ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΝΑΥΠΗΓΩΝ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΤΟΜΕΑΣ ΝΑΥΤΙΚΗΣ ΚΑΙ ΘΑΛΑΣΣΙΑΣ ΥΔΡΟΔΥΝΑΜΙΚΗΣ

# Παραλληλοποίηση κωδίκων επίλυσης προβλημάτων υπολογιστικής ρευστομηχανικής με χρήση του ανοιχτού προτύπου OpenCL

Διπλωματική εργασία

Φοιτητής: Γαλανός Δημήτριος

Επιβλέπων καθηγητής: Τζαμπίρας Γεώργιος

*Αθήνα, Ιούλιος 2018*

*(Η σελίδα αυτή είναι σκόπιμα κενή)*

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΝΑΥΠΗΓΩΝ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΤΟΜΕΑΣ ΝΑΥΤΙΚΗΣ ΚΑΙ ΘΑΛΑΣΣΙΑΣ ΥΔΡΟΔΥΝΑΜΙΚΗΣ

# Παραλληλοποίηση κωδίκων επίλυσης προβλημάτων υπολογιστικής ρευστομηχανικής με χρήση του ανοιχτού προτύπου OpenCL

Διπλωματική εργασία

Φοιτητής: Γαλανός Δημήτριος

Επιβλέπων καθηγητής: Τζαμπίρας Γεώργιος

*Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Δευτέρα 09 Ιουλίου 2018*

| | | |
|---|---|---|
| ……. | ……. | ……. |
| Τζαμπίρας Γεώργιος,<br>Καθηγητής<br>Σχολή Ναυπηγών<br>Μηχανολόγων Μηχανικών<br>Τομ. Ναυτικής & Θαλάσ.<br>Υδροδυναμικής | Πολίτης Γεράσιμος,<br>Καθηγητής<br>Σχολή Ναυπηγών<br>Μηχανολόγων Μηχανικών<br>Τομ. Ναυτικής & Θαλάσ.<br>Υδροδυναμικής | Τριανταφύλλου Γεώργιος,<br>Καθηγητής<br>Σχολή Ναυπηγών<br>Μηχανολόγων Μηχανικών<br>Τομ. Ναυτικής & Θαλάσ.<br>Υδροδυναμικής |

*Αθήνα, Ιούλιος 2018*

# Περίληψη

Σκοπός της Διπλωματικής Εργασίας είναι η διερεύνηση του κατά πόσον πραγματικοί αλγόριθμοι επίλυσης προβλημάτων υπολογιστικής ρευστομηχανικής μπορούν να επιταχυνθούν μέσω παραλληλοποίησης με χρήση σύγχρονου και φθηνού υλικού (*hardware*) και του ανοιχτού προτύπου/framework *OpenCL* (*Open Computing Language*). Το πρότυπο ορίζει ένα σύνολο δομών, από τη γλώσσα συγγραφής του κώδικα (βασισμένη στη *C99*), ως τις βιβλιοθήκες συναρτήσεων το API και τον αντίστοιχο compiler που επιτρέπουν την εκτέλεση του κώδικα στη συσκευή του κάθε κατασκευαστή χωρίς γνώση του υποκείμενου υλικού. Το πρότυπο είναι ανοιχτό, αναπτύσσεται με συνεργασία των μεγαλύτερων κατασκευαστών του κλάδου και υποστηρίζει πλήθος συσκευών όπως ενδεικτικά οικιακοί και server-grade επεξεργαστές (*CPUs*), επιταχυντές γραφικών (*GPUs*), επεξεργαστές τύπου *FPGA*, καθώς και chips τύπου *DSP*.

Αρχικά αναπτύχθηκε ένας απλός αλγόριθμος επίλυσης δις-διάστατης ροής ρευστού πάνω σε επίπεδη πλάκα, ξεκινώντας από τις εξισώσεις Navier-Stokes. Ολοκληρώνοντας αριθμητικά τις εξισώσεις αυτές πάνω σε όγκους ελέγχου ενός δομημένου πλέγματος και λαμβάνοντας υπ’ όψιν ορισμένες απλοποιήσεις, παράγουμε ένα σύστημα γραμμικών εξισώσεων, κατάλληλων για επίλυση σε ηλεκτρονικό υπολογιστή (Η/Υ). Το υπόβαθρο της θεωρίας καθώς και η μεθοδολογία του πλέγματος και της αριθμητικής ολοκλήρωσης που ακολουθήθηκε περιγράφονται στο βιβλίο *“An Introduction to Computational Fluid Dynamics – The Finite Volume Method (2nd edition)”* (H. K. Versteeg & W. Malalasekera - ISBN: 978-0-13-127498-3). Ορισμένες τροποποιήσεις του πλέγματος χρειάστηκε να γίνουν ώστε να είναι πιο άμεσος ο έλεγχος με αλγορίθμους που χρησιμοποιούνται από το Εργαστήριο Ναυτικής & Θαλάσσιας Υδροδυναμικής της σχολής Ναυπηγών Μηχανολόγων Μηχανικών ΕΜΠ.

Έλεγχοι της απόδοσης της παραλληλοποίησης πραγματοποιήθηκαν σε πλήθος συσκευών και αρχιτεκτονικών ικανών να υποστηρίξουν το πρότυπο, ενδεικτικά σε κλασσικούς “desktop” οικιακούς Η/Υ με πολυπύρηνους επεξεργαστές (υλοποιώντας τεχνολογίες *Simultaneous multi-threading* και υπαρκτούς παράλληλους πυρήνες), σε κάρτες γραφικών των κατασκευαστών AMD & Nvidia, και σύγχρονες φορητές συσκευές κινητής τηλεφωνίας (*smartphones*). Σε κάθε περίπτωση καταγράφεται το ποσοστό βελτίωσης (επιτάχυνσης) σε σύγκριση με την εκτέλεση της σειριακής έκδοσης του προγράμματος στην ίδια συσκευή (όπου αυτό ήταν δυνατόν) και γίνεται μια μικρή ανάλυση κόστους, αφενός ως προς το κόστος κτήσης, αφετέρου ως προς το κόστος λειτουργίας (κατά βάσιν ηλεκτρικής κατανάλωσης) με σκοπό να βοηθηθεί η επιλογή της κατάλληλης συσκευής, αναλόγως του φορτίου.

Όπως θα φανεί και στη συνέχεια, η επιτάχυνση του κώδικα είναι εμφανής και εκμεταλλεύσιμη, οι συσκευές που το υποστηρίζουν είναι πολλές, με σχεδόν κάθε σύγχρονο μικρο-επεξεργαστή να παρέχει υποστήριξη στο υλικό του, οι υποστηριζόμενες συσκευές καλύπτουν ένα ευρύτατο φάσμα τεχνολογιών, ενώ μπορούμε να συνθέσουμε ένα σύνολο συσκευών, υπολογιστών ή δικτύου που θα δουλεύουν ταυτόχρονα, ανάλογα με τις ανάγκες μας σε συνδυασμό με τη διαθέσιμη χρηματική δαπάνη. Τέλος, η πορεία της βιομηχανίας επεξεργαστών προς πολυπύρηνους επεξεργαστές παντός είδους (*heterogeneous computing*) μας προσφέρει πολύ μειωμένες καταναλώσεις (*“performance per watt”*), ιδιαίτερα για προβλήματα εκ φύσεως εύκολα παραλληλοποιήσιμα όπως εν προκειμένω.

# Πίνακας περιεχομένων

## 1.1 Γενικά περί παράλληλου προγραμματισμού

Ο πραγματικός κόσμος είναι εγγενώς *παράλληλος*: Από την κίνηση των πλανητών, των αεροπλάνων, αλλά και των οχημάτων σε μια μεγάλη πόλη, έως τη μορφή και εξέλιξη της ατμόσφαιρας, την κίνηση των υδάτινων μαζών του ωκεανού αλλά και της ροής του αίματος σε ένα ανθρώπινο σώμα, όλα τα πραγματικά φαινόμενα συμβαίνουν παράλληλα μεταξύ τους. Πολλά από τα πιο σύγχρονα μοντέλα που διαθέτουμε για την περιγραφή μιας πληθώρας φυσικών φαινομένων όπως και των προαναφερθέντων στηρίζονται στην εκτέλεση πράξεων ή ενεργειών πάνω σε ένα σύνολο (ανεξάρτητων) *σημείων* του εκάστοτε *χώρου εργασίας*. Τα σημεία αυτά μπορεί να αντιπροσωπεύουν μόρια (μοντελοποίηση ατμόσφαιρας & λοιπών ρευστών), πληροφορίες (μοντέλα πρόβλεψης οικονομικών δεικτών), ή απλά αδιάστατους αριθμούς (για παράδειγμα πράξεις σε πίνακες).

Η κλασσική μέθοδος συγγραφής αλγορίθμων και προγραμμάτων επίλυσης είναι εγγενώς *σειριακή*: Το πρόβλημα προς επίλυση χωρίζεται σε διαχειρίσιμα υποπροβλήματα (που αντιστοιχούν σε functions, subroutines κλπ, ανάλογα την ονοματολογία της γλώσσας προγραμματισμού), τα οποία με τη σειρά τους αναλύονται στις βασικές εντολές της εκάστοτε γλώσσας προγραμματισμού, ώστε το ολοκληρωμένο πρόγραμμα αποτελείται από ένα σύνολο αυτών, και στην ολότητά του επιλύει το αρχικό πρόβλημα. Οι εντολές αυτές εκτελούνται *μια προς μια* σε μια κεντρική μονάδα επεξεργασίας (*CPU*), και η ροή του προγράμματος είναι απολύτως προβλέψιμη (*deterministic*): Οποιαδήποτε στιγμή μπορούμε να γνωρίζουμε ποια εντολή εκτελείται, ποιες είναι οι τιμές στα μητρώα του επεξεργαστή (*registers*) ποια εντολή ακολουθεί, κλπ.

Υπάρχουν βεβαίως πολλοί τρόποι επιτάχυνσης της εκτέλεσης ενός σειριακού προγράμματος σε ηλεκτρονικό υπολογιστή (Η/Υ). Η πιο προφανής είναι η βελτιστοποίηση του ίδιου του αλγορίθμου. Για παράδειγμα η ταξινόμηση στοιχείων ενός πίνακα ανάλογα με την τιμή τους έχει πολλές διαφορετικές μεθόδους επίλυσης, με μεγάλη διαφορά στο χρόνο εκτέλεσής τους. Άλλες μέθοδοι μπορεί να είναι ταχύτερες στην εκτέλεση εις βάρος του κόστους σε μνήμη, άλλες μέθοδοι μπορεί να δουλεύουν καλύτερα σε μεγαλύτερους πίνακες έναντι μικρότερων, κλπ. Σε κάθε περίπτωση, η αλγοριθμική βελτίωση του προγράμματος είναι συνήθως αρκετά δύσκολη, ενώ σε ώριμους ή και απλούς αλγορίθμους έχει πρακτικά φτάσει στο άνω φράγμα της, το οποίο πολλές φορές προσδιορίζεται και θεωρητικά.

Άλλη μέθοδος επιτάχυνσης της σειριακής εκτέλεσης προγράμματος αποτελεί πάντα η αγορά ταχύτερου υλικού (hardware). Δεδομένου του κόστους αυτό δεν είναι πάντα θεμιτό ή δυνατόν, ενώ αποτελεί σπατάλη του ήδη υπάρχοντος υλικού. Σαν μέθοδος εξάλλου ήταν ίσως πιο πρόσφορη σε προηγούμενα χρόνια οπότε και η κάθε γενιά επεξεργαστών έφερνε μεγάλες βελτιώσεις στην ταχύτητα κυρίως (αλλά όχι μόνο) λόγω αύξησης της βασικής συχνότητας λειτουργίας τους (*core clock frequency*). Τώρα όπως αποδεικνύεται και από τη στροφή της βιομηχανίας σε πολυπύρηνους επεξεργαστές, η συχνότητα είναι ένας δείκτης που έχει παραμείνει στάσιμος λόγω θερμοδυναμικών ορίων, και οι βελτιώσεις βρίσκονται σε άλλα σημεία της αρχιτεκτονικής (ταχύτητα και ποσότητα cache, pipeline depth, instructions per clock, κλπ).

Μια ακόμη ενδεικτική μέθοδος για να επιταχύνουμε ένα σειριακό πρόγραμμα είναι η απλοποίηση των υπολογισμών. Για παράδειγμα, με χρήση μεταβλητών μικρότερης ακρίβειας είναι δυνατόν η κάθε εντολή να εκτελείται ταχύτερα, επομένως όπου η μειωμένη ακρίβεια δεν αλλάζει σημαντικά τα αποτελέσματα, αυτή είναι μια απτή μέθοδος επιτάχυνσης.

Ακόμα και με τις παραπάνω ενδεικτικές μεθόδους επιτάχυνσης, προκύπτει και σήμερα η ανάγκη περαιτέρω μείωσης του χρόνου εκτέλεσης πολλών προγραμμάτων που αναλύουν φυσικές ή μη διεργασίες. Αυτή η ανάγκη μεγεθύνεται και από τη στροφή πολλών κλάδων επεξεργασίας δεδομένων σε τόσο μεγάλα ή ταχέως μεταβαλλόμενα σύνολα δεδομένων που θα ήταν αδύνατη ή άκαιρη η επεξεργασία τους με κλασσικές μεθόδους (*big data sets*). Η ακρίβεια πολλών φυσικών μοντέλων βασίζεται στη *λεπτή* διακριτοποίηση του χώρου εργασίας τους παράγοντας επίσης πολύ μεγάλο όγκο δεδομένων. Καθίσταται πλέον σαφές ότι απαιτείται να γίνει εκμετάλλευση της παραλληλίας που υπάρχει εκ φύσεως σε πολλά προβλήματα πρακτικού και επιστημονικού ενδιαφέροντος ώστε να προκύπτουν και σήμερα βελτιωμένες, έγκαιρες και ακριβείς απαντήσεις. Παράλληλη επεξεργασία επομένως ορίζεται η χρήση δύο η περισσότερων επεξεργαστικών μονάδων (πυρήνων, επεξεργαστών, υπολογιστών) ταυτόχρονα.

Δεν είναι βέβαια όλα τα προβλήματα άμεσα, εύκολα ή εν τέλει έστω και λίγο παραλληλοποιήσιμα. Εάν μια εργασία μπορεί να χωριστεί σε έναν τυχαίο αριθμό υπό-εργασιών, που όλες απαιτούν τον ίδιο χρόνο, και είναι ανεξάρτητες μεταξύ τους, τότε η εργασία μπορεί προφανώς να παραλληλοποιηθεί. Αυτό είναι τόσο εύκολο, ώστε προβλήματα που ανήκουν σε αυτήν την κατηγορία ονομάζονται *“embarrassingly parallel” (από το “υπερβολικά” - exceedingly παράλληλα)*, τονίζοντας το γεγονός ότι η λογική επίλυσης μπορεί με μικρές ή και καθόλου αλλαγές να μεταφερθεί σε ένα παράλληλο (πολυεπεξεργαστικό) περιβάλλον, και το όφελος σε επιτάχυνση είναι ανάλογο του αριθμού των πυρήνων ή επεξεργαστών που διαθέτουμε. Παράδειγμα αυτής της κατηγορίας αποτελεί η προσπάθεια εύρεσης του κρυπτογραφικού κλειδιού για ένα κρυπτογραφημένο κείμενο (ciphertext) δοκιμάζοντας όλο το εύρος των πιθανών συνδυασμών, οι οποίοι ως ανεξάρτητοι μεταξύ τους μπορούν να δοκιμάζονται παράλληλα.

Από την άλλη υπάρχουν προβλήματα που δεν επιδέχονται καμία επιτάχυνση κατά τη μεταφορά τους στο αντίστοιχο παράλληλο περιβάλλον. Αυτό συμβαίνει συνήθως διότι υπάρχει αλληλεξάρτηση των εντολών ή/και των αποτελεσμάτων του προγράμματος σε πολλά σημεία μεταξύ τους ή απαιτείται πολύ συχνός συγχρονισμός. Για παράδειγμα, ο κλασσικός (δηλαδή με βάση τον ορισμό) τρόπος υπολογισμού της σειράς Fibonacci προϋποθέτει τον υπολογισμό ενός αριθμού *προτού* υπολογιστεί ο επόμενος αφού για τον υπολογισμό του επόμενου, απαιτείται η τιμή του πρώτου. Επομένως η εκτέλεση των δύο αυτών εντολών υπολογισμού δεν μπορεί να γίνει ταυτόχρονα. Σε προσπάθεια παραλληλοποίησης τέτοιων προγραμμάτων ενδέχεται αντί για επιτάχυνση να έχουμε έως και μείωση των επιδόσεων που οφείλεται στο υπαρκτό επιπλέον επεξεργαστικό κόστος του να δημιουργηθούν οι δομές δεδομένων που θα επέτρεπαν την παράλληλη επεξεργασία.

Τα περισσότερα προβλήματα εμπίπτουν κάπου μεταξύ των δύο ακραίων παραδειγμάτων που αναφέρθηκαν. Ο νόμος του Amdahl μπορεί να μας δώσει το θεωρητικό ποσοστό επιτάχυνσης ενός προγράμματος από την παραλληλοποίησή και μεταφορά του σε ένα παράλληλο περιβάλλον. Για παράδειγμα εάν ένα πρόγραμμα απαιτεί 10 ώρες για την ολοκλήρωσή του, και η μια εξ αυτών καταναλώνεται για εκτέλεση ενός τμήματος που δεν μπορεί να παραλληλοποιηθεί, τότε ανεξάρτητα

από τον αριθμό των επεξεργαστικών πυρήνων που θα διαθέσουμε στο πρόγραμμα, η επιτάχυνση που θα πετύχουμε δεν μπορεί να είναι μεγαλύτερη από *10x*, αφού ο χρόνος εκτέλεσης δεν μπορεί να γίνει μικρότερος από 1 ώρα.

Ένας χρήσιμος διαχωρισμός παραλληλοποιήσιμων προγραμμάτων έχει να κάνει με *παράλληλα κατά ενέργεια (task-parallel)* και *παράλληλα κατά τα δεδομένα (data-parallel)* προβλήματα. Όπως ετυμολογικά μπορεί να αντιληφθεί κανείς, *παράλληλα κατά ενέργεια* προβλήματα επωφελούνται από τη δημιουργία πολλών ανεξάρτητων *νημάτων εκτέλεσης (threads)* δηλαδή ανεξάρτητων ακολουθιών εντολών που εκτελούνται ταυτόχρονα σε διαφορετικούς πυρήνες, επεξεργαστές, ή υπολογιστές ενώ συγχρονίζονται ή/και ανταλλάσσουν δεδομένα κάθε όποτε αυτό απαιτείται. Η παραλληλοποίησή τους είναι συνήθως πιο δύσκολη ενώ χρειάζονται πλήρεις επεξεργαστικούς πυρήνες για την εκτέλεση των ανεξάρτητων νημάτων, και επομένως δεν μπορούν να επιταχυνθούν από την παρουσία εξειδικευμένων επιταχυντών τύπου *SIMD* (*Single Instruction Multiple Data*) όπως για παράδειγμα καρτών γραφικών GPU, οι οποίοι είναι συνήθως κατά πολύ φθηνότεροι ανά μονάδα επεξεργαστικής ισχύος.

*Παράλληλα κατά τα δεδομένα* προβλήματα είναι αυτά που εκτελούν τις ίδιες συγκεκριμένες πράξεις ή ενέργειες σε ένα σύνολο δεδομένων που αποτελεί το χώρο εργασίας. Οι πράξεις μπορεί να διαφοροποιούνται ως προς τα αριθμητικά αποτελέσματα, αλλά η εντολή στο μικρο-επεξεργαστή είναι η ίδια για κάθε σημείο (για παράδειγμα η εντολή “κάθε σημείο του χώρου εργασίας να πολλαπλασιαστεί με το αντίστοιχο στοιχείο ενός ήδη υπάρχοντος πίνακα” είναι παραλληλοποιήσιμη στο σύνολο των σημείων του χώρου εργασίας). Τέτοια προβλήματα προκύπτουν συνήθως από την επεξεργασία των προαναφερθέντων big data sets, είναι εύκολα παραλληλοποιήσιμα, ενώ επωφελούνται από την παρουσία μονάδων τύπου SIMD όπως αυτές που περιέχονται σε κάθε σύγχρονο επεξεργαστή (CPU) για αυτόν ακριβώς το σκοπό (για παράδειγμα εντολές SSE σε Intel επεξεργαστές).

## 1.2 Το υλικό

Παρόλο που το υλικό των σύγχρονων υπολογιστών δεν έχει αλλάξει ως προς το θεωρητικό του μοντέλο, εντούτοις έχουμε σήμερα στη διάθεσή μας πολλές καινοτομίες, όπως πολλαπλά instruction sets, πολυπύρηνους επεξεργαστές, τεχνολογίες τύπου Simultaneous ή hardware multithreading, μονάδες SIMD εντός των επεξεργαστών, βοηθητικούς επεξεργαστές (co-processors) τύπου GPU, FPGA, ASIC, ενοποιημένη μνήμη (*unified address space*) μεταξύ των επεξεργαστών (με αποτέλεσμα τη μείωση των εντολών μεταφοράς δεδομένων) καθώς και πληθώρα άλλων τεχνολογιών που μπορούν να επιταχύνουν την εκτέλεση ενός προγράμματος. Επίσης η ποικιλία των αρχιτεκτονικών επεξεργαστών αλλά και των ειδών των συσκευών διαθέσιμων σήμερα είναι σημαντικά αυξημένες. Καθίσταται επομένως άξια η προσπάθεια επανεγγραφής παραλληλοποιήσιμων προγραμμάτων με σκοπό να εκμεταλλευτούμε κάθε πυρήνα επεξεργασίας και κάθε συσκευή που διαθέτουμε σήμερα.

Εάν επιχειρήσουμε να κατατάξουμε τους διαθέσιμους σήμερα “οικιακούς” μικρο επεξεργαστές ως προς την επεξεργαστική τους ισχύ ανά μονάδα κόστους, σύγχρονοι επεξεργαστές τύπου GPU σίγουρα θα βρίσκονται κοντά στην κορυφή της λίστας. Το πως κατέληξαν επεξεργαστές απόδοσης

γραφικών σε οθόνη να μπορούν να χρησιμοποιηθούν για γενικούς υπολογισμούς (*GPGPU – General Purpose computing on GPUs*) και μάλιστα με τόσο μεγάλη επεξεργαστική ισχύ είναι εξαιρετικά ενδιαφέρον δεδομένης και της πολύχρωμης ιστορίας των συσκευών αυτών. Πρόκειται για πολυπύρηνους streaming processors που αρχικά αναπτύχθηκαν για την επιτάχυνση της παραγωγής εικόνας για απόδοσή της στην οθόνη σε σύνθετα γραφικά περιβάλλοντα όπως παιχνίδια 3D, και εμφανίστηκαν στη σύγχρονη μορφή τους (ως πρόσθετοι co-processors με τη μορφή "κάρτας") τη δεκαετία του 1990 για προσωπικούς υπολογιστές (PC). Co-processors παντός είδους υπήρχαν βεβαίως πολύ πριν από τότε και δη για επιτάχυνση γραφικών σε κονσόλες παιχνιδιών ήδη από τη δεκαετία του 1980. Τα πιο σύγχρονα μοντέλα σήμερα διαθέτουν επεξεργαστική ισχύ της τάξης των 10,000 GFLOPS για single-precision floating point arithmetic (*AMD RX Vega 64* (98)).

Το Close to Metal (CTM) API, ή όπως αρχικά ήταν γνωστό, THIN (Thin Hardware Interface), ήταν η πρώτη ίσως "*production*" προσπάθεια της ATi (τώρα πλέον AMD Graphics division/AMD Graphics Product Group) να επιτρέψει πρόσβαση των προγραμματιστών σε "*low level functionality*" (πιθανόν πλήρη πρόσβαση στο instruction set) των GPU chips που χρησιμοποιούσε τότε (R580 GPU, παρόν στην κάρτα ATi X1900) με σκοπό εργασίες διάφορες της απόδοσης γραφικών που ήδη έδιναν άλλα APIs όπως DirectX και OpenGL, έχοντας αντιληφθεί την μεγάλη τους επεξεργαστική ισχύ (Η AMD εξέδωσε press release στις 14/11/2006, το οποίο ευτυχώς διασώζεται χάρη στο αγαπητό Internet Archive (98)). Το CTM, ιδιαίτερα σημαίνον όπως τελικά φάνηκε στη διαμόρφωση της μετέπειτα ιστορίας του GPGPU, μετονομάστηκε έπειτα σε AMD Stream SDK (κατ' αντιστοιχία μάλλον με την επαγγελματική σειρά καρτών γραφικών/co-processors AMD FireStream), και επέτρεπε των προγραμματισμό των GPUs της μέσω μιας βελτιωμένης έκδοσης της γλώσσας "*Brook*" (προερχόμενη από τη γλώσσα C, αναπτυγμένη και βελτιστοποιημένη για χρήση σε παράλληλους stream processors από το πανεπιστήμιο του Stanford), που ονομάστηκε "*Brook+*". Ακολούθως, η AMD επέλεξε να υποστηρίξει την OpenCL μέσω του "*AMD APP SDK*" (*Accelerated Parallel Processing Software Development Kit*).

Αντίστοιχη πορεία είχαμε (όπως συνήθως συμβαίνει) και με τον κύριο ανταγωνιστή του πεδίου των GPUs, Nvidia, αναπτύσσοντας το δικό της framework με την ονομασία "*CUDA*" (*Compute Unified Device Architecture*). Η Nvidia είχε τότε όπως ίσως και σήμερα μεγαλύτερο μερίδιο αγοράς απ' ότι η AMD στο συγκεκριμένο χώρο, και μια φήμη για καλύτερη υποστήριξη του υλικού της, πιο σταθερούς drivers κλπ. Αυτό σε συνδυασμό ίσως και με το γεγονός της εναλλαγής υποστήριξης της AMD για διάφορα πρότυπα μη δίνοντας σαφή εντύπωση για το ποιο είναι το τρέχον, οδήγησε στην ευρύτερη διάδοση του CUDA στο πεδίο του GPGPU και των αντίστοιχων εμπορικών προγραμμάτων που είχαν ήδη αρχίσει να κάνουν την εμφάνισή τους (χαρακτηριστικά για εργασίες όπως video encoding). Το πρότυπο της Nvidia όμως δεν μπορεί να χρησιμοποιηθεί για συσκευές άλλου κατασκευαστή.

Είναι χαρακτηριστικό ότι ένα μεγάλο ποσοστό του τζίρου της Nvidia προέρχεται από πωλήσεις της σειράς προϊόντων της "*Tesla*" που αποτελούν αυτόνομοι co-processors ή πλήρη συστήματα για GPGPU stream processing χρησιμοποιώντας τα ίδια εν πολλοίς chips που χρησιμοποιούνται στην "οικιακή" σειρά καρτών γραφικών της, *GeForce*! Η ίδια η πορεία της εταιρίας έχει πλέον ξεφύγει από τα στενά όρια του *consumer graphics processing*, αναπτύσσοντας εφαρμογές από high performance computing (HPC), GPGPU και υλικό για supercomputers, έως ρομποτική και Artificial Intelligence (*AI*). Ακόμη πιο ενδιαφέρον είναι το γεγονός ότι για τους ταχύτερους

supercomputers στον κόσμο (ο υπερυπολογιστής "*Summit*", καταλαμβάνοντας την πρώτη θέση της λίστας *Top500* από 8 Ιουνίου 2018 έχει ήδη σπάσει το φράγμα του *exaflop* χρησιμοποιώντας μάλιστα, μεταξύ άλλων, και Nvidia V100 (Tesla) co-processors!), το ποσοστό των *floating point operations per second* (*FLOPS*) που προέρχονται από GPU chips έχει κατά πολύ ξεπεράσει αυτό των προερχόμενων από κλασσικούς CPU, τάση που αναμένεται να ενισχυθεί ακόμη περισσότερο στο μέλλον (98). Τέλος, η περίοδος που διανύουμε αποτελεί μια μάλλον ατυχή περίοδο για φανατικούς gamers καθώς η διαθεσιμότητα των πιο σύγχρονων καρτών γραφικών δοκιμάζεται υπό το βάρος της εξαιρετικά αυξημένης ζήτησης που προκαλεί το "*crypto currency mining*", μιας χρήσης των δυνατοτήτων της μαζικής παράλληλης επεξεργασίας των GPUs για υποστήριξη του παγκόσμιου *blockchain-type* οικοσυστήματος των *crypto coins*, αποφέροντας κέρδη στους συμμετέχοντες. Σίγουρα ελάχιστοι θα μπορούσαν να είχαν προβλέψει τέτοιου είδους χρήση όταν τον Ιανουάριο του 2009 δημοσιεύτηκε το αντίστοιχο whitepaper περιγράφοντας τη λειτουργία του πρώτου crypto coin, *Bitcoin* (98).

Όσον αφορά στους επεξεργαστές τύπου CPU, κάθε σύγχρονος consumer ή server grade επεξεργαστής από τους μεγαλύτερους αυτήν τη στιγμή κατασκευαστές (Intel & AMD) διαθέτει πολλαπλούς πυρήνες. Επιπλέον αυτού, οι περισσότεροι διαθέτουν τεχνολογίες τύπου Simultaneous ή hardware multithreading για τη δημιουργία πολλαπλών νημάτων εκτέλεσης από το λειτουργικό σύστημα τα οποία εκτελούνται εικονικά σε περισσότερους πυρήνες απ' ότι υπάρχουν στην πραγματικότητα, εκμεταλλευόμενα τον αδρανή χρόνο σε ορισμένα τμήματα του επεξεργαστή. Σύγχρονοι επεξεργαστές στην αγορά σήμερα διαθέτουν ως και 32 πυρήνες και 64 threads (πχ *AMD Threadripper* 2ης γενιάς με 64 εικονικούς πυρήνες εκτέλεσης 98). Φυσικά δεν γίνεται εδώ λόγος για πιο σύνθετες και εξειδικευμένες αρχιτεκτονικές όπως αυτές που κατασκευάζονται κατά παραγγελία (πχ για υπερ-υπολογιστές κρατικών προδιαγραφών).

Εκτός όμως της υψηλής επεξεργαστικής ισχύος που μας προσφέρει το σύγχρονο υλικό Η/Υ, αυτή η ισχύς έρχεται και με πολύ χαμηλότερο κόστος. Αφενός το κόστος κτήσης έχει μειωθεί ανά μονάδα επεξεργαστικής ισχύος (όπως είναι αναμενόμενο) αλλά και το κόστος λειτουργίας (κατανάλωσης) παρουσιάζεται ιδιαίτερα μειωμένο καθώς αποτελεί διαρκή στόχο της βιομηχανίας και χαρακτηριστικό που, με τη στροφή σε *mobile computing*, επιδιώκεται από τη σημερινή αγορά περισσότερο από ποτέ. Συσκευές με επεξεργαστές αρχιτεκτονικής ARM όπως ενδεικτικά η σειρά Raspberry Pi αποτελούν *πλήρεις* Η/Υ ικανότατων επιδόσεων με ελάχιστο κόστος κτήσης (~32€ μετά φόρων, στην ελληνική αγορά, *τιμές Ιουνίου 2018*), πολύ μειωμένη κατανάλωση (~5Watt), καταλαμβάνουν ελάχιστο χώρο, δεν προϋποθέτουν δαπάνες κλιματισμού, ενώ παράλληλα η αύξηση της επεξεργαστικής τους ισχύος όπως αποτυπώνεται και παρακάτω, είναι ραγδαία.

*Πίνακας 1: Σύγκριση χαρακτηριστικών υπολογιστών της σειράς "Raspberry Pi"*

| Συσκευή | Επεξεργαστική ισχύς [αριθμοί κατά προσέγγιση] (single precision GFLOPS) | Κατανάλωση (Watt) | Κόστος αγοράς στην Ελλάδα, μετά φόρων, Ιούνιος 2018 (€) | Ημερομηνία κυκλοφορίας |
|---|---|---|---|---|
| Raspberry Pi Model B | 0.041 (CPU) | 3.0 | [-] | Ιούνιος 2012 |
| Raspberry Pi 2 Model B | 1.47 (CPU) | 3.6 | [-] | Φεβρουάριος 2015 |
| Raspberry Pi 3 Model B | 3.62 (CPU) | 4.8 | 32 | Φεβρουάριος 2016 |

Πολλοί επεξεργαστές υλοποιούν εσωτερικά (στο hardware) επεξεργασία διανυσμάτων (vector processing). Πολλοί κατασκευαστές το ονομάζουν αυτό superscalar processing. Διάνυσμα στην περίπτωση αυτή έχει την έννοια της διατεταγμένης ν-άδας ίδιου τύπου μεταβλητών, και για επεξεργαστές που υλοποιούν αυτό το χαρακτηριστικό, με χρήση των κατάλληλων εντολών είναι δυνατό να επεξεργαστούμε *ταυτόχρονα* όλα τα στοιχεία του διανύσματος στον ίδιο κύκλο επεξεργασίας. Με το πρότυπο OpenCL όπως θα δούμε παρακάτω, η δημιουργία και επεξεργασία διανυσμάτων αποχωρίζεται από τις συγκεκριμένες εντολές του instruction set του κάθε επεξεργαστή και καθίσταται δυνατή με τρόπο φορητό σε όλες τις συσκευές που το υποστηρίζουν.

Η τάση της σημερινής ευρύτερης αγοράς Η/Υ είναι η στροφή προς πολλές μικρές "έξυπνες" συσκευές. Πλέον διαθέτουμε συσκευές που μπορούν να χαρακτηριστούν Η/Υ σε ρολόγια, ψυγεία, πλυντήρια, συναγερμούς, εκτυπωτές, αισθητήρες παντός είδους, οχήματα, στύλους ηλεκτροφωτισμού, φωτεινούς σηματοδότες, κινητά τηλέφωνα, και πλήθος άλλων συσκευών. Αν και πολλές από τις συσκευές αυτές δεν μπορούν (εύκολα) να προγραμματιστούν για διαφορετικό σκοπό από αυτόν που έχουν παραχθεί, σίγουρα υπάρχουν πλέον στη διάθεσή μας πολλές περισσότερες μονάδες επεξεργασίας απ' ότι στο παρελθόν, συχνά εκεί που δεν το φανταζόμαστε, με τον αριθμό τους να αυξάνεται διαρκώς.

## 1.3 Το πρότυπο OpenCL

Η αμερικάνικη εταιρεία *Apple Inc.* είναι περισσότερο γνωστή, πέραν από την εμμονή των οπαδών της, για τη δημιουργία των πολύ επιτυχημένων σειρών συσκευών *Macintosh* (Η/Υ), *iPod* (φορητή συσκευή αναπαραγωγής πολυμέσων), *iPhone* (κινητό τηλέφωνο τύπου smartphone), & *iPad* (tablet PC). Έχει τεράστιο μερίδιο αγοράς εντός και εκτός ΗΠΑ. Όμως η Apple δεν κατασκευάζει τους επεξεργαστές που χρησιμοποιεί στις συσκευές της, οι οποίοι προέρχονται από λοιπούς τρίτους κατασκευαστές όπως Intel, AMD, IBM, κλπ. Για κάθε νέα σειρά ή μοντέλο, επιλέγεται ο επεξεργαστής που κρίνεται καταλληλότερος τη στιγμή εκείνη.

Το 2008 η Apple εκμεταλλευόμενη την δεσπόζουσα θέση της στην αγορά, ηγήθηκε της προσπάθειας να συντεθεί ένα πρότυπο το οποίο θα επέτρεπε τον αποχωρισμό του προγραμματισμού των συσκευών της από το υποκείμενο υλικό, επιτρέποντας στους προγραμματιστές (developers) που ασχολούνται με το οικοσύστημα των συσκευών και λειτουργικών συστημάτων της να μην χρειάζεται να γνωρίζουν τις λεπτομέρειες υλικού για κάθε διαφορετικό επεξεργαστή για τον οποίο έγραφαν κώδικα. Το πρώτο προσχέδιο του προτύπου προέκυψε λιγότερο από έναν χρόνο αργότερα. Για την ανάπτυξη του προτύπου συντέθηκε το OpenCL Working Group, το οποίο είναι μια από τις πολλές ομάδες του Khronos Group, μιας κοινοπραξίας εταιρειών για την έρευνα και ανάπτυξη στο χώρο των γραφικών και πολυμέσων. Τα κύρια χαρακτηριστικά που κάνουν ελκυστική τη σύνταξη προγραμμάτων σε OpenCL είναι η παράλληλη επεξεργασία, φορητότητα, και χρήση διανυσμάτων για ταχύτερη επεξεργασία μεγάλου όγκου δεδομένων.

Δεν θα γίνει εδώ εκτενής περιγραφή του προτύπου, του τρόπου λειτουργίας, ή λοιπών πλεονεκτημάτων και μειονεκτημάτων που τυγχάνει. Θα αναφερθεί μόνο το γεγονός που αφορά τη συγκεκριμένη εργασία και είναι ότι μέσω του προγραμματισμού σε OpenCL και της φορητότητας που το πρότυπο αυτό προσφέρει είμαστε σε θέση να εκμεταλλευτούμε όλες τις συσκευές που το υλοποιούν από τον κατασκευαστή τους, προσφέροντας μας μια ευρεία γκάμα από υπερσύγχρονους και ταχύτατους ως πολύ αποδοτικούς, μικρούς και ευέλικτους μικρο-επεξεργαστές, σε πλήθος συσκευών όπως ενδεικτικά από κλασσικούς desktop και server-grade Η/Υ, FPGA’s, GPU’s, Smartphones, και single-board computers τύπου Raspberry Pi, προγραμματίζοντας σε μια ενιαία γλώσσα, και χρησιμοποιώντας τις ίδιες δομές δεδομένων.

## 2.1 Θεωρία του προβλήματος

Η υπολογιστική ρευστομηχανική γνωρίζει εδώ και αρκετό καιρό μεγάλη ανάπτυξη, λόγω του ότι ο μεγάλος φόρτος εργασίας που προκύπτει μπορεί εύκολα να ανατεθεί σε ηλεκτρονικούς υπολογιστές (Η/Υ) προς επίλυση, έχοντας κανείς συγγράψει το κατάλληλο λογισμικό. Προφανώς η ανάθεση σε Η/Υ προϋποθέτει πεπερασμένη ακρίβεια των υπολογισμών που προκύπτουν και από εκεί προέρχεται και ο όρος *υπολογιστική* ρευστομηχανική. Σε κάθε περίπτωση όμως η ακρίβεια που μπορούμε να έχουμε είναι υπέρ-αρκετή για τις περισσότερες εφαρμογές ενώ μπορούμε να έχουμε και αύξηση ακρίβειας (εις βάρος του χρόνου ολοκλήρωσης του προγράμματος) εάν αυτό απαιτείται.

Όπως γνωρίζουμε οι εξισώσεις Navier-Stokes σε πλήρη μορφή δίνουν την ακριβή επίλυση ενός πεδίου ροής ιξώδους ρευστού. Για τη μεταφορά και επίλυση των εξισώσεων αυτών στον υπολογιστή είναι απαραίτητη η *αριθμητική ολοκλήρωση* τους, ώστε από διαφορικές να καταλήξουμε σε ένα σύστημα γραμμικών αλγεβρικών εξισώσεων για την επίλυση του πεδίου ροής. Μια συνήθης μέθοδος ολοκλήρωσης, καθώς και ίσως η πιο οικεία, είναι η ολοκλήρωση σε *όγκους ελέγχου*. Αυτή προϋποθέτει τη δημιουργία ενός *πλέγματος*, διαστάσεων ίδιων με το *χώρο εργασίας* μας (μονο- δισ- τρισ- διάστατο) που θα χωρίζει το χώρο εργασίας σε διακριτούς *όγκους ελέγχου*. Για τους όγκους αυτούς μπορούμε να κάνουμε ορισμένες απλοποιητικές παραδοχές κατά την ολοκλήρωση των εξισώσεων, και εάν το πλέγμα που έχουμε επιλέξει είναι αρκούντως πυκνό, τα αποτελέσματα κατά την επίλυση του πεδίου ροής θα είναι πολύ κοντά στην πραγματικότητα.

Έχοντας ολοκληρώσει τις εξισώσεις Navier-Stokes σε όγκους ελέγχου ενός πλέγματος καλούμαστε να υπολογίσουμε (ανάλογα και με το σχήμα διαφορών που έχουμε επιλέξει βλ. βιβλιογραφία) ορισμένους αδιάστατους συντελεστές για κάθε όγκο ελέγχου. Οι συντελεστές αυτοί προκύπτουν *ανεξάρτητα για κάθε όγκο ελέγχου*. Η ανεξαρτησία της παραγωγής των συντελεστών αυτών είναι το βασικό γεγονός το οποίο επιτρέπει την παραλληλοποίηση του υπολογισμού τους, και καθιστά το τμήμα αυτό του προγράμματος *"embarrassingly parallel"* δεδομένου ότι η επιτάχυνση σε αυτό το κομμάτι μπορεί να είναι ανάλογη του αριθμού των επεξεργαστικών μονάδων που διαθέτουμε.

Για την εύρεση του νέου πεδίου ροής είναι απαραίτητη ακολούθως η επίλυση ενός τρι-διαγώνιου, πεντα-διαγώνιου ή επτα-διαγώνιου συστήματος γραμμικών εξισώσεων (για μονοδιάστατα, δισδιάστατα ή τρισδιάστατα πλέγματα αντιστοίχως) η παραλληλοποίηση της οποίας δεν πραγματοποιείται στην παρούσα υλοποίηση για λόγους που θα εξηγηθούν παρακάτω. Η επίλυση αυτή μπορεί να γίνει είτε επαναληπτικά (ενδεικτικά: μέθοδοι *Gauss-Seidel* & *Jacobi*) είτε ακριβώς με τη μέθοδο επίλυσης *Gauss*.

Η εύρεση του τελικού πεδίου ροής γίνεται (χονδροειδώς) επαναλαμβάνοντας την παραπάνω διαδικασία σε συνδυασμό με την επίλυση του πεδίου της πίεσης μέσω του αλγορίθμου *SIMPLE (Semi-Implicit Method for Pressure Linked Equations)* συγκλίνοντας στην τελική λύση.

## 2.2 Γλώσσες προγραμματισμού

Για τους σκοπούς της παρούσας Διπλωματικής εργασίας συντάχθηκαν 3 διαφορετικά προγράμματα αντιμετώπισης του ίδιου προβλήματος – επίλυση πεδίου ροής για ροή ιξώδους ρευστού πάνω σε επίπεδη πλάκα. Μπορεί κατά περίπτωση τα προγράμματα αυτά να τροποποιούνται ώστε να βλέπουμε πως άλλες αλλαγές (πέραν την παραλληλοποίησης ή της διαφορετικής γλώσσας) επηρεάζουν την ταχύτητα εκτέλεσης.

Το πρώτο πρόγραμμα είναι γραμμένο σε FORTRAN 90/95 και χρησιμοποιήθηκε ως σημείο αναφοράς ως προς τις επιδόσεις που μπορούσαμε να πετύχουμε, δεδομένου ότι σχεδόν όλα τα παρόμοια προγράμματα υπολογιστικής ρευστομηχανικής που χρησιμοποιούνται στο εργαστήριο του Τομέα Ναυτικής & Θαλάσσιας Υδροδυναμικής της σχολής ΝΜΜ είναι γραμμένα στην ίδια γλώσσα. Δεν προσφέρει τον ίδιο έλεγχο που προσφέρουν άλλες παρόμοιες γλώσσες όπως η C, δεν υλοποιεί πολύ χρήσιμες δομές (όπως block commenting που επιταχύνει το debugging προγραμμάτων), δεν υλοποιεί pointers με τον ίδιο τρόπο όπως στη C και γενικότερα σε καμία περίπτωση δεν προσφέρεται κατά τη γνώμη μου για συγγραφή προγραμμάτων εξ’ αρχής εκτός και εάν αναπτύσσουμε ένα ήδη υπάρχον.

Το δεύτερο πρόγραμμα είναι η υλοποίηση του ίδιου αλγορίθμου σε C. Η συγγραφή του είναι σαφώς πιο εύκολη και η πορεία του προγράμματος πιο κατανοητή. Η γλώσσα C επιλέχθηκε γιατί είναι η ίδια γλώσσα που προβλέπει το πρότυπο OpenCL (υπάρχουν υλοποιήσεις και για άλλες γλώσσες εκτός προτύπου), καθώς και επιτρέπει στον προγραμματιστή να έχει έχει μεγάλο έλεγχο ως προς όλες τις διεργασίες που λαμβάνουν χώρα (για παράδειγμα τη δέσμευση και κατανομή μνήμης).

Το τρίτο πρόγραμμα είναι η υλοποίηση σε C (του host προγράμματος) και OpenCL C (των kernels) της επίλυσης του ίδιου προβλήματος παράλληλα. Το τμήμα του προγράμματος που εκτελείται παράλληλα είναι η παραγωγή των συντελεστών. Και άλλα τμήματα μπορούν εν δυνάμει να παραλληλοποιηθούν, όπως για παράδειγμα η επίλυση του γραμμικού συστήματος μέσω επαναληπτικών μεθόδων, όμως αυτό δεν συμβαίνει στην παρούσα εργασία για λόγους που θα εξηγηθούν παρακάτω.

Η μέθοδος επίλυσης του προβλήματος που ακολουθήθηκε περιγράφεται αναλυτικότατα στο βιβλίο *“An Introduction to Computational Fluid Dynamics – The Finite Volume Method (2$^{nd}$ edition)”* (H. K. Versteeg & W. Malalasekera - ISBN: 978-0-13-127498-3) από την αρχική θεώρηση (παράγοντας τις εξισώσεις Navier-Stokes), έως την υλοποίηση υπολογιστικά με χρήση πλέγματος. Εδώ θα ανακαλύψουμε τον τροχό ξανά (εν μέρει), παράγοντας τη θεωρία που αφορά στο δικό μας πρόβλημα (δις-διάστατη ροή ρευστού πάνω σε επίπεδη πλάκα, με χρήση δομημένου πλέγματος), από τις διαφορικές εξισώσεις Navier-Stokes, έως τις γραμμικές εξισώσεις παραγωγής των συντελεστών. Για λόγους ευκολίας στην ανάγνωση, τα περιθώρια της σελίδας έχουν ελαφρώς αυξηθεί.

## 3.1 Εξισώσεις Navier Stokes

Ξεκινάμε με την εξίσωση μετατροπής του ρυθμού μεταβολής σωματιδίου ρευστού σε ρυθμό μεταβολής στάσιμου «στοιχείου» ρευστού, που προκύπτει από τον ορισμό:

$$(1)\quad \rho\frac{D\varphi}{Dt}\overset{\text{def}}{\equiv}\rho\left[\frac{\partial\varphi}{\partial t}+\vec{u}\cdot grad(\varphi)\right]=\rho\left[\frac{\partial\varphi}{\partial t}+\vec{u}\cdot grad(\varphi)\right]+\underbrace{\varphi\left[\frac{\partial\rho}{\partial t}+div(\rho\vec{u})\right]}_{=0\ \text{από διατήρηση μάζας}}=[\dots]=\frac{\partial(\rho\varphi)}{\partial t}+div(\rho\varphi\vec{u})$$

Επομένως οι μονοδιάστατες εξισώσεις για σωματίδιο ρευστού εν κινήσει:

$$\rho\frac{Du}{Dt}=-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left[2\mu\cdot\frac{\partial u}{\partial x}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial u}{\partial y}+\mu\cdot\frac{\partial v}{\partial x}\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial u}{\partial z}+\mu\cdot\frac{\partial w}{\partial x}\right]+S_{Mx}\overset{(1)}{\Rightarrow}$$

$$\rho\frac{Dv}{Dt}=-\frac{\partial p}{\partial y}+\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial y}+\mu\cdot\frac{\partial v}{\partial x}\right]+\frac{\partial}{\partial y}\left[2\mu\cdot\frac{\partial v}{\partial y}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial v}{\partial z}+\mu\cdot\frac{\partial w}{\partial y}\right]+S_{My}\overset{(1)}{\Rightarrow}$$

$$\rho\frac{Dw}{Dt}=-\frac{\partial p}{\partial z}+\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial z}+\mu\cdot\frac{\partial w}{\partial x}\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial v}{\partial z}+\mu\cdot\frac{\partial w}{\partial y}\right]+\frac{\partial}{\partial z}\left[2\mu\cdot\frac{\partial w}{\partial z}+\lambda\cdot div(\vec{u})\right]+S_{Mz}\overset{(1)}{\Rightarrow}$$

Μετατρέπονται σε μονοδιάστατες εξισώσεις για στάσιμο στοιχείο ρευστού:

$$\overset{(1)}{\Rightarrow}\frac{\partial(\rho u)}{\partial t}+div(\rho u\vec{u})=-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left[2\mu\cdot\frac{\partial u}{\partial x}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial u}{\partial y}+\mu\cdot\frac{\partial v}{\partial x}\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial u}{\partial z}+\mu\cdot\frac{\partial w}{\partial x}\right]+S_{Mx}$$

$$\overset{(1)}{\Rightarrow}\frac{\partial(\rho v)}{\partial t}+div(\rho v\vec{u})=-\frac{\partial p}{\partial y}+\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial y}+\mu\cdot\frac{\partial v}{\partial x}\right]+\frac{\partial}{\partial y}\left[2\mu\cdot\frac{\partial v}{\partial y}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial v}{\partial z}+\mu\cdot\frac{\partial w}{\partial y}\right]+S_{My}$$

$$\overset{(1)}{\Rightarrow}\frac{\partial(\rho w)}{\partial t}+div(\rho w\vec{u})=-\frac{\partial p}{\partial z}+\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial z}+\mu\cdot\frac{\partial w}{\partial x}\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial v}{\partial z}+\mu\cdot\frac{\partial w}{\partial y}\right]+\frac{\partial}{\partial z}\left[2\mu\cdot\frac{\partial w}{\partial z}+\lambda\cdot div(\vec{u})\right]+S_{Mz}$$

Με ανακατανομή των όρων έχουμε:

$$\Rightarrow\frac{\partial(\rho u)}{\partial t}+div(\rho u\vec{u})=\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial x}\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial u}{\partial y}\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial u}{\partial z}\right]+\left\{\boldsymbol{-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial u}{\partial x}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial y}\left(\mu\cdot\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial z}\left(\mu\cdot\frac{\partial w}{\partial x}\right)}\right\}+S_{Mx}$$

$$\Rightarrow\frac{\partial(\rho v)}{\partial t}+div(\rho v\vec{u})=\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial v}{\partial x}\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial v}{\partial y}\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial v}{\partial z}\right]+\left\{\boldsymbol{-\frac{\partial p}{\partial y}+\frac{\partial}{\partial x}\left(\mu\cdot\frac{\partial u}{\partial y}\right)+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial v}{\partial y}+\lambda\cdot div(\vec{u})\right]+\frac{\partial}{\partial z}\left(\mu\cdot\frac{\partial w}{\partial y}\right)}\right\}+S_{My}$$

$$\Rightarrow\frac{\partial(\rho w)}{\partial t}+div(\rho w\vec{u})=\frac{\partial}{\partial x}\left[\mu\cdot\frac{\partial w}{\partial x}\right]+\frac{\partial}{\partial y}\left[\mu\cdot\frac{\partial w}{\partial y}\right]+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial w}{\partial z}\right]+\left\{\boldsymbol{-\frac{\partial p}{\partial z}+\frac{\partial}{\partial x}\left(\mu\cdot\frac{\partial u}{\partial z}\right)+\frac{\partial}{\partial y}\left(\mu\cdot\frac{\partial v}{\partial z}\right)+\frac{\partial}{\partial z}\left[\mu\cdot\frac{\partial w}{\partial z}+\lambda\cdot div(\vec{u})\right]}\right\}+S_{Mz}$$

Με ομαδοποίηση των όρων σε χρώμα έχουμε:

$$\Rightarrow\frac{\partial(\rho u)}{\partial t}+div(\rho u\vec{u})=div[\mu\cdot grad(u)]+\boldsymbol{[S_x]}+S_{Mx}$$

$$\Rightarrow\frac{\partial(\rho v)}{\partial t}+div(\rho v\vec{u})=div[\mu\cdot grad(v)]+\boldsymbol{[S_y]}+S_{My}$$

$$\Rightarrow\frac{\partial(\rho w)}{\partial t}+div(\rho w\vec{u})=div[\mu\cdot grad(w)]+\boldsymbol{[S_z]}+S_{Mz}$$

Οι όροι πηγής $S_{Mx}$ , $S_{My}$ and $S_{Mz}$ περιλαμβάνουν τις επιρροές λόγω δυνάμεων στο σώμα του ρευστού. Για παράδειγμα η δύναμη της βαρύτητας θα μοντελοποιείτο όπως παρακάτω:

$$S_{Mx} = 0, \qquad S_{My} = 0\,, \qquad S_{Mz} = -\rho g$$

Στο πρόβλημά μας δεν λαμβάνονται υπ' όψιν τέτοιες δυνάμεις.

Για τη μελέτη που μας ενδιαφέρει έχουμε ασυμπίεστη μόνιμη δισδιάστατη ροή ($\frac{\partial(.)}{\partial t} = 0,\ \rho{:}\,\sigma\tau\alpha\theta,$ $\mu{:}\,\sigma\tau\alpha\theta, w = 0$) άρα οι εξισώσεις τροποποιούνται όπως παρακάτω:

$$\rho \cdot div(u\vec{u}) = div[\mu \cdot grad(u)] + \left\{-\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\mu \cdot \frac{\partial u}{\partial x}\right] + \frac{\partial}{\partial y}\left(\mu \cdot \frac{\partial v}{\partial x}\right)\right\} + S_{Mx}$$

$$\rho \cdot div(v\vec{u}) = div[\mu \cdot grad(v)] + \left\{-\frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left(\mu \cdot \frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial y}\left[\mu \cdot \frac{\partial v}{\partial y}\right]\right\} + S_{My}$$

Οι εξισώσεις αυτές έχουν τη γενική μορφή της εξίσωσης μεταφοράς:

$$\underbrace{\frac{\partial(\rho\varphi)}{\partial t}}_{Rate\ of\ change} + \underbrace{div(\rho\varphi\vec{u})}_{Convection} = \underbrace{div[\Gamma \cdot grad(\varphi)]}_{Diffusion} + \underbrace{[S_\varphi]}_{Source}$$

Και άρα μπορούν να λυθούν όπως παρακάτω:

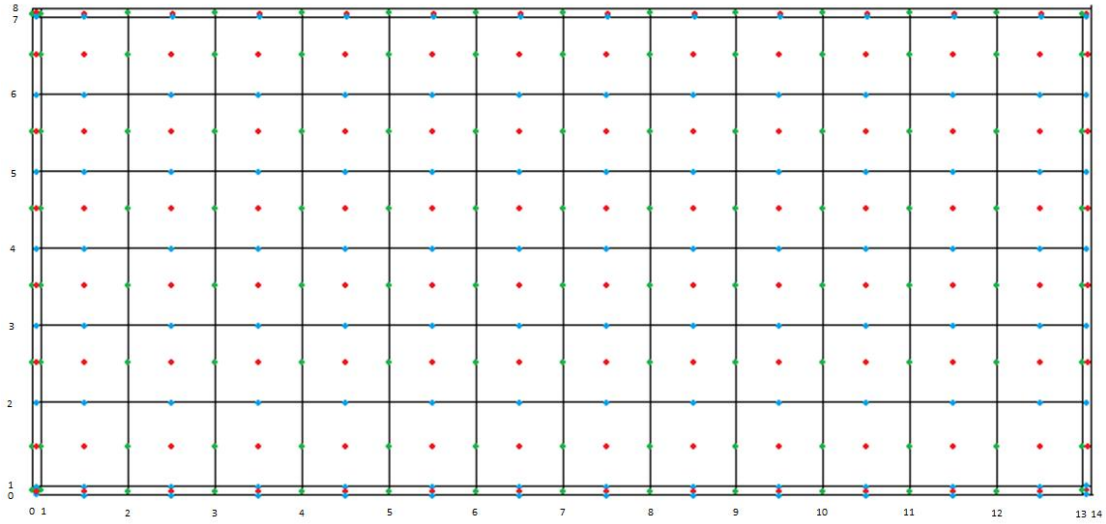
$$\frac{\partial(\rho\varphi)}{\partial t} + div(\rho\varphi\vec{u}) = div[\Gamma \cdot grad(\varphi)] + [S_\varphi] \overset{\int_{CV}}{\Longrightarrow}$$

$$\overset{\int_{CV}}{\Longrightarrow} \int_{CV} \frac{\partial(\rho\varphi)}{\partial t} dV + \int_{CV} div(\rho\varphi\vec{u}) dV = \int_{CV} div[\Gamma \cdot grad(\varphi)] dV + \int_{CV} [S_\varphi] dV \overset{GAUSS}{\Longrightarrow}$$

$$\overset{GAUSS}{\Longrightarrow} \int_{CV} \frac{\partial(\rho\varphi)}{\partial t} dV + \int_{A} \hat{n}(\rho\varphi\vec{u}) dA = \int_{A} \hat{n}[\Gamma \cdot grad(\varphi)] dA + \underbrace{\bar{S} \cdot \Delta V}_{(S_u + S_p \cdot \varphi_p) \cdot \Delta V}$$

Ακολουθεί η περιγραφή του πλέγματος που χρησιμοποιήθηκε και η παραγωγή των συντελεστών του γραμμικού συστήματος για την κάθε ταχύτητα χωριστά.

## 3.2 Περιγραφή πλέγματος




Σκοπός είναι η εφαρμογή του αλγορίθμου SIMPLE για τον υπολογισμό των $U, V, P$ για ένα σύνολο κόμβων χρησιμοποιώντας δομημένο ορθογώνιο πλέγμα.

Το πλέγμα ορίζεται από δύο σύνολα σημείων: Ένα σύνολο σημείων στον $X$-άξονα και ένα σύνολο σημείων στον $Y$-άξονα (τα σημεία αυτά αποθηκεύονται στους αντίστοιχους πίνακες $X, Y$ στο πρόγραμμα). Στα όρια $E, W, S, N$ υπάρχει διπλή πλεγματική γραμμή (κατά $x$ για τα $E, W$, και κατά $y$ για τα $N, S$) δηλαδή οι γραμμές αυτές βρίσκονται στο ίδιο σημείο του πραγματικού χώρου (για λόγους που θα φανούν παρακάτω). Η αρίθμηση των γραμμών ξεκινάει από το 0, για απλοποίηση του κώδικα. Δημιουργούμε ενδεικτικά ένα απλό πλέγμα, όπου οι γραμμές ισαπέχουν μεταξύ τους κατά 1 στον οριζόντιο και κατακόρυφο άξονα. Με βάση τα παραπάνω, ισχύει

$X(0) = X(1) = 0$

$X(i) = i - 1 \, για \, i = 2, \dots, 13$

$X(14) = X(13) = 12$

$Y(0) = Y(1) = 0$

$Y(j) = j - 1 \, για \, j = 2, \dots, 7$

$Y(8) = Y(7) = 6$

Αν πάρουμε σαν δεδομένο το παραπάνω πλέγμα, έχουμε έναν πίνακα κατά $X$ χωρητικότητας 15 σημείων $(i = 0,1, \dots, 14)$ αλλά ο αριθμός των κόμβων $u, v, P$ κατά $X$ είναι κατά 1 μονάδα λιγότερος γιατί δεν ορίζεται στην τελευταία πλεγματική γραμμή. Επίσης ο αριθμός των *διαφορετικών* αποστάσεων κατά $x$ (δηλαδή των διαφορετικών σημείων στον πραγματικό χώρο κατά $x$) είναι κατά 2 μονάδες λιγότερος αφού υπάρχουν 2 πλεονάζουσες πλεγματικές γραμμές κατά $x$, στα όρια $E$ και $W$. Ανάλογα ισχύουν και για τον άξονα $y$.

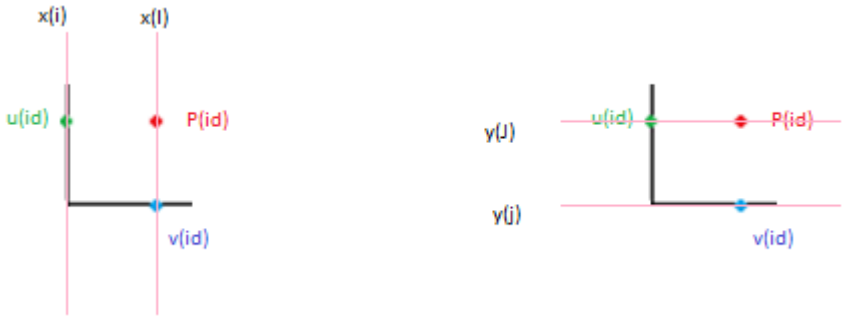
Ορίζουμε τον αριθμό των διαφορετικών σημείων στον πραγματικό χώρο κατά $x$ ως $xpoints$ και κατά $y$ ως $ypoints$. Αντίστοιχα ο αριθμός των κόμβων $u, v, P$ κατά $X$ είναι $xpoints + 1$ για τον οποίο χρησιμοποιούμε τη μεταβλητή $xpointsp1$ και κατά $Y$ είναι $ypoints + 1$ για τον οποίο χρησιμοποιούμε τη μεταβλητή $ypointsp1$. Για το πρόγραμμα δεν θα χρειαστούμε τις αντίστοιχες μεταβλητές $xpointsp2$,

$ypointsp2$, που θα εξέφραζαν τον αριθμό των πλεγματικών γραμμών κατά $X$ & $Y$ αντίστοιχα, όμως οι πίνακες $X$ & $Y$ περιέχουν ακριβώς τόσα στοιχεία.

Οι θέσεις όπου υπολογίζουμε τις ταχύτητες $u$ παριστάνονται στο πλέγμα με πράσινο χρώμα οι θέσεις των $v$ με μπλέ και οι θέσεις των πιέσεων $P$ με κόκκινο

Οι κόμβοι έχουν τη δις-διάσταση αρίθμηση που ορίζεται από το ζεύγος $X(i)$, $Y(j)$, και τη μονοδιάστατη αρίθμηση που ορίζεται ως εξής: $id = Y(j) \cdot xpointsp1 + X(i)$. *Δηλαδή η μονοδιάστατη αρίθμηση ξεκινάει από κάτω αριστερά και τελειώνει πάνω δεξιά*

Στις εξισώσεις παραγωγής των συντελεστών του γραμμικού συστήματος γίνεται χρήση των συντεταγμένων κατά $x, y$ των κόμβων $u, v, P$, όμως οι κόμβοι αυτοί δεν βρίσκονται στην ίδια θέση στον πραγματικό χώρο ακόμα και εάν έχουν την ίδια «*ταυτότητα*» $id$, όπως φαίνεται και στο παρακάτω σχήμα:




Επομένως γίνεται απαραίτητη η εξής διάκριση:

Η απόσταση $x$ του κόμβου ταχύτητας $u$ συμβολίζεται με $x_i$ ενώ των κόμβων $v, P$ με $x_I$

Η απόσταση $y$ του κόμβου ταχύτητας $v$ συμβολίζεται με $y_j$ ενώ των κόμβων $u, P$ με $y_J$

Προφανώς η ποσότητα $x_i$ ταυτίζεται με την ποσότητα $X(i)$ του πίνακα που κρατάει τις θέσεις των πλεγματικών γραμμών $X$ ενώ η ποσότητα $x_I$ ταυτίζεται με την ποσότητα $\frac{X(i)+X(i+1)}{2}$. Αντίστοιχα η ποσότητα $y_j$ ταυτίζεται με την ποσότητα $Y(j)$ του πίνακα που κρατάει τις θέσεις των πλεγματικών γραμμών $Y$ ενώ η ποσότητα $y_J$ ταυτίζεται με την ποσότητα $\frac{Y(j)+Y(j+1)}{2}$.

Στο πρόγραμμα όταν απαιτείται ο υπολογισμός των ποσοτήτων $x_i - x_{i-1}$ ή $x_{i+1} - x_i$ αυτές συμβολίζονται με τις μεταβλητές `Xi_Xim1`, και `Xip1_Xi`, ενώ για τις αντίστοιχες $x_I - x_{I-1}$, $x_{I+1} - x_I$, οι μεταβλητές είναι `XI_XIM1`, και `XIP1_XI`. Το πώς προκύπτει η ονοματολογία είναι εμφανές. Τα ίδια ισχύουν και για τις μεταβλητές για τις αποστάσεις κατά $Y$

Επειδή η FORTRAN δεν κάνει διαχωρισμό μεταξύ κεφαλαίων και πεζών χαρακτήρων όσον αφορά στα ονόματα των μεταβλητών, στο πρόγραμμα σε FORTRAN (σε αντίθεση με αυτό σε c) τα ονόματα των μεταβλητών που περιέχουν διαφορές αποστάσεων με κεφαλαίους ενδείκτες συμπληρώνεται με το πρόθεμα `CAP_`. Για παράδειγμα η μεταβλητή `XI_XIM1` γίνεται `CAP_XI_XIM1`.:

Οι οριακές συνθήκες είναι (ενδεικτικά):

- Για όριο $E$: $u_{id} = u_{id-1}$, $v_{id} = v_{id-1}$
- Για όριο $W$: $u_{id} = velin$, $v_{id} = 0$
- Για όριο $S$: $u_{id} = 0$, $v_{id} = 0$
- Για όριο $N$: $u_{id} = velin$, $v_{id} = 0$

## 3.3 Παραγωγή συντελεστών γραμμικού συστήματος u

$$\rho \cdot div(u\vec{u}) = div[\mu \cdot grad(u)] + \left\{-\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\mu\frac{\partial u}{\partial x}\right] + \frac{\partial}{\partial y}\left[\mu\frac{\partial v}{\partial x}\right]\right\} + \cancel{S_{Mx}} \xrightarrow{u=\varphi}$$

$$\int_{CV} \rho \cdot div(\varphi\vec{u})\, dV = \int_{CV} div[\mu \cdot grad(\varphi)]\, dV + \int_{CV} SdV \xrightarrow{\mu=\sigma\tau\alpha\theta}$$

$$\int_{A} \rho \cdot \hat{n} \cdot \varphi \cdot \vec{u}\, dA = \int_{A} n \cdot [\mu \cdot grad(\varphi)] \cdot dA + \int_{CV} SdV \Rightarrow$$

$$|\rho \cdot u \cdot A \cdot \varphi|_e - |\rho \cdot u \cdot A \cdot \varphi|_w + |\rho \cdot u \cdot A \cdot \varphi|_n - |\rho \cdot u \cdot A \cdot \varphi|_s = \left|\mu A\frac{\partial \varphi}{\partial x}\right|_e - \left|\mu A\frac{\partial \varphi}{\partial x}\right|_w + \left|\mu A\frac{\partial \varphi}{\partial y}\right|_n - \left|\mu A\frac{\partial \varphi}{\partial y}\right|_s + \int_{CV} SdV \Rightarrow$$

$$F_e\varphi_e A_e - F_w\varphi_w A_w + F_n\varphi_n A_n - F_s\varphi_s A_s = \mu A_e\frac{(\Phi_E - \Phi_P)}{x_{i+1} - x_i} - \mu A_w\frac{(\Phi_P - \Phi_W)}{x_i - x_{i-1}} + \mu A_n\frac{(\Phi_N - \Phi_P)}{y_{J+1} - y_J} - \mu A_s\frac{(\Phi_P - \Phi_S)}{y_J - y_{J-1}} + \int_{CV} SdV \xrightarrow{i,J} \Rightarrow$$

$$F_e A_e \cdot \left[\frac{\max(F_e,0)}{F_e} \cdot u(i,J) + \frac{\max(-F_e,0)}{-F_e} \cdot u(i+1,J)\right] - F_w A_w \cdot \left[\frac{\max(F_w,0)}{F_w} \cdot u(i-1,J) + \frac{\max(-F_w,0)}{-F_w} \cdot u(i,J)\right] +$$

$$+F_n A_n \cdot \left[\frac{\max(F_n,0)}{F_n} \cdot u(i,J) + \frac{\max(-F_n,0)}{-F_n} \cdot u(i,J+1)\right] - F_s A_s \cdot \left[\frac{\max(F_s,0)}{F_s} \cdot u(i,J-1) + \frac{\max(-F_s,0)}{-F_s} \cdot u(i,J)\right] =$$

$$\mu A_e\frac{(u(i+1,J) - u(i,J))}{x_{i+1} - x_i} - \mu A_w\frac{(u(i,J) - u(i-1,J))}{x_i - x_{i-1}} + \mu A_n\frac{(u(i,J+1) - u(i,J))}{y_{J+1} - y_J} - \mu A_s\frac{(u(i,J) - u(i,J-1))}{y_J - y_{J-1}} +$$

$$+\frac{P(I-1,J) - P(I,J)}{x_I - x_{I-1}} \cdot \delta V + \frac{\mu \cdot \delta V}{x_I - x_{I-1}} \cdot \left(\frac{u(i+1,J) - u(i,J)}{x_{i+1} - x_i} - \frac{u(i,J) - u(i-1,J)}{x_i - x_{i-1}}\right) +$$

$$+\frac{\mu \cdot \delta V}{y_{j+1} - y_j} \cdot \left(\frac{v(I,j+1) - v(I-1,j+1)}{x_I - x_{I-1}} - \frac{v(I,1) - v(I-1,j)}{x_I - x_{I-1}}\right) \Rightarrow$$

$$u(i,J) \cdot \left\{\max(F_e,0)\,A_e + \max(-F_w,0)\,A_w + \max(F_n,0)\,A_n + \max(-F_s,0)\,A_s + \frac{\mu A_e}{x_{i+1} - x_i} + \frac{\mu A_w}{x_i - x_{i-1}} + \frac{\mu A_n}{y_{J+1} - y_J}\right.$$
$$\left. + \frac{\mu A_s}{y_J - y_{J-1}} + \frac{\mu \cdot \delta V}{(x_I - x_{I-1}) \cdot (x_{i+1} - x_i)} + \frac{\mu \cdot \delta V}{(x_I - x_{I-1}) \cdot (x_i - x_{i-1})}\right\} =$$

$$u(i+1,J) \cdot \left\{\max(-F_e,0)\,A_e + \frac{\mu A_e}{x_{i+1} - x_i} + \frac{\mu \cdot \delta V}{(x_I - x_{I-1}) \cdot (x_{i+1} - x_i)}\right\} +$$

$$u(i-1,J) \cdot \left\{\max(F_w,0)\,A_w + \frac{\mu A_w}{x_i - x_{i-1}} + \frac{\mu \cdot \delta V}{(x_I - x_{I-1}) \cdot (x_i - x_{i-1})}\right\} +$$

$$u(i,J+1) \cdot \left\{\max(-F_n,0)\,A_n + \frac{\mu A_n}{y_{J+1} - y_J}\right\} +$$

$$u(i,J-1) \cdot \left\{\max(F_s,0)\,A_s + \frac{\mu A_s}{y_J - y_{J-1}}\right\} +$$

$$+\frac{P(I-1,J) - P(I,J)}{x_I - x_{I-1}} \cdot \delta V + \frac{\mu \cdot \delta V}{y_{j+1} - y_j} \cdot \left(\frac{v(I,j+1) - v(I-1,j+1)}{x_I - x_{I-1}} - \frac{v(I,1) - v(I-1,j)}{x_I - x_{I-1}}\right) \xrightarrow{\div \delta V} \Rightarrow$$

$$u(i,J)\cdot\left\{\frac{\max(F_e,0)}{x_I-x_{I-1}}+\frac{\max(-F_w,0)}{x_I-x_{I-1}}+\frac{\max(F_n,0)}{y_{j+1}-y_j}+\frac{\max(-F_s,0)}{y_{j+1}-y_j}+\frac{2\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}+\frac{2\mu}{(x_I-x_{I-1})\cdot(x_i-x_{i-1})}\right.$$
$$\left.+\frac{\mu}{(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)}+\frac{\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right\}=$$

$$u(i+1,J)\cdot\left\{\frac{\max(-F_e,0)}{x_I-x_{I-1}}+\frac{2\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}\right\}+$$

$$u(i-1,J)\cdot\left\{\frac{\max(F_w,0)}{x_I-x_{I-1}}+\frac{2\mu}{(x_I-x_{I-1})\cdot(x_i-x_{i-1})}\right\}+$$

$$u(i,J+1)\cdot\left\{\frac{\max(-F_n,0)}{y_{j+1}-y_j}+\frac{\mu}{(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)}\right\}+$$

$$u(i,J-1)\cdot\left\{\frac{\max(F_s,0)}{y_{j+1}-y_j}+\frac{\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right\}+$$

$$+\frac{P(I-1,J)-P(I,J)}{x_I-x_{I-1}}+\frac{\mu}{(y_{j+1}-y_j)\cdot(x_I-x_{I-1})}\cdot[v(I,j+1)-v(I-1,j+1)-v(I,1)+v(I-1,j)]$$

Πολλαπλασιάζοντας με $(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}$ έχουμε απαλοιφή όλων των κλασμάτων:

$$u(i,J)\cdot\left\{\begin{array}{rl}
\max(F_e,0)\cdot(y_{j+1}-y_j)\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+ & \\
\max(-F_w,0)\cdot(y_{j+1}-y_j)\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+ & \\
\max(F_n,0)\cdot(x_I-x_{I-1})\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+ & \\
\max(-F_s,0)\cdot(x_I-x_{I-1})\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+ & \\
+2\mu\cdot(y_{j+1}-y_j)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})+ & \leftarrow term1\\
+2\mu\cdot(y_{j+1}-y_j)\cdot(x_{i+1}-x_i)\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})+ & \leftarrow term2\\
+\mu\cdot(x_I-x_{I-1})\cdot(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_J-y_{J-1})+ & \leftarrow term3\\
+\mu\cdot(x_I-x_{I-1})\cdot(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J) & \leftarrow term4
\end{array}\right\}=$$

$$u(i+1,J)\cdot\left\{\max(-F_e,0)\cdot(y_{j+1}-y_j)\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+term1\right\}+$$

$$u(i-1,J)\cdot\left\{\max(F_w,0)\cdot(y_{j+1}-y_j)\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+term2\right\}+$$

$$u(i,J+1)\cdot\left\{\max(-F_n,0)\cdot(x_I-x_{I-1})\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+term3\right\}+$$

$$u(i,J-1)\cdot\left\{\max(F_s,0)\cdot(x_I-x_{I-1})\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}+term4\right\}+$$

$$+\left\{[P(I-1,J)-P(I,J)]\cdot(y_{j+1}-y_j)+\mu\cdot[v(I,j+1)-v(I-1,j+1)-v(I,j)+v(I-1,j)]\right\}\cdot\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})}$$

Θέτοντας $\boldsymbol{(x_{i+1}-x_i)\cdot(x_i-x_{i-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})=term}$, έχουμε:

$$u(i,J)\cdot\{[\max(F_e,0)\cdot(y_{j+1}-y_j)+\max(-F_w,0)\cdot(y_{j+1}-y_j)+\max(F_n,0)\cdot(x_I-x_{I-1})+\max(-F_s,0)\cdot(x_I-x_{I-1})]\cdot \mathbf{term}+term1+term2+term3+term4\}=$$

$$u(i+1,J)\cdot\{\max(-F_e,0)\cdot(y_{j+1}-y_j)\cdot \mathbf{term}+term1\}+$$

$$u(i-1,J)\cdot\{\max(F_w,0)\cdot(y_{j+1}-y_j)\cdot \mathbf{term}+term2\}+$$

$$u(i,J+1)\cdot\{\max(-F_n,0)\cdot(x_I-x_{I-1})\cdot \mathbf{term}+term3\}+$$

$$u(i,J-1)\cdot\{\max(F_s,0)\cdot(x_I-x_{I-1})\cdot \mathbf{term}+term4\}+$$

$$+\{[P(I-1,J)-P(I,J)]\cdot(y_{j+1}-y_j)+\mu\cdot[v(I,j+1)-v(I-1,j+1)-v(I,j)+v(I-1,j)]\}\cdot \mathbf{term}$$

*Η παραπάνω εξίσωση είναι αυτή που θα χρησιμοποιηθεί για την παραγωγή των συντελεστών u του γραμμικού συστήματος στο πρόγραμμα.*

Βάση βιβλιογραφίας, σύγκλιση του αλγορίθμου SIMPLE επιβάλλει την παρακάτω ανισότητα: $AP_u(i,J)\geq\sum_{nb}(a_{nb})$. Με βάση το σχήμα διαφορών που έχουμε επιλέξει, προκύπτει ότι ισχύει η ισότητα όπως αποδεικνύεται ακολούθως:

$$AP_u(i,J)=\sum_{nb}(a_{nb})\Rightarrow$$

$$\Rightarrow[\max(F_e,0)\cdot(y_{j+1}-y_j)+\max(-F_w,0)\cdot(y_{j+1}-y_j)+\max(F_n,0)\cdot(x_I-x_{I-1})+\max(-F_s,0)\cdot(x_I-x_{I-1})]\cdot \mathbf{term}+term1+term2+term3+term4=$$

$$=\max(-F_e,0)\cdot(y_{j+1}-y_j)\cdot \mathbf{term}+term1+$$

$$+\max(F_w,0)\cdot(y_{j+1}-y_j)\cdot \mathbf{term}+term2+$$

$$+\max(-F_n,0)\cdot(x_I-x_{I-1})\cdot \mathbf{term}+term3+$$

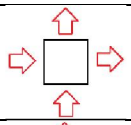
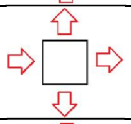
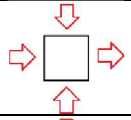
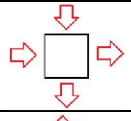
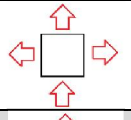
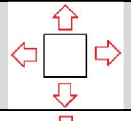
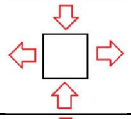
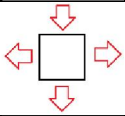
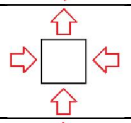
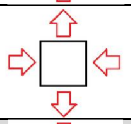
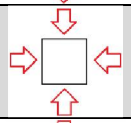
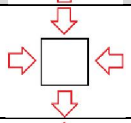
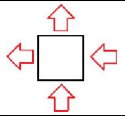
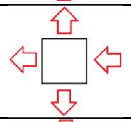
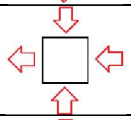
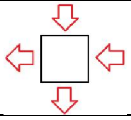
$$+\max(F_s,0)\cdot(x_I-x_{I-1})\cdot \mathbf{term}+term4\Rightarrow$$

$$\frac{\max(F_e,0)}{x_I-x_{I-1}}+\frac{\max(-F_w,0)}{x_I-x_{I-1}}+\frac{\max(F_n,0)}{y_{j+1}-y_j}+\frac{\max(-F_s,0)}{y_{j+1}-y_j}=\frac{\max(-F_e,0)}{x_I-x_{I-1}}+\frac{\max(F_w,0)}{x_I-x_{I-1}}+\frac{\max(-F_n,0)}{y_{j+1}-y_j}+\frac{\max(F_s,0)}{y_{j+1}-y_j}$$

Εδώ απαιτείται εξέταση των προσήμων των $F_e,F_w,F_n,F_s$ και προς αυτήν την κατεύθυνση κατασκευάζουμε τον παρακάτω πίνακα. Προφανώς σε κάθε περίπτωση ισχύει η εξίσωση διατήρησης της μάζας στον όγκο ελέγχου της ταχύτητας u και αυτή απαγορεύει ορισμένους συνδυασμούς προσήμων (εκτός και εάν οι ταχύτητες είναι μηδενικές κατ' απόλυτη τιμή, βλ. παρακάτω με σκίαση).

$$div(\vec{u})=0\xrightarrow{CV\,(u)}\int_{CV}div(\vec{u})dV=0\Rightarrow\int_A\hat{n}\cdot\vec{u}\cdot dA=0\Rightarrow u_eA_e-u_wA_w+u_nA_n-u_sA_s=0\Rightarrow$$

$$F_eA_e-F_wA_w+F_nA_n-F_sA_s=0\Rightarrow\frac{F_e}{x_I-x_{I-1}}-\frac{F_w}{x_I-x_{I-1}}+\frac{F_n}{y_{j+1}-y_j}-\frac{F_s}{y_{j+1}-y_j}=0$$

| Ισχύει η εξίσωση | $F_e$ | $F_w$ | $F_n$ | $F_s$ | $CV\ (u)$ |
|---|---|---|---|---|---|
| ΝΑΙ | $(+)$ | $(+)$ | $(+)$ | $(+)$ |  |
| ΝΑΙ | $(+)$ | $(+)$ | $(+)$ | $(-)$ |  |
| ΝΑΙ | $(+)$ | $(+)$ | $(-)$ | $(+)$ |  |
| ΝΑΙ | $(+)$ | $(+)$ | $(-)$ | $(-)$ |  |
| ΝΑΙ | $(+)$ | $(-)$ | $(+)$ | $(+)$ |  |
| *ΝΑΙ* | $(+)$ | $(-)$ | $(+)$ | $(-)$ |  |
| ΝΑΙ | $(+)$ | $(-)$ | $(-)$ | $(+)$ |  |
| ΝΑΙ | $(+)$ | $(-)$ | $(-)$ | $(-)$ |  |
| ΝΑΙ | $(-)$ | $(+)$ | $(+)$ | $(+)$ |  |
| ΝΑΙ | $(-)$ | $(+)$ | $(+)$ | $(-)$ |  |
| *ΝΑΙ* | $(-)$ | $(+)$ | $(-)$ | $(+)$ |  |
| ΝΑΙ | $(-)$ | $(+)$ | $(-)$ | $(-)$ |  |
| ΝΑΙ | $(-)$ | $(-)$ | $(+)$ | $(+)$ |  |
| ΝΑΙ | $(-)$ | $(-)$ | $(+)$ | $(-)$ |  |
| ΝΑΙ | $(-)$ | $(-)$ | $(-)$ | $(+)$ |  |
| ΝΑΙ | $(-)$ | $(-)$ | $(-)$ | $(-)$ |  |

# 3.4 Παραγωγή συντελεστών γραμμικού συστήματος v

$$\rho \cdot div(v\vec{u}) = div[\mu \cdot grad(v)] + \left\{-\frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left[\mu\frac{\partial u}{\partial y}\right] + \frac{\partial}{\partial y}\left[\mu\frac{\partial v}{\partial y}\right]\right\} + \cancel{S_{My}} \overset{v=\varphi}{\Longrightarrow}$$

$$\int_{CV} \rho \cdot div(\varphi\vec{u})dV = \int_{CV} div[\mu \cdot grad(\varphi)]dV + \int_{CV} SdV \overset{Gauss}{\Longrightarrow}$$

$$\int_{A} \rho \cdot \hat{n} \cdot \varphi \cdot \vec{u} \cdot dA = \int_{A} \hat{n} \cdot \mu \cdot grad(\varphi) \cdot dA + \int_{CV} SdV \Rightarrow$$

$$|\rho \cdot u \cdot A \cdot \varphi|_e - |\rho \cdot u \cdot A \cdot \varphi|_w + |\rho \cdot v \cdot A \cdot \varphi|_n - |\rho \cdot v \cdot A \cdot \varphi|_s = \left|\mu A\frac{\partial \varphi}{\partial x}\right|_e - \left|\mu A\frac{\partial \varphi}{\partial x}\right|_w + \left|\mu A\frac{\partial \varphi}{\partial y}\right|_n - \left|\mu A\frac{\partial \varphi}{\partial y}\right|_s + \int_{CV} SdV \Rightarrow$$

$$F_eA_e\varphi_e - F_wA_w\varphi_w + F_nA_n\varphi_n - F_sA_s\varphi_s = \mu A_e\frac{(\Phi_E - \Phi_P)}{x_{I+1} - x_I} - \mu A_w\frac{(\Phi_P - \Phi_W)}{x_I - x_{I-1}} + \mu A_n\frac{(\Phi_N - \Phi_P)}{y_{j+1} - y_j} - \mu A_s\frac{(\Phi_P - \Phi_S)}{y_j - y_{j-1}} + \int_{CV} SdV \overset{i,J}{\Rightarrow}$$

$$F_eA_e \cdot \left[\frac{\max(F_e,0)}{F_e} \cdot v(I,j) + \frac{\max(-F_e,0)}{-F_e} \cdot v(I+1,j)\right] - F_wA_w \cdot \left[\frac{\max(F_w,0)}{F_w} \cdot v(I-1,j) + \frac{\max(-F_w,0)}{-F_w} \cdot v(I,j)\right] +$$

$$+F_nA_n \cdot \left[\frac{\max(F_n,0)}{F_n} \cdot v(I,j) + \frac{\max(-F_n,0)}{-F_n} \cdot v(I,j+1)\right] - F_sA_s \cdot \left[\frac{\max(F_s,0)}{F_s} \cdot v(I,j-1) + \frac{\max(-F_s,0)}{-F_s} \cdot v(I,j)\right] =$$

$$\mu A_e\frac{(v(I+1,j) - v(I,j))}{x_{I+1} - x_I} - \mu A_w\frac{(v(I,j) - v(I-1,j))}{x_I - x_{I-1}} + \mu A_n\frac{(v(I,j+1) - v(I,j))}{y_{j+1} - y_j} - \mu A_s\frac{(v(I,j) - v(I,j-1))}{y_j - y_{j-1}} +$$

$$+\frac{P(I,J-1) - P(I,J)}{y_J - y_{J-1}} \cdot \delta V + \frac{\mu \cdot \delta V}{x_{i+1} - x_i} \cdot \left(\frac{u(i+1,J) - u(i+1,J-1)}{y_J - y_{J-1}} - \frac{u(i,J) - u(i,J-1)}{y_J - y_{J-1}}\right) +$$

$$+\frac{\mu \cdot \delta V}{y_J - y_{J-1}} \cdot \left(\frac{v(I,j+1) - v(I,j)}{y_{j+1} - y_j} - \frac{v(I,j) - v(I,j-1)}{y_j - y_{j-1}}\right) \Rightarrow$$

$$v(I,j) \cdot \left\{\max(F_e,0)\,A_e + \max(-F_w,0)\,A_w + \max(F_n,0)\,A_n + \max(-F_s,0)\,A_s + \frac{\mu A_e}{x_{I+1} - x_I} + \frac{\mu A_w}{x_I - x_{I-1}} + \frac{\mu A_n}{y_{j+1} - y_j}\right.$$
$$\left. + \frac{\mu A_s}{y_j - y_{j-1}} + \frac{\mu \cdot \delta V}{(y_J - y_{J-1}) \cdot (y_{j+1} - y_j)} + \frac{\mu \cdot \delta V}{(y_J - y_{J-1}) \cdot (y_j - y_{j-1})}\right\} =$$

$$v(I+1,j) \cdot \left\{\max(-F_e,0)\,A_e + \frac{\mu A_e}{x_{I+1} - x_I}\right\} +$$

$$v(I-1,j) \cdot \left\{\max(F_w,0)\,A_w + \frac{\mu A_w}{x_I - x_{I-1}}\right\} +$$

$$v(I,j+1) \cdot \left\{\max(-F_n,0)\,A_n + \frac{\mu A_n}{y_{j+1} - y_j} + \frac{\mu \cdot \delta V}{(y_J - y_{J-1}) \cdot (y_{j+1} - y_j)}\right\} +$$

$$v(I,j-1) \cdot \left\{\max(F_s,0)\,A_s + \frac{\mu A_s}{y_j - y_{j-1}} + \frac{\mu \cdot \delta V}{(y_J - y_{J-1}) \cdot (y_j - y_{j-1})}\right\} +$$

$$+\frac{P(I,J-1) - P(I,J)}{y_J - y_{J-1}} \cdot \delta V + \frac{\mu \cdot \delta V}{x_{i+1} - x_i} \cdot \left(\frac{u(i+1,J) - u(i+1,J-1)}{y_J - y_{J-1}} - \frac{u(i,J) - u(i,J-1)}{y_J - y_{J-1}}\right) \overset{\div \delta V}{\Longrightarrow}$$

$$v(I,j)\cdot\left\{\frac{\max(F_e,0)}{x_{i+1}-x_i}+\frac{\max(-F_w,0)}{x_{i+1}-x_i}+\frac{\max(F_n,0)}{y_J-y_{J-1}}+\frac{\max(-F_s,0)}{y_J-y_{J-1}}+\frac{\mu}{(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}+\frac{\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}\right.$$
$$\left.+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_j-y_{j-1})}\right\}=$$

$$v(I+1,j)\cdot\left\{\frac{\max(-F_e,0)}{x_{i+1}-x_i}+\frac{\mu}{(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}\right\}+$$

$$v(I-1,j)\cdot\left\{\frac{\max(F_w,0)}{x_{i+1}-x_i}+\frac{\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}\right\}+$$

$$v(I,j+1)\cdot\left\{\frac{\max(-F_n,0)}{y_J-y_{J-1}}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right\}+$$

$$v(I,j-1)\cdot\left\{\frac{\max(F_s,0)}{y_J-y_{J-1}}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_j-y_{j-1})}\right\}+$$

$$+\frac{P(I,J-1)-P(I,J)}{y_J-y_{J-1}}+\frac{\mu}{(x_{i+1}-x_i)\cdot(y_J-y_{J-1})}\cdot[u(i+1,J)-u(i+1,J-1)-u(i,J)+u(i,J-1)]$$

Πολλαπλασιάζοντας με $(x_{i+1}-x_i)\cdot(y_J-y_{J-1})\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}$ έχουμε απαλοιφή όλων των κλασμάτων:

$$v(i,J)\cdot\left\{\begin{array}{ll}
\max(F_e,0)\cdot(y_J-y_{J-1})\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+ & \\
\max(-F_w,0)\cdot(y_J-y_{J-1})\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+ & \\
\max(F_n,0)\cdot(x_{i+1}-x_i)\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+ & \\
\max(-F_s,0)\cdot(x_{i+1}-x_i)\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+ & \\
+\mu\cdot(y_J-y_{J-1})\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})+ & \leftarrow term1\\
+\mu\cdot(y_J-y_{J-1})\cdot(x_{I+1}-x_I)\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})+ & \leftarrow term2\\
+2\mu\cdot(x_{i+1}-x_i)\cdot(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_j-y_{j-1})+ & \leftarrow term3\\
+2\mu\cdot(x_{i+1}-x_i)\cdot(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)+ & \leftarrow term4
\end{array}\right\}=$$

$$v(i+1,J)\cdot\left\{\max(-F_e,0)\cdot(y_J-y_{J-1})\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+term1\right\}+$$

$$v(i-1,J)\cdot\left\{\max(F_w,0)\cdot(y_J-y_{J-1})\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+term2\right\}+$$

$$v(i,J+1)\cdot\left\{\max(-F_n,0)\cdot(x_{i+1}-x_i)\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+term3\right\}+$$

$$v(i,J-1)\cdot\left\{\max(F_s,0)\cdot(x_{i+1}-x_i)\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}+term4\right\}+$$

$$+\{[P(I,J-1)-P(I,J)]\cdot(x_{i+1}-x_i)+\mu\cdot[u(i+1,J)-u(i+1,J-1)-u(i,J)+u(i,J-1)]\}\cdot\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})}$$

Θέτοντας $\mathbf{(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{j+1}-y_j)\cdot(y_j-y_{j-1})=term}$, έχουμε:

$$v(i,J)\cdot[\max(F_e,0)\cdot(y_J-y_{J-1})+\max(-F_w,0)\cdot(y_J-y_{J-1})+\max(F_n,0)\cdot(x_{i+1}-x_i)+\max(-F_s,0)\cdot(x_{i+1}-x_i)]\cdot \boldsymbol{term}+ term1+term2+term3+term4=$$

$$v(i+1,J)\cdot\left\{\max(-F_e,0)\cdot(y_J-y_{J-1})\cdot \boldsymbol{term}+term1\right\}+$$

$$v(i-1,J)\cdot\left\{\max(F_w,0)\cdot(y_J-y_{J-1})\cdot \boldsymbol{term}+term2\right\}+$$

$$v(i,J+1)\cdot\{\max(-F_n,0)\cdot(x_{i+1}-x_i)\cdot \boldsymbol{term}+term3\}+$$

$$v(i,J-1)\cdot\{\max(F_s,0)\cdot(x_{i+1}-x_i)\cdot \boldsymbol{term}+term4\}+$$

$$+\{[P(I,J-1)-P(I,J)]\cdot(x_{i+1}-x_i)+\mu\cdot[u(i+1,J)-u(i+1,J-1)-u(i,J)+u(i,J-1)]\}\cdot \boldsymbol{term}$$

*Η παραπάνω εξίσωση είναι αυτή που θα χρησιμοποιηθεί για την παραγωγή των συντελεστών ν του γραμμικού συστήματος στο πρόγραμμα.*

Όπως και για την περίπτωση της ταχύτητας u για σύγκλιση του αλγορίθμου SIMPLE απαιτείται να ισχύει η ανισότητα: $AP_v(i,J)\geq\sum_{nb}(a_{nb})$. Λόγω του σχήματος διαφορών, προκύτει τελικά ότι στην περίπτωσή μας ισχύει η ισότητα όπως αποδεικνύεται παρακάτω:

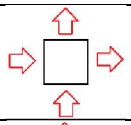
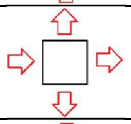
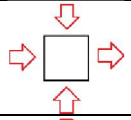
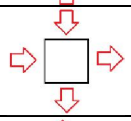
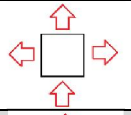
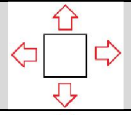
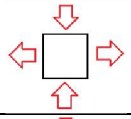
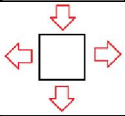
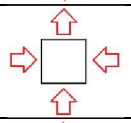
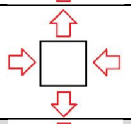
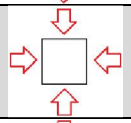
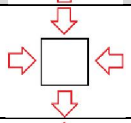
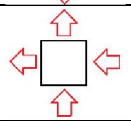
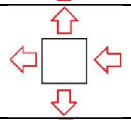
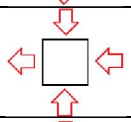
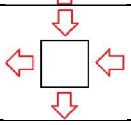
$$AP_v(i,J)=\sum_{nb}(a_{nb})\Rightarrow\left\{\frac{\max(F_e,0)}{x_{i+1}-x_i}+\frac{\max(-F_w,0)}{x_{i+1}-x_i}+\frac{\max(F_n,0)}{y_J-y_{J-1}}+\frac{\max(-F_s,0)}{y_J-y_{J-1}}+\frac{\mu}{(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}+\frac{\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_j-y_{j-1})}\right\}=\left\{\frac{\max(-F_e,0)}{x_{i+1}-x_i}+\frac{\mu}{(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}\right\}+\left\{\frac{\max(F_w,0)}{x_{i+1}-x_i}+\frac{\mu}{(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}\right\}+\left\{\frac{\max(-F_n,0)}{y_J-y_{J-1}}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right\}+\left\{\frac{\max(F_s,0)}{y_J-y_{J-1}}+\frac{2\mu}{(y_J-y_{J-1})\cdot(y_j-y_{j-1})}\right\}\Rightarrow$$

$$\frac{\max(F_e,0)}{x_{i+1}-x_i}+\frac{\max(-F_w,0)}{x_{i+1}-x_i}+\frac{\max(F_n,0)}{y_J-y_{J-1}}+\frac{\max(-F_s,0)}{y_J-y_{J-1}}=\frac{\max(-F_e,0)}{x_{i+1}-x_i}+\frac{\max(F_w,0)}{x_{i+1}-x_i}+\frac{\max(-F_n,0)}{y_J-y_{J-1}}+\frac{\max(F_s,0)}{y_J-y_{J-1}}$$

Εδώ θα πρέπει να ελέγξουμε τα πρόσημα των $F_e,F_w,F_n,F_s$ και προς αυτήν την κατεύθυνση κατασκευάζουμε τον παρακάτω πίνακα. Προφανώς σε κάθε περίπτωση ισχύει η εξίσωση διατήρησης της μάζας στον όγκο ελέγχου της ταχύτητας ν και αυτή απαγορεύει ορισμένους συνδυασμούς προσήμων (εκτός και εάν οι ταχύτητες είναι μηδενικές κατ' απόλυτη τιμή, βλ. παρακάτω με σκίαση).

$$div(\vec{u})=0\overset{CV\,(v)}{\Longrightarrow}\int_{CV}div(\vec{u})dV=0\Rightarrow\int_A\hat{n}\cdot\vec{u}\cdot dA=0\Rightarrow v_eA_e-v_wA_w+v_nA_n-v_sA_s=0\Rightarrow$$

$$F_eA_e-F_wA_w+F_nA_n-F_sA_s=0\Rightarrow\frac{F_e}{x_{i+1}-x_i}-\frac{F_w}{x_{i+1}-x_i}+\frac{F_n}{y_J-y_{J-1}}-\frac{F_s}{y_J-y_{J-1}}=0$$

| Ισχύει η εξίσωση | $F_e$ | $F_w$ | $F_n$ | $F_s$ | $CV\ (v)$ |
|---|---|---|---|---|---|
| ΝΑΙ | (+) | (+) | (+) | (+) |  |
| ΝΑΙ | (+) | (+) | (+) | (−) |  |
| ΝΑΙ | (+) | (+) | (−) | (+) |  |
| ΝΑΙ | (+) | (+) | (−) | (−) |  |
| ΝΑΙ | (+) | (−) | (+) | (+) |  |
| *ΝΑΙ* | (+) | (−) | (+) | (−) |  |
| ΝΑΙ | (+) | (−) | (−) | (+) |  |
| ΝΑΙ | (+) | (−) | (−) | (−) |  |
| ΝΑΙ | (−) | (+) | (+) | (+) |  |
| ΝΑΙ | (−) | (+) | (+) | (−) |  |
| *ΝΑΙ* | (−) | (+) | (−) | (+) |  |
| ΝΑΙ | (−) | (+) | (−) | (−) |  |
| ΝΑΙ | (−) | (−) | (+) | (+) |  |
| ΝΑΙ | (−) | (−) | (+) | (−) |  |
| ΝΑΙ | (−) | (−) | (−) | (+) |  |
| ΝΑΙ | (−) | (−) | (−) | (−) |  |

# 3.5 Επίλυση του πεδίου της πίεσης μέσω του αλγορίθμου SIMPLE

Ορίζουμε:

$$P = P^* + P'$$

$$u = u^* + u'$$

$$v = v^* + v'$$

Προφανώς ισχύουν:

$$u(i,J) = u^*(i,J) + u'(i,J)$$

$$u(i+1,J) = u^*(i+1,J) + u'(i+1,J)$$

$$v(I,j) = v^*(I,j) + v'(I,j)$$

$$v(I,j+1) = v^*(I,j+1) + v'(I,j+1)$$

Από προηγούμενες εξισώσεις ισχύει:

$$\begin{cases} u(i,J)\cdot AP_u(i,J) = \sum_{nb}(a_{nb}\cdot u_{nb}) + \dfrac{P(I-1,J)-P(I,J)}{x_I - x_{I-1}} + \underbrace{\dfrac{\mu}{(y_{j+1}-y_j)\cdot(x_I - x_{I-1})}\cdot[v(I,j+1)-v(I-1,j+1)-v(I,1)+v(I-1,j)]}_{\hat{b}(i,J)} \\ u^*(i,J)\cdot AP_u(i,J) = \sum_{nb}(a_{nb}\cdot u^*_{nb}) + \dfrac{P^*(I-1,J)-P^*(I,J)}{x_I - x_{I-1}} + \underbrace{\dfrac{\mu}{(y_{j+1}-y_j)\cdot(x_I - x_{I-1})}\cdot[v(I,j+1)-v(I-1,j+1)-v(I,1)+v(I-1,j)]}_{\hat{b}(i,J)} \end{cases}$$

$$\begin{cases} v(I,j)\cdot AP_v(I,j) = \sum_{nb}(a_{nb}\cdot v_{nb}) + \dfrac{P(I,J-1)-P(I,J)}{y_J - y_{J-1}} + \underbrace{\dfrac{\mu}{(x_{i+1}-x_i)\cdot(y_J - y_{J-1})}\cdot[u(i+1,J)-u(i+1,J-1)-u(i,J)+u(i,J-1)]}_{\hat{b}(I,j)} \\ v^*(I,j)\cdot AP_v(I,j) = \sum_{nb}(a_{nb}\cdot v^*_{nb}) + \dfrac{P^*(I,J-1)-P^*(I,J)}{y_J - y_{J-1}} + \underbrace{\dfrac{\mu}{(x_{i+1}-x_i)\cdot(y_J - y_{J-1})}\cdot[u(i+1,J)-u(i+1,J-1)-u(i,J)+u(i,J-1)]}_{\hat{b}(I,j)} \end{cases}$$

Με αφαίρεση κατά μέλη προκύπτει:

$$u'(i,J)\cdot AP_u(i,J) = \sum_{nb}(a_{nb}\cdot u'_{nb}) + \frac{P'(I-1,J)-P'(I,J)}{x_I - x_{I-1}} \Rightarrow u'(i,J)\cdot AP_u(i,J) \approx \frac{P'(I-1,J)-P'(I,J)}{x_I - x_{I-1}} \Rightarrow u'(i,J) \approx \frac{P'(I-1,J)-P'(I,J)}{AP_u(i,J)\cdot(x_I - x_{I-1})}$$

$$v'(I,j)\cdot AP_v(I,j) = \sum_{nb}(a_{nb}\cdot v'_{nb}) + \frac{P'(I,J-1)-P'(I,J)}{y_J - y_{J-1}} \Rightarrow v'(I,j)\cdot AP_v(I,j) \approx \frac{P'(I,J-1)-P'(I,J)}{y_J - y_{J-1}} \Rightarrow v'(I,j) \approx \frac{P'(I,J-1)-P'(I,J)}{AP_v(I,j)\cdot(y_J - y_{J-1})}$$

Από την εξίσωση συνέχειας στον όγκο ελέγχου της πίεσης έχουμε:

$$div(\vec{u}) = 0 \overset{CV\ (P)}{\Longrightarrow} \int_{CV} div(\vec{u})dV = 0 \Rightarrow \int_A \hat{n}\cdot\vec{u}\cdot dA = 0 \Rightarrow u(i+1,J)\cdot A_e - u(i,J)\cdot A_w + v(I,j+1)\cdot A_n - v(I,j)\cdot A_s = 0 \Rightarrow$$

$$\Rightarrow \frac{u(i+1,J)-u(i,J)}{x_{i+1}-x_i} + \frac{v(I,j+1)-v(I,j)}{y_{j+1}-y_j} = 0 \Rightarrow$$

$$\Rightarrow \frac{u^*(i+1,J)+u'(i+1,J)-u^*(i,J)-u'(i,J)}{x_{i+1}-x_i} + \frac{v^*(I,j+1)+v'(I,j+1)-v^*(I,j)-v'(I,j)}{y_{j+1}-y_j} = 0 \Rightarrow$$

$$\Rightarrow \left[u^*(i+1,J) + \frac{P'(I,J)-P'(I+1,J)}{AP_u(i+1,J)\cdot(x_{I+1}-x_I)} - u^*(i,J) - \frac{P'(I-1,J)-P'(I,J)}{AP_u(i,J)\cdot(x_I - x_{I-1})}\right]\cdot\frac{1}{x_{i+1}-x_i} +$$

$$+\left[v^*(I,j+1) + \frac{P'(I,J)-P'(I,J+1)}{AP_v(I,j+1)\cdot(y_{J+1}-y_J)} - v^*(I,j) - \frac{P'(I,J-1)-P'(I,J)}{AP_v(I,j)\cdot(y_J - y_{J-1})}\right]\cdot\frac{1}{y_{j+1}-y_j} = 0 \Rightarrow$$

$$P'(I,J)\cdot\left[\frac{1}{AP_u(i+1,J)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}+\frac{1}{AP_u(i,J)\cdot(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}+\frac{1}{AP_v(I,j+1)\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)}+\frac{1}{AP_v(I,j)\cdot(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right]=$$

$$=P'(I+1,J)\cdot\left[\frac{1}{AP_u(i+1,J)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)}\right]+P'(I-1,J)\cdot\left[\frac{1}{AP_u(i,J)\cdot(x_I-x_{I-1})\cdot(x_{i+1}-x_i)}\right]+$$

$$+P'(I,J+1)\cdot\left[\frac{1}{AP_v(I,j+1)\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)}\right]+P'(I,J-1)\cdot\left[\frac{1}{AP_v(I,j)\cdot(y_J-y_{J-1})\cdot(y_{j+1}-y_j)}\right]+$$

$$+\frac{u^*(i,J)-u^*(i+1,J)}{x_{i+1}-x_i}+\frac{v^*(I,j)-v^*(I,j+1)}{y_{j+1}-y_j}$$

Για απαλοιφή των κλασμάτων, πολλαπλασιάζουμε με:

$$AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)\cdot(y_J-y_{J-1})$$

Και έχουμε:

$$P'(I,J)\cdot\left[AP_u(i,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)\cdot(y_J-y_{J-1})+AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)\cdot(y_J-y_{J-1})+AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)\cdot(x_I-x_{I-1})\cdot(y_J-y_{J-1})+AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)\right]$$

$$=P'(I+1,J)\cdot AP_u(i,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)\cdot(y_J-y_{J-1})+$$

$$+P'(I-1,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(y_{J+1}-y_J)\cdot(y_{j+1}-y_j)\cdot(y_J-y_{J-1})+$$

$$+P'(I,J+1)\cdot AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)\cdot(x_I-x_{I-1})\cdot(y_J-y_{J-1})+$$

$$+P'(I,J-1)\cdot AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot(x_{I+1}-x_I)\cdot(x_{i+1}-x_i)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)+$$

$$+\{[u^*(i,J)-u^*(i+1,J)]\cdot(y_{j+1}-y_j)+[v^*(I,j)-v^*(I,j+1)]\cdot(x_{i+1}-x_i)\}\cdot AP_u(i,J)\cdot AP_u(i+1,J)\cdot AP_v(I,j+1)\cdot AP_v(I,j)\cdot(x_{I+1}-x_I)\cdot(x_I-x_{I-1})\cdot(y_{J+1}-y_J)\cdot(y_J-y_{J-1})$$

*Η παραπάνω εξίσωση είναι αυτή που θα χρησιμοποιηθεί για την παραγωγή των συντελεστών pp του γραμμικού συστήματος στο πρόγραμμα.*

## 4.1 Πρόβλημα διαγώνιας υπεροχής – Συνάρτηση επίλυσης diag5solver

Έχοντας ετοιμάσει όλες τις παραπάνω εξισώσεις ακολουθεί η εφαρμογή του αλγορίθμου SIMPLE για την επίλυση του πεδίου. Το πρόβλημα όμως που προκύπτει είναι ότι με βάση το σχήμα (ανάντη) διαφορών που έχουμε επιλέξει οι συντελεστές των γραμμικών συστημάτων AP είναι ίσοι κατ' απόλυτη τιμή (θεωρητικά) με το άθροισμα των αντίστοιχων υπόλοιπων συντελεστών στην εξίσωση όπως έχει περιγραφεί και παραπάνω. Δηλαδή:

$$AP = AE + AW + AN + AS$$

Εξάλλου, λόγω των αρχικών υποθετικών πεδίων, αλλά και των "ενδιάμεσων" του αλγορίθμου SIMPLE, που δεν ανταποκρίνονται κατ' ανάγκη στην πραγματικότητα (δηλαδή δεν παράγουν υπαρκτά (δυνατά) πεδία τιμών) ώστε να περιορίζονται από τις εξισώσεις της θεωρίας, οι συντελεστές AP πολλές φορές προκύπτουν μικρότεροι από το εν λόγω άθροισμα, δηλαδή:

$$AP < AE + AW + AN + AS$$

Επομένως μια επαναληπτική διαδικασία για την επίλυση των συστημάτων αυτών, αν και θα ήταν πολύ πιο γρήγορη, δεν εγγυάται σύγκλιση, και όπως προέκυψε, δίνει λανθασμένα αποτελέσματα, ή δεν συγκλίνει καθόλου, λόγω μή ύπαρξης διαγώνιας υπεροχής στον πίνακα του γραμμικών εξισώσεων του συστήματος.

Για να αποφευχθεί το πρόβλημα αυτό, το σύστημα μπορεί να επιλυθεί με την κλασσική μέθοδο *Gauss*, με μερική οδήγηση κατά στήλη για αυξημένη ακρίβεια.

Επειδή η μέθοδος *Gauss* απαιτεί την κατασκευή πίνακα $n \cdot n$ (n είναι ο συνολικός αριθμός των κόμβων του πεδίου) και γίνονται πράξεις σε όλη την έκταση του πίνακα, γίνεται σπατάλη πόρων μνήμης και επεξεργαστικής ισχύος, αφού ο πίνακας του συστήματος είναι πεντα-διαγώνιος, και άρα περιέχει πολλά μηδενικά στοιχεία. Για να επιταχυνθεί η διαδικασία επινοήθηκε αλγόριθμος (συνάρτηση "*diag5solver*" στο πρόγραμμα) ο οποίος χρησιμοποιεί μόνο όσες θέσεις μνήμης χρειάζεται ***ακριβώς*** η μέθοδος επίλυσης *Gauss* ενώ δεν γίνεται σπατάλη χρόνου εκτελώντας πράξεις σε μηδενικά στοιχεία. Έτσι έχουμε:

- **ακριβή** λύση του πεντα-διαγώνιου συστήματος, με τη μέθοδο επίλυσης *Gauss* με μερική οδήγηση κατά στήλη για αυξημένη ακρίβεια,
- στον **ελάχιστο** δυνατό χρόνο, μη κάνοντας πράξεις σε στοιχεία του πίνακα όπου δεν απαιτείται
- με τη **μικρότερη** δυνατή κατανάλωση μνήμης (κάτι εξαιρετικά σημαντικό όταν ο αριθμός των κόμβων γίνεται πολύ μεγάλος) και,
- **χωρίς** τον περιορισμό της αυστηρής διαγώνιας υπεροχής που επιβάλλουν οι επαναληπτικές μέθοδοι.

Σε κάθε περίπτωση όμως, μια επαναληπτική μέθοδος είναι και πάλι ταχύτερη και μάλιστα ο αριθμός των επαναλήψεων που απαιτείται για τη λύση γίνεται μικρότερος όσο η διαφορά των απολύτων τιμών του $AP$ με το άθροισμα των υπολοίπων συντελεστών γίνεται μεγαλύτερη.

Η συνάρτηση καθώς και η σύγκρισή της με την πλήρη μέθοδο *Gauss*, περιγράφονται αναλυτικά στο αντίστοιχο παράρτημα.

## 4.2 Πρόβλημα μη σύγκλισης αλγορίθμου

Ακόμα και με ακριβή επίλυση των γραμμικών συστημάτων για u, v, p, έγινε γρήγορα φανερό ότι δεν είχαμε σύγκλιση του αλγορίθμου SIMPLE στο σύνολό του όταν ο αριθμός των κόμβων γινόταν μεγάλος. Κατόπιν εξέτασης όλης της θεωρίας εκ νέου, επειδή δεν κατέστη δυνατό να εντοπιστεί το σφάλμα, αφαιρέθηκε από το πρόγραμμα το κομμάτι υπολογισμού του πεδίου της πίεσης. Αυτό επέτρεψε τελικά στο πρόγραμμα να συγκλίνει προβλέψιμα και για κάθε επιλογή πλέγματος (μικρού ή μεγάλου), για τις ταχύτητες u, v.

Για το λόγο αυτό, τα προγράμματα που παρουσιάζονται στην παρούσα εργασία, δεν διαθέτουν το κομμάτι υπολογισμού των διαφορών πιέσεων που περιέχεται κανονικά στον αλγόριθμο SIMPLE. Το πρόγραμμα σε FORTRAN χρησιμοποιεί για επίλυση των γραμμικών συστημάτων τη μέθοδο επίλυσης *Gauss* έχοντας μετατρέψει τα διανύσματα σε πεντα-διαγώνιο πίνακα, καθότι κατέστη πολύ δύσκολη η μετατροπή της συνάρτησης *diag5solver* που είχε γραφτεί αρχικά σε C για τη FORTRAN με δεδομένους όλους τους περιορισμούς, παράλογους *ή λιγότερο παράλογους*, που αυτή η γλώσσα επιβάλλει. Για τη FORTRAN χρειάστηκε επίσης, οι subroutines μετατροπής των διανυσμάτων σε πίνακα και επίλυσης να περιληφθούν σε μια δομή τύπου modules καθώς αυτός ήταν ο μόνος τρόπος να μεταγλωττιστεί (compile) το πρόγραμμα.

Το πρόβλημα όπως γίνεται φανερό είναι αλγοριθμικό και δεν έχει να κάνει με συγκεκριμένη γλώσσα προγραμματισμού. Παρατίθενται παρόλα αυτά στα αντίστοιχα παραρτήματα τα αρχικά προγράμματα που κατασκευάστηκαν και περιγράφουν το πρόβλημα στο σύνολό του ακόμα και με το πρόβλημα σύγκλισης.

Για τη σύγκλιση των προγραμμάτων ήταν αρκετό να αφαιρεθεί από τα προγράμματα το κομμάτι του αλγορίθμου SIMPLE που υπολογίζει το νέο πεδίο της πίεσης. Για τον έλεγχο της επιτάχυνσης μέσω παραλληλοποίησης όμως τελικά χρησιμοποιήθηκε μόνο το τμήμα του υπολογισμού των συντελεστών καθώς όπως θα φανεί και παρακάτω η συνάρτηση επίλυσης του γραμμικού συστήματος χρησιμοποιεί τάξεις μεγέθους περισσότερο χρόνο για να ολοκληρωθεί, σε σχέση με τον υπολογισμό των συντελεστών και είναι μια συνάρτηση που στην παρούσα της μορφή δεν μπορεί να παραλληλοποιηθεί (τουλάχιστον όχι πολύ εύκολα) γιατί είναι μη-επαναληπτική, ακριβής μέθοδος επίλυσης γραμμικού συστήματος με τη μέθοδο GAUSS, και είναι ήδη βελτιστοποιημένη για την επίλυση πεντα- διαγώνιου συστήματος. Το σχήμα διαφορών και η μαθηματική θεωρία που έχουμε αναπτύξει δεν μας επιτρέπουν τη χρήση μιας επαναληπτικής μεθόδου για τον υπολογισμό των πεδίων, κάτι το οποίο θα μπορούσε να επιταχυνθεί μέσω παραλληλοποίησης (ενδεικτικά υπάρχουν στη βιβλιογραφία πολλές υλοποιήσεις του επαναληπτικού αλγορίθμου επίλυσης γραμμικών συστημάτων Jacobi σε παράλληλο περιβάλλον) λόγω της ισότητας $AP = AE + AW + AN + AS$ .

Στην επόμενη ενότητα θα δούμε αναλυτικά τους χρόνους για τον υπολογισμό των συντελεστών και για την επίλυση των γραμμικών συστημάτων σε ένα ενδεικτικό hardware για το πρόγραμμα γραμμένο σε C & FORTRAN.

## 5.1 Περιγραφή αλγορίθμου και μέτρησης χρόνου για υπολογισμό πεδίων u, v μόνο

Αρχικά ο αλγόριθμος αποτυπώθηκε σε σειριακό πρόγραμμα γραμμένο σε γλώσσα C:

- Ορίζονται ορισμένα μεγέθη (πυκνότητα, ιξώδες),
- ορίζεται ο αριθμός των κόμβων κατά x, y,
- παράγεται δισδιάστατο το πλέγμα,
- γίνεται δέσμευση της απαιτούμενης μνήμης,
- ορίζεται ένα αρχικό πεδίο τιμών για τα πεδία u, v, p.

Έπειτα ξεκινούν οι επαναλήψεις του αλγορίθμου SIMPLE σε έναν επαναληπτικό βρόγχο για τον υπολογισμό των τιμών των συντελεστών των εσωτερικών κόμβων για τα συστήματα των u, v, ώστε ελαχιστοποιείται κατά το δυνατόν ο χρόνος υπολογισμού (πχ ορισμένα γεωμετρικά μεγέθη χρησιμοποιούνται για τον υπολογισμό των συντελεστών και για τα δύο πεδία).

Το πρόγραμμα συμπληρώνει κατόπιν τις τιμές για τους συντελεστές των ακραίων κόμβων σύμφωνα με τις οριακές συνθήκες που έχουμε ορίσει.

Τέλος γίνεται επίλυση των πεδίων τιμών με χρήση της ακριβούς μεθόδου επίλυσης (*diag5solver*) ώστε να αποφεύγονται τα προβλήματα από την έλλειψη διαγώνιας υπεροχής στα συστήματα, και γίνεται έλεγχος για σύγκλιση της μεθόδου SIMPLE.

Σε αυτό το σημείο θα πρέπει να τονιστεί ότι ο compiler του κώδικα (για οποιαδήποτε πλατφόρμα, αρχιτεκτονική ή λειτουργικό σύστημα) μπορεί να βελτιστοποιήσει τον παραγόμενο *“κώδικα μηχανής”* εάν για τη δεδομένη αρχιτεκτονική στην οποία μεταγλωττίζει και για δεδομένο *πηγαίο κώδικα* μπορεί να βρει (αυτόματα) τρόπους για ταχύτερη εκτέλεση, μειωμένη χρήση μνήμης ή μειωμένο μέγεθός εκτελέσιμου αρχείου εις βάρος του χρόνου μεταγλώττισης. Αυτή η διαδικασία είναι αυτόματη, οι βελτιώσεις δεν είναι δυνατόν να παραχθούν από το χρήστη στον κώδικα, και άπτεται στην ικανότητα του compiler. Για το συγκεκριμένο project ο compiler που χρησιμοποιήθηκε είναι η έκδοση 5.1.0 του *GNU GCC* για Windows που περιέχεται στο *CodeBlocks IDE* v17.12. Οι βελτιώσεις σε ταχύτητα περνώντας τα κατάλληλα *“flags”* κατά το compilation μπορεί να είναι δραματικές αλλά ενδέχεται να υπάρχει κόστος ως προς την ακρίβεια των υπολογισμών, όπως θα φανεί παρακάτω. Σε κάθε περίπτωση μπορεί να γίνει ο βέλτιστος συγκερασμός των δυο αυτών αντικρουόμενων κριτηρίων.

Η μέτρηση χρόνου που επιβάλλει (υποχρεωτικά) το πρότυπο της C σε όλες τις υλοποιήσεις της δεν έχει ικανοποιητική ανάλυση, ενώ υπάρχει πληθώρα μη-standard συναρτήσεων που μετρούν το χρόνο με διαφορετικό τρόπο (πχ η συνάρτηση `clock()` μετράει CPU time βάσει προτύπου, αλλά η υλοποίησή της σε Windows δίνει αποτελέσματα σε Wall time). Για τις παρακάτω μετρήσεις χρησιμοποιήθηκε η συνάρτηση `gettimeofday()` που μας επιτρέπει (θεωρητική) ανάλυση microsecond. Οπωσδήποτε οι μετρήσεις χρόνου στο πρόγραμμά μας μπορούν να βελτιωθούν όπως θα φανεί και παρακάτω, αλλά τα συμπεράσματα που εξάγουμε είναι έγκυρα ανεξαρτήτως επιλεχθείσας συνάρτησης.

Αρχικά έγινε μια ενδεικτική εκτέλεση του σειριακού προγράμματος, μεταγλωττισμένου χωρίς optimization flags για 100 κόμβους κατά x και 50 κόμβους κατά y, άρα για n=5000, μετρώντας το χρόνο για κάθε τμήμα εκτέλεσης του προγράμματος:

- Προαπαιτούμενες ενέργειες (ορισμός μεταβλητών, δέσμευση μνήμης κτλ)
- Υπολογισμός συντελεστών γραμμικών συστημάτων

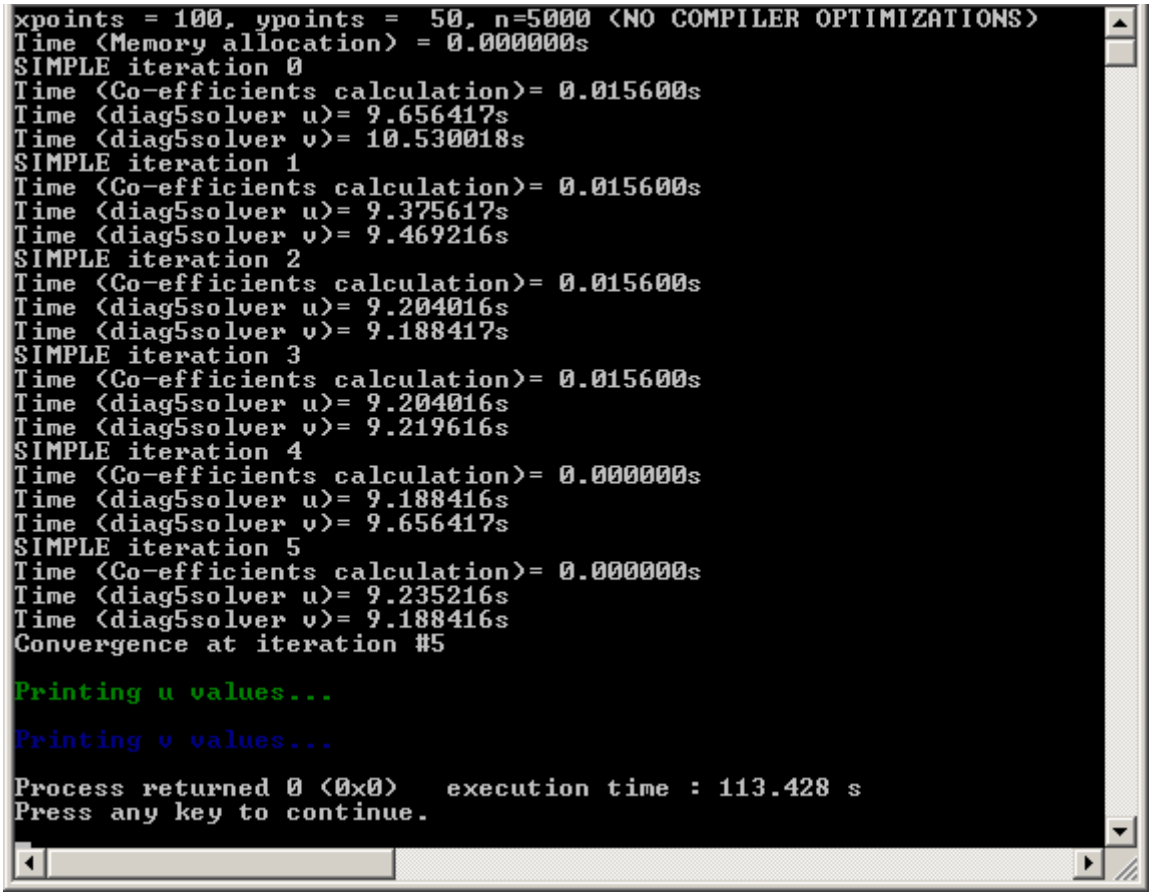
- Επίλυση γραμμικού συστήματος για u
- Επίλυση γραμμικού συστήματος για v

Οι εκτελέσεις του προγράμματος έγιναν σε ένα μικρό netbook (το οποίο έτυχε να χρησιμοποιείται και για τη συγγραφή της συγκεκριμένης εργασίας κατά την περίοδο αυτή) με διπύρηνο (4 threads) επεξεργαστή Intel Atom N2800 (x86) χρονισμένο στα 1.86GHz (98), 4GB DDR3-1333 RAM, σε σύνηθες desktop περιβάλλον Windows 7 SP1 x86 Home edition με πλήθος άλλων εφαρμογών να εκτελούνται ταυτόχρονα.

Όπως φαίνεται και στο παρακάτω screenshot, ο χρόνος υπολογισμού για ορισμένα τμήματα του κώδικα εμφανίζεται μηδενικός, προφανώς λόγω περιορισμών της ανάλυσης της συνάρτησης ή/και περιορισμών υλικού. Επίσης, λόγω της δομής του προγράμματος ο χρόνος υπολογισμού των συντελεστών των γραμμικών συστημάτων περιλαμβάνει και τμήμα των ενεργειών για έλεγχο της σύγκλισης, το οποίο είναι και αυτό εξαιρετικά μικρού χρόνου υπολογισμού. Αυτό όμως δεν επηρεάζει το κύριο συμπέρασμα που είναι ότι:

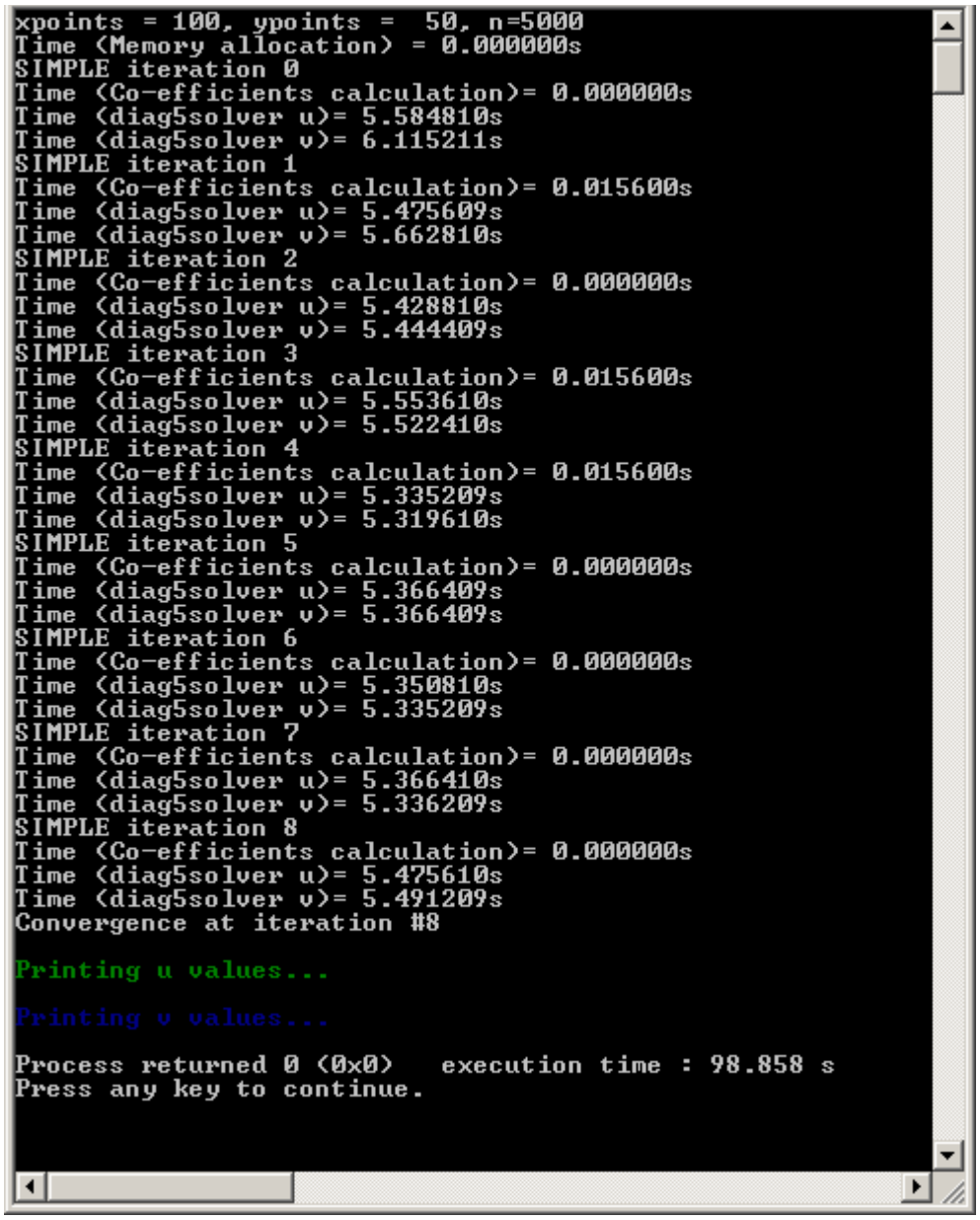
*Η επίλυση των συστημάτων απαιτεί τάξεις μεγέθους μεγαλύτερο χρόνο από τον υπολογισμό των συντελεστών.*

Αυτό είναι ένα πρόβλημα που στην παρούσα κατάσταση (με τις δεδομένες εξισώσεις παραγωγής συντελεστών) δεν μπορούμε να αποφύγουμε, αφού όπως έχει ήδη αναφερθεί, δεν εξασφαλίζεται διαγώνια υπεροχή στο σύστημα.




*Εικόνα 1: Ενδεικτική εκτέλεση σειριακού προγράμματος με μετρήσεις χρόνου (n=5,000, C, no compiler optimizations) [Intel Atom N2800]*

Το ενδιαφέρον κομμάτι ως προς τα optimizations που μπορεί να κάνει ο compiler κατά τη μεταγλώττιση φαίνεται στο επόμενο screenshot: Περνώντας το *-O3 flag* στον *GCC* κατά τη μεταγλώττιση, οι χρόνοι εκτέλεσης του loop του ίδιο ακριβώς προγράμματος είναι δραματικά μειωμένοι, με το αντίστοιχο κόστος ως προς την ακρίβεια, που αποτυπώνεται στο μεγαλύτερο αριθμό επαναλήψεων του αλγορίθμου SIMPLE για την ικανοποίηση του ίδιου κριτηρίου σύγκλισης (το οποίο και αυτό είναι μόνο ενδεικτικό και σε καμία περίπτωση το βέλτιστο για τη χρήση που κάνουμε). Ο συνολικός χρόνος εκτέλεσης του προγράμματος είναι μειωμένος.




*Εικόνα 2: Ενδεικτική εκτέλεση σειριακού προγράμματος με μετρήσεις χρόνου (n=5,000, C, -o3 flag) [Intel Atom N2800]*

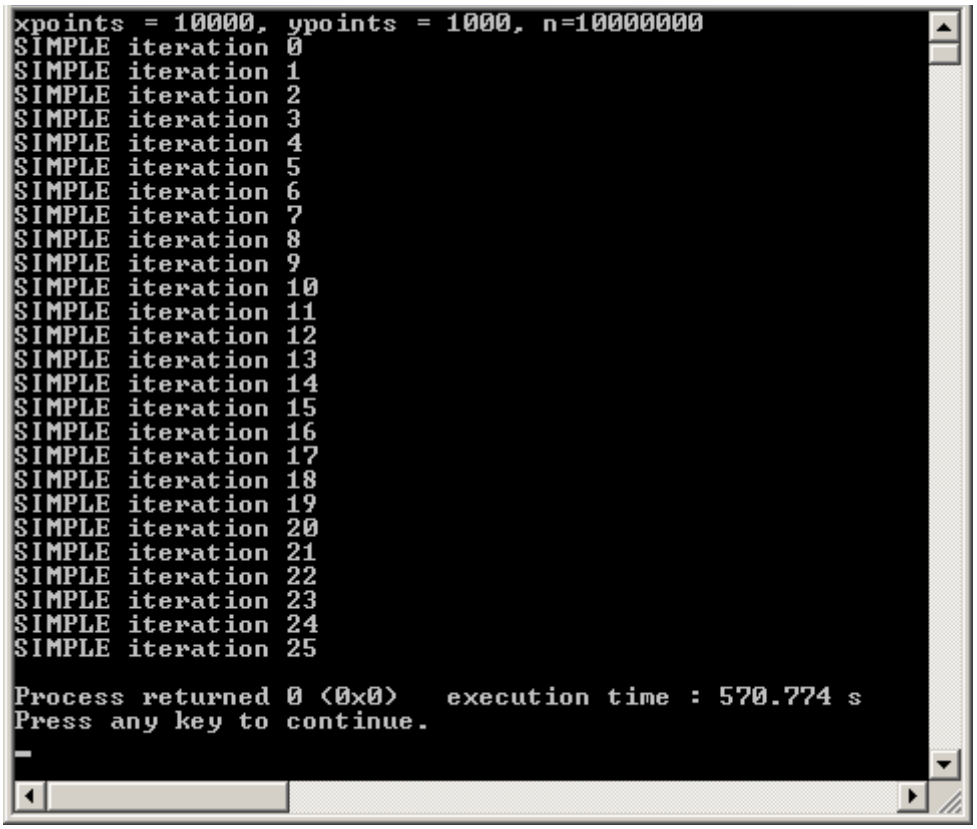
## 6.1 Ανάλυση χρόνων υπολογισμού συντελεστών γραμμικών συστημάτων για τη σειριακή υλοποίηση

Επειδή όπως αναφέρθηκε και παραπάνω *η επίλυση των συστημάτων απαιτεί τάξεις μεγέθους μεγαλύτερο χρόνο από τον υπολογισμό των συντελεστών,* προκειμένου να μετρήσουμε τις διαφορές σε χρόνο εκτέλεσης του παράλληλου προγράμματος έναντι του σειριακού, αφαιρέθηκε από το πρόγραμμα το κομμάτι επίλυσης των γραμμικών συστημάτων. Εξάλλου η επίλυση του συστήματος με την επιλεχθείσα (ακριβή) μέθοδο είναι κάτι που δεν παραλληλοποιείται, ενώ σε περίπτωση υπολογισμών των συντελεστών με πιο περίπλοκα συστήματα διαφορών, παύει μεν να καταναλώνει ποσοστιαία το μεγαλύτερο κομμάτι της χρονικής εκτέλεσης του προγράμματος, αφ' ετέρου μπορεί να αντικατασταθεί με παραλληλοποιήσιμες επαναληπτικές μεθόδους, εφόσον εξασφαλίζεται η διαγώνια υπεροχή του πίνακα.

Με δεδομένα τα παραπάνω, το πρόγραμμα τροποποιήθηκε ώστε να εκτελεί 25 επαναλήψεις υπολογίζοντας απλά τους συντελεστές των γραμμικών συστημάτων για ένα πλέγμα 10,000,000 κόμβων (10,000 σημεία κατά x και 1,000 σημεία κατά y).

Τα αποτελέσματα των χρόνων εκτέλεσης του σειριακού προγράμματος γραμμένου σε γλώσσες C & FORTRAN, για το μηχάνημα που περιγράφηκε παραπάνω έχουν δοθεί συγκεντρωτικά στον παρακάτω πίνακα, και παρόλο που σαν απόλυτα νούμερα δεν έχουν ουσιαστική χρησιμότητα, οι ποσοστιαίες διαφορές τους με αυτών από την παράλληλη υλοποίηση του προγράμματος αποτελούν τη βάση για ενδιαφέροντα συμπεράσματα.

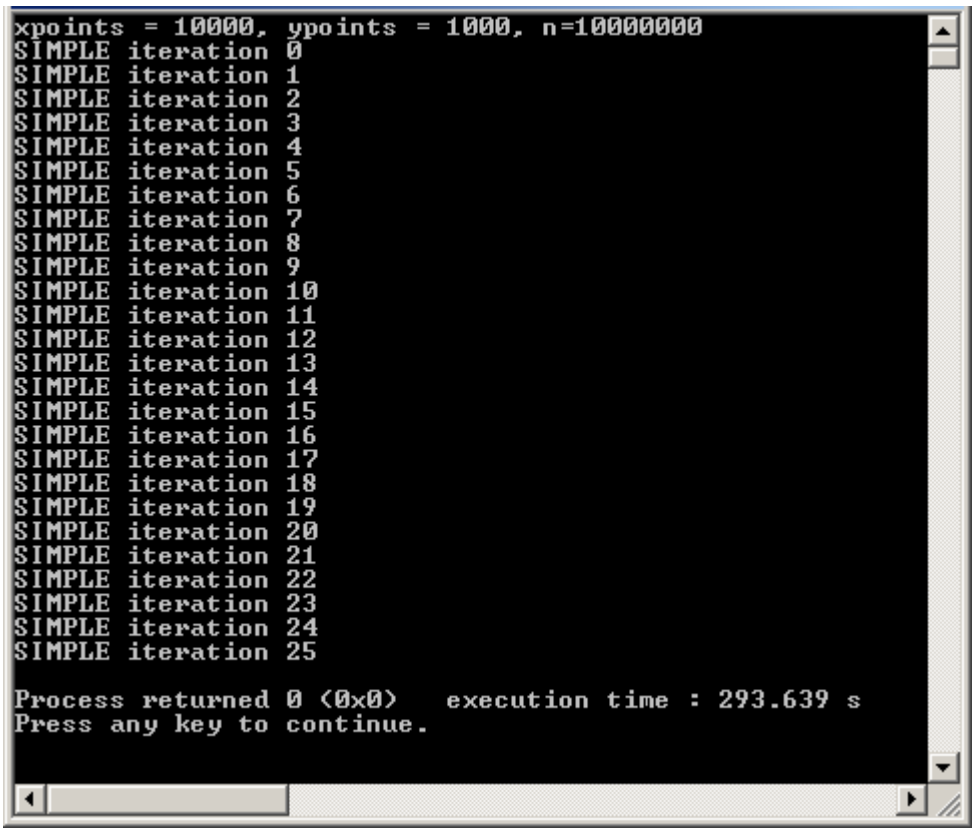
Η υλοποίηση σε FORTRAN είναι άμεση μετατροπή των εντολών του προγράμματος σε C δεδομένου ότι πρόκειται για δύο imperative γλώσσες και σε απλά προγράμματα υπάρχει εύκολη αντιστοιχία. Ο compiler της FORTRAN είναι ο *GNU FORTRAN compiler* 6.3.0-1 (*gfortran*). Βλέπουμε ότι το πρόγραμμα είναι ταχύτερο από αυτό γραμμένο σε C, κάτι που δεν είναι σπάνιο για διαφορετικές γλώσσες προγραμματισμού ή/και compilers.



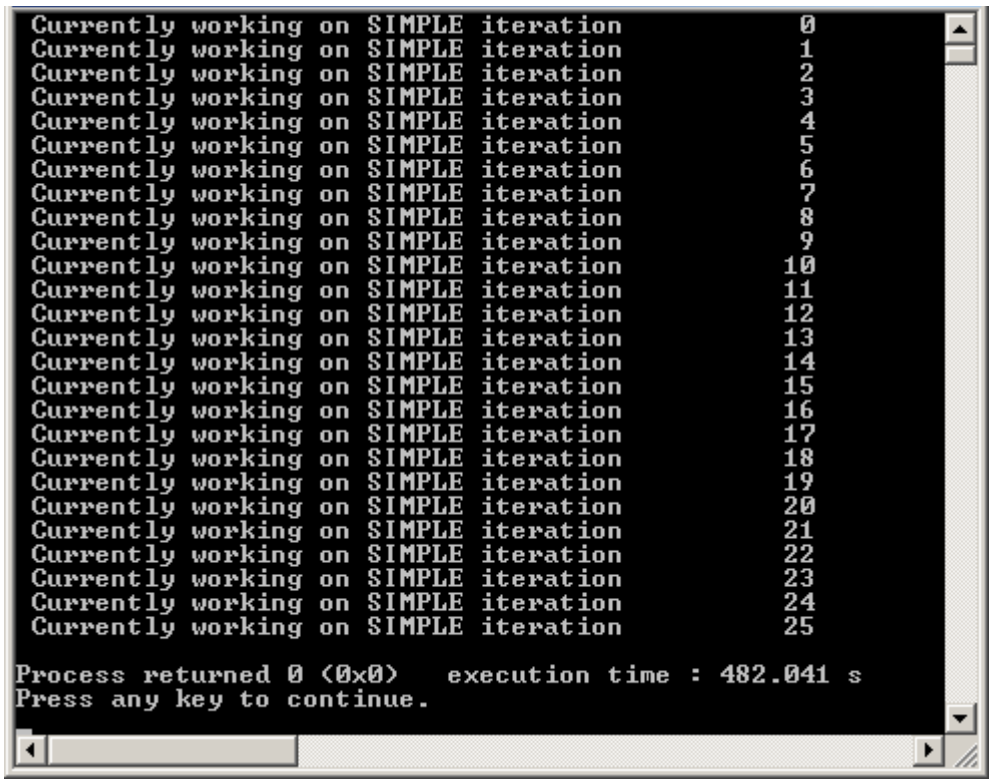

*Εικόνα 3: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε σειριακή υλοποίηση (C, no compiler optimizations) [Intel Atom N2800]*

*Πίνακας 2: Εκτέλεση προγράμματος υπολογισμού συντελεστών σε σειριακή υλοποίηση (C & FORTRAN) [Intel Atom N2800]*

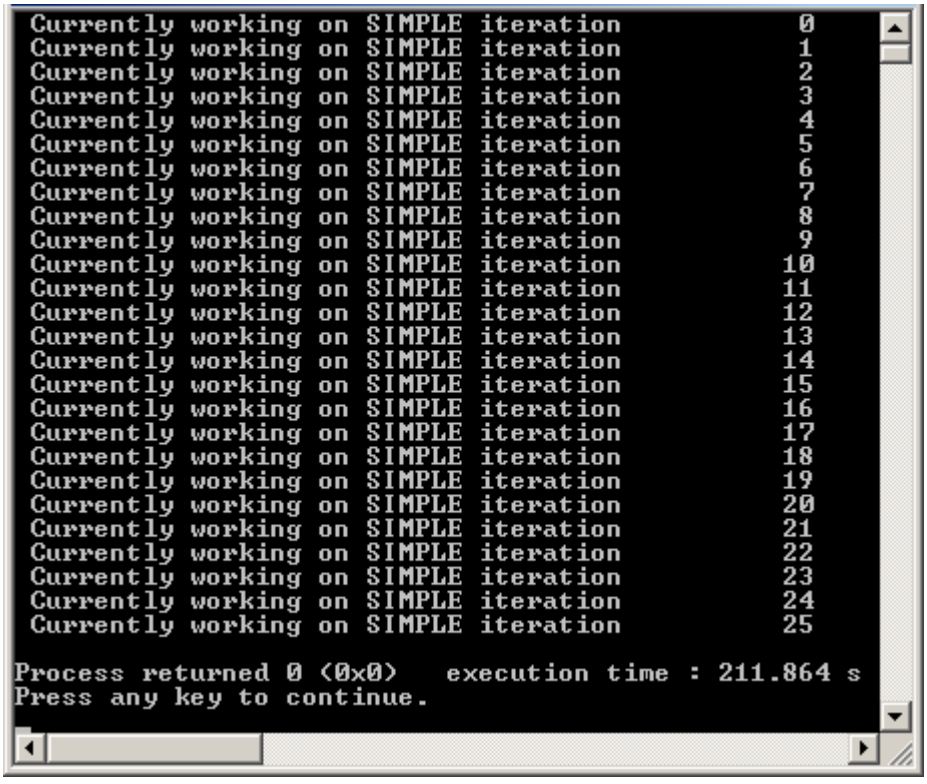
| *n=10,000,000. 25 Iterations* | *Σειριακό σε C (unoptimized)* | *Σειριακό σε C (-o3 flag)* | *Σειριακό σε FORTRAN (unoptimized)* | *Σειριακό σε FORTRAN (-o3 flag)* |
|---|---|---|---|---|
| Run #1 | 571.616 | 294.326 | 481.604 | 212.348 |
| Run #2 | 570.961 | 293.889 | 482.805 | 212.488 |
| Run #3 | 570.789 | 293.827 | 482.088 | 211.474 |
| Run #4 | 570.774 | 293.639 | 482.041 | 211.864 |
| Μέσος όρος χρόνων | *571.035* | *293.920* | *482.135* | *212.044* |




*Εικόνα 4: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε σειριακή υλοποίηση (C, -o3 flag) [Intel Atom N2800]*




*Εικόνα 5: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε σειριακή υλοποίηση (FORTRAN, no compiler flags) [Intel Atom N2800]*




*Εικόνα 6: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε σειριακή υλοποίηση (FORTRAN, -o3 flag) [Intel Atom N2800]*

Θα πρέπει να τονιστεί σε αυτό το σημείο ότι ο μετρούμενος χρόνος σε αυτήν την περίπτωση γίνεται με μεθόδους που ορίζει το περιβάλλον ανάπτυξης (*CodeBlocks IDE* v17.12), τοποθετώντας σημεία μέτρησης χρόνου στην αρχή και το τέλος του προγράμματος, περιλαμβάνοντας *όλες τις διαδικασίες* που αυτό εκτελεί και όχι μόνο το βρόγχο επανάληψης. Βλέπουμε και σε αυτήν την περίπτωση τη δραματική επιρροή των *optimization flags* στο χρόνο εκτέλεσης του προγράμματος.

## 6.2 Ανάλυση χρόνων υπολογισμού συντελεστών γραμμικών συστημάτων για την παράλληλη υλοποίηση

Για παράλληλη επεξεργασία μέσω OpenCL είναι απαραίτητη η εγκατάσταση μιας "*πλατφόρμας*" ή "*υλοποίησης*" του προτύπου στο λειτουργικό σύστημα από κάποιον κατασκευαστή. Είναι επίσης απαραίτητη η σύνδεση μέσω του *linker* (κατόπιν της μεταγλώττισης) των αντίστοιχων *libraries* στο πρόγραμμα μέσω των οποίων έχουμε πρόσβαση στο *OpenCL runtime* και τους πόρους που αυτό μας διαθέτει.

Για το συγκεκριμένο project χρησιμοποιήθηκε η πλατφόρμα της AMD (*AMD-APP-SDKInstaller-v3.0.130.135-GA-windows-F-x86*) που είναι ελεύθερα προσβάσιμη από το site του κατασκευαστή, και υλοποιεί την έκδοση *OpenCL v2.0*. Οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι αυτές που περιλαμβάνει το SDK (η x86 έκδοσή τους στην παρούσα φάση λόγω αρχιτεκτονικής του επεξεργαστή), ενώ τα header files που ορίζουν τις εσωτερικές συναρτήσεις της OpenCL είναι αυτά που προσφέρει το *Khronos Group* (έκδοσης v2.1 του προτύπου OpenCL). Ο επεξεργαστής του συστήματος δηλώνει υποστήριξη του *FULL PROFILE* της OpenCL έκδοσης 1.2 (δεν μπορούν σε αυτήν τη συσκευή να χρησιμοποιηθούν συναρτήσεις που ορίζονται σε μεταγενέστερα πρότυπα). Ο επεξεργαστής είναι 2 πυρήνων με τεχνολογία Intel hyperthreading που εμφανίζει στο λειτουργικό σύστημα 4 εικονικούς πυρήνες που αντιστοιχούν στα *Compute Units* της OpenCL.

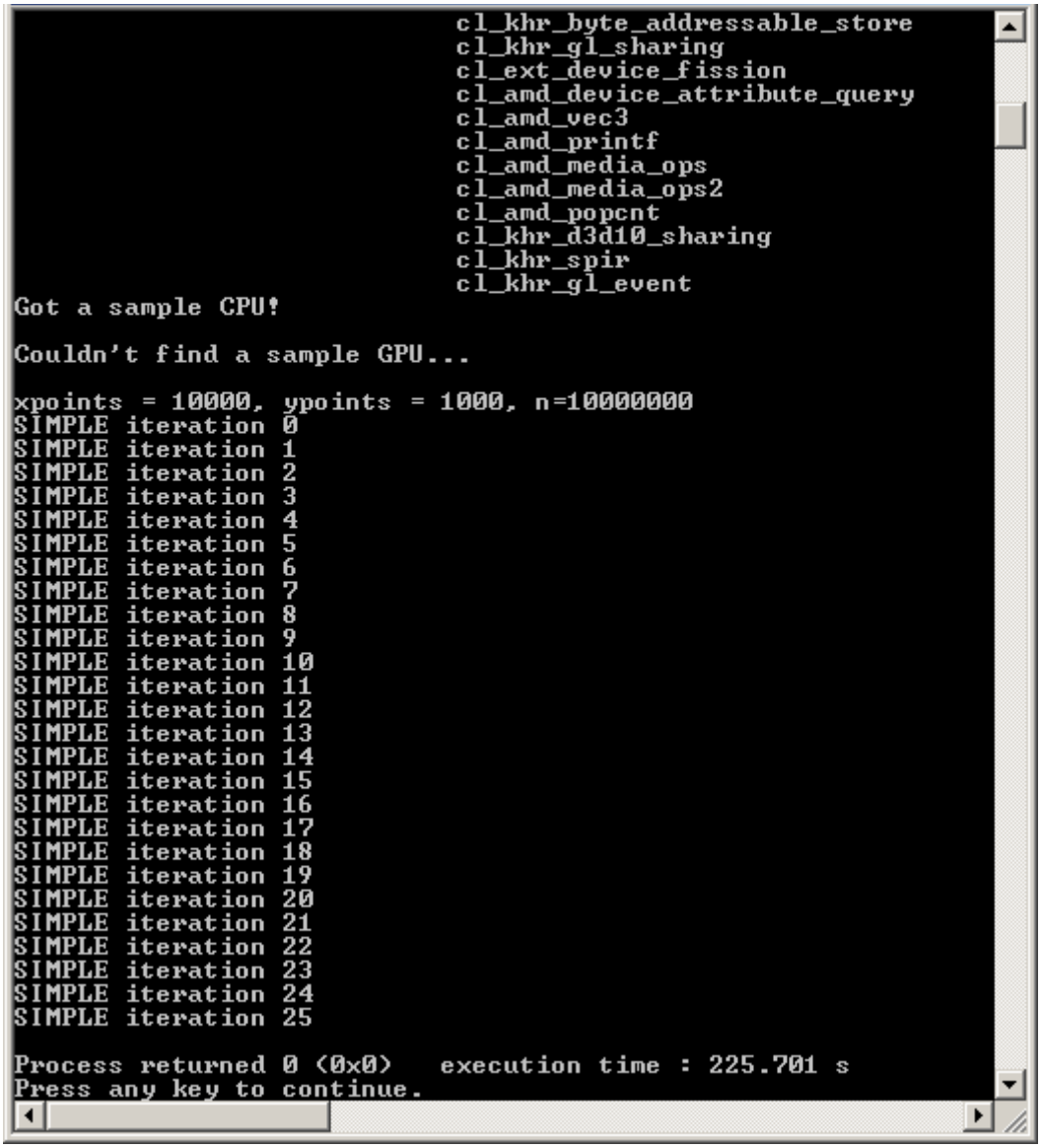
Για διευκόλυνση του χρήστη η παράλληλη υλοποίησή του προγράμματος εμφανίζει σε κάθε εκτέλεσή της ένα πλήθος χρήσιμων πληροφοριών για το εκάστοτε υλικό και την υποστήριξη του προτύπου. Επίσης για την χρήση των πόρων της OpenCL απαιτείται ο ορισμός κάποιων βασικών δομών δεδομένων (ενδεικτικά: *command queue, context, platform, devices, program, kernel* κλπ) το οποίο γίνεται στο παρόν πρόγραμμα μια φορά στην εκκίνηση. Αυτά προσθέτουν μια μικρή χρονοκαθυστέρηση σε κάθε εκτέλεση του προγράμματος κάτι το οποίο θα πρέπει να ληφθεί υπ' όψιν κατά την ανάλυση των αποτελεσμάτων.

Ο compiler για τον κώδικα σε OpenCL είναι *run-time only* δηλαδή επικαλείται και μεταγλωττίζει κατά την εκτέλεση του host προγράμματος και δεν είναι γενικά προσβάσιμος στο χρήστη, πλήν όμως δέχεται και αυτός optimization flags. Αρχικά λοιπόν γίνεται εκτέλεση του κώδικα δηλώνοντας αυστηρά τη μη βελτιστοποίηση του μέσω του `-cl-opt-disable` flag και έπειτα, εκτέλεση του ίδιου προγράμματος χωρίς να ορίζουμε κανένα compiler flag.

Θα πρέπει εδώ να αναφερθεί οτι δεν υπάρχει σαφής αντιστοιχία μεταξύ των δύο αυτών συνθηκών μεταγλώττισης του OpenCL compiler με αυτές της μεταγλώττισης που εξετάστηκαν νωρίτερα για τον GNU C compiler. Αυτό διότι (στην περίπτωση των unoptimized προγραμμάτων) στη μια περίπτωση ζητούμε ρητά τη μη βελτιστοποίηση, ακόμα και αν αυτή γινόταν ανέξοδα ως προς την ακρίβεια (εφόσον για παράδειγμα το επιτρέπει η αρχιτεκτονική), ενώ στη δεύτερη είναι η *default* επιλογή. Αντίστοιχα, στα optimized προγράμματα, από τον GNU C compiler ζητούμε την επιτάχυνσή της εκτέλεσης γνωρίζοντας ότι υπάρχει κόστος ως προς την ακρίβεια (κάτι που περιγράφεται αναλυτικά και στα manual pages του compiler, και στις περισσότερες περιπτώσεις συστήνεται να αποφεύγεται) ενώ από τον OpenCL compiler ζητούμε απλά τη μεταγλώττιση του

προγράμματος, χωρίς κόστος στην αναμενόμενη ακρίβεια. Το τι optimizations ορίζει ο κάθε compiler είναι θέμα που δεν εξετάζεται στην παρούσα περίπτωση. Ενδεικτικά αναφέρεται ότι το -o3 flag του GCC είναι συντομογραφία για πάρα πολλά optimizations (το σύνολο όλων όσων προσφέρει ο compiler) και είναι μάλλον άδικο να συγκριθεί με τη default ρύθμιση του OpenCL compiler της AMD.

Τα δεδομένα εκτέλεσης είναι τα ίδια με αυτά του σειριακού προγράμματος όπως παραπάνω (hardware, λειτουργικό σύστημα, αριθμός κόμβων κλπ). Οι χρόνοι εκτέλεσης φαίνονται στον αντίστοιχο πίνακα.




*Εικόνα 7: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε παράλληλη υλοποίηση (OpenCL, -cl-opt-disable flag) [Intel Atom N2800]*

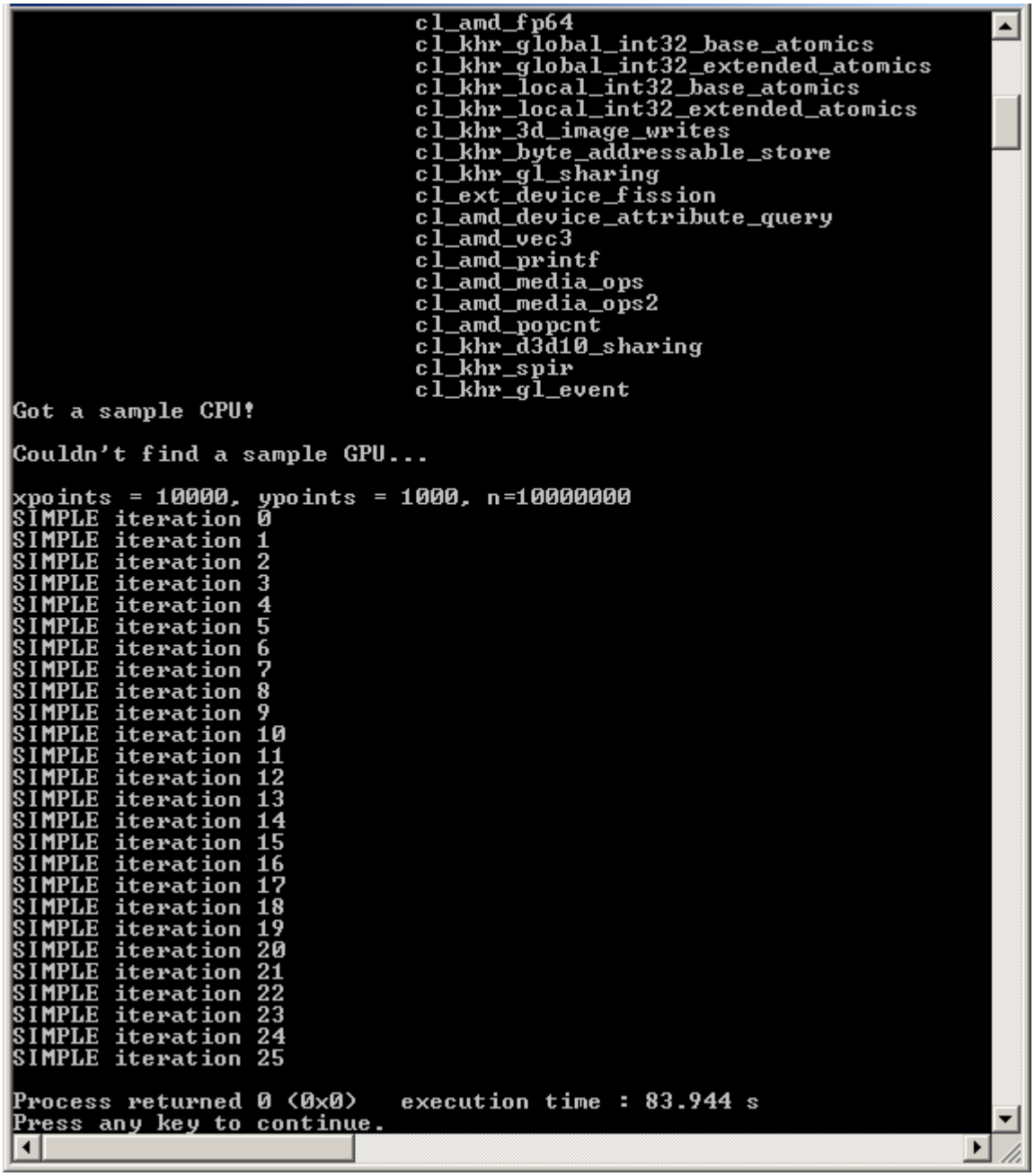
Σε αυτό το σημείο θα πρέπει να τονιστεί ότι η παράλληλη υλοποίηση περιλαμβάνει εκτός του ορισμού των βασικών δομών δεδομένων που περιγράφηκαν παραπάνω, και τις συναρτήσεις μεταφοράς των αποτελεσμάτων από την OpenCL συσκευή προς την κεντρική μνήμη του συστήματος (στην περίπτωσή μας η OpenCL συσκευή είναι ο κεντρικός επεξεργαστής και αυτό ίσως εκμηδενίζει το κόστος σε χρόνο των διαδικασιών αυτών, αλλά ενδέχεται να έχει σημασία σε άλλες υλοποιήσεις).

Εύκολα φαίνεται η δραματική μείωση του χρόνου υπολογισμού των συντελεστών από τη *βέλτιστη υλοποίηση του σειριακού προγράμματος που μπορούσαμε να επιτύχουμε* κατά **71%** και αυτό χρησιμοποιώντας την ίδια ακριβώς συσκευή! Τα αποτελέσματα είναι προκαταρκτικά και επιδέχονται πληθώρα περαιτέρω βελτιώσεων, όπως ενδεικτικά οι παρακάτω:

- Βελτιώσεις σε επίπεδο κώδικα για να αποτυπώνεται καλύτερα ο αλγόριθμος στην εκάστοτε συσκευή που χρησιμοποιούμε (πχ Nvidia OpenCL best practices για Nvidia GPUs).
- Βελτιώσεις/optimizations που προσφέρει η OpenCL (ενδεικτικά multiply and add operations, `cl-relaxed-math`). Αυτές πραγματοποιούνται από τον compiler περνώντας flags κατά τρόπο αντίστοιχο με τα optimizations του GNU C Compiler.
- Χρήση vectors στον κώδικα επίλυσης (OpenCL kernel) που ενδέχεται να βελτιώσει ακόμη περισσότερο τους χρόνους επίλυσης όπως θα δούμε και παρακάτω.
- Χρήση GPUs για το αμιγώς παράλληλο φορτίο, και συγχρονισμός με την κεντρική CPU εκτελούσα το σειριακό φορτίο με το βέλτιστο δυνατό τρόπο με χρήση `cl_event` structures.
- Εκμετάλλευση task-parallelization που προσφέρουν ορισμένες καινούριες συσκευές.

*Πίνακας 3: Εκτέλεση προγράμματος υπολογισμού συντελεστών σε παράλληλη υλοποίηση (OpenCL) [Intel Atom N2800]*

| *n=10,000,000. 25 Iterations* | *Παράλληλο (unoptimized,* -cl-opt-disable*) (s)* | *Παράλληλο (no compiler flags) (s)* |
|---|---|---|
| Run #1 | 223.860 | 84.552 |
| Run #2 | 226.392 | 83.819 |
| Run #3 | 223.424 | 83.491 |
| Run #4 | 225.701 | 83.944 |
| Μέσος όρος χρόνων | *224.844* | *83.952* |




*Εικόνα 8: Ενδεικτική εκτέλεση 25 επαναλήψεων υπολογισμού συντελεστών για πλέγμα 10,000,000 κόμβων σε παράλληλη υλοποίηση (OpenCL, no compiler flags) [Intel Atom N2800]*

## 7.1 Επέκταση συμπερασμάτων παραλληλοποίησης για περισσότερες συσκευές

Έχοντας προκαταρκτικά διαπιστώσει τα οφέλη της παραλληλοποίησης σε μια τυχαία αρχιτεκτονική (τον Η/Υ συγγραφής της παρούσας εργασίας, με επεξεργαστή Intel Atom N2800), και έχοντας διαπιστώσει την επιρροή των compiler flags καθώς και των διαφορετικών γλωσσών προγραμματισμού στο χρόνο εκτέλεσης του ίδιου προγράμματος, θα προχωρήσουμε σε περισσότερες μετρήσεις και συγκρίσεις παραλληλοποίησης για διαφορετικούς επεξεργαστές/αρχιτεκτονικές. Οι συσκευές που εξετάστηκαν στην παρούσα ενότητα είναι:

- Intel Core i3-2100 (Sandy Bridge) @3.10GHz: 2 cores, 4 threads, Λιθογραφία 32nm, TDP: 65w (98)
- Intel Core 2 Quad Q9550 (Yorkfield) @2.83GHz: 4 cores, 4 threads, Λιθογραφία 45nm, TDP: 95w (98)
- Gigabyte AMD Radeon HD7950 (Tahiti Pro) @1,000MHz (core clock): 28 Compute units, 1,792 Processing elements, Λιθογραφία 28nm, TDP: 200w (98)

Στην τρίτη συσκευή της λίστας έχουμε μόνο παράλληλη υλοποίηση. Τα προγράμματα είναι ίδια με αυτά που εκτελέστηκαν και στην προηγούμενη ενότητα. Έλεγχος των αποτελεσμάτων μεταξύ των διαφορετικών προγραμμάτων, αρχιτεκτονικών και compilers έγινε τυπώνοντας τυχαίους υπολογισθέντες συντελεστές και συγκρίνοντας τους αριθμούς. Αποκλίσεις υπήρξαν, ήταν όμως ικανοποιητικά και αναμενόμενα μικρές. Για παράδειγμα:

Για υλοποίηση του αλγορίθμου με αρχικό πλέγμα

- `u[i] = 456`
- `v[i] = 123`
- `p[i] = 789`

Οι τυχαίοι συντελεστές `APu[50,000], APu[60,000], APu[70,000], APu[80,000], APu[90,000]` υπολογίζονταν από το πρόγραμμα σε GNU C ως:

`579.018005`,

ενώ από το πρόγραμμα που παρήγαγε ο OpenCL compiler ως:

`579.017944`

Το πρότυπο OpenCL επιβάλλει σε όλες τις συσκευές που το υποστηρίζουν να παρέχουν τη μεταβλητή `float`, που είναι η αναπαράσταση πραγματικού αριθμού στην μνήμη του υπολογιστή. Απαιτεί επίσης από τις συσκευές να ακολουθούν πολλές από τις διατάξεις του προτύπου *IEEE-754* (98) αλλά όχι όλες από αυτές. Για εφαρμογές που η μεγάλη ακρίβεια αποτελεί κρίσιμη απαίτηση είναι απαραίτητο να γνωρίζουμε εκ των προτέρων τις δυνατότητες ακρίβειας των συσκευών μας, να επιβάλλουμε τη χρήση των κατάλληλων μεταβλητών και να ελέγχουμε τα optimizations που εφαρμόζονται. Παρακάτω παρατίθεται ο πίνακας των αποτελεσμάτων:

*Πίνακας 4: Σειριακή (σε C) και παράλληλη (σε OpenCL) εκτέλεση προγράμματος υπολογισμού συντελεστών για n=10,000,000 και 25 επαναλήψεις του αλγορίθμου SIMPLE για διάφορες συσκευές*

| Συσκευή | Σειριακή εκτέλεση (C, GNU C Compiler, no compiler flags) | Σειριακή εκτέλεση (C, GNU C Compiler, -o3 flag) | Παράλληλη εκτέλεση (OpenCL, no compiler flags) | Παράλληλη εκτέλεση (OpenCL, -cl-opt-disable flag) |
|---|---|---|---|---|
| Intel Core i3-2100 (Sandy Bridge) | | | | |
| Run #1 | 55.264s | 40.512s | 24.964s | 49.791s |
| Run #2 | 55.957s | 40.255s | 25.790s | 47.741s |
| Run #3 | 53.757s | 40.141s | 23.752s | 46.641s |
| Run #4 | 56.955s | 40.739s | 23.941s | 45.964s |
| *Μέσος όρος* | *55.483s* | *40.412s* | *24.612s* | *47.534s* |
| Intel Core 2 Quad Q9550 (Yorkfield) | | | | |
| Run #1 | 117.287s | 96.530s | 24.865s | 45.434s |
| Run #2 | 116.983s | 97.087s | 25.065s | 45.848s |
| Run #3 | 116.960s | 97.239s | 24.681s | 44.449s |
| Run #4 | 117.269s | 97.160s | 24.979s | 44.017s |
| *Μέσος όρος* | *117.125s* | *97.004s* | *24.898s* | *44.937s* |
| Gigabyte AMD Radeon HD7950 (Tahiti Pro) | | | | |
| Run #1 | - | - | 5.285s | 5.343s |
| Run #2 | - | - | 5.524s | 5.453s |
| Run #3 | - | - | 5.490s | 5.695s |
| Run #4 | - | - | 5.321s | 5.712s |
| *Μέσος όρος* | - | - | *5.403s* | *5.551s* |

Όπως βλέπουμε η παραλληλοποίηση παρέχει απτή μείωση του χρόνου εκτέλεσης του ίδιου προγράμματος, σε πολυπύρηνους κεντρικούς επεξεργαστές σε σχέση με τη σειριακή εκτέλεση στην ίδια ακριβώς συσκευή. Ακόμα πιο δραστική είναι η μείωση με χρήση desktop GPU, όπως φαίνεται και παραπάνω. Πιθανότατα η ταχύτητα ολοκλήρωσης του προγράμματος με χρήση GPU να περιορίζεται κατά το μεγαλύτερο ποσοστό από τη μεταφορά δεδομένων από και προς τη συσκευή. Με περαιτέρω μελέτη είναι δυνατόν να βελτιστοποιήσουμε και το τμήμα αυτό (της μεταφοράς δεδομένων) εκμεταλλευόμενοι την καλύτερη για τη συγκεκριμένη αρχιτεκτονική συνάρτηση μεταφοράς και δημιουργίας *buffer* μέσα στο πρόγραμμα.

Διαπιστώνουμε ακόμα ότι το `-cl-opt-disable` flag δεν επηρεάζει ουσιαστικά την εκτέλεση του προγράμματος στην περίπτωση της εκτέλεσης σε GPU. Σε προσπάθεια παραπάνω επιτάχυνσης

της εκτέλεσης, έγινε ένα προσωρινό *over-clock* του Tahiti Pro GPU από 1,000MHz σε 1,200MHz, και έγιναν εκ νέου οι 4 δοκιμές. Τα αποτελέσματα παρουσιάζονται παρακάτω:

*Πίνακας 5: Παράλληλη (σε OpenCL) εκτέλεση προγράμματος υπολογισμού συντελεστών για n=10,000,000 και 25 επαναλήψεις του αλγορίθμου SIMPLE για την κάρτα Gigabyte AMD Radeon HD7950 overclocked στα 1200MHz (core clock)*

| *n=10,000,000. 25 Iterations* | Gigabyte AMD Radeon HD7950 (Tahiti Pro) O/C 1,200MHz |
|---|---|
| Run #1 | 5.182s |
| Run #2 | 5.387s |
| Run #3 | 5.258s |
| Run #4 | 5.605s |
| *Μέσος όρος* | *5.358s* |

Διαπιστώνουμε ότι δεν υπάρχει ουσιαστική επίδραση στο χρόνο εκτέλεσης, κάτι που ίσως επιβεβαιώνει και την υπόθεση ότι η μεταφορά των δεδομένων είναι αυτή που περιορίζει τελικά το πρόγραμμα.

## 7.2 Εκτέλεση του προγράμματος σε αρχιτεκτονικές φορητών συσκευών

Όπως αναφέρθηκε και στις εισαγωγικές σημειώσεις, ένα από τα προνόμια του προγραμματισμού σε OpenCL έναντι άλλων κλειστών προτύπων (πχ Nvidia CUDA) είναι η λεγόμενη *φορητότητα (portability).* Ο κώδικας συντάσσεται μια φορά και μπορεί να εκτελεστεί σε μια πληθώρα συσκευών/αρχιτεκτονικών και ο OpenCL compiler θα παράξει κάθε φορά machine code για την αντίστοιχη αρχιτεκτονική.

Σε αυτό το σημείο θα γίνει μια επίδειξη των δυνατοτήτων των (όχι και τόσο) σύγχρονων συσκευών κινητών τηλεφώνων. Αρχικά επιλέχθηκε η συσκευή Samsung Galaxy S7 που ήταν η *flagship* συσκευή της Samsung το έτος 2016 οπότε και κυκλοφόρησε. Η συσκευή διατίθεται σε δύο παραλλαγές ως προς το υλικό της (Qualcomm Snapdragon & Samsung Exynos chipsets). Στη συγκεκριμένη συσκευή έχουμε το *Exynos 8890 (Octa)* chipset το οποίο ενσωματώνει το *Mali-T880 MP12* GPU (98). Στη συγκεκριμένη υλοποίησή του, πρόκειται για GPU με 12 OpenCL Compute Units, κάθε ένα από τα οποία περιέχει πολλαπλά processing elements, χρονισμένο στα 650-1000MHz και υλοποιεί το πρότυπο OpenCL 1.2 και μάλιστα FULL PROFILE (98). Σύμφωνα με την ίδια πηγή διαθέτει επεξεργαστική ισχύ από 265.2 έως 408 GFLOPS (το άνω άκρο της οποίας είναι μάλλον υπερβολή. Κατά πάσα πιθανότητα ισχύει το ~250 GFLOPS. Το πρόγραμμα OpenCL-Z δίνει επεξεργαστική ισχύ για *single precision float arithmetic*: 21.28 GFLOPS). Η συσκευή χρησιμοποιεί custom Android, έκδοσης 7.0 (έκδοση kernel: 3.18.14-00063) που προσφέρει στο χρήστη δικαιώματα superuser. Σύμφωνα με τα παραπάνω, περιμένουμε μεγαλύτερες επιδόσεις από τους desktop επεξεργαστές που εξετάσαμε έως τώρα.

Η χρήση του OpenCL σε Android μπορεί κατά περίπτωση να είναι προβληματική. Η Google που είναι ο κύριος developer του AOSP (Android Open Source Project) αντιτίθεται στη χρήση του OpenCL και προωθεί το Renderscript API, αλλά μπορεί να υλοποιείται στο υλικό από τον κατασκευαστή της κάθε συσκευής. Δεν είναι πάντα σαφές ποιες συσκευές το υποστηρίζουν και ποιες όχι, αλλά φαίνεται ότι όλες οι σύγχρονες και επεξεργαστικά ισχυρές συσκευές το υποστηρίζουν. Μια μέθοδος για γρήγορη επισκόπηση της υποστήριξης είναι να εγκαταστήσουμε το OpenCL-Z διαθέσιμο δωρεάν από το Play Store (Google Play) της Google. Σύμφωνα με αυτό, η εν λόγω συσκευή υποστηρίζεται και παρουσιάζει ως OpenCL device μόνο το GPU και όχι το CPU του chipset. Το GPU διαθέτει όπως αναφέρθηκε παραπάνω 12 OpenCL Compute Units ενώ δεν υπάρχει standard συνάρτηση που να επιστρέφει τον αριθμό των Processing Elements ανά Compute Unit, περισσότερο γιατί αυτό έχει παντελώς διαφορετική έννοια ανά κατασκευαστή και αρχιτεκτονική.

Επόμενο βήμα είναι να εγκαταστήσουμε τον GNU C compiler για Android. Για τη συγκεκριμένη εργασία εγκαταστάθηκε το CPP Droid Integrated Development Environment (IDE) που είναι διαθέσιμο δωρεάν από το Play store. Σαν λογισμικό δεν είναι απαλλαγμένο από bugs αλλά προσφέρει το πλήρες πακέτο IDE & compiler & libraries, απαλλάσσοντας το χρήστη από περιττό πονοκέφαλο, καθότι και η υποστήριξη των τελευταίων εκδόσεων του GNU C compiler στο Android φαίνεται να μην υποστηρίζεται. Η έκδοση του GNU C compiler που χρησιμοποιεί το IDE είναι v4.8.

Αφού ενσωματώσουμε τα δύο απαραίτητα *header files* `cl.h` & `cl_platform.h` στο πρόγραμμά μας (δεν υπάρχει σαφές πεδίο στο IDE για χρήση header files), θα πρέπει να συνδέσουμε κατόπιν της μεταγλώττισης τα ακόλουθα Shared Objects (σε περιβάλλοντα Linux): `libOpenCL.so`, `liblog.so`, `libc++.so`. Επειδή οι φάκελοι που περιέχουν τα συγκεκριμένα αρχεία φαίνεται να μην είναι στα search directories by default (παρότι θα έπρεπε), προσθέτουμε τα εξής 2 directories μέσω linker flags: /system/vendor/lib & /system/lib

Τελικά τα flags στον compiler είναι τα ακόλουθα:

```
["-lOpenCL","-llog","-lc++","-L/system/vendor/lib","-L/system/lib"]
```

Επειδή η εγκατεστημένη πλατφόρμα υποστηρίζει την έκδοση v1.2 του OpenCL, συναρτήσεις που τυχόν χρησιμοποιούμε στο αρχικό μας πρόγραμμα και ορίζονται σε μεταγενέστερα πρότυπα πρέπει να τροποποιηθούν (παράδειγμα αυτού η συνάρτηση `clCreateCommandQueueWithProperties()`)

Το IDE σε αντίθεση με αυτό σε Windows που είχαμε χρησιμοποιήσει νωρίτερα δεν μετρά το χρόνο βάζοντας σημεία στην αρχή και στο τέλος του προγράμματος (κάτι το οποίο δεν είναι απαραίτητο). Επομένως χρειάστηκε να τοποθετηθούν σημεία μέτρησης με χρήση της συνάρτησης `gettimeofday()` όπως είχε γίνει και σε προγενέστερη ενότητα για μέτρηση των επιμέρους διεργασιών του σειριακού προγράμματος. Για αισθητικούς λόγους, το αποτέλεσμα σε second επιλέχθηκε να χρωματίζεται σε πράσινο, αναδεικνύοντας την εξαιρετική AMOLED οθόνη του Galaxy S7 :).

Αφότου κάνουμε τις παραπάνω ενέργειες, είμαστε έτοιμοι να δοκιμάσουμε την εκτέλεση του ίδιου προγράμματος στη συγκεκριμένη συσκευή. Τα αποτελέσματα δικαίωσαν τον κόπο που καταβλήθηκε και φαίνονται στον παρακάτω πίνακα:

*Πίνακας 6: Παράλληλη (σε OpenCL) εκτέλεση προγράμματος υπολογισμού συντελεστών για n=10,000,000 και 25 επαναλήψεις του αλγορίθμου SIMPLE για τη συσκευή Samsung Galaxy S7*

| n=10,000,000. 25 Iterations | Galaxy S7 (Mali T880 MP12) (no compiler flags) | Galaxy S7 (Mali T880 MP12) (-cl-opt-disable) |
|---|---|---|
| Run #1 | 11.943 | 9.463 |
| Run #2 | 10.444 | 9.392 |
| Run #3 | 9.378 | 9.365 |
| Run #4 | 9.522 | 9.536 |
| *Μέσος όρος χρόνων* | *10.322* | *9.439* |

Όπως διαπιστώνουμε και από τους παραπάνω αριθμούς, ο μικρός επεξεργαστής απόδοσης γραφικών εντός του Samsung Galaxy S7 είναι ικανότερος για παράλληλα φορτία από έναν από τους κορυφαίους desktop επεξεργαστές της Intel πριν από μια δεκαετία (Core 2 Quad Q9550)! Βλέπουμε επίσης ότι όπως και στην περίπτωση της αυτόνομης κάρτας γραφικών AMD Radeon HD 7950 το flag `-cl-opt-disable` πρακτικά δεν επηρεάζει καθόλου το χρόνο εκτέλεσης του προγράμματος. Ενδεχομένως για επεξεργαστές τέτοιου είδους ο OpenCL compiler να μην είναι σε θέση να εισάγει πολλά optimizations by default.

Βάσει τιμολογίων της 08/11/2008 η τιμή αγοράς για την *TRAY* έκδοση του Intel Core 2 Quad Q9550 από γνωστή αλυσίδα εμπορίου hardware υπολογιστών στην Ελλάδα ήταν 300.68€ μετά φόρων. Επίσης βάσει τιμολογίου, η τιμή αγοράς της συσκευής Samsung Galaxy S7 την 30/12/2017 ήταν 251.08€ από γνωστή αλυσίδα, σε μια προσφορά που θεωρήθηκε εξαιρετική ευκαιρία. Αντίστοιχα, η τιμή αγοράς της Gigabyte AMD Radeon HD 7950 ήταν 325.40€ την 31/07/2013. Παρότι οι αγορές έχουν πραγματοποιηθεί σε δραστικά μεγάλο χρονικό εύρος (δεκαετίας) παρουσιάζονται στον παρακάτω πίνακα ορισμένα ενδιαφέροντα στοιχεία. Ο χρόνος ολοκλήρωσης του προγράμματος αφορά στην περίπτωση του Q9550 στη σειριακή εκτέλεση (όπως θα κάναμε την περίοδο του 2008), ενώ για τις άλλες συσκευές αφορά την υλοποίηση σε OpenCL (καθώς δεν έχουμε και άλλο, εύκολα προσβάσιμο τρόπο να χρησιμοποιήσουμε τις συσκευές αυτές για γενικούς υπολογισμούς). Ποσοστιαία βελτίωση ως προς σειριακή εκτέλεση είναι το ποσοστό βελτίωσης του χρόνου εκτέλεσης σε σχέση με τη σειριακή εκτέλεση στον Q9550 χωρίς το -o3 flag του GNU C Compiler. Οι εκτελέσεις σε OpenCL θεωρούνται ως εκτελούμενες με το `-cl-opt-disable` flag το οποίο για τις συγκεκριμένες δύο δεν είχε ουσιαστική επίδραση. Η μέτρηση της κατανάλωσης του Galaxy S7 έγινε με μετρητή κατανάλωσης σε οικιακή πρίζα, καθώς η συσκευή εκτελούσε τους βρόχους επανάληψης SIMPLE επ’ άπειρον, ούσα συνδεδεμένη στο ρεύμα και έχοντας τη φόρτιση στο 100% (η μέτρηση είναι αρκετά “συντηρητική”, εμφανίζοντας την κατανάλωση αρκετά υψηλή). Δεν γίνεται αναγωγή της χρηματικής αξίας (λόγω πληθωρισμού) των συσκευών σε σημερινές, καθότι αυτό θα ήταν παντελώς ανούσιο όταν το ίδιο το υλικό απαξιώνεται με τόσο γρήγορους ρυθμούς.

Οι τιμές που παρατίθενται είναι πιο πολύ ενδεικτικές ώστε να σχηματίσουμε μια γνώμη για το ποια συσκευή είναι η καταλληλότερη, αναλόγως του ποιο χαρακτηριστικό θέλουμε να βελτιστοποιήσουμε. Όπως είναι αναμενόμενο, επεξεργαστές τύπου GPU χαμηλής ισχύος είναι

κατάλληλοι για μεγιστοποίηση του “*Performance per watt”*, ενώ επεξεργαστές τύπου GPU μεγάλης ισχύος για desktop υπολογιστές είναι καταλληλότεροι για μεγιστοποίηση του “*Performance per dollar”*. Για ορισμένες συσκευές και συνδυασμούς αγοράς (πχ αγορά μεταχειρισμένων, αγορά από το εξωτερικό, αγορές σε μεγάλη κλίμακα) ενδέχεται να πετυχαίνουμε έναν εξαιρετικό συμβιβασμό μεταξύ των δύο, πετυχαίνοντας και τα δύο χαρακτηριστικά τάξεις μεγέθους βελτιωμένα σε σχέση με κλασσικούς επεξεργαστές CPU ακόμη και σημερινούς (εξάλλου οι χρησιμοποιηθείσες συσκευές δεν είναι σε καμία περίπτωση στην *αιχμή της τεχνολογίας*). Είναι χαρακτηριστικό ότι σήμερα υπάρχουν στο εμπόριο GPUs με μνήμη έως 32GB, και αποτελούν συσκευές κατάλληλες για επίλυση ενός τεράστιου εύρους παραλληλοποιήσιμων προβλημάτων.

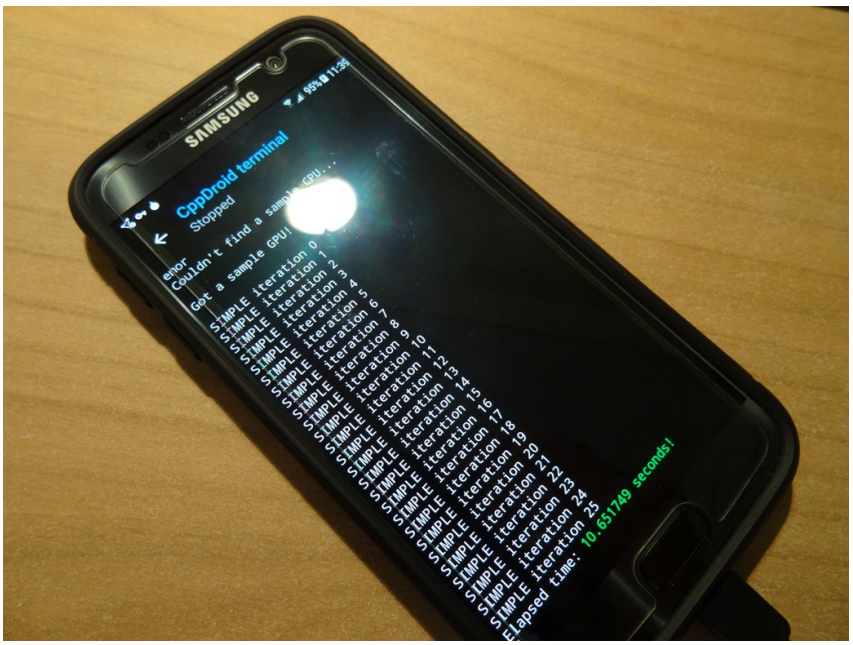
*Πίνακας 7: Ενδεικτικός πίνακας σύγκρισης χαρακτηριστικών για διαφορετικού τύπου συσκευές*

| Συσκευή | Ημ/νία αγοράς | Κόστος αγοράς (μετά φόρων σε €) | Ποσοστιαία βελτίωση ως προς σειριακή εκτέλεση | Κατανάλωση (W) | *Performance per watt* (αναγωγή στον επεξεργαστή Q9550) | *Performance per dollar* (αναγωγή στον επεξεργαστή Q9550) |
|---|---|---|---|---|---|---|
| Intel Core 2 Quad Q9550 | 2008/11/08 | 300.68 | 100% | 95*(98) | 1 | 1 |
| Samsung Galaxy S7 (Mali T880 MP12) | 2017/12/30 | 251.08 | 1240.86% | 7.3 (Κατά τη χρήση)** | 161.48 | 14.86 |
| AMD Radeon HD 7950 | 2013/07/31 | 325.40 | 2110.08% | ~180*** | 11.13 | 19.50 |

*Μέγιστη θερμική ισχύς (TDP) βάσει προδιαγραφών.

**Ενδεικτική κατανάλωση της συσκευής συνολικά μαζί με τις απώλειες του φορτιστή & καλωδίου. Ενδέχεται να είναι ανακριβής καθώς παράγοντες όπως φωτεινότητα οθόνης, χρήση δεδομένων δικτύου κλπ, δεν φάνηκε να την επηρεάζουν. Κατά πάσα πιθανότητα η πραγματική κατανάλωση είναι πολύ μικρότερη.

***Κατανάλωση κατά προσέγγιση για το reference μοντέλο AMD Radeon HD7950. Εξαρτάται από το είδος του φορτίου, το χρονισμό, την υλοποίηση του κατασκευαστή κλπ.




*Εικόνα 9: Εκτέλεση του προγράμματος υπολογισμού συντελεστών παράλληλα στο Samsung Galaxy S7 (μέτρηση χρόνου με* `gettimeofday()`)

*Εικόνα 10: Διάταξη για τη μέτρηση κατανάλωσης του Samsung Galaxy S7 κατά τη χρήση*

*Εικόνα 11: Προσπάθεια μέτρησης κατανάλωσης του Samsung Galaxy S7*

Για λόγους σύγκρισης, παρατίθεται παρακάτω ο χρόνος εκτέλεσης του ίδιου σειριακού προγράμματος σε C στον επεξεργαστή CPU του Samsung Galaxy S7. Οι χρόνοι μετρήθηκαν με τη μέθοδο της συνάρτησης `gettimeofday()` και αποτυπώνονται στην πρώτη στήλη του παρακάτω πίνακα. Βλέπουμε ότι οι χρόνοι είναι οι μεγαλύτεροι από όλες τις συσκευές που εξετάσαμε έως τώρα.

Αυτό που θα πρέπει βέβαια να προσέξουμε σε αυτήν την περίπτωση αλλά και σε όλες τις περιπτώσεις εκτέλεσης σε συσκευές με έμφαση στην αυτονομία και φορητότητα, είναι επέμβαση του power manager του λειτουργικού συστήματος για τη μετάβαση του επεξεργαστή στα states της χαμηλότερης κατανάλωσής του (πχ συνήθως με τη μετάβαση σε κατάσταση αδράνειας ή το κλείδωμα της συσκευής, οπότε και “σβήνει” η οθόνη, αλλά και σε όλη τη διάρκεια χρήσης της συσκευής). Ο τυπικός χρήστης δεν έχει πρόσβαση σε επιλογές που καθορίζουν τις συχνότητες του επεξεργαστή, καθώς απαιτούνται δικαιώματα superuser. Με τα δικαιώματα αυτά όμως είμαστε σε θέση, με χρήση της κατάλληλης εφαρμογής, να περιορίσουμε την ελάχιστη συχνότητα του επεξεργαστή, καθώς και να αλλάξουμε τη συμπεριφορά του *governor* για το συγκεκριμένο επεξεργαστή.

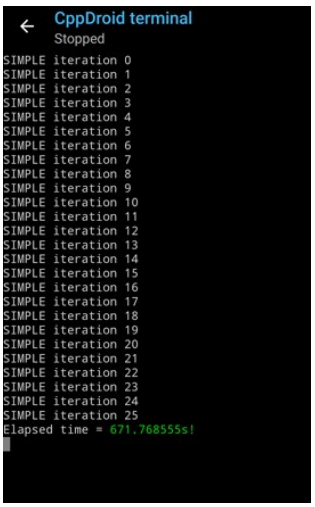
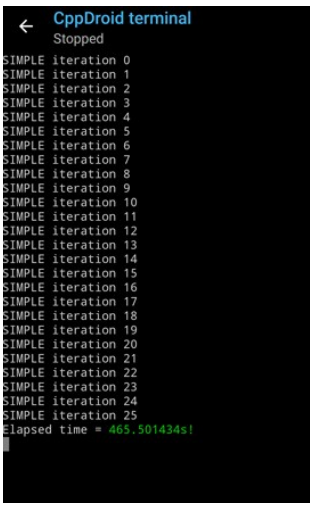
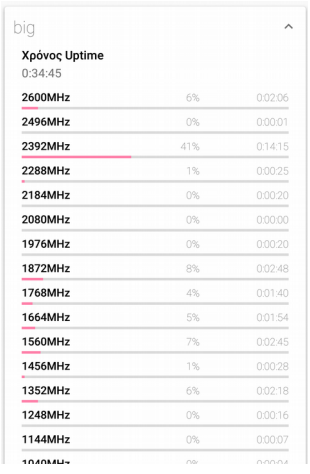
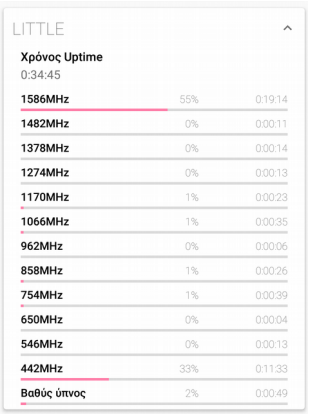
Στον παρακάτω πίνακα γίνεται σύγκριση των αποτελεσμάτων των χρόνων εκτέλεσης μεταξύ των περιπτώσεων των *default* ρυθμίσεων και αυτής του περιορισμού της ελάχιστης συχνότητας των επεξεργαστών. Στην πρώτη στήλη αποτυπώνουν την εκ προεπιλογής μετάβαση του επεξεργαστή σε states χαμηλότερης κατανάλωσης (επηρεαζόμενη από το εάν η συσκευή χρησιμοποιείτο, εάν υπήρχαν ενημερώσεις ή τρέχοντα προγράμματα που χρησιμοποιούσαν τον επεξεργαστή κλπ), κάνοντας τους χρόνους εκτέλεσης να είναι ιδιαίτερα μεγάλοι. Αφετέρου, την σύγκλιση και βελτίωση των χρόνων εκτέλεσης του ίδιου (σειριακού) προγράμματος την περίπτωση που περιορίζουμε την ελάχιστη συχνότητα του επεξεργαστή. Στην συγκεκριμένη περίπτωση, η ελάχιστη

συχνότητα του τετραπύρηνου *“Mongoose”* CPU τέθηκε στα 2392MHz (από 728MHz) ενώ του τετραπύρηνου *“Cortex-A53”* τέθηκε στα 1378MHz (από 442MHz) (δεν μπορούμε να γνωρίσουμε ποιόν ή ποιους επεξεργαστές επιλέγει το λειτουργικό για εκτέλεση του προγράμματος, είναι πολύ πιθανόν όμως η εκτέλεση να γινόταν σε έναν εκ των δύο επεξεργαστών). Οι governors τέθηκαν σε *performance* mode, ενώ απετράπη το κλείδωμα της συσκευής. Παρατηρήθηκε εμφανής αύξηση της θερμοκρασίας της συσκευής.

*Πίνακας 8: Σειριακή εκτέλεση προγράμματος υπολογισμού συντελεστών στο Samsung Galaxy S7 (n=10,000,000, 25 iterations)*

| Σειριακό πρόγραμμα στο Galaxy S7 | Εκτέλεση με default ρυθμίσεις (s) | Εκτέλεση με περιορισμό της ελάχιστης συχνότητας (s) |
|---|---|---|
| Run #1 | 615.903 | 485.462 |
| Run #2 | 687.941 | 451.975 |
| Run #3 | 658.149 | 471.273 |
| Run #4 | 671.769 | 465.501 |
| *Μέσος όρος χρόνων* | *658.441* | *468.553* |

Στον επόμενο πίνακα παρουσιάζονται, εκτός από τις ενδεικτικές εκτελέσεις για τις καταστάσεις που αναφέρθηκαν παραπάνω, τα ποσοστά χρήσης συχνοτήτων για τους δύο επεξεργαστές στην περίπτωση του περιορισμού της ελάχιστης συχνότητάς τους (τέτοια εικόνα είναι ιδιαίτερα ασυνήθιστη για φορητό κινητό τηλέφωνο, διότι θα κατανάλωνε πολύ γρήγορα τη διαθεσιμότητα της μπαταρίας). Συμπερασματικά αναφέρεται ότι ο επεξεργαστής CPU του Galaxy S7 είναι τελικά ικανότερος για το συγκεκριμένο φορτίο από τον Intel N2800 (αρχιτεκτονικής x86) του netbook που εξετάστηκε νωρίτερα.

| | | | |
|---|---|---|---|
|  |  |  |  |
| Ενδεικτική εκτέλεση με default ρυθμίσεις (671.768555s) | Ενδεικτική εκτέλεση με περιορισμό της ελάχιστης συχνότητας (465.501434s) | Ποσοστά χρήσης των διαθέσιμων συχνοτήτων του *“Mongoose”* CPU έχοντας περιορίσει την ελάχιστη συχνότητα | Ποσοστά χρήσης των διαθέσιμων συχνοτήτων του *“Cortex-A53”* CPU έχοντας περιορίσει την ελάχιστη συχνότητα |

Για λόγους πληρότητας παρατίθενται παρακάτω τα αποτελέσματα για την παράλληλη και σειριακή εκτέλεση στην ανταγωνιστική πλατφόρμα (Snapdragon 801) της συσκευής Oneplus X, που είχε κυκλοφορήσει περίπου την ίδια περίοδο (Νοέμβριος 2015) αν και, λόγω τιμής, αφορά σε διαφορετικό *market segment* (98). Η συσκευή ενσωματώνει τετραπύρηνο επεξεργαστή CPU Krait 400 στα 2.30GHz, και τον επεξεργαστή γραφικών Adreno 330. Το λειτουργικό σύστημα είναι near-stock Android έκδοσης 6.0.1 (OxygenOS) Εδώ η βιβλιοθήκη του κατασκευαστή είχε πολλά περισσότερα dependencies και τα flags στον compiler τελικά ήταν τα ακόλουθα:

```
{"linkOptions":["-lOpenCL","-llog","-lc++","-lgsl","-lcutils","-lz","-lutils","-lui","-lCB","-lbacktrace","-lhardware","-lsync","-lbase","-lunwind"]}
```

Τα αποτελέσματα φαίνονται παρακάτω:

*Πίνακας 9: Σειριακή και παράλληλη εκτέλεση προγράμματος υπολογισμού συντελεστών στο Oneplus X (n=10,000,000, 25 iterations)*

| N=10,000,000. 25 Iterations | Oneplus X GPU (Adreno 330) (no compiler flags) | Oneplus X CPU (Krait 400) |
|---|---|---|
| Run #1 | 55.757 | 680.429 |
| Run #2 | 55.699 | 678.476 |
| Run #3 | 55.674 | 677.912 |
| Run #4 | 55.581 | 681.085 |
| *Μέσος όρος χρόνων* | *55.678* | *679.476* |

Όπως βλέπουμε ο επεξεργαστής γραφικών της συσκευής αυτής είναι αρκετά υποδεέστερος για το συγκεκριμένο φορτίο απ’ ότι ο αντίστοιχος του Samsung Galaxy S7. Το αξιοσημείωτο σε αυτήν την περίπτωση είναι ότι το πρόγραμμα OpenCL-Z δίνει επεξεργαστική ισχύ για *single precision float arithmetic* 42-45 GFLOPS που είναι μεγαλύτερη από αυτήν του Samsung Galaxy S7 (δεν γνωρίζουμε τον τρόπο που το πρόγραμμα παίρνει αυτές τις μετρήσεις). Επίσης βλέπουμε ότι ο power manager του συγκεκριμένου λειτουργικού συστήματος δεν επενέβη με σημαντικό τρόπο στο χρονισμό των πυρήνων του επεξεργαστή, δεδομένου ότι οι αριθμοί είναι πολύ κοντινοί μεταξύ τους.

## 8.1 OpenCL Vectors

Η έννοια του διανύσματος στην OpenCL έχει (όπως αναφέρθηκε και νωρίτερα) την έννοια της διατεταγμένης ν-άδας μεταβλητών ίδιου τύπου. Υπό την προϋπόθεση ότι το υλικό το υποστηρίζει, μαθηματικές ενέργειες σε διανύσματα είναι δυνατόν να εκτελούνται πολύ πιο γρήγορα απ’ ότι το

άθροισμα των ίδιων ενεργειών σε κάθε μια από τις ανεξάρτητες μεταβλητές τους. Το πρότυπο παρέχει τυποποιημένες συναρτήσεις για να *"ρωτήσουμε"* το υλικό κατά την εκτέλεση (*runtime*) ποιο είναι το προτιμώμενο μήκος διανύσματος (αυτό για το οποίο υπάρχει η βέλτιστη υποστήριξη από το υλικό) για κάθε τύπου μεταβλητή. Οι τιμές αυτές τυπώνονται από το παράλληλο πρόγραμμα σε κάθε εκτέλεση ως *"preferred vector width"*.

Με τη χρήση διανυσμάτων, συγκεκριμένος αριθμός σημείων του χώρου εργασίας ομαδοποιείται και συμπεριφέρεται σαν μια μεταβλητή (διατηρούμε την πρόσβαση στις πρωτόγονες μεταβλητές μεσώ πρόσβασης στα *στοιχεία* του διανύσματος, *vector elements*). Επειδή όμως στον αλγόριθμο που αναπτύξαμε χρησιμοποιούνται για κάθε σημείο, μεταξύ άλλων, τιμές μεταβλητών γειτονικών κόμβων (διαμήκης θέση X, εγκάρσια θέση Y & ταχύτητες u, v γειτονικών κόμβων) δηλαδή δεν μπορούμε να έχουμε χονδροειδέστερο ("*coarser*") πλέγμα για αυτούς τους πίνακες από αυτό που παριστάνει τους κόμβους ως αυτόνομους, η εφαρμογή και χρήση διανυσμάτων μπορεί να γίνει μόνο για τον υπολογισμό των συντελεστών. Αυτό βέβαια επιβάλλει τη χρήση ταυτόχρονα διανυσμάτων και των αντίστοιχων αυτόνομων μεταβλητών στο ίδιο kernel κάτι που περιπλέκει αρκετά τη δομή του προγράμματος και θα χρειαζόταν εκ νέου συγγραφή και εντοπισμό σφαλμάτων. Εξάλλου στο πρόγραμμα που έχει κατασκευαστεί, αφήνουμε εκτός παράλληλων υπολογισμών τους ακραίους κόμβους, κάτι το οποίο ενδέχεται να δυσκολέψει ακόμη παραπάνω τη χρήση διανυσμάτων. Μια λύση για το πρόβλημα αυτό είναι ο υπολογισμός των συντελεστών για όλους τους κόμβους του πλέγματος (συμπεριλαμβανομένων των ακραίων), παράγοντας "*garbage values*" για τους ακραίους, και ο ορισμός των τιμών για τους ακραίους εκ νέου (βάσει οριακών συνθηκών, σειριακά (κάτι που γίνεται ούτως ή άλλως στην παρούσα υλοποίηση). Επίσης η πρόσβαση στα κατάλληλα στοιχεία των διανυσμάτων, επειδή ορισμένες φορές αυτά ανήκουν στο ίδιο διάνυσμα με αυτό που χειρίζεται ο kernel μέσω του *global ID* του, και άλλες σε γειτονικό διάνυσμα, μπορεί να γίνει ανεξάρτητα του παραπάνω περιορισμού μέσω χρήσης συναρτήσεων που αφορούν *memory operations*. Σε αυτήν την περίπτωση θα πρέπει απαραιτήτως να γνωρίζουμε το *byte ordering* της συσκευής που χρησιμοποιούμε (*endianness*), πληροφορία την οποία μας παρέχουν εγγενείς συναρτήσεις της OpenCL.

Καταλήγουμε επομένως ότι ναι μεν η εφαρμογή διανυσμάτων είναι δυνατή, η χρήση τους όμως απαιτεί εκ νέου συγγραφή του kernel και κυρίως εκ νέου εντοπισμό σφαλμάτων. Στο παρόν κεφάλαιο κρίνεται σκόπιμο να γίνει μια σύγκριση της επιτάχυνσης που μπορούμε να πετύχουμε μέσω διανυσμάτων, με εφαρμογή σε πρόβλημα εκτός αυτού που αναπτύχθηκε έως τώρα ώστε να εξασφαλίζεται η ανεξαρτησία των αυτόνομων μεταβλητών ως προς την πρόσβαση σε γειτονικές τους.

Έτσι ξεφεύγουμε από τη μέχρι τώρα ανάλυση αναπτύσσοντας ένα θεωρητικό (τυχαίο) πρόβλημα και καταγράφοντας την ενδεχόμενη βελτίωση σε σχέση με την ήδη παράλληλη επεξεργασία. Μιας και το πρόβλημα αυτό κατασκευάζεται ακριβώς για αυτό το σκοπό, επιλέχθηκε να χρησιμοποιηθούν ορισμένες πιο "εξωτικές" συναρτήσεις όπως η συνάρτηση σφάλματος Gauss και ο φυσικός λογάριθμος της συνάρτησης Γάμμα, προσθέτοντας επιπλέον υπολογιστικό κόστος και κάνοντας επίδειξη των εσωτερικών συναρτήσεων των γλωσσών C & OpenCL C.

Συνάρτηση σφάλματος:

$$erf(x)=\frac{2}{\sqrt{\pi}}\cdot\int_0^x \mathrm{e}^{-t^2}\,dt$$

Συνάρτηση “Log Gamma”:

$$\ln(\Gamma(z))=\ln\left(\int_0^\infty x^{z-1}\cdot\mathrm{e}^{-x}\,dx\right)$$

Εκτός των παραπάνω χρησιμοποιούνται και ορισμένες πιο τετριμμένες συναρτήσεις όπως η εκθετική συνάρτηση, το τόξο εφαπτομένης, η απόλυτη τιμή και ο φυσικός λογάριθμος. Ορίζονται επίσης 3 πίνακες (A, B, C) με δεδομένα εισόδου τυχαίους πραγματικούς αριθμούς (μέσω εσωτερικών συναρτήσεων παραγωγής τυχαιότητας) από το 0 έως το 100. Έτσι μπορούμε, έχοντας παράξει τους πίνακες των τυχαίων αριθμών, μέσω της χρήσης των ίδιων συναρτήσεων με την ίδια σειρά να παράγουμε κάθε φορά τα ίδια αποτελέσματα (είτε σειριακά είτε παράλληλα, με επιλογή διαφορετικού μήκους διανύσματος κάθε φορά) και να έχουμε έλεγχο αναμεταξύ τους.

- Τρεις πίνακες τυχαίων τιμών (a, b, c) μεταφέρονται στη συσκευή.
- Για κάθε στοιχείο i υπολογίζεται η παρακάτω παράσταση:

$$\Gamma\left(\left|100\cdot erf\left(\cos\left(1+\mathrm{e}^{5\cdot\arctan\left(\left|\ln\left(a[i]\cdot b[i]\div c[i]^2\right)\right|\right)}\right)\right)\right|\right)$$

- Η παράσταση αποθηκεύεται στο αντίστοιχο στοιχείο του πίνακα εξόδου

Αρχικά υπολογίζεται και αποθηκεύεται σε πίνακα εξόδου η παραπάνω παράσταση σειριακά, για ένα συγκεκριμένο μήκος πίνακα που έχει οριστεί ώστε στο τέλος να χρησιμοποιηθεί για έλεγχο των αποτελεσμάτων. Εξετάζοντας και συγκρίνοντας τα αποτελέσματα, ενώ είναι προφανές ότι οι δύο γλώσσες (C & OpenCL C) υπολογίζουν τον ίδιο αριθμό, υπάρχουν αποκλίσεις ως προς την ακρίβεια. Αυτό οφείλεται βέβαια στο ότι η παράσταση και οι συναρτήσεις που χρησιμοποιήθηκαν είναι αρκετά περίπλοκες, ώστε η OpenCL C επιβάλλοντας λιγότερο αυστηρούς περιορισμούς ως προς την ακρίβεια, όπως αναφέρθηκε και παραπάνω (η αποθήκευση και οι πράξεις μεταξύ πραγματικών αριθμών στον υπολογιστή μπορεί να κρύβουν πολλές εκπλήξεις (98)) να παράγει μικρά σφάλματα τα οποία διαδίδονται και ενδεχομένως μεγεθύνονται. Τα τελικά αποτελέσματα διέφεραν ελαφρώς κατά απόλυτη τιμή, αλλά μικρές διαφορές κατά απόλυτη τιμή για μικρούς αριθμούς αποτελούν μεγάλες ποσοστιαίες διαφορές. Και πάλι εάν η ακρίβεια των αποτελεσμάτων αποτελεί κρίσιμη παράμετρο, θα πρέπει να επιβληθούν συγκεκριμένοι περιορισμοί ώστε να έχουμε τις ίδιες προσδοκίες ακρίβειας με γλώσσες που ακολουθούν πιστά τις οδηγίες του *IEEE-754* (ή μπορούμε να χρησιμοποιήσουμε μεταβλητές μεγαλύτερης ακρίβειας).

Κατόπιν ορίζονται μεθοδικά διανύσματα μήκους 1, 2, 4, 8 & 16 στοιχείων και γίνεται η παράλληλη επεξεργασία των ίδιων δεδομένων κατά τον ίδιο τρόπο ενώ καταγράφεται το μέσο κόστος σε χρόνο για κάθε μήκος διανύσματος κάνοντας 10 δοκιμές (*runs*). Εδώ χρησιμοποιήθηκε εκτός της συνάρτησης `gettimeofday()` και η δυνατότητα που μας δίνει η OpenCL Runtime για

profiling κατευθείαν πάνω στα *events* που ορίζουν την κάθε επεξεργασία που συμβαίνει. Έτσι έχουμε ακριβέστερη εικόνα για το που καταναλώνεται ο χρόνος, μεγαλύτερη ακρίβεια, ενώ έχουμε πολύ καλύτερη ανάλυση χρόνων (τυπικά κάτω του 1μs, με ορισμένες συσκευές να αναφέρουν χρονική ανάλυση 1ns). Ο τύπος μεταβλητών που χρησιμοποιήθηκε ήταν πραγματικοί αριθμοί μονής ακρίβειας (`float`).

Έγινε προσπάθεια να χρησιμοποιηθούν όσο το δυνατόν μεγαλύτερα arrays ώστε να έχουμε το μέγιστο υπολογιστικό φορτίο και φορτίο μεταφοράς δεδομένων. Όμως παρόλο που στις περισσότερες περιπτώσεις το συνολικό μέγεθος της διαθέσιμης μνήμης θα επέτρεπε θεωρητικά τη δέσμευση της μνήμης που ζητάει το πρόγραμμα, αυτό δεν κατέστη δυνατόν να είναι το ίδιο σε όλες τις πλατφόρμες που εξετάστηκαν. Η συμπεριφορά του memory allocator του λειτουργικού συστήματος δεν είναι η ίδια για διαφορετικές αρχιτεκτονικές και λειτουργικά συστήματα, και μπορεί να μην είναι αναπαράξιμη ακόμα και στο ίδιο σύστημα, σε δεύτερο χρόνο. Έτσι τα αποτελέσματα είναι συγκρίσιμα μόνο ως προς τη συσκευή που αναφέρονται, και όχι μεταξύ των συσκευών (πχ συμπεραίνοντας ποια είναι ταχύτερη, αφού ο όγκος των δεδομένων ήταν διαφορετικός κάθε φορά). Το μήκος των arrays που χρησιμοποιήθηκαν αναφέρεται μαζί με το όνομα της συσκευής. Το ποσοστό βελτίωσης αναφέρεται στην ποσοστιαία επιτάχυνση του συνόλου των εργασιών της παράλληλης επεξεργασίας σε σχέση με τη σειριακή εκτέλεση, για τις 2 περιπτώσεις εκτέλεσης σε επεξεργαστές CPU, μετρούμενο με τη συνάρτηση `gettimeofday()`, ενώ για την περίπτωση του GPU, αφορά στην ποσοστιαία επιτάχυνση του συνόλου των εργασιών της παράλληλης επεξεργασίας σε σχέση με την παράλληλη επεξεργασία χωρίς διανύσματα (αφού δεν μπορούμε να έχουμε σειριακή εκτέλεση στην συγκεκριμένη συσκευή). Ο χρόνος στο πεδίο kernel execution είναι ο χρόνος εκτέλεσης του kernel όπως τον δίνει η OpenCL Runtime, δηλαδή ο χρόνος εκτέλεσης των αριθμητικών πράξεων και μόνο.

Όπως είναι προφανές, και φάνηκε σε όλες τις περιπτώσεις επεξεργαστών είτε CPU είτε GPU, διαφορετικό μήκος διανύσματος επηρεάζει μόνο τους χρόνους εκτέλεσης των kernels και όχι τη μεταφορά των δεδομένων (αυτό είναι μια λεπτομέρεια που μόνο μέσω του timing που προσφέρει η OpenCL μπορούμε να επιβεβαιώσουμε). Επίσης το `-cl-opt-disable` flag δεν επηρεάζει τους χρόνους μεταφοράς των δεδομένων. Παρατηρούμε ακόμη ότι το “preferred vector width” που αναφέρει η συσκευή δεν είναι αυτό που επιτρέπει πάντα τους ταχύτερους υπολογισμούς, είναι όμως πολύ κοντά. Κάτι αξιομνημόνευτο αποτελεί το γεγονός ότι η runtime αναφέρει το χρόνο μεταφοράς του πίνακα εξόδου πίσω στον host στην περίπτωση του GPU ως μηδενικό.

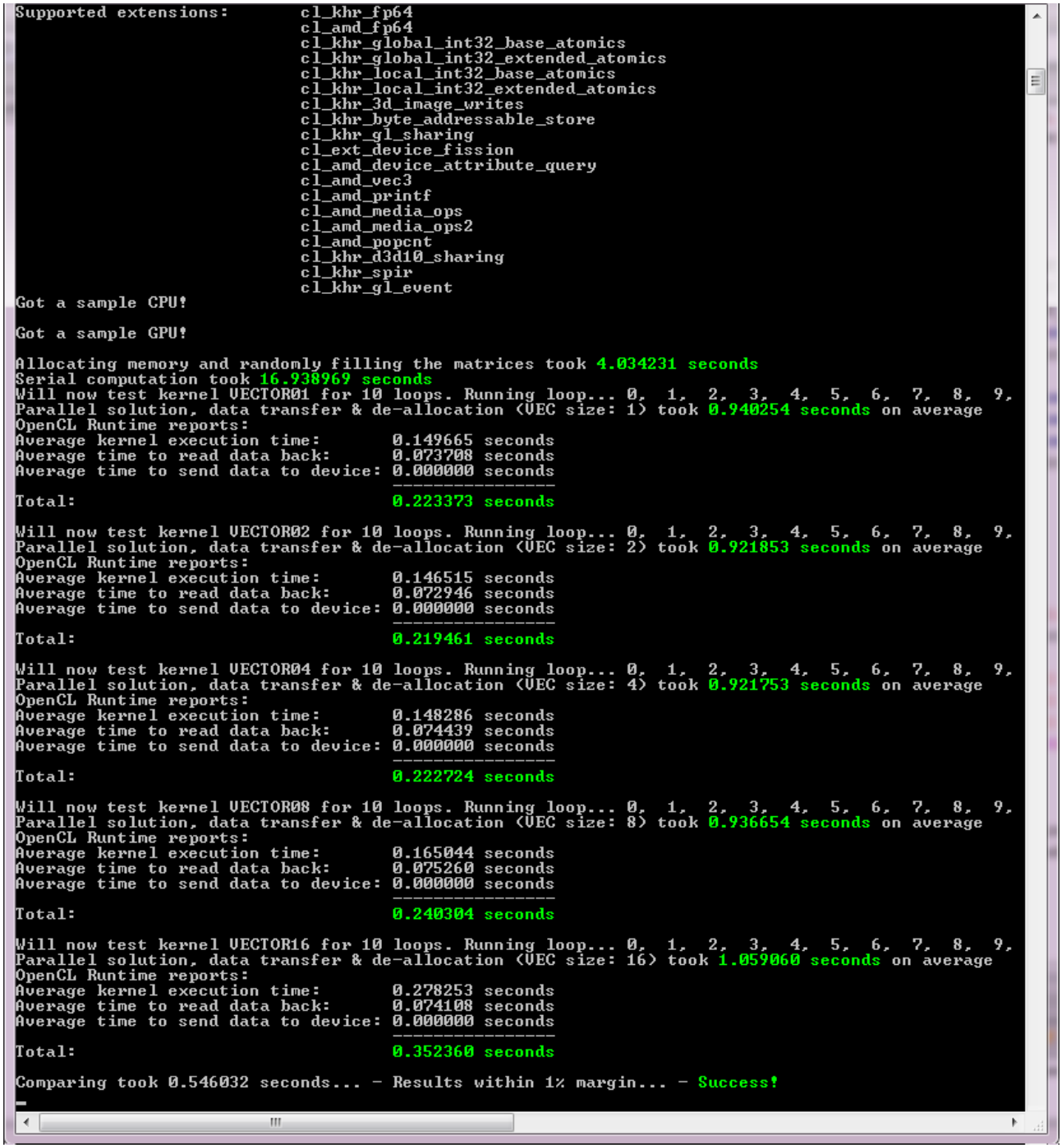
Το πρόγραμμα που χρησιμοποιήθηκε παρατίθεται στο αντίστοιχο παράρτημα.

*Πίνακας 10: Σύγκριση βελτίωσης της παραλληλοποίησης με χρήση διανυσμάτων για διάφορες συσκευές*

| Συσκευή | Παράμετροι | Μέτρηση | Σειριακή εκτέλεση (s) | Παράλληλη εκτέλεση χωρίς διανύσ-ματα (s) | Παράλληλη εκτέλεση (μήκος διανύσματος: 2) (s) | Παράλληλη εκτέλεση (μήκος διανύσματος: 4) (s) | Παράλληλη εκτέλεση (μήκος διανύσματος: 8) (s) | Παράλληλη εκτέλεση (μήκος διανύσματος: 16) (s) |
|---|---|---|---|---|---|---|---|---|
| Intel Core i3-2100 Sandy Bridge @ 3.10GHz<br><br>Preferred float vector width: 8<br><br>(n=65,000,000) | No compiler flags | Συνολικά | 14.277817 | 3.095777 | 2.876864 | 2.271530 | 2.407938 | 2.422139 |
| | | Kernel execution | - | 2.854223 | 2.648944 | 2.042061 | 2.174112 | 2.191164 |
| | | Ποσοστό βελτίωσης | - | 461.20% | 496.30% | 628.56% | 592.95% | 589.47% |
| | -cl-opt-disable | Συνολικά | - | 9.029716 | 8.487485 | 5.749029 | 5.890837 | 5.672325 |
| | | Kernel execution | - | 8.792962 | 8.260663 | 5.523845 | 5.659778 | 5.444055 |
| | | Ποσοστό βελτίωσης | - | 158.12% | 168.22% | 248.35% | 242.37% | 251.71% |
| Intel Core 2 Quad Q9550 @ 2.83GHz (Yorkfield)<br><br>Preferred float vector width: 4<br><br>(n=60,000,000) | No compiler flags | Συνολικά | 16.888966 | 3.657509 | 3.478699 | 2.931367 | 2.869864 | 2.794360 |
| | | Kernel execution | - | 2.866753 | 2.706167 | 2.153459 | 2.083451 | 2.011790 |
| | | Ποσοστό βελτίωσης | | 461.76% | 485.50% | 576.15% | 588.49% | 604.39% |
| | -cl-opt-disable | Συνολικά | - | 9.833762 | 9.321433 | 6.270059 | 6.385365 | 6.394166 |
| | | Kernel execution | - | 9.098479 | 8.592875 | 5.544607 | 5.650118 | 5.662501 |
| | | Ποσοστό βελτίωσης | - | 171.74% | 181.18% | 269.36% | 264.49% | 264.13% |
| Gigabyte AMD Radeon HD7950 @ 1GHz (Tahiti Pro)<br><br>Preferred float vector width: 1<br><br>(n=60,000,000) | No compiler flags | Συνολικά | - | 0.940254 | 0.921853 | 0.921753 | 0.936654 | 1.059060 |
| | | Kernel execution | - | 0.149665 | 0.146515 | 0.148286 | 0.165044 | 0.278253 |
| | | Ποσοστό βελτίωσης | | (100%) | 102.00% | 102.01% | 100.38% | 88.78% |
| | -cl-opt-disable | Συνολικά | - | 0.925453 | 0.918152 | 0.916152 | 0.927253 | 0.951054 |
| | | Kernel execution | - | 0.149205 | 0.147037 | 0.148849 | 0.151316 | 0.186070 |
| | | Ποσοστό βελτίωσης | - | (100%) | 100.80% | 101.01% | 99.81% | 97.31% |

Όπως μπορούμε να διαπιστώσουμε και από τα παρατεθημένα στοιχεία, η χρήση διανυσμάτων μπορεί να επιταχύνει την ήδη παράλληλη εκτέλεση, μειώνοντας το χρόνο εκτέλεσης του kernel από 28%~29% στην περίπτωση που δεν χρησιμοποιούμε compiler flags έως 38%~39% στην περίπτωση χρήσης του `-cl-opt-disable` flag όταν η εκτέλεση γίνεται σε επεξεργαστές CPU. Οπωσδήποτε αυτή είναι μια πολύ σημαντική επιτάχυνση, η οποία έρχεται με μηδενικό επιπλέον κόστος, πέραν αυτού της συγγραφής συνθετότερων προγραμμάτων (μόνο για όσα προγράμματα επεξεργάζονται στοιχεία που έχουν κάποια αλληλεπίδραση μεταξύ τους. Για παντελώς ανεξάρτητα στοιχεία, η αύξηση της περιπλοκότητας είναι μηδενική όπως για παράδειγμα στη συγκεκριμένη περίπτωση).

Για επεξεργαστές GPU η αύξηση ταχύτητας με χρήση διανυσμάτων είναι από μηδενική έως αρνητική. Αυτό είναι αναμενόμενο για τις συσκευές αυτές αποτελούμενες από πολύ απλούς streaming processors μη υλοποιώντας επιπλέον SIMD δομές, και που ενδεχομένως το buffer για το κάθε processing element να μην είναι aligned ή να μην χωράει μεγάλα διανύσματα (όπως φαίνεται στη δραστική μείωση της ταχύτητας με χρήση διανύσματος 16 μεταβλητών `float`, που κατά πάσα πιθανότητα αντιστοιχεί σε 16 x 4 = 64 bytes = 512 bits). Μακροσκοπικά οι συσκευές αυτές φαίνονται πιο “δύσκαμπτες” ως προς τις ρυθμίσεις που μπορούν να δεχθούν – αυτό έχει ήδη φανεί με τη μηδενική επίδραση του `-cl-opt-disable` flag σε όλους τους ελέγχους που έχουμε κάνει έως τώρα, και φαίνεται μια ακόμη φορά με τη μη επίδραση του μήκους του διανύσματος.




*Εικόνα 12: Ενδεικτική επίδραση της χρήσης διανυσμάτων (AMD Radeon HD7950)*

# Επίλογος

Στην παρούσα εργασία έγινε επίδειξη των δυνατοτήτων του συνδυασμού της παράλληλης επεξεργασίας με το σημερινό διαθέσιμο υλικό υπολογιστών. Τα αποτελέσματα είναι προκαταρκτικά και σε καμία περίπτωση δεν εξαντλούνται οι πιθανές διαθέσιμες βελτιώσεις. Έγινε συνοπτική παρουσίαση του προτύπου που χρησιμοποιήθηκε, που είναι και το προτιμητέο την περίοδο συγγραφής της παρούσας, με την προοπτική μελλοντικής ενσωμάτωσης στο *Vulkan* API. Διαπιστώσαμε ότι για αμιγώς παράλληλο φορτίο, συσκευές κατά πολύ φθηνότερες, και με πολύ πιο μικρή κατανάλωση απ' ότι επεξεργαστές αρχιτεκτονικής *x86* είναι πιο γρήγορες από τους εν λόγω επεξεργαστές. Ελέγξαμε τα αποτελέσματα της παραλληλοποίησης σε ένα ευρύ φάσμα συσκευών καταλήγοντας στα ίδια συμπεράσματα. Διαπιστώσαμε την υποστήριξη του προτύπου σε πλήθος αρχιτεκτονικών, δραστικά διαφορετικών από κλασσικούς desktop/server υπολογιστές. Τέλος, έγινε επίδειξη περαιτέρω αύξησης της ταχύτητας με χρήση OpenCL vectors. Το πρότυπο είναι ανοιχτό και διαθέσιμο σε όλους, δεν απαιτούνται άδειες ή χρηματικές καταβολές για την ανάπτυξη του αντίστοιχου λογισμικού, ενώ είναι διαθέσιμο για κάθε σύγχρονο λειτουργικό σύστημα υπολογιστή.

Εάν διορθωθεί το αλγοριθμικό σφάλμα που υπάρχει στα προγράμματα πλήρους επίλυσης του πεδίου, θα έχουμε καλύτερη εικόνα για την επιτάχυνση που μπορούμε να πετύχουμε για πραγματικούς αλγορίθμους. Σε αυτό το στάδιο, δεν θα έχουμε ανάγκη ακριβούς επίλυσης των συστημάτων γραμμικών εξισώσεων που προκύπτουν, και έτσι επαναληπτικές μέθοδοι θα μας δώσουν και σε αυτήν την περίπτωση ταχύτερα αποτελέσματα. Εξάλλου η παραλληλοποίηση των μεθόδων αυτών είναι άμεση, με ενδεικτική αυτήν του αλγορίθμου *Jacobi* στη βιβλιογραφία.

Χρήση διανυσμάτων θα μας δώσει τέλος ακόμη καλύτερες επιδόσεις, κάνοντας όμως το πρόγραμμα αρκετά πιο περίπλοκο. Ίσως χρειαστούν συναρτήσεις κατ' ευθείαν πρόσβασης και χειρισμού της μνήμης της συσκευής, που προϋποθέτει γνώση του byte ordering. Το κυρίως πρόγραμμα μπορεί σε αυτήν την περίπτωση να ελέγχει κατά την εκτέλεση το endianness της συσκευής και να δίνει τις κατάλληλες πληροφορίες στους kernels.

Κατόπιν των παραπάνω, βελτιώσεις ως προς τη μεταφορά των δεδομένων μπορούν να πραγματοποιηθούν μετρώντας μέσω της OpenCL Runtime (βλ. αντίστοιχο παράρτημα) τους χρόνους για διαφορετικές μεθόδους τροφοδοσίας των συσκευών με τα buffers εισόδου/εξόδου, οι οποίοι προφανώς θα διαφέρουν από συσκευή σε συσκευή (πχ η μεταφορά δεδομένων στην κάρτα γραφικών ενός desktop υπολογιστή περιορίζεται από τη συχνότητα λειτουργίας του διαύλου *PCI Express*, ενώ για τροφοδοσία του επεξεργαστή, δεν απαιτείται μεταφορά καθότι χρησιμοποιούνται τα pointers των αρχικών δεδομένων στη μνήμη).

Συμβουλές και *best practices* για υλοποίηση kernels για παντός είδους συσκευές μπορεί να αναζητήσει κανείς από τους κατασκευαστές των συσκευών αυτών, οι οποίοι γνωρίζουν καλύτερα την αρχιτεκτονική του υλικού τους, και ποιες δομές της OpenCL την αποτυπώνουν καλύτερα. Μικρές αλγοριθμικές αλλαγές μπορούν να εκμεταλλεύονται καλύτερα το υλικό που έχουμε διαθέσιμο κάθε φορά (πχ *memory coalescing* για καλύτερη εκμετάλλευση της ταχύτερης cache) Κάποιο γενικοί κανόνες ισχύουν για συγκεκριμένες ομάδες συσκευών (πχ GPUs). Χρήσιμη σε αυτήν την περίπτωση είναι και η μελέτη του μοντέλου μνήμης της OpenCL.

# ΠΑΡΑΡΤΗΜΑ 1 - Επισκόπηση και παρουσίαση πλήρους προγράμματος FORTRAN

Αρχικά ορίζεται το MODULE «FUNCTIONS» που περιέχει 2 διαδικασίες. Η πρώτη ($SUBROUTINE\ VECTORS\_TO\_MATRIX\ (XP, YP, AP, AE, AW, AN, AS, A)$) δέχεται ως ορίσματα τα διανύσματα $AP, AE, AW, AN, AS$ και τα αποτυπώνει στον πενταδιαγώνιο πίνακα $A$. Ο πίνακας αυτός θα χρησιμοποιηθεί στη δεύτερη διαδικασία ($SUBROUTINE\ AXB\_SOLVER\ (N, A, B, X)$) που είναι επιλύτης γραμμικού συστήματος εξισώσεων με τη μέθοδο Gauss.

Ο λόγος που χρησιμοποιείται επιλύτης γραμμικού συστήματος εξισώσεων με τη μέθοδο Gauss έναντι κάποιας επαναληπτικής μεθόδου είναι ότι οι συντελεστές $AP$ ενδέχεται να προκύψουν μικρότεροι κατά απόλυτη τιμή από το άθροισμα των $AE, AW, AN, AS$, ιδιαίτερα στους πρώτους κύκλους του αλγορίθμου SIMPLE όπου και οι αρχικές τιμές είναι αυθαίρετες. Έτσι μια επαναληπτική διαδικασία στον πίνακα $A$ που δεν έχει απαραίτητα αυστηρή διαγώνια υπεροχή δεν θα είχε εγγυημένη σύγκλιση.

Το πρόγραμμα ξεκινάει ορίζοντας τις μεταβλητές και τους πίνακες που θα χρησιμοποιηθούν:

- Διανύσματα $u, v, pp$ και τα αντίστοιχα διανύσματα των συντελεστών τους ($AP, AE, AW, AN, AS$)
- Πίνακας $A$ για χρήση στον γραμμικό επιλύτη,
- Ροές $FE, FW, FN, FS$
- 8 μεταβλητές που αναπαριστούν τις διαφορές αποστάσεων που θα χρησιμοποιηθούν συνολικά στο πρόγραμμα (CAP_XI_XIM1, Yjp1_Yj, Xip1_Xi, Xi_Xim1, CAP_YJP1_YJ, CAP_YJ_YJM1, CAP_XIP1_XI, Yj_Yjm1),
- $TERM, TERM1, TERM2, TERM3, TERM4$ κατά αντιστοιχία με την παραπάνω θεωρία για ευκολότερη γραφή των συντελεστών
- Πυκνότητα, ιξώδες, αρχική ταχύτητα ($velin$), συντελεστές υπο-χαλάρωσης για $u, v, p$ καθώς και
- $XPOINTS, YPOINTS, N$ (αριθμός διαφορετικών σημείων στον πραγματικό χώρο) $XPOINTSP1, YPOINTSP1, M$ (αριθμός κόμβων $u, v, p$)

Γίνεται αρχικοποίηση των $u, v, p$ καθώς και των συντεταγμένων των πλεγματικών γραμμών όπως έχει περιγραφεί παραπάνω. Το $DO\ LOOP$ με το μετρητή $K$ είναι οι επαναλήψεις του αλγορίθμου SIMPLE. Γίνεται αρχικά υπολογισμός των συντελεστών $AP, AE, AW, AN, AS$ των **<u>εσωτερικών κόμβων του πλέγματος</u>** για $u, v$, γι' αυτό και οι μετρητές $IDX$ & $IDY$ έχουν τα όρια $1, XPOINTSP1 - 2$ και $1, YPOINTSP1 - 2$ αντίστοιχα. Τα πλήρη όρια είναι ο αριθμός των κόμβων κατά $x$ & $y$ που είναι $XPOINTSP1$ και $YPOINTSP1$ δηλαδή

$0\ έως\ (XPOINTSP1 - 1)$

$0\ έως\ (YPOINTSP1 - 1)$

Μετά τον υπολογισμό για τους εσωτερικούς κόμβους γίνεται ο υπολογισμός για τους οριακούς κόμβους σύμφωνα με τη θεωρία παραπάνω:

Για όριο $E$: $u_{id} = u_{id-1}$, $v_{id} = v_{id-1}$

Για όριο $W$: $u_{id} = velin$, $v_{id} = 0$

Για όριο $S$: $u_{id} = 0$, $v_{id} = 0$

Για όριο $N$: $u_{id} = velin$, $v_{id} = 0$

Μετά την επίλυση των παραπάνω συστημάτων, γίνεται υπολογισμός των συντελεστών $AP, AE, AW, AN, AS$ για το σύστημα που θα λύσει τις «διαφορές πιέσεων» (P prime, ή $pp$). Ο υπολογισμός γίνεται μόνο για τους εσωτερικούς κόμβους, ενώ για τους οριακούς οι συντελεστές τίθενται έτσι ώστε το pp να μηδενιστεί (όταν το σύνολο των $pp$ υπερτεθεί στο σύνολο των $p$ οι **ακραίοι** κόμβοι του **νέου** συνόλου θα πάρουν τις τιμές των γειτονικών τους. Επομένως δεν χρειάζεται τιμή $pp$ για τους οριακούς κόμβους).

Μετά την επίλυση του συστήματος για το $pp$, έχουμε την υπέρθεση του συνόλου $pp$ πάνω στο $p$ με έναν συντελεστή $urf\_p$ **μόνο για τους εσωτερικούς κόμβους.** Οι ακραίοι κόμβοι θα πάρουν τις τιμές των γειτονικών τους ανάλογα με το ποιο όριο βρίσκονται. Έτσι έχουμε το νέο σύνολο των πιέσεων $p$ το οποίο συμπληρώνει την τριάδα των διανυσμάτων $u, v, p$ και ολοκληρώνεται μια επανάληψη του αλγορίθμου SIMPLE.

Το επόμενο διπλό DO LOOP εκτυπώνει στην οθόνη τις ταχύτητες $u$ που έχουν προκύψει έπειτα από την ολοκλήρωση του προγράμματος.

```
MODULE FUNCTIONS
CONTAINS

SUBROUTINE VECTORS_TO_MATRIX (XP, YP, AP, AE, AW, AN, AS, A)
    INTEGER, INTENT(IN):: XP, YP
    REAL, DIMENSION (:), INTENT(IN):: AP(0:XP*YP-1), AE(0:XP*YP-1), AW(0:XP*YP-1), AN(0:XP*YP-1), AS(0:XP*YP-1)
    REAL, DIMENSION (:), ALLOCATABLE, INTENT(OUT):: A

    INTEGER:: N, I

    N=XP*YP

    ALLOCATE( A(0:N*N-1) )

    DO I=0,N-1
        A(I*N+I)=-AP(I)
    END DO
    DO I=0,N-2
        A(I*N+I+1)=AE(I)
    END DO
    DO I=1,N-1
        A(I*N+I-1)=AW(I)
    END DO
    DO I=0,N-1
        A(I*N+I+XP)=AN(I)
    END DO
    DO I=XP,N-1
        A(I*N+I-XP)=AS(I)
    END DO

END SUBROUTINE

SUBROUTINE AXB_SOLVER (N, A, B, X)
    INTEGER, INTENT(IN):: N
    REAL, DIMENSION (:), INTENT(INOUT):: A(0:N*N-1), B(0:N-1)
    REAL, DIMENSION (:), ALLOCATABLE, INTENT(OUT)::X
    INTEGER::I,J,K, INDX
    REAL::BIGGEST, TEMP, SM

    ALLOCATE( X(0:N-1) )

    DO J=0,N-2
        BIGGEST = ABS(A(J*N+J))
        INDX=J
        DO K=J+1,N-1
            IF (ABS(A(K*N+J))>BIGGEST) THEN
                BIGGEST = ABS(A(K*N+J))
                INDX = K
            END IF
        END DO

        IF (INDX/=J) THEN
            DO K=J,N-1
                TEMP = A(J*N+K)
                A(J*N+K) = A(INDX*N+K)
                A(INDX*N+K)=TEMP
            END DO
            TEMP=B(J)
            B(J)=B(INDX)
            B(INDX)=TEMP
        END IF

        DO I=J+1,N-1
            TEMP=A(I*N+J)/A(J*N+J)
            DO K=J,N-1
                A(I*N+K) = A(I*N+K) - TEMP*A(J*N+K)
            END DO
            B(I) = B(I) - TEMP*B(J)
        END DO
    END DO



    X(N-1) = -B(N-1)/A(N*N-1)
    DO I=N-2,0,-1
        SM=0
        DO J=I+1,N-1
            SM = SM + A(I*N+J)*X(J)
        END DO
        X(I) = (-B(I)-SM)/A(I*N+I)
    END DO
END SUBROUTINE

END MODULE FUNCTIONS
```

```
PROGRAM FORTEST
    USE FUNCTIONS
    IMPLICIT NONE
    INTEGER::I, J, K, IDX, IDY, ID, XPOINTS, YPOINTS, XPOINTSP1, YPOINTSP1, N, M
    REAL, DIMENSION(:), ALLOCATABLE:: APU, AWU, AEU, ANU, ASU, BU
    REAL, DIMENSION(:), ALLOCATABLE:: APV, AWV, AEV, ANV, ASV, BV
    REAL, DIMENSION(:), ALLOCATABLE:: APP, AWP, AEP, ANP, ASP, BP
    REAL, DIMENSION(:), ALLOCATABLE:: A
    REAL, DIMENSION(:), ALLOCATABLE:: X, Y
    REAL, DIMENSION(:), ALLOCATABLE:: U, V, P, PP
    REAL::FE, FW, FN, FS
    REAL::TERM, TERM1, TERM2, TERM3, TERM4
    REAL::CAP_XI_XIM1, Yjp1_Yj, Xip1_Xi, Xi_Xim1, CAP_YJP1_YJ, CAP_YJ_YJM1, CAP_XIP1_XI, Yj_Yjm1
    REAL::DENS=1., VISC=0.003, VELIN=30., URF_U=0.5, URF_V=0.5, URF_P=0.5

    XPOINTS = 10
    YPOINTS = 5

    XPOINTSP1 = XPOINTS + 1
    YPOINTSP1 = YPOINTS + 1

    N = XPOINTS*YPOINTS
    M = XPOINTSP1*YPOINTSP1

    ALLOCATE( A(0:M*M-1) )

    ALLOCATE( X(0:XPOINTS+1), Y(0:YPOINTS+1) )
    ALLOCATE( U(0:M-1), V(0:M-1), P(0:M-1), PP(0:M-1) )

    ALLOCATE( APU(0:M-1), AEU(0:M-1), AWU(0:M-1), ANU(0:M-1), ASU(0:M-1), BU(0:M-1) )
    ALLOCATE( APV(0:M-1), AEV(0:M-1), AWV(0:M-1), ANV(0:M-1), ASV(0:M-1), BV(0:M-1) )
    ALLOCATE( APP(0:M-1), AEP(0:M-1), AWP(0:M-1), ANP(0:M-1), ASP(0:M-1), BP(0:M-1) )

    DO I=0,M-1
        U(I)=1.0
        V(I)=1.0
        P(I)=0.0
    END DO

    DO I=1,XPOINTS
        X(I) = I-1.
    END DO

    DO I=1,YPOINTS
        Y(I) = I-1.
    END DO

    X(0) = X(1)
    X(XPOINTS+1)=X(XPOINTS)
    Y(0) = Y(1)
    Y(YPOINTS+1) = Y(YPOINTS)

    DO K=0,10
        PRINT*,"Currently working on SIMPLE iteration", K
        DO IDY=1,YPOINTSP1-2
            DO IDX=1,XPOINTSP1-2
                ID = IDY*XPOINTSP1 + IDX

                !USED TO COMPUTE CO-EFFICIENTS FOR BOTH U & V
                CAP_XI_XIM1 = (X(IDX)+X(IDX+1))/2 - (X(IDX-1)+X(IDX))/2
                Yjp1_Yj = Y(IDY+1) - Y(IDY)
                Xip1_Xi = X(IDX+1) - X(IDX)
                CAP_YJ_YJM1 = (Y(IDY)+Y(IDY+1))/2 - (Y(IDY-1)+Y(IDY))/2

                !USED TO COMPUTE CO-EFFICIENTS FOR U
                Xi_Xim1 = X(IDX) - X(IDX-1)
                CAP_YJP1_YJ = (Y(IDY+1)+Y(IDY+2))/2 - (Y(IDY)+Y(IDY+1))/2

                !USED TO COMPUTE CO-EFFICIENTS FOR V
                CAP_XIP1_XI = (X(IDX+1)+X(IDX+2))/2 - (X(IDX)+X(IDX+1))/2
                Yj_Yjm1 = Y(IDY) - Y(IDY-1)

                ! ------------------------------
                ! Calculating u co-efficients
                ! ------------------------------

                FE = DENS*(U(ID)+U(ID+1))/2
                FW = DENS*(U(ID)+U(ID-1))/2
                FN = DENS*(V(ID+XPOINTSP1)+V(ID+XPOINTSP1-1))/2
                FS = DENS*(V(ID)+V(ID-1))/2

                TERM  = Xip1_Xi*Xi_Xim1*CAP_YJP1_YJ*CAP_YJ_YJM1
                TERM1 = 2*VISC*Yjp1_Yj *Xi_Xim1*CAP_YJP1_YJ*CAP_YJ_YJM1
                TERM2 = 2*VISC*Yjp1_Yj *Xip1_Xi*CAP_YJP1_YJ*CAP_YJ_YJM1
```

```
        TERM3 =   VISC*CAP_XI_XIM1 *Xip1_Xi*Xi_Xim1*CAP_YJ_YJM1
        TERM4 =   VISC*CAP_XI_XIM1 *Xip1_Xi*Xi_Xim1*CAP_YJP1_YJ

        APU(ID) = ((MAX(FE,0.)+MAX(-FW,0.))*Yjp1_Yj + (MAX(FN,0.)+MAX(-FS,0.))*CAP_XI_XIM1)*TERM &
                  + TERM1+TERM2+TERM3+TERM4
        AEU(ID) = MAX(-FE,0.)*Yjp1_Yj*TERM + TERM1
        AWU(ID) = MAX(FW,0.) *Yjp1_Yj*TERM + TERM2
        ANU(ID) = MAX(-FN,0.)*CAP_XI_XIM1*TERM + TERM3
        ASU(ID) = MAX(FS,0.) *CAP_XI_XIM1*TERM + TERM4

        BU(ID) = ((P(ID-1)-P(ID))*Yjp1_Yj + &
        VISC*(V(ID+XPOINTSP1) - V(ID-1+XPOINTSP1) - V(ID) + V(ID-1)))*TERM

        ! ------------------------------
        ! Calculating v co-efficients
        ! ------------------------------

        FE = DENS*(U(ID+1)+U(ID+1-XPOINTSP1))/2
        FW = DENS*(U(ID)+U(ID-XPOINTSP1))/2
        FN = DENS*(V(ID)+V(ID+XPOINTSP1))/2
        FS = DENS*(V(ID)+V(ID-XPOINTSP1))/2

        TERM  = CAP_XIP1_XI*CAP_XI_XIM1*Yjp1_Yj*Yj_Yjm1
        TERM1 =   VISC*CAP_YJ_YJM1*CAP_XI_XIM1*Yjp1_Yj*Yj_Yjm1
        TERM2 =   VISC*CAP_YJ_YJM1*CAP_XIP1_XI*Yjp1_Yj*Yj_Yjm1
        TERM3 = 2*VISC*Xip1_Xi*CAP_XIP1_XI*CAP_XI_XIM1*Yj_Yjm1
        TERM4 = 2*VISC*Xip1_Xi*CAP_XIP1_XI*CAP_XI_XIM1*Yjp1_Yj

        APV(ID) = ((MAX(FE,0.)+MAX(-FW,0.))*CAP_YJ_YJM1 + (MAX(FN,0.)+MAX(-FS,0.))*Xip1_Xi)*TERM &
                + TERM1+TERM2+TERM3+TERM4
        AEV(ID) = MAX(-FE,0.)*CAP_YJ_YJM1*TERM + TERM1
        AWV(ID) = MAX(FW,0.) *CAP_YJ_YJM1*TERM + TERM2
        ANV(ID) = MAX(-FN,0.)*Xip1_Xi*TERM + TERM3
        ASV(ID) = MAX(FS,0.) *Xip1_Xi*TERM + TERM4

        BV(ID) = ((P(ID-XPOINTSP1)-P(ID))*Xip1_Xi &
               + VISC*(U(ID+1) - U(ID+1-XPOINTSP1) - U(ID) + U(ID-XPOINTSP1)))*TERM

    END DO
END DO

DO I=XPOINTSP1, M-2*XPOINTSP1, XPOINTSP1
    APU(I)=1
    AEU(I)=0
    AWU(I)=0
    ANU(I)=0
    ASU(I)=0
    BU(I)=VELIN

    APV(I)=1
    AEV(I)=0
    AWV(I)=0
    ANV(I)=0
    ASV(I)=0
    BV(I)=0
END DO

DO I=2*XPOINTSP1-1, M-1-XPOINTSP1, XPOINTSP1
    APU(I)=1
    AEU(I)=0
    AWU(I)=1
    ANU(I)=0
    ASU(I)=0
    BU(I)=0

    APV(I)=1
    AEV(I)=0
    AWV(I)=1
    ANV(I)=0
    ASV(I)=0
    BV(I)=0
END DO

DO I=0, XPOINTSP1-1
    APU(I)=1
    AEU(I)=0
    AWU(I)=0
    ANU(I)=0
    ASU(I)=0
    BU(I)=0

    APV(I)=1
    AEV(I)=0
    AWV(I)=0
    ANV(I)=0
    ASV(I)=0
```

```
            BV(I)=0
        END DO

        DO I=M-XPOINTSP1, M-1
            APU(I)=1
            AEU(I)=0
            AWU(I)=0
            ANU(I)=0
            ASU(I)=0
            BU(I)=VELIN

            APV(I)=1
            AEV(I)=0
            AWV(I)=0
            ANV(I)=0
            ASV(I)=0
            BV(I)=0
        END DO

        A=0.0
        CALL VECTORS_TO_MATRIX(XPOINTSP1, YPOINTSP1, APU, AEU, AWU, ANU, ASU, A)
        CALL AXB_SOLVER (M, A, BU, U)

        A=0.0
        CALL VECTORS_TO_MATRIX(XPOINTSP1, YPOINTSP1, APV, AEV, AWV, ANV, ASV, A)
        CALL AXB_SOLVER (M, A, BV, V)

        DO IDY=1,YPOINTSP1-2
             DO IDX=1,XPOINTSP1-2
               ID=IDY*XPOINTSP1 + IDX

                CAP_XIP1_XI = (X(IDX+1)+X(IDX+2))/2 - (X(IDX)+X(IDX+1))/2
                Xip1_Xi = X(IDX+1) - X(IDX)
                CAP_XI_XIM1 = (X(IDX)+X(IDX+1))/2 - (X(IDX-1)+X(IDX))/2
                CAP_YJP1_YJ = (Y(IDY+1)+Y(IDY+2))/2 - (Y(IDY)+Y(IDY+1))/2
                Yjp1_Yj = Y(IDY+1) - Y(IDY)
                CAP_YJ_YJM1 = (Y(IDY)+Y(IDY+1))/2 - (Y(IDY-1)+Y(IDY))/2

                TERM1 = APU(ID) * APV(ID+XPOINTSP1) * APV(ID) * CAP_XI_XIM1 * CAP_YJP1_YJ * Yjp1_Yj *
CAP_YJ_YJM1
                TERM2 = APU(ID+1) * APV(ID+XPOINTSP1) * APV(ID) * CAP_XIP1_XI * CAP_YJP1_YJ * Yjp1_Yj *
CAP_YJ_YJM1
                TERM3 = APU(ID) * APU(ID+1) * APV(ID) * CAP_XIP1_XI * Xip1_Xi * CAP_XI_XIM1 * CAP_YJ_YJM1
                TERM4 = APU(ID) * APU(ID+1) * APV(ID+XPOINTSP1) * CAP_XIP1_XI * Xip1_Xi * CAP_XI_XIM1 *
CAP_YJP1_YJ

                APP(ID) = TERM1 + TERM2 + TERM3 + TERM4
                AEP(ID) = TERM1
                AWP(ID) = TERM2
                ANP(ID) = TERM3
                ASP(ID) = TERM4

                BP(ID)  = ((U(ID)-U(ID+1))*Yjp1_Yj + (V(ID)-V(ID+XPOINTSP1))*Xip1_Xi) &
                            * APU(ID)*APV(ID)*APU(ID+1)*APV(ID+XPOINTSP1) &
                            * CAP_XIP1_XI*CAP_XI_XIM1*CAP_YJP1_YJ*CAP_YJ_YJM1

            END DO
        END DO

        DO I=0,XPOINTSP1-1
           APP(I)=1
            AEP(I)=0
            AWP(I)=0
            ANP(I)=0
            ASP(I)=0
            BP(I)=0
        END DO

        DO I=M-XPOINTSP1,M-1
            APP(I)=1
            AEP(I)=0
            AWP(I)=0
            ANP(I)=0
            ASP(I)=0
            BP(I)=0
        END DO

        DO I=XPOINTSP1,M-2*XPOINTSP1,XPOINTSP1
            APP(I)=1
            AEP(I)=0
            AWP(I)=0
            ANP(I)=0
            ASP(I)=0
            BP(I)=0
        END DO
```

```
        DO I=2*XPOINTSP1-1,M-1-XPOINTSP1,XPOINTSP1
             APP(I)=1
            AEP(I)=0
            AWP(I)=0
            ANP(I)=0
            ASP(I)=0
            BP(I)=0
        END DO

        A=0.0
        CALL VECTORS_TO_MATRIX(XPOINTSP1, YPOINTSP1, APP, AEP, AWP, ANP, ASP, A)
        CALL AXB_SOLVER (M, A, BP, PP)

        DO I=0,M-1
            P(I) = P(I) + URF_P*PP(I)
        END DO

        DO I=XPOINTSP1,M-2*XPOINTSP1,XPOINTSP1
            P(I) = P(I+1)
        END DO

        DO I=2*XPOINTSP1-1,M-1-XPOINTSP1,XPOINTSP1
            P(I) = P(I-1)
        END DO

        DO I=0,XPOINTSP1-1
            P(I) = P(I+XPOINTSP1)
        END DO

        DO I=M-XPOINTSP1,M-1
            P(I) = P(I-XPOINTSP1)
        END DO

    END DO !SIMPLE ITERATIONS

    DO J=YPOINTSP1-1,0,-1
        DO I=0,XPOINTSP1-1
            WRITE(*,'(F9.6)',ADVANCE='NO') U(J*XPOINTSP1+I)
            WRITE(*,'(A2)',ADVANCE='NO')'  '
        END DO
        WRITE(*,*)
    END DO

END PROGRAM
```

# ΠΑΡΑΡΤΗΜΑ 2 - Επισκόπηση και παρουσίαση πλήρους προγράμματος C

Το πρόγραμμα σε C ακολουθεί την ίδια λογική με αυτό σε FORTRAN. Αρχικά γίνονται οι απαραίτητοι ορισμοί μεταβλητών και δέσμευση μνήμης. Επειδή η C κάνει διάκριση μεταξύ κεφαλαίων και πεζών χαρακτήρων στα ονόματα των μεταβλητών, δεν χρειαζόμαστε το πρόθεμα `CAP_` που χρησιμοποιήθηκε στο πρόγραμμα σε FORTRAN.

Στο πρώτο διπλό loop εντός του loop του αλγορίθμου SIMPLE γίνεται υπολογισμός των συντελεστών για τις ταχύτητες u, v, ενώ έπειτα γίνεται ο ορισμός τους στα ακραία σημεία. Για την επίλυση με τη συνάρτηση `diag5solver` χρειάζεται οι συντελεστές AP να αποκτήσουν αντίθετο πρόσημο, γιατί μπαίνουν στον ίδιο πίνακα με τους υπόλοιπους συντελεστές, για την επίλυση του συστήματος (την ενέργεια αυτή είχε αναλάβει στο πρόγραμμα FORTRAN η υπορουτίνα `VECTORS_TO_MATRIX`). Επειδή οι συντελεστές θα ξαναχρησιμοποιηθούν, γίνεται επαναφορά στο αρχικό τους πρόσημο, μετά την επίλυση του συστήματος.

Έπειτα το πρόγραμμα προχωρά στον υπολογισμό των συντελεστών για το πεδίο P prime που θα προστεθεί στο αρχικό πεδίο πιέσεων. Η λογική είναι η ίδια με πριν, και αφού υπολογιστούν οι συντελεστές και επιλυθεί το πεδίο, προστίθεται στο πεδίο P, με χρήση ενός συντελεστή υποχαλάρωσης. Οι ακραίοι κόμβοι του πεδίου P παίρνουν τις τιμές των γειτονικών τους ανάλογα με το όριο που βρίσκονται.

Στο τέλος το πρόγραμμα τυπώνει σε αρχεία, τις τιμές των πεδίων u, v, p που υπολογίστηκαν. Για απόδοση χρωμάτων στα MS Windows χρησιμοποιούνται οι δομές `HANDLE` και οι συναρτήσεις `GetStdHandle, SetConsoleTextAttribute.`

Οι μορφές δεδομένων `cl_int` είναι ισοδύναμες με `int` και τοποθετήθηκαν επί τούτου για διευκόλυνση της αλληλομετατροπής του σειριακού προγράμματος σε παράλληλο και ανάποδα (για λόγους συμβατότητας).

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

#if defined(_WIN32)
#include <windows.h>
#endif

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

typedef double datatype;
/* datatype can be (based on required precision): float,  double,  __float128 */

void diag5solver (unsigned int xpoints, unsigned int ypoints, datatype *AP, datatype *AE, datatype *AW, datatype
*AN, datatype *AS, datatype *b, datatype *x);


int main() {

    HANDLE  hConsole;                           //For colors
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //For colors

    FILE* outputu = fopen ("u.txt", "w+");
    FILE* outputv = fopen ("v.txt", "w+");
    FILE* outputp = fopen ("p.txt", "w+");

    cl_int i,j,k,idx,idy,id;

    datatype *u, *v, *p, *pp, *X, *Y;

    datatype Fe, Fw, Fn, Fs;
    datatype XI_XIM1, Yjp1_Yj, Xip1_Xi, Xi_Xim1, YJP1_YJ, YJ_YJM1, XIP1_XI, Yj_Yjm1;
    datatype term1, term2, term3, term4, term;

    datatype *APu, *AEu, *AWu, *ANu, *ASu, *Bu;
    datatype *APv, *AEv, *AWv, *ANv, *ASv, *Bv;
    datatype *APp, *AEp, *AWp, *ANp, *ASp, *Bp;

    cl_uint xpoints=0, ypoints=0, xpointsp1=0, ypointsp1=0, n=0, m=0;
    datatype urf_u=0.5, urf_v=0.5, urf_p=0.5;
    datatype visc=0.003, dens=1.0, velin=30.0;

    A:
    xpoints=10;
    ypoints=5;
    xpointsp1=xpoints+1;
    ypointsp1=ypoints+1;

    n = xpoints*ypoints;
    m = xpointsp1*ypointsp1;

    u = malloc(m*sizeof(*u));
    v = malloc(m*sizeof(*v));
    p = malloc(m*sizeof(*p));
    pp = malloc(m*sizeof(*pp));

    X = malloc((xpoints+2)*sizeof(*X));
    Y = malloc((ypoints+2)*sizeof(*Y));

    APu = malloc(m*sizeof(*APu));
    AEu = malloc(m*sizeof(*AEu));
    AWu = malloc(m*sizeof(*AWu));
    ANu = malloc(m*sizeof(*ANu));
    ASu = malloc(m*sizeof(*ASu));
    Bu = malloc(m*sizeof(*Bu));

    APv = malloc(m*sizeof(*APv));
    AEv = malloc(m*sizeof(*AEv));
    AWv = malloc(m*sizeof(*AWv));
    ANv = malloc(m*sizeof(*ANv));
    ASv = malloc(m*sizeof(*ASv));
    Bv = malloc(m*sizeof(*Bv));

    APp = malloc(m*sizeof(*APp));
```

```
    AEp = malloc(m*sizeof(*AEp));
    AWp = malloc(m*sizeof(*AWp));
    ANp = malloc(m*sizeof(*ANp));
    ASp = malloc(m*sizeof(*ASp));
    Bp = malloc(m*sizeof(*Bp));

    for (i=0;i<=m-1;i++) {
        u[i]=5;
        v[i]=1;
        p[i]=1;
        //pp[i]=0;
    }

    for (i=1;i<=xpoints;i++) {
        X[i]=(float)i-1;
    }

    for (i=1;i<=ypoints;i++) {
        Y[i]=(float)i-1;
    }

    X[0]=X[1];
    X[xpoints+1]=X[xpoints];
    Y[0]=Y[1];
    Y[ypoints+1]=Y[ypoints];

    for (k=0;k<=10;k++) {//SIMPLE iterations
        printf("Currently working on SIMPLE iteration %d\n",k);
            for (idy=1; idy<=ypointsp1-2; idy++) {
                for (idx=1; idx<=xpointsp1-2; idx++) {
                    id = idy*xpointsp1+idx;

                    //Geometrikes posothtes gia paragwgh syntelestwn u kai v
                    XI_XIM1 = (X[idx]+X[idx+1])/2 - (X[idx-1]+X[idx])/2;
                    Yjp1_Yj = Y[idy+1] - Y[idy];
                    Xip1_Xi = X[idx+1] - X[idx];
                    YJ_YJM1 = (Y[idy]+Y[idy+1])/2 - (Y[idy-1]+Y[idy])/2;

                    //Geometrikes posothtes gia paragwgh syntelestwn u
                    Xi_Xim1 = X[idx] - X[idx-1];
                    YJP1_YJ = (Y[idy+1]+Y[idy+2])/2 - (Y[idy]+Y[idy+1])/2;

                    //Geometrikes posothtes gia paragwgh syntelestwn v
                    XIP1_XI = (X[idx+1]+X[idx+2])/2 - (X[idx]+X[idx+1])/2;
                    Yj_Yjm1 = Y[idy]-Y[idy-1];

                    // -------------------------------
                    // Calculating u co-efficients
                    // -------------------------------

                    Fe = dens*(u[id]+u[id+1])/2;
                    Fw = dens*(u[id]+u[id-1])/2;
                    Fn = dens*(v[id+xpointsp1]+v[id+xpointsp1-1])/2;
                    Fs = dens*(v[id]+v[id-1])/2;

                    term  = Xip1_Xi*Xi_Xim1*YJP1_YJ*YJ_YJM1;
                    term1 = 2*visc*Yjp1_Yj *Xi_Xim1*YJP1_YJ*YJ_YJM1;
                    term2 = 2*visc*Yjp1_Yj *Xip1_Xi*YJP1_YJ*YJ_YJM1;
                    term3 =   visc*XI_XIM1 *Xip1_Xi*Xi_Xim1*YJ_YJM1;
                    term4 =   visc*XI_XIM1 *Xip1_Xi*Xi_Xim1*YJP1_YJ;

                    APu[id] = ((fmax(Fe,0)+fmax(-Fw,0))*Yjp1_Yj + (fmax(Fn,0)+fmax(-Fs,0))*XI_XIM1)*term +
term1+term2+term3+term4;
                    AEu[id] = fmax(-Fe,0)*Yjp1_Yj*term + term1;
                    AWu[id] = fmax(Fw,0) *Yjp1_Yj*term + term2;
                    ANu[id] = fmax(-Fn,0)*XI_XIM1*term + term3;
                    ASu[id] = fmax(Fs,0) *XI_XIM1*term + term4;

                    Bu[id] = ((p[id-1]-p[id])*Yjp1_Yj + visc*(v[id+xpointsp1] - v[id-1+xpointsp1] - v[id] + v[id-
1]))*term;

                    // -------------------------------
                    // Calculating v co-efficients
                    // -------------------------------

                    Fe = dens*(u[id+1]+u[id+1-xpointsp1])/2;
                    Fw = dens*(u[id]+u[id-xpointsp1])/2;
                    Fn = dens*(v[id]+v[id+xpointsp1])/2;
                    Fs = dens*(v[id]+v[id-xpointsp1])/2;

                    term  = XIP1_XI*XI_XIM1*Yjp1_Yj*Yj_Yjm1;
```

```
                    term1 =   visc*YJ_YJM1*XI_XIM1*Yjp1_Yj*Yj_Yjm1;
                    term2 =   visc*YJ_YJM1*XIP1_XI*Yjp1_Yj*Yj_Yjm1;
                    term3 = 2*visc*Xip1_Xi*XIP1_XI*XI_XIM1*Yj_Yjm1;
                    term4 = 2*visc*Xip1_Xi*XIP1_XI*XI_XIM1*Yjp1_Yj;

                    APv[id] = ((fmax(Fe,0)+fmax(-Fw,0))*YJ_YJM1 + (fmax(Fn,0)+fmax(-Fs,0))*Xip1_Xi)*term +
term1+term2+term3+term4;
                    AEv[id] = fmax(-Fe,0)*YJ_YJM1*term + term1;
                    AWv[id] = fmax(Fw,0) *YJ_YJM1*term + term2;
                    ANv[id] = fmax(-Fn,0)*Xip1_Xi*term + term3;
                    ASv[id] = fmax(Fs,0) *Xip1_Xi*term + term4;

                    Bv[id] =((p[id-xpointsp1]-p[id])*Xip1_Xi + visc*(u[id+1] - u[id+1-xpointsp1] - u[id] + u[id-
xpointsp1]))*term;
                }
            }

            for (i=xpointsp1; i<=m-2*xpointsp1; i=i+xpointsp1) {
                APu[i]=1;                   APv[i]=1;
                AEu[i]=0;                   AEv[i]=0;
                AWu[i]=0;                   AWv[i]=0;
                ANu[i]=0;                   ANv[i]=0;
                ASu[i]=0;                   ASv[i]=0;
                Bu[i]=velin;                Bv[i]=0;
            }

            for (i=2*xpointsp1-1; i<=m-1-xpointsp1; i=i+xpointsp1) {
                APu[i]=1;                   APv[i]=1;
                AEu[i]=0;                   AEv[i]=0;
                AWu[i]=1;                   AWv[i]=1;
                ANu[i]=0;                   ANv[i]=0;
                ASu[i]=0;                   ASv[i]=0;
                Bu[i]=0;                    Bv[i]=0;
            }

            for (i=0; i<=xpointsp1-1; i++) {
                APu[i]=1;                   APv[i]=1;
                AEu[i]=0;                   AEv[i]=0;
                AWu[i]=0;                   AWv[i]=0;
                ANu[i]=0;                   ANv[i]=0;
                ASu[i]=0;                   ASv[i]=0;
                Bu[i]=0;                    Bv[i]=0;
            }

            for (i=m-xpointsp1; i<=m-1; i++) {
                APu[i]=1;                   APv[i]=1;
                AEu[i]=0;                   AEv[i]=0;
                AWu[i]=0;                   AWv[i]=0;
                ANu[i]=0;                   ANv[i]=0;
                ASu[i]=0;                   ASv[i]=0;
                Bu[i]=velin;                Bv[i]=0;
            }

            for (i=0; i<=m-1; i++) {
                APu[i] = -APu[i];
                APv[i] = -APv[i];
            }

            diag5solver (xpointsp1, ypointsp1, APu, AEu, AWu, ANu, ASu, Bu, u);

            for (i=0; i<=m-1; i++) {
                APu[i] = -APu[i];
            }

            diag5solver (xpointsp1, ypointsp1, APv, AEv, AWv, ANv, ASv, Bv, v);

            for (i=0; i<=m-1; i++) {
                APv[i] = -APv[i];
            }

            // ------------------------------
            // Calculating pp co-efficients
            // ------------------------------

            for (idy=1; idy<=ypointsp1-2; idy++) {
                for (idx=1; idx<=xpointsp1-2; idx++) {
                    id = idy*xpointsp1+idx;

                    XIP1_XI = (X[idx+1]+X[idx+2])/2 - (X[idx]+X[idx+1])/2;
                    Xip1_Xi = X[idx+1] - X[idx];
                    XI_XIM1 = (X[idx]+X[idx+1])/2 - (X[idx-1]+X[idx])/2;
```

```
                    YJP1_YJ = (Y[idy+1]+Y[idy+2])/2 - (Y[idy]+Y[idy+1])/2;
                    Yjp1_Yj = Y[idy+1] - Y[idy];
                    YJ_YJM1 = (Y[idy]+Y[idy+1])/2 - (Y[idy-1]+Y[idy])/2;

                    term1 = APu[id] * APv[id+xpointsp1] * APv[id] * XI_XIM1 * YJP1_YJ * Yjp1_Yj * YJ_YJM1;
                    term2 = APu[id+1] * APv[id+xpointsp1] * APv[id] * XIP1_XI * YJP1_YJ * Yjp1_Yj * YJ_YJM1;
                    term3 = APu[id] * APu[id+1] * APv[id] * XIP1_XI * Xip1_Xi * XI_XIM1 * YJ_YJM1;
                    term4 = APu[id] * APu[id+1] * APv[id+xpointsp1] * XIP1_XI * Xip1_Xi * XI_XIM1 * YJP1_YJ;

                    APp[id] = term1+term2+term3+term4;
                    AEp[id] = term1;
                    AWp[id] = term2;
                    ANp[id] = term3;
                    ASp[id] = term4;
                    Bp[id] = ((u[id]-u[id+1])*Yjp1_Yj + (v[id]-v[id+xpointsp1])*Xip1_Xi) *
APu[id]*APv[id]*APu[id+1]*APv[id+xpointsp1] * XIP1_XI*XI_XIM1*YJP1_YJ*YJ_YJM1;
                }
            }

            for (i=0; i<=xpointsp1-1; i++) {
                APp[i]=1;
                AEp[i]=0;
                AWp[i]=0;
                ANp[i]=0;
                ASp[i]=0;
                Bp[i]=0;
            }

            for (i=m-xpointsp1; i<=m-1; i++) {
                APp[i]=1;
                AEp[i]=0;
                AWp[i]=0;
                ANp[i]=0;
                ASp[i]=0;
                Bp[i]=0;
            }

            for (i=xpointsp1; i<=m-2*xpointsp1; i=i+xpointsp1) {
                APp[i]=1;
                AEp[i]=0;
                AWp[i]=0;
                ANp[i]=0;
                ASp[i]=0;
                Bp[i]=0;
            }

            for (i=2*xpointsp1-1; i<=m-1-xpointsp1; i=i+xpointsp1) {
                APp[i]=1;
                AEp[i]=0;
                AWp[i]=0;
                ANp[i]=0;
                ASp[i]=0;
                Bp[i]=0;
            }

            for (i=0; i<=m-1; i++) {
                APp[i] = -APp[i];
            }

            diag5solver (xpointsp1, ypointsp1, APp, AEp, AWp, ANp, ASp, Bp, pp);

            for (i=0; i<=m-1; i++) {
                p[i] = p[i] + urf_p*pp[i];
            }

            for (i=xpointsp1; i<=m-2*xpointsp1; i=i+xpointsp1) {
                p[i] = p[i+1];
            }

            for (i=2*xpointsp1-1; i<=m-1-xpointsp1; i=i+xpointsp1) {
                p[i] = p[i-1];
            }

            for (i=0; i<=xpointsp1-1; i++) {
                p[i] = p[i+xpointsp1];
            }

            for (i=m-xpointsp1; i<=m-1; i++) {
                p[i] = p[i-xpointsp1];
            }
    } //k iteration
```

```
    B: SetConsoleTextAttribute(hConsole, 2);
    printf("\nPrinting u values...\n");
    SetConsoleTextAttribute(hConsole, 7);
    for (j=ypointsp1-1;j>=0;j--) {
        for (i=0;i<=xpointsp1-1;i++) {
            fprintf(outputu, "%9.6f ",(float) (u[j*xpointsp1+i]));
        }
        fprintf(outputu, "\n");
    }

    SetConsoleTextAttribute(hConsole, 1);
    printf("\nPrinting v values...\n");
    SetConsoleTextAttribute(hConsole, 7);
    for (j=ypointsp1-1;j>=0;j--) {
        for (i=0;i<=xpointsp1-1;i++) {
            fprintf(outputv, "%9.6f ",(float) (v[j*xpointsp1+i]));
        }
        fprintf(outputv, "\n");
    }

    SetConsoleTextAttribute(hConsole, 4);
    printf("\nPrinting P values...\n");
    SetConsoleTextAttribute(hConsole, 7);
    for (j=ypointsp1-1;j>=0;j--) {
        for (i=0;i<=xpointsp1-1;i++) {
            fprintf(outputp, "%9.6f ",(float) (p[j*xpointsp1+i]));
        }
        fprintf(outputp, "\n");
    }

    return 0;
} //END
```

# ΠΑΡΑΡΤΗΜΑ 3 – Ενδεικτική παρουσίαση παράλληλης υλοποίησης

Στο παρόν παράρτημα γίνεται μια παρουσίαση του παράλληλου προγράμματος υπολογισμού των συντελεστών. Το πρόγραμμα αυτό έχει τροποποιηθεί αρκετές φορές στο πλαίσιο των ελέγχων των αποτελεσμάτων για διαφορετικές αρχιτεκτονικές, υποστηρίξεις του προτύπου και ελέγχοντας διαφορετικά τμήματα του προγράμματος, και δεν κρίνεται σκόπιμο να παρουσιάζεται κάθε εκδοχή του. Το παράρτημα έχει σκοπό να παρουσιαστούν οι βασικές δομές που επιτρέπουν την παράλληλη εκτέλεση μέσω του OpenCL.

Αρχικά ορίζονται οι OpenCL-specific δομές:

```
cl_device_id CPU, GPU, device;
cl_context context;
cl_command_queue queue;
cl_kernel CALC;
cl_program program;
```

και οι:

```
cl_platform_id **platform_array=NULL;
cl_device_id ***device_array=NULL;
cl_uint number_of_plats=0, **devs_per_plat=NULL;
```

οι οποίες θα γεμίσουν από τη συνάρτηση `OpenCLScout()`.

Τα `cl_mem` objects είναι ο τρόπος που μεταφέρει δεδομένα η OpenCL μεταξύ host & συσκευών. Επιλέγεται μια ενδεικτική συσκευή (πχ CPU) και γίνεται compiled το πρόγραμμα το οποίο περιέχεται σε ξεχωριστό αρχείο (“kernels.cl”). Εάν κάτι στη μεταγλώττιση αποτύχει, τυπώνεται το log του compiler.

Το πρόγραμμα έπειτα προχωρά στη δέσμευση της απαραίτητης μνήμης και κάνει τις απαραίτητες αρχικοποιήσεις δεδομένων. Στο παρόν πρόγραμμα επιλέγεται η διαδικασία της δημιουργίας buffer και κατόπιν η μεταφορά του στη συσκευή μέσω της `clEnqueueWriteBuffer()`, έναντι πιο σύντομων λύσεων ώστε να έχουμε τη δυνατότητα να χρονομετρήσουμε και τη μεταφορά δεδομένων (η συνάρτηση `clEnqueueWriteBuffer()` μπορεί να παράγει και δομές `cl_event` επιτρέποντας την χρονομέτρηση εάν το θελήσουμε).

Έπειτα αφού πρώτα δημιουργηθεί η δομή `CALC` από το αρχείο (“kernels.cl”), περνώνται τα arguments με τη σειρά που βρίσκονται και στο εν λόγω αρχείο (στο συγκεκριμένο kernel) και εκτελείται ο kernel με

```
global_work_offset[0] = 1;
global_work_offset[1] = 1;
global_work_size[0] = xpointsp1-2;
global_work_size[1] = ypointsp1-2;
```

λόγω του μη υπολογισμού των ακραίων συντελεστών. Οι υπολογισμένοι συντελεστές διαβάζονται τέλος πίσω στον host. Στο παρόν αρχείο παρουσιάζεται και η συνάρτηση `OpenCLScout()`, καθώς και μια ενδεικτική υλοποίηση επαναληπτικής μεθόδου επίλυσης των γραμμικών συστημάτων που μπορεί να χρησιμοποιηθεί σε περίπτωση ικανοποίησης της διαγώνιας υπεροχής για πιο γρήγορη επίλυση των γραμμικών συστημάτων.

Τέλος ακολουθεί το αρχείο kernels.cl που περιέχει τη συνάρτηση `CALC` που είναι ο kernel που περιγράφει ακριβώς τις μαθηματικές πράξεις για την παραγωγή των συντελεστών.

```
#define PROG_FILE "kernels.cl"
#define COMPILE_BUILD_OPTIONS ""

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

#if defined(_WIN32)
#include <windows.h>
#endif

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

typedef float datatype;
/* datatype can be (based on required precision): float,  double,  __float128 */

void OpenCLScout (cl_uint* number_of_plats, cl_platform_id** platform_array, cl_uint** devs_per_plat,
cl_device_id*** device_array, cl_device_id* sample_CPU, cl_device_id* sample_GPU);
void iterative_solver (unsigned int xpoints, unsigned int ypoints, datatype *AP, datatype *AE, datatype *AW,
datatype *AN, datatype *AS, datatype *b, datatype *x, unsigned int max_iterations);
const char* clewErrorString(cl_int error);


int main() {

    //  ---------------------------------------
    //  Prerequisites
    //  ---------------------------------------

    #if defined(_WIN32)
    HANDLE  hConsole;                           //For colors
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //For colors
    #endif

    cl_device_id CPU, GPU, device;
    cl_context context;
    cl_command_queue queue;
    cl_kernel CALC;
    cl_program program;
    size_t global_work_size[]={0,0};    //Initialized & used before EVERY kernel
    size_t global_work_offset[]={0,0};  //Initialized & used before EVERY kernel
    size_t local_work_size[]={0,0};    //Initialized & used before EVERY kernel
    const cl_queue_properties queue_properties[] = {CL_QUEUE_PROPERTIES, CL_QUEUE_PROFILING_ENABLE, 0};
    cl_event ev_CALC;

    cl_platform_id **platform_array=NULL;
    cl_device_id ***device_array=NULL;
    cl_uint number_of_plats=0, **devs_per_plat=NULL;

    FILE* outputu = fopen ("u.txt", "w+");
    FILE* outputv = fopen ("v.txt", "w+");

    FILE *program_handle;
    char **program_buffer;
    size_t program_size;
    char* program_log;
    unsigned int log_size;

    cl_int i,j,k, status, err=-1;

    datatype *u, *v, *p, *pp, *X, *Y;
    cl_mem uobj, vobj, pobj, Xobj, Yobj;

    datatype *temp_u, *temp_v;

    datatype *APu, *AEu, *AWu, *ANu, *ASu, *Bu;
    datatype *APv, *AEv, *AWv, *ANv, *ASv, *Bv;

    cl_mem APuobj, AEuobj, AWuobj, ANuobj, ASuobj, Buobj;
    cl_mem APvobj, AEvobj, AWvobj, ANvobj, ASvobj, Bvobj;

    cl_uint xpoints=0, ypoints=0, xpointsp1=0, ypointsp1=0, n=0, m=0;       //Numbers will be set later...
    datatype urf_u=0.5, urf_v=0.5, urf_p=0.5;
    datatype visc=0.003, dens=1.0, velin=30.0;

    //  ---------------------------------------
    //  Get Platforms & devices. Create Context & Command queue
    //  ---------------------------------------
```

```
platform_array = (cl_platform_id**) malloc(sizeof(cl_platform_id**));
device_array = (cl_device_id***) malloc(sizeof(cl_device_id***));
devs_per_plat = (cl_uint**) malloc(sizeof(cl_uint**));

OpenCLScout(&number_of_plats, platform_array, devs_per_plat, device_array, &CPU, &GPU);

device = CPU;
context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
    if (err!=0) {printf("clCreateContext error: %s\n", clewErrorString(err)); return 1;}
queue = clCreateCommandQueueWithProperties(context, device, queue_properties, &err);
    if (err!=0) {printf("clCreateCommandQueueWithProperties error: %s\n", clewErrorString(err)); return 1;}

//  ----------------------------------------
//  Create program
//  ----------------------------------------

program_handle = fopen(PROG_FILE, "r");
    if(program_handle == NULL) {printf("Can't find the kernels' file"); return 1;}

fseek(program_handle, 0, SEEK_END);
program_size = ftell(program_handle);
rewind(program_handle);

program_buffer = (char**) malloc(sizeof(char*));
*program_buffer = (char*) malloc(sizeof(char)*(program_size+1));
program_buffer[0][program_size] = '\0';
fread(*program_buffer, sizeof(char), program_size, program_handle);
fclose(program_handle);

program = clCreateProgramWithSource(context, 1, (const char**) program_buffer, &program_size, &err);
    if (err!=0) {printf("clCreateProgramWithSource error: %s\n", clewErrorString(err)); return 1;}

err = clBuildProgram(program, 1, &device, COMPILE_BUILD_OPTIONS, NULL, NULL);

//  ----------------------------------------
//  In case of program error...
//  ----------------------------------------

if (err!=0) {
        printf("clBuildProgram error, getting build log...\n");
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0, NULL, &log_size);
        program_log = (char*)malloc(log_size+1);
        program_log[log_size] = '\0';
        clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, log_size+1, program_log, NULL);
        printf("%s\n", program_log);
        free(program_log);
        return 1;
}

free(program_buffer[0]);
free(program_buffer);

//  ----------------------------------------
//  Memory allocation
//  ----------------------------------------

A:
xpoints=10000;
ypoints=1000;

xpointsp1=xpoints+1;
ypointsp1=ypoints+1;

n = xpoints*ypoints;
m = xpointsp1*ypointsp1;

u = malloc(m*sizeof(*u));
v = malloc(m*sizeof(*v));
p = malloc(m*sizeof(*p));
    if ((u==NULL)||(v==NULL)||(p==NULL)) {printf("Memory allocation failed"); return 1;}

temp_u = malloc(m*sizeof(*temp_u));
temp_v = malloc(m*sizeof(*temp_v));
    if ((temp_u==NULL)||(temp_u==NULL)) {printf("Memory allocation failed"); return 1;}

X = malloc((xpoints+2)*sizeof(*X));
Y = malloc((ypoints+2)*sizeof(*Y));
    if ((X==NULL)||(Y==NULL)) {printf("Memory allocation failed"); return 1;}

APu = malloc(m*sizeof(*APu));
AEu = malloc(m*sizeof(*AEu));
AWu = malloc(m*sizeof(*AWu));
ANu = malloc(m*sizeof(*ANu));
ASu = malloc(m*sizeof(*ASu));
Bu = malloc(m*sizeof(*Bu));
    if ((APu==NULL)||(AEu==NULL)||(AWu==NULL)||(ANu==NULL)||(ASu==NULL)||(Bu==NULL)) {printf("Memory
```

```
allocation failed"); return 1;}

    APv = malloc(m*sizeof(*APv));
    AEv = malloc(m*sizeof(*AEv));
    AWv = malloc(m*sizeof(*AWv));
    ANv = malloc(m*sizeof(*ANv));
    ASv = malloc(m*sizeof(*ASv));
    Bv = malloc(m*sizeof(*Bv));
        if ((APv==NULL)||(AEv==NULL)||(AWv==NULL)||(ANv==NULL)||(ASv==NULL)||(Bv==NULL)) {printf("Memory
allocation failed"); return 1;}

    //  ----------------------------------------
    //  Data initialization
    //  ----------------------------------------

    for (i=0;i<=m-1;i++) {
        u[i]=5;
        v[i]=1;
        p[i]=1;
        //pp[i]=0;
    }

    for (i=1;i<=xpoints;i++) {
        X[i]=(float)i-1;
    }

    for (i=1;i<=ypoints;i++) {
        Y[i]=(float)i-1;
    }

    X[0]=X[1];
    X[xpoints+1]=X[xpoints];
    Y[0]=Y[1];
    Y[ypoints+1]=Y[ypoints];

    Xobj = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR | CL_MEM_ALLOC_HOST_PTR,
(xpoints+2)*sizeof(*X), X, &err);
    Yobj = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR | CL_MEM_ALLOC_HOST_PTR,
(ypoints+2)*sizeof(*Y), Y, &err);
        if (err!=0) {printf("clCreateBuffer error (x,y): %s\n", clewErrorString(err)); return 1;}

    uobj = clCreateBuffer(context, CL_MEM_READ_ONLY, m*sizeof(*u), NULL, &err);
    vobj = clCreateBuffer(context, CL_MEM_READ_ONLY, m*sizeof(*v), NULL, &err);
    pobj = clCreateBuffer(context, CL_MEM_READ_ONLY, m*sizeof(*p), NULL, &err);
        if (err!=0) {printf("clCreateBuffer error: %s\n", clewErrorString(err)); return 1;}

    err += clEnqueueWriteBuffer(queue, uobj, CL_TRUE, 0, m*sizeof(*u), u, 0, NULL, NULL);
    err += clEnqueueWriteBuffer(queue, vobj, CL_TRUE, 0, m*sizeof(*v), v, 0, NULL, NULL);
    err += clEnqueueWriteBuffer(queue, pobj, CL_TRUE, 0, m*sizeof(*p), p, 0, NULL, NULL);
        if (err!=0) {printf("clEnqueueWriteBuffer error: %s\n", clewErrorString(err)); return 1;}

    APuobj = clCreateBuffer(context, CL_MEM_READ_WRITE, m*sizeof(*APu), NULL, &err);
    AEuobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*AEu), NULL, &err);
    AWuobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*AWu), NULL, &err);
    ANuobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*ANu), NULL, &err);
    ASuobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*ASu), NULL, &err);
    Buobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*Bu), NULL, &err);
        if (err!=0) {printf("clCreateBuffer error (u): %s\n", clewErrorString(err)); return 1;}

    APvobj = clCreateBuffer(context, CL_MEM_READ_WRITE, m*sizeof(*APv), NULL, &err);
    AEvobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*AEv), NULL, &err);
    AWvobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*AWv), NULL, &err);
    ANvobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*ANv), NULL, &err);
    ASvobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*ASv), NULL, &err);
    Bvobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, m*sizeof(*Bv), NULL, &err);
        if (err!=0) {printf("clCreateBuffer error (v): %s\n", clewErrorString(err)); return 1;}

    CALC = clCreateKernel(program, "CALC", &err);
        if (err!=0) {printf("clCreateKernel error: %s\n", clewErrorString(err)); return 1;}

    for (k=0;k<=25;k++) {        //SIMPLE iterations
            printf("SIMPLE iteration %d\n",k);

            //  ------------------------
            //          CALC
            //  ------------------------

            global_work_offset[0] = 1;
            global_work_offset[1] = 1;

            global_work_size[0] = xpointsp1-2;
            global_work_size[1] = ypointsp1-2;

            err += clSetKernelArg(CALC,  0, sizeof(cl_mem), (void*) &uobj);
            err += clSetKernelArg(CALC,  1, sizeof(cl_mem), (void*) &vobj);
            err += clSetKernelArg(CALC,  2, sizeof(cl_mem), (void*) &pobj);
```

```
            err += clSetKernelArg(CALC,  3, sizeof(cl_mem), (void*) &Xobj);
            err += clSetKernelArg(CALC,  4, sizeof(cl_mem), (void*) &Yobj);

            err += clSetKernelArg(CALC,  5, sizeof(cl_mem), (void*) &APuobj);
            err += clSetKernelArg(CALC,  6, sizeof(cl_mem), (void*) &AEuobj);
            err += clSetKernelArg(CALC,  7, sizeof(cl_mem), (void*) &AWuobj);
            err += clSetKernelArg(CALC,  8, sizeof(cl_mem), (void*) &ANuobj);
            err += clSetKernelArg(CALC,  9, sizeof(cl_mem), (void*) &ASuobj);
            err += clSetKernelArg(CALC, 10, sizeof(cl_mem), (void*) &Buobj);

            err += clSetKernelArg(CALC, 11, sizeof(cl_mem), (void*) &APvobj);
            err += clSetKernelArg(CALC, 12, sizeof(cl_mem), (void*) &AEvobj);
            err += clSetKernelArg(CALC, 13, sizeof(cl_mem), (void*) &AWvobj);
            err += clSetKernelArg(CALC, 14, sizeof(cl_mem), (void*) &ANvobj);
            err += clSetKernelArg(CALC, 15, sizeof(cl_mem), (void*) &ASvobj);
            err += clSetKernelArg(CALC, 16, sizeof(cl_mem), (void*) &Bvobj);

            err += clSetKernelArg(CALC, 17, sizeof(cl_int), (void*) &xpointsp1);
            err += clSetKernelArg(CALC, 18, sizeof(cl_float), (void*) &visc);
            err += clSetKernelArg(CALC, 19, sizeof(cl_float), (void*) &dens);
                if (err!=0) {printf("clSetKernelArg error: %s\n", clewErrorString(err)); return 1;}

            err = clEnqueueNDRangeKernel(queue, CALC, 2, global_work_offset, global_work_size, NULL, 0, NULL,
&ev_CALC);
                if (err!=0) {printf("clEnqueueNDRangeKernel error: %s\n", clewErrorString(err)); return 1;}

            //  -----------------------------------------
            //  Reading data back...
            //  -----------------------------------------

            err += clEnqueueReadBuffer(queue, APuobj, CL_TRUE, 0, m*sizeof(*APu), APu, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, AEuobj, CL_TRUE, 0, m*sizeof(*AEu), AEu, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, AWuobj, CL_TRUE, 0, m*sizeof(*AWu), AWu, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, ANuobj, CL_TRUE, 0, m*sizeof(*ANu), ANu, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, ASuobj, CL_TRUE, 0, m*sizeof(*ASu), ASu, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, Buobj, CL_TRUE, 0, m*sizeof(*Bu), Bu, 1, &ev_CALC, NULL);

            err += clEnqueueReadBuffer(queue, APvobj, CL_TRUE, 0, m*sizeof(*APv), APv, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, AEvobj, CL_TRUE, 0, m*sizeof(*AEv), AEv, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, AWvobj, CL_TRUE, 0, m*sizeof(*AWv), AWv, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, ANvobj, CL_TRUE, 0, m*sizeof(*ANv), ANv, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, ASvobj, CL_TRUE, 0, m*sizeof(*ASv), ASv, 1, &ev_CALC, NULL);
            err += clEnqueueReadBuffer(queue, Bvobj, CL_TRUE, 0, m*sizeof(*Bv), Bv, 1, &ev_CALC, NULL);

                if (err!=0) {printf("clEnqueueReadBuffer error: %s\n", clewErrorString(err)); return 1;}


    }   //SIMPLE Iterations

    //  -----------------------------------------
    //  Finishing...
    //  -----------------------------------------

    err = clFlush(queue);
    err = clFinish(queue);
    err = clReleaseCommandQueue(queue);
    err = clReleaseProgram(program);
    err = clReleaseContext(context);

    return 0;


}


void OpenCLScout(cl_uint* number_of_plats, cl_platform_id** platform_array, cl_uint** devs_per_plat,
cl_device_id*** device_array, cl_device_id* sample_CPU, cl_device_id* sample_GPU) {

    cl_uint uintinfo;
    cl_ulong mem_size;
    char* info;
    size_t sz;
    cl_device_fp_config flag;
    cl_int err=-1, bvalue, i,j,k;

    err = clGetPlatformIDs(0, NULL, number_of_plats);
if (err!=0) {printf("clGetPlatformIDs error: %s\n", clewErrorString(err)); exit(1);}
    printf("Number of platforms identified: %d\n\n", *number_of_plats);
    *platform_array = (cl_platform_id*) malloc(sizeof(cl_platform_id)*(*number_of_plats));
    err = clGetPlatformIDs(*number_of_plats, *platform_array, NULL);
    *device_array = (cl_device_id**) malloc(*number_of_plats*sizeof(cl_device_id*));
    *devs_per_plat = (cl_uint*) malloc(*number_of_plats*sizeof(cl_uint));


    for (i=0;i<=*number_of_plats-1;i++) {
        printf("Platform no %d:\n",i);
```

```
        printf("--------------------------------------------------------------------------------\n");
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_NAME, 0, NULL, &sz);
        info = (char*) malloc(sz);
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_NAME, sz, info, NULL);
        printf("Name:                       %s\n",info, sz);
        free(info);

        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_VENDOR, 0, NULL, &sz);
        info = (char*) malloc(sz);
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_VENDOR, sz, info, NULL);
        printf("Vendor:                     %s\n",info, sz);
        free(info);

        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_VERSION, 0, NULL, &sz);
        info = (char*) malloc(sz);
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_VERSION, sz, info, NULL);
        printf("OpenCL version:             %s\n",info, sz);
        free(info);

        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_PROFILE, 0, NULL, &sz);
        info = (char*) malloc(sz);
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_PROFILE, sz, info, NULL);
        printf("Profile:                    %s\n",info, sz);
        free(info);

        err = clGetDeviceIDs((*platform_array)[i],  CL_DEVICE_TYPE_ALL, 0, NULL, &uintinfo);
if (err!=0) {printf("clGetDeviceIDs error: %s\n", clewErrorString(err)); exit(1);}
        printf("Number of devices:          %u\n", uintinfo);
        (*device_array)[i]=(cl_device_id*) malloc(sizeof(cl_device_id)*uintinfo);
        err = clGetDeviceIDs((*platform_array)[i],  CL_DEVICE_TYPE_ALL, uintinfo, (*device_array)[i], NULL);
        (*devs_per_plat)[i]=uintinfo;

        err = clGetDeviceIDs((*platform_array)[i],  CL_DEVICE_TYPE_GPU, 0, NULL, &uintinfo);
        printf("Number of GPUs:             %u\n", uintinfo);

        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_EXTENSIONS, 0, NULL, &sz);
        info = (char*) malloc(sz);
        err = clGetPlatformInfo((*platform_array)[i], CL_PLATFORM_EXTENSIONS, sz, info, NULL);
        printf("Supported extensions:       ");
        for (j=0;j<sz;j++) {
            if (info[j+1]=='\0') {printf("\n");break;}      //SPACE CHARACTER EXISTS BEFORE NULL CHARACTER
            if (info[j]==' ') {
                printf("\n                            ");
                continue;
            }
            printf("%c", info[j]);
        }
        free(info);

        for (j=0;j<=(*devs_per_plat)[i]-1;j++) {
            printf("Platform %d/Device %d:\n",i, j);
            printf("--------------------------------------------------------------------------------\n");

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_NAME, 0, NULL, &sz);                         if
(err!=0) {printf("clGetDeviceInfo error: %s\n", clewErrorString(err)); exit(1);}
            info = (char*) malloc(sz);
            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_NAME, sz, info, NULL);
            printf("Device name:                  %s\n", info);
            free(info);

            err = clGetDeviceInfo((*device_array)[i][j],  CL_DEVICE_ENDIAN_LITTLE, 4, &bvalue, NULL);
            if (bvalue==1) {printf("Device endianness:            Little Endian\n");}
            else if (bvalue==0) {printf("Device endianness:            Big Endian\n");}

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_MAX_CLOCK_FREQUENCY, 4, &uintinfo, NULL);
            printf("Clock Frequency:              %uMHz\n", uintinfo);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_VERSION, 0, NULL, &sz);
            info = (char*) malloc(sz);
            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_VERSION, sz, info, NULL);
            printf("Device OpenCL Version:        %s\n", info);
            free(info);

            err = clGetDeviceInfo((*device_array)[i][j],  CL_DEVICE_HOST_UNIFIED_MEMORY, 4, &bvalue, NULL);
            if (bvalue==1) {printf("Unified Memory space w/ host: Yes\n");}
            else if (bvalue==0) {printf("Unified Memory space w/ host: No\n");}


            err = clGetDeviceInfo((*device_array)[i][j], CL_DRIVER_VERSION, 0, NULL, &sz);
            info = (char*) malloc(sz);
            err = clGetDeviceInfo((*device_array)[i][j], CL_DRIVER_VERSION, sz, info, NULL);
            printf("Driver Version:               %s\n", info);
            free(info);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_GLOBAL_MEM_SIZE, sizeof(mem_size), &mem_size,
NULL);
```

```
            printf("Global Memory Size:           %uMB\n", mem_size/1024/1024);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_GLOBAL_MEM_CACHE_SIZE, sizeof(mem_size),
&mem_size, NULL);
            printf("Global Memory Cache Size:     %uKB\n", mem_size/1024);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_LOCAL_MEM_SIZE, sizeof(mem_size), &mem_size,
NULL);
            printf("Local Memory Size:            %uKB\n", mem_size/1024);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE, sizeof(mem_size),
&mem_size, NULL);
            printf("Max Constant Buffer Size:     %uKB\n", mem_size/1024);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_ADDRESS_BITS, sizeof(uintinfo), &uintinfo,
NULL);
            printf("Address Width:                %ubits\n", uintinfo);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS, sizeof(uintinfo),
&uintinfo, NULL);
            printf("Max work item dimensions:     %u\n", uintinfo);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_MAX_COMPUTE_UNITS, 4, &uintinfo, NULL);
            printf("Number of Compute Units:      %u\n", uintinfo);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_MAX_WORK_GROUP_SIZE, sizeof(sz), &sz, NULL);
            printf("Max Workgroup size:           %u work items\n", sz);

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PROFILING_TIMER_RESOLUTION, sizeof(sz), &sz,
NULL);
            if (sz<1000) {printf("Profiling timer resolution:   %uns\n\n", sz);}
            else {printf("Profiling timer resolution:   %3.3fus\n\n", (float)sz/1000);}

            printf("Preferred vector width...");
            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin chars:                   %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin ints:                    %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin doubles:                 %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin halfs:                   %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin longs:                   %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin shorts:                  %u", uintinfo);

            clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT, sizeof(uintinfo),
&uintinfo, NULL);
            printf("\nin floats:                  %u", uintinfo);

            printf("\n\n");

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_SINGLE_FP_CONFIG, sizeof(flag), &flag, NULL);
            printf("Float Processing Features:  ");
            if(flag & CL_FP_INF_NAN) {printf("INF and NaN values supported.\n                          ");}
            if(flag & CL_FP_DENORM) {printf("Denormalized numbers supported.\n                         ");}
            if(flag & CL_FP_ROUND_TO_NEAREST) {printf("Round To Nearest Even mode supported.\n
");}
            if(flag & CL_FP_ROUND_TO_INF) {printf("Round To Infinity mode supported.\n
");}
            if(flag & CL_FP_ROUND_TO_ZERO) {printf("Round To Zero mode supported.\n
");}
            if(flag & CL_FP_FMA) {printf("Floating-point multiply-and-add operation supported.\n");}

            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_EXTENSIONS, 0, NULL, &sz);
            info = (char*) malloc(sz);
            err = clGetDeviceInfo((*device_array)[i][j], CL_DEVICE_EXTENSIONS, sz, info, NULL);
            printf("\nSupported extensions:       ");
            for (k=0;k<sz;k++) {
                if (info[k+1]=='\0') {printf("\n");break;}      //SPACE CHARACTER EXISTS BEFORE NULL CHARACTER
                if (info[k]==' ') {
                    printf("\n                            ");
                    continue;
                }
```

```
            printf("%c", info[k]);
            }
            free(info);
        }
    }

    for (i=0;i<=*number_of_plats-1;i++) {
        err = clGetDeviceIDs((*platform_array)[i], CL_DEVICE_TYPE_CPU, 1, sample_CPU, NULL);
        if (err!=0 && i==*number_of_plats-1) {printf("Couldn't find a sample CPU...\n\n"); break;}
        if (err!=0) {continue;}
        printf("Got a sample CPU!\n\n");
        break;
    }
    for (i=0;i<=*number_of_plats-1;i++) {
        err = clGetDeviceIDs((*platform_array)[i], CL_DEVICE_TYPE_GPU, 1, sample_GPU, NULL);
        if (err!=0 && i==*number_of_plats-1) {printf("Couldn't find a sample GPU...\n\n"); break;}
        if (err!=0) {continue;}
        printf("Got a sample GPU!\n\n");
        break;
    }
}

void iterative_solver (unsigned int xpoints, unsigned int ypoints, datatype *AP, datatype *AE, datatype *AW,
datatype *AN, datatype *AS, datatype *b, datatype *x, unsigned int max_iterations) {

    //Function only modifies the x vector argument. No other argument is modified.

    int i,j,k;
    datatype *bigx, *residual;
    datatype total_res=0;
    unsigned int n;

    n = xpoints*ypoints;

    // -------------------------------
    // Ensuring convergence (diagonal dominance) before any actual computation or allocation
    // -------------------------------

    for (i=0;i<=n-1;i++) {
        if (fabs(AP[i]) <= fabs(AS[i]) + fabs(AW[i]) + fabs(AE[i]) + fabs(AN[i])) {
            printf("\nConvergence for this system using Gauss-Seidel is not guaranteed:\n");
            printf("AP[%d]=%4.3f, AS[%d]=%4.3f, AW[%d]=%4.3f, AE[%d]=%4.3f, AN[%d]=%4.3f\n", i, (float)AP[i], i,
(float)AS[i], i, (float)AW[i], i, (float)AE[i], i, (float)AN[i]);
            printf("Function will exit right away...\n");
            exit(1);
        }
    }

    // -------------------------------
    // Solve iteratively using Gauss-Seidel
    // -------------------------------

    bigx = calloc(n+2*xpoints, sizeof(*bigx));
    residual = malloc(n*sizeof(*residual));
    if (bigx==NULL||residual==NULL) {
        HANDLE  hConsole=GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(hConsole, 12);
        printf("Iterative Solver: Error allocating memory...");
        SetConsoleTextAttribute(hConsole, 7);
        exit(1);
    }

    //Not necessary if initial vector x[i]=0 (i=0,1,...,n-1)
    for (i=0;i<=n-1;i++) {
        bigx[xpoints+i] = x[i];
    }

    for (k=1;k<=max_iterations;k++) {
        if(k%10==0) {
            total_res=0;
            for (i=0;i<=n-1;i++) {
                residual[i] = bigx[xpoints+i];
                bigx[xpoints+i] = (-b[i] - AS[i]*bigx[i] - AW[i]*bigx[xpoints+i-1] - AE[i]*bigx[xpoints+i+1] -
AN[i]*bigx[i+2*xpoints]) / AP[i];
                total_res += fabs(residual[i] - bigx[xpoints+i]);
            }
            if (total_res<=0.000001*n) {/*printf(" Total residual=%f ", total_res);*/ goto A;}
            else {k++; continue;}
        }
        for (i=0;i<=n-1;i++) {
            bigx[xpoints+i] = (-b[i] - AS[i]*bigx[i] - AW[i]*bigx[xpoints+i-1] - AE[i]*bigx[xpoints+i+1] -
AN[i]*bigx[i+2*xpoints]) / AP[i];
        }
    }

    if (k>=max_iterations) {
```

```
        HANDLE  hConsole=GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(hConsole, 12);
        printf("Iterative Solver did not converge...\n");
        SetConsoleTextAttribute(hConsole, 7);
        exit(1);
    }

A:  for (i=0;i<=n-1;i++) {
        x[i] = bigx[xpoints+i];
    }
    printf(" (Iterative solver returned after %4d iterations) ", k);
    return;

}

const char* clewErrorString(cl_int error)  {
    static const char* strings[] =
    {
        // Error Codes
          "CL_SUCCESS"                                  //   0
        , "CL_DEVICE_NOT_FOUND"                         //  -1
        , "CL_DEVICE_NOT_AVAILABLE"                     //  -2
        , "CL_COMPILER_NOT_AVAILABLE"                   //  -3
        , "CL_MEM_OBJECT_ALLOCATION_FAILURE"            //  -4
        , "CL_OUT_OF_RESOURCES"                         //  -5
        , "CL_OUT_OF_HOST_MEMORY"                       //  -6
        , "CL_PROFILING_INFO_NOT_AVAILABLE"             //  -7
        , "CL_MEM_COPY_OVERLAP"                         //  -8
        , "CL_IMAGE_FORMAT_MISMATCH"                    //  -9
        , "CL_IMAGE_FORMAT_NOT_SUPPORTED"               //  -10
        , "CL_BUILD_PROGRAM_FAILURE"                    //  -11
        , "CL_MAP_FAILURE"                              //  -12

        , ""    //  -13
        , ""    //  -14
        , ""    //  -15
        , ""    //  -16
        , ""    //  -17
        , ""    //  -18
        , ""    //  -19

        , ""    //  -20
        , ""    //  -21
        , ""    //  -22
        , ""    //  -23
        , ""    //  -24
        , ""    //  -25
        , ""    //  -26
        , ""    //  -27
        , ""    //  -28
        , ""    //  -29

        , "CL_INVALID_VALUE"                            //  -30
        , "CL_INVALID_DEVICE_TYPE"                      //  -31
        , "CL_INVALID_PLATFORM"                         //  -32
        , "CL_INVALID_DEVICE"                           //  -33
        , "CL_INVALID_CONTEXT"                          //  -34
        , "CL_INVALID_QUEUE_PROPERTIES"                 //  -35
        , "CL_INVALID_COMMAND_QUEUE"                    //  -36
        , "CL_INVALID_HOST_PTR"                         //  -37
        , "CL_INVALID_MEM_OBJECT"                       //  -38
        , "CL_INVALID_IMAGE_FORMAT_DESCRIPTOR"          //  -39
        , "CL_INVALID_IMAGE_SIZE"                       //  -40
        , "CL_INVALID_SAMPLER"                          //  -41
        , "CL_INVALID_BINARY"                           //  -42
        , "CL_INVALID_BUILD_OPTIONS"                    //  -43
        , "CL_INVALID_PROGRAM"                          //  -44
        , "CL_INVALID_PROGRAM_EXECUTABLE"               //  -45
        , "CL_INVALID_KERNEL_NAME"                      //  -46
        , "CL_INVALID_KERNEL_DEFINITION"                //  -47
        , "CL_INVALID_KERNEL"                           //  -48
        , "CL_INVALID_ARG_INDEX"                        //  -49
        , "CL_INVALID_ARG_VALUE"                        //  -50
        , "CL_INVALID_ARG_SIZE"                         //  -51
        , "CL_INVALID_KERNEL_ARGS"                      //  -52
        , "CL_INVALID_WORK_DIMENSION"                   //  -53
        , "CL_INVALID_WORK_GROUP_SIZE"                  //  -54
        , "CL_INVALID_WORK_ITEM_SIZE"                   //  -55
        , "CL_INVALID_GLOBAL_OFFSET"                    //  -56
        , "CL_INVALID_EVENT_WAIT_LIST"                  //  -57
        , "CL_INVALID_EVENT"                            //  -58
        , "CL_INVALID_OPERATION"                        //  -59
        , "CL_INVALID_GL_OBJECT"                        //  -60
        , "CL_INVALID_BUFFER_SIZE"                      //  -61
        , "CL_INVALID_MIP_LEVEL"                        //  -62
        , "CL_INVALID_GLOBAL_WORK_SIZE"                 //  -63
```

```
        , "CL_UNKNOWN_ERROR_CODE"
    };

    if (error >= -63 && error <= 0)
        return strings[-error];
    else
        return strings[64];
}
```

```
__kernel void CALC (__global float* u,
                    __global float* v,
                    __global float* p,
                    __global float* X,
                    __global float* Y,

                    __global float* APu,
                    __global float* AEu,
                    __global float* AWu,
                    __global float* ANu,
                    __global float* ASu,
                    __global float* Bu,

                    __global float* APv,
                    __global float* AEv,
                    __global float* AWv,
                    __global float* ANv,
                    __global float* ASv,
                    __global float* Bv,
                    const unsigned int xpointsp1,
                    const float visc,
                    const float dens) {

    int id, idx=get_global_id(0), idy=get_global_id(1);  // idx & idy for 2D notation, id for accessing
converted 1d array arguments
    float Fe, Fw, Fn, Fs;
    float XI_XIM1, Yjp1_Yj, Xip1_Xi, Xi_Xim1, YJP1_YJ, YJ_YJM1, XIP1_XI, Yj_Yjm1;
    float term1, term2, term3, term4, term, temp;

    id=idy*xpointsp1+idx;

    XI_XIM1 = (X[idx]+X[idx+1])/2 - (X[idx-1]+X[idx])/2;
    Yjp1_Yj = Y[idy+1] - Y[idy];
    Xip1_Xi = X[idx+1] - X[idx];
    YJ_YJM1 = (Y[idy]+Y[idy+1])/2 - (Y[idy-1]+Y[idy])/2;

    Xi_Xim1 = X[idx] - X[idx-1];
    YJP1_YJ = (Y[idy+1]+Y[idy+2])/2 - (Y[idy]+Y[idy+1])/2;

    XIP1_XI = (X[idx+1]+X[idx+2])/2 - (X[idx]+X[idx+1])/2;
    Yj_Yjm1 = Y[idy]-Y[idy-1];

    //  ------------------------
    //          CALCU
    //  ------------------------

    Fe = dens*(u[id]+u[id+1])/2;
    Fw = dens*(u[id]+u[id-1])/2;
    Fn = dens*(v[id+xpointsp1]+v[id+xpointsp1-1])/2;
    Fs = dens*(v[id]+v[id-1])/2;

    term  = Xip1_Xi*Xi_Xim1*YJP1_YJ*YJ_YJM1;
    term1 = 2*visc*Yjp1_Yj *Xi_Xim1*YJP1_YJ*YJ_YJM1;
    term2 = 2*visc*Yjp1_Yj *Xip1_Xi*YJP1_YJ*YJ_YJM1;
    term3 =   visc*XI_XIM1 *Xip1_Xi*Xi_Xim1*YJ_YJM1;
    term4 =   visc*XI_XIM1 *Xip1_Xi*Xi_Xim1*YJP1_YJ;

    temp = (Fe-Fw)*Yjp1_Yj + (Fn-Fs)*XI_XIM1;

    APu[id] = -(((fmax(Fe,0)+fmax(-Fw,0))*Yjp1_Yj + (fmax(Fn,0)+fmax(-Fs,0))*XI_XIM1 - temp)*term +
term1+term2+term3+term4);

    AEu[id] = fmax(-Fe,0)*Yjp1_Yj*term + term1;
    AWu[id] = fmax(Fw,0) *Yjp1_Yj*term + term2;
    ANu[id] = fmax(-Fn,0)*XI_XIM1*term + term3;
    ASu[id] = fmax(Fs,0) *XI_XIM1*term + term4;

    Bu[id] = ((p[id-1]-p[id])*Yjp1_Yj + visc*(v[id+xpointsp1] - v[id-1+xpointsp1] - v[id] + v[id-1]))*term;

    //  ------------------------
    //          CALCV
    //  ------------------------

    Fe = dens*(u[id+1]+u[id+1-xpointsp1])/2;
    Fw = dens*(u[id]+u[id-xpointsp1])/2;
    Fn = dens*(v[id]+v[id+xpointsp1])/2;
    Fs = dens*(v[id]+v[id-xpointsp1])/2;

    term  = XIP1_XI*XI_XIM1*Yjp1_Yj*Yj_Yjm1;
    term1 =   visc*YJ_YJM1*XI_XIM1*Yjp1_Yj*Yj_Yjm1;
    term2 =   visc*YJ_YJM1*XIP1_XI*Yjp1_Yj*Yj_Yjm1;
    term3 = 2*visc*Xip1_Xi*XIP1_XI*XI_XIM1*Yj_Yjm1;
    term4 = 2*visc*Xip1_Xi*XIP1_XI*XI_XIM1*Yjp1_Yj;

    temp = (Fe-Fw)*YJ_YJM1 + (Fn-Fs)*Xip1_Xi;
```

```
    APv[id] = -(((fmax(Fe,0)+fmax(-Fw,0))*YJ_YJM1 + (fmax(Fn,0)+fmax(-Fs,0))*Xip1_Xi - temp)*term +
term1+term2+term3+term4);
    AEv[id] = fmax(-Fe,0)*YJ_YJM1*term + term1;
    AWv[id] = fmax(Fw,0) *YJ_YJM1*term + term2;
    ANv[id] = fmax(-Fn,0)*Xip1_Xi*term + term3;
    ASv[id] = fmax(Fs,0) *Xip1_Xi*term + term4;

    Bv[id] =((p[id-xpointsp1]-p[id])*Xip1_Xi + visc*(u[id+1] - u[id+1-xpointsp1] - u[id] + u[id-
xpointsp1]))*term;
```

# ΠΑΡΑΡΤΗΜΑ 4 - Παράλληλη επεξεργασία με χρήση διανυσμάτων

Στο παρόν παράρτημα παρουσιάζεται η συνάρτηση main() του προγράμματος υπολογισμού της παράστασης:

$$\Gamma\left(\left|100\cdot erf\left(\cos\left(1+e^{5\cdot\arctan\left(\left|\ln\left(a[i]\cdot b[i]\div c[i]^2\right)\right|\right)}\right)\right)\right|\right)$$

για μήκος πινάκων 65,000,000, καθώς επίσης και το αντίστοιχο kernels.cl αρχείο που περιέχει τα προγράμματα υπολογισμού με ή χωρίς τη χρήση διανυσμάτων.

Αρχικά γίνεται ο υπολογισμός της παράστασης σειριακά με χρήση των εγγενών συναρτήσεων της C και μάλιστα αυτών προοριζόμενων για μεταχείριση μεταβλητών μονής ακρίβειας (που πιθανόν θα ήταν ταχύτερες απ' ότι οι αντίστοιχες για μεταβλητές διπλής ακρίβειας). Τα αποτελέσματα αποθηκεύονται σε ξεχωριστό πίνακα 65,000,000 θέσεων, και έπειτα ξεκινά η επεξεργασία με χρήση OpenCL, αρχικά χωρίς να ορίζονται διανύσματα (kernel: VECTOR01) και μετά με χρήση διανυσμάτων.

Διάσπαρτα στο πρόγραμμα είναι σημεία μέτρησης του χρόνου μέσω της συνάρτησης `gettimeofday()`, ώστε τα αποτελέσματα να μπορούν να συγκριθούν μεταξύ τους. Ειδικά για τα events της OpenCL υπάρχουν και μετρήσεις που δίνει η OpenCL Runtime ξεχωριστά για μεταφορά δεδομένων και επεξεργασία.

Τέλος, τα αποτελέσματα συγκρίνονται για την ομοιότητά τους μεταξύ του 1% του αθροίσματος των απολύτων τιμών των αποτελεσμάτων. Πιο αυστηρά κριτήρια ενδεχομένως να αποτύγχαναν.

```
#define PROG_FILE "kernels.cl"
#define MAGNITUDE 65000000
#define COMPILE_BUILD_OPTIONS "-cl-opt-disable"//-cl-mad-enable"  //-Werror -cl-std=CL1.1 -DVECTOR_SIZE_128

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

#if defined(_WIN32)
#include <windows.h>
#endif

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

typedef float datatype;
/* datatype can be (based on required precision): float,  double,  __float128 */

void OpenCLScout (cl_uint* number_of_plats, cl_platform_id** platform_array, cl_uint** devs_per_plat,
cl_device_id*** device_array, cl_device_id* sample_CPU, cl_device_id* sample_GPU);
const char* clewErrorString(cl_int error);
void set_colour (char* colour);


int main() {

    //  ----------------------------------------
    //  Prerequisites
    //  ----------------------------------------

    HANDLE  hConsole;                           //For colors
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //For colors
    srand(time(NULL));

    cl_device_id CPU, GPU, device;
    cl_context context;
    cl_command_queue queue;
    cl_kernel CALC, *kernels;
    cl_program program;
    size_t sz, global_work_size[]={MAGNITUDE};    //Initialized & used before EVERY kernel
    const cl_queue_properties queue_properties[] = {CL_QUEUE_PROPERTIES, CL_QUEUE_PROFILING_ENABLE, 0};
    cl_event ev_CALC, ev_READ, ev_WRITE1, ev_WRITE2, ev_WRITE3;
    char* kname;

    cl_platform_id **platform_array=NULL;
    cl_device_id ***device_array=NULL;
    cl_uint number_of_plats=0, **devs_per_plat=NULL;
    cl_ulong cl_time_start, cl_time_end, cl_time_calc, cl_time_read, cl_time_write;

    FILE *program_handle;
    char **program_buffer;
    size_t program_size;
    char* program_log;
    unsigned int log_size;

    cl_int i,j,k, status, err=-1;
    cl_uint vec=1, run;

    datatype *a, *b, *c, *result, *result_serial, temp;
    cl_mem aobj, bobj, cobj, resultobj;

    long int microsecs, secs;
    struct timeval tval_before, tval_after, tval_result;
    float time, timesum;

    //  ----------------------------------------
    //  Get Platforms & devices. Create Context & Command queue
    //  ----------------------------------------

    platform_array = (cl_platform_id**) malloc(sizeof(cl_platform_id**));
    device_array = (cl_device_id***) malloc(sizeof(cl_device_id***));
    devs_per_plat = (cl_uint**) malloc(sizeof(cl_uint**));

    OpenCLScout(&number_of_plats, platform_array, devs_per_plat, device_array, &CPU, &GPU);

    device = CPU;
    context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
        if (err!=0) {printf("clCreateContext error: %s\n", clewErrorString(err)); return 1;}
    queue = clCreateCommandQueueWithProperties(context, device, queue_properties, &err);
        if (err!=0) {printf("clCreateCommandQueueWithProperties error: %s\n", clewErrorString(err)); return 1;}
```

```
    //  ----------------------------------------
    //  Create program
    //  ----------------------------------------

    program_handle = fopen(PROG_FILE, "r");
        if(program_handle == NULL) {printf("Can't find the kernels' file"); return 1;}

    fseek(program_handle, 0, SEEK_END);
    program_size = ftell(program_handle);
    rewind(program_handle);

    program_buffer = (char**) malloc(sizeof(char*));
    *program_buffer = (char*) malloc(sizeof(char)*(program_size+1));
    program_buffer[0][program_size] = '\0';
    fread(*program_buffer, sizeof(char), program_size, program_handle);
    fclose(program_handle);

    program = clCreateProgramWithSource(context, 1, (const char**) program_buffer, &program_size, &err);
        if (err!=0) {printf("clCreateProgramWithSource error: %s\n", clewErrorString(err)); return 1;}

    err = clBuildProgram(program, 1, &device, COMPILE_BUILD_OPTIONS, NULL, NULL);

    //  ----------------------------------------
    //  In case of program error...
    //  ----------------------------------------

    if (err!=0) {
            printf("clBuildProgram error, getting build log...\n");
            clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0, NULL, &log_size);
            program_log = (char*)malloc(log_size+1);
            program_log[log_size] = '\0';
            clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, log_size+1, program_log, NULL);
            printf("%s\n", program_log);
            free(program_log);
            return 1;
    }

    free(program_buffer[0]);
    free(program_buffer);

    //  ----------------------------------------
    //  Memory allocation
    //  ----------------------------------------

    kernels = malloc(5*sizeof(*kernels));
    err = clCreateKernelsInProgram(program, 5, kernels, NULL);
    if (err!=0) {printf("clCreateKernel error: %s\n", clewErrorString(err)); return 1;}

    gettimeofday(&tval_before, NULL);

    a = malloc(MAGNITUDE*sizeof(*a));
    b = malloc(MAGNITUDE*sizeof(*b));
    c = malloc(MAGNITUDE*sizeof(*c));
    result = malloc(MAGNITUDE*sizeof(*result));
    result_serial = malloc(MAGNITUDE*sizeof(*result_serial));
    if ((a==NULL)||(b==NULL)||(c==NULL)||(result==NULL)||(result_serial==NULL)) {printf("Memory allocation
failed"); exit(1);}

    //  ----------------------------------------
    //  Data initialization
    //  ----------------------------------------

    for (i=0;i<=MAGNITUDE-1;i++) {
        a[i]=(datatype)rand()/ (datatype)RAND_MAX * 100;
        b[i]=(datatype)rand()/ (datatype)RAND_MAX * 100;
        c[i]=(datatype)rand()/ (datatype)RAND_MAX * 100;
    }

    gettimeofday(&tval_after, NULL);
    tval_result.tv_sec=tval_after.tv_sec - tval_before.tv_sec;
    tval_result.tv_usec=tval_after.tv_usec - tval_before.tv_usec;
    microsecs = (long int)tval_result.tv_usec;
    secs = (long int)tval_result.tv_sec;
    time=secs + (float)microsecs/1000000;
    printf("Allocating memory and randomly filling the matrices took ");
    set_colour("green");
    printf("%f seconds\n", time);
    set_colour("normal");

    gettimeofday(&tval_before, NULL);
    for (i=0;i<=MAGNITUDE-1;i++) {
        result_serial[i] =  lgammaf(fabsf(100*erff(cosf(1+expf(5*atanf(fabsf (logf (a[i]*b[i] / c[i] /
c[i]))))))));
    }
    gettimeofday(&tval_after, NULL);
```

```
    tval_result.tv_sec=tval_after.tv_sec - tval_before.tv_sec;
    tval_result.tv_usec=tval_after.tv_usec - tval_before.tv_usec;
    microsecs = (long int)tval_result.tv_usec;
    secs = (long int)tval_result.tv_sec;
    time=secs + (float)microsecs/1000000;
    printf("Serial computation took ");
    set_colour("green");
    printf("%f seconds\n", time);
    set_colour("normal");

    //getchar();

    for (i=0;i<=4;i++) {
        run=0;
        timesum=0;
        cl_time_calc=0;
        cl_time_read=0;
        cl_time_write=0;

        err = clGetKernelInfo (kernels[i], CL_KERNEL_FUNCTION_NAME, 0, NULL, &sz);
        kname = malloc(sz*sizeof(*kname));
        err = clGetKernelInfo (kernels[i], CL_KERNEL_FUNCTION_NAME, sz, kname, NULL);
        printf("Will now test kernel %s for 10 loops. Running loop... ", kname);

        while (run<10) {
            printf("%u,  ", run);
            gettimeofday(&tval_before, NULL);

            aobj = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR, MAGNITUDE*sizeof(*a), a,
&err);
            bobj = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR, MAGNITUDE*sizeof(*b), b,
&err);
            cobj = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR, MAGNITUDE*sizeof(*c), c,
&err);
            resultobj = clCreateBuffer(context, CL_MEM_WRITE_ONLY, MAGNITUDE*sizeof(*result), NULL, &err);
                if (err!=0) {printf("clCreateBuffer error: %s\n", clewErrorString(err)); return 1;}

            err += clEnqueueWriteBuffer(queue, aobj, CL_TRUE, 0, MAGNITUDE*sizeof(*a), a, 0, NULL, &ev_WRITE1);
            err += clEnqueueWriteBuffer(queue, bobj, CL_TRUE, 0, MAGNITUDE*sizeof(*b), b, 0, NULL, &ev_WRITE2);
            err += clEnqueueWriteBuffer(queue, cobj, CL_TRUE, 0, MAGNITUDE*sizeof(*c), c, 0, NULL, &ev_WRITE3);
                if (err!=0) {printf("clEnqueueWriteBuffer error: %s\n", clewErrorString(err)); return 1;}

            CALC = kernels[i];

            err += clSetKernelArg(CALC, 0, sizeof(cl_mem), (void*) &aobj);
            err += clSetKernelArg(CALC, 1, sizeof(cl_mem), (void*) &bobj);
            err += clSetKernelArg(CALC, 2, sizeof(cl_mem), (void*) &cobj);
            err += clSetKernelArg(CALC, 3, sizeof(cl_mem), (void*) &resultobj);
                if (err!=0) {printf("clSetKernelArg error: %s\n", clewErrorString(err)); return 1;}

            if      (kname[sz-2]=='2') {*global_work_size=MAGNITUDE/2; vec=2;}
            else if (kname[sz-2]=='4') {*global_work_size=MAGNITUDE/4; vec=4;}
            else if (kname[sz-2]=='8') {*global_work_size=MAGNITUDE/8; vec=8;}
            else if (kname[sz-2]=='6') {*global_work_size=MAGNITUDE/16; vec=16;}

            //printf("Global work size = %d\n", *global_work_size);

            err = clEnqueueNDRangeKernel(queue, CALC, 1, NULL, global_work_size, NULL, 0, NULL, &ev_CALC);
                if (err!=0) {printf("clEnqueueNDRangeKernel error: %s\n", clewErrorString(err)); return 1;}

            //  -----------------------------------------
            //  Reading data back...
            //  -----------------------------------------

            err += clEnqueueReadBuffer(queue, resultobj, CL_TRUE, 0, MAGNITUDE*sizeof(*result), result, 1,
&ev_CALC, &ev_READ);
                if (err!=0) {printf("clEnqueueReadBuffer error: %s\n", clewErrorString(err)); return 1;}

            clReleaseMemObject(aobj);
            clReleaseMemObject(bobj);
            clReleaseMemObject(cobj);
            clReleaseMemObject(resultobj);

            gettimeofday(&tval_after, NULL);
            tval_result.tv_sec=tval_after.tv_sec - tval_before.tv_sec;
            tval_result.tv_usec=tval_after.tv_usec - tval_before.tv_usec;
            microsecs = (long int)tval_result.tv_usec;
            secs = (long int)tval_result.tv_sec;
            time=secs + (float)microsecs/1000000;

            timesum = timesum+time;

            err += clGetEventProfilingInfo(ev_CALC, CL_PROFILING_COMMAND_START, sizeof(cl_time_start),
&cl_time_start, NULL);
            err += clGetEventProfilingInfo(ev_CALC, CL_PROFILING_COMMAND_END, sizeof(cl_time_end), &cl_time_end,
NULL);
```

```
            cl_time_calc = cl_time_calc + cl_time_end-cl_time_start;

            err += clGetEventProfilingInfo(ev_READ, CL_PROFILING_COMMAND_START, sizeof(cl_time_start),
&cl_time_start, NULL);
            err += clGetEventProfilingInfo(ev_READ, CL_PROFILING_COMMAND_END, sizeof(cl_time_end), &cl_time_end,
NULL);
            cl_time_read = cl_time_read + cl_time_end-cl_time_start;

            err += clGetEventProfilingInfo(ev_WRITE1, CL_PROFILING_COMMAND_START, sizeof(cl_time_start),
&cl_time_start, NULL);
            err += clGetEventProfilingInfo(ev_WRITE1, CL_PROFILING_COMMAND_END, sizeof(cl_time_end),
&cl_time_end, NULL);
            cl_time_write = cl_time_write + cl_time_end-cl_time_start;

            err += clGetEventProfilingInfo(ev_WRITE2, CL_PROFILING_COMMAND_START, sizeof(cl_time_start),
&cl_time_start, NULL);
            err += clGetEventProfilingInfo(ev_WRITE2, CL_PROFILING_COMMAND_END, sizeof(cl_time_end),
&cl_time_end, NULL);
            cl_time_write = cl_time_write + cl_time_end-cl_time_start;

            err += clGetEventProfilingInfo(ev_WRITE3, CL_PROFILING_COMMAND_START, sizeof(cl_time_start),
&cl_time_start, NULL);
            err += clGetEventProfilingInfo(ev_WRITE3, CL_PROFILING_COMMAND_END, sizeof(cl_time_end),
&cl_time_end, NULL);
            cl_time_write = cl_time_write + cl_time_end-cl_time_start;

            run++;
        }

        err = clReleaseKernel(kernels[i]);
        free(kname);
        printf("\nParallel solution, data transfer & de-allocation (VEC size: %u) took ", vec);
        set_colour("green");
        printf("%f seconds", timesum/10);
        set_colour("normal");
        printf(" on average\n");
        printf("OpenCL Runtime reports:\n");
        printf("Average kernel execution time:       %f seconds\n", (float)cl_time_calc/10000000000);
        printf("Average time to read data back:      %f seconds\n", (float)cl_time_read/10000000000);
        printf("Average time to send data to device: %f seconds\n", (float)cl_time_write/10000000000);
        printf("                                     ----------------\n");
        printf("Total:                               ");
        set_colour("green");
        printf("%f seconds\n\n",
(float)cl_time_calc/10000000000+(float)cl_time_read/10000000000+(float)cl_time_write/10000000000);
        set_colour("normal");

    }  //FOR ALL VECTOR KERNELS

    //  -----------------------------------------
    //  Finishing...
    //  -----------------------------------------

    err = clFlush(queue);
    err = clFinish(queue);
    err = clReleaseCommandQueue(queue);
    err = clReleaseProgram(program);
    err = clReleaseContext(context);

    return 0;


}
```

```
__kernel void VECTOR01 (__global float  * a,
                        __global float  * b,
                        __global float  * c,
                        __global float  * result) {

    int id=get_global_id(0);

    result[id] = lgamma(fabs(100*erf(cos(1+exp(5*atan(fabs (log (a[id]*b[id] / c[id] / c[id])))))))));

}

__kernel void VECTOR02 (__global float2 * a,
                        __global float2 * b,
                        __global float2 * c,
                        __global float2 * result) {

    int id=get_global_id(0);

    result[id] = lgamma(fabs(100*erf(cos(1+exp(5*atan(fabs (log (a[id]*b[id] / c[id] / c[id])))))))));

}

__kernel void VECTOR04 (__global float4 * a,
                        __global float4 * b,
                        __global float4 * c,
                        __global float4 * result) {

    int id=get_global_id(0);

    result[id] = lgamma(fabs(100*erf(cos(1+exp(5*atan(fabs (log (a[id]*b[id] / c[id] / c[id])))))))));

}

__kernel void VECTOR08 (__global float8 * a,
                        __global float8 * b,
                        __global float8 * c,
                        __global float8 * result) {

    int id=get_global_id(0);

    result[id] = lgamma(fabs(100*erf(cos(1+exp(5*atan(fabs (log (a[id]*b[id] / c[id] / c[id])))))))));

}

__kernel void VECTOR16 (__global float16* a,
                        __global float16* b,
                        __global float16* c,
                        __global float16* result) {

    int id=get_global_id(0);

    result[id] = lgamma(fabs(100*erf(cos(1+exp(5*atan(fabs (log (a[id]*b[id] / c[id] / c[id])))))))));

}
```

# ΠΑΡΑΡΤΗΜΑ 5 – Επεξήγηση diag5solver

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $AP_{00}$ | $AE_{00}$ | | | $AN_{00}$ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| $AW_{01}$ | $AP_{01}$ | $AE_{01}$ | | | $AN_{01}$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | $AW_{02}$ | $AP_{02}$ | $AE_{02}$ | | | $AN_{02}$ | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | $AW_{03}$ | $AP_{03}$ | 0 | | | $AN_{03}$ | | | | | | | | | | | | | | | | | | | | | | | | |
| $AS_{04}$ | | | 0 | $AP_{04}$ | $AE_{04}$ | | | $AN_{04}$ | | | | | | | | | | | | | | | | | | | | | | | |
| | $AS_{05}$ | | | $AW_{05}$ | $AP_{05}$ | $AE_{05}$ | | | $AN_{05}$ | | | | | | | | | | | | | | | | | | | | | | |
| | | $AS_{06}$ | | | $AW_{06}$ | $AP_{06}$ | $AE_{06}$ | | | $AN_{06}$ | | | | | | | | | | | | | | | | | | | | | |
| | | | $AS_{07}$ | | | $AW_{07}$ | $AP_{07}$ | 0 | | | $AN_{07}$ | | | | | | | | | | | | | | | | | | | | |
| | | | | $AS_{08}$ | | | 0 | $AP_{08}$ | $AE_{08}$ | | | $AN_{08}$ | | | | | | | | | | | | | | | | | | | |
| | | | | | $AS_{09}$ | | | $AW_{09}$ | $AP_{09}$ | $AE_{09}$ | | | $AN_{09}$ | | | | | | | | | | | | | | | | | | |
| | | | | | | $AS_{10}$ | | | $AW_{10}$ | $AP_{10}$ | $AE_{10}$ | | | $AN_{10}$ | | | | | | | | | | | | | | | | | |
| | | | | | | | $AS_{11}$ | | | $AW_{11}$ | $AP_{11}$ | 0 | | | $AN_{11}$ | | | | | | | | | | | | | | | | |
| | | | | | | | | $AS_{12}$ | | | 0 | $AP_{12}$ | $AE_{12}$ | | | $AN_{12}$ | | | | | | | | | | | | | | | |
| | | | | | | | | | $AS_{13}$ | | | $AW_{13}$ | $AP_{13}$ | $AE_{13}$ | | | $AN_{13}$ | | | | | | | | | | | | | | |
| | | | | | | | | | | $AS_{14}$ | | | $AW_{14}$ | $AP_{14}$ | $AE_{14}$ | | | $AN_{14}$ | | | | | | | | | | | | | |
| | | | | | | | | | | | $AS_{15}$ | | | $AW_{15}$ | $AP_{15}$ | 0 | | | $AN_{15}$ | | | | | | | | | | | | |
| | | | | | | | | | | | | $AS_{16}$ | | | 0 | $AP_{16}$ | $AE_{16}$ | | | $AN_{16}$ | | | | | | | | | | |
| | | | | | | | | | | | | | $AS_{17}$ | | | $AW_{17}$ | $AP_{17}$ | $AE_{17}$ | | | $AN_{17}$ | | | | | | | | | |
| | | | | | | | | | | | | | | $AS_{18}$ | | | $AW_{18}$ | $AP_{18}$ | $AE_{18}$ | | | $AN_{18}$ | | | | | | | | |
| | | | | | | | | | | | | | | | $AS_{19}$ | | | $AW_{19}$ | $AP_{19}$ | 0 | | | $AN_{19}$ | | | | | | | |
| | | | | | | | | | | | | | | | | $AS_{20}$ | | | 0 | $AP_{20}$ | $AE_{20}$ | | | $AN_{20}$ | | | | | | |
| | | | | | | | | | | | | | | | | | $AS_{21}$ | | | $AW_{21}$ | $AP_{21}$ | $AE_{21}$ | | | $AN_{21}$ | | | | | |
| | | | | | | | | | | | | | | | | | | $AS_{22}$ | | | $AW_{22}$ | $AP_{22}$ | $AE_{22}$ | | | | $AN_{22}$ | | | |
| | | | | | | | | | | | | | | | | | | | $AS_{23}$ | | | $AW_{23}$ | $AP_{23}$ | 0 | | | $AN_{23}$ | | | |
| | | | | | | | | | | | | | | | | | | | | $AS_{24}$ | | | 0 | $AP_{24}$ | $AE_{24}$ | | | $AN_{24}$ | | |
| | | | | | | | | | | | | | | | | | | | | | $AS_{25}$ | | | $AW_{25}$ | $AP_{25}$ | $AE_{25}$ | | | $AN_{25}$ | |
| | | | | | | | | | | | | | | | | | | | | | | $AS_{26}$ | | | $AW_{26}$ | $AP_{26}$ | $AE_{26}$ | | | $AN_{26}$ | |
| | | | | | | | | | | | | | | | | | | | | | | | $AS_{27}$ | | | $AW_{27}$ | $AP_{27}$ | 0 | | | $AN_{27}$ |
| | | | | | | | | | | | | | | | | | | | | | | | | $AS_{28}$ | | | 0 | $AP_{28}$ | $AE_{28}$ | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | $AS_{29}$ | | | $AW_{29}$ | $AP_{29}$ | $AE_{29}$ | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | $AS_{30}$ | | | $AW_{30}$ | $AP_{30}$ | $AE_{30}$ |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | $AS_{31}$ | | | $AW_{31}$ | $AP_{31}$ |

Έστω ότι έχουμε ένα 5-διαγώνιο σύστημα (γραμμικών εξισώσεων) $[A] * x + b = 0$ προς επίλυση. Ενδεικτικά έχουμε παραπάνω τον αντίστοιχο πίνακα για 5-διαγώνιο σύστημα 32x32, που προκύπτει από πρόβλημα υπολογιστικής ρευστομηχανικής με $xp = 4$ & $yp = 8$ ($n = xp * yp$). Λόγω της δομής του πλέγματος:

| | | | |
|---|---|---|---|
| 28 | • | • | 31 |
| • | • | • | • |
| • | • | • | • |
| • | • | • | • |
| • | • | • | • |
| • | • | • | • |
| • | • | • | • |
| 0 | • | • | 3 |

*Πίνακας 1: xp κατά τον x άξονα & yp κατά τον y άξονα*

κάποια σημεία δεν έχουν γειτονικό κόμβο προς τα δεξιά ή τα αριστερά γι' αυτό και τα διανύσματα AW & AE έχουν μηδενικές τιμές σε ορισμένα στοιχεία τους (εκτός των ενδιαμέσων, το AW έχει πάντα μηδενική τιμή στο πρώτο στοιχείο του, και το AE στο τελευταίο στοιχείο του). Αντίστοιχα τα AS & AN έχουν επίσης μηδενικά στοιχεία, αλλά είναι συγκεντρωμένα είτε στην αρχή είτε στο τέλος. *Στην συνάρτηση τα διανύσματα AP, AE, AW, AN, AS δίνονται με pointers που δείχνουν στο <u>αρχικό</u> στοιχείο του κάθε διανύσματος (υποθέτοντας μήκος n=xp*yp), ανεξαρτήτως αν είναι μηδενικό ή όχι* (αυτό έχει σημασία μόνο για τα AS & AW). Δηλαδή κάθε στοιχείο έχει κάθε συντελεστή. Η συνάρτηση έπειτα δουλεύει και κάνει allocation «κόβοντας» τα μηδενικά στοιχεία του AS & AN (xpoints από το κάθε ένα) καθώς και τα ακραία μηδενικά στοιχεία του AE & AW (1 από το κάθε ένα). Το εάν τα AE & AW έχουν μηδενικά ενδιάμεσα στοιχεία ή όχι δεν επηρεάζει τη λύση (τυχαίνει να έχουν στα συγκεκριμένα προβλήματα υπολογιστικής ρευστομηχανικής, αλλά η συνάρτηση μπορεί να λύσει οποιοδήποτε 5διαγώνιο σύστημα).

Η απόσταση της διαγωνίου AN είναι $xp$ από την κεντρική διαγώνιο και της AS επίσης $xp$ από την κεντρική διαγώνιο.

Η λογική επίλυσης είναι η μέθοδος απαλοιφής Gauss με μερική οδήγηση κατά στήλη, χρησιμοποιώντας τόσα διανύσματα (διαγώνιες του πίνακα) όσα είναι απολύτως απαραίτητα για την επίλυση (δεν χρειάζεται να κρατάμε ολόκληρο τον πίνακα $nxn$, αφού είναι κατά βάση κενός). Προκύπτει ότι συνολικά χρειαζόμαστε xp διανύσματα ΚΑΤΩ από την κύρια διαγώνιο (περιλαμβανομένων των διανυσμάτων AW & AS) και $2 * xp$ διανύσματα ΠΑΝΩ από την κύρια διαγώνιο. Αυτά ομαδοποιούνται στα pointers **DOWN & **UP, με το **UP να περιέχει και την κύρια διαγώνιο. Κάθε ένα έχει το απαραίτητο μήκος (πχ δεν απαιτείται μήκος n για το διάνυσμα AN ή AS).

Τοποθετούμε τα διανύσματα AP, AE, AW, AN, AS στα αντίστοιχα UP[i] & DOWN[i] έτσι ώστε το $1^{ο}$ στοιχείο κάθε UP[i] & DOWN[i] να είναι το κόκκινο που φαίνεται παραπάνω (θυμίζουμε ότι το διάνυσμα πχ AS έχει μηδενικά στην αρχή του. Εδώ τοποθετείται ξεκινώντας από τις πραγματικές τιμές που είναι και οι μόνες που χρησιμοποιούνται).

Εφαρμόζουμε μερική οδήγηση κατά στήλη. Αναζητούμε για κάθε j το μεγαλύτερο κατ' απόλυτη τιμή στο αντίστοιχο κελί από όλα τα DOWN[i] διανύσματα, για να μπει στη θέση pivot (που είναι πάντα στο UP[0]). Αν βρεθεί μεγαλύτερος (if (index!=-1)) τότε κάνουμε το swap όλων των στοιχείων της **γραμμής πίνακα**, ξεχωριστά για τα DOWN, UP, & b.

Έπειτα γίνεται η τριγωνοποίηση (`for (k=0;k<=imin(xpoints-1,(int)n -j -2);k++)`), όπου επηρεάζονται και τα UP και τα DOWN της γραμμής όπως επίσης και το b.
Μετά γίνεται η πίσω αντικατάσταση.

Notes:

- *Η συνάρτηση είναι καταστροφική για το διάνυσμα b. Αφότου κληθεί, ΔΕΝ θα πρέπει να χρησιμοποιηθεί το διάνυσμα b σε άλλη διεργασία (θα πρέπει να έχει αντιγραφεί εκ των προτέρων για αυτό το σκοπό).*
- *Η συνάρτηση συνοδεύεται και από την debugging_printer() που μπορεί να τοποθετείται εμβόλιμα σε οποιοδήποτε σημείο της diag5solver() και να δείχνει την κατάσταση του πίνακα A σε αυτό το σημείο (ο πίνακας A δεν «υπάρχει» αλλά κατασκευάζεται εκείνη τη στιγμή από τα διανύσματα με τα οποία δουλεύουμε), με τη γραμμή που δουλεύουμε χρωματισμένη πράσινη και το αντίστοιχο στοιχείο της κεντρικής διαγωνίου χρωματισμένο κόκκινο ως εξής:*

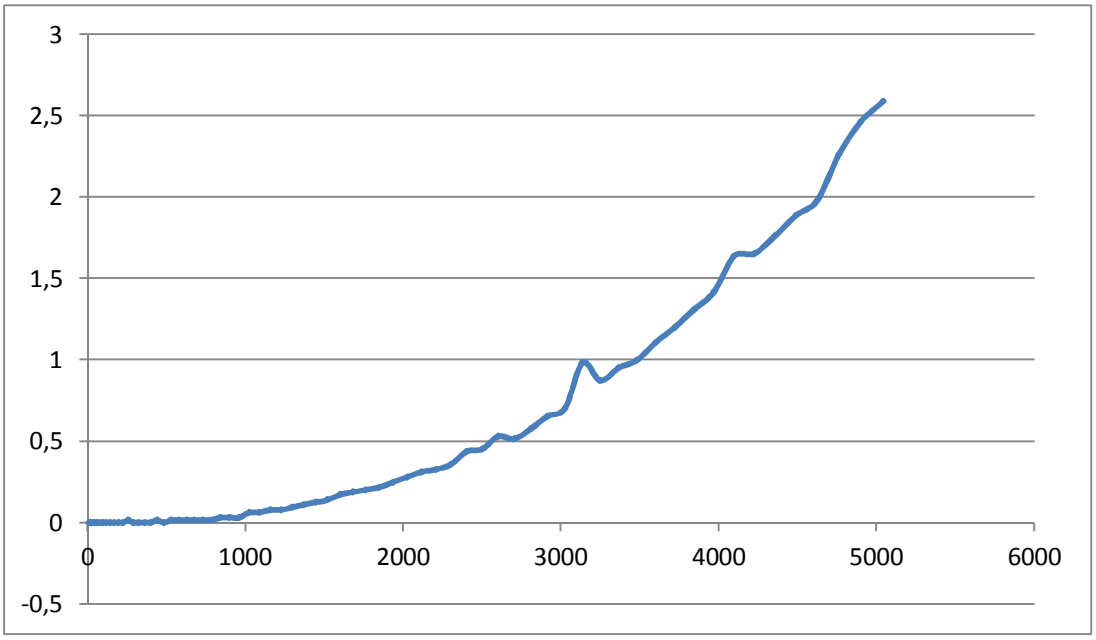
```
debugging_printer (n, xpoints, j, DOWN, UP, b);
```

  *Οπωσδήποτε να ΜΗΝ βάζει η main() μεγάλα νούμερα στα xp & yp, και βέβαια οι τιμές των AP, AE, AW, AN, AS να είναι κατά προτίμηση <1000 κατά το initialization για να είναι οι στήλες aligned και να μπορεί ο πίνακας να χωράει στην οθόνη του cmd. Προτείνονται για το πρόγραμμα cmd:*
  *Μέγεθος buffer οθόνης: 235 x 300*
  *Μέγεθος παραθύρου: 235 x 50*
  *Θέση παραθύρου: 0, 400*
- *Έχουμε επίσης και τη συνάρτηση raw_vecs_to_matrix() που θα κατασκευάσει και θα επιστρέψει τον πίνακα A (nxn) χρησιμοποιώντας τα πρωτόγονα διανύσματα AP, AE, AW, AN, AS. Αυτό μπορεί να χρησιμεύσει για να δοθεί ο ίδιος πίνακας σε συνάρτηση που λύνει πλήρες γραμμικό σύστημα κανονικά, για έλεγχο της λύσης.*
- *Η συνάρτηση ελέγχει ότι το σύστημα έχει μοναδική λύση ⇔ ο πίνακας A είναι ομαλός ⇔* $det(A) \neq 0$. *Ειδοποιεί εάν κατά την πορεία επίλυσης βρεθεί πιθανώς μηδενικό στοιχείο στην θέση pivot και τερματίζει το πρόγραμμα. Προς τούτο έγινε χρήση των όσων περιγράφονται στο παρακάτω link:*
  *http://qucs.sourceforge.net/tech/node99.html*
  *"The determinant of a triangulated matrix is the product of the diagonal elements. If the determinant* $detA$ *is non-zero the equation system has a solution"*
- *Το πρόγραμμα είναι έτοιμο για εκτέλεση και αρχίζει και λύνει ασταμάτητα (μέχρι το πρώτο σφάλμα) για τυχαίες τιμές xp & yp. Μετά από κάποιες εκτελέσεις (ανάλογα το τι* datatype *χρησιμοποιηθεί) θα έχει πρόβλημα να κάνει allocate μνήμη για τον πίνακα A, λόγω fragmentation του heap.*

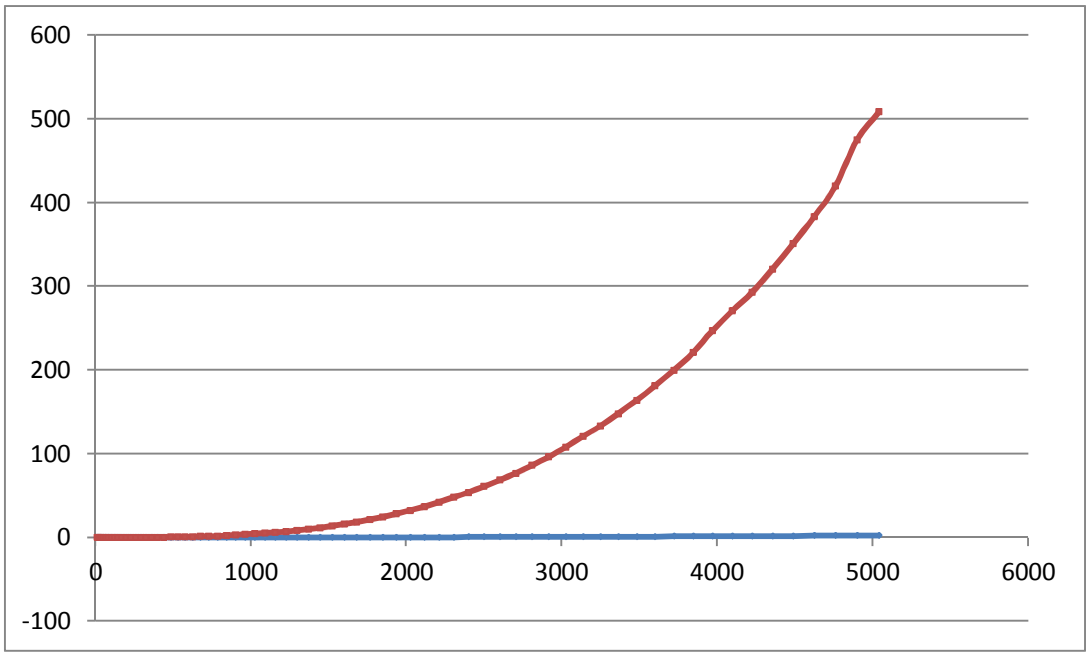
- *Η συνθήκη ελέγχου του αποτελέσματος στη main() (`if (fabs(sum+c[i])>0.000001)`) θα πρέπει να λάβει υπ' όψιν την ακρίβεια των μεταβλητών με τις οποίες δουλεύουμε, επειδή τα όποια (μικρά) σφάλματα ΟΛΩΝ των στοιχείων προστίθενται.*

- *ΠΡΟΣΟΧΗ: Ο diag5solver λύνει το σύστημα Ax+b=0 και όχι το Ax=b. Επομένως θα πρέπει τα διανύσματα που δέχεται να έχουν ήδη το κατάλληλο πρόσημο.*

Ακολουθούν δύο ενδεικτικά γραφήματα που περιγράφουν τους χρόνους επίλυσης του ίδιου πενταδιαγώνιου γραμμικού συστήματος με τη μέθοδο επίλυσης Gauss και με τη συνάρτηση που περιγράφεται στο παρόν παράρτημα (Intel Atom N2800 CPU):




*Εικόνα 1 - Χρόνοι επίλυσης σε sec πενταδιαγώνιου συστήματος με τη συνάρτηση diag5solver για διάφορες τιμές του n=xp*yp (xp=yp). Παρατηρούμε ότι η συνάρτηση αυξάνεται εκθετικά*




*Εικόνα 2 - Χρόνοι επίλυσης σε sec πενταδιαγώνιου συστήματος με τη συνάρτηση diag5solver (μπλέ χρώμα) και του ιδιου συστήματος με χρήση της κλασσικής μεθόδου Gauss (κόκκινο χρώμα) για τον αντίστοιχο πενταδιαγώνιο πίνακα, για διάφορες τιμές του n=xp*yp (xp=yp). Στο παρόν γράφημα η μπλε συνάρτηση, συγκρινόμενη με την κόκκινη, μοιάζει γραμμική*

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

#if defined(_WIN32)
#include <windows.h>
#endif

typedef __float128 datatype;
/* datatype can be (based on required precision): float,  double,  __float128 */

 //Solve 5-diagonal system of linear equations (see below)
void diag5solver (unsigned int xpoints, unsigned int ypoints, datatype *AP, datatype *AE, datatype *AW, datatype
*AN, datatype *AS, datatype *b, datatype *x);
void debugging_printer (unsigned int n, unsigned int xp, int j, datatype **DOWN, datatype **UP, datatype *b);
void raw_vecs_to_matrix (unsigned int xp, unsigned int yp, datatype *AP, datatype *AE, datatype *AW, datatype
*AN, datatype *AS, datatype *A);
void set_colour (char* colour);

int main() {

    unsigned int xp, yp, n;
    int i,j,k, flag;
    datatype *AP, *AE, *AW, *AN, *AS, *b, *c, *A, *x, *y, sum;

    srand(time(NULL));

    j=1;

    do {

    flag=0;

    xp = rand()%50+2;
    yp = rand()%50+2;

    n=xp*yp;

    printf("%3d: xp=%3u, yp=%3u - ", j, xp, yp);

    AP = malloc(n*sizeof(*AP));             if (AP==NULL) {set_colour("red"); printf("Error allocating memory
for AP..."); set_colour("normal"); exit(1);}
    AE = malloc(n*sizeof(*AE));             if (AE==NULL) {set_colour("red"); printf("Error allocating memory
for AE..."); set_colour("normal"); exit(1);}
    AW = malloc(n*sizeof(*AW));             if (AW==NULL) {set_colour("red"); printf("Error allocating memory
for AW..."); set_colour("normal"); exit(1);}
    AN = malloc(n*sizeof(*AN));             if (AN==NULL) {set_colour("red"); printf("Error allocating memory
for AN..."); set_colour("normal"); exit(1);}
    AS = malloc(n*sizeof(*AS));             if (AS==NULL) {set_colour("red"); printf("Error allocating memory
for AS..."); set_colour("normal"); exit(1);}
    b  = malloc(n*sizeof(*b));              if (b==NULL)  {set_colour("red"); printf("Error allocating memory
for b..."); set_colour("normal"); exit(1);}
    c  = malloc(n*sizeof(*c));              if (c==NULL)  {set_colour("red"); printf("Error allocating memory
for c..."); set_colour("normal"); exit(1);}

    x = malloc(n*sizeof(*x));               if (x==NULL) {set_colour("red"); printf("Error allocating memory for
x..."); set_colour("normal"); exit(1);}
    y = malloc(n*sizeof(*y));               if (y==NULL) {set_colour("red"); printf("Error allocating memory for
y..."); set_colour("normal"); exit(1);}

    A = calloc(n*n, sizeof(*A));  // Matrix A is filled with zeros during initialization when using calloc().
    if (A==NULL) {set_colour("red"); printf("Error allocating memory for A..."); set_colour("normal"); exit(1);}

    //  ------------------------------------------
    //  BUILDING PROPER INPUT VECTORS
    //  ------------------------------------------

    for (i=0;i<=n-1;i++) {
        AP[i] = (datatype) (rand()%500+1);
        AE[i] = (datatype) (rand()%500+1);
        AW[i] = (datatype) (rand()%500+1);
        AN[i] = (datatype) (rand()%500+1);
        AS[i] = (datatype) (rand()%500+1);
        b[i] = (datatype) (rand()%500+1);
        c[i] = b[i];
    }

    /*
    for (i=0;i<=yp-1;i++) {
        AW[i*xp]=0;
        AE[i*xp+xp-1]=0;
    }
    for (i=0;i<=xp-1;i++) {
```

```
        AS[i]=0;
        AN[n-1-i]=0;
    }
    */

    printf("Attempting... ");
    diag5solver (xp, yp, AP, AE, AW, AN, AS, b, x);
    raw_vecs_to_matrix (xp, yp, AP, AE, AW, AN, AS, A);

    for (i=0;i<=n-1;i++) {
        sum=0;
        for (k=0;k<=n-1;k++) {
            sum = sum + A[i*n+k]*x[k];
        }
        if (fabs((double)(sum+c[i]))>0.000001) {
            FILE* data = fopen ("Debugging data.txt", "w+");
            set_colour("red");
            printf("Error: |sum+c[%d]|=%f (!=0.000000)\n", i, fabs((double) (sum+c[i])));
            set_colour("normal");
            flag=1;
            break;
        }
    }

    if (flag==0) {
        set_colour("green");
        printf("Success!");
        set_colour("normal");
    }

    free(AP);
    free(AE);
    free(AW);
    free(AN);
    free(AS);
    free(b);
    free(c);

    free(x);
    free(y);

    free(A);

    j++;

    printf("  -\n");

    } while (flag==0);

    return 0;
}


void diag5solver (unsigned int xpoints, unsigned int ypoints, datatype *AP, datatype *AE, datatype *AW, datatype
*AN, datatype *AS, datatype *b, datatype *x) {

     int imin(int x, int y) {
        if (x<y) {return x;}
        else {return y;}
     }

     int i,j,k,index;
     unsigned int n=xpoints*ypoints;
     datatype temp, sum, biggest, poll, **UP, **DOWN;

     // ------------------------------
     // Memory allocation for all UP & DOWN vectors
     // ------------------------------
     UP = malloc((2*xpoints+1)*sizeof(*UP));
     DOWN = malloc(xpoints*sizeof(*DOWN));
     if (UP==NULL||DOWN==NULL) {printf("Error allocating memory..."); exit(1);}

     // ------------------------------
     // Memory allocation for all vector elements
     // ------------------------------
     for (i=0;i<=2*xpoints;i++) {
        UP[i] = calloc(n-i, sizeof(**UP));
        if (UP[i]==NULL) {printf("Error allocating memory..."); exit(1);}
     }
     for (i=0;i<=xpoints-1;i++) {
        DOWN[i] = calloc(n-1-i, sizeof(**DOWN));
        if (DOWN[i]==NULL) {printf("Error allocating memory..."); exit(1);}
     }

     // ------------------------------
     // Overlay of input data to proper UP & DOWN vectors
```

```
    // -------------------------------
    for (i=0;i<=n-1;i++) {
       (UP[0])[i] = AP[i];
    }
    for (i=0;i<=n-2;i++) {
       (UP[1])[i] = AE[i];
    }
    for (i=0;i<=n-1-xpoints;i++) {
       (UP[xpoints])[i] = AN[i];
    }
    for (i=0;i<=n-2;i++) {
       (DOWN[0])[i] = AW[i+1];
    }
    for (i=0;i<=n-xpoints-1;i++) {
       (DOWN[xpoints-1])[i] = AS[i+xpoints];
    }

    for (j=0;j<=n-2;j++) {
       biggest = fabs((UP[0])[j]);
       index = -1;
       // -------------------------------
       // Searching for biggest pivot
       // -------------------------------
       for (k=0;k<=imin(xpoints-1,(int)n -j -2);k++) { // Variable k enumerates DOWN vectors
           if (fabs((DOWN[k])[j])>biggest) {
               biggest = fabs((DOWN[k])[j]);
               index = k;  //DOWN[index] contains the largest element in the j-th column (by absolute value) to
be used as pivot
           }
       }

       // -------------------------------
       // Swapping whole line
       // -------------------------------
       if (index!=-1) {
           for (k=0;k<=index;k++) {  //Swapping of proper DOWN elements
               temp = (DOWN[index-k])[j+k];
               (DOWN[index-k])[k+j] = (UP[k])[j];
               (UP[k])[j] = temp;
           }
           for (k=0;k<=imin(2*xpoints-1-index,(int)n-j-2-index);k++) { //Swapping of proper UP elements
               temp = (UP[k])[index+1+j];
               (UP[k])[index+1+j] = (UP[index+1+k])[j];
               (UP[index+1+k])[j] = temp;
           }

           //Swapping values of b
           temp = b[j];
           b[j] = b[index+1+j];
           b[index+1+j] = temp;
       } //Swap is complete

       if (fabs(UP[0][j])<0.000001) {
           printf("Possible division by zero, function will exit right away...\n");
           exit(1);
       }

       // -------------------------------
       // Triangulation
       // -------------------------------
       for (k=0;k<=imin(xpoints-1,(int)n -j -2);k++) { //k enumerates lines, up to xpoints-1 or the end of the
matrix
           poll = (DOWN[k])[j] / (UP[0])[j];
           for (i=0;i<=k;i++) {    //Elements of all DOWN vectors that correspond to the k-th line
               (DOWN[k-i])[i+j] = (DOWN[k-i])[i+j] - poll*(UP[i])[j];
           }
           for (i=0;i<=imin(2*xpoints-1-k,n-2-j-k);i++) {  //Elements of all UP vectors that correspond to the
k-th line
               (UP[i])[k+1+j] = (UP[i])[k+1+j] - poll*(UP[i+1+k])[j];
           }
           b[k+j+1] = b[k+j+1] - poll*b[j];
       }

    } //Triangulation complete

    // -------------------------------
    // Backwards substitution
    // -------------------------------
    x[n-1] = -b[n-1]/(UP[0])[n-1];
    for (i=n-2;i>=0;i--) {
       sum = 0;
       for (j=1;j<=imin(2*xpoints,n-i-1);j++) {
           sum = sum + (UP[j])[i] * x[i+j];
       }
       x[i] = (-b[i]-sum)/(UP[0])[i];
    }
```

```
}

void debugging_printer (unsigned int n, unsigned int xp, int j, datatype **DOWN, datatype **UP, datatype *b) {

    /* Function should only be used to print values up to and excluding 1000 (for proper column alignment) */

    int i,k, proper_output=1;
    datatype* A;

    A = calloc(n*n, sizeof(*A));

    for (k=0;k<=2*xp;k++) {
        for (i=0;i<=n-1-k;i++) {
            A[i*n+i+k] = (UP[k])[i];
        }
    }

    printf("\n");
    for (k=0;k<=xp-1;k++) {
        for (i=0;i<=n-2-k;i++) {
            A[i*n+i+(k+1)*n] = (DOWN[k])[i];
        }
    }

    for (i=0;i<=n-1;i++) {
        if (i==j) {set_colour("green");}
        for (k=0;k<=n-1;k++) {
            if (k==j&&i==j) {set_colour("red");}
            printf("%+.2f  ", (float)A[i*n+k]);
            if (fabs((double)A[i*n+k])<10) {printf("  ");}
            if ((fabs((double)A[i*n+k])>=10)&&(fabs((double)A[i*n+k])<100)) {printf(" ");}
            if (fabs((double)A[i*n+k])>=1000) {proper_output=0;}
            if (k==j&&i==j) {
                if (i==j) {set_colour("green");}
                else {set_colour("normal");}
            }
        }
        printf("  %+.2f", (float)b[i]);
        printf("\n");
        if (i==j) {set_colour("normal");}
    }
    if (proper_output==0) {printf("\nMatrix columns above may not be properly aligned (Values >=1000?)\n");}
    getchar();
}

void raw_vecs_to_matrix (unsigned int xp, unsigned int yp, datatype *AP, datatype *AE, datatype *AW, datatype
*AN, datatype *AS, datatype *A) {

    unsigned int n = xp*yp;
    int i;

    //A = calloc(n*n, sizeof(*A)); => ALL elements of A are ZERO

    for (i=0;i<=n-1;i++) {
        A[i*n+i] = AP[i];
    }
    for (i=0;i<=n-2;i++) {
        A[i*n+i+1] = AE[i];
    }
    for (i=1;i<=n-1;i++) {  //AW[0] = 0!
        A[i*n+i-1] = AW[i];
    }
    for (i=0;i<=n-1-xp;i++) {
        A[i*n+i+xp] = AN[i];
    }
    for (i=xp;i<=n-1;i++) {
        A[i*n+i-xp] = AS[i];
    }
}

void set_colour (char* colour) {

    #if defined(_WIN32)
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    int col;

    if (strcmp("deep_blue", colour)==0) {col=1;}
    else if (strcmp("deep_green", colour)==0) {col=2;}
    else if (strcmp("deep_cyan", colour)==0) {col=3;}
    else if (strcmp("deep_red", colour)==0) {col=4;}
    else if (strcmp("deep_magenta", colour)==0) {col=5;}
    else if (strcmp("deep_yellow", colour)==0) {col=6;}
    else if (strcmp("normal", colour)==0) {col=7;}
    else if (strcmp("grey", colour)==0) {col=8;}
    else if (strcmp("blue", colour)==0) {col=9;}
```

```
    else if (strcmp("green", colour)==0) {col=10;}
    else if (strcmp("cyan", colour)==0) {col=11;}
    else if (strcmp("red", colour)==0) {col=12;}
    else if (strcmp("magenta", colour)==0) {col=13;}
    else if (strcmp("yellow", colour)==0) {col=14;}
    else if (strcmp("white", colour)==0) {col=15;}

    else {col=7;}  //NORMAL COLOUR

    SetConsoleTextAttribute(hConsole, col);

    #elif
defined(__linux__)||defined(__sun)||defined(__FreeBSD__)||defined(__NetBSD__)||defined(__OpenBSD__)||defined(__A
PPLE__)

    #define KNRM  "\x1B[0m"
    #define KRED  "\x1B[31m"
    #define KGRN  "\x1B[32m"
    #define KYEL  "\x1B[33m"
    #define KBLU  "\x1B[34m"
    #define KMAG  "\x1B[35m"
    #define KCYN  "\x1B[36m"
    #define KWHT  "\x1B[37m"

    if (strcmp("red", colour)==0) {printf(KRED);}
    else if (strcmp("green", colour)==0) {printf(KGRN);}
    else if (strcmp("yellow", colour)==0) {printf(KYEL);}
    else if (strcmp("blue", colour)==0) {printf(KBLU);}
    else if (strcmp("magenta", colour)==0) {printf(KMAG);}
    else if (strcmp("cyan", colour)==0) {printf(KCYN);}
    else if (strcmp("white", colour)==0) {printf(KWHT);}
    else if (strcmp("normal", colour)==0) {printf(KNRM);}

    else {printf(KNRM);}  //NORMAL COLOUR

    #endif

    return;
}
```

## Παραπομπές


https://www.gsmarena.com/samsung_galaxy_s7-7821.php

https://en.wikipedia.org/wiki/Mali_(GPU)

https://ark.intel.com/products/33924/Intel-Core2-Quad-Processor-Q9550-12M-Cache-2_83-GHz-1333-MHz-FSB

https://ark.intel.com/products/53422/Intel-Core-i3-2100-Processor-3M-Cache-3_10-GHz

https://en.wikipedia.org/wiki/Radeon_HD_7000_Series#Radeon_HD_7900

https://ark.intel.com/products/58917/Intel-Atom-Processor-N2800-1M-Cache-1_86-GHz

https://en.wikipedia.org/wiki/Ryzen

https://www.gsmarena.com/oneplus_x-7630.php

https://en.wikipedia.org/wiki/AMD_RX_Vega_series

https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf

https://web.archive.org/web/20070401023156/https://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~114147,00.html

https://www.top500.org/

https://bitcoin.org/bitcoin.pdf

# Βιβλιογραφία


- *An Introduction to Computational Fluid Dynamics – The Finite Volume Method (2nd edition) (H. K. Versteeg & W. Malalasekera - ISBN: 978-0-13-127498-3)*

- *The OpenCL™ Specification*
  *Khronos OpenCL Working Group*
  *Version 2.2-7, Sat, 12 May 2018 13:21:25 +0000*