev_y of size: N= %d, n=%d\n", *N, n);
exit(-1);
}
printf("dev_x =%p, dev_y=%p\n",dev_x, dev_y);
}



}

void daxpygpu_(int *Np, double *alpha, double *x, int *incx,
 double *y, int *incy, int *times)
```

```
{
cublasStatus_t error;
cudaEvent_t start,stop;
float time;
int i;


cudaEventCreate( &start );
cudaEventCreate( &stop );

cudaEventRecord( start, 0 );

error = cublasSetVector(*Np, sizeof(double), x, 1, (void*)dev_x, 1);
if ( error != CUBLAS_STATUS_SUCCESS ) {
printf("== ERROR setting x vector, N= %d\n", *Np);
exit(-1);
}

error = cublasSetVector(*Np, sizeof(double), y, 1, dev_y, 1);
if ( error != CUBLAS_STATUS_SUCCESS ) {
printf("== ERROR setting y vector, N= %d\n", *Np);
exit(-1);
}
cudaEventRecord( stop, 0 );
cudaEventSynchronize( stop );
cudaEventElapsedTime( &time, start, stop );
```

```
printf("Sending time : %f\n", time/1000.0);

cudaEventRecord( start, 0 );

for (i=1;i<= *times;i++){
error = cublasDaxpy(handleId, *Np, alpha, dev_x, *incx, dev_y, *incy);
if ( error != CUBLAS_STATUS_SUCCESS ) {
printf("== ERROR running cublasDaxpy, N= %d\n", *Np);
exit(-1);
}
}

cudaEventRecord( stop, 0 );
cudaEventSynchronize( stop );
cudaEventElapsedTime( &time, start, stop );
printf("Daxpy time : %f\n", time/1000.0);

cudaEventRecord( start, 0 );

error = cublasGetVector(*Np, sizeof(double), dev_y, 1, y, 1);

if ( error != CUBLAS_STATUS_SUCCESS ) {
printf("== ERROR getting y vector, N= %d\n", *Np);
exit(-1);
}
cudaEventRecord( stop, 0 );
cudaEventSynchronize( stop );
```

```
cudaEventElapsedTime( &time, start, stop );

printf("Receiving time : %f\n", time/1000.0);

// cudaFree(dev_x);
// cudaFree(dev_y);

   return;
}
}
```

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] 0. Axelsson and A. Barker, "Finite element solution of boundary value problems". *Theory and computation*, Academic Press, Orlando, Fl., 1984.

[4] G. Birkiff, M. H. Schultz and R. S. Varga, "Piecewise Hermite in one and two Variables with Applications to Partial Differential Equations", *Numer. Math.* 11, 232-256, 1968.

[5] C. de Boor and Swartz, "Collocation at Gaussian Points", *SIAM J. Numer. Anal.* 10, 582-606, 1973.

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

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

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

[9] 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.

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

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

[12] *code.google.com/p/cusp-library*.

[13] J. Douglas and T. Dupont, "Collocation Methods for Parabolic Equations in a Single Space Variable", *Springer-Verlag Lecture Notes 385*, Berlin/New York,1974.

[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] A. Hadjidimos, T.S. Papatheodorou and Y. G. Saridakis, "Optimal Block Iterative Schemes for Certain Large Sparse and Non-symmetric Linear Systems", *Linear Algebra Appl.*, 110, 285-318, 1988.

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

[18] E. N. Houstis, "Application of the method of Collocation on Lines for Solving Nonlinear Hyperbolic Problems",*Math. Comp.* 31, 443 - 456, 1977.

[19] E. N. Houstis, W. Mitchell, J. R. Rice, "Collocation Software for Second Order Elliptic Partial Differential Equations", *ACM Trans. Math. Software* 11, 379-412, 1985.

[20] Houstis, E. N.; Mitchell, W. F.; Rice, J. R. Algorithm 637. GENCOL: collocation on general domains with bicubic Hermite polynomials. ACM Trans. Math. Software 11 (1985), no. 4, 413-415.

[21] Houstis, E. N.; Mitchell, W. F.; Rice, J. R. Algorithm 638. INTCOL and HERMCOL: collocation on rectangular domains with bicubic Hermite polynomials. ACM Trans. Math. Software 11 (1985), no. 4, 416-418.

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

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

[24] Yu-Lin Lai, A. Hadjidimos, E. N. Houstis and J. R. Rice, "On the iterative solution of Hermite Collocation Equations", *SIAM J Mat. Analysics*, vol. 16 (1), pp. 254-277, 1995.

[25] 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.

[26] 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.

[27] 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.

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

[29] E. N. Mathioudakis and E. P. Papadopoulou, "Grid computing for Bi-CGSTAB applied to the solution of modified Helmholtz equation", *INt. J. Applied Maths and comp. sciences*, vol. 4, no. 3, pp. 179-184, 2007.

[30] T.S. Papatheodorou, "Inverses for a Class of Banded Matrices and Applications to Piecewise Cubic Approximation", *J. Comp. Appl. Math.* 8 (4), 285-288, 1982.

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

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

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

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

[35] R.S. Varga, "Matrix Iterative Analysis",*Prentice Hall*, Englewood Cliffs, NJ, 1962

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

[37] *www.culatools.com*.

[38] *www.khronos.org/opencl*.

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

[40] *www.pgroup.com*.

[41] *www-users.cs.umn.edu/ saad/software*.