Εθνικό Μετσόβιο Πολυτεχνείο
Σχολη Μηχανολογων Μηχανικων
Τομεας Ρευστων
Εργαστηριο Θερμικων Στροβιλομηχανων
Μοναδα Παραλληλης Υπολογιστικης
Ρευστοδυναμικης και Βελτιστοποιησης

Προγραμματισμός επιλύτη 3Δ εξισώσεων ροής ατριβούς ρευστού σε δομημένα πλέγματα, σε κάρτες γραφικών

Διπλωματική Εργασία

Ιωάννης Σ. Καββαδίας

Επιβλέπων: Κ.Χ. Γιαννάκογλου
Καθηγητής ΕΜΠ

Οκτώβριος 2011

Ευχαριστίες

Από τη θέση αυτή θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας Κ. Γιαννάκογλου, Καθηγητή ΕΜΠ, για τη δυνατότητα που μου έδωσε να ασχοληθώ με το παρόν αντικείμενο, για την καθοδήγησή του καθώς και την δυνατότητα χρήσης του τεχνικού εξοπλισμού της Μονάδας Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης. Επίσης θα ήθελα να ευχαριστήσω τον υποψήφο διδάκτορα Ξ. Τρομπούκη, τη Δρ. Β. Ασούτη καθώς και όλη την υπόλοιπη ερευνητική ομάδα της Μονάδας Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης για την πολύτιμη βοήθειά τους ανεξαρτήτως ώρας και μέρας. Τέλος, θέλω να ευχαριστήσω τους φίλους μου και την οικογένειά μου για την συνεχή υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Στη μνήμη του πατέρα μου.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολη Μηχανολογων Μηχανικων
Τομεας Ρευστων
Εργαστηριο Θερμικων Στροβιλομηχανων
Μοναδα Παραλληλης Υπολογιστικης Ρευστοδυναμικης και Βελτιστοποιησης

Προγραμματισμός επιλύτη 3Δ εξισώσεων ροής ατριβούς ρευστού σε δομημένα πλέγματα, σε κάρτες γραφικών.

Διπλωματική Εργασία
Ιωάννη Σ. Καββαδία
Επιβλέπων: Κ.Χ. Γιαννάκογλου
Καθηγητής ΕΜΠ

Οκτώβριος 2011

Τα τελευταία τρία έτη, η Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης του Εργαστηρίου Θερμικών Στροβιλομηχανών (ΕΘΣ) του ΕΜΠ δραστηριοποιείται ερευνητικά στην εκμετάλλευση των δυνατοτήτων των καρτών γραφικών σε κώδικες και εφαρμογές υπολογιστικής ρευστοδυναμικής και βελτιστοποίησης. Σχετικά με τους επιλύτες των εξισώσεων ροής (εξισώσεις Euler, Navier-Stokes), το λογισμικό που ήδη έχει αναπτυχθεί σε CUDA C δίνει, σε κάρτες γραφικών τελευταίας τεχνολογίας της NVIDIA, επιταχύνσεις μεγαλύτερες του $\times 40$, ανάλογα με το πλέγμα, την εφαρμογή κλπ. Οι επιλύτες που αναπτύχθηκαν χειρίζονται μη-δομημένα πλέγματα με μεθόδους πεπερασμένων όγκων κεντροκομβικής διατύπωσης των εξισώσεων. Η χρήση μη-δομημένου πλέγματος με κεντροκομβική διατύπωση είναι, προφανώς, η δυσμενέστερη περίπτωση από την πλευρά της αποδοτικής χρήσης μνήμης στις κάρτες γραφικών. Στην παρούσα διπλωματική εργασία διερευνάται η χρήση δομημένων πλεγμάτων για την αποδοτικότερη αξιοποίηση των δυνατοτήτων των καρτών γραφικών, σε εφαρμογές υπολογιστικής ρευστοδυναμικής. Έτσι, στη διπλωματική αυτή εργασία, αναπτύχθηκε/προγραμματίστηκε (σε CUDA C) επιλύτης των 3Δ εξισώσεων Euler συμπιεστού ρευστού με χρήση δομημένων πλεγμάτων, η διακριτοποίηση των οποίων γίνεται με σχήμα πεπερασμένων όγκων κεντροκομβικής διατύπωσης και η επίλυσή τους γίνεται με χρήση μεθόδου χρονοπροέλασης (time marching). Η ανάπτυξη του επιλύτη ροών για δομημένα πλέγματα βασίστηκε στον υπάρχοντα επιλύτη ροών για μη-δομημένα πλέγματα του ΕΘΣ, σε κάρτες γραφικών. Ο προγραμματισθείς κώδικας, εκτελούμενος σε μία κάρτα γραφικών NVIDIA TESLA M2050, συγκρίθηκε με τον υπάρχοντα οικείο και πιστοποιημένο κώδικα σε FORTRAN, που έτρεξε σε έναν πυρήνα του κεντρικού επεξεργαστή. Η λύση της ροής και η σύγκλιση που παράγουν οι δύο επιλύτες είναι ταυτόσημη και η σύγκριση των επιδόσεων των επιλυτών, τελικά, παρουσιάζει το αναμενόμενο κέρδος από τη χρήση δομημένων πλεγμάτων, με τον επιλύτη της κάρτας γραφικών να δίνει επιτάχυνση μεγαλύτερη του $\times 50$.

National Technical University of Athens
Department of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Programming of a GPU-enabled 3D Euler equations' solver on structured grids

Diploma Thesis

Ioannis S. Kavvadias

Advisor: K.C. Giannakoglou

October 2011

In the last three years, the Parallel CFD & Optimization Unit of the Lab. of Thermal Turbomachines of the National Technical University of Athens is focusing on the exploitation of the computational power of new generation Graphics Processing Units (GPU) for CFD and Optimization software and applications. In the field of CFD, in-house solvers, for both the Euler and Navier-Stokes equations, have already been programmed in CUDA C. These codes, running on modern NVIDIA GPU, have achieved speed ups greater than $\times 40$ (compared to a modern CPU), depending on the grid in use, the application etc. The above solvers use vertex-centered finite volume methods for unstructured grids which is the worst-case scenario when trying to exploit the full extent of a GPU's computational power. In this diploma thesis, the use of structured grids is tried out instead of unstructured grids because, by that way, it is expected to have a better exploitation of the computational power of a GPU. So, a 3D compressible Euler equations solver was programmed (using CUDA C), using a vertex-centered finite volume method for structured grids and a time-marching method. The development of the structured grid solver was based on the in-house GPU-enabled unstructured grid solver. Performance tests were carried out between the GPU-enabled solver running on a single NVIDIA TESLA M2050 GPU and the certified CPU-enabled solver of the Parallel CFD & Optimization Unit, programmed using FORTRAN, running on a single core of a CPU. The results of both solvers are identical and the speed-up between the GPU and the CPU is greater than $\times 50$, as expected.

Περιεχόμενα

Κεφάλαιο 1

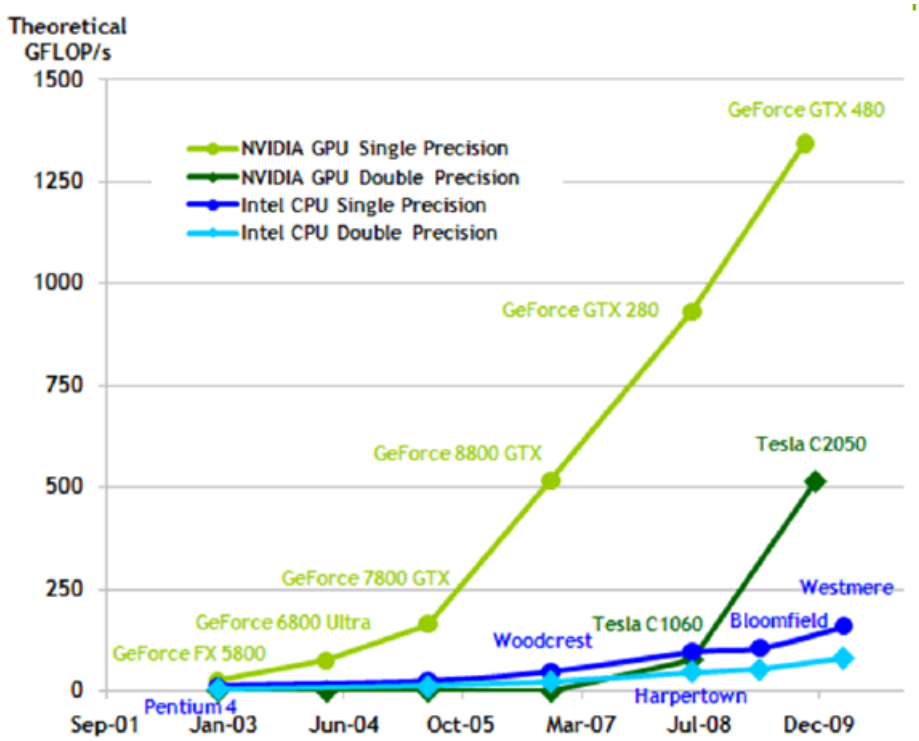
Εισαγωγή

Ένα από τα βασικά προβλήματα που αντιμετωπίζει η υπολογιστική ρευστοδυναμική, στο σημερινό της επίπεδο, είναι η έλλειψη υπολογιστικής ισχύος, καθώς η επίλυση των θεωρητικών μοντέλων προσομοίωσης ροής είναι χρονοβόρα και ο χρόνος που απαιτείται είναι ακόμα και απαγορευτικός, σε κάποιες περιπτώσεις, για την βιομηχανία. Στην παρούσα εργασία εξετάζεται η χρήση κάρτας γραφικών (GPU, Graphics Processing Units), αντί του κεντρικού επεξεργαστή του υπολογιστή (CPU, Central Processing Unit) στην Υπολογιστική Ρευστοδυναμική, αφού οι σημερινές GPU προσφέρουν υπολογιστική ισχύ μεγαλύτερη των CPU τουλάχιστον κατά μία τάξη μεγέθους, όπως φαίνεται και στα σχήματα 1.1 και 1.2. Για το σκοπό αυτό αναπτύχθηκε και πιστοποιήθηκε επιλύτης των εξισώσεων Euler, συμβατός με εκτέλεση σε GPU, και αξιολογήθηκε η απόδοσή του συγκριτικά με τον αντίστοιχο επιλύτη σχεδιασμένο για εκτέλεση από την CPU. Η σύγκριση έγινε με χρήση GPU και CPU τελευταίας τεχνολογίας.

1.1 Οι Κάρτες Γραφικών σήμερα.

Μέχρι πριν από λίγα χρόνια, η ισχύς ενός υπολογιστή εξαρτάτο από την ταχύτητα χρονισμού του πυρήνα της CPU και για την επίτευξη μεγαλύτερης υπολογιστικής ισχύος χρειαζόταν πιο γρήγορος χρονισμός (clock rate). Όμως, η ενέργεια που απαιτείται για τη λειτουργία του επεξεργαστή είναι ανάλογη του τετραγώνου της ταχύτητας χρονισμού του πυρήνα του. Έτσι φαίνεται πως η συνεχής αύξηση της ταχύτητας χρονισμού του πυρήνα του επεξεργαστή δεν είναι αποδεκτή λύση καθώς οδηγεί σε δυσανάλογα μεγάλη κατανάλωση ενέργειας. Για αυτό το λόγο οι κατασκευαστές ξεκίνησαν να αυξάνουν το πλήθος των πυρήνων ανά επεξεργαστή, αντί της ταχύτητας χρονισμού του κάθε πυρήνα ξεχωριστά. Η τεχνική αυτή έχει γίνει ευρέως αποδεκτή και, σήμερα, υπάρχουν στην παραγωγή επεξεργαστές που αποτελούνται μέχρι και απο 12 πυρήνες ενώ ετοιμάζονται επεξεργαστές 16 πυρήνων.

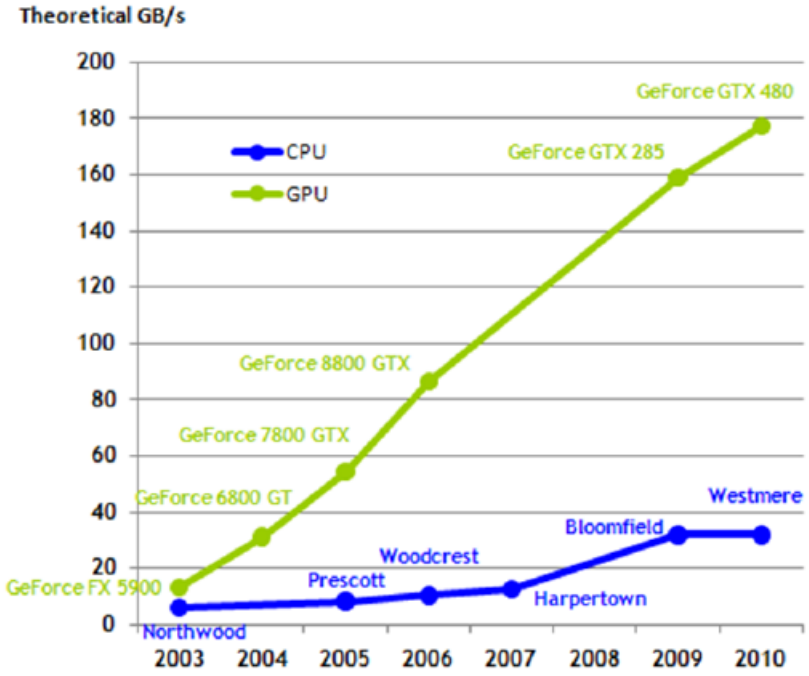
Την ίδια τεχνική χρήσης πολλών πυρήνων χρησιμοποιούν και οι GPU, οι οποίες ακολουθούν πλήρως παράλληλη λογική και αποτελούνται από εκατοντάδες πυρήνες η καθεμιά. Συγκριτικά με τους πυρήνες των CPU, οι πυρήνες των GPU είναι αισθητά



Σχήμα 1.1: Ταχύτητα εκτέλεσης πράξεων κινητής υποδιαστολής [1].

πιο αργοί όμως, λόγω του πλήθους τους, οι σημερινές GPU πετυχαίνουν ταχύτητες επεξεργασίας δεδομένων πολλαπλάσιες των αντίστοιχων που επιτυγχάνονται από τις CPU. Έτσι, τα τελευταία χρόνια αναπτύχθηκαν GPU αρχιτεκτονικών συμβατών με τα διεθνή πρότυπα εκτέλεσης πράξεων καθώς και μέθοδοι προγραμματισμού των GPU αυτών βασισμένες σε γλώσσες προγραμματισμού όπως η C/C++, η FORTRAN και η OpenCL. Οι αρχιτεκτονικές αυτές είναι η CUDA, που αντιστοιχεί στα αρχικά των «Compute Unified Device Architecture», και η εξέλιξη της τεχνολογίας των Streams, οι οποίες αναπτύχθηκαν από τις εταιρίες NVIDIA και ATI αντίστοιχα. Η ATI ονομάζει την καινούργια αυτή τεχνολογία AMD APP, όπου τα αρχικά APP σημαίνουν Accelerated Parallel Processing. Και οι δύο αρχιτεκτονικές, αν και αναπτύχθηκαν ξεχωριστά, επιτρέπουν τον παράλληλο προγραμματισμό των GPU για οποιαδήποτε εφαρμογή. Αυτό είναι γνωστό και ως General-Purpose Computing on Graphics Processing Units ή GPGPU.

Ακόμα ένα σημαντικό πλεονέκτημα των GPU συγκριτικά με τις CPU είναι το χαμηλό κόστος κτήσης. Η ραγδαία αύξηση των δυνατοτήτων των GPU τα τελευταία χρόνια οφείλεται σε μεγάλο βαθμό στις απαιτήσεις της παιχνιδοβιομηχανίας για καλύτερα και πιο αληθοφανή γραφικά. Οπότε, δεν θα αρκούσε η δημιουργία πολύ ισχυρών GPU αν το αγοραστικό κοινό δεν ήταν σε θέση να τις αποκτήσει. Στη σημερινή πραγματικότητα, μία GPU μπορεί να έχει εντυπωσιακές αποδόσεις, ενώ το κόστος της είναι μόλις ένα κλάσμα της τιμής κτήσης ενός αντίστοιχου υπολογιστικού συστήματος.



Σχήμα 1.2: Εύρος Ζώνης (Bandwidth) της μνήμης των GPU και CPU τα τελευταία χρόνια [1].

1.2 Ανάπτυξη εφαρμογών σε GPU.

Ο συνδυασμός της μεγάλης υπολογιστικής ισχύος, του εύκολου πλέον προγραμματισμού καθώς και του χαμηλού κόστους κτήσης οδήγησε την επιστημονική κοινότητα στην διερεύνηση των καινούργιων δυνατοτήτων που προσφέρουν οι GPU σε όλους τους τομείς της έρευνας και των εφαρμογών. Τα αποτελέσματα είναι εντυπωσιακά, καθώς παρατηρούνται επιταχύνσεις εφαρμογών από 10 εώς και πάνω από 200 φορές, ανάλογα με το είδος της εκάστοτε εφαρμογής και τον βαθμό παραλληλοποίησής της. Ακόμη, σημαντικό είναι πως η καινούργια αυτή τεχνολογία δεν απευθύνεται σε ένα μόνο τομέα, όπως είναι η ρευστοδυναμική, η οποία και εξετάζεται στην παρούσα εργασία, αλλά σε οποιοδήποτε τομέα που χρειάζεται περισσότερη υπολογιστική ισχύ.

Ενδεικτικά αναφέρονται μερικές δημοσιεύσεις από τη βιβλιογραφία, μαζί με την επιτάχυνση που έχει επιτευχθεί στην εκάστοτε εφαρμογή, από διάφορους τομείς. Έχουν αναπτυχθεί:

- Λογισμικό επεξεργασίας φωτογραφιών αξονικού τομογράφου, με χρήση GPU, με χρονική επιτάχυνση 14 φορές [2].
- Γεννήτρια ψευδοτυχαίων αριθμών, με επιτάχυνση πάνω από 33 φορές [3].
- Επιλύτης ροής υγρών κρυστάλλων, με επιτάχυνση πάνω από 50 φορές [4].

- Μοντέλο πρόβλεψης κινδύνου για την αποθήκευση διοξειδίου του άνθρακα στο υπέδαφος, με επιτάχυνση πάνω από 60 φορές [5].

- Μοντέλο προσομοίωσης μεταφοράς νετρονίων με χρήση της μεθόδου του Monte Carlo και επιλύτης μεταφοράς θερμότητας με επιτάχυνση μεγαλύτερη από 100 φορές [6]

Ακόμα αναφέρονται και μερικές δημοσιεύσεις από την βιβλιογραφία από τον τομέα της υπολογιστικής ρευστομηχανικής, με τον οποίο και ασχολείται η παρούσα εργασία, μαζί με την αντίστοιχη επιτάχυνση κάθε εφαρμογής. Έχουν αναπτυχθεί:

- Επιλύτης εξισώσεων Euler για ροές γύρω από υπερηχητικό αεροσκάφος, με επιτάχυνση πάνω από 40 φορές [7].

- Προσομοιωτής ροών γύρω από μη-αεροδυναμικό σώμα (bluff-body) με χρήση μεθόδων στροβιλιζόμενων σωματιδίων (vortex particle methods) με επιτάχυνση πάνω από 30 φορές [8].

- Προσομοιωτής ροών μέσα σε κτίρια με χρήση γρήγορων μοντέλων υπολογιστικής ρευστοδυναμικής με επιτάχυνση πάνω από 30 φορές [9].

- Μέθοδος παράλληλης επεξεργασίας της μεθόδου χτισίματος - κύβου (Building-Cube Method) με επιτάχυνση περίπου 3 φορές [10].

- Προσομοιωτής διδιάστατου μοντέλου παλίρροιας με επιτάχυνση έως και 88 φορές [11].

1.3 Στόχος της διπλωματικής εργασίας.

Στη Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής και Βελτιστοποίησης του Εργαστήριο Θερμικών Στροβιλομηχανών (ΕΘΣ), από την οποία και εκπονήθηκε η παρούσα διπλωματική εργασία, τα τελευταία χρόνια έχει αποκτηθεί μία σημαντική εμπειρία σχετικά με τις GPUs και τον τρόπο χρήσης τους. Έχουν αναπτυχθεί επιλύτες συνεκτικών και μη-συνεκτικών, χρονικά μόνιμων και μη-μόνιμων, διδιάστατων και τριδιάστατων ροών [12], [13] και παρουσιάζονται επιλύτες έως και 46 φορές πιο γρήγοροι από τους αντίστοιχους σειριακούς επιλύτες της CPU. Ακόμα, έχουν εφαρμοστεί μέθοδοι βελτιστοποίησης, με χρήση GPU [14], [15], [16], [17]. Όμως, όλα τα παραπάνω έχουν εφαρμοστεί σε μη-δομημένα πλέγματα, τα οποία, παρά την ευελιξία τους, έχουν ένα μεγάλο μειονέκτημα, την άτακτη αρίθμηση των κόμβων του πλέγματος. Όπως αναλύεται στο κεφάλαιο 2, άτακτη προσπέλαση της μνήμης της GPU μειώνει αισθητά την τελική επίδοση της εφαρμογής.

Στόχος της παρούσας διπλωματικής εργασίας είναι η διερεύνηση των επιδόσεων μίας GPU, όταν η επίλυση της ροής γίνεται με χρήση δομημένων πλεγμάτων. Αναμένεται με χρήση δομημένων πλεγμάτων να βελτιωθεί σημαντικά ο σχετικός χρόνος ανάμεσα στην επίλυση της GPU και της CPU καθώς η ταξινόμηση των κόμβων μεταφράζεται σε καλύτερη διαχείριση της μνήμης, η οποία σημαίνει και καλύτερη

αξιοποίηση των πόρων που προσφέρει η GPU. Ακόμα, η χρήση δομημένου πλέγματος καθιστά γνωστό και σταθερό τον αριθμό των γειτόνων κάθε κόμβου με αποτέλεσμα ο επιλύτης να μπορεί να παραλληλοποιηθεί πιο ομαλά, καθώς η ανάγκη για πληροφορίες από τους γειτονικούς κόμβους παρουσιάζεται πολύ συχνά κατά την επίλυση των εξισώσεων της ροής. Ο επιλύτης που αναπτυχθήκε βασίζεται στην αρχιτεκτονική CUDA της NVIDIA και πιο συγκεκριμένα στις δυνατότητες που προσφέρουν οι **GPU αρχιτεκτονικής Fermi υπολογιστικής ικανότητας** 2.0.

1.4 Δομή της εργασίας.

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός επιλύτη εξισώσεων Euler, με γνώμονα την παράλληλη λειτουργία των GPU. Έτσι, η εργασία είναι δομημένη ως εξής :

- Στο κεφάλαιο 2 παρουσιάζεται η αρχιτεκτονική CUDA, ο τρόπος προγραμματισμού τους καθώς και οι δυνατότητες ανάλογα με την υπολογιστική ικανότητα της κάθε GPU.
- Στο κεφάλαιο 3 παρουσιάζεται η μαθηματική διατύπωση των εξισώσεων Euler για συμπιεστό ρευστό. Ακόμα αναλύεται το κεντροκομβικό σχήμα των πεπερασμένων όγκων το οποίο και χρησιμοποιήθηκε για τη διακριτοποίηση και την αριθμητική επίλυση των εξισώσεων.
- Στο κεφάλαιο 4 παρουσιάζεται ο επιλύτης που αναπτύχθηκε και εξηγείται η λογική της ανάπτυξής του.
- Στο κεφάλαιο 5 παρουσιάζονται τα αποτελέσματα όπως προέκυψαν από τον παραπάνω επιλύτη και οι συγκριτικές του επιδόσεις με αντίστοιχο επιλύτη σχεδιασμένο για εκτέλεση από τη CPU.
- Τέλος, στο κεφάλαιο 6 γίνεται μία σύντομη ανασκόπηση ολόκληρης της διπλωματικής εργασίας καθώς και παρουσιάζονται τα συμπεράσματα που προέκυψαν από αυτή.

Κεφάλαιο 2

Αρχιτεκτονική και περιβάλλον CUDA

Το ακρώνυμο CUDA [18] προέρχεται από το "Compute Unified Device Architecture", ή στα Ελληνικά «Αρχιτεκτονική Ενοποιημένης Υπολογιστικής Συσκευής». Ο σκοπός σχεδιασμού της ήταν να δοθεί στους χρήστες η δυνατότητα ανάπτυξης εφαρμογών, οποιασδήποτε μορφής, στη GPU, πιστοποιημένα[1], εύκολα και αποδοτικά. Οι κάρτες γραφικών CUDA είναι σχεδιασμένες να υποστηρίζουν όλες τις διεργασίες του συμβατικού προγραμματισμού, μερικές εκ των οποίων είναι οι πράξεις ακεραίων, πράξεις κινητής υποδιαστολής, λογικές πράξεις και διαχείριση δεικτών. Σε τελευταίες εκδόσεις (σε κάρτες υπολογιστικής ικανότητας 2.x ή μεταγενέστερες) της CUDA υποστηρίζεται ακόμα και εκτύπωση κατευθείαν στην οθόνη. Σε προγραμματιστικό επίπεδο, ο χρήστης δεν χρειάζεται πλέον να γνωρίζει κάποια εξειδικευμένη, για κάρτες γραφικών, γλώσσα προγραμματισμού, καθώς ο προγραμματισμός τους γίνεται μέσα από επεκτάσεις παραδοσιακών γλωσσών, υψηλού αλλά και χαμηλού επιπέδου. Τη στιγμή που γράφεται το κείμενο αυτό, υποστηρίζονται επεκτάσεις για γλώσσες όπως η C/C++ [1][19], η FORTRAN και η OpenCL, καθώς και προγραμματιστικά περιβάλλοντα όπως το DirectCompute. Στην παρούσα διπλωματική εργασία έχει χρησιμοποιηθεί η επέκταση της C/C++, γνωστή και ως CUDA C.

2.1 Παράδειγμα αλγορίθμου-Επίλυση εξίσωσης Laplace

Για την καλύτερη εξήγηση και την πιο εύκολη κατανόηση των επιμέρους τμημάτων και των δυνατοτήτων των GPU θα χρησιμοποιηθεί ένα απλό παράδειγμα. Το παράδειγμα που επιλέχθηκε είναι η επίλυση της διδιάστατης εξίσωσης Laplace. Όπως είναι γνωστό, η εξίσωση Laplace, στη διδιάστατη μορφή της, γράφεται:

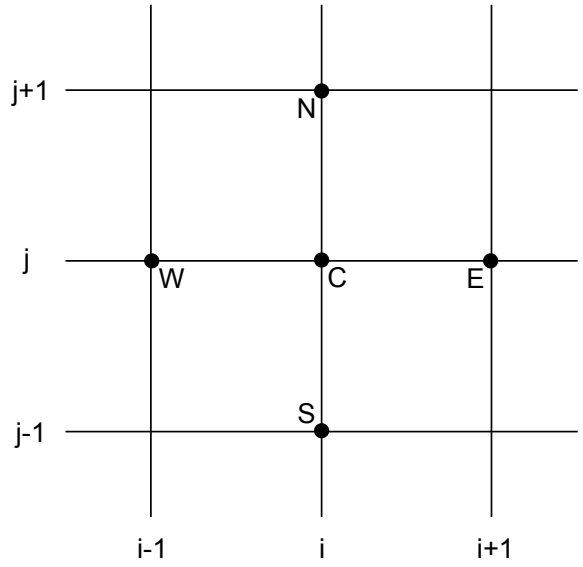
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0 \tag{2.1}$$

[1]Οι GPU αρχιτεκτονικής CUDA ακολουθούν τα πρότυπα της IEEE για την εκτέλεση πράξεων κινητής υποδιαστολής.

Για την αριθμητική επίλυση της εξίσωσης 2.1 χρειάζεται αριθμητικό πλέγμα, πάνω στο οποίο θα λυθεί η εξίσωση σύμφωνα με την διακριτοποιημένη μορφή της. Υιοθετείται ένα τετραγωνικό χωρίο, δημιουργείται σε αυτό ομοιόμορφο πλέγμα 500×500 κόμβων, με τετραγωνικές κυψέλες και, σε αυτό, η διακριτοποιημένη μορφή της εξίσωσης 2.1 είναι η:

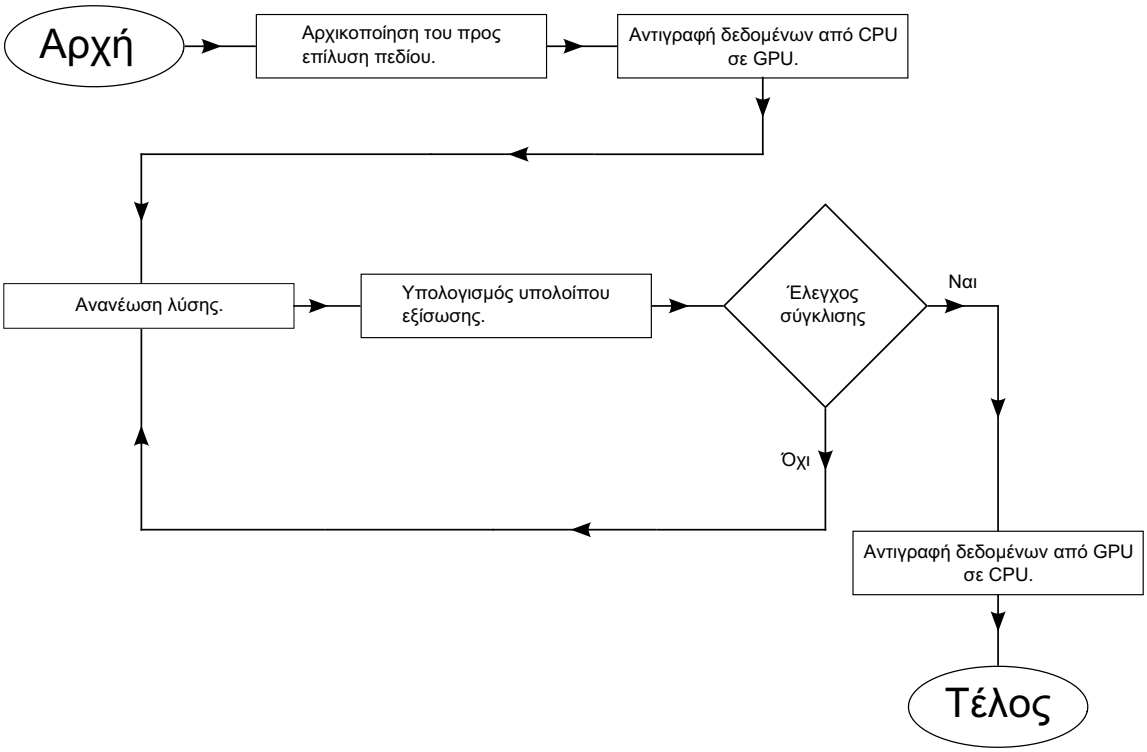
$$f_{i,j}^{NEW} = \frac{f_{i+1,j}^{OLD} + f_{i,j+1}^{OLD} + f_{i-1,j}^{OLD} + f_{i,j-1}^{OLD}}{4} = \frac{f_{E}^{OLD} + f_{N}^{OLD} + f_{W}^{OLD} + f_{S}^{OLD}}{4} \tag{2.2}$$

Όπου E, W, N, S συμβολίζουν τους άμεσους γείτονες χρησιμοποιώντας δείχτες γεωγραφικού προσανατολισμού (East, West, North, South) και φαίνονται και στο σχήμα 2.1.



Σχήμα 2.1: Σχετική θέση γειτονικών κόμβων.

Ο αλγόριθμος της αριθμητικής επίλυσης της εξίσωσης Laplace από την GPU αρχικοποιεί το πεδίο της συνάρτησης f στους κόμβους του αριθμητικού πλέγματος και αντιγράφει τον πίνακα στον οποίο είναι αποθηκευμένη η συνάρτηση f από την CPU στην GPU. Στην GPU, ξεκινά η επαναληπτική διαδικασία επίλυσης, η οποία συμπεριλαμβάνει την επαναληπτική ανανέωση της λύσης σε κάθε εσωτερικό κόμβο του πλέγματος σύμφωνα με την εξίσωση 2.2 και υπολογισμό του υπολοίπου της εξίσωσης όλων των κόμβων. Όταν το κριτήριο σύγκλισης του αλγορίθμου ικανοποιηθεί, τότε θα σταματήσει η επαναληπτική διαδικασία επίλυσης και η λύση της εξίσωσης πρέπει να αντιγραφεί από τη GPU στη CPU, για εκτύπωση. Ο αλγόριθμος αυτός φαίνεται και στο λογικό διάγραμμα του σχήματος 2.2.



Σχήμα 2.2: Λογικό διάγραμμα αλγορίθμου επίλυσης της εξίσωσης Laplace σε απλό υπολογιστικό πλέγμα, με χρήση GPU.

2.2 Αρχιτεκτονική CUDA

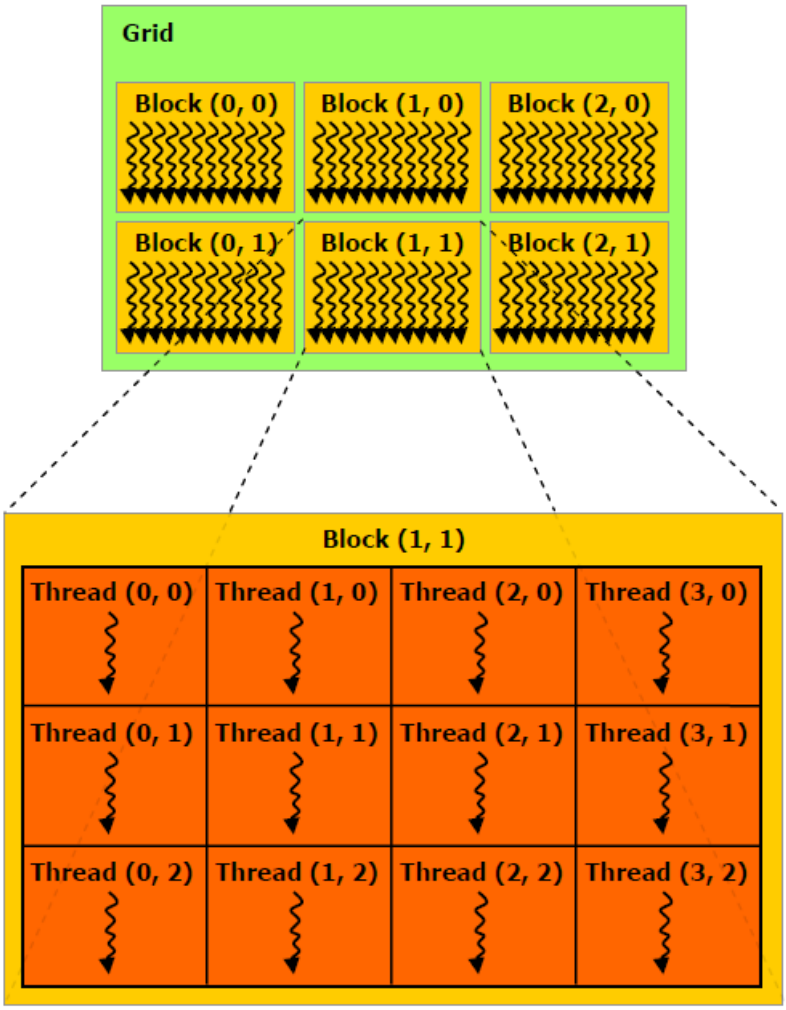
Στην CUDA, ο προγραμματιστής έχει τη δυνατότητα να προγραμματίσει «υπορουτίνες», οι οποίες δεν πρόκειται να εκτελεστούν από την CPU, αλλά από την GPU. Αυτές οι «υπορουτίνες» ονομάζονται kernels (πυρήνες). Η αλληλουχία εντολών ενός kernel εκτελείται παράλληλα από την GPU για έναν προκαθορισμένο αριθμό εκτελέσεων. Καθεμιά από τις εκτελέσεις του ίδιου kernel την αναλαμβάνει ένα CUDA thread (νήμα). Τα threads, για καλύτερη διαχείριση, ομαδοποιούνται σε blocks (δομικά στοιχεία). Τα threads εσωτερικά ενός block μπορεί να έχουν μονοδιάστατη, διδιάστατη ή και τριδιάστατη κατανομή. Όμως, λόγω φυσικών περιορισμών, ένα block μπορεί να αποτελείται μέχρι και από 512 threads (για GPU υπολογιστικής ικανότητας $1.x$) ή 1024 threads (για GPU υπολογιστικής ικανότητας $2.x$) ανεξάρτητα από το είδος κατανομής.

Στη συνέχεια, τα blocks οργανώνονται, μονοδιάστατα, διδιάστατα ή τριδιάστατα, και το σύνολο των blocks αποτελεί το grid (πλέγμα). Ένα grid μπορεί να αποτελείται μέχρι και από 65535 blocks ανά διάσταση. Δηλαδή, ένα grid με μονοδιάστατη κατανομή blocks, μπορεί να αποτελείται μέχρι και από 65535 blocks, με διδιάστατη κατανομή μέχρι και 65535×65535 blocks ενώ για τριδιάστατη κατανομή μέχρι και $65535 \times 65535 \times 65535$ blocks. Η επιλογή του τρόπου οργάνωσης των threads ανά block και των blocks ανά grid, η οποία μπορεί να είναι μονοδιάστατη, διδιάστατη

ή τριδιάσταση, είναι επιλογή του προγραμματιστή, σύμφωνα με τις ανάγκες κάθε εφαρμογής. Έτσι, στην τελική κατανομή των threads, για τον ακριβή καθορισμό της θέσης ενός thread χρειάζονται δύο στοιχεία, η τοπική αρίθμηση μέσα στο block και η αρίθμηση του block μέσα στο grid. Παράδειγμα κατανομής threads - blocks - grid φαίνεται στο σχήμα 2.3.

Το παράδειγμα της παραγράφου 2.1 θα μπορούσε να αποτελείται από 2 kernels, ένα που θα αναλάμβανε την ανανέωση λύσης κάθε κόμβου του πλέγματος, σύμφωνα με τη σχέση 2.2, και ένα που θα υπολόγιζε το υπόλοιπο της εξίσωσης, συνολικά, για τους κόμβους του πλέγματος των οποίων τα αντίστοιχα threads ανήκουν στο ίδιο block. Έχει επιλεγεί, κάθε thread να αντιστοιχεί σε έναν κόμβο του πλέγματος. Για την επίλυση του συγκεκριμένου προβλήματος, ο προγραμματιστής θα μπορούσε να επιλέξει διδιάστατη κατανομή threads ανά block και διδιάστατη κατανομή blocks στο grid. Ακόμα πρέπει να αποφασίσει για τον αριθμό των threads ανά block. Μερικές πιθανές αποφάσεις θα μπορούσαν να είναι το κάθε block να αποτελείται από $8 \times 8 = 64$ threads, από $16 \times 16 = 256$ threads ή οποιοδήποτε άλλο συνδιασμό, με την προϋπόθεση τα threads ανά block να είναι μέσα στα επιτρεπτά όρια. Όμως, για καλύτερη απόδοση, είναι σημαντικό το πλήθος των threads ανά block να είναι πολλαπλάσιο του 32. Επιλέγοντας την κατανομή 16×16 threads ανά block, το τελικό grid θα αποτελούνταν από $(\frac{500}{16}+1) \times (\frac{500}{16}+1)$ blocks. Όπως εύκολα φαίνεται, το προγραμματιστικό grid είναι μεγαλύτερο από το πλέγμα επίλυσης της εξίσωσης, κάτι το οποίο είναι συνηθισμένο κατά την επίλυση προβλημάτων με χρήση GPU, αλλά όχι επιθυμητό. Κατά την εκτέλεση του αλγορίθμου, όλα τα threads που ανήκουν στο προγραμματιστικό grid αλλά όχι στο υπολογιστικό πλέγμα παραμένουν ανενεργά και αποτελούν «σπατάλη» υπολογιστικής ισχύος της GPU.

Μετά την οργάνωση των threads ακολουθεί η εκτέλεσή τους. Τα blocks στέλνονται στους streaming multiprocessors ή SM όπου και εκτελούνται παράλληλα. Η λειτουργία κάθε SM αναλύεται στην παράγραφο 2.3. Σε κάθε SM μπορούν να εκτελούνται παράλληλα μέχρι και 8 blocks, λόγω φυσικών περιορισμών. Το πλήθος των blocks που θα εκτελεσθούν παράλληλα σε κάθε SM είναι συνάρτηση των απαιτήσεων κάθε thread σε πόρους, δηλαδη σε shared μνήμη και σε registers και του πλήθους των threads ανά block. Τα επιμέρους τμήματα της GPU, που σε αυτό το σημείο αναφέρονται απλά ονομαστικά, εξηγούνται στη συνέχεια. Σε κάθε SM, όπως θα αναφερθεί και στη συνέχεια, τα threads του κάθε block χωρίζονται σε ομάδες των 32. Αυτό συμβαίνει επειδή η GPU είναι σχεδιασμένη με τέτοιο τρόπο ώστε να εκτελεί τα threads πάντα σε ομάδες των 32. Κάθε τέτοια ομάδα ονομάζεται warp (στημόνι). Κάθε thread που ανήκει στο ίδιο warp εκτελεί τις εντολές που του αναλογούν συγχρόνως με όλα τα άλλα υπόλοιπα threads του ίδιου warp. Αντιθέτως, warps του ίδιου block εκτελούνται παράλληλα μέσα σε έναν SM, αλλά ασύγχρονα. Το ότι τα threads ενός warp εκτελούν εντολές σύγχρονα σημαίνει ότι και τα 32 threads ενός warp εκτελούν ταυτόχρονα τις ίδιες εντολές σε διαφορετικά δεδομένα. Σε περίπτωση που ένα ή περισσότερα threads του ίδιου warp χρειαστεί να εκτελέσουν κάποια εντολή διαφορετική από τα υπόλοιπα, όπως θα μπορούσε να συμβεί από την χρήση μίας εντολής if, τότε, ενώ εκτελείται η διαφορετική εντολή, όλα τα υπόλοιπα threads περιμένουν ανενεργά μέχρι την ολοκλήρωση αυτής. Από την άλλη, ανάμεσα σε δύο διαφορετικά warps, που



Σχήμα 2.3: Παράδειγμα ενός grid διδιάστατης κατανομής block, με threads διδιάστατης κατανομής εσωτερικά του κάθε block. Όπως φαίνεται και στο σχήμα, κάθε thread έχει τοπική αρίθμηση μέσα στο block που βρίσκεται και κάθε block έχει αρίθμηση σχετική με το συνολικό grid. Αφού πρόκειται για διδιάστατη αρίθμηση, τόσο για τα blocks όσο και για τα threads, για τον καθορισμό της θέσης ενός block μέσα στο grid ή για τον καθορισμό της τοπικής θέσης ενός thread μέσα σε ένα block χρειάζονται 2 δείκτες. Αντίστοιχα, σε μονοδιάστατη κατανομή θα χρειάζονταν ένας δείκτης, ενώ σε τριδιάστατη κατανομή τρεις. Στο παράδειγμα του σχήματος, για τον πλήρη καθορισμό της θέσης ενός thread στο grid χρειάζονται συνολικά 4 δείκτες, όπως εύκολα προκύπτει από τα παραπάνω, 2 για τον καθορισμό της τοπικής θέσης του thread στο block στο οποίο ανήκει και 2 για τον καθορισμό της θέσης του block αυτού στο grid.

εκτελούνται παράλληλα και ασύγχρονα, δεν είναι απαραίτητο να εκτελείται η ίδια εντολή του kernel και από τα δύο ταυτόχρονα. Ολόκληρη η παραπάνω διαδικασία αποτελεί εσωτερική διαδικασία της GPU όπου ο προγραμματιστής δεν μπορεί να επέμβει.

Σε αυτό το σημείο γίνεται κατανοητό το γιατί το πλήθος των threads ανά block είναι επιθυμητό να είναι πολλαπλάσιο του 32. Σε αντίθετη περίπτωση θα προέκυπταν, στο τέλος κάθε block, warps που θα αποτελούνταν από λιγότερα των 32 threads, με αποτέλεσμα να έμεναν ανενεργοί πόροι της GPU. Για παράδειγμα, αν σε ένα block αντιστοιχούν 100 threads, τότε κατά την εκτέλεση του κάθε block θα εκτελεστούν 3 warps αποδοτικά και 1 warp με 4 ενεργά threads και 28 ανενεργές θέσεις επεξεργασίας.

Ακόμα, σε αρκετές εφαρμογές, είναι απαραίτητη η επικοινωνία μεταξύ threads τα οποία εκτελούνται παράλληλα. Για threads που ανήκουν στο ίδιο block μπορεί να χρησιμοποιηθεί η shared μνήμη, η οποία είναι προσβάσιμη από όλα τα threads του ίδιου block. Για threads που δεν ανήκουν στο ίδιο block απαιτείται η χρήση της global μνήμης. Ένα τέτοιο παράδειγμα είναι ο υπολογισμός του υπολοίπου της εξίσωσης κατά της επίλυση της εξίσωσης Laplace. Για τον υπολογισμό αυτό, το κάθε block αναλαμβάνει την άθροιση των υπολοίπων των threads που του αντιστοιχούν. Για την άθροιση, το κάθε thread φορτώνει την τιμή του υπολοίπου που του αντιστοιχεί στην shared μνήμη και, στη συνέχεια, το πρώτο thread κάθε block αναλαμβάνει την άθροιση των επιμέρους υπολοίπων, αφού πλέον, μέσω της shared μνήμης, αυτά του είναι γνωστά. Αυτός δεν είναι ο πιο αποδοτικός τρόπος άθροισης αλλά είναι απλός και εύκολα προγραμματίσιμος. Ένας πιο αποδοτικός τρόπος άθροισης παρουσιάζεται στην παράγραφο 4.3.7.

Επίσης, υπάρχει η δυνατότητα για συγχρονισμό των threads εσωτερικά ενός block, με χρήση μίας εντολής. Αυτό σημαίνει πως πρώτα θα ολοκληρωθούν όλες οι εντολές του kernel από όλα τα threads ενός block, θα γίνει ο συγχρονισμός τους και, μόνο τότε, θα συνεχίσει η εκτέλεση των επόμενων εντολών από τα threads του block. Όταν απαιτειτα συγχρονισμός ολόκληρου του grid, η μόνη λύση, σύμφωνα με την παρούσα τεχνολογία, είναι ο διαχωρισμός του kernel σε 2 ξεχωριστά kernels. Αυτό συμβαίνει επειδή, από τη μία, πριν την εκκίνηση ενός kernel έχει ολοκληρωθεί η εκτέλεση όλων των προηγούμενων kernel, οπότε πρακτικά έχει επιτευχθεί ο απαιτούμενος συγχρονισμός και, από την άλλη, δεν υπάρχει κάποια εντολή που να επιτρέπει το συγχρονισμό σε επίπεδο grid (κατά την διάρκεια εκτέλεσης ενός kernel) επειδή είναι πολύ δύσκολο να ενσωματωθεί μια τέτοια λειτουργία στις GPU. Για την άθροιση του υπολοίπου της εξίσωσης, που αναφέρθηκε στην προηγούμενη παράγραφο, έγινε χρήση της shared μνήμης. Πριν το πρώτο thread κάθε block ξεκινήσει την άθροιση του υπολοίπου πρέπει να είναι σίγουρο πως όλα τα άλλα threads του ίδιου block έχουν ολοκληρώσει το γράψιμο στην shared μνήμη. Οπότε σε αυτό το σημείο χρειάζεται να γίνει συγχρονισμός του block.

2.3 Η Δομή μιας GPU αρχιτεκτονικής CUDA.

Οι GPUs αποτελούνται από ένα πλήθος πολυεπεξεργαστών, τους SM. Πόσους SM έχει η κάθε GPU εξαρτάται από το μοντέλο της. Όμως σε κάθε GPU, ο SM είναι η βασική μονάδα παράλληλης επεξεργασίας δεδομένων. Ο κάθε SM έχει στη διάθεσή του ένα μεγάλο αριθμό πυρήνων (CUDA cores), έναν ή δύο Warp schedulers (προγραμματιστές) και άλλους τόσους Instruction Dispatch Unit (μονάδες αποστολής

εντολών), επίσης ανάλογα με το μοντέλο. Ακόμα, κάθε SM έχει πρόσβαση σε κάποιες μνήμες περιορισμένης χρήσης, οι οποίες ειναι η local, η shared καθώς και η L1 μνήμη, σε δύο ή τέσσερις μονάδες ειδικών συναρτήσεων (Special Function Unit ή SFU) και σε μερικές μονάδες φόρτωσης/αποθήκευσης δεδομένων (Load/Store Units). Επίσης, κάθε SM έχει ένα συγκεκριμένο αριθμό registers (καταχωρητών). Ακόμα, όλοι οι SM της GPU έχουν πρόσβαση στην ενιαία μνήμη RAM, που ονομάζεται global μνήμη και στην texture μνήμη, την constant μνήμη και την L2 μνήμη η οποίες είναι περιορισμένης χρήσης, δηλαδή η χρήση τους συμφέρει όταν ικανοποιούνται συγκεκριμένες συνθήκες. Όλα τα παραπάνω αναλύονται στη συνέχεια. Στο σχήμα 2.4 φαίνεται συνολικά μία GPU αρχιτεκτονικής Fermi [20], στο σχήμα 2.5 φαίνεται ένας SM τρίτης γενιάς, όπως αυτού που χρησιμοποιούνται στις GPU αρχιτεκτονικής Fermi, ενώ στο σχήμα 2.6 παρουσιάζονται συνοπτικά τα τεχνικά χαρακτηριστικά των CUDA GPUs ανάλογα με την κάθε αρχιτεκτονική.

Κατά την παρουσίαση των διάφορων αρχιτεκτονικών CUDA θα δοθεί ιδιαίτερη έμφαση στις GPU αρχιτεκτονικής Fermi, καθώς ο επιλύτης που αναπτύχθηκε βασίζεται πάνω σε αυτή.

2.3.1 Πυρήνας CUDA

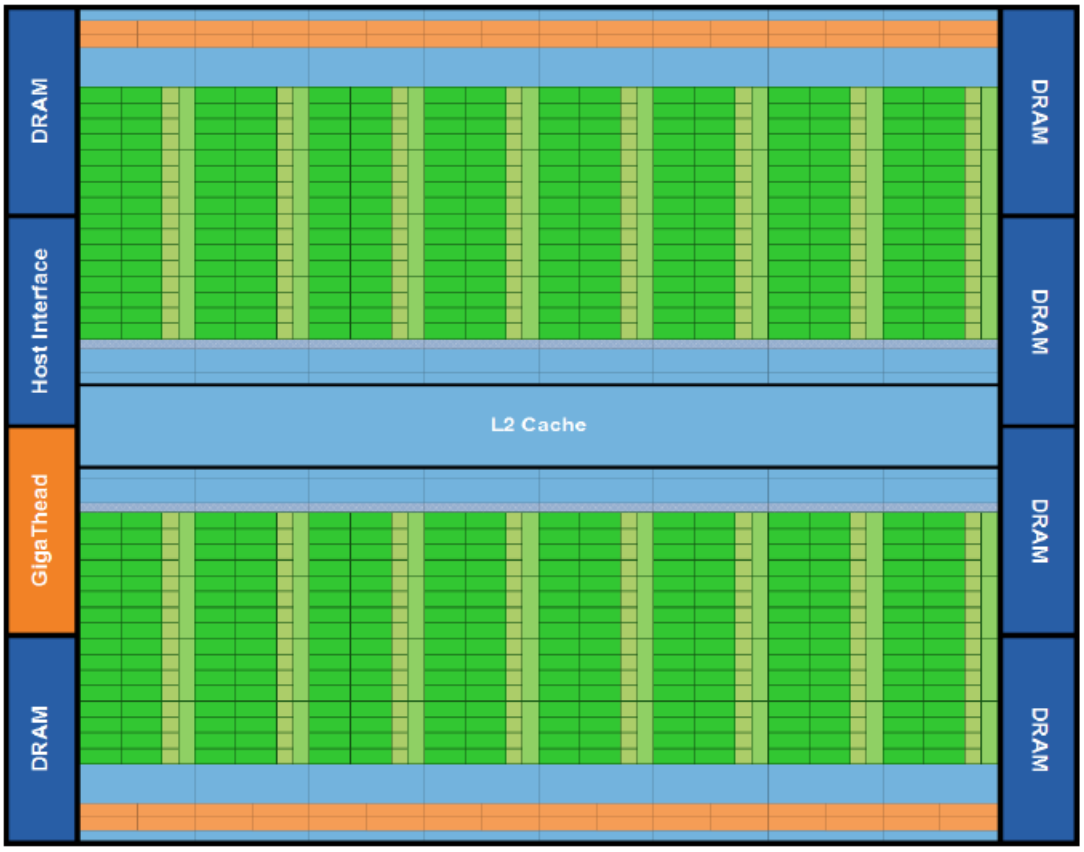
Ο κάθε πυρήνας CUDA είναι εφοδιασμένος με ανεξάρτητες Arithmetic Logic Units ή ALU (μονάδες αριθμητικής και λογικών πράξεων) και Floating Point Units ή FPU (μονάδες πράξεων κινητής υποδιαστολής). Στις μονάδες ALU εκτελούνται οι πράξεις μεταξύ ακεραίων και οι λογικές πράξεις ενώ στις μονάδες FPU εκτελούνται οι πράξεις κινητής υποδιαστολής, μονής και διπλής ακρίβειας. Οι GPU αρχιτεκτονικής Fermi για τις πράξεις κινητής υποδιαστολής χρησιμοποιούν το πρότυπο IEEE 756-2008, ενώ στις προηγούμενες αρχιτεκτονικές (G80 και GT200) χρησιμοποιούνταν το πρότυπο IEEE 754-1985.

2.3.2 Warp Scheduler & Instruction Dispatch Unit

Σε κάθε SM ομαδοποιούνται τα threads του κάθε block σε ομάδες των 32, που ονομάζονται warps. Ο Warp Scheduler επιλέγει τη σειρά με την οποία θα εκτελεστούν τα warps και, στη συνέχεια, το Instruction Dispatch Unit αναλαμβάνει τη μεταφορά των εντολών προς εκτέλεση σε κάθε CUDA core. Είναι σημαντικό να τονιστεί ξανά πως εσωτερικά ενός warp οι εντολές εκτελούνται σύγχρονα ανάμεσα στα threads. Αντίθετα, δύο threads του ίδου block, τα οποία όμως ανήκουν σε διαφορετικά warps, εκτελούνται παράλληλα αλλά ασύγχρονα, όπως αναφέρεται στην παράγραφο 2.2.

2.3.3 Load/Store Units

Οι μονάδες φόρτωσης / αποθήκευσης δεδομένων υπολογίζουν σε ποια θέση μνήμης απευθύνεται κάθε thread κάθε φορά που χρειάζεται να προσπελαθεί κάποια μνήμη, είτε για διάβασμα δεδομένων από αυτή, είτε για γράψιμο αποτελεσμάτων πράξεων

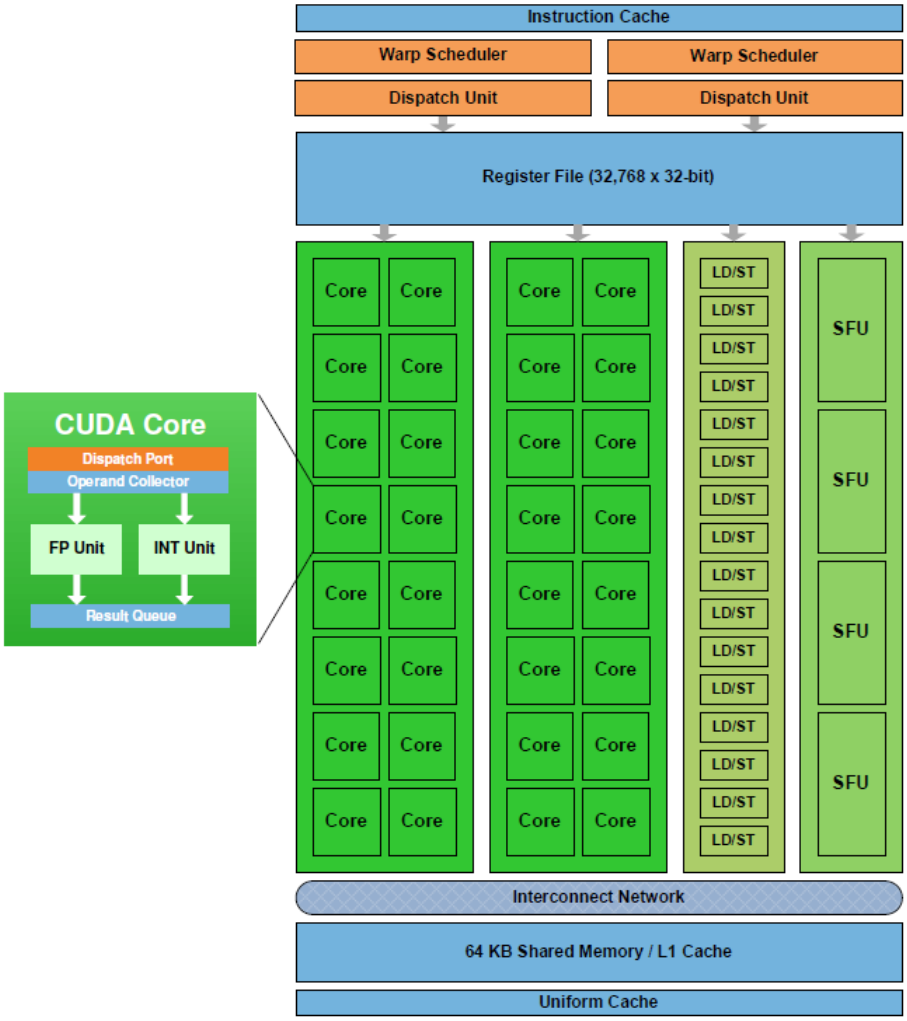


Σχήμα 2.4: Κάθε GPU αρχιτεκτονικής Fermi έχει 16 SM στοιχειοθετημένους γύρω από την μνήμη L2. Κάθε SM μοιάζει με το παραπάνω παραλληλόγραμμο και αποτελείται από ένα πορτοκαλί τμήμα, το οποίο αντιστοιχεί στους Warp Schedulers και στους Instruction Dispatch Units, ένα πράσινο τμήμα όπου βρίσκονται οι μονάδες για τις εκτελέσεις των πράξεων καθώς και από το γαλάζιο τμήμα, το οποίο αντιστοιχεί στους registers και στην Shared/L1 μνήμη.

σε αυτή. Πίσω από τις μονάδες φόρτωσης / αποθήκευσης υπάρχουν δευτερεύουσες μονάδες που αναλαμβάνουν το καθαυτό διάβασμα ή γράψιμο της μνήμης.

2.3.4 Special Function Units

Οι μονάδες ειδικών συναρτήσεων προσφέρουν την δυνατότητα μερικές συνηθισμένες συναρτήσεις, όπως το ημίτονο, το συνημίτονο και η τετραγωνική ρίζα, να εκτελούνται πολύ πιο γρήγορα, αλλά με μειωμένη μαθηματική ακρίβεια. Οπότε, σε εφαρμογές όπου η ταχύτητα είναι σημαντικότερη από την ακρίβεια, μπορούν να χρησιμοποιηθούν οι μονάδες ειδικών συναρτήσεων για καλύτερους χρόνους εκτέλεσης. Όμως, η ακρίβεια είναι πάρα πολύ σημαντική στην υπολογιστική ρευστοδυναμική οπότε δεν έχουν χρησιμοποιηθεί καθόλου οι μονάδες ειδικών συναρτήσεων στον επιλύτη που αναπτύχθηκε εδώ.



Σχήμα 2.5: Σχηματική απεικόνιση ενός SM αρχιτεκτονικής Fermi.

2.3.5 Διαθέσιμες Μνήμες

Global μνήμη

Η βασική μνήμη της GPU είναι η global μνήμη. Η global μνήμη είναι μεγάλης χωρητικότητας και είναι προσπελάσιμη από όλα τα threads ενός kernel, άσχετα με το block στο οποίο ανήκουν. Στην global μνήμη αποθηκεύεται σχεδόν κάθε πληροφορία, μεταβλητή, πίνακας, ή ότι άλλο χρησιμοποιειται σε ένα πρόγραμμα. Κατά την επίλυση της εξίσωσης Laplace, στην global μνήμη είναι αποθηκευμένη η λύση της εξίσωσης καθως και το άθροισμα του υπόλοιπου της εξίσωσης των threads που ανήκουν σε κάθε block. Ακόμα, θα μπορούσε να είναι αποθηκευμένοι και οι πίνακες με το υπολογιστικό πλέγμα εάν χρειαζόντουσαν, όπως λ.χ. αν το πλέγμα δεν αποτελούνταν από τετραγωνικές κυψέλες.

Το βασικό μειονέκτημά της είναι η αργή ταχύτητα προσπέλασής της. Για α-νάπτυξη γρήγορων εφαρμογών στις GPU [21] είναι πολύ σημαντικό η προσπέλαση

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Σχήμα 2.6: Τεχνικά χαρακτηριστικά κάθε αρχιτεκτονικής CUDA. Υπενθιμίζεται ότι σκοπός της παρούσας διπλωματικής εργασίας είναι η εκμετάλλευση των δυνατοτήτων των GPU αρχιτεκτονικής Fermi σε εφαρμογές υπολογιστικής ρευστοδυναμικής.

της global μνήμης να γίνεται με βέλτιστο τρόπο, καθώς κάθε μη-βελτιστοποιημένη προσπέλαση μειώνει αισθητά την τελική επιτάχυνση της εφαρμογής. Εκτεταμένη χρήση της global μνήμης οδηγεί συνήθως σε φαινόμενο «λαιμού» (bottleneck effect) καθώς τα δεδομένα δεν προωθούνται αρκετά γρήγορα στους πυρήνες για επεξεργασία, με αποτέλεσμα να μην αξιοποιείται επαρκώς η υπολογιστική δύναμη της GPU. Μία πολύ σημαντική δυνατότητα της global μνήμης είναι η ενοποιημένη προσπέλαση μνήμης (coalesced memory transactions), δηλαδή η δυνατότητα, όταν ένα warp απαιτεί προσπέλαση της global μνήμης, η προσπέλαση να μην γίνει μία φορά για κάθε thread του warp, αλλά να ενοποιηθούν οι προσπελάσεις σε λιγότερες προσπελάσεις. Με κατάλληλο χειρισμό οι προσπελάσεις μνήμης ενός warp μπορούν να ενοποιηθούν μέχρι και σε 1 προσπέλαση της global μνήμης. Ο μηχανισμός πίσω από την ενοποίηση των προσπελάσεων μνήμης είναι άμεσα συνδεδεμένος με την υπολογιστική ικανότητα της GPU.

Σε GPU υπολογιστικής ικανότητας 1.0 και 1.1, ενοποιούνται οι προσπελάσεις μνήμης όταν για κάθε **μισό warp** τηρούνται τα παρακάτω:

- Το μέγεθος κάθε word (καταχώρηση στη μνήμη προς προσπέλαση) πρέπει να είναι 4, 8 ή 16 bytes, όπως, για παράδειγμα, μεταβλητές τύπου float, double και double2[2] αντίστοιχα.

[2]Οι μεταβλητές double2 είναι προγραμματιστικές μεταβλητές της CUDA C όπου σε κάθε τέτοια μεταβλητή αποθηκεύονται 2 μεταβλητές τύπου double. Στον επιλύτη που αναπτύχθηκε δεν χρησιμοποιήθηκαν μεταβλητές τέτοιου είδους και εδώ αναφέρονται απλά ως παράδειγμα μεταβλητών μεγέθους 16 bytes.

- Ανάλογα με το μέγεθος κάθε word ισχύει:
  - Όταν κάθε word είναι 4 bytes πρέπει και τα 16 words να βρίσκονται στο ίδιο 64-byte τμήμα της μνήμης,
  - Όταν κάθε word είναι 8 bytes πρέπει και τα 16 words να βρίσκονται στο ίδιο 128-byte τμήμα της μνήμης,
  - Όταν κάθε word είναι 16 bytes πρέπει και τα πρώτα 8 words να βρίσκονται στο ίδιο 128-byte τμήμα της μνήμης και τα επόμενα 8 words να βρίσκονται στο αμέσως επόμενο 128-byte τμήμα της μνήμης.
- Τα threads και οι θέσεις μνήμης προς προσπέλαση πρέπει να είναι αντιστοιχισμένα στη σειρά, δηλαδή το ν-ιοστο thread να αντιστοιχεί στη ν-ιοστή θέση μνήμης, σε κάθε μισό warp.

Όταν ισχύουν τα παραπάνω, για κάθε μισό warp εκτελούνται ή μία 64-byte εντολή προσπέλασης μνήμης, ή μία 128-byte εντολή προσπέλασης μνήμης, ή δύο 128-byte εντολές προσπέλασης μνήμης αν το μήκος κάθε word είναι 4, 8 ή 16 bytes, αντίστοιχα. Ενοποίηση επιτυγχάνεται ακόμα και αν κάποιο thread είναι ανενεργό και δεν εκτελεί εντολή προσπέλασης μνήμης ταυτόχρονα με τα υπόλοιπα. Είναι σημαντικό να τονιστει πως τα 64 και 128-byte τμήματα που αναφέρθηκαν παραπάνω, δεν είναι οποιεσδήποτε 16 συνεχόμενες θέσης μνήμης, μεγέθους 64 bytes όταν το κάθε word είναι 4 bytes ή μεγέθους 128 όταν το κάθε word είναι 8 bytes, ή οποιεσδήποτε 8 συνεχόμενες θέσης μνήμης όταν το κάθε word είναι 16 bytes, αλλά είναι προκαθορισμένα τμήματα μνήμης και ορισμένα στη σειρά από την αρχή της δεσμευμένης μνήμης έως το τέλος της. Για παράδειγμα, σε ένα μονοδιάστατο πίνακα που αποτελείται απο αριθμούς διπλής ακρίβειας (double), όπου ο κάθε ένας αριθμός διπλής ακρίβειας έχει μέγεθος 8 bytes, το πρώτο τμήμα της μνήμης αντιστοιχεί στις θέσεις του πίνακα 0 έως 15, το δεύτερο τμήμα στις θέσεις 16 έως 31, το τρίτο στις θέσεις 32 έως 47 κτλ. Όπως φαίνεται, το κάθε τμήμα της μνήμης αποτελείται από 16 μεταβλητές, όπου η καθεμία έχει μέγεθος 8 bytes, και έχει συνολικό μέγεθος 128 bytes.

Αν δεν τηρούνται τα παραπάνω, εκτελούνται 16 ξεχωριστές 32-byte εντολές προσπέλασης μνήμης με αποτέλεσμα να επιβραδυνθεί σημαντικά ο συνολικός χρόνος προσπέλασης όλων των threads.

Σε GPU υπολογιστικής ικανότητας 1.2 και 1.3, ενοποιούνται οι προσπελάσεις μνήμης όταν όλα τα threads ανά **μισό warp** απλά να απευθύνονται σε θέσεις μνήμης που ανήκουν στο ίδιο τμήμα της μνήμης, ανεξάρτητα με τη σειρά, ακόμα και αν 2 ή περισσότερα threads απευθύνονται στην ίδια θέση μνήμης. Πιο συγκεκριμένα, για τον προσδιορισμό του αριθμού των εντολών που πρέπει να εκτελεστούν για την προσπέλαση της μνήμης ακολουθείται η παρακάτω διαδικασία:

- Πρώτα καθορίζεται το μέγεθος της προσπελάσιμης μνήμης, ανά εντολή προσπέλασης μνήμης, βάσει του πρώτου ενεργού thread, ανά μισό warp. Το μέγεθος είναι συνάρτηση του μεγέθους των words που χειριζονται τα threads και είναι:
  - 32 bytes όταν κάθε word έχει μέγεθος 1-byte,

  - 64 bytes όταν κάθε word έχει μέγεθος 2-bytes,
  - 128 bytes όταν κάθε word έχει μέγεθος 4-, 8- ή 16-bytes.

- Στη συνέχεια εντοπίζονται όλα τα υπόλοιπα ενεργά threads που απευθύνονται σε θέσεις μνήμης του ίδιου τμήματος της μνήμης.

- Το μέγεθος της εντολής προσπέλασης μειώνεται ακόμα περισσότερο, αν αυτό είναι δυνατό, σύμφωνα με τα παρακάτω:

  - Άν το μέγεθος της προσπελάσιμης μνήμης, ανά εντολή προσπέλασης, είναι 128 bytes και χρησιμοποιείται μόνο το πρώτο μισό, ή μόνο το δεύτερο μισό, το μέγεθος της εντολής μειώνεται στα 64 bytes.
  - Άν το μέγεθος της προσπελάσιμης μνήμης, ανά εντολή προσπέλασης, είναι 64 bytes, είτε αυτό είναι το αρχικό της μέγεθος, είτε αυτό προήλθε από την παραπάνω περίπτωση, και όμοια με πριν χρησιμοποιείται μόνο το μισό από της εύρος, τότε μειώνεται σε εντολή μεγέθους 32 bytes.

- Κάθε thread που εκτελεί την προσπέλαση μνήμης που του αναλογεί σημειώνεται ως ανενεργό μόλις ολοκληρωθεί η προσπέλαση της μνήμης.

- Ο παραπάνω κύκλος επαναλαμβάνεται μέχρι να σημειωθούν όλα τα threads ώς ανενεργά.

Σε GPU υπολογιστικής ικανότητας 2.*x* το κανονικό μήκος εντολών προσπέλασης μνήμης είναι 128, και απευθύνεται σε **ολόκληρα warps** αντίθετα με τα προηγούμενα, όπου κάθε εντολή προσπέλασης μνήμης απευθυνόταν σε μισό warp. Ακόμα έχουν προστεθεί οι cache L1 και L2 μνήμες, όπου ο τρόπος χρήσης τους αναλύεται στην συνέχεια της παραγράφου. Όταν χρησιμοποιείται μόνο η L2 cache μνήμη ως βοηθητική, χωρίς την χρήση της L1, οι εντολές προσπέλασης μνήμης εκτελούνται με μήκος 32 bytes. Οπότε σε περιπτώσεις μεγάλης διασποράς των πληροφοριών στην μνήμη, η χρήση μόνο της L2 μνήμης συνήθως έχει θετικά αποτελέσματα.

Αν το μέγεθος κάθε word είναι μεγαλύτερο από 4 bytes, κάθε εντολή προσπέλασης της μνήμης διασπάται σε περισσότερες, όπου καθεμία εκτελείται ανεξάρτητα.

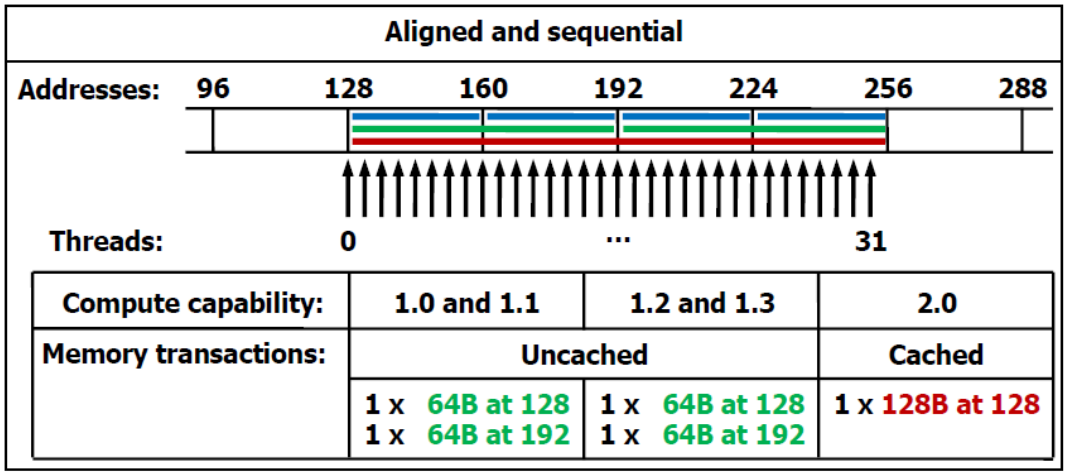
- Άν το μέγεθος κάθε word είναι 8 bytes εκτελούνται 2 εντολές προσπέλασης μνήμης μεγέθους 128 bytes, μία για κάθε μισό warp.

- Άν το μέγεθος κάθε word είναι 16 bytes εκτελούνται 4 εντολές προσπέλασης μνήμης μεγέθους 128 bytes, μία για κάθε τέταρτο του warp.

Κάθε εντολή προσπέλασης μνήμης διασπάται σε ανεξάρτητες εντολές προσπέλασης της cache μνήμης, οι οποίες εκτελούνται ξεχωριστά και ανεξάρτητα. Σε περίπτωση που οι πληροφορίες που απαιτεί κάποια εντολή βρίσκονται στην chache μνήμη διαβάζονται από εκεί, εναλλακτικά, εάν δεν βρεθούν εκεί, οι εντολές προσπέλασης προωθούνται στην global μνήμη.

Και στις GPU υπολογιστικής ικανότητας 2.x δεν έχει σημασία η σειρά με την οποία τα threads προσπελαύνουν τις θέσεις μνήμης, αλλά όπως και στις GPU υπολογιστικής ικανότητας 1.2 και 1.3, απλά να βρίσκονται στο ίδιο τμήμα μνήμης, ακόμα και αν 2 ή περισσότερα threads απευθύνονται στην ίδια θέση μνήμης.

Παραδείγματα με όλα τα παραπάνω φαίνονται στα σχήματα 2.7, 2.8 και 2.9.

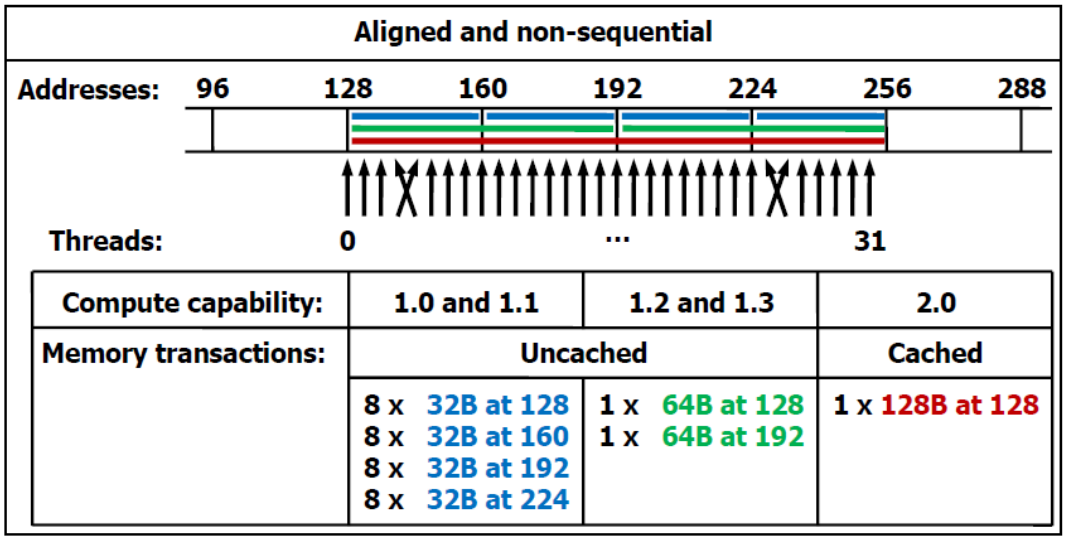


Σχήμα 2.7: Παράδειγμα προσπέλασης της global μνήμης από ένα warp, word μήκους 4 bytes συναρτήσει της υπολογιστικής ικανότητας μίας GPU, όταν η προσπέλαση γίνεται σε συνεχόμενες θέσης μνήμης ενός τμήματος της μνήμης και με σειρά. Το παράδειγμα αυτό αντιστοιχεί στο παράδειγμα της παραγράφου 2.1, όταν τα 32 πρώτα threads του grid προσπελαύνουν τις θέσεις μνήμης όπου είναι αποθηκευμένες οι τιμές f_N ή f_S, κατά την ανανέωση της λύσης, θεωρώντας των πίνακα f ως πίνακα μεταβλητών float. Έτσι, τα threads με αύξοντα αριθμό από 0 έως 31 προσπελαύνουν ένα τμήμα της μνήμης μεγέθους 128 bytes το οποίο αντιστοιχεί σε 32 μεταβλητές μεγέθους 4 byte καθεμιά. [1]

Constant και Texture μνήμες.

Για βελτίωση της απόδοσης υπάρχουν άλλες δύο μνήμες που είναι ορατές από ολόκληρο το kernel, αλλά έχουν πολύ μεγαλύτερες ταχύτητες προσπέλασης. Η μία ειναι η texture μνήμη και η άλλη είναι η constant μνήμη. Καθεμιά από αυτές έχει σχεδιαστεί διαφορετικά και βελτιώνει την τελική απόδοση κάθε εφαρμογής κάτω από διαφορετικές συνθήκες. Τα κοινά τους χαρακτηριστικά είναι πως και οι δύο είναι περιορισμένης χωρητικότητας, καθώς και «read-only memory», δηλαδή μνήμες που επιτρέπουν στον προγραμματιστή μόνο να διαβάζει από αυτές κατά την διάρκεια εκτέλεσης ενός kernel.

Στην constant μνήμη αποθηκεύονται πληροφορίες, από τον προγραμματιστή, πρίν την εκτέλεση ενός kernel και δεν μπορούν να αλλαχθούν κατά την εκτέλεσή του και ο αποθηκευτικός της χώρος είναι 64 KBytes. Υπό κατάλληλες συνθήκες, προσπέλαση



Σχήμα 2.8: Παράδειγμα προσπέλασης της global μνήμης από ένα warp, word μήκους 4 bytes συναρτήσει της υπολογιστικής ικανότητας μίας GPU, όταν η προσπέλαση γίνεται σε συνεχόμενες θέσης μνήμης, ενός τμήματος της μνήμης, αλλά όχι στη σειρά. Σε δομημένα προβλήματα, όπως το παράδειγμα της παραγράφου 2.1, δεν συναντάται τέτοιου είδους προσπέλαση μνήμης. [1]

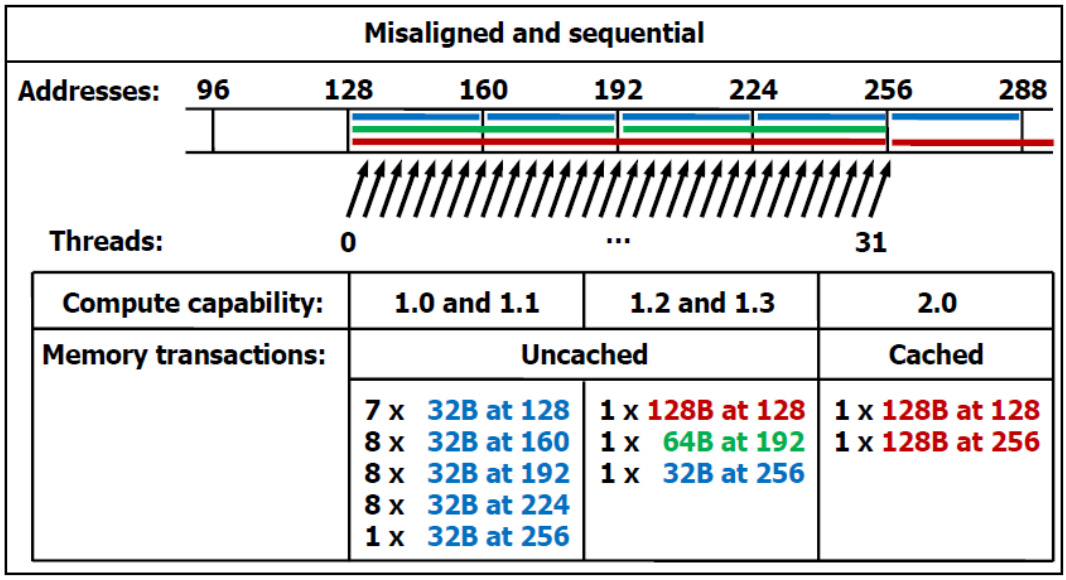
πληροφοριών με χρήση της constant μνήμης μπορεί να βελτιώσει τις επιδόσης μία εφαρμογής επειδή:

- Ένα «διάβασμα» από την constant μνήμη μπορεί προβληθεί σε όλα τα υπόλοιπα threads του ίδιου μισού warp, ουσιαστικά γλιτώνοντας 15 «διαβάσματα».

- Η constant μνήμη είναι cached μνήμη και έτσι συνεχόμενα «διαβάσματα» της ίδιας θέσης μνήμης δεν φορτώνει τη μνήμη με παραπάνω προσπελάσεις, αλλά τα διαβάζει από την cached μνήμη.

Όμως χρειάζεται προσοχή η χρήση της constant μνήμης καθώς όταν και τα 16 threads που απαρτίζουν μισό warp «διαβάζουν» την ίδια θέση μνήμης οι επιδόσεις βελτιώνονται, αλλά για να επιτευχθεί η δυνατότητα προβολής ενός «διαβάσματος» σε 16 threads, έχει αφερεθεί η δυνατότητα παράλληλων διαβασμάτων από την constant μνήμη. Έτσι, σε περίπτωση που κάθε ένα από τα 16 threads χρειάζεται πληροφορία από διαφορετική διεύθυνση της constant, τότε τα «διαβάσματα» θα εκτελεστούν σειριά και θα χρειαστεί 16-πλάσιος χρόνος για τη συνολική διεργασία. Σε τέτοιες περιπτώσεις, κατα πάσα πιθανότητα, το «διάβασμα» από την constant μνήμη θα είναι πιο αργό από ότι από την global μνήμη.

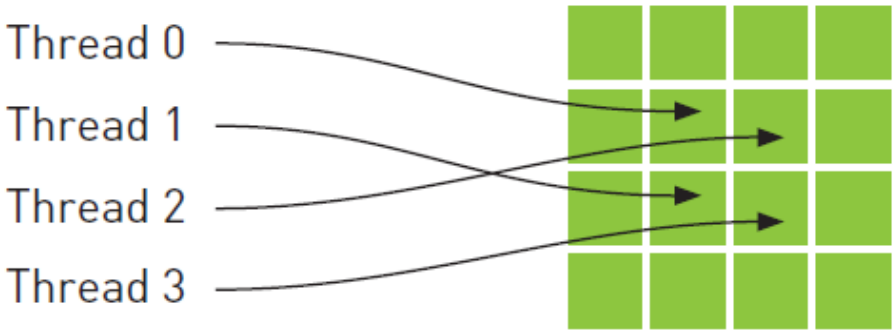
Στην constant μνήμη γενικά αποθηκεύονται οι σταθερές ενός προβλήματος. Έτσι, σε ένα πρόβλημα ρευστοδυναμικής θα μπορούσαν να αποθηκευτούν στην constant μνήμη η σταθερά του τελείου αερίου R_g, οι ειδικές θερμοχωρητικότητες c_p και c_v και ο εκθέτης ισεντροπικής μεταβολής γ.



Σχήμα 2.9: Παράδειγμα προσπέλασης της global μνήμης από ένα warp, word μήκους 4 bytes συναρτήσει της υπολογιστικής ικανότητας μίας GPU, όταν η προσπέλαση γίνεται σε συνεχόμενες θέσης μνήμης, στη σειρά, αλλά δεν ανήκουν σε ένα τμήμα της μνήμης. Η περίπτωση αυτή αντιστοιχεί στο παράδειγμα της παραγράφου 2.1, όταν τα 32 πρώτα threads του grid προσπελαύνουν τις θέσεις μνήμης όπου είναι αποθηκευμένες οι τιμές f_E ή f_W, κατά την ανανέωση της λύσης. Η ίδια μη-στοιχισμένη προσπέλαση μνήμης παρουσιάζεται, αντίστοιχα, και σε όλες της εφαρμογές που χρησιμοποιούνται δομημένα πλέγματα και δεν μπορεί να αποφευχθεί. [1]

Η texture μνήμη, αν και αρχικά σχεδιάστηκε για χρήση σε εφαρμογές επεξεργασίας γραφικών, είναι ιδιαίτερα αποδοτική και σε εφαρμογές γενικού προγραμματισμού. Η texture μνήμη αποτελεί και αυτή cached μνήμη και είναι σχεδιασμένη για εφαρμογές όπου τα μοτίβα προσπέλασης της μνήμης παρουσιάζουν έντονη χωρική τοπικότητα (spacial locality), δηλαδή όπου κάθε thread «διαβάζει» θέσεις μνήμης κοντά στις θέσεις μνήμης που διαβάζουν τα διπλανά του threads, όπως φαίνεται στο σχημα 2.10.

Η texture μνήμη, κατά την εκτέλεση ενός kernel, χρησιμοποιείται ως ενδιάμεση μνήμη, αφού είναι cached, για να μειωθούν οι προσπελάσεις στην global μνήμη. Για τη χρήση της texture μνήμης, ο προγραμματιστής το μόνο που έχει να κάνει είναι να απευθύνει την εντολή προσπέλασης της μνήμης στην texture μνήμη και όχι στην global μνήμη. Τότε, η GPU αναλαμβάνει να ελέγξει αν υπάρχει η επιθυμητή πληροφορία στην texture μνήμη. Αν υπάρχει, την «διαβάζει» από την texture μνήμη ενώ αν δεν υπάρχει αντιγράφει την επιθυμητή πληροφορία από την global μνήμη στην texture μνήμη, όπου και την αποθηκεύει για μελλοντική χρήση. Όμως, η texture μνήμη είναι μικρή σε χωρητικότητα (6 έως 8 KBytes ανά SM) και για κάθε καινούργια εγγραφή σε αυτή συνήθως χρειάζεται να διαγραφεί μία προηγούμενη καταχώρηση. Ο προγραμματιστής δεν μπορεί γνωρίζει τι είναι αποθηκευμένο κάθε χρονική στιγμή της εκτέλεσης ενός kernel στην texture μνήμη, αλλά η γνώση αυτή δεν του χρειάζεται.



Σχήμα 2.10: Ενδεδειγμένη προσπέλαση στην texture μνήμη, σε μία διδιάστατα κατανεμημένη περιοχή της μνήμης.

Όταν η προσπέλαση της μνήμης γίνεται με μοτίβα όπως αυτά που ακολουθούν, για την προσπέλαση της μνήμης συνήθως συμφέρει η χρήση της texture μνήμης.

Αριθμητικά, οι 4 θέσεις μνήμης που φαίνονται στο σχήμα 2.10 δεν είναι συνεχόμενες και δεν θα βρίσκονταν στο ίδιο σημείο της cached μνήμης σε ένα τυπικό σύστημα cached μνήμης CPU. Όμως, επειδή η texture μνήμη στις GPU έχει σχεδιαστεί για να επιταχύνει μοτίβα προσπέλασης ακριβώς σαν αυτό, η χρήση της texture μνήμης, αντί για χρήση της global μνήμης, συνήθως βελτιώνει το χρόνο εκτέλεσης της εφαρμογής.

Κατά την επίλυση της εξίσωσης Laplace, χρειάζεται πληροφορία από τους 4 γείτονες κάθε εσωτερικού κόμβου κατά την ανανέωση της λύσης. Σε αυτήν την «τοπικότητα» είναι που κυρίως χρησιμοποιείται η texture μνήμη για επιτάχυνση της εφαρμογής. Οπότε, η προσπέλαση των θέσεων μνήμης N, E, S, W κάθε κόμβου (σχήμα 2.1) συμφέρει γίνεται μέσω της texture μνήμης, αντί της global μνήμης.

Η L2 μνήμη.

Στις GPU αρχιτεκτονικής Fermi έχει προστεθεί μία ακόμα μνήμη, η L2 μνήμη. Η L2 είναι μνήμη «cache» και είναι ενδιάμεση της global μνήμης και του kernel. Σχετικά με την χρήση των L1 και L2 μνημών, ο προγραμματιστής καλείται μόνο να αποφασίσει αν θα χρησιμοποιηθούν και οι 2 cached μνήμες ή αν θα απενεργοποιήσει την L1 μνήμη και θα χρησιμοποιήσει μόνο την L2 μνήμη, σε κάθε kernel. Πέρα από αυτό, ο προγραμματιστής δεν έχει καμία άλλη συμμετοχή στη χρήση της L2, αλλά όλη η διαδικασία είναι αυτοματοποιημένη και εσωτερική της GPU. Έτσι, όταν μία θέση της global μνήμης καλείται επανειλλημένα σε ένα kernel, κατά την εκτέλεσή του, καλείται μία φορά το περιεχόμενο της θέσης από τη χρονοβόρα global μνήμη και αποθηκεύεται στην L2, έτσι ώστε οι επόμενες κλήσεις να γίνουν από την cache μνήμη, L2. Όπως και με την texture μνήμη, όταν χρειάζεται να εγγραφεί καινούργια πληροφορία στην L2 μνήμη αλλά δεν υπάρχει αρκετή ελεύθερη μνήμη, διαγράφεται κάποια προηγούμενη. Και η L2 μνήμη είναι περιορισμένης χωρητικότητας, όπως οι μνήμες texture και constant. Όμως, αντίθετα με τις μνήμες texture και constant, η χρήση της μνήμης L2 γίνεται δίχως επέμβαση του προγραμματιστή.

Οι On-chip μνήμες.

Πέρα από τις παραπάνω ενιαίες μνήμες, οι οποίες μπορούν να προσπελαθούν ελεύθερα από όλα τα threads του kernel, ο κάθε SM έχει πρόσβαση σε κάποιες «on-chip» μνήμες, δηλαδή μνήμες που βρίσκονται, υλικά, πάνω στον SM. Τέτοιες είναι η shared μνήμη, η L1 μνήμη και οι registers. Ακόμα κάθε SM μπορεί να χρησιμοποιήσει και την local μνήμη, η οποία, αν και ο τρόπος χρήσης της θυμίζει τις «on-chip» μνήμες, βρίσκεται δίπλα στη global μνήμη.

Οι Shared και L1 μνήμες.

Η shared μνήμη χρησιμοποιείται όταν χρειάζεται επικοινωνία μεταξύ μεταξύ δύο ή περισότερων threads, που όμως ανήκουν στο ίδιο block. Αυτό συμβαίνει γιατί η shared μνήμη είναι προσπελάσιμη από όλα τα threads ενός block. Ακόμα, η προσπέλασή της γίνεται με πολύ μεγαλύτερη ταχύτητα από ότι της global μνήμης και πρέπει να προτιμάται όπου η εφαρμογή το επιτρέπει.

Για να αξιοποιηθεί σωστά η shared μνήμη πρέπει να αναλυθεί λίγο περισσότερο ο τρόπος λειτουργίας της. Η shared μνήμη χωρίζεται σε ισομεγέθη τμήματα μνήμης, που ονομάζονται αποθετήρια (banks), τα οποία μπορούν να προσπελαστούν ταυτόχρονα. Κάθε εντολή προσπέλασης που αντιστοιχεί σε προσπέλαση μίας θέσης μνήμης ανά αποθετήριο μπορεί να εκτελεστεί ταυτόχρονα, επιταχύνοντας έτσι την εφαρμογή. Όμως, αν δύο ή παραπάνω θέσεις μνήμης, από τις ζητούμενες, βρίσκονται στο ίδιο αποθετήριο, τότε οι δύο προσπελάσεις δεν μπορούν να εκτελεστούν ταυτόχρονα, αλλά εκτελούνται σειριακά με αποτέλεσμα να επιβραδύνουν τους υπολογισμούς (bank conflict).

Στις GPU υπολογιστικής ικανότητας 1.*x* η shared μνήμη αποτελείται από 16 αποθετήρια και είναι έτσι οργανωμένα ώστε συνεχόμενες μεταβλητές μήκους 32-bit να τοποθετούνται από μόνες τους σε συνεχόμενα αποθετήρια. Κάθε αποθετήριο έχει εύρος ζώνης (bandwidth) 32-bit ανά 2 κύκλους ρολογιού. Κάθε εντολή προσπέλασης shared μνήμης, όπου στη συνέχεια, για λόγους απλότητας, θα αναφέρεται απλώς ως «εντολή μνήμης», διασπάται σε 2 ξεχωριστές εντολές μνήμης, μία για κάθε μισό warp, οι οποίες εκτελούνται ανεξάρτητα. Έτσι δεν γίνεται να δημιουργηθεί bank conflict μεταξύ 2 threads εκ των οποίων το πρώτο βρίσκεται στο πρώτο μισό warp και το δεύτερο στο δεύτερο μισό warp, αφού το κάθε ένα από τα δύο threads αντιστοιχεί σε διαφορετική εντολή μνήμης.

Ακόμα μία σημαντική δυνατότητα που προσφέρει η shared μνήμη είναι ο μηχανισμός «προβολής» words μήκους 32-bits. Ένα word μήκους 32-bits μπορεί να διαβαστεί και να «προβληθεί» σε παραπάνω από ένα thread κατά την εκτέλεση μίας μόνο εντολής μνήμης. Δηλαδή, αν, για παράδειγμα, τρία threads που ανήκου στο ίδιο μισό warp χρειάζονται την ίδια πληροφορία από την shared μνήμη, με μία μόνο εντολή μνήμης θα «διαβάσουν» και τα τρία threads την ζητούμενη θέση μνήμης παράλληλα. Πιο συγκεκριμένα, μία εντολή μνήμης, η οποία αποτελείται από πολλές θέσεις μνήμης, εκτελείται σε αρκετά βήματα στην πορεία του χρόνου, εκτελώντας ένα υποσύνολο των συνολικών προσπελάσεων, χωρίς bank conflicts ανά βήμα, μέχρι όλες οι διευθύν-

σεις να έχουν προσπελαθεί. Σε κάθε βήμα, το υποσύνολο των διευθύνσεων που θα προσπελαθούν κατασκευάζεται από τον SM βάσει των διευθύνσεων που δεν έχουν ήδη προσπελαθεί σύμφωνα με την παρακάτω διαδικασία:

- Πρώτα επιλέγεται ένα word, από τα words που πρόκειται να προσπελαθούν, ως το word που θα προβληθεί.

- Στη συνέχεια προστίθενται στις διευθύνσεις που θα προσπελαστούν οι παρακάτω διευθύνσεισ:

  - Όλες οι διευθύνσεις που βρίσκονται μέσα στο υπό προβολή word,
  - Μία διεύθυνση από κάθε αποθετήριο (εκτός από το αποθετήριο στο οποίο ανήκει η υπό προβολή λέξη) στην οποία υπάρχει τουλάχιστον μία διεύθυνση προς προσπέλαση.

Ποιο word θα επιλεγεί για να προβληθεί καθώς και ποια διεύθυνση από κάθε αποθετήριο θα επιλεγεί σε κάθε κύκλο δεν είναι αυστηρά ορισμένο.

Μία συνηθισμένη περίπτωση όπου και δεν υπάρχει bank conflict είναι όταν όλα τα threads μισού warp προσπελαύνουν μία θέση μνήμης μέσα στο ίδιο word μήκους 32-bits.

Στις GPU υπολογιστικής ικανότητας 2.*x* η shared μνήμη αποτελείται από 32 αποθετήρια και, όμοια με πριν, είναι οργανωμένη με τέτοιο τρόπο ώστε συνεχόμενα words μήκους 32-bits να βρίσκονται σε συνεχόμενα αποθετήρια. Και εδώ, το εύρος ζώνης είναι 32 bits ανά 2 κύκλους ρολογιού. Έτσι, σε αυτή την περίπτωση, είναι δυνατό να υπάρχει διαμάχη αποθετηρίων μεταξύ 2 threads, δίχως να έχει σημασία αν είναι στο πρώτο ή στο δεύτερο μισό warp.

Εδώ, bank conflict δημιουργείται μόνο στην περίπτωση που δύο ή περισσότερα threads προσπελαύνουν θέσεις μνήμης που ανήκουν σε διαφορετικά words μήκους 32-bits του ίδιου αποθετηρίου. Αν δύο ή περισσότερα threads προσπελαύνουν θέσεις μνήμης που ανήκουν στο ίδιο word μήκους 32-bits δεν υπάρχει διαμάχη αποθετηρίων. Για εντολές διαβάσματος (read operations) η λέξη προβάλλεται σε όλα τα threads (όπου, αντίθετα με τις κάρτες μικρότερης υπολογιστικής ικανότητας, πολλαπλές λέξεις μπορούν να προβάλλονται ταυτόχρονα). Για εντολές γραφής (write operations) κάθε byte γράφεται από μόνο ένα από τα threads, αν και το ποιο thread θα εκτελέσει την εντολή γραφής δεν είναι αυστηρά καθορισμένο.

Στις GPU αρχιτεκτονικής Fermi, έχει προστεθεί η χρήση άλλης μίας, ακόμα, μνήμης, της μνήμης L1. Η L1 χρησιμοποιείται ως μία επιπλέον cache μνήμη, σαν την L2, όμως βρίσκεται πάνω στον SM, επιτυγχάνοντας ακόμα μεγαλύτερες ταχύτητες προσπέλασης. Το σημαντικότερο στοιχείο της L1 μνήμης είναι ότι δεν είναι αυτόνομη, αλλά μοιράζεται τον ίδιο χώρο με την shared μνήμη, συνολικής έκτασης 64 KBytes. Έτσι, κατά τον προγραμματισμό μίας εφαρμογής, ο προγραμματιστής μπορεί να διαλέξει αν θα χρησιμοποιήσει την 48kB shared μνήμης και 16kB L1 μνήμης ή το αντίστροφο, όπως φαίνεται και στο σχήμα 2.6.

Registers

Στην κορυφή της ιεραρχίας της μνήμης βρίσκοντοι οι registers. Οι registers αποτελούν μία ειδική κατηγορία μνήμης στην οποία αποθηκεύονται όλες οι τοπικές μεταβλητές κάθε thread. Έχουν αμελητέο κόστος προσπέλασης (access latency) και είναι η πιο γρήγορη μορφή μνήμης που διαθέτει η GPU. Στους registers φορτώνονται αυτόματα όλες οι τοπικές μεταβλητές του τρέχοντος kernel και χρησιμοποιούνται και για την αποθήκευση ενδιάμεσων τιμών σε περιπτώσεις σύνθετων πράξεων.

Όταν, όμως, ένα thread έχει μεγαλύτερες απαιτήσεις σε registers από όσους μπορεί να διαθέσει η GPU, συμβαίνει υπερχείλιση των registers (register spilling) και η GPU μεταφέρει αυτόματα τις παραπάνω μεταβλητες στη local μνήμη. Το σενάριο αυτό δεν είναι επιθυμητο, καθώς ο χρόνος που χρειάζεται για την προσπέλαση της local μνήμης είναι ο ίδιος με τον χρόνο προσπέλασης της global μνήμης και για αυτό το λόγο πρέπει να αποφεύγονται kernels υπερβολικών απαιτήσεων για registers. Ακόμα, υπάρχει περίπτωση ο μεταγλωττιστής να ορίσει κάποιες τοπικές μεταβλητές στην local μνήμη σε περίπτωση μεγάλων πινάκων ή δομών που θεωρεί ότι θα καταλάμβαναν μεγάλο χώρο από τους registers. Στις GPU αρχιτεκτονικής Fermi, για βελτίωση της απόδοσης της local μνήμης, χρησιμοποιούνται οι cache μνήμες L1 και L2 με τον ίδιο ακριβώς τρόπο που χρησιμοποιούνται για την προσπέλαση της global μνήμης.

2.3.6 Άλλες προγραμματιστικές δυνατότητες

CUDA streams

Μία ακόμα δυνατότητα που έχει ένας προγραμματιστής όταν προγραμματίζει σε CUDA είναι προσθέσει στον κώδικά του CUDA streams (ροές CUDA), τα οποία του δίνουν τη δυνατότητα να προγραμματίζει μία σειρά εργασιών που πρέπει να εκτελεστούν σειριακά.

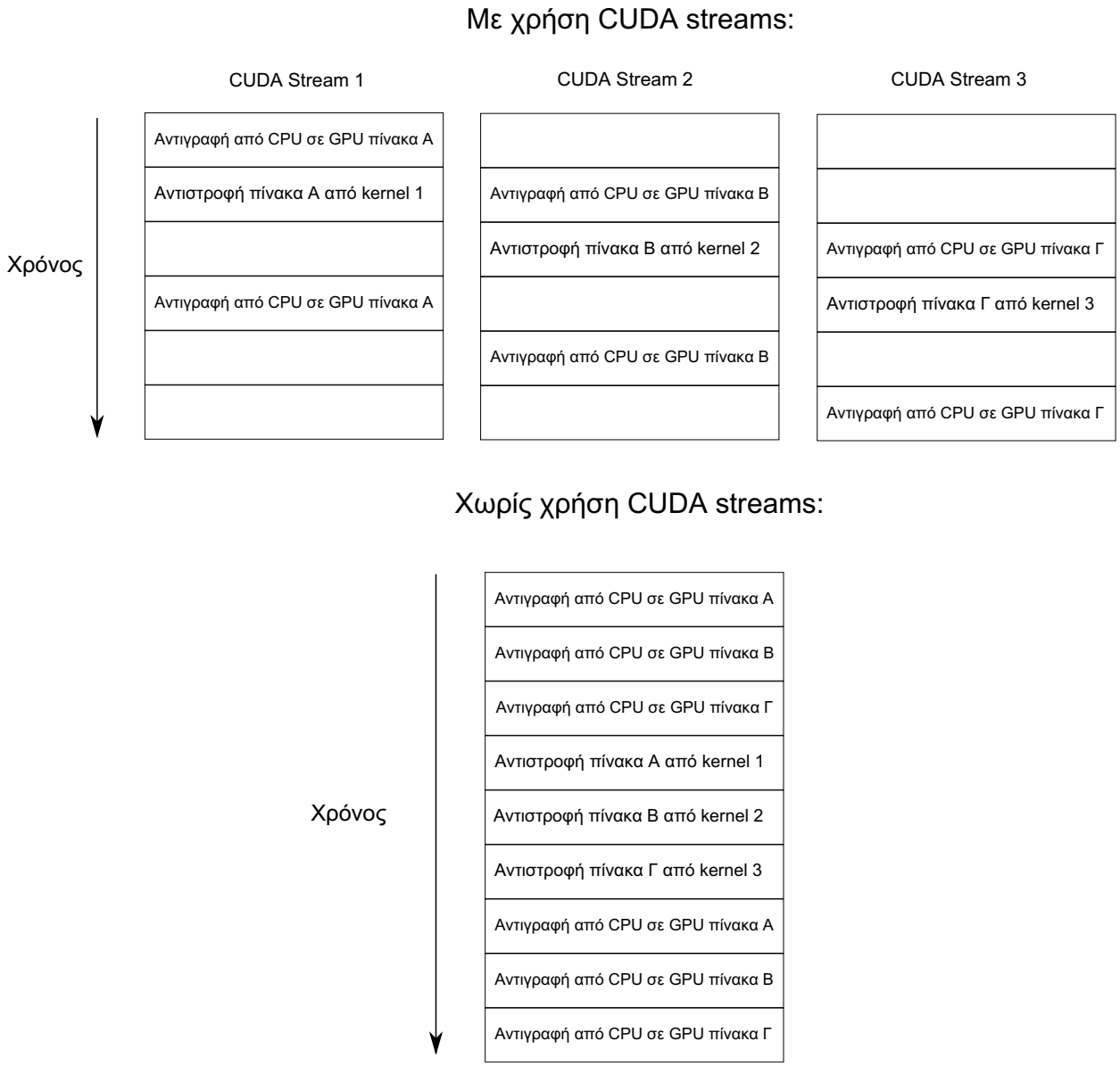
Ένα παράδειγμα τέτοιων εργασιών θα μπορούσε να είναι η αντιγραφή ενός πίνακα από τη CPU στη GPU, η αντιστροφή του πίνακα από τη GPU και, στη συνέχεια, η αντιγραφή πίσω στη CPU. Όταν περισσότερα του ενός CUDA streams εκτελούνται παράλληλα, οι εργασίες καθενός stream εκτελούνται παράλληλα και ασύγχρονα με τις εργασίες των υπολοίπων streams, με τρόπο τέτοιο ώστε και η CPU και η GPU να λειτουργούν παράλληλα. Θεωρώντας πως το παραπάνω παράδειγμα έπρεπε να εφαρμοστεί σε τρεις πίνακες αντι για έναν, δίχως χρήση των CUDA streams θα έπρεπε πρώτα να αντιγραφούν όλοι οι πίνακες από την CPU στην GPU, στη συνέχεια να γίνει ολόκληρη η επεξεργασία από την GPU και μόνο αφού είχε τελειώσει η GPU την επεξεργασία θα μπορούσε να ξεκινήσει η αντιγραφή από την GPU πίσω στην CPU. Με χρήση των CUDA streams, η παραπάνω διαδικασία παραλληλοποιείται. Μόλις τελειώσει η αντιγραφή του πρώτου πίνακα από τη CPU στη GPU, ξεκινά και το πρώτο kernel να αντιστρέφει τον πρώτο πίνακα. Παράλληλα, η CPU αντιγράφει το δεύτερο πίνακα και, στη συνέχεια, τον τρίτο, ενώ η GPU εκτελεί ακόμα το πρώτο kernel. Μόλις η GPU τελείωσει την αντιστροφή του πρώτου πίνακα, θεωρώντας πως έχει τελειώσει η αντιγραφή του δεύτερου πίνακα, ξεκινά άμεσα την αντιστρο-

φή αυτού. Προφανώς, μετά το δεύτερο πίνακα, με το ίδιο κριτήριο, ξεκινά και η αντιστροφή του τρίτου πίνακα. Η CPU, από την άλλη, μόλις ολοκληρωθούν και οι τρείς αντιγραφές από τη CPU στη GPU καθώς και η αντιστροφή του πρώτου kernel, ξεκινά την αντιγραφή από τη GPU στη CPU, δίχως απαραίτητα να έχουν τελειώσει και οι αντιστροφές των άλλων δύο πινάκων. Η διαδικασία αυτή συνεχίζεται μέχρι εκτελεστούν όλες οι εργασίες που είναι προγραμματισμένες σε κάθε CUDA stream. Αν η κάθε αντιγραφή πίνακα χρειαζόταν περίπου τον ίδιο χρόνο με την αντιστροφή του, το παραπάνω παράδειγμα σκιαγραφείται στα σχήματα 2.11, 2.12 με ή χωρίς τη χρήση CUDA streams.

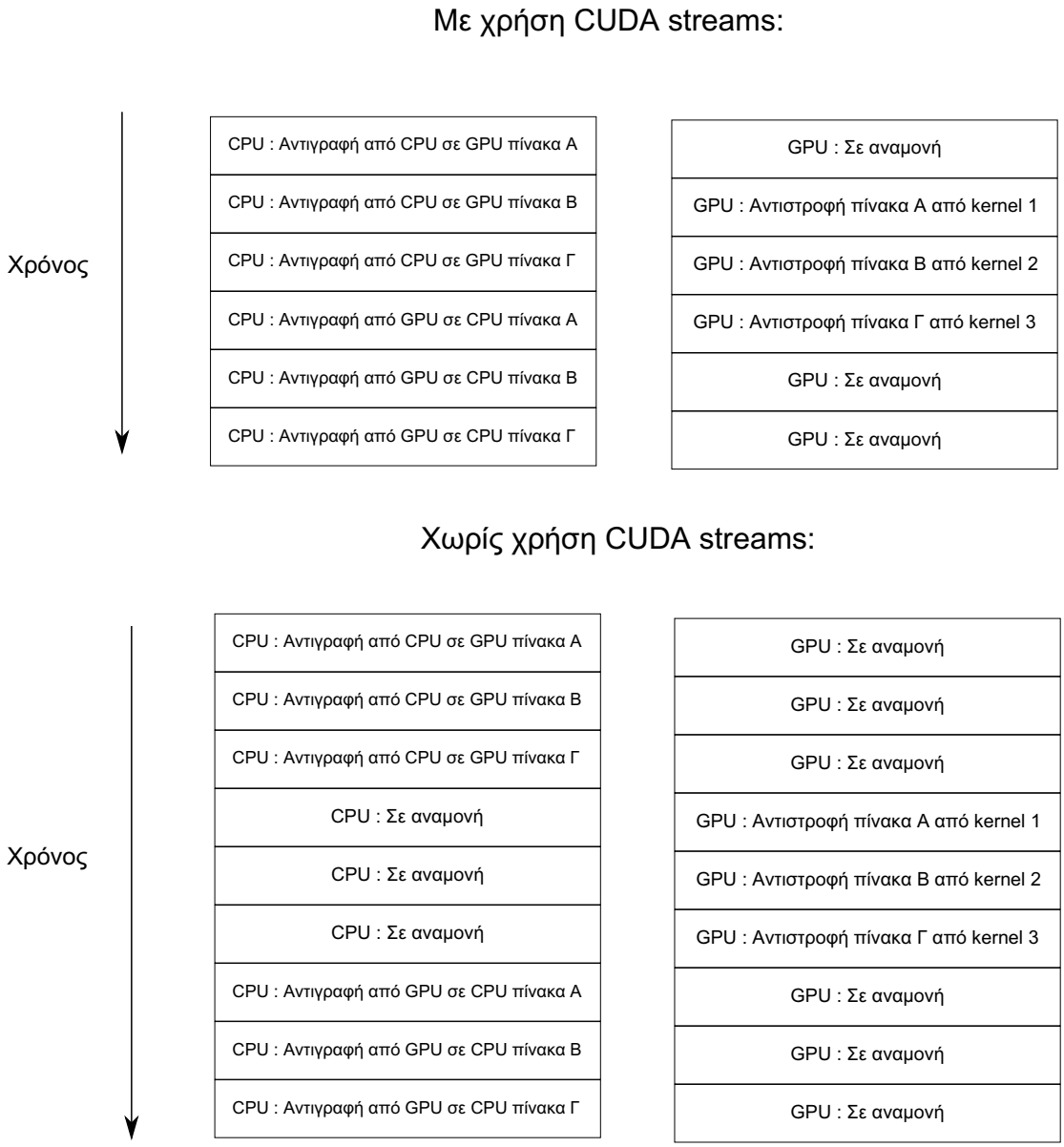
Στον κώδικα που αναπτύχθηκε χρησιμοποιούνται CUDA streams για την άθροιση των υπολοίπων των εξισώσεων της ροής, όπως αναλύεται στην παράγραφο 5.

Παράλληλη εκτέλεση kernels

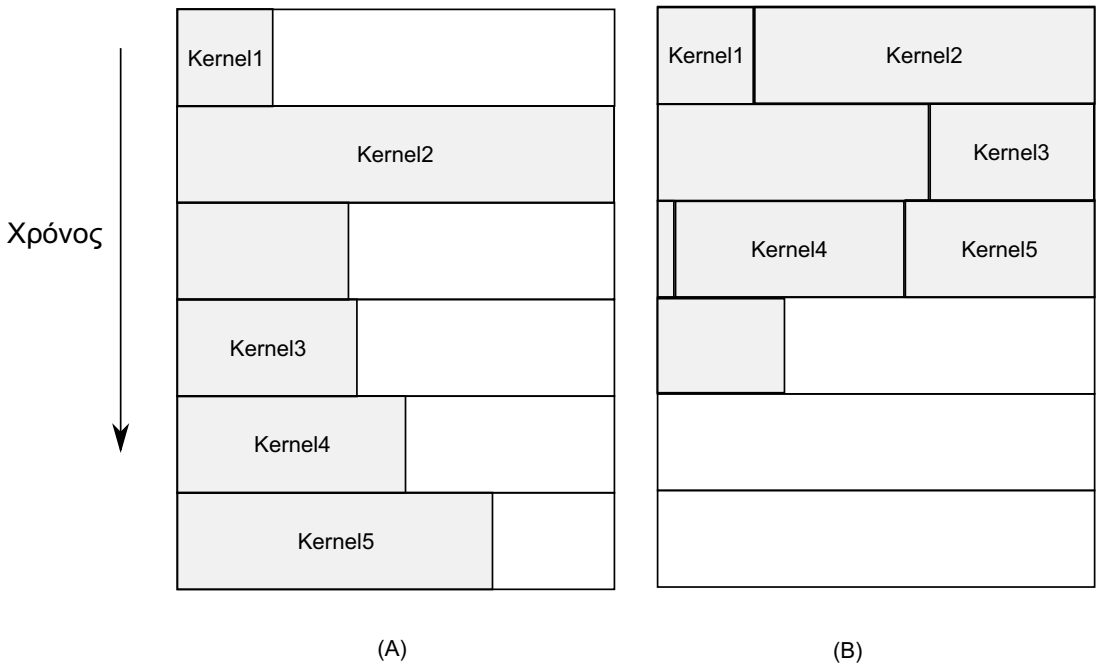
Μερικές από τις GPU αρχιτεκτονικής Fermi προσφέρουν τη δυνατότητα στο χρήστη να εκτελεί παράλληλα περισσότερα του ενός kernel, στην ίδια GPU. Αυτή η δυνατότητα του επιτρέπει να χρησιμοποιεί τους πόρους ολόκληρης της GPU ακόμα και όταν τα επιμέρους kernels έχουν μικρές απαιτήσεις και δεν επαρκούν το καθένα από μόνο του για την πλήρη αξιοποίηση της GPU. Ποιοτική απεικόνιση της βελτίωσης αυτής φαίνεται στο σχήμα 2.13.



Σχήμα 2.11: Σειρά εκτέλεσης διεργασιών που πρέπει να πραγματοποιηθούν για την αντιστροφή 3 πινάκων Α, Β και Γ, συναρτήσει του χρόνου. Με χρήση CUDA streams προκύπτει το πάνω χρονοδιάγραμμα, ενώ χωρίς χρήση CUDA streams προκύπτει το κάτω χρονοδιάγραμμα.



Σχήμα 2.12: Διεργασίες των GPU και CPU, του ίδιου προβλήματος με το σχήμα 2.11, συναρτήσει του χρόνου. Όπως και στο προηγούμενο σχήμα, παρουσιάζεται στο πάνω μέρος η περίπτωση χρήσης CUDA streams και στο κάτω η περίπτωση χωρίς χρήση CUDA streams.



Σχήμα 2.13: Στο σχήμα (*A*) φαίνεται ο απαιτούμενος χρόνος εκτέλεσης για σειριακή εκτέλεση των kernels ενώ στο σχήμα (*B*) ο αντίστοιχος χρόνος εκτέλεσης όταν τα kernels εκτελούνται παράλληλα. Το οριζόντιο τμήμα κάθε χρονικού βήματος συμβολίζει την πλήρη υπολογιστική ισχύ της GPU σε κάθε χρονική στιγμή. Προφανώς, για να επωφεληθεί ένας κώδικας από παράλληλη εκτέλεση των επιμέρους kernels, τα kernels που τρέχουν παράλληλα δεν πρέπει να αλληλοεξαρτόνται.

Κεφάλαιο 3

Παρουσίαση των τριδιάστατων εξισώσεων Euler και διακριτοποίησης αυτών

Στο κεφάλαιο αυτό παρουσιάζονται οι εξισώσεις ροής, σε συντηρητική και μη- συντηρητική γραφή, για χρονικά μη-μόνιμη, τριδιάστατη ροή ατριβούς συμπιεστού ρευστού, δηλαδη οι εξισώσεις Euler. Ακόμη παρουσιάζεται ο τρόπος αριθμητικής επίλυσης των εξισώσεων αυτών μέσω μεθόδου χρονοπροέλασης (time marching) πεπερασμένων όγκων με σκοπό να εφαρμοστεί σε δομημένα, τριδιάστατα πλέγματα [22], [23].

3.1 Οι χρονικά μόνιμες εξισώσεις Euler

3.1.1 Διαφορική γραφή των χρονικά μόνιμων εξισώσεων Euler

Οι εξισώσεις που παρουσιάζονται είναι οι εξισώσεις Euler, σε αδιάστατη συντηρητική γραφή, για χρονικά μη-μεταβαλλόμενη ροή συμπιεστού ρευστού, απουσία βαρυτικών δυνάμεων, στο καρτεσιανό σύστημα αναφοράς, για τριδιάστατες ροές.

$$\frac{\partial \varrho}{\partial t} + \frac{\partial(\varrho u_r)}{\partial x_r} = 0$$

$$\frac{\partial(\varrho u_i)}{\partial t} + \frac{\partial}{\partial x_r}(\varrho u_r u_i + p\delta_{ri}) = 0 \quad (i = 1, 2, 3) \tag{3.1}$$

$$\frac{\partial(\varrho E)}{\partial t} + \frac{\partial}{\partial x_r}[u_r(\varrho E + p)] = 0$$

όπου ϱ είναι η πυκνότητα του ρευστού, u_r $(r = 1, 2, 3)$ η συνιστώσα της ταχύτητας κατά την κατεύθυνση x_r, E η ολική ενέργεια ανά μονάδα μάζας, e η εσωτερική ενέργεια ανά μονάνδα μάζας, και p η πίεση του ρευστού. Ο χρονικός όρος που υπάρχει στις εξισώσεις αποτελεί έναν ψευδοχρονικό όρο και έχει προστεθεί για την

εκμετάλλευση των ιδιοτήτων των υπερβολικών συστημάτων και για την εφαρμογή μίας μεθόδου χρονοποέλασης. Όπου υπάρχουν διπλά επαναλαμβανόμενοι δείκτες νοείται άθροιση σύμφωνα με τη σύμβαση του Einstain, εκτός αν δηλώνεται ρητά το αντίθετο. Η ολική ενέργεια ανά μονάδα μάζας εκφράζεται από τη σχέση:

$$E = e + \frac{1}{2} u_r u_r \tag{3.2}$$

Ορίζοντας το διάνυσμα των συντηρητικών μεταβλητών

$$\overrightarrow{U} = \begin{bmatrix} \varrho \\ \varrho u_1 \\ \varrho u_2 \\ \varrho u_3 \\ E \end{bmatrix} = \begin{bmatrix} \varrho \\ \varrho u \\ \varrho v \\ \varrho w \\ E \end{bmatrix} \tag{3.3}$$

καθώς και το διανύσμα της μη-συνεκτικής ροής κατά την κατεύθυνση x_r

$$\overrightarrow{F_r} = \begin{bmatrix} \varrho u_r \\ \varrho u_1 u_r + p\delta_{1r} \\ \varrho u_2 u_r + p\delta_{2r} \\ \varrho u_3 u_r + p\delta_{3r} \\ u_r(\varrho E + p) \end{bmatrix} \tag{3.4}$$

οι εξισώσεις 3.1 μπορούν να γραφούν στην παρακάτω διανυσματική γραφη

$$\frac{\partial \overrightarrow{U}}{\partial t} + \frac{\partial \overrightarrow{F_r}}{\partial x_r} = 0 \tag{3.5}$$

Ακόμα, γίνεται η παραδοχή του τέλειου αερίου. Ως τέλειο αέριο ορίζεται το αέριο του οποίου η συμπεριφορά ακολουθεί την καταστατική εξίσωση:

$$p = \varrho R_g T \tag{3.6}$$

όπου R_g είναι η σταθερά του τέλειου αερίου και υπολογίζεται σύμφωνα με τις ειδικές θερμοχωρητικότητες υπό σταθερή πίεση και όγκο, c_p και c_v αντίστοιχα, από την σχέση:

$$R_g = c_p - c_v \tag{3.7}$$

Ως θερμοχωρητικότητα ορίζεται η θερμότητα που απορροφά ή εκλύει ένα σώμα κατά τη μεταβολή της θερμοκρασίας του κατά έναν βαθμό Κελσίου (^oC). Η ειδική θερμοχωρητικότητα εκφράζει την αντίστοιχη ποσότητα θερμοκρασίας ανά μονάδα μάζας, ενώ οι δείκτες «υπό σταθερή πίεση» ή «υπό σταθερό όγκο» εκφράζουν μεταβολές με σταθερή πίεση ή όγκο, αντιστοιχα. Έτσι, οι μαθηματικές εκφράσεις της ειδικής θερμοχωρητικότητας ορίζονται:

$$c_p = \left(\frac{\partial q}{\partial T}\right)_p \ , \ \ c_v = \left(\frac{\partial q}{\partial T}\right)_v$$

ή ισοδύναμα με χρήση του πρώτου θερμοδυναμικού αξιώματος:

$$c_p = (\frac{\partial h}{\partial T})_p \ , \ c_v = (\frac{\partial e}{\partial T})_v \tag{3.8}$$

όπου με h συμβολίζεται η ενθαλπία του ρευστού ανά μονάδα μάζας. Η σχέση που συνδέει τη στατική ενθαλπία με την εσωτερική ενέργεια είναι η:

$$h = e + \frac{p}{\varrho} \tag{3.9}$$

Γενικά, οι ειδικές θερμοχωτητικότητες μπορούν να υπολογιστούν γνωρίζοντας δύο οποιαδήποτε θερμοδυναμικά μεγέθη. Στα τέλεια αέρια χρειάζεται μόνο η γνώση της θερμοκρασίας. Όμως η μεταβολή συναρτήσει της θερμοκρασίας είναι μικρή για τα τέλεια αέρια και για αυτό συνήθως παραλείπεται και οι ειδικές θερμοχωρητικότητες θεωρούνται ως σταθερές. Σύμφωνα με αυτά, ολοκληρώνοντας τις εξισώσεις 3.8 προκύπτει:

$$e = c_v T \ , \ h = c_p T \tag{3.10}$$

Τέλος, οριζεται ο εκθέτης ισεντροπικής μεταβολής γ σύμφωνα με τη σχέση:

$$\gamma = \frac{c_p}{c_v} \tag{3.11}$$

Ενδεικτικά, θεωρώντας τον αέρα ως τέλειο αέριο οι παραπάνω σταθερές που τον χαρακτηρίζουν είναι:

$$\begin{array}{lcll} R_g & = & 287.04 & m^2 sec^{-2} K^{-1} \\ c_p & = & 1004.64 & m^2 sec^{-2} K^{-1} \\ c_v & = & 717.6 & m^2 sec^{-2} K^{-1} \\ \gamma & = & 1.4 & \end{array}$$

Με χρήση των προηγούμενων, η εξίσωση 3.2 μπορεί να γραφεί και ως:

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\varrho u_r u_r \tag{3.12}$$

Επίσης, η ολική ενθαλπία δίνεται από τη σχέση:

$$h_t = \frac{E + p}{\varrho} = \frac{\gamma p}{\varrho(\gamma - 1)} + \frac{1}{2}u_r u_r \tag{3.13}$$

3.1.2 Αδιαστατοποίηση των εξισώσεων

Πριν από την διακριτοποίηση και την επίλυση των εξισώσεων προηγείται η αδιαστατοποίηση αυτών. Για την αδιαστατοποίηση των εξισώσεων επιλέγονται κάποιες καταλληλες παράμετροι αδιαστατοποίησης, τα μεγέθη αναφοράς (reference), σύμφωνα με τις οποίες προκύπτουν τα ακόλουθα αδιάστατα μεγέθη:

$$\breve{x}_i = \frac{x_i}{L_{ref}} \ , \ \breve{u}_i = \frac{u_i}{U_{ref}} \ , \ \breve{\varrho} = \frac{\varrho}{\varrho_{ref}} \tag{3.14}$$

όπου L_{ref}, U_{ref} και ϱ_{ref} συμβολίζονται το μήκος αδιαστατοποίησης, η ταχύτητα αδιαστατοποίησης και η πυκνότητα αδιαστατοποίησης, αντίστοιχα. Με το σύμβολο $\breve{}$ συμβολίζονται οι αντίστοιχες αδιάστατες ποσότητες.

Ακόμα επιλέγεται:

$$\breve{R}_g = \gamma - 1 \tag{3.15}$$

Δηλαδή:

$$\left(R_g\right)_{ref} = c_v \tag{3.16}$$

Όμως, είναι επιθυμητό η τελική μορφή των εξισώσεων να διατηρήσει την αρχική μορφή της, έχοντας αντί για τις διαστατές τιμές, τις αντίστοιχες αδιάστατες. Με αυτό, ως γνώμονα, αντικαθιστώντας τα διαστατά μεγέθη στις γνωστές εξισώσεις με τα αδιάστατα και συγκρίνοντας την καινούργια μορφη κάθε εξίσωσης με την επιθυμητή προκύπτουν τα μεγέθη αναφοράς και των υπόλοιπων παραμέτρων.

Αδιαστατοποίηση πίεσης

$p_t = p + \frac{1}{2}\varrho\left|\vec{u}\right|^2 \Rightarrow \left|\vec{u}\right| = \sqrt{\frac{2}{\varrho}(p_t - p)} \Rightarrow \left|\breve{\vec{u}}\right| = \frac{1}{U_{ref}}\sqrt{\frac{2}{\breve{\varrho}}\frac{p_{ref}}{\varrho_{ref}}(\breve{p}_t - \breve{p})}$
όμως η μορφή των σχέσεων είναι επιθυμητό να παραμένει ίδια μετά την εισαγωγή των αδιάστατων μεγεθών, δηλαδή:

$$\left|\breve{\vec{u}}\right| = \sqrt{\frac{2}{\breve{\varrho}}(\breve{p}_t - \breve{p})}$$

Επιλέγουμε δηλαδή:

$$U_{ref} = \sqrt{\frac{p_{ref}}{\varrho_{ref}}} \Rightarrow p_{ref} = \varrho_{ref} U_{ref}^2$$

Αδιαστατοποίηση θερμοκρασίας

$$p = \varrho R_g T \Rightarrow \breve{p} = \left(\frac{\varrho_{ref}\left(R_g\right)_{ref} T_{ref}}{p_{ref}}\right)\breve{\varrho}\breve{R}_g\breve{T} \longrightarrow \frac{\varrho_{ref}\left(R_g\right)_{ref} T_{ref}}{p_{ref}} = 1 \Rightarrow T_{ref} = \frac{U_{ref}^2}{c_v}$$

Αδιαστατοποίηση θερμοχωρητικοτήτων

$$T_t = T + \frac{1}{2c_p}\left|\vec{u}\right|^2 \Rightarrow \left|\vec{u}\right| = \sqrt{2c_p\left(T_t - T\right)} \Rightarrow \left|\breve{\vec{u}}\right| = \sqrt{\left(\frac{(c_p)_{ref}T_{ref}}{U_{ref}^2}\right)2\breve{c}_p\left(\breve{T}_t - \breve{T}\right)} \longrightarrow$$
$$\left(\frac{(c_p)_{ref}T_{ref}}{U_{ref}}\right) = 1 \Rightarrow \left(c_p\right)_{ref} = \frac{U_{ref}^2}{T_{ref}} \Rightarrow \left(c_p\right)_{ref} = c_v$$

$$\gamma = \frac{c_p}{c_v} \Rightarrow \gamma = \left(\frac{(c_p)_{ref}}{(c_v)_{ref}}\right)\frac{\breve{c}_p}{\breve{c}_v} \longrightarrow (c_v)_{ref} = c_v$$

Αδιαστατοποίηση εσωτερικής ενέργειας

$$e = c_vT \Rightarrow \breve{e} = \left(\frac{(c_v)_{ref}T_{ref}}{e_{ref}}\right)\breve{c}_v\breve{T} \longrightarrow e_{ref} = U_{ref}^2$$

Αδιαστατοποίηση ενθαλπίας

$$h = c_pT \Rightarrow \breve{h} = \left(\frac{(c_p)_{ref}T_{ref}}{h_{ref}}\right)\breve{c}_p\breve{T} \longrightarrow h_{ref} = U_{ref}^2$$

Αδιαστατοποίηση ολικής ενέργειας

$$E = \frac{p}{\gamma-1} + \frac{1}{2}\varrho\left|\vec{u}\right|^2 \Rightarrow \breve{E} = \frac{1}{E_{ref}}\left[\left(p_{ref}\right)\frac{\breve{p}}{\gamma-1} + \frac{1}{2}\left(\varrho_{ref}U_{ref}^2\right)\breve{\varrho}\left|\breve{\vec{u}}\right|^2\right] \Rightarrow$$
$$\breve{E} = \frac{\varrho_{ref}U_{ref}^2}{E_{ref}}\left(\frac{\breve{p}}{\gamma-1} + \frac{1}{2}\breve{\varrho}\left|\breve{\vec{u}}\right|^2\right) \longrightarrow$$
$$E_{ref} = \varrho_{ref}U_{ref}^2$$

Τελικά η αδιαστατοποίηση των υπολοίπων θερμοδυναμικών μεγεθών (δηλαδή πλην αυτών που φαίνονται στις σχέσεις 3.14) γίνεται όπως παρακάτω:

$$\breve{p} = \frac{p}{\varrho_{ref}U_{ref}^2} \;,\; \breve{T} = \frac{T}{U_{ref}^2/c_v} \;,\; \breve{e} = \frac{e}{U_{ref}^2} \;,\; \breve{h} = \frac{h}{U_{ref}^2} \;,\; \breve{E} = \frac{E}{\varrho_{ref}U_{ref}^2} \qquad (3.17)$$

Όπως επίσης :

$$\breve{c}_p = \gamma \;,\; \breve{c}_v = 1 \qquad (3.18)$$

Διατύπωση αδιάστατων εξισώσεων

- **Εξίσωση συνέχειας**

Η εξίσωση της συνέχειας αποτελεί την πρώτη γραμμή των εξισώσεων 3.1. Για πρακτικούς λόγους, οι εξισώσεις συνέχειας, ορμής και ενέργειας θα γραφούν στην μορφή της εξίσωσης 3.5, με χρήση των διανυσμάτων $\overrightarrow{U}$ και $\overrightarrow{F_r}$, όπως αυτά ορίστικαν στις σχέσεις 3.3 και 3.4.
Εξίσωση συνέχειας

$$\frac{\partial U_1}{\partial t} + \frac{\partial (F_r)_1}{\partial x_r} = 0 \Rightarrow$$

$$\frac{(U_1)_{ref}}{t_{ref}} \frac{\partial \breve{U}_1}{\partial \breve{t}} + \frac{((F_r)_1)_{ref}}{L_{ref}} \frac{\partial (\breve{F_r})_1}{\partial \breve{x}_r} = 0$$

Όμως

$U_1 \quad = \varrho \longrightarrow (U_1)_{ref} = \varrho_{ref}$
$(F_1)_1 = \varrho u \longrightarrow (F_1)_{ref} = \varrho_{ref} U_{ref}$
$(F_2)_1 = \varrho v \longrightarrow (F_2)_{ref} = \varrho_{ref} U_{ref}$
$(F_3)_1 = \varrho w \longrightarrow (F_3)_{ref} = \varrho_{ref} U_{ref}$

Επομένως

$$\left(\frac{L_{ref}}{U_{ref} t_{ref}} \right) \frac{\partial \breve{U}_1}{\partial \breve{t}} + \frac{\partial (\breve{F_r})_1}{\partial \breve{x}_r} = 0$$

Ορίζοντας κατάλληλα το χρονικό μέγεθος αναφοράς t_{ref}, η ποσότητα μπροστα από τον χρονικό όρο μπορεί να απαλειφθεί, ενώ παράλληλα έχει επιτευχθεί και ο αρχικός στόχος, η διαστατή και η αδιάστατη εξίσωση να είναι ίδιες. Επιλέγεται, συνεπώς :

$$t_{ref} = \frac{L_{ref}}{U_{ref}} \quad , \quad \breve{t} = \frac{t}{L_{ref}/U_{ref}} \tag{3.19}$$

- **Διατήρηση ορμής**

Οι εξισώσεις διατήρησης της ορμής είναι η δεύτερη γραμμή των εξισώσεων 3.1, και αποτελείται από τρείς εξισώσεις, μία για κάθε κατεύθυνση x, y και z. Θα ακολουθήσει η εξαγωγή μόνο της αδιάστατης κατά x διατήρησης της ορμής, με εκείνες για την y και την z κατευθύνσεις να ακολουθούν τον ίδιο δρόμο και να διαφοροποιούνται μόνο στους δείκτες του διανύσματος των συντηρητικών μεταβλητών και των διανυσμάτων ροής (3 και 4 αντί για 2).

Εξίσωση ορμής κατά x

$$\frac{\partial U_2}{\partial t} + \frac{\partial (F_r)_2}{\partial x_r} = 0 \Rightarrow$$

$$\frac{(U_2)_{ref}}{t_{ref}} \frac{\partial \breve{U}_2}{\partial \breve{t}} + \frac{((F_r)_2)_{ref}}{L_{ref}} \frac{\partial (\breve{F}_r)_2}{\partial \breve{x}_r} = 0$$

Όμως

$$\begin{aligned}
U_2 &= \varrho u \longrightarrow (U_2)_{ref} = \varrho_{ref} U_{ref} \\
(F_1)_2 &= \varrho u^2 + p \longrightarrow ((F_1)_2)_{ref} = \varrho_{ref} U_{ref}^2 \\
(F_2)_2 &= \varrho uv \longrightarrow ((F_2)_2)_{ref} = \varrho_{ref} U_{ref}^2 \\
(F_3)_2 &= \varrho uw \longrightarrow ((F_3)_2)_{ref} = \varrho_{ref} U_{ref}^2
\end{aligned}$$

Οπότε η αδιάστατη εξίσωση ορμής κατά x, με απλή αντικατάσταση, παίρνει τη μορφή:

$$\frac{\partial \breve{U}_2}{\partial \breve{t}} + \frac{\partial (\breve{F}_r)_2}{\partial \breve{x}_r} = 0$$

- **Διατήρηση ενέργειας**

Εξίσωση ενέργειας

$$\frac{\partial U_5}{\partial t} + \frac{\partial (F_r)_5}{\partial x_r} = 0 \Rightarrow$$

$$\frac{(U_5)_{ref}}{t_{ref}} \frac{\partial \breve{U}_5}{\partial \breve{t}} + \frac{((F_r)_5)_{ref}}{L_{ref}} \frac{\partial (\breve{F}_r)_5}{\partial \breve{x}_r} = 0$$

Όμως

$$\begin{aligned}
U_5 &= E \longrightarrow (U_5)_{ref} = \varrho_{ref} U_{ref}^2 \\
(F_1)_5 &= (E + p)u \longrightarrow ((F_1)_5)_{ref} = \varrho_{ref} U_{ref}^3 \\
(F_2)_5 &= (E + p)v \longrightarrow ((F_2)_5)_{ref} = \varrho_{ref} U_{ref}^3 \\
(F_3)_5 &= (E + p)w \longrightarrow ((F_3)_5)_{ref} = \varrho_{ref} U_{ref}^3
\end{aligned}$$

Και εδώ, με απλή αντικατάσταση προκύπτει:

$$\frac{\partial \breve{U}_5}{\partial \breve{t}} + \frac{\partial (\breve{F}_r)_5}{\partial \breve{x}_r} = 0$$

Τελικά, το κέρδος της αδιαστατοποίησης, είναι ότι πλέον, η κάθε επίλυση ενός συστήματος εξισώσεων Euler δεν εκφράζει μία μεμονωμένη περίπτωση, αλλά μία

οικογένεια όμοιων εφαρμογών, ενώ το τελικό σύστημα δεν διαφέρει σε τίποτα από το αρχικό των εξισώσεων 3.1. Όμως, ολόκληρη η διαδικασία της αδιαστατοποίησης βασίστηκε στα μεγέθη αναφοράς L_{ref}, U_{ref} και ϱ_{ref}. Η επιλογή των μεγεθών αυτών είναι απόφαση του χρήστη και είναι άμεσα συνδεδεμένη με το είδος του προβλήματος προς επίλυση.

Σε προβλήματα εσωτερικής αεροδυναμικής, δεδομένου ότι συνήθως δίνονται η ολική πίεση p_t^{in} και ολική θερμοκρασία T_t^{in} στην είσοδο, τα μεγέθη αναφοράς επιλέγονται με τέτοιο τρόπο ώστε $\breve{p}_t^{in} = 1$ και $\breve{T}_t^{in} = 1$. Αντίθετα, σε προβλήματα εξωτερικής αεροδυναμικής, όπου συνήθως δίνονται η στατική πυκνότητα ϱ^∞ και η ταχύτητα $\left|\vec{u}\right|^\infty$ στο επ' άπειρο όριο, τα μεγέθη αναφοράς επιλέγονται με τέτοιο τρόπο ώστε να ισχύει $\breve{\varrho}^\infty = 1$ και $\left|\vec{\breve{u}}\right|^\infty$.

Στη συνέχεια, οποιαδήποτε αναφορά στις εξισώσεις αναφέρεται στις αδιάστατες εξισώσεις, αλλά για πρακτικούς λόγους παραλείπεται το αντίστοιχο σύμβολο σε κάθε μέγεθος.

3.2 Διακριτοποίηση του χωρίου ροής

Όπως είναι γνωστό, πριν την αριθμητική επίλυση οποιουδήποτε προβλήματος ροής, απαιτείται η διακριτοποίηση του χωρίου ροής, δηλαδή η κατασκευή του πλέγματος στους κόμβους του οποίου θα επιλυθούν οι εξισώσεις ροής. Τα πλέγματα χωριζονται σε δύο βασικές κατηγορίες, τα δομημένα και τα μη-δομημένα. Ο επιλύτης που αναπτύχθηκε στα πλαίσια αυτής της εργασίας έχει ως σκοπό τη διερεύνηση των επιδόσεων των GPU σε προβλήματα ροής διακριτοποιημένα με χρήση δομημένων πλεγμάτων. Έτσι, όλες οι παρακάτω αναφορές απευθύνονται σε δομημένα πλέγματα. Τα δομημένα πλέγματα αποτελούνται από αυστηρά ταξινομημένους κόμβους και αποκλειστικά απο τετραπλευρικά στοιχεία, για διδιάστατα πλέγματα, ή εξαεδρικά στοιχεία, για τριδιάστατα πλέγματα. Τα πλεονεκτήματα των δομημένων πλεγμάτων απορρέουν από την αυστηρή ταξινόμηση των κόμβων τους. Βασικό πλεονέκτημα της δομής είναι η εκ των προτέρων γνωστή σχετική θέση κάθε κόμβου, οπότε είναι γνωστοί και οι γείτονες του εκάστοτε κόμβου, δίχως την ανάγκη για αποθήκευση επιπλέον πληροφοριών, όπως συμβαίνει στα μη-δομημένα πλέγματα. Αυτή η βασική διαφορά επιτρέπει στους επιλύτες δομημένων πλεγμάτων να μπορούν να λειτουργήσουν με μικρότερες απαιτήσεις μνήμης, συγκριτικά με αντίστοιχους επιλύτες μη-δομημένων πλεγμάτων. Επίσης, στα δομημένα πλέγματα υπάρχει η δυνατότητα ανάπτυξης επιλυτών, με σχήματα πεπερασμένων διαφορών, μεγαλύτερης ακρίβειας και τα μητρώα των συντελεστών που προκύπτουν είναι διαγώνιας μορφής. Ένα ακόμα σημαντικό πλεονέκτημα είναι ότι λόγω της δομής των κόμβων του πλέγματος, αντίστοιχα είναι ταξινομημένες όλες οι πληροφορίες μέσα στην μνήμη του υπολογιστή. Καλύτερη προσπέλαση μνήμης κατά την εκτέλεση του προγράμματος μεταφράζεται σε καλύτερους χρόνους επίλυσης, ιδιαίτερα στις GPU όπου ο τρόπος προσπέλασης της μνήμης είναι καθοριστικής σημασίας.

3.3 Διακριτοποίηση των εξισώσεων ροής

3.3.1 Ορισμός όγκων ελέγχου

Για την διακριτοποίηση των εξισώσεων 3.5 χρησιμοποιείται ένα σχήμα πεπερασμένων όγκων, με κεντροκομβική προσέγγιση. Για την εφαρμογή αυτού χρειάζεται ο ορισμός των όγκων ελέγχου γύρω από τους κόμβους του πλέγματος, στους οποίους και αποθηκεύονται όλες οι μεταβλητές της ροής. Μία σωστή επίλυση του προβλήματος ροής επιβάλλει οι όγκοι ελέγχου να καλύπτουν ολόκληρο το χωρίο ροής, αλλά και να μην αλληλοκαλύπτονται.

Η μέθοδος επίλυσης που αναπτύσσεται στην παρούσα εργασία αναφέρεται σε δομημένα, τριδιάστατα πλέγματα, οπότε όλα τα πλεγματικά στοιχεία που περιβάλλουν έναν τυχαίο κόμβο P είναι εξαεδρικά. Ακόμα, είναι γνωστό εκ των προτέρων πως ένας εσωτερικός κόμβος P περιβάλλεται από 8 εξάεδρα και 6 γειτονικούς κόμβους πρώτου βαθμού. Για έναν οριακό κόμβο P τα περιβάλλοντα εξάεδρα μπορεί να είναι 4, 2 ή 1 και οι γειτονικοί κόμβοι 5, 4 ή 3 άμα ο κόμβος είναι οριακός σε 1, 2 ή και 3 από τους άξονες i, j, k του πλέγματος, αντίστοιχα. Κάθε γειτονικό εξάεδρο προσφέρει στον όγκο ελέγχου ενός τυχαίου κόμβου P έναν όγκο, όπως απεικονίζεται στο σχήμα 3.1, που ορίζεται από το σημείο P, τους μεσόκομβους κάθε ακμής που συντρέχει στον κόμβο P, που συμβολίζονται με S_1, S_2 και S_3 στο σχήμα, τα κέντρα βάρους των πλευρών στις οποίες ανήκουν οι προαναφερθείσες ακμές, τα σημεία F_1, F_2 και F_3, καθώς και το κέντρο βάρους του εκάστοτε εξαέδρου, το σημείο G. Για λόγους ευκρίνειας παρουσιάζεται μόνο ένα από τα 8 γειτονικά εξάεδρα, αντί για ολόκληρο τον όγκο ελέγχου.

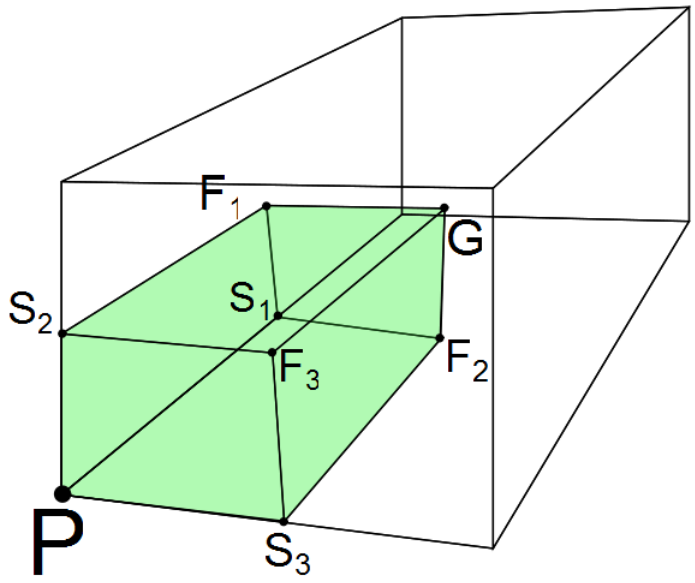
Είναι σημαντικό να αναφερθεί πως τα 4 σημεία που ορίζουν κάθε επιφάνεια δεν είναι απαραίτητα συνεπίπεδα, οπότε και πρέπει να βρεθεί μονοσήμαντος τρόπος διαχείρισης των επιφανειών αυτών ώστε να τηρείται η συνθήκη της μη αλληλοκάλυψης των όγκων ελέγχου. Η μέθοδος που χρησιμοποιήθηκε στην παρούσα εργασία είναι η διάσπαση κάθε επιφάνειας σε 2 τρίγωνα όπως φαίνεται στο σχήμα 3.2. Κατά την διάσπαση κάθε επιφάνειας χρειάζεται προσοχή, καθώς η διάσπαση πρέπει να ορίζεται μονοσήμαντα και πρέπει να γίνει με τέτοιο τρόπο ώστε 2 γειτονικά εξάεδρα να μην αλληλοκαλύπτονται, αλλά και να μην δημιουργείται κενό ανάμεσά τους.

3.3.2 Ολοκλήρωση στους όγκους ελέγχου

Στη συνέχεια, γίνεται ολοκλήρωση των εξισώσεων 3.5 στους όγκους ελέγχου. Η ολοκλήρωση του όγκου ελέγχου ενός τυχαίου κόμβου P δίνει:

$$\iiint\limits_{V^P} \frac{\partial \vec{U}}{\partial t} dV + \iiint\limits_{V^P} \frac{\partial \vec{F_r}}{\partial x_r} dV = 0 \tag{3.20}$$

Εφαρμόζοντας το θεώρημα Green-Gauss και θεωρώντας ότι $\iiint\limits_{V^P} \frac{\partial \vec{U}}{\partial t} dV = \left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P$, ισχύει:



Σχήμα 3.1: Προσφορά κάθε γειτονικού εξάεδρου στον όγκο ελέγχου.

$$\left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P + \iint_{\partial V^P} \vec{F}_r \hat{n}_r d(\partial V) = 0 \tag{3.21}$$

όπου V^P είναι ο όγκος του όγκου ελέγχου, ∂V^P η οριακή επιφάνεια αυτού και $\vec{\hat{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$ το κάθετο μοναδιαίο διάνυσμα στην οριακή επιφάνεια με φορά προς το εξωτερικό του. Θέτοντας :

$$\begin{aligned} \vec{\hat{H}} &= \vec{F}_r \hat{n}_r \\ \vec{H} &= \vec{F}_r n_r \end{aligned} \tag{3.22}$$

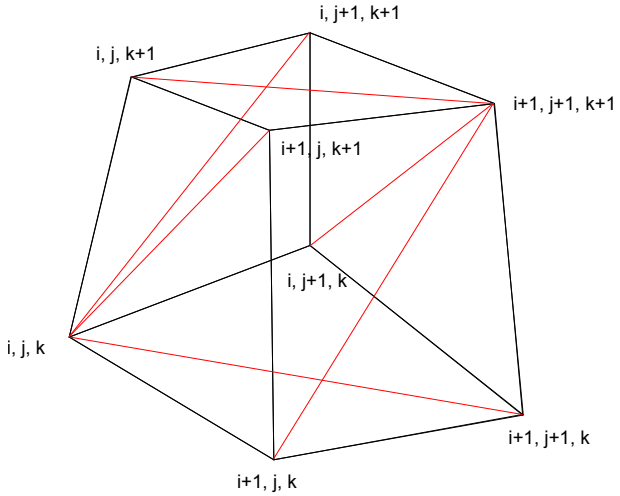
η έκφραση 3.21 γίνεται:

$$\left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P + \iint_{\partial V^P} \vec{\hat{H}} d(\partial V) = 0$$

ή

$$\left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P + \sum_{Q \in K(P)} \vec{\Phi}_{PQ} = 0 \tag{3.23}$$

όπου Q οι γειτονιτοί κόμβοι του P και το διάνυσμα ροής $\vec{\Phi}_{PQ}$ ορίζεται:



Σχήμα 3.2: Διάσπαση κάθε επιφάνειας ενός εξάεδρου στοιχείου σε 2 τρίγωνα.

$$\vec{\Phi}_{PQ} = \iint_{\partial V^P} \vec{\hat{H}} d(\partial V) \tag{3.24}$$

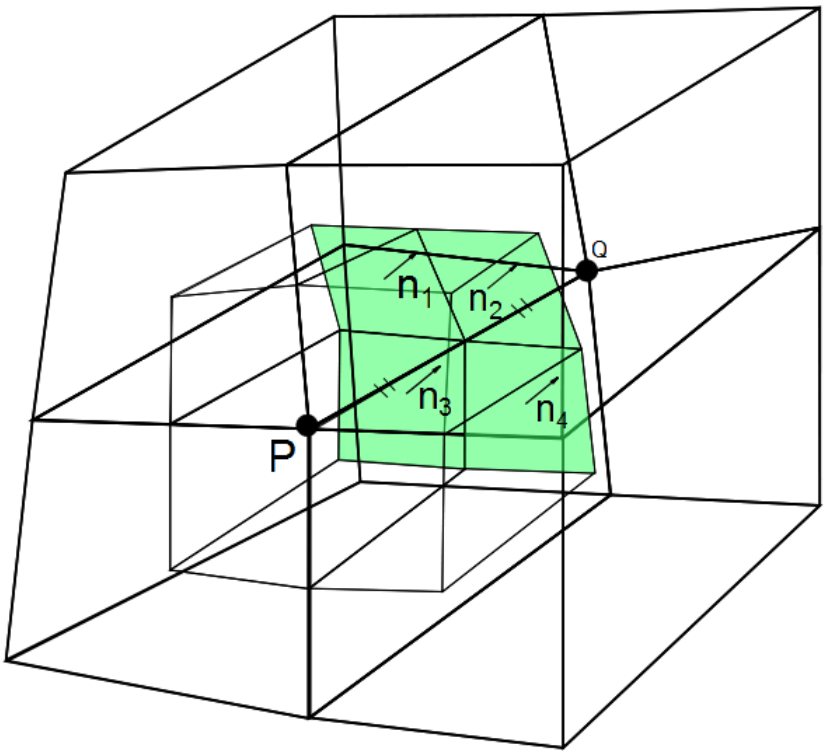
Η επιφάνεια ολοκλήρωσης για τον υπολογισμό του διανύσματος ροής $\vec{\Phi}_{PQ}$ αντιστοιχεί στο κοινό όριο των όγκων ελέγχου, του κόμβου P και του εκάστοτε γειτονικού κόμβου Q, όπως φαίνεται στο σχήμα 3.3.

3.3.3 Υπολογισμός διανύσματος ροής

Το διάνυσμα ροής $\vec{\Phi}_{PQ}$ υπολογίζεται σε κάθε ακμή και αφαιρείται ή προστίθεται ανάλογα, στον ισολογισμό της κυψέλης στην οποία αναφέρεται. Ο υπολογισμός του γίνεται σύμφωνα με τις τιμές των συντηρητικών μεταβλητών εκατέρωθεν του μέσου της ακμής PQ, οι οποίες υπολογίζονται συναρτήσει των αντίστοιχων τιμών στους κόμβους P και Q, με προεκβολή ακρίβειας δεύτερης τάξης. Επίσης συνυπολογίζεται το κάθετο διάνυσμα $\vec{n}_{PQ}$ που είναι το διανυσματικό άθροισμα των $\vec{n}_1, \vec{n}_2, \vec{n}_3$ και $\vec{n}_4$, όπως αυτά φαίνονται στο σχήμα 3.3. Όμως, όπως έχει αναφερθεί και παραπάνω, τα 4 σημεία που ορίζουν κάθε επιφάνεια σε κάθε στοιχειώδες εξάεδρο δεν είναι κατ' ανάγκη συνεπίπεδα. Έτσι, για τον υπολογισμό καθενός από τα διανύσματα αυτά, χρειάζεται η διαίρεση της επιφάνειας σε 2 τριγωνικές, επίσης όπως έχει αναφερθεί παραπάνω, ο υπολογισμός των κάθετων προς αυτές διανυσμάτων και η άθροισή τους, ανά δύο, για να προκύψει το αποτέλεσμα του σχήματος 3.3. Έτσι προκύπτει:

$$\vec{n}_{PQ} = \vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4 \tag{3.25}$$

$$\vec{\Phi}_{PQ} = \vec{f}\left(\vec{U}^L_{PQ}, \vec{U}^R_{PQ}, \vec{n}_{PQ}\right) \tag{3.26}$$



Σχήμα 3.3: Κοινό όριο των όγκων ελέγχου ανάμεσα στους κόμβους P και Q.

Πριν τη διατύπωση της έκφρασης του διανύσματος ροής, χρειάζεται να οριστεί το ιακωβιανό μητρώο του διανύσματος $\overrightarrow{F}_r$ ως προς τις συντηρητικές μεταβλητές $\overrightarrow{U}$.

$$A_r \hat{=} \frac{\partial \overrightarrow{F}_r}{\partial \overrightarrow{U}} \tag{3.27}$$

Ακόμα οριζεται:

$$\underline{A} \hat{=} A_r n_r \Rightarrow \underline{A} = \frac{\partial \overrightarrow{F}_r}{\partial \overrightarrow{U}} n_r = \frac{\partial (\overrightarrow{F}_r n_r)}{\partial \overrightarrow{U}} \Rightarrow \underline{A} = \frac{\partial \overrightarrow{H}}{\partial \overrightarrow{U}} \tag{3.28}$$

Από την καταστατική εξίσωση των τελείων αερίων εύκολα μπορεί να αποδειχθεί ότι ικανοποιείται η έκφραση $p = \varrho f(e)$. Σε συνδυασμό με τον τρόπο ορισμού του ιακωβιανού μητρώου A_r, το διάνυσμα της ροής $\overrightarrow{F}_r$ είναι ομογενής συνάρτηση πρώτου βαθμού, κύρια ιδιότητα των οποίων είναι:

$$\overrightarrow{F}_r = A_r \overrightarrow{U} \tag{3.29}$$

Ακολουθώντας την πορεία της εξαγωγής της σχέσης 3.28 εύκολα προκύπτει:

$$\overrightarrow{H} = \underline{A} \overrightarrow{U} \tag{3.30}$$

Υπολογισμός ιακωβιανού μητρώου $\underline{A}$

Επαναλαμβάνεται, για πρακτικούς λόγους, ο ορισμός του διανύσματος ροής $\overrightarrow{F}_r$

$$\overrightarrow{F}_r = \begin{bmatrix} \varrho u_r \\ \varrho u_1 u_r + p\delta_{1r} \\ \varrho u_2 u_r + p\delta_{2r} \\ \varrho u_3 u_r + p\delta_{3r} \\ u_r(\varrho E + p) \end{bmatrix}$$

Επομένως, σύμφωνα με τον ορισμό του $\overrightarrow{H}$ ισχύει:

$$\overrightarrow{H} = \overrightarrow{F}_r n_r = \begin{bmatrix} \varrho(u_r n_r) \\ \varrho u(u_r n_r) + p n_x \\ \varrho v(u_r n_r) + p n_y \\ \varrho w(u_r n_r) + p n_z \\ (\varrho E + p)(u_r n_r) \end{bmatrix} \tag{3.31}$$

Για την εύρεση του ιακωβιανού μητρώου εξυπηρετεί η μετονομασία των συντηρητικών μεταβλητών ως εξής :

$$\overrightarrow{U} = \begin{bmatrix} \varrho \\ \varrho u \\ \varrho v \\ \varrho w \\ \varrho E \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \end{bmatrix} \tag{3.32}$$

Έτσι, το διάνυσμα ροής $\overrightarrow{H}$, σύμφωνα με την ονομασία αυτή, γράφεται:

$$\overrightarrow{H} = \begin{bmatrix} \mu_2 n_x + \mu_3 n_y + \mu_4 n_z \\ \frac{\mu_2}{\mu_1}(\mu_2 n_x + \mu_3 n_y + \mu_4 n_z) + (\gamma - 1)\left[\mu_5 - \frac{1}{2}\frac{\mu_2^2+\mu_3^2+\mu_4^2}{\mu_1}\right] n_x \\ \frac{\mu_3}{\mu_1}(\mu_2 n_x + \mu_3 n_y + \mu_4 n_z) + (\gamma - 1)\left[\mu_5 - \frac{1}{2}\frac{\mu_2^2+\mu_3^2+\mu_4^2}{\mu_1}\right] n_y \\ \frac{\mu_4}{\mu_1}(\mu_2 n_x + \mu_3 n_y + \mu_4 n_z) + (\gamma - 1)\left[\mu_5 - \frac{1}{2}\frac{\mu_2^2+\mu_3^2+\mu_4^2}{\mu_1}\right] n_z \\ \left(\gamma\mu_5 - \frac{\gamma-1}{2}\frac{\mu_2^2+\mu_3^2+\mu_4^2}{\mu_1}\right)\frac{\mu_2 n_x+\mu_3 n_y+\mu_4 n_z}{\mu_1} \end{bmatrix} \tag{3.33}$$

Μετά από παραγώγιση κάθε όρου του διανύσματος ροής προκύπτει:

$$\frac{\partial H_1}{\partial \mu_1} = 0$$
$$\frac{\partial H_1}{\partial \mu_2} = n_x$$
$$\frac{\partial H_1}{\partial \mu_3} = n_y$$
$$\frac{\partial H_1}{\partial \mu_4} = n_z$$
$$\frac{\partial H_1}{\partial \mu_5} = 0$$

$$\frac{\partial H_2}{\partial \mu_1} = -u\left(u_r n_r\right) + \frac{\gamma - 1}{2}\left(u_r u_r\right) n_x$$
$$\frac{\partial H_2}{\partial \mu_2} = u_r n_r + (2 - \gamma) u n_x$$
$$\frac{\partial H_2}{\partial \mu_3} = u n_y - (\gamma - 1) v n_x$$
$$\frac{\partial H_2}{\partial \mu_4} = u n_z - (\gamma - 1) w n_x$$
$$\frac{\partial H_2}{\partial \mu_5} = (\gamma - 1) n_x$$

$$\frac{\partial H_3}{\partial \mu_1} = -v\left(u_r n_r\right) + \frac{\gamma - 1}{2}\left(u_r u_r\right) n_y$$
$$\frac{\partial H_3}{\partial \mu_2} = v n_x - (\gamma - 1) u n_y$$
$$\frac{\partial H_3}{\partial \mu_3} = u_r n_r + (2 - \gamma) v n_y$$
$$\frac{\partial H_3}{\partial \mu_4} = v n_z - (\gamma - 1) w n_y$$
$$\frac{\partial H_3}{\partial \mu_5} = (\gamma - 1) n_y$$

$$\frac{\partial H_4}{\partial \mu_1} = -w\left(u_r n_r\right) + \frac{\gamma-1}{2}\left(u_r u_r\right) n_z$$
$$\frac{\partial H_4}{\partial \mu_2} = w n_x - (\gamma-1) u n_z$$
$$\frac{\partial H_4}{\partial \mu_3} = w n_y - (\gamma-1) v n_z$$
$$\frac{\partial H_4}{\partial \mu_4} = u_r n_r + (2-\gamma) w n_z$$
$$\frac{\partial H_4}{\partial \mu_5} = (\gamma-1) n_z$$

$$\frac{\partial H_5}{\partial \mu_1} = \left[-\gamma E + (\gamma-1)\left(u_r u_r\right)\right]\left(u_r n_r\right)$$
$$\frac{\partial H_5}{\partial \mu_2} = \left[\gamma E - \frac{\gamma-1}{2}\left(u_r u_r\right)\right] n_x - (\gamma-1) u \left(u_r n_r\right)$$
$$\frac{\partial H_5}{\partial \mu_3} = \left[\gamma E - \frac{\gamma-1}{2}\left(u_r u_r\right)\right] n_y - (\gamma-1) v \left(u_r n_r\right)$$
$$\frac{\partial H_5}{\partial \mu_4} = \left[\gamma E - \frac{\gamma-1}{2}\left(u_r u_r\right)\right] n_z - (\gamma-1) w \left(u_r n_r\right)$$
$$\frac{\partial H_5}{\partial \mu_5} = \gamma\left(u_r n_r\right)$$

Οπότε είναι:

$$\underline{A}(:,1) = \begin{bmatrix} 0 \\ -u\left(u_r n_r\right) + \frac{\gamma-1}{2}\left(u_r u_r\right) n_x \\ -v\left(u_r n_r\right) + \frac{\gamma-1}{2}\left(u_r u_r\right) n_y \\ -w\left(u_r n_r\right) + \frac{\gamma-1}{2}\left(u_r u_r\right) n_z \\ \left[-\gamma E + (\gamma-1)\left(u_r u_r\right)\right]\left(u_r n_r\right) \end{bmatrix}$$

$$\underline{A}(:,2) = \begin{bmatrix} n_x \\ u_r n_r + (2-\gamma) u n_x \\ v n_x - (\gamma-1) u n_y \\ w n_x - (\gamma-1) u n_z \\ \left[\gamma E - \frac{\gamma-1}{2}\left(u_r u_r\right)\right] n_x - (\gamma-1) u \left(u_r n_r\right) \end{bmatrix}$$

$$\underline{A}(:,3) = \begin{bmatrix} n_y \\ u n_y - (\gamma-1) v n_x \\ u_r n_r + (2-\gamma) v n_y \\ w n_y - (\gamma-1) v n_z \\ \left[\gamma E - \frac{\gamma-1}{2}\left(u_r u_r\right)\right] n_y - (\gamma-1) v \left(u_r n_r\right) \end{bmatrix} \tag{3.34}$$

$$\underline{A}(:,4) = \begin{bmatrix} n_z \\ un_z - (\gamma-1)wn_x \\ vn_z - (\gamma-1)wn_y \\ u_r n_r + (2-\gamma)wn_z \\ \left[\gamma E - \frac{\gamma-1}{2}(u_r u_r)\right] n_z - (\gamma-1)w(u_r n_r) \end{bmatrix}$$

$$\underline{A}(:,5) = \begin{bmatrix} 0 \\ (\gamma-1)n_x \\ (\gamma-1)n_y \\ (\gamma-1)n_z \\ \gamma(u_r n_r) \end{bmatrix}$$

οι ιδιοτιμές του μητρώου $\underline{A}$ έχουν υπολογιστεί και είναι:

$$\begin{aligned} \lambda_1 &= \overrightarrow{u} \cdot \overrightarrow{n} \\ \lambda_2 &= \overrightarrow{u} \cdot \overrightarrow{n} \\ \lambda_3 &= \overrightarrow{u} \cdot \overrightarrow{n} \\ \lambda_4 &= \left(\overrightarrow{u} \cdot \overrightarrow{\hat{n}} + c\right)\left|\overrightarrow{n}\right| \\ \lambda_5 &= \left(\overrightarrow{u} \cdot \overrightarrow{\hat{n}} - c\right)\left|\overrightarrow{n}\right| \end{aligned} \tag{3.35}$$

Ακόμα, υπολογίζονται τα αριστερά και δεξιά ιδιοδιανύσματα του μητρώου $\underline{A}$. Τα δεξιά ιδιοδιανύσματα ικανοποιούν την σχέση $(\underline{A} - \lambda_k I_k) r_k = 0$, ενώ τα αριστερά ιδιοδιανύσματα ικανοποιούν την σχέση $l_k(\underline{A} - \lambda_k I) = 0$ $(k = 1, 2, 3, 4, 5)$. Με I συμβολίζεται ο μοναδιαίος πίνακας. Πλέον, το μητρώο $\underline{A}$ μπορεί εύκολα να γραφεί στην μορφή

$$\underline{A} = P \Lambda P^{-1} \tag{3.36}$$

όπου Λ διαγώνιος πίνακας με τις ιδιοτιμές του $\underline{A}$ και

$$P = \begin{bmatrix} r_1 & r_2 & r_3 & r_4 & r_5 \end{bmatrix}, P^{-1} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ l_5 \end{bmatrix} \tag{3.37}$$

Ακόμα, ορίζονται τα μητρώα:

$$\underline{A}^+ = P \Lambda^+ P^{-1}, \underline{A}^- = P \Lambda^- P^{-1} \tag{3.38}$$

$$|\underline{A}| = \underline{A}^+ - \underline{A}^- \tag{3.39}$$

όπου το μητρώο Λ^+ περιέχει τις θετικές ιδιοτιμές, ενώ το μητρώο Λ^- τις αρνητικές ιδιοτιμές.

Έχοντας ορίσει τα ιακωβιανά μητρώα, ακολουθεί ο ορισμός του διανύσματος της ροής Φ_{PQ}, σύμφωνα με το σχήμα του Roe [24]. Έτσι, ισχύει:

$$
\begin{aligned}
\overrightarrow{\Phi}_{PQ} &= \frac{1}{2}\left[\overrightarrow{H}\left(\overrightarrow{U}^R_{PQ}, \overrightarrow{n}_{PQ}\right) + \overrightarrow{H}\left(\overrightarrow{U}^L_{PQ}, \overrightarrow{n}_{PQ}\right)\right] - \frac{1}{2}\left|\underline{\tilde{A}}_{PQ}\right|\left(\overrightarrow{U}^R_{PQ} - \overrightarrow{U}^L_{PQ}\right) \Rightarrow \\
\overrightarrow{\Phi}_{PQ} &= \frac{1}{2}\left[\underline{A}_R\overrightarrow{U}^R_{PQ} + \underline{A}_L\overrightarrow{U}^L_{PQ}\right] - \frac{1}{2}\left|\underline{\tilde{A}}_{PQ}\right|\left(\overrightarrow{U}^R_{PQ} - \overrightarrow{U}^L_{PQ}\right) \Rightarrow \\
\overrightarrow{\Phi}_{PQ} &= \frac{1}{2}\left(\underline{A}_R - \left|\underline{\tilde{A}}_{PQ}\right|\right)\overrightarrow{U}^R_{PQ} + \frac{1}{2}\left(\underline{A}_L - \left|\underline{\tilde{A}}_{PQ}\right|\right)\overrightarrow{U}^L_{PQ} \Rightarrow \\
\overrightarrow{\Phi}_{PQ} &= \mathcal{A}_R\overrightarrow{U}^R_{PQ} + \mathcal{A}_L\overrightarrow{U}^L_{PQ}
\end{aligned}
\tag{3.40}
$$

όπου $|\underline{\tilde{A}}_{PQ}|$ είναι το μητρώο που προκύπτει από τις απόλυτες τιμές των ιδιοτιμών του $\underline{A}$, σχέση 3.39, υπολογισμένο με βάση τις κατά Roe μέσες τιμές των πρωτευουσών μεταβλητών. Αυτές δίνονται:

$$
\overrightarrow{\tilde{U}}_{PQ} = \begin{bmatrix} \tilde{\varrho} & \tilde{u} & \tilde{v} & \tilde{w} & \tilde{p} \end{bmatrix}^T \tag{3.41}
$$

Για τον υπολογισμό των συνιστωσών, πέρα της πίεσης, χρησιμοποιείται η σχέση 3.42, ενώ για τον υπολογισμό της μέσης, κατά Roe πίεσης, υπολογίζεται η μέση τιμή της ολικής ενθαλπίας σύμφωνα με την σχέση 3.43 και στη συνέχεια υπολογίζεται η πίεση.

$$
\overrightarrow{\tilde{U}}_{PQ} = \frac{\sqrt{\varrho_L}\overrightarrow{U}_L + \sqrt{\varrho_R}\overrightarrow{U}_R}{\sqrt{\varrho_L} + \sqrt{\varrho_R}} \tag{3.42}
$$

$$
h_t = \frac{\gamma p}{(\gamma - 1)\varrho} + \frac{1}{2}(u_r u_r) \tag{3.43}
$$

3.3.4 Αύξηση της ακρίβειας του σχήματος και χρήση περιοριστών

Στην μέχρι τωρα ανάλυση, έχει γίνει αναφορά στις τιμές εκατέρωθεν του μέσου της ακμής PQ, αλλά δεν έχει αναφερθεί ο τρόπος υπολογισμού των τιμών αυτών. Ο τρόπος υπολογισμού των τιμών αυτών σχετίζεται με την ακρίβεια της χωρικής διακριτοποίησης των εξισώσεων ροής. Η πιο απλή λύση είναι η θεώρηση

$$
\begin{aligned}
\overrightarrow{U}^L_{PQ} &= \overrightarrow{U}_P \\
\overrightarrow{U}^R_{PQ} &= \overrightarrow{U}_Q
\end{aligned}
\tag{3.44}
$$

η οποία και αντιστοιχεί σε χωρική διακριτοποίηση πρώτης τάξης. Εναλλακτικά, με χρήση του θεωρήματος του Taylor μπορεί να αυξηθεί η τάξη της χωρικής διακριτοποίησης. Για χωρική διακριτοποίηση δεύτερης τάξης ακρίβειας, ο τύπος υπολογισμού των τιμών εκατέρωθεν του μέσου της ακμής PQ είναι:

$$
\begin{aligned}
\overrightarrow{U}^L_{PQ} &= \overrightarrow{U}_P + \frac{1}{2}\overrightarrow{PQ}\cdot\left(\nabla\overrightarrow{U}\right)_P \\
\overrightarrow{U}^R_{PQ} &= \overrightarrow{U}_Q - \frac{1}{2}\overrightarrow{PQ}\cdot\left(\nabla\overrightarrow{U}\right)_Q
\end{aligned}
\tag{3.45}
$$

Βασικό μειονέκτημα της αύξησης της ακρίβειας της χωρικής διακριτοποίησης με αυτόν τον τρόπο, είναι η ανάγκη για υπολογισμό της πρώτης παραγώγου των μεταβλητών της ροής.

Περιοριστής Van Leer-Van Albada [25]

Ο περιοριστής αυτός, αποτελεί τροποποίηση του αρχικού περιοριστη του Van Leer, ώστε σε περιοχές μικρής κλίσης να παίρνει τιμές πλησιέστερες στη μονάδα, δηλαδή να επεμβαίνει λιγότερο στην υπολογισμένη κατά Taylor προεκβολή σε περιοχές που δεν υπάρχει λόγος περιορισμού της κλίσης. Είναι ένας από τους πρώτους περιοριστές που αναπτύχθηκαν, είναι απλός στην εφαρμογή και δεν επηρεάζει ιδιαίτερα την σύγκλιση των εξισώσεων ροής. Τα βασικότερα μειονεκτήματά του είναι ο μονοδιάστατος χαρακτήρας του, δηλαδή η εξάλειψη των ταλαντώσεων γίνεται μόνο κατά τη διεύθυνση της προεκβολής, δηλαδή αυτήν της ακμής στην οποία υπολογίζεται το διάνυσμα ροής, και η έλλειψη μαθηματικού μηχανισμού απενεργοποίησής του σε περιοχές που δεν είναι απαραίτητος, όπως σε περιοχές ελεύθερης ροής, ενώ μπορεί να μειώσει την τάξη ακρίβειας της λύσης σε περιοχές του πεδίου όπου υπάρχουν ακρότατα στη λύση. Με χρήση του περιοριστη Van Leer-Van Albada, ο τύπος υπολογισμού των τιμών εκατέρωθεν του μέσου της ακμής PQ είναι:

$$\begin{aligned}
\overrightarrow{U}^L_{PQ} &= \overrightarrow{U}_P + \frac{1}{2} LIM\left[\left(2\left(\frac{\partial \overrightarrow{U}}{\partial x_r}\right)^P x_r^{PQ} - \Delta\overrightarrow{U}^{PQ}\right), \Delta\overrightarrow{U}^{PQ}\right] \\
\overrightarrow{U}^R_{PQ} &= \overrightarrow{U}_Q - \frac{1}{2} LIM\left[\left(2\left(\frac{\partial \overrightarrow{U}}{\partial x_r}\right)^Q x_r^{PQ} - \Delta\overrightarrow{U}^{PQ}\right), \Delta\overrightarrow{U}^{PQ}\right]
\end{aligned} \tag{3.46}$$

όπου

$$\Delta\overrightarrow{U}^{PQ} = \overrightarrow{U}^Q - \overrightarrow{U}^P$$

και

$$LIM(a,b) = \begin{cases} \frac{(a^2+\eta)b+(b^2+\eta)a}{a^2+b^2+2\eta}, & ab > 0 \\ 0, & ab <= 0 \end{cases}$$

3.3.5 Διακριτοποίηση του χρονικού όρου και επιλογή του χρονικού βήματος

Στη παρούσα εργασία αναλύεται η μέθοδος επίλυσης χρονικά μόνιμων ροών, όπως παρατηρείται η ύπαρξη του χρονικού όρου $\left(\frac{\partial \overrightarrow{U}}{\partial t}\right)_P V^P$. Ο χρονικός όρος αποτελεί ψευδοχρονικό όρο και έχει προστεθεί για διευκόλυνση της σύγκλισης των εξισώσεων, σύμφωνα με την τεχνική της χρονοπροέλασης. Η διακριτοποίησή του γίνεται μέσω πρώτης τάξης σχήματος ανάντι διαφόρισης του Euler

$$\left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P = \frac{V^P}{\Delta t^P}\Delta\vec{U}^P \tag{3.47}$$

όπου $\Delta\vec{U}^P = \left(\vec{U}^P\right)^{n+1} - \left(\vec{U}^P\right)^n$ (ο εκθέτης n αντιστοιχεί στο τρέχον χρονικό βήμα). Για επιτάχυνση της σύγκλισης εφαρμόζεται η τεχνική του τοπικού χρονικού βήματος [26]. Ο τύπος από τον οποίο προκύπτει το ψευδοχρονικό βήμα σε κάθε κόμβο είναι:

$$\Delta t^P = CFL\frac{V^P}{C} \tag{3.48}$$

όπου CFL είναι ο αριθμός Courant-Friedrichs-Lewy [27] και ο όρος C υπολογίζεται από την σχέση

$$C = \left(|u_r^P| + c^P\right) S_r^P \tag{3.49}$$

όπου S_i^P είναι η προβολή των τμημάτων που απαρτίζουν τα όρια του όγκου ελέγχου του κόμβου P κατά την κατεύθυνση i, δηλαδή

$$S_r^P = \frac{1}{2}\sum_{Q\in K(P)} |n_r^{PQ}| \tag{3.50}$$

3.4 Οριακές συνθήκες

Ολόκληρη η ανάλυση που προηγήθηκε θεωρούσε έναν τυχαίο κόμβο P, ο οποίος ήταν εσωτερικός του πλέγματος. Όμως, σε περίπτωση οριακού κόμβου πρέπει να συμπεριληφθεί ένας ακόμα όρος, το διάνυσμα της ροής που εξέρχεται προς το περιβάλλον από τον όγκο ελέγχου. Έτσι, η ολοκλήρωση των εξισώσεων 3.5 καταλήγει στη μορφή:

$$\left(\frac{\partial \vec{U}}{\partial t}\right)_P V^P + \sum_{Q\in K(P)} \vec{\Phi}_{PQ} + \vec{\Phi}_{\text{Όριο χωρίου ροής}} = 0 \tag{3.51}$$

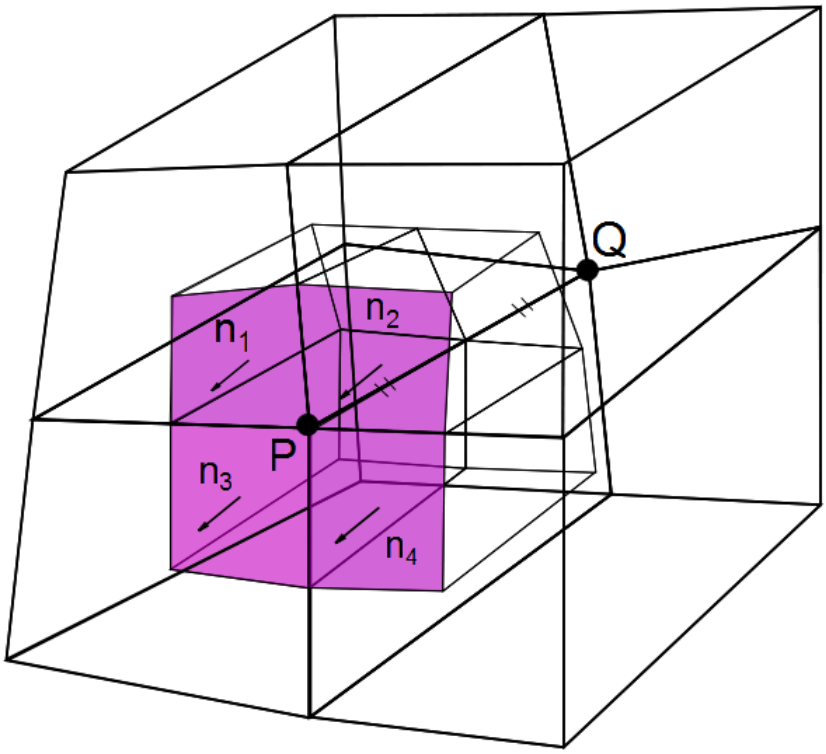
Το οριακό διάνυσμα ροής υπολογίζεται ανάλογα με το είδος του ορίου και στη συνέχεια θα παρουσιαστούν οι τρόποι υπολογισμού για οριακούς κόμβους τοιχωμάτων, ειδόδου-εξόδου, επιφάνειας συμμετρίας ή και περιοδικότητας.

3.4.1 Στερεά Τοιχώματα

Στα στερεά τοιχώματα, για ατριβείς ροές, επιβάλλεται η συνθήκη μη-εισχώρησης ($\vec{u}\cdot\vec{n} = 0$). Η επιβολή γίνεται με ασθενή διατύπωση, δηλαδή με εισαγωγή της συνθήκης στο διάνυσμα της ροής, το οποίο, κατά μήκος των στερεών τοιχωμάτων, λαμβάνει τη μορφή:

$$\overrightarrow{\Phi}_{\text{όριο}} = \overrightarrow{F}_r^P n_r^P = \begin{bmatrix} \varrho(u_r n_r) \\ \varrho u(u_r n_r) + p n_x \\ \varrho v(u_r n_r) + p n_y \\ \varrho w(u_r n_r) + p n_z \\ (\varrho E + p)(u_r n_r) \end{bmatrix}_P = \begin{bmatrix} 0 \\ p n_x \\ p n_y \\ p n_z \\ 0 \end{bmatrix}_P \tag{3.52}$$

με $\overrightarrow{n} = \overrightarrow{n_1} + \overrightarrow{n_2} + \overrightarrow{n_3} + \overrightarrow{n_4}$, όπου τα διανύσματα $\overrightarrow{n_1}$, $\overrightarrow{n_2}$, $\overrightarrow{n_3}$ και $\overrightarrow{n_4}$ αντιστοιχούν στα διανύσματα από το όριο του όγκου ελέγχου προς το τοίχωμα, όπως φαίνεται στο σχήμα 3.4.



Σχήμα 3.4: Όγκος ελέγχου οριακού κόμβου.

3.4.2 Όρια εισόδου και εξόδου της ροής

Έχει αποδειχθεί ότι το πρόσημο των ιδιοτιμών του μητρώου $\underline{A}$ καθορίζει την κατεύθυνση μεταφοράς της «πληροφορίας» μέσα στη ροή. Από τις ιδιοτιμές του μητρώου $\underline{A}$, όπως έχουν υπολογιστεί στην σχέση 3.35, είναι θετικές οι λ_1, λ_2, λ_3 και λ_4 ενώ η λ_5 εξαρτάται από το αν η ροή είναι υποηχητική ή υπερηχητική. Για τις θετικές ιδιοτιμές, η αντίστοιχη «πληροφορία» μεταφέρεται μαζί με την ροή, ενώ όταν η ιδιοτιμή είναι αρνητική, η «πληροφορία» ταξιδεύει αντίθετα από την τοπική ταχύτητα της ροής. Έτσι, για το κλείσιμο των εξισώσεων ροής, απαιτούνται 4 μεγέθη στην είσοδο της ροής και 1 στην έξοδο σε υποηχητικές ροές, ενώ σε υπερηχητικές ροές απαιτούνται

5 μεγέθη στην είδοδο της ροής. Σε εφαρμογές εξωτερικής αεροδυναμικής, τα μεγέθη αυτά είναι η πυκνότητα (ϱ_∞) και το μέτρο ($\left|\overrightarrow{u}_\infty\right|$) και οι γωνίες θ_1 και θ_2 της επ΄ άπειρον ταχύτητας, αλλά και ο αριθμός Mach της επ΄ άπειρον ροής. Αντιθετα, εφαρμογές εσωτερικής αεροτομής δίνονται οι τιμές της ολικής πίεσης (p_t) και θερμοκρασίας (T_t) στην είσοδο της ροής, τις γωνίες θ_1 και θ_2 της ταχύτητας στην είσοδο της ροής και την τιμή της στατικής πίεσης στην έξοδο της ροής, για υποηχητική ροή, ή στην είσοδο, για υπερηχητική ροή. Συχνά, στην πράξη, αντί για την στατική πίεση, δίνεται ο αριθμός Mach, ισεντροπικός εξόδου σε υποηχητικές ροές και εισόδου σε υπερηχητικές, από τον οποίο και υπολογίζεται η τιμή της στατικής πίεσης.

Το διάνυσμα ροής στους οριακού κόμβους εισόδου ή εξόδου υπολογίζεται σύμφωνα με το ανάντι σχήμα πρώτης τάξης των Steger-Warming [28], το οποίο για έναν οριακό κόμβο P γράφεται:

$$\overrightarrow{\Phi}^P_{out} = \underline{A}^+_P \overrightarrow{U}_P + \underline{A}^-_P \overrightarrow{U}_{out} \tag{3.53}$$

όπου με «out» συμβολίζεται ένας υποθετικός κόμβος, εξωτερικά του πεδίου ροής, στον οποίο και επιβάλλονται οι οριακές συνθήκες.

- **Εξωτερική Αεροδυναμική**

Όπως έχει αναφερθεί νωρίτερα, σε εφαρμογές εξωτερικής αεροδυναμικής, δηλαδή σε εφαρμογές που η ροή γύρω από ένα αεροδυναμικό σώμα επηρεάζεται μόνο από την παρουσία του ιδίου, τα μεγέθη που συνήθως δίνονται στο επ΄ άπειρο όριο για το κλείσιμο των εξισώσεων ροής είναι η πυκνότητα, το διάνυσμα της ταχύτητας και ο αριθμός $Mach$ της επ΄ άπειρον ροής $\left(\varrho_\infty,\ \left|\overrightarrow{u}_\infty\right|,\ \theta_{1\infty},\ \theta_{2\infty},\ M_\infty\right)$. Επομένως :

$$\begin{aligned}
\varrho_{out} &= \varrho_\infty \\
(\varrho u)_{out} &= \varrho_\infty \left|\overrightarrow{u}_\infty\right| \cos\theta_{1\infty} \cos\theta_{2\infty} \\
(\varrho v)_{out} &= \varrho_\infty \left|\overrightarrow{u}_\infty\right| \sin\theta_{1\infty} \\
(\varrho w)_{out} &= \varrho_\infty \left|\overrightarrow{u}_\infty\right| \cos\theta_{1\infty} \sin\theta_{2\infty}
\end{aligned}$$

$$\left.\begin{aligned}
M_\infty &= \frac{\left|\overrightarrow{u}_\infty\right|}{c} = \frac{\left|\overrightarrow{u}_\infty\right|}{\sqrt{\gamma(\gamma-1)T_\infty}} \\
\frac{p_{out}}{\varrho_\infty} &= (\gamma-1)T_\infty
\end{aligned}\right\} \Rightarrow M_\infty = \left|\overrightarrow{u}_\infty\right| \sqrt{\frac{\varrho_\infty}{p_{out}}} \Rightarrow p_{out} = \varrho_\infty \left(\frac{\left|\overrightarrow{u}_\infty\right|}{M_\infty}\right)^2$$

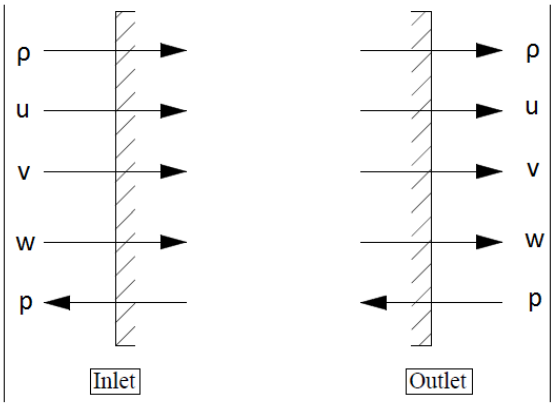
$$E_{out} = \frac{p_{out}}{\gamma-1} + \frac{1}{2}\varrho_{far}\left|\overrightarrow{u}_\infty\right|^2$$

- **Εσωτερική Αεροδυναμική**

Προηγουμένως έγινε αναφορά για μεταφορά «πληροφορίας» προς κατεύθυνση ανάλογα το πρόσημο της αντίστοιχης ιδιοτιμής. Ας θεωρηθεί, εδώ, ως «πληροφορία»

οι συντηρητικές μεταβλητές της ροής. Έχει αποδειχθεί όμως ότι αν γράψουμε τις εξισώσεις Euler συναρτήσει των πρωτευουσών μεταβλητών με ανάλογη διαδικασία προκύπτει ένα αντίστοιχο μητρώο $\underline{A}$ με ιδιοτιμές ίδιες με εκείνες που παρουσιάστηκαν νωρίτερα. Δηλαδή το πρόσημο των ιδιοτιμών μπορεί να δώσει τη κατεύθυνση που ακολουθούν οι πρωτεύουσες μεταβλητές της ροής στο εσωτερικό του πεδίου ροής.

Στην περίπτωση υποηχητικής ροής χρειάζεται να ορισθούν τέσσερα μεγέθη στην είσοδο $(p_t, T_t, \theta_1, \theta_2)$ και ένα στην έξοδο (M_{is}). Τα υπόλοιπα στοιχεία των θεωρητικών κόμβων θα προκύψουν από το εσωτερικό του πεδίου ροής. Δηλαδή:



Είσοδος

$$p_{out} = p_P$$

$$T_{out} = T_t \left(\frac{p_{out}}{p_t} \right)^{\frac{\gamma-1}{\gamma}}$$

$$T_t = T + \frac{1}{2\gamma} \left| \vec{u}_{out} \right|^2 \Rightarrow \left| \vec{u}_{out} \right| = \sqrt{2\gamma (T_t - T)}$$

$$u_{out} = \left| \vec{u}_{out} \right| \cos\theta_1 \cos\theta_2$$

$$v_{out} = \left| \vec{u}_{out} \right| \sin\theta_1$$

$$w_{out} = \left| \vec{u}_{out} \right| \cos\theta_1 sin\theta_2$$

Επομένως

$$\varrho_{out} = \frac{p_{out}}{(\gamma - 1) T_{out})}$$

$$(\varrho u)_{out} = \varrho_{out} u_{out}$$

$$(\varrho v)_{out} = \varrho_{out} v_{out}$$

$$(\varrho w)_{out} = \varrho_{out} w_{out}$$

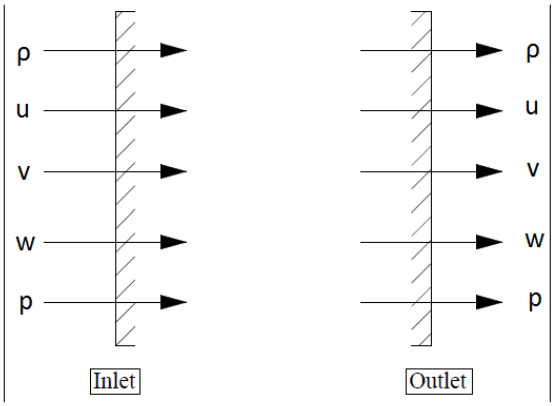
$$E_{out} = \frac{p_{out}}{\gamma - 1} + \frac{1}{2} \varrho_{out} \left| \vec{u}_{out} \right|^2$$

΄Εξοδος

$$p_{out} = p_t \left(1 + \frac{\gamma - 1}{2} M_{is}^2\right)^{-\frac{\gamma}{\gamma-1}}$$

$$\begin{aligned}
\varrho_{out} &= \varrho_P \\
(\varrho u)_{out} &= (\varrho u)_P \\
(\varrho v)_{out} &= (\varrho v)_P \\
(\varrho w)_{out} &= (\varrho w)_P \\
E_{out} &= \frac{p_{out}}{\gamma - 1} + \frac{1}{2}\varrho_P \left(u_P^2 + v_P^2\right)
\end{aligned}$$

Αντίθετα σε υπερηχητικές ροές όπου και οι πέντε ιδιοτιμές είναι θετικές ουσιαστικά θα δοθούν όλα τα στοιχεία που απαιτούνται για τον υπολογισμό των συντηρητικών μεταβλητών στον ψευδο-κόμβο εισόδου, ενώ οι αντίστοιχες τιμές του ψευδο-κόμβου εξόδου ταυτίζονται με εκείνες του αντίστοιχου οριακού κόμβου εφόσον η πληροφορία μεταφέρεται εξ ολοκλήρου προς τη κατεύθυνση της ροής . Τα μεγέθη που δίνονται σε τέτοιου είδους εφαρμογές είναι όπως και πριν η ολική πίεση (p_t), η θερμοκρασία (T_t) και οι γωνίες θ_1 και θ_2. Αντίθετα ο αριθμός *Mach* αντιστοιχεί στην είσοδο.



Είσοδος

$$
\begin{aligned}
p_{out} &= p_t \left(1 + \frac{\gamma - 1}{2} M^2\right)^{-\frac{\gamma}{\gamma-1}} \\
T_{out} &= T_t \left(\frac{p_{out}}{p_t}\right)^{\frac{\gamma-1}{\gamma}} \\
\left|\overrightarrow{u}_{out}\right| &= \sqrt{2\gamma(T_t - T_{out})} \\
u_{out} &= \left|\overrightarrow{u}_{out}\right| \cos\theta_1 \cos\theta_2 \\
v_{out} &= \left|\overrightarrow{u}_{out}\right| \sin\theta_1 \\
w_{out} &= \left|\overrightarrow{u}_{out}\right| \cos\theta_1 \sin\theta_2
\end{aligned}
$$

$$
\begin{aligned}
\varrho_{out} &= \frac{p_{out}}{(\gamma - 1) T_{out}} \\
(\varrho u)_{out} &= \varrho_{out} u_{out} \\
(\varrho v)_{out} &= \varrho_{out} v_{out} \\
(\varrho w)_{out} &= \varrho_{out} w_{out} \\
E_{out} &= \frac{p_{out}}{\gamma - 1} + \frac{1}{2} \varrho_{out} \left|\overrightarrow{u}_{out}\right|^2
\end{aligned}
$$

Έξοδος

$$
\begin{aligned}
\varrho_{out} &= \varrho_P \\
(\varrho u)_{out} &= (\varrho u)_P \\
(\varrho v)_{out} &= (\varrho v)_P \\
(\varrho w)_{out} &= (\varrho w)_P \\
E_{out} &= \frac{p_P}{\gamma - 1} + \frac{1}{2} \varrho_P (u_r u_r)_P
\end{aligned}
$$

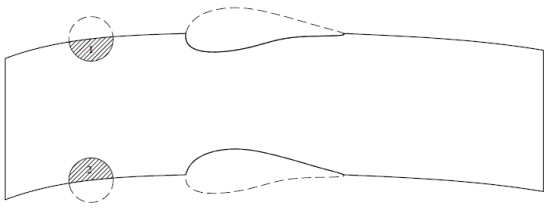
3.4.3 Αξονική συμμετρία

Αρκετά συχνά συναντώνται χωρία ροής συμμετρικά ως προς κάποιο επίπεδο. Στις περιπτώσεις αυτές για προφανείς λόγους συμφέρει να χωρίζεται το χωρίο στα δύο συμμετρικά όμοια κομμάτια του και να γίνεται αριθμητική επίλυση της ροής σε ένα από αυτά. Ο υπολογισμός λοιπόν του διανύσματος ροής $\left(\overrightarrow{\Phi}\right)$ που εξέρχεται από όρια συμμετρίας είναι ίδιος με εκείνον στα στερεά τοιχώματα. Αυτό συμβαίνει επειδή η ύπαρξη της συμμετρίας υπαγορεύει το διάνυσμα της ταχύτητας να είναι εφαπτομενικό στο όριο, δηλαδή να ισχύει $\overrightarrow{u} \cdot \overrightarrow{n} = 0$, κάτι το οποίο θυμίζει έντονα τη συνθήκη μη εισχώρησης για τα ατριβή ρευστά κατά μήκος των στερεών τοιχωμάτων.

3.4.4 Περιοδικά όρια

Υπολογιστικά χωρία όπως λ.χ. οι πτερυγώσεις στροβιλομηχανών είναι χαρακτηριστικά παραδείγματα χωρίων με περιοδικά όρια. Στα χωρία αυτά οι υπολογιστικές κυψέλες του ενός περιοδικού ορίου συμπληρώνουν τις κυψέλες των αντίστοιχων κόμβων του άλλου περιοδικού ορίου. Οπότε στις περιπτώσεις αυτές δεν είναι απαραίτητος ο υπολογισμός των εξερχόμενων διανυσμάτων ροής από τα περιοδικά όρια του χωρίου, καθώς στα διανύσματα ροής που έχουν υπολογιστεί για μία κυψέλη προστίθενται τα διανύσματα που έχουν υπολογιστεί για την κυψέλη του αντίστοιχου κόμβου του άλλου περιοδικού ορίου. Τα παραπάνω γίνονται εύκολα κατανοητά από το ακόλουθο σχήμα όπου παρίσταται μία επιφάνεια από πτερύγιο σε πτερύγιο. Το πάνω και κάτω όριο του χωρίου (εκτός των τοιχωμάτων των αεροτομών) αποτελούν περιοδικά όρια εφόσον το εν λόγω χωρίο επαναλαμβάνεται και προς τα πάνω και προς τα κάτω ώστε να συμπληρωθεί μία επιφάνεια S_1 μίας στροβιλομηχανής. Από το σχήμα φαίνεται ότι ουσιαστικά αυτό που «λείπει» από την κυψέλη 1 είναι η κυψέλη 2 και αντιστρόφως. Επιπλέον αφού το χωρίο επαναλαμβάνεται τα χαρακτηριστικά της ροής πάνω στο κάτω όριο του χωρίου ροής πρέπει να είναι τα ίδια με εκείνα εξωτερικά του χωρίου, πάνω από το πάνω όριο. Δηλαδή, η ένωση των δύο κυψελών δομεί την κυψέλη που θα σχηματίζονταν στους αντίστοιχους κόμβους αν ως υπολογιστικό χωρίο χρησιμοποιούσαμε ολόκληρη την αλληλουχία αεροτομών πτερυγίων κατά την κατεύθυνση του βήματος (αντί μόνο μίας πτερύγωσης). Για τον λόγο αυτό, το εξερχόμενο από το όριο του χωρίου διάνυσμα ροής στη περιοχή της κυψέλης 1 ισούται με το άθροισμα των υπολογισμένων διανυσμάτων ροής γύρω από την κυψέλη 2, και αντιστρόφως.

Τα παραπάνω ισχύουν σε εφαρμογές γραμμικών πτερυγώσεων. Σε εφαρμογές περιφερειακών πτερυγώσεων επαναλαμβάνονται τα παραπάνω, με μόνη διαφορά ότι η περιοδικότητα εμφανίζεται στις πολικές συντεταγμένες και, ως εκ τούτου, χρειάζεται περιστροφή των χωρίων πριν την άθροισή τους, κατά τον άξονα της ροής.



Σχήμα 3.5: Περιοδικό χωρίο ροής.

3.5 Επίλυση διακριτοποιημένων εξισώσεων

3.5.1 Μέθοδος αριθμητικής επίλυσης

Οι διακριτοποιημένες εξισώσεις μπορούν να ξαναγραφούν με χρήση του τελεστή υπολοίπου $\overrightarrow{R}$ στην παρακάτω δέλτα-μορφή:

$$\frac{V^P}{\Delta t^P}\Delta\overrightarrow{U}^P + \overrightarrow{R}^{p,n+1} = 0 \tag{3.54}$$

όπου n το παρών ψευδοχρονικό βημα.

Η ανανέωση των μεταβλητών σε κάθε ψευδοχρονικό βήμα γίνεται με χρήση σημειακά πεπλεγμένου σχήματος και εσωτερικών επαναλήψεων του σημειακά πεπλεγμένου επιλυτή Jacobi. Με γραμμικοποίηση του τελεστή υπολοίπου $\overrightarrow{R}$ προκύπτει το παρακάτω σχήμα για την νέα επανάληψη του ψευδοχρονικού βήματος

$$\left[\frac{V^P}{\Delta t^P}I + \frac{\partial\overrightarrow{R}^P}{\partial\overrightarrow{U}^P}\right]\Delta\overrightarrow{U}^P + \sum_{Q\in K_P}\left(\frac{\partial\overrightarrow{R}^P}{\partial\overrightarrow{U}^Q}\right)\Delta\overrightarrow{U}^Q = -\overrightarrow{R}^P \tag{3.55}$$

ή εναλλακτικά, σε τανυστική γραφή

$$\left[\frac{V^P}{\Delta t^P}\delta_{nk}\delta_{PK} + \frac{\partial R_n^P}{\partial U_k^K}\right]\Delta U_k^K = -R_n^P \tag{3.56}$$

όπου K είναι όλοι οι κόμβοι που συμμετέχουν στην εξίσωση του κόμβου P συμπεριλαμβανόμενου και του ίδιου. Το παραπάνω γραμμικοποιημένο σύστημα επιλύεται με την επαναληπτική μέθοδο Jacobi. Η επιλογή αυτή δικαιολογείται από το γεγονός ότι η μέθοδος επωφελείται από τη διαγώνια κυριαρχία που παρέχει το ανάντι σχήμα διακριτοποίησης [29] και από το γεγονός ότι προσφέρεται για παραλληλοποίηση, στοιχείο πολύ σημαντικό καθώς ο επιλύτης προορίζεται για GPU η οποία και λειτουργεί εντόνως παράλληλα. Η γραμμικοποίηση που χρησιμοποιειται αποτελεί προσέγγιση της ακριβούς και επιλέγεται έτσι ώστε να ενισχύεται η διαγώνια κυριαρχία, προκειμένου να διευκολυνθεί η σύγκλιση της μεθόδου και να μη αυξηθούν υπερβολικά οι απαιτήσεις σε μνήμη υπολογιστή. Τα παραπάνω επιτυγχάνονται αν δε ληφθούν οι μη-διαγώνιες συνεισφορές όλων των κόμβων πέρα από τους πρώτους γείτονες.

3.5.2 Αριστερό μέλος του όρου μεταφοράς

Η διαφόριση ως προς τις συντηρητικές μεταβλητές του διανύσματος ροής, το οποίο φαίνεται στην εξίσωση 3.40, με το οποίο συνεισφέρει μία ακμή PQ δίνει την παρακάτω έκφραση:

$$\frac{\partial\Phi_n^{PQ}}{\partial U_k^K} = \mathcal{A}_{nj}^L\frac{\partial U_j^L}{\partial U_k^K} + \mathcal{A}_{nj}^R\frac{\partial U_j^R}{\partial U_k^K} + \frac{\partial\mathcal{A}_{nj}^L}{\partial U_k^K}U_j^L + \frac{\partial\mathcal{A}_{nj}^R}{\partial U_k^K}U_j^R \tag{3.57}$$

Ώς προς τα διανύσματα της ροής, το αριστερό μέλος των εξισώσεων 3.55 διαμορφώνεται υπό τις εξής δύο παραδοχές :

- Οι όροι $\frac{\partial \mathcal{A}^L_{nj}}{\partial U^K_k} U^L_j$ και $\frac{\partial \mathcal{A}^R_{nj}}{\partial U^K_k} U^R_j$ θεωρούνται πρακτικά αμελητέοι. Η παραδοχή αυτή επιβάλλεται για μείωση υπολογιστικού κόστους και φαίνεται ότι είναι αρκετά ρεαλιστική, από δοκιμές που έχουν πραγματοποιηθεί.
- Χρησιμοποιείται σχήμα πρώτης τάξης. Με αυτόν τον τρόπο αποφεύγεται η αποθήκευση μη-διαγώνιων όρων που προκύπτουν από συνεισφορες κόμβων που δεν είναι πρώτοι γείτονες του P και ενισχύεται η διαγώνια κυριαρχία του αριστερού μέλους.

Σύμφωνα με τα παραπάνω, η εξισώσεις 3.57 λαμβάνουν την παρακάτω μορφή:

$$\frac{\partial \Phi^{PQ}_n}{\partial U^K_k} = \mathcal{A}^L_{nj} \frac{\partial U^L_j}{\partial U^K_k} + \mathcal{A}^R_{nj} \frac{\partial U^R_j}{\partial U^K_k} \tag{3.58}$$

3.5.3 Αριστερό μέλος των οριακών συνθηκών

Αντίστοιχα με τους όρους μεταφοράς γραμμικοποιούνται τα διανύσματα της ροής στα όρια εισόδου και εξόδου του χωρίου. Η διαφόριση της σχεσης 3.53 δίνει:

$$\frac{\partial \Phi^{P,out}_n}{\partial U^P_m} = A^+_{ni} + A^-_{nj} \frac{\partial U^{out}_j}{\partial U^P_m} \tag{3.59}$$

Η παραπάνω σχέση δίνει μόνο διαγώνιες συνεισφορές στον οριακό κόμβο P. Αντίστοιχα διαφορίζεται και η σχέση 3.52, δίνοντας διαγώνιες συνεισφορές στους οριακούς κόμβους του ορίου συμμετρίας καθώς και του στερεού ορίου.

3.6 Διαφορική γραφή των χρονικά μη-μόνιμων εξισώσεων Euler

Για την διατύπωση των χρονικά μη-μόνιμων εξισώσεων ροής Euler χρειάζεται μόνο να προστεθεί ο χρονικός όρος $\frac{\partial \vec{U}}{\partial \tau}$ στην εξίσωση 3.5. Έτσι προκύπτει η εξίσωση:

$$\frac{\partial \vec{U}}{\partial \tau} + \frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F_r}}{\partial x_r} = 0 \tag{3.60}$$

στην οποία υπάρχει και ο πραγματικός χρονικός όρος $\frac{\partial \vec{U}}{\partial \tau}$ και ο ψευδοχρονικός όρος $\frac{\partial \vec{U}}{\partial t}$. Για τη διακριτοποίηση του πραγματικού χρόνου χρησιμοποιείται σχήμα δεύτερης τάξης ώς προς το χρόνο και η χρονική παράγωγος γράφεται:

$$\left(\frac{\partial \vec{U}}{\partial \tau} \right)^k = \frac{3\vec{U}^k - 4\vec{U}^{k-1} - \vec{U}^{k-2}}{2\Delta\tau} \tag{3.61}$$

όπου $\Delta\tau$ είναι το πραγματικό χρονικό βήμα και k ο μετρητής των πραγματικών χρονικών βημάτων. Με k συμβολίζεται η τρέχουσα (προς επίλυση) χρονική στιγμή ενώ με $k-1$ και $k-2$ οι δύο προηγούμενες. Το πραγματικό χρονικό βήμα $\Delta\tau$ επιλέγεται από τον χρήστη. Ο ψευδοχρονικός όρος διατηρείται και χρησιμοποιείται κατά την επίλυση κάθε πραγματικού χρονικού βήματος για καλύτερη σύγκλιση των εξισώσεων.

Τέλος, με ολοκλήρωση του πραγματικού χρονικού όρου, για την τρέχουσα χρονική στιγμή, στον όγκο ελέγχου ενός τυχαίου κόμβου P προκύπτει:

$$\iiint_{V^P} \left(\frac{\partial \vec{U}}{\partial \tau} \right)^k dV = \left(\frac{\partial \vec{U}}{\partial \tau} \right)_P^k V^P \tag{3.62}$$

όπου με V^P συμβολίζεται ο όγκος του όγκου ελέγχου κατά τα γνωστά.

Κεφάλαιο 4

Προγραμματισμός Επιλύτη εξισώσεων Euler σε GPU

Σε αυτό το κεφάλαιο παρουσιάζεται ο επιλύτης των εξισώσεων Euler, όπως αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας, καθώς και η λογική πίσω από την ανάπτυξή του, έχοντας ως γνώμονα τον παράλληλο τρόπο επεξεργασίας δεδομένων των CUDA GPU και ως στόχο τη βέλτιστη αξιοποίηση όλων των πόρων που προσφέρουν οι GPU τελευταίας τεχνολογίας για την επίτευξη όσο το δυνατόν συντομότερων χρόνων εκτέλεσης, ενώ παράλληλα αναπαράγει πιστά τα αποτελέσματα του αντίστοιχου, πιστοποιημένου, κώδικα, ο οποίος εκτελείται από την CPU. Πιο συγκεκριμένα, ο κώδικας αναπτύχθηκε, πιστοποιήθηκε και αξιολογήθηκε στην παράλληλη υπολογιστική συστοιχία καρτών γραφικών της Μονάδας Παράλληλης Υπολογιστικής Ρευστοδυναμικής και Βελτιστοποίησης του Εργαστηρίου Θερμικών Στροβιλομηχανών του Εθνικού Μετσόβιου Πολυτεχνείου, η οποία αποτελείται από GPU TESLA M2050 από την εταιρεία NVIDIA. Η αναλυτική περιγραφή του παράλληλου συστήματος του ΕΘΣ γίνεται στο κεφάλαιο 5.

Ο επιλύτης που αναπτύχθηκε, όπως έχει αναφερθεί και στο κεφάλαιο 3, επιλύει τριδιάστατες, χρονικά μη-μόνιμες ροές συμπιεστού ρευστού, απουσία βαρυτικών δυνάμεων, στο καρτεσιανό σύστημα, διακριτοποιημένες με ένα σχήμα πεπερασμένων όγκων κεντροκομβικής προσέγγισης, για δομημένα πλέγματα με χρήση μεθόδου χρονοπροέλασης. Ο σχεδιασμός του επιλύτη βασίστηκε σε ήδη υπάρχοντα επιλύτη εξισώσεων Euler του ΕΘΣ. Ο επιλύτης «βάσης» αφορά την ίδια κατηγορία ροών και λειτουργεί με την ίδια μέθοδο διακριτοποίησης και επίλυσης, όμως χρησιμοποιεί μη-δομημένα πλέγματα.

Κατά την επίλυση της ροής με χρήση δομημένων πλεγμάτων παρουσιάζεται δόμηση, αντίστοιχη της δόμησης του πλέγματος, σε όλες τις μεταβλητές της ροής που χρειάζονται αποθήκευση κατά της εκτέλεση του προγράμματος. Έτσι, πέρα από τις προφανείς αλλαγές που έπρεπε να πραγματοποιηθούν, ώστε ο κώδικας να διαχειρίζεται δομημένα πλέγματα, έγιναν αλλαγές για την εκμετάλλευση μερικών δυνατοτήτων των GPU για τη βελτίωση της επιτάχυνσης της εφαρμογής, από τις οποίες μπορεί η εφαρμογή να επωφεληθεί λόγω της δόμησης του πλέγματος. Ένα τέτοιο παράδειγμα παρουσιάζεται στην παράγραφο 4.3.9.

4.1 Γενικά ζητήματα προγραμματισμού σε GPU

Πριν ξεκινήσει η ανάλυση του κώδικα του επιλύτη είναι χρήσιμο να αναφερθούν δύο γενικά προβλήματα με τα οποία θα έρθει αντιμέτωπος οποιοσδήποτε προγραμματιστής επιχειρήσει να προγραμματίσει σε GPU, δίχως να έχει σημασία αν το αντικείμενο της εφαρμογής είναι η υπολογιστική ρευστομηχανική, όπως εδώ, ή οτιδήποτε άλλο.

4.1.1 Καταμερισμός εργασίας σε kernels

Όπως αναφέρθηκε στην αρχή του κεφαλαίου, ώς στόχος ορίστηκε η βέλτιστη αξιοποίηση όλων των πόρων που προσφέρουν οι GPU για την επίτευξη όσο το δυνατόν συντομότερους χρόνους εκτέλεσης. Όμως, στην πράξη, αυτά τα δύο είναι πολλές φορές αντικρουόμενα. Όσο περισσότερα kernels χρησιμοποιηθούν, δηλαδή όσο περισσότερο κατακερματιστεί το αρχικό πρόβλημα, το κάθε kernel θα έχει μικρότερο φόρτο εργασίας άρα και μικρότερες απαιτήσεις σε registers και shared μνήμη. Για μικρότερες απαιτήσεις σε registers και shared μνήμη συνεπάγεται και καλύτερη παραλληλοποίηση του κώδικα [21] καθώς μπορούν να εκτελούνται παράλληλα περισσότερα blocks σε κάθε SM, όπως έχει αναλυθεί στην παράγραφο 2.2. Η παραπάνω διαδικασία, θεωρητικά οδηγεί σε καλύτερους χρόνους εκτέλεσης, καθώς οι δυνατότητες της GPU αξιοποιούνται βέλτιστα. Όμως, ο καταμερισμός σε kernels αναγκαστικά οδηγεί την εφαρμογή σε μεγαλύτερη χρήση της global μνήμης, καθώς είναι ο μόνος τρόπος να επικοινωνήσουν τα διάφορα kernels μεταξύ τους. Βέβαια, η προσπέλαση της global μνήμης είναι αρκετά χρονοβόρα. Έτσι, όσο το αρχικό πρόβλημα διαιρείται σε περισσότερα επιμέρους kernels από τη μία αυξάνεται η απόδοση της GPU, από την άλλη προστίθενται επιπλέον απαιτήσεις global μνήμης, τόσο σε ποσότητα όσο και σε προσπελάσεις. Από τα παραπάνω εύκολα φαίνεται πως η επιλογή του αριθμού των επιμέρους kernels αποτελεί στην ουσία ένα πρόβλημα βελτιστοποίησης μοναδικό για κάθε εφαρμογή, με αντικρουόμενους στόχους την χρήση λιγότερων registers και shared μνήμης από τη μία και τη χρήση λιγότερης global μνήμης από την άλλη.

4.1.2 Αποθήκευση πινάκων δύο η περισσότερων διαστάσεων.

Στις εφαρμογές της υπολογιστικής ρευστοδυναμικής, ανεξάρτητα από τον αριθμό διαστάσεων της κάθε εφαρμογής, παρουσιάζεται τακτικά η ανάγκη για αποθήκευση δεδομένων με διδιάστατη μορφή. Ένα παράδειγμα από την παρούσα εργασία είναι το διάνυσμα των μεταβλητών της ροής, όπως έχει οριστεί από την σχέση 3.3 και είναι

$$\overrightarrow{U} = \begin{bmatrix} \varrho \\ \varrho u \\ \varrho v \\ \varrho w \\ E \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix}$$

το οποίο πρέπει να αποθηκευτεί σε κάθε κόμβο του πλέγματος για την αριθμητική επίλυση των εξισώσεων. Οπότε προγραμματιστικά πρέπει να δεσμευτούν για την

αποθήκευση των μεταβλητών ροής $5*ns$ θέσεις μνήμης, όπου ns ο αριθμός των κόμβων του πλέγματος. Σε λογική σειριακής εκτέλεσης πράξεων, το βέλτιστο είναι να είναι αποθηκευμένες στη σειρά οι μεταβλητές κάθε κόμβου ώστε κατά την προσπέλαση να διαβάζονται σειριακά από συνεχόμενες θέσεις μνήμης. Η ταξινόμηση αυτή φαίνεται στο σχήμα 4.1. Σε γλώσσα C/C++ η παραπάνω κατανομή θα μεταφραζόταν σε έναν διδιάστατο πίνακα $U\,[ns]\,[5]$, ενώ σε γλώσσα FORTRAN η αντίστοιχη δήλωση θα ήταν $U\,(5, ns)$. Εναλλακτικά, θα μπορούσαν να αποθηκευτούν με μονοδιάστατη κατανομή «κορδόνι» (string) σε έναν πίνακα διάστασης $5 * ns$.

Στην GPU, όπου οι πράξεις και οι προσπελάσεις εκτελούνται παράλληλα, η λογική που πρέπει να ακολουθηθεί είναι λίγο διαφορετική, καθώς σε κάθε warp εκτελείται παράλληλα η ίδια εντολή μία φορά από κάθε thread, για διαφορετικές θέσεις μνήμης. Ακόμα, σύμφωνα με όσα ειπώθηκαν στην παράγραφο 2.3.5, η βέλτιστη προσπέλαση γίνεται όταν συνεχόμενα threads προσπελαύνουν συνεχόμενες θέσεις μνήμης και μάλιστα όταν όλα τα threads βρίσκονται μέσα στα προκαθορισμένα όρια ενός τμήματος της μνήμης, όπως φαίνεται στο σχήμα 2.7.

Έτσι, κατά την εκτέλεση ενός kernel στο οποίο θα ζητείται η προσπέλαση της τιμής U_1, τα threads 0 έως 31 όπου και ανήκουν στο πρώτο warp πρέπει να προσπελάσουν τις θέσεις μνήμης $U\,[0]$ έως $U\,[31]$ αντίστοιχα, θεωρώντας πως κάθε thread αντιπροσωπεύει και έναν κόμβο του πλέγματος. Αντίστοιχα για την τιμή U_2, από τα ίδια threads πρέπει να προσπελαστούν οι θέσεις μνήμης $U\,[index + 0]$ έως $U\,[index + 31]$, όπου $index$ σταθερή τιμή και πολλαπλάσιο του 32. Όμοια συνεχίζεται ο συλλογισμός και για τις υπόλοιπες τιμές του διανύσματος. Προφανώς για να επιτευχθούν οι παραπάνω απαιτήσεις δεν εξυπηρετούν οι διδιάστατες κατανομές όπως αναφέρθηκαν παραπάνω, αλλα χρησιμοποιείται μόνο η αποθήκευση με χρήση μονοδιάστατου πίνακα.

Ένα πρόβλημα που προκύπτει από τον παραπάνω τρόπο αποθήκευσης είναι ο υπολογισμός της τιμής του $index$. Θεωρώντας οι μεταβλητές θα αποθηκευτούν ως

- Στις θέσεις μνήμης $U[0]$ έως $U[31]$ αποθηκεύονται οι μεταβλητές U_1 των κόμβων με αύξοντα αριθμό 0 έως 31
- Θέσεις μνήμης $U[32]$ έως $U[63] \leftarrow U_2$ των κόμβων 0 έως 31
- Θέσεις μνήμης $U[64]$ έως $U[95] \leftarrow U_3$ των κόμβων 0 έως 31
- Θέσεις μνήμης $U[96]$ έως $U[127] \leftarrow U_4$ των κόμβων 0 έως 31
- Θέσεις μνήμης $U[128]$ έως $U[159] \leftarrow U_5$ των κόμβων 0 έως 31
- Θέσεις μνήμης $U[160]$ έως $U[191] \leftarrow U_1$ των κόμβων 32 έως 63
- Θέσεις μνήμης $U[192]$ έως $U[223] \leftarrow U_2$ των κόμβων 32 έως 63
- Θέσεις μνήμης $U[224]$ έως $U[255] \leftarrow U_3$ των κόμβων 32 έως 63
- Θέσεις μνήμης $U[256]$ έως $U[287] \leftarrow U_4$ των κόμβων 32 έως 63

  ⋮

εύκολα φαίνεται πως ο δείκτης της θέσης μνήμης στην οποία είναι αποθηκευμένη η μεταβλητή U_j του κόμβου n, με $j \in [1,5]$ και $0 \leq n < ns$, είναι:

$$i = (n\ div\ 32) * 32 * 5 + (j-1) * 32 + (n\ mod\ 32) \tag{4.1}$$

όπου ο τελεστής *div* δηλώνει το πηλίκο της ευκλείδειας διαίρεσης και ο τελεστής *mod* το υπόλοιπο της ευκλείδειας διαίρεσης.

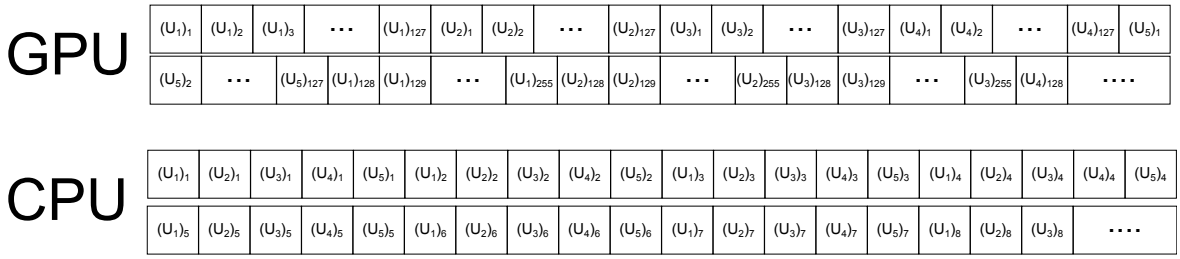
Το εύρος των συνεχόμενων αποθηκευμένων μεταβλητών «E», όπου στο παραπάνω παράδειγμα είναι $E = 32$, επιλέγεται αυθαίρετα από το χρήστη. Δοκιμές έχουν δείξει ότι βέλτιστη απόδοση επιτυγχάνεται όταν $E = nthreads$, όπου $nthreads$ είναι ο αριθμός των threads ανά block. Σε αυτήν την περίπτωση, η σχέση 4.1 μπορεί να ξαναγραφεί στη μορφή

$$i = blockId * nthreads * 5 + (j-1) * nthreads + it \tag{4.2}$$

όπου *blockId* είναι ο αύξοντας αριθμός του block στο οποίο ανήκει ο n-ιοστός κόμβος και *it* ο αύξοντας αριθμός του thread μέσα στο block, θεωρώντας μονοδιάστατη κατανομή threads και blocks. Η σχέση 4.3 μπορεί να μετατραπεί ώστε να δίνει την θέση μνήμης ενός τυχαίου στοιχείου $a_{j,n}$ ενός διδιάστατου πίνακα $A\ [J, ns]$, όπου $1 \leq j \leq J$ και είναι:

$$i = blockId * nthreads * J + (j-1) * nthreads + it \tag{4.3}$$

Στο σχήμα 4.1 φαίνεται ο τρόπος αποθήκευσης του διανύσματος των μεταβλητών της ροής του παραδείγματος για $E = 128$.



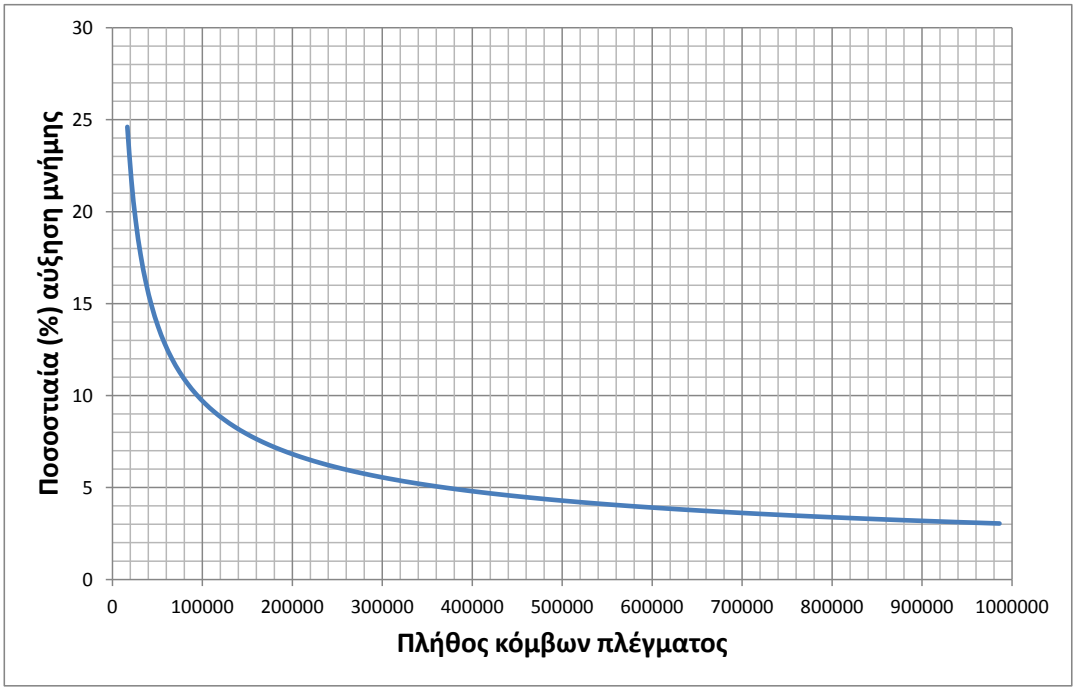
Σχήμα 4.1: Τρόπος αποθήκευση διανύσματος μεταβλητών ροής $(U_i)_j$ ($i = 1,2,3,4,5$ $j = 0,1,\ldots,Ns$) στην GPU και στην CPU για βέλτιστη προσπέλαση μνήμης.

Πίνακες περισσότερων διαστάσεων μπορούν να αποθηκευτούν με χρήση διδιάστατων πινάκων, σύμφωνα με τα παραπάνω. Έτσι ένας πίνακας διαστάσεων $[5,5,ns]$ θα μπορούσε να θεωρηθεί και να διαχειριστεί ως διδιάστατος πίνακας της μορφής $[25,ns]$.

Η παραπάνω μέθοδος, αν και επιταχύνει σε σημαντικό βαθμό την οποιαδήποτε εφαρμογή, έχει ένα μειονέκτημα το οποίο δεν έχει αναφερθεί παραπάνω. Έχει αυξημένες απαιτήσεις μνήμης. Με τον συμβατικό τρόπο αποθήκευσης, οι απαιτήσεις σε μνήμη είναι $J * ns$ θέσεις μνήμης. Με την μέθοδο αποθήκευσης που προτείνεται παραπάνω η μνήμη που πρέπει να δεσμευτεί για την ομαλή λειτουργία του αλγορίθμου

είναι $Ns = J * [(ns + E - 1)\, divE] * E$ θέσεις μνήμης. Από την σχέση αυτή φαίνεται πως όταν το ns είναι πολλαπλάσιο του εύρους E δεν υπάρχει άσκοπη δέσμευση μνήμης, ενώ όταν το ns δεν είναι πολλαπλάσιο του E δεσμεύονται περισσότερες θέσεις μνήμης από όσες χρειάζονται για την αποθήκευση του πίνακα και το πλεόνασμα είναι $J * (ns\ mod\ E)$ θέσεις μνήμης. Σε μικρά πλέγματα, η σπατάλη μνήμης φαίνεται πιο έντονα, ενώ όσο μεγαλώνει ο αριθμός των κόμβων του πλέγματος, τόσο το πλεόνασμα έχει μικρότερη σημασία, αφού η ποσοστιαία αύξηση της απαιτούμενης μνήμης μειώνεται. Παραδείγματα δέσμευσης πλεονάζουσας μνήμης παρουσιάζονται στα σχήματα 4.2 και 4.3.

Σχήμα 4.2: Μέγιστη ποσοστιαία αύξηση δέσμευσης μνήμης με μονοδιάστατη κατανομή threads ανά block και blocks στο grid. Για το παράδειγμα αυτό έχει επιλεχθεί $nthreads = 128$.

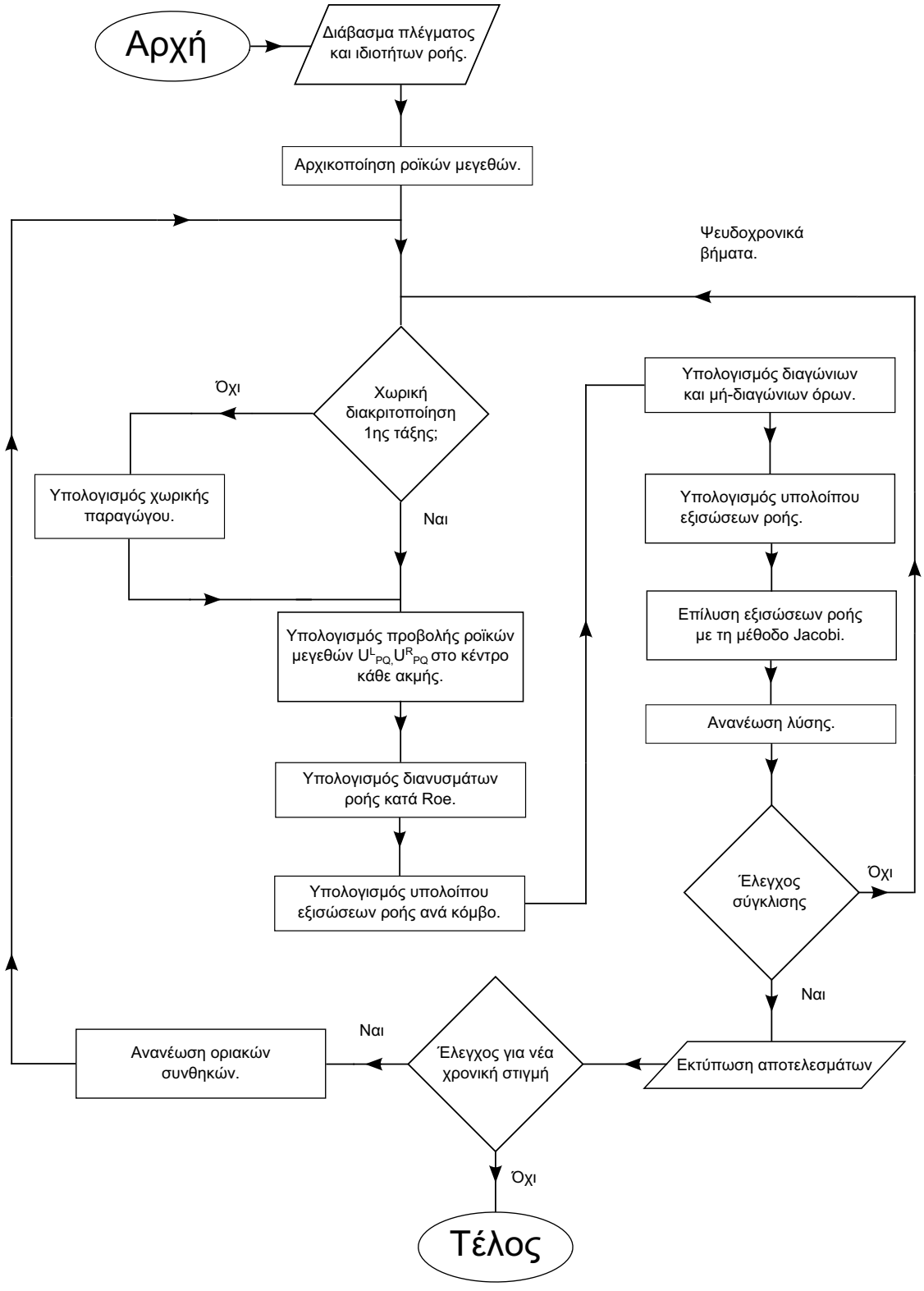


Σχήμα 4.3: Μέγιστη ποσοστιαία αύξηση δέσμευσης μνήμης με διδιάστατη κατανομή threads ανά block και blocks στο grid. Για το παράδειγμα αυτό έχει επιλεχθεί $nthreads = 16 \times 16$.

4.2 Γενική εποπτεία του κώδικα επίλυσης της ροής.

Έχοντας αναλύσει τα παραπάνω σημαντικά προβλήματα που σχετίζονται με τον προγραμματισμό σε GPU ανεξάρτητα το είδος της εφαρμογής, ακολουθεί μία σύντομη περιγραφή του κώδικα επίλυσης της ροής. Ακόμα, παρουσιάζεται το λογικό διάγραμμα του κώδικα στο σχήμα 4.4.

Όπως είναι προφανές, στο πρώτο τμήμα του κώδικα «διαβάζονται» το πλέγμα πάνω στο οποίο θα λυθούν οι εξισώσεις Euler, οι σταθερές που χαρακτηρίζουν τη ροή, όπως το γ και το R_g, και οι οριακές συνθήκες. Στη συνέχεια υπολογίζονται τα απαραίτητα, από το πλέγμα, στοιχεία, όπως είναι τα κάθετα διανύσματα (μέτρου ίσου με το εμβαδόν) της οριακής επιφάνειας μεταξύ των όγκων αναφοράς και ο όγκος κάθε όγκου ελέγχου. Το τελευταίο στάδιο της προεργασίας αυτής είναι η αρχικοποίηση της ροής, ώστε να ξεκινήσει ο κυρίως επιλύτης την επαναληπτική αριθμητική επίλυση του πεδίου της ροής για κάθε χρονική στιγμή.



Σχήμα 4.4: Λογικό διάγραμμα του αλγορίθμου επίλυσης της ροής.

Κάθε επανάληψη ξεκινάει με τον υπολογισμό, εάν χρειάζεται, της χωρικής παραγώγου του διανύσματος των μεταβλητών της ροής $\nabla\overrightarrow{U}$. Η παράγωγος χρησιμοποιείται για τον υπολογισμό των προεκβολών των μεταβλητών της ροής στο μέσο κάθε διανύσματος $\overrightarrow{PQ}$, για σχήματα χωρικής διακριτοποίησης δεύτερης τάξης ή χρήσης του περιοριστή των Van Leer-Van Albada, όπως αναλύθηκε στο προηγούμενο κεφάλαιο.

Ακολουθεί ο υπολογισμός των προβολών, με χρήση της αντίστοιχης σχέσης από την παράγραφο 3.3.4, $\overrightarrow{U}^L_{PQ}$ και $\overrightarrow{U}^R_{PQ}$ για κάθε ακμή του πλέγματος. Με χρήση των τιμών αυτών, υπολογίζονται τα ιακωβιανά μητρώα $\underline{A}_R$, $\underline{A}_L$ και $\left|\underline{\tilde{A}}_{PQ}\right|$, όπως ορίστηκαν στην παράγραφο 3.3.3. Γνωρίζοντας τις παραπάνω τιμές, υπολογίζεται το διάνυσμα της ροής $\overrightarrow{\Phi}_{PQ}$, καθώς και οι όροι $\mathcal{A}_L$ και $\mathcal{A}_R$ σε κάθε ακμή του πλέγματος, συναρτήσει του σχήματος του Roe. Σε κάθε κόμβο, με χρήση των παραπάνω τιμών κάθε γειτονικής ακμής του εκάστοτε κόμβου, υπολογίζονται το υπόλοιπο της ροής $\overrightarrow{R}$ καθώς και οι διαγώνιοι και μη-διαγώνιοι όροι, που θα χρησιμοποιηθούν στη συνέχεια για την επίλυση του συστήματος των εξισώσεων.

Μετά τον υπολογισμό τον διαγωνίων όρων κάθε κόμβου, προστίθενται σε αυτούς οι συνεισφορές του ψευδοχρονικού όρου για ενίσχυση της διαγώνιας κυριαρχίας και αξιοποίησης των ιδιοτήτων των υπερβολικών συστημάτων. Ο χρονικός όρος υπολογίζεται σύμφωνα με όσα ειπώθηκαν στην παράγραφο 3.3.5.

Έπειτα ακολουθεί η αριθμητική επίλυση των διακριτοποιημένων εξισώσεων γραμμένες στη μορφή που παρουσιάζονται στην εξίσωση 3.55 με χρήση της μεθόδου Jacobi. Πριν την επίλυση των εξισώσεων αυτών, προηγείται η αντιστροφή του πίνακα των διαγωνίων όρων, ο οποίος φέρει και την συνεισφορά του ψευδοχρονικού όρου, αφού εύκολα φαίνεται πως η επίλυση της σχέσης 3.55 ώς προς τον όρο $\Delta\overrightarrow{U}^P$, ο οποίος και είναι το ζητούμενο, δίνει:

$$\Delta\overrightarrow{U}^P = -\left[\frac{V^P}{\Delta t^P}I + \frac{\partial\overrightarrow{R}^P}{\partial\overrightarrow{U}^P}\right]^{-1}\left(\overrightarrow{R}^P + \sum_{Q\in K_P}\left(\frac{\partial\overrightarrow{R}^P}{\partial\overrightarrow{U}^Q}\right)\Delta\overrightarrow{U}^Q\right)$$

Η παραπάνω σχέση είναι πεπλεγμένη και για την επίλυσή της απαιτείται χρήση επαναληπτικής μεθόδου αριθμητικής ανάλυσης. Στον αλγόριθμο αυτό, χρησιμοποιείται η μέθοδος σταθερού σημείου, με τιμή εκκίνησης $\Delta\overrightarrow{U} = 0$. Για την μέθοδο σταθερού σημείου δεν χρησιμοποιείται κριτήριο σύγκλισης, αλλά προκαθορίζεται ο αριθμός των εσωτερικών επαναλήψεων από το χρήστη, με βάση την εμπειρία του.

Κλείνοντας την επαναληπτική διαδικασία, έχοντας υπολογίσει την τιμή του όρου $\Delta\overrightarrow{U}^P$ σε κάθε κόμβο, γίνεται η ανανέωση της λύσης σύμφωνα με τη σχέση:

$$\overrightarrow{U}^{n+1} = \overrightarrow{U}^n + \Delta\overrightarrow{U} \tag{4.4}$$

όπου ο δείκτης n αναφέρεται στην τρέχουσα ψευδοχρονική στιγμή και ο δείκτης $n+1$ στην αμέσως επόμενη.

Τέλος, γίνεται έλεγχος σύγκλισης του αλγορίθμου. Στην περίπτωση που η συνθήκη σύγκλισης δεν ικανοποιείται, ξαναξεκινά η επαναληπτική διαδικασία επεξεργαζόμενη το επόμενο ψευδοχρονικό βήμα. Εναλλακτικά, ο αλγόριθμος αποθηκεύει τα

αποτελέσματα της παραπάνω διαδικασίας και αφού ανανεώσει τις οριακές συνθήκες ξεκινάει την επίλυση του επόμενου πραγματικού χρονικού βήματος.

4.3 Ανάπτυξη επιλύτη σε προγραμματιστικό επίπεδο.

Έχοντας αναλύσει τον τρόπο λειτουργίας της GPU, τον μαθηματικό τρόπο επίλυσης των εξισώσεων Euler καθώς και έχοντας παρουσιάσει συνοπτικά τη λειτουργία του προγραμματισμένου επιλύτη Euler, παρακάτω ακολουθεί ο τρόπος με τον οποίο έγινε εκμετάλλευση των δυνατοτήτων της GPU και η λογική πίσω από κάθε απόφαση.

4.3.1 Αποθήκευση πινάκων.

Αρχικά, η ανάλυση του κώδικα θα ξεκινήσει με τα θέματα που θίχτηκαν στην αρχή του παρόντος κεφαλαίου. Ένα από αυτά είναι η αποθήκευση πινάκων διαστάσεων μεγαλύτερων της μίας. Σε ολόκληρο τον κώδικα, σχεδόν όλοι οι πίνακες που χρησιμοποιήθηκαν περιέχουν μεταβλητές και διανύσματα που για την αποθήκευσή τους αντιστοιχούν σε διδιάστατους, τριδιάστατους, ακόμα και τετραδιάστατους πίνακες. Όλοι οι παραπάνω πίνακες αποθηκεύτηκαν και έγινε η διαχείρισή τους με βάση όλα όσα αναφέρθηκαν στην παράγραφο 4.1.2. Έτσι, στον κώδικα συναντώνται διδιάστατοι πίνακες, όπως ο πίνακας στον οποίο αποθηκεύονται τα διανύσματα των μεταβλητών της ροής, τριδιάστατοι πίνακες, όπως ο πίνακας στον οποίο αποθηκεύονται τα διαγώνια μητρώα των συντελεστών κάθε κόμβου, και ακόμα και τετραδιάστατοι πίνακες, όπως ο πίνακας στον οποίο αποθηκεύονται τα μη-διαγώνια μητρώα των συντελεστών κάθε κόμβου. Όπως αναφέρθηκε και παραπάνω για πλέγμα ns κόμβων χρειάζονται N θέσεις μνήμης, για την διάσταση του πίνακα που αναφέρεται στην «ανά κόμβο» διάσταση και $N = J * [E - (ns \ mod \ E)]$. Έτσι, για τα παραδείγματα αυτά ισχύει:

- Ο πίνακας στον οποίο αποθηκεύονται τα διανύσματα μεταβλητών της ροής απαιτεί 5 θέσεις μνήμης ανά κόμβο, αφού η ροή είναι τριδιάστατη, και τελικά είναι διάστασης $5 * N$.

- Ο πίνακας στον οποίο αποθηκεύονται οι διαγώνιοι όροι απαιτεί ένα ιακωβιανό μητρώο ανά κόμβο, διάστασης $5 * 5 = 25$ θέσεων μνήμης, και τελικά είναι διάστασης $25 * N$.

- Ο πίνακας στον οποίο αποθηκεύονται οι μη-διαγώνιοι όροι απαιτεί ένα ιακωβιανό μητρώο για κάθε πρώτο γείτονα κάθε κόμβου, διάστασης $5 * 5 = 25$ θέσεων μνήμης και επειδή το πλέγμα είναι δομημένο είναι γνωστό πως οι πρώτοι γείτονες ανά κόμβο είναι 6 (για όλους τους εσωτερικούς κόμβους). Τελικά, η διάσταση του πίνακα αυτού προκύπτει $150 * N$.

4.3.2 Διαχωρισμός σε επιμέρους kernels.

Όσο αναφορά τον διαχωρισμό των kernels, πέραν των όσων αναφέρονται στο παρόν κεφάλαιο, υπάρχουν σημεία που απαιτείται ο διαχωρισμός σε περισσότερα του ενός kernel, καθώς για τη σωστή εκτέλεση των πράξεων χρειάζεται ολικός συγχρονισμός του προγραμματιστικού πλέγματος (grid) και όπως έχει αναφερθεί στην παράγραφο 2.2, ο διαχωρισμός του kernel είναι ο μόνος τρόπος να επιτευχθεί αυτό. Τέτοιο παράδειγμα είναι η υπορουτίνα όπου επιλύονται οι εξισώσεις ροής με τη μέθοδο Jacobi, όπου χρησιμοποιούνται ενδιάμεσοι πίνακες για αποθήκευση κατά την επαναληπτική επίλυση του τοπικά πεπλεγμένου συστήματος. Ο λόγος για τον οποίο είναι απαραίτητος ο συγχρονισμός του προγραμματιστικού πλέγματος στην υπορουτίνα αυτή εξηγείται αναλυτικά μαζί με την ανάλυση της υπορουτίνας στην παράγραφο 4.3.9. Στις υπόλοιπες περιπτώσεις, ο διαχωρισμός των kernels, όπου τελικά επιλέχθηκε, έγινε με κριτήρια συμβατότητας ενοποίησης διεργασιών. Όπου ήταν δυνατό το τελικό αποτέλεσμα να προκύψει με παραπάνω της μίας οργάνωσης των kernels, η τελική επιλογή έγινε μετά από δοκιμές με κριτήρια απόδοσης με στόχο τον μικρότερο δυνατό χρόνο εκτέλεσης του αλγορίθμου.

4.3.3 Χρήση constant μνήμης.

Όπως εξηγήθηκε στην παράγραφο 2.3.5, η constant μνήμη προορίζεται για αποθήκευση σταθερών δεδομένων και κυρίως όταν τα δεδομένα αυτά αναφέρονται σε ολόκληρο το τρέχον warp. Στην συγκεκριμένη εφαρμογή, την επίλυση των εξισώσεων Euler, εμφανίζονται σταθερές του προβλήματος οι οποίες παρουσιάζουν ακριβώς τα παραπάνω χαρακτηριστικά. Τέτοια σταθερά είναι ο ισεντροπικός εκθέτης ισεντροπικής μεταβολής γ, καθώς και οι συνηθισμένες εκφράσεις $\gamma - 1$, $\frac{\gamma}{\gamma-1}$, $\frac{1}{\gamma-1}$ και $\frac{\gamma-1}{\gamma}$, οι οποίες επειδή συναντώνται αρκετά συχνά είναι προτιμότερο να υπολογίζονται μία φορά και να αποθηκεύονται.

Άλλη μία πολλή σημαντική πληροφορία είναι η σχετική θέση κάθε γειτονικού κόμβου ενός τυχαίου κόμβου P. Είναι εκ τον προτέρων γνωστό πως ο επόμενος κατά i κόμβος είναι $is+1$ ενώ ο επόμενος κατά j κόμβος είναι $is+ni$ σε ένα τυπικό δομημένο πλέγμα. Στα παραπάνω ώς is συμβολίζεται ο αύξοντας αριθμός του κόμβου P, σε μονοδιάστατη κατανομή «κορδόνι» και ως ni η διάσταση του πλέγματος κατά i. Σε τυχόν χρωματισμό του πλέγματος οι εκφράσεις των σχετικών θέσεων των γειτόνων δεν είναι απαραίτητα τόσο απλές, αλλά η λογική παραμένει η ίδια. Ο τρόπος με τον οποίο θα προσπελαθεί αυτό το είδος πληροφορίας είναι ιδανικός για αποθήκευση στην constant μνήμη καθώς βρίσκεται σε απόλυτη συμφωνία με τις απαιτήσεις της για καλές επιδόσεις. Πέρα από τις σχετικές θέσεις των 6 γείτονων πρώτου βαθμού, αποθηκεύονται και οι σχετικές θέσεις των γειτόνων, όπως αυτοί ορίζονται στην παράγραφο 4.3.4, για τον υπολογισμό της χωρικής παραγώγου του διανύσματος των μεταβλητών της ροής.

4.3.4 Υπολογισμός χωρικής παραγώγου.

Όπως φαίνεται και στο λογικό διάγραμμα του αλγορίθμου, η πρώτη διεργασία που εκτελείται είναι ο υπολογισμός της πρώτης παραγώγου των μεταβλητών της ροής, η οποία χρειάζεται για την αύξηση της χωρικής διακριτοποίησης, σύμφωνα με τη σχέση 3.45. Για τον υπολογισμό της χωρικής παραγώγου κάθε κόμβου χρειάζονται πληροφορίες από όλα τα γειτονικά εξάεδρα του εκάστοτε κόμβου. Δηλαδή κάθε εσωτερικός κόμβος χρειάζεται πληροφορίες από όλους τους 26 κόμβους που τον περιβάλλουν, καθώς και από τον εαυτό του. Το μοτίβο προσπέλασης της μνήμης θυμίζει αρκετά όσα αναφέρθηκαν για τη σωστή χρήση της texture μνήμης και οι τελικές επιδόσεις υποστηρίζουν την προσπέλαση των γειτονικών θέσεων μνήμης κάθε κόμβου, με σκοπό το «διάβασμα» του διανύσματος των μη-συντηρητικών μεταβλητών, με χρήση της texture μνήμης, κατά τον υπολογισμό της παραγώγου σε κάθε κόμβο.

Ο υπολογισμός της χωρικής παραγώγου κάθε κόμβου αποτελεί ένα αυτόνομο kernel. Αφού υπολογιστεί η χωρική παράγωγος κάθε κόμβου, αποθηκεύεται στην global μνήμη, έτοιμη να χρησιμοποιηθεί για την προβολή των διανυσμάτων των μεταβλητών της ροής στα μέσα κάθε ακμής του πλέγματος.

4.3.5 Υπολογισμός σχήματος Roe

Έχοντας υπολογιστή η χωρική παράγωγος, υπολογίζονται οι προεκβολές των διανυσμάτων των μεταβλητών της ροής στο μέσο κάθε ακμής του πλέγματος. Βάση των προβολών αυτών υπολογίζονται και αποθηκεύονται τα ιακωβιανά μητρώα $\underline{A}_L$ και $\underline{A}_R$ καθώς και το $\left|\underline{\tilde{A}}_{PQ}\right|$. Από αυτά προκύπτουν και τα μητρώα $\mathcal{A}_L$ και $\mathcal{A}_R$. Έτσι, με χρήση αυτών, υπολογίζονται τελικά σε κάθε κόμβο το υπόλοιπο των εξισώσεων της ροής, οι διαγώνιοι και οι μή διαγώνιοι όροι. Για να γίνουν τα παραπάνω, προγραμματιστικά, εκτελούνται τα παρακάτων βήματα:

- Πρώτα χρωματίζεται το πλέγμα σε μία τριδιάστατη «σκακέρα», δηλαδή σε μαύρους και λευκούς κόμβους, διαδοχικά. Έτσι, όλοι οι άσπροι κόμβοι έχουν μαύρους γείτονες πρώτου βαθμού και, αντίστοιχα, οι μαύροι κόμβοι έχουν μόνο άσπρους γείτονες.

- Τη σάρωση όλων των ακμών του πλέγματος την αναλαμβάνουν εξ ολοκλήρου οι μαύροι κόμβοι, όπου ο κάθε μαύρος κόμβος θα εκτελέσει όλες τις πράξεις που σχετίζονται με τις ακμές του πλέγματος στις οποίες αυτός είναι άκρο. Όταν όλοι οι μαύροι κόμβοι ολοκληρώσουν την σάρωση ακμών, θα έχουν σαρωθεί όλες οι ακμές του πλέγματος, ενώ η κάθε ακμή θα έχει σαρωθεί μόνο μία φορά.

- Ανά ακμή, υπολογίζονται τα μητρώα $\underline{A}_L$, $\underline{A}_R$, $\left|\underline{\tilde{A}}_{PQ}\right|$,$\mathcal{A}_L$ και $\mathcal{A}_R$. Από τα μητρώα $\underline{A}_L$, $\underline{A}_R$ και $\left|\underline{\tilde{A}}_{PQ}\right|$ υπολογίζεται η ροή (flux) κάθε ακμής, σύμφωνα με την σχέση 3.40, ενώ τα μητρώα $\mathcal{A}_L$ και $\mathcal{A}_R$ αποτελούν τις συνεισφορές κάθε ακμής στους διαγώνιους και μή-διαγώνιους όρους.

- Σε κάθε κόμβο, η άθροιση των ροών κάθε γειτονικής ακμής δίνει το υπόλοιπο της ροής σε κάθε κόμβο του πλέγματος. Προγραμματιστικά έχει ιδιαίτερο

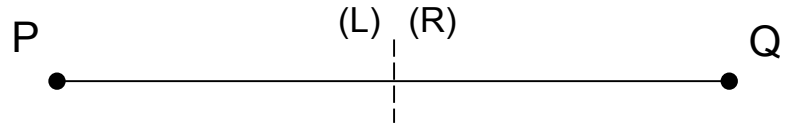
ενδιαφέρον το γεγονός ότι η παραπάνω διαδικασία επιτρέπει στους μαύρους κόμβους να αθροίζουν τοπικά (με χρήση τοπικών μεταβλητών) τις συνεισφορές κάθε ακμής και να αποθηκεύουν στην κύρια μνήμη μόνο το τελικό άθροισμα, δηλαδή επιβαρύνονται με «μία» μόνο προσπέλαση στην κύρια μνήμη, ενώ οι λευκοί κόμβοι αναγκαστικά δέχονται τις συνεισφορές ξεχωριστά, μία από κάθε γειτονικό μαύρο κόμβο, με αποτέλεσμα να χρειάζονται 6 προσπελάσεις στην κύρια μνήμη για τον σχηματισμό του υπολοίπου της ροής. Αν, μάλιστα, ως μία προσπέλαση ορισθεί μία προσπέλαση για κάθε εξίσωση ροής, δηλαδή χρειάζονται 5 πραγματικές προσπελάσεις ανά κόμβο για τους μαύρους κόμβους ή 30 προσπελάσεις ανά κόμβο για τους λευκούς.

- Για το σχηματισμό των διαγωνίων και μή-διαγωνίων όρων η διαδικασία που ακολουθείται είναι η εξής. Με τον τρόπο που έχουν χρωματιστεί οι κόμβοι και γίνεται η εκτέλεση της διαδικασίας, οι μαύροι κόμβοι αντιστοιχούν πάντα σε κόμβους P και οι λευκοί κόμβοι σε κόμβους Q, όπως αυτοί συμβολίζονται στο σχήμα 4.5. Με χρήση των δεικτών L και R σύμφωνα με τη σύμβαση του σχήματος, οι διαγώνιοι όροι των μαύρων κόμβων προκύπτουν από άθροιση του πίνακα $\mathcal{A}_L$ των γειτονικών ακμών και οι μη-διαγώνιοι όροι αποτελούνται από τους επιμέρους πίνακες $\mathcal{A}_R$, ένας όρος ανά κατεύθυνση ακμής. Αντίθετα, οι διαγώνιοι όροι των λευκών όρων προκύπτουν από άθροιση των όρων $-\mathcal{A}_R$ των γειτονικών τους ακμών και οι μη-διαγώνιοι όροι τους αποτελούνται από τους όρους $-\mathcal{A}_L$. Το αρνητικό πρόσημο προκύπτει επειδή οι πίνακες $\mathcal{A}_R$ και $\mathcal{A}_L$ έχουν υπολογιστεί βάση του διανύσματος $\overrightarrow{PQ}$ το οποίο αντιστοιχεί στον κόμβο P και όχι βάση του διανύσματος $\overrightarrow{QP}$, όπως θα έπρεπε για υπολογισμό των σχετικών τιμών του κόμβου Q. Η ίδια αλλαγή προσήμου πραγματοποιείται και στην αντίστοιχη άθροιση για τον υπολογισμό του υπολοίπου της ροής στους λευκούς κόμβους. Για την άθροιση των διαγωνίων όρων παρατηρείται το ίδιο φαινόμενο με την άθροιση των υπολοίπων της ροής, δηλαδή για τους μαύρους όρους απαιτείται «μία» προσπέλαση στην κύρια μνήμη ανά κόμβο, αφού αθροίζονται τοπικά, ενώ στους λευκούς κόμβους απαιτούνται 6 προσπελάσεις, όπου, εδώ, ως μία προσπέλαση εννοείται το σύνολο των 25 προσπελάσεων που απαιτείται για την αποθήκευση κάθε όρου, αφού κάθε διαγώνιος (και κάθε μή-διαγώνιος) όρος είναι ένας πίνακας διαστάσεων $5 * 5$.

Σε όσα αναφέρθηκαν παραπάνω είναι ιδιαίτερης σημασίας ο χρωματισμός του πλέγματος, καθώς μόνο μέσα από αυτόν είναι δυνατόν να εκτελεστεί η παραπάνω διαδικασία με σωστή διαχείριση μνήμης, κάτι το οποίο θα τονιστεί για ακόμα μία φορά πόσο σημαντικό είναι για την αξιοποίηση των δυνατοτήτων των GPU. Ακόμα, το σχήμα αυτό επιλέχθηκε καθώς από αυτό προκύπτει ο μικρότερος δυνατός αριθμός προσπελάσεων στην global μνήμη, δίχως να χρειάζεται να επανεκτελεστούν οι ίδιες πράξεις κάθε ακμής 2 φορές, μία για τον κόμβο P και μία για τον κόμβο Q.

Ολόκληρη η παραπάνω διαδικασία εκτελείται από ένα ιδιαίτερα απαιτητικό, από άποψη πόρων, kernel, το οποίο έχει καλύτερη απόδοση, περίπου κατά 15%, από από τις αντίστοιχες διεργασίες χωρισμένες σε περισσότερα kernels. Αυτό αναφέρεται

καθώς μέχρι πρότινος δεν ήταν δυνατή η σύμπτυξή του σε ένα μόνο kernel επειδή οι GPU προηγούμενων αρχιτεκτονικών (G80 και GT200) δεν πρόσφεραν αρκετούς πόρους για την εκτέλεση ενός τόσο απαιτητικού kernel.



Σχήμα 4.5: Μία τυχαία ακμή *PQ* του πλέγματος και οι ενδιάμεσες θέσεις L=Left και R=Right, όπου προεκβάλλονται τα μεγέθη της ροής.

4.3.6 Συνεισφορά του χρονικού όρου στο υπόλοιπο των εξισώσεων της ροής

Για να ολοκληρωθεί ο υπολογισμός του υπολοίπου των εξισώσεων της ροής πρέπει ακόμα να προστεθεί η συνεισφορά του πραγματικού χρονικού όρου, σύμφωνα με τις εξισώσεις 3.61 και 3.62. Για το λόγο αυτό, μετά το kernel που εκτελεί τους υπολογισμούς του σχήματος Roe, εκτελείται ένα kernel το οποίο προσθέτει τις συνεισφορές του χρονικού όρου στο υπόλοιπο των εξισώσεων της ροής. Το kernel αυτό αποτελεί ένα απλό, από προγραμματιστικής άποψης, kernel το οποίο δεν παρουσιάζει κάποιο ιδιαίτερο ενδιαφέρον, όπως και το kernel που αναλαμβάνει την ανανέωση της λύσης της ροής που ακολουθεί στη συνέχεια.

4.3.7 Υπολογισμός του υπολοίπου των εξισώσεων της ροής

Το επόμενο, κατά σειρά, kernel διεκπεραιώνει τους υπολογισμούς για την εύρεση του υπολοίπου των εξισώσεων της ροής. Όπως είναι γνωστό, η εύρεση του αθροίσματος των στοιχείων ενός πίνακα είναι μία έντονα σειριακή διαδικασία. Όμως, επειδή είναι και αρκετά συνηθισμένη, έχουν αναπτυχθεί διάφορες μέθοδοι ώστε να παραλληλοποιηθεί σημαντικά και να εκτελεστεί στην GPU αξιοποιώντας την παραλληλία που αυτή προσφέρει. Μία τέτοια μέθοδος έχει χρησιμοποιηθεί και εδώ και αναλύεται στη συνέχεια.

Αρχικά, το ζητούμενο δεν είναι η εύρεση του αθροίσματος των στοιχείων ενός πίνακα, αλλά 5, καθώς το πρόβλημα που επιλύεται αποτελείται από 5 εξισώσεις. Το πρόβλημα της άθροισης των στοιχείων του καθενός από τους 5 πίνακες, έχει διαχωριστεί σε 2 υποπροβλήματα, ένα το οποίο εκτελείται στην GPU και ένα το οποίο εκτελείται στην CPU. Πιο συγκεκριμένα, η GPU υπολογίζει το άθροισμα του υπολοίπου των εξισώσεων της ροής σε κάθε block και τα αποθηκεύει σε έναν πίνακα διαστάσεως nblock, όπου nblock είναι ο αριθμός των block από τα οποία αποτελείται το grid. Ο πίνακας αυτός, στη συνέχεια, αντιγράφεται στην CPU και εκεί αθροίζεται σειριακά.

Για την εκτέλεση των παραπάνω δημιουργήθηκαν 5 CUDA streams, ένα για τον υπολογισμό του υπολοίπου κάθε εξίσωσης, όπου το καθένα αποτελείται από την

εύρεση του υπολοίπου της ροής ανά block και, στη συνέχεια την αντιγραφή των αποτελεσμάτων στην CPU. Μόλις ολοκληρωθεί αυτή η διαδικασία, η CPU αναλαμβάνει την σειριακή άθροιση των υπολοίπων ανά block.

Πέρα από την χρήση των CUDA streams, ενδιαφέρον παρουσιάζει και ο τρόπος με τον οποίο κάθε block βρίσκει το άθροισμα των υπολοίπων των εξισώσεων της ροής. Για τον υπολογισμό του υπολοίπου ανά block αναπτύχθηκε ένα kernel μέσα στο οποίο πραγματοποιούνται τα παρακάτω:

- Αρχικά, κάθε thread του block αποθηκεύει στη shared μνήμη, η οποία υπενθυμίζεται πως είναι ορατή από όλα τα threads ενός block, την τιμή του υπολοίπου που αντιστοιχεί στον κόμβο του.

- Συγχρονίζονται όλα τα threads του block, αφού για να προσπελαθεί με ασφάλεια μία θέση μνήμης πρέπει να είναι σίγουρο πως δεν χρησιμοποιείται από άλλο thread.

- Τα threads που ανήκουν στο πρώτο μισό του block προσθέτουν στις τιμές που τους αντιστοιχούν τις τιμές των αντίστοιχων threads του δεύτερου μισού του block και το block ξανασυγχρονίζεται.

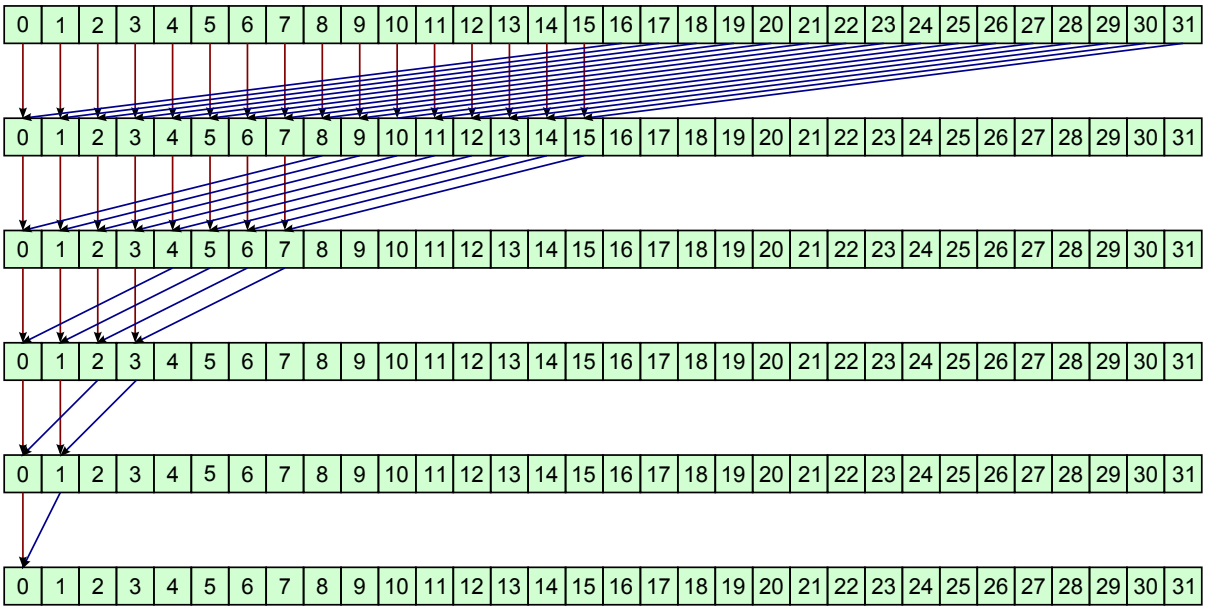
- Όσα threads ανήκουν στο πρώτο μισό του block ξαναδιαχωρίζονται στη μέση και τα πρώτα προσθέτουν στα υπόλοιπα που τους αντιστοιχούν τα υπόλοιπα των δεύτερων μισών και το block ξανασυγχρονίζεται.

- Η διαδικασία αυτή επαναλαμβάνεται μέχρι όλα τα υπόλοιπα ροής που αντιστοιχούν στο εκάστοτε block να έχουν αθροιστεί στο thread 0.

Η παραπάνω διαδικασία φαίνεται στο παράδειγμα του σχήματος 4.6 όπου έχει θεωρηθεί block με 32 threads. Αν και η τεχνική που παρουσιάστηκε έχει πολύ καλές επιδόσεις, συνοδεύεται από ένα σημαντικό μειονέκτημα. Όπως εύκολα φαίνεται, για να λειτουργήσει αποδοτικά πρέπει τα threads ανά block να είναι δύναμη του 2.

4.3.8 Συνεισφορά ψευδοχρονικού και χρονικού όρου και προεργασία επίλυσης σχήματος Jacobi

Το επόμενο, κατά σειρά εκτέλεσης kernel προσθέτει στο διαγώνιο όρο κάθε κόμβου τη συνεισφορά του ψευδοχρονικού όρου και του πραγματικού χρονικού όρου, αντιστρέφει τον πίνακα $\left[\frac{V^P}{\Delta t^P} I + \frac{\partial \vec{R}^P}{\partial \vec{U}^P}\right]$, για λόγους που έχουν αναφερθεί στην παράγραφο 4.2, και ξεκινά την επαναληπτική διαδικασία επίλυσης της εξίσωσης 3.55, θέτοντας ως αρχική τιμή $\Delta\vec{U} = 0$. Το παρόν kernel δεν έχει κάτι ουσιαστικό να επιδείξει από προγραμματιστική σκοπιά, πέρα από τη σύμπτυξη διαφορετικών εργασιών σε ένα kernel, για τη βελτίωση του τελικού αποτελέσματος.



Σχήμα 4.6: Τρόπος άθροισης στοιχείων ενός διανύσματος, ανά block.

4.3.9 Επίλυση σχήματος Jacobi

Στη συνέχεια ακολουθεί η επαναληπτική επίλυση της εξίσωσης 3.55. Σε κάθε υπολογιστικό βήμα επίλυσης, το κάθε thread θα χρειαστεί πληροφορίες από τον πίνακα στον οποίο είναι αποθηκευμένοι οι μη-διαγώνιοι όροι και από τον πίνακα που είναι αποθηκευμένος ο αντιστραμμένος πίνακας των διαγωνίων όρων. Για την προσπέλαση των μη-διαγωνίων όρων και του πίνακα των αντιστραμμένων διαγώνιων όρων το μόνο σημαντικό στοιχείο είναι η προσπέλασή τους, από την global μνήμη, να γίνει σύμφωνα με όσα έχουν αναφερθεί στην παράγραφο 2.3.5, για βέλτιστη ταχύτητα προσπέλασης. Όμως, χρειάζονται ακόμα οι τιμές του $\Delta\overrightarrow{U}$ των γειτόνων κάθε κόμβου. Το «διάβασμα» των τιμών αυτών γίνεται μέσω της texture μνήμης, καθώς η κατανομή τους είναι η τριδιάστατη παραλλαγή της κατανομής η οποία παρουσιάστηκε ως βέλτιστη για χρήση σε συνδιασμό με την texture μνήμη στην παράγραφο 2.3.5. Όμως, πέρα από την χρήση της texture μνήμης, το συγκεκριμένο kernel επωφελήθηκε και από την shared μνήμη. Όπως γίνεται η σάρωση των γειτόνων, εύκολα φαίνεται πως το κάθε στοιχείο θα προσπελαθεί «μία» φορά ανά γείτονα, δηλαδή 6 φορές. Και σε αυτό το σημείο, η κάθε «μία» προσπέλαση δεν αντιστοιχεί σε μία πραγματική προσπέλαση αλλά σε 5, μία για κάθε εξίσωση. Βέβαια, αρκετοί από του γείτονες του προαναφερθέντος τυχαίου κόμβου πιθανόν να βρίσκονται στο ίδιο block με τον εξεταζόμενο κόμβο. Οπότε, η τελική διαδικασία που ακολουθείται είναι η εξής :

- Στην αρχή του kernel, ο κάθε κόμβος αποθηκεύει το διάνυσμα $\Delta\overrightarrow{U}$ ενός πρώτο του γείτονα στη shared μνήμη. Ο γείτονας που επιλέγεται είναι ο γείτονας με αύξοντα αριθμό $is_{nei} = is \mp \frac{N}{2}$, όπου is είναι ο αύξοντας αριθμός του τρέχοντος κόμβου, N το μέγεθος του προγραμματιστικού grid. Η πρόσθεση γίνεται όταν ο τρέχον κόμβος είναι μαύρος, ενώ η αφαίρεση όταν ο κόμβος είναι λευκός

(4.3.11). Με τη διαδικασία αυτή ο κάθε κόμβος αποθηκεύει το διάνυσμα $\Delta\overrightarrow{U}$ του $i+1$ ή του $i-1$ γειτονά του στην shared μνήμη. Το αν ο γείτονας είναι ο $i+1$ ή ο $i-1$ εξαρτάται από το αν το τρέχον κόμβος είναι μαύρος ή άσπρος και από το j του κόμβου.

- Ακολουθεί συγχρονισμός του block, καθώς απαιτείται για σωστή λειτουργία του kernel.

- Στη συνέχεια, όποτε ένας κόμβος χρειάζεται ένα γειτονικό διάνυσμα $\Delta\overrightarrow{U}$, πριν απευθυνθεί στην global μνήμη, ελέγχει αν ο γειτονικός κόμβος βρίσκεται στο ίδιο block με αυτόν. Σε περίπτωση που όντως βρίσκεται, το «διάβασμα» του διανύσματος πραγματοποιείται από τη shared μνήμη, ενώ σε αντίθετη περίπτωση η προσπέλαση γίνεται κανονικά στη global μνήμη.

Προφανώς, ο λόγος που γίνονται τα παραπάνω είναι η μείωση των προσπελάσεων στην global μνήμη και η αντικατάστασή τους από προσπελάσεις στη shared μνήμη. Όπως έχει αναφερθεί και στην παράγραφο 2.3.5, μία τέτοια «ανταλλαγή» είναι επικερδής επειδή ο χρόνος που χρειάζεται για την προσπέλαση της shared μνήμης είναι αρκετά μικρότερος από τον αντίστοιχο για προσπέλαση στη global μνήμη.

Άλλο ένα σημείο που παρουσιάζει προγραμματιστικό ενδιαφέρον είναι η αποθήκευση των διανυσμάτων $\Delta\overrightarrow{U}$. Το kernel επίλυσης της μεθόδου Jacobi ξεκινάει με διάβασμα του διανύσματος $\Delta\overrightarrow{U}$ και αφού τελειώσει την επεξεργασία αποθηκεύθει το καινούργιο διάνυσμα $\Delta\overrightarrow{U}$. Όμως, το διάβασμα αντιστοιχεί σε γειτονικές θέσεις μνήμης, ενώ η αποθήκευση στη θέση μνήμης αντιστοιχεί στον κόμβο του ενεργού thread. Για την σωστή εκτέλεση των πράξεων, θα έπρεπε πριν την αποθήκευση των καινούργιων διανυσμάτων $\Delta\overrightarrow{U}$ να προηγηθεί ολικός συγχρονισμός στο προγραμματιστικό grid. Βέβαια, υπό την παρούσα τεχνολογία των GPU, κάτι τέτοιο δεν είναι δυνατό, οπότε, για την άρση του προβλήματος αυτού χρησιμοποιούνται 2 πίνακες, ένας από τον οποίο θα διαβαστούν τα διανύσματα στην αρχή του kernel και ένας στον οποίο θα αποθηκευτούν τα διανύσματα μετά την επεξεργασία τους. Χρησιμοποιώντας 2 πίνακες, έναν από τον οποίο διαβάζονται τα δεδομένα στην αρχή του kernel και έναν στον οποίο αποθηκεύονται τα δεδομένα στο τέλος του kernel, λύνεται το πρόβλημα του απαιτούμενου συγχρονισμού, αλλά έχοντας ένα kernel που διαβάζει από έναν πίνακα A και γράφει σε έναν πίνακα B φαίνεται πως είναι απαραίτητη η αντιγραφή του πίνακα B στον πίνακα A πριν την εκκίνηση του επόμενου υπολογιστικού βήματος. Όμως, επειδή και η αντιγραφή αυτή είναι χρονοβόρα, στο πλαίσιο της επίλυσης του σχήματος Jacobi αναπτύχθηκαν 2 kernels, των οποίων η μόνη διαφορά είναι οι πίνακες από τους οποίους διαβάζουν και οι πίνακες στους οποίους αποθηκεύουν. Έτσι, θεωρώντας πως το πρώτο kernel διαβάζει τα διανύσματα $\Delta\overrightarrow{U}$ από έναν πίνακα A και τα αποθηκεύει σε έναν πίνακα B, τότε το δεύτερο kernel εκτελεί ακριβώς τις ίδιες ενέργειες με το πρώτο, με μόνη διαφορά ότι διαβάζει τα διανύσματα $\Delta\overrightarrow{U}$ από τον πίνακα B και τα αποθηκεύει στον πίνακα A. Οι εκτελέσεις των δύο kernels διαδέχονται η μία την άλλη και έτσι φαίνεται πως πλέον δεν υπάρχει ανάγκη για αντιγραφή του πίνακα B στον A, ή το αντίθετο.

4.3.10 Ανανέωση λύσης

Το τελευταίο στάδιο του επιλύτη είναι η ανανέωση της λύσης του προβλήματος της ροής μέσω της σχέσης 4.4. Προγραμματιστικά, το kernel που ανανεώνει τη λύση της ροής δεν παρουσιάζει κάποιο ιδιαίτερο ενδιαφέρον. Στο kernel αυτό, το κάθε tread αναλαμβάνει το διάβασμα του διανύσματος $\Delta\overrightarrow{U}$ ενός κόμβου και την άθροισή του με το διάνυσμα των μεταβλητών της ροής του κόμβου αυτού. Μετά την ανανέωση της λύσης γίνεται ο έλεγχος της σύγκλισης και είτε ξεκινά το επόμενο ψευδοχρονικό βήμα, είτε τελειώνει η επίλυση του τρέχοντος χρονικού βήματος, εκτυπώνοντας τα αποτελέσματα.

4.3.11 Χρωματισμός και διαχείριση πλέγματος

Στην παράγραφο 4.3.5 αναφέρθηκε η χρήση χρωματισμένου πλέγματος, χωρίς όμως να δοθεί ιδιαίτερη σημασία στο πως γίνεται και πως χρησιμοποιείται ο χρωματισμός αυτός. Για να εκτελεστεί σωστά η διαδικασία που περιγράφεται στην παράγραφο 4.3.5 πρέπει οι κόμβοι που είναι «ενεργοί» σε κάθε χρονική στιγμή να μην είναι πρώτοι γείτονες μεταξύ τους. Η προφανής λύση, δεδομένου ότι το πλέγμα είναι δομημένο, είναι οι κόμβοι του πλέγματος να είναι «ένας ενεργός - ένας ανενεργός» διαδοχικά, προς κάθε κατεύθυνση i,j,k του πλέγματος. Το τελικό αποτέλεσμα είναι μία αλληλουχία ενεργών και ανενεργών κόμβων, ή αλλιώς μια τριδιάστατη «σκακέρα» όπως αναφέρθηκε κατά την πρώτη αναφορά στο χρωματισμό του πλέγματος στην παράγραφο 4.3.5, με μαύρους και λευκούς κόμβους, όπου οι μαύροι έστω αναπαριστούν τους ενεργούς κόμβους και οι λευκοί τους ανενεργούς. Όμως, για βέλτιστη προσπέλαση στη global μνήμη, πρέπει συνεχόμενα threads να προσπελαύνουν συνεχόμενες θέσεις μνήμης. Θεωρώντας συμβατικό τρόπο αποθήκευσης, για παράδειγμα αποθηκεύοντας την κατά-i συνιστώσα κάθε κόμβου στην αντίστοιχη θέση μνήμης του αύξοντα αριθμού του, δηλαδή ο κόμβος 0 θα ήταν αποθηκευμένος στην θέση του αντίστοιχου πίνακα 0, ο κόμβος 1 στην θέση 1 κλπ, το τελικό αποτέλεσμα θα ήταν βέβαια σωστό, αλλά λόγω της μη-βέλτιστης προσπέλασης της μνήμης, δεν θα πρόκυπτε η βέλτιστη απόδοση από άποψη χρόνου εκτέλεσης. Για τη βέλτιστη προσπέλαση της μνήμης, ο αύξοντας αριθμός κάθε κόμβου παύει πλέον να είναι $i + j * ni + k * ni * nj$, όπως είναι η συνηθισμένη του έκφραση, αλλά οι κόμβοι ταξινομούνται με τέτοιο τρόπο ώστε να είναι ομαδοποιημένοι όλοι οι μαύροι κόμβοι μαζί και όλοι οι λευκοί κόμβοι μαζί. Πρακτικά, αυτό σημαίνει πως αποθηκεύονται όλοι οι μαύροι κόμβοι σε συνεχόμενες θέσεις μνήμης, στο πρώτο μισό του αντίστοιχου πίνακα, και όλοι οι λευκοί κόμβοι στο δεύτερο μισό του ίδιου πίνακα. Έτσι, όταν ένα warp θα χρειαστεί οποιαδήποτε πληροφορία σχετικά με τους μαύρους κόμβους, η πληροφορία αυτή θα βρίσκεται σε 32 συνεχόμενες θέσεις μνήμης. Αντίστοιχα, όταν χρειάζεται πληροφορία σχετική με τους λευκούς κόμβους, και αυτή βρίσκεται σε 32 συνεχόμενες θέσεις μνήμης.

Λόγω της διαχείρισης που απαιτεί η καινούργια κατανομή των κόμβων στη μνήμη της GPU, πρέπει να είναι γνωστά σε κάθε κόμβο, όχι μόνο ο αύξοντας αριθμός που του αντιστοιχεί, αλλά και οι συνιστώσες του κόμβου j και k. Το γιατί επιβάλλεται αυτό θα γίνει κατανοητό στη συνέχεια. Ο πιο εύκολος τρόπος για να είναι γνωστές οι

συνιστώσες j και k θα ήταν η τριδιάστατη κατανομή threads και blocks, όπου άμεσα από την αρίθμηση τους θα προέκυπταν και οι ζητούμενες συνιστώσες. Βέβαια, μία τέτοια επιλογή θα αύξανε πάρα πολύ τις απαιτήσεις σε μνήμη σε περίπτωση μή «ιδανικού» πλέγματος. Είναι χρήσιμο να υπενθυμιστεί πως σε μία μονοδιάστατη κατανομή blocks μέσα στο grid, σε ένα πλέγμα με ns κόμβους, θα χρειαστούν $nblocks$ blocks, όπου

$$nblocks = \frac{ns + nthreads - 1}{nthreads}$$

όπου $nthreads$ είναι ο αριθμός των threads ανά block. Ακόμα υπενθυμίζεται πως η μνήμη που απαιτείται για την αποθήκευση ενός μη-μονοδιάστατου πίνακα είναι $a * nblocks * nthreads$ και όχι $a * ns$, όπου a η δεύτερη διάσταση του πίνακα, όπως έχει αναλυθεί στην παράγραφο 4.1.2. Αντίστοιχα, σε μία τριδιάστατη κατανομή blocks, ο αριθμός των blocks που χρειάζεται είναι

$$nblock = nblock_i * nblock_j * nblock_k =$$

$$= \frac{ni + nthreads_i - 1}{nthreads_i} * \frac{nj + nthreads_j - 1}{nthreads_j} * \frac{nk + nthreads_k - 1}{nthreads_k}$$

με ni,nj και nk το πλήθος των κόμβων ανά ανάξονα και οι δείκτες i,j,k συμβολίζουν την αντίστοιχη ποσότητα ανά άξονα. Όλες οι παραπάνω διαιρέσεις είναι πράξεις μεταξύ ακεραίων και αντιστοιχούν σε ευκλείδιες διαιρέσεις. Οπότε και γίνεται πλέον φανερό πως σε ένα πλέγμα μονοδιάστατης κατανομής, επαρκούς μεγέθους, η δέσμευση πλεονάζουσας μνήμης που ίσως να χρειάζεται δεν είναι σημαντική, ενώ αντίστοιχα στην τρισδιάστατη κατανομή, η χείριστη περίπτωση έχει πάντα μεγάλη δέσμευση πλεονάζουσας μνήμης.

Πέρα από θέμα μνήμης, η επιλογή ανάμεσα σε μονοδιάστατη και τριδιάστατη κατανομή συνδέεται και με την τελική απόδοση κάθε κατανομής. Στην τριδιάστατη κατανομή, οι πράξεις που χρειάζονται για την εύρεση των j και k κάθε κόμβου είναι αμελητέες. Από την άλλη πλευρά, για εύρεση των ίδιων συντεταγμένων με χρήση μονοδιάστατης κατανομής χρειάζονται αρκετές διαιρέσεις και πράξεις υπολοίπου (mod) και όπως είναι γνωστό οι διαιρέσεις είναι αρκετά χρονοβόρες στον Η/Υ. Όμως, πέρα από το πλήθος των πράξεων, όταν η διαχείριση του πλέγματος γίνεται με μονοδιάστατη κατανομή είναι πολύ πιθανό κόμβοι διαφορετικών j ή k να εκτελούνται ταυτόχρονα στο ίδιο warp. Κάτι τέτοιο μειώνει την απόδοση, καθώς, όπως θα δειχθεί παρακάτω, κόμβοι διαφορετικών j ή k έχουν διαφορετική σχετική θέση γειτόνων, με αποτέλεσμα να αναγκάζεται το τρέχον warp να χωρίζεται και έτσι να μειώνεται η παραλληλία της εφαρμογής.

Όπως και στα περισσότερα προβλήματα τέτοιου είδους, ως βέλτιστη επιλογή φαίνεται να είναι μία ενδιάμεση λύση. Τελικά, επιλέχθηκε μία διδιάστατη κατανομή η οποία χειρίζεται μονοδιάστατα κάθε i-j επίπεδο και ως δεύτερη διάσταση χρησιμοποιεί την k διεύθυνση. Με χρήση αυτής της κατανομής γίνεται εύκολα γνωστή η συνιστώσα k κάθε κόμβου και το κάθε kernel επιβαρύνεται μόνο με τον υπολογισμό της j συνιστωσας. Αυτό είναι ιδιαίτερα βολικό όταν η διεύθυνση της ροής συμβαδίζει

με τη διεύθυνση k. Ακόμα, οι απαιτήσεις για δέσμευση πλεονάζουσας μνήμης έχουν μειωθεί σημαντικά συγκριτικά με την τριδιάστατη κατανομή, αφού πλέον δεσμεύεται πλεονάζουσα μνήμη μόνο στο τέλος κάθε «μονοδιάστατου»-προγραμματιστικά επιπέδου i-j. Κατά τη δεύτερη διεύθυνση k της κατανομής, δεν δεσμεύεται καθόλου πλεονάζουσα μνήμη καθώς κάθε block είναι ορισμένο έτσι ώστε να αντιστοιχεί πάντα σε ένα και μόνο επίπεδο i-j. Με χρήση αυτής της κατανομής είναι ακόμα σίγουρο πως τα threads κάθε warp ανήκουν πάντα στο ίδιο επίπεδο i-j, δηλαδή έχουν σταθερό k. Κάτι αντίστοιχο δεν μπορεί να γίνει για τον έλεγχο σχετικά με ίδια συνιστώσα j δίχως να χρησιμοποιηθεί κανονική τριδιάστατη κατανομή. Το μόνο που μπορεί να γίνει από την μεριά του χρήστη για την αποφυγή του διαχωρισμού του warp είναι η επιλογή κατάλληλου πλέγματος, όταν αυτό είναι δυνατό, ώστε το ni να είναι πολλαπλάσιο του 32 οπότε και κάθε thread του ίδιου warp να έχει την ίδια συνιστώσα j.

Ο λόγος που χρειάζονται οι συνιστώσες j και k κάθε κόμβου είναι ότι η σχετική θέση των γειτόνων κάθε κόμβου εξαρτάται από αυτές. Σε ένα πλέγμα όπου το ni είναι ζυγός αριθμός, η θέση κάθε γειτονικού κόμβου isn ενός τυχαίου κόμβου (i,j,k) με αυξοντα αριθμό is εξαρτάται από 3 παράγωντες.

- Αν πρόκειται για μαύρο ή λευκό κόμβο.
- Αν το j του κόμβου είναι μονό ή ζυγό.
- Αν το k του κόμβου είναι μονό ή ζυγό.

Για έναν τυχαίο μαύρο κόμβο ισχύει:

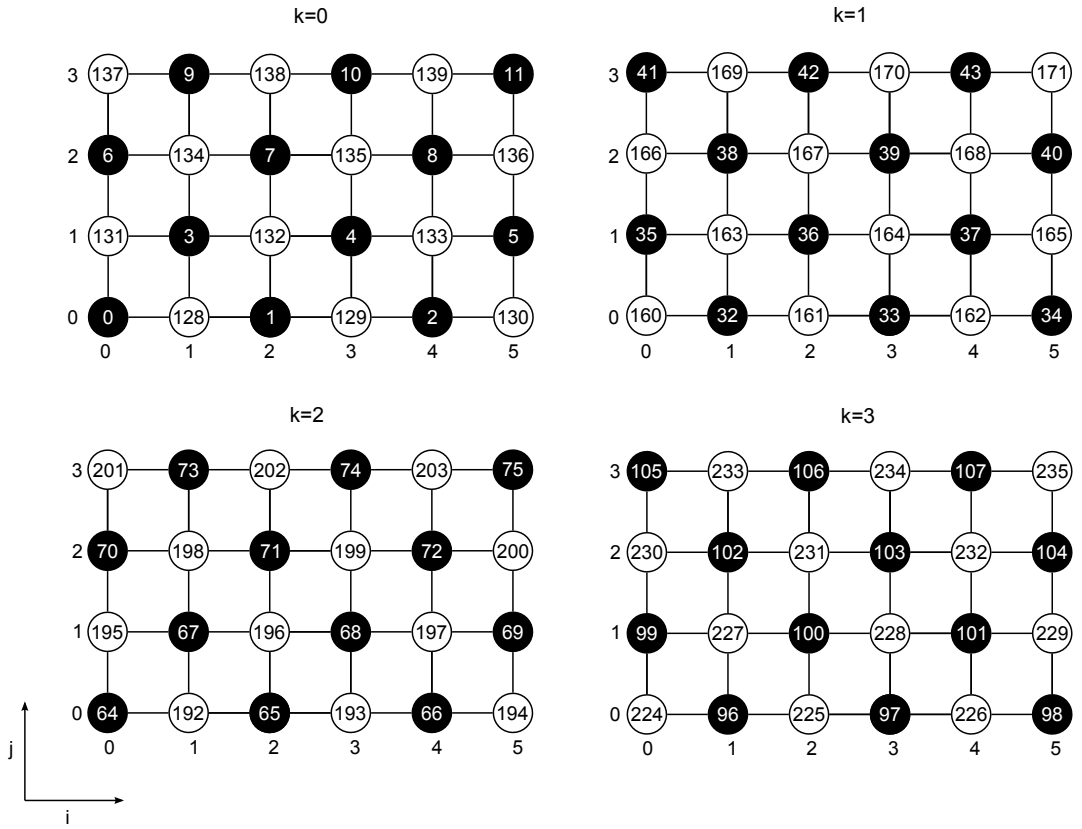
- Όταν το k είναι ζυγό:
  - Για j ζυγό:
    - $isn_{i+1} = is + X$
    - $isn_{i-1} = is - 1 + X$
  - Για j μονό:
    - $isn_{i+1} = is + 1 + X$
    - $isn_{i-1} = is + X$
- Όταν το k είναι μονό:
  - Για j ζυγό:
    - $isn_{i+1} = is + 1 + X$
    - $isn_{i-1} = is + X$
  - Για j μονό:
    - $isn_{i+1} = is + X$
    - $isn_{i-1} = is - 1 + X$
- Ανεξαρτήτως j,k:

- $isn_{j+1} = is + \frac{ni}{2} + X$
- $isn_{j-1} = is - \frac{ni}{2} + X$
- $isn_{k+1} = is + \frac{ni}{2} * nj + X$
- $isn_{k-1} = is - \frac{ni}{2} * nj + X$

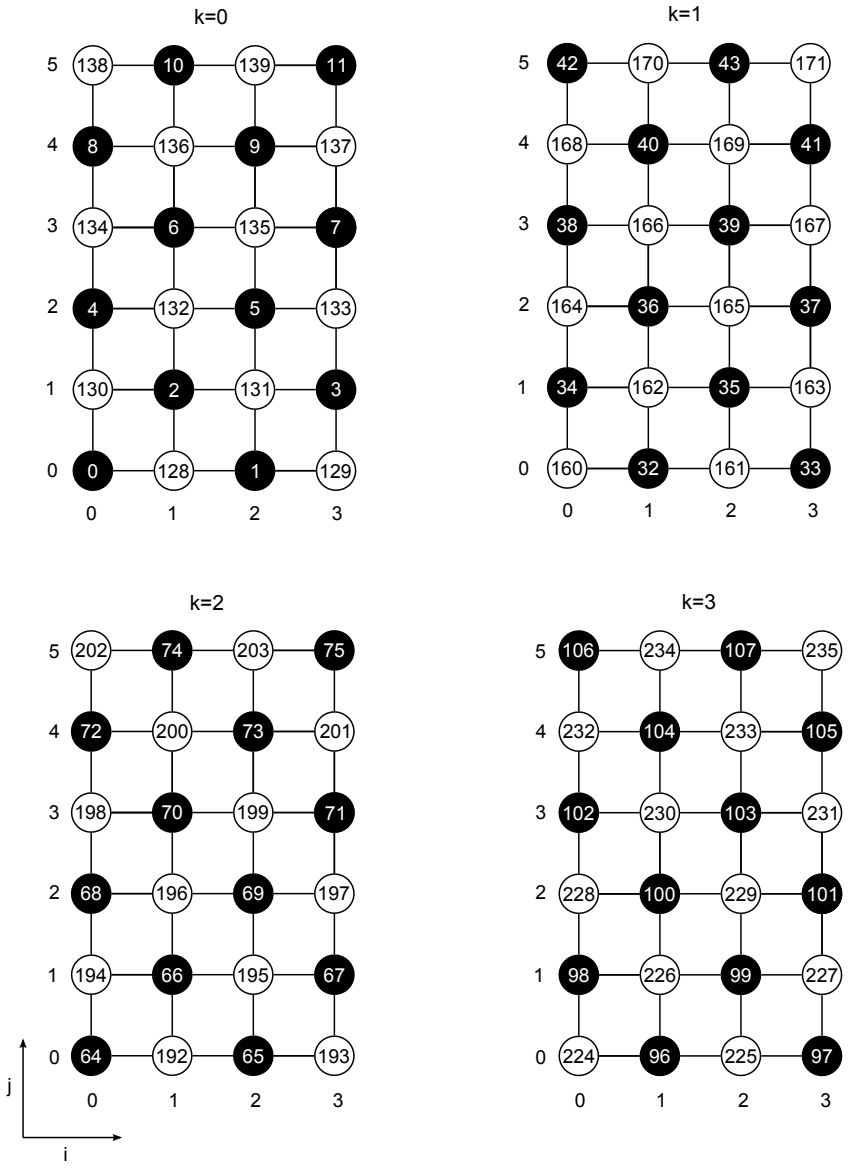
όπου με X συμβολίζεται το μέγεθος της μισής διάστασης του πίνακα αποθήκευσης και προστίθεται σε όλους του όρους καθώς όλοι οι γειτονικοί κόμβοι είναι λευκοί, άρα βρίσκονται στο δεύτερο μισό του πίνακα. Οι αντίστοιχες σχέσεις για τους λευκούς κόμβους προκύπτουν εύκολα από τα παραπάνω, αντιστρέφοντας το k, από μονό σε ζυγό και το αντίθετο. Ακόμα αντί να προστίθεται ο όρος X, θα αφαιρείται. Οι σχετικές θέσεις των γειτόνων όταν το *ni* είναι μονός αριθμός είναι παρόμοιες με όσες παρουσιάστηκαν, αν και λίγο πιο απλές.

Ένας περιορισμός που επιβάλλεται μαζί με την χρήση του χρωματισμού του πλέγματος είναι το πλήθος των κόμβων σε κάθε επίπεδο *i-j* να είναι ζυγός αριθμός. Αυτό επιδιώκεται επειδή, σε αντίθετη περίπτωση, το πλήθος των μαύρων και των λευκών κόμβων θα μεταβαλλόταν από επίπεδο σε επίπεδο και κάτι τέτοιο δεν είναι επιθυμητό προγραμματιστικά, αφού θα αύξανε δραματικά την πολυπλοκότητα προγραμματισμού, καθώς και θα είχε αρκετή επιβάρυνση στις περισσότερες εφαρμογές, ενώ μεγάλη επιβάρυνση στο χρόνο εκτέλεσης ορισμένων ειδικών περιπτώσεων.

Ένα παράδειγμα χειρισμού ενός χρωματισμένου πλέγματος φαίνεται στα σχήματα 4.7 και 4.8. Στο σχήματα αυτά παρουσιάζονται 2 περιπτώσεις, μία με μονό *ni* και μία με ζυγό. Πάνω σε κάθε κόμβο φαίνεται ο αύξοντας αριθμός του και όσοι αριθμοί δεν φαίνονται στο σχήμα αντιστοιχούν σε δεσμευμένη πλεονάζουσα μνήμη. Όπως είναι εύκολο να αντιληφθεί κανείς, με χρήση πλέγματος κατάλληλων διαστάσεων η πλεονάζουσα μνήμη μπορεί να εξαλειφθεί τελείως. Και για τις δύο περιπτώσεις έχει υποτεθεί ότι ο αριθμός των threads ανά block είναι 32.



Σχήμα 4.7: Τρόπος αποθήκευσης στη μνήμη ενός πλέγματος $6\times4\times4$ με $nthreads = 32$. Σε κάθε κόμβο του πλέγματος φαίνονται οι συντεταγμένες του i, j, k και ο αύξοντας αριθμός του, σύμφωνα με τον οποίο είναι αποθηκευμένος στην μνήμη της GPU. Σε αυτό το παράδειγμα, όπου χρησιμοποιείται μικρό πλέγμα, φαίνεται έντονα και η πλεονάζουσα μνήμη που απαιτείται από αυτή τη διαχείριση του πλέγματος, καθώς για ένα πλέγμα 96 κόμβων απαιτείται δέσμευση μνήμης 256 θέσεων.



Σχήμα 4.8: Τρόπος αποθήκευσης στη μνήμη ενός πλέγματος 5×6×4, με $nthreads = 32$.

Κεφάλαιο 5

Αποτελέσματα και συγκριτικές επιδόσεις.

Στο κεφάλαιο αυτό γίνεται παρουσίαση των αποτελεσμάτων του επιλύτη που αναπτύχθηκε καθώς και οι συγκριτικές του επιδόσεις με τον αντίστοιχο κώδικα εκτελεσμένο από την CPU. Η ανάπτυξη, η πιστοποίηση και η αξιολόγηση του κώδικας πραγματοποιήθηκε στην παράλληλη υπολογιστική συστοιχία καρτών γραφικών του ΕΘΣ, η οποία αποτελείται από 12 GPU TESLA M2050 από την εταιρεία NVIDIA, οι οποίες έχουν 3 GB GDDR5 μνήμης, μέγιστη απόδοση 515 GFlops και μέχρι 148 GB/sec ταχύτητα μεταφοράς δεδομένων. Σε κάθε 3 GPU αντιστοιχούν 2 κύριοι επεξεργαστές Quad core Intel Xeon E5620@2.4 ghz, οι οποίοι και έχουν 12 MB λανθάνουσα μνήμη και 16 MB μνήμη RAM. Είναι σημαντικό να αναφερθεί η σύγκριση γίνεται ανάμεσα στον επιλύτη που αναπτύχθηκε και τον γρηγορότερο επιλύτη της CPU του τομέα Παράλληλης Υπολογιστικής Ρευστοδυναμικής και Βελτιστοποίησης του ΕΘΣ, ο οποίος όμως είναι επιλύτης υβριδικών πλεγμάτων αν και του ανατίθενται προβλήματα με δομημένη οργάνωση. Λόγω χρήσης υβριδικού επιλύτη από την μεριά της CPU, οι πραγματικές επιταχύνσεις μεταξύ GPU και CPU είναι λίγο μικρότερες από αυτές που παρουσιάζονται στη συνέχεια.

Τα αποτελέσματα που παρουσιάζονται στο παρόν κεφάλαιο έχουν προκύψει από εκτέλεση του επιλύτη της GPU. Τα αποτελέσματα αυτά είναι ταυτόσημα με τα αποτελέσματα που προκύπτουν από τον επιλύτη της CPU, ο οποίος είναι πιστοποιημένος επιλύτης του ΕΘΣ, και έτσι, δεν χρειάζεται να πραγματοποιηθεί πιστοποίηση του επιλύτη της GPU.

Στη συνέχεια παρουσιάζονται τα αποτελέσματα από δύο διαφορετικά προβλήματα. Πρώτα παρουσιάζονται τα αποτελέσματα από την επίλυση της ροής μέσα σε έναν αγωγό και στη συνέχεια τα αποτελέσματα της επίλυσης της ροής μέσα σε πτερύγωση στροβίλου. Τέλος, παρουσιάζεται η σύγκριση ανάμεσα στους χρόνους επίλυσης της GPU και της CPU, όπου είναι και το αντικείμενο της παρούσας διπλωματικής εργασίας.

5.1 Επίλυση ροής μέσα σε αγωγό.

Ο αγωγός, μέσα στον οποίο και επιλύεται η ροή, φαίνεται στο σχήμα 5.1 σε πρόοψη και σε τριδιάστατη απεικόνιση.

Η ροή επιλύθηκε για μηδενική γωνία εισόδου και $M_{2,is} = 0.6$, όπου $M_{2,is}$ είναι ο ισεντροπικός αριθμός Mach στην έξοδο της ροής. Στο σχήμα 5.2 φαίνεται το δομημένο πλέγμα που χρησιμοποιήθηκε για την επίλυση των εξισώσεων ροής. Στο σχήμα 5.3 παρουσιάζεται η πίεση και ο αριθμός Mach της ροής πάνω στο άνω και κάτω τοίχωμα του αγωγού. Τέλος, στο σχήματα 5.4 παρουσιάζεται η κατανομή πίεσης και του αριθμού Mach σε τριδιάστατη απεικόνιση.

5.2 Επίλυση ροής μέσα σε πτερύγωση στροβίλου.

Ακόμα, με χρήση του ίδιου κώδικα, επιλύθηκε η ροή γύρο από την πτερύγωση στροβίλου που παρουσιάζεται στο σχήμα 5.5. Στο σχήμα 5.6 φαίνεται το υπολογιστικό πλέγμα πάνω στους κόμβους του οποίου λύθηκαν οι εξισώσεις ροής.

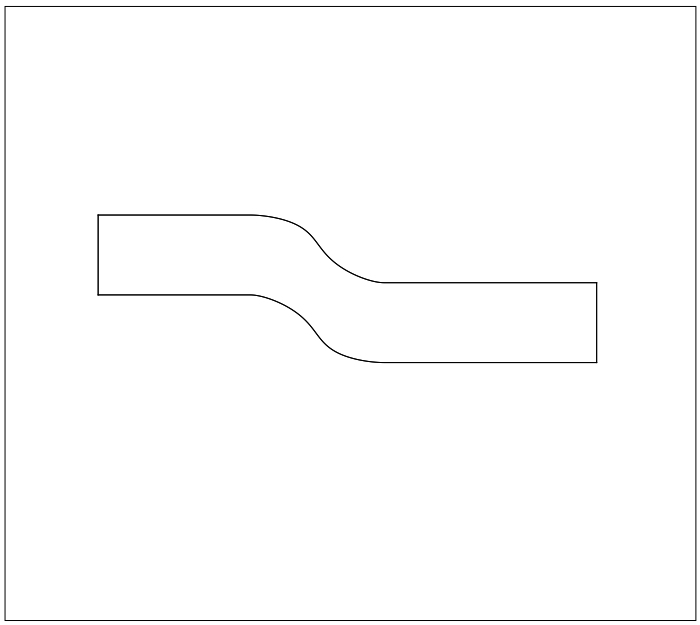
Η ροή επιλύθηκε για γωνία εισόδου της ροής $a_1 = -19.3^o$ και $M_{2,is} = 0.45$. Στο σχήμα 5.7 παρουσιάζονται οι κατανομές πίεσης και αριθμού Mach πάνω στο πτερύγιο κατά μήκος της ροής, για διάφορες ακτινικές θέσεις. Στα σχήματα 5.8 και 5.9 παρουσιάζονται οι κατανομές πίεσης και αριθμού Mach στις πλευρές υπερπίεσης και υποπίεσης του πτερυγίου και, τέλος, στο σχήμα 5.10 παρουσιάζονται οι ίδιες κατανομές, κατά μήκος της ροής, ανάμεσα σε δύο διαδοχικά πτερύγια.

5.3 Συγκριτικές επιδόσεις ανάμεσα σε GPU και CPU.

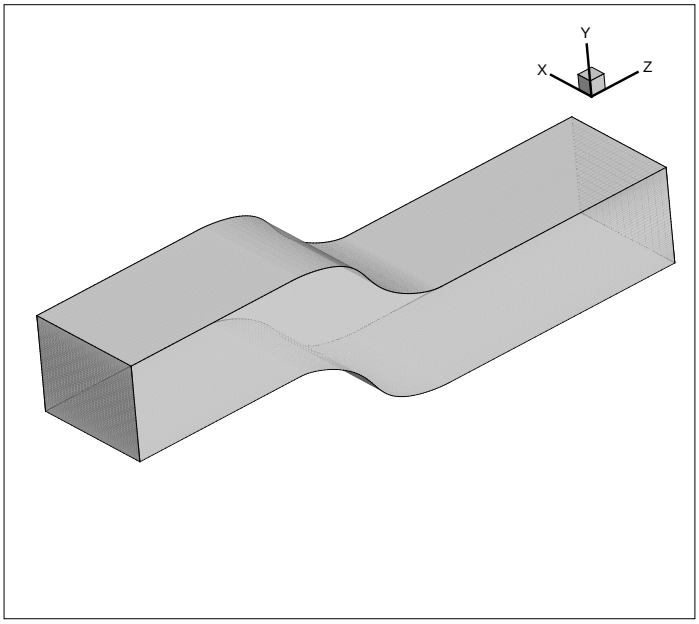
Για την σύγκριση των επιδόσεων ανάμεσα στην GPU και στη CPU, χρησιμοποιήθηκε ο επιλύτης που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας, ο οποίος λύνει τις εξισώσεις της ροής μέσω της GPU, και επιλύτης του τομέα Παράλληλης Υπολογιστικής Ρευστοδυναμικής και Βελτιστοποίησης του ΕΘΣ, ο οποίος λύνει τις εξισώσεις της ροής μέσω CPU.

Είναι σημαντικό να αναφερθεί ξανά πως ο επιλύτης της CPU επιλύει της εξισώσεις της ροής σε υβριδικό πλέγμα, οπότε και υπάρχει χρονική επιβάρυνση των επιδόσεών του, λόγο της μη-δόμησης του πλέγματος. Βέβαια, τα πλέγματα στα οποία κλήθηκε να επιλύσει τις εξισώσεις ροής είναι δομημένα, οπότε εκτιμάται πως η επιβάρυνση αυτή είναι μικρή και η πραγματική επιτάχυνση της λύσης των εξισώσεων, αν και λογικά είναι μικρότερη, δεν απέχει πολύ από αυτή που παρουσιάζεται στη συνέχεια.

Στο σχήμα 5.11 παρουσιάζεται το διάγραμμα της επιτάχυνσης του επιλύτη των εξισώσεων Euler που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας. Όπως φαίνεται και στο σχήμα, ο επιλύτης της GPU είναι από 32 έως 51 φορές πιο γρήγορος από τον επιλύτη της CPU. Αν και με την αύξηση των κόμβων του πλέγματος υπάρχει ξεκάθαρη ανοδική τάση της επιτάχυνσης του επιλύτη, φαίνεται έντονα η επίδραση της «καταλληλότητας» του πλέγματος το οποίο χρησιμοποιείται κάθε φορά. Έτσι, δημιουργούνται στο γράφημα τοπικά ελάχιστα και μέγιστα τα οποία προκύπτουν από



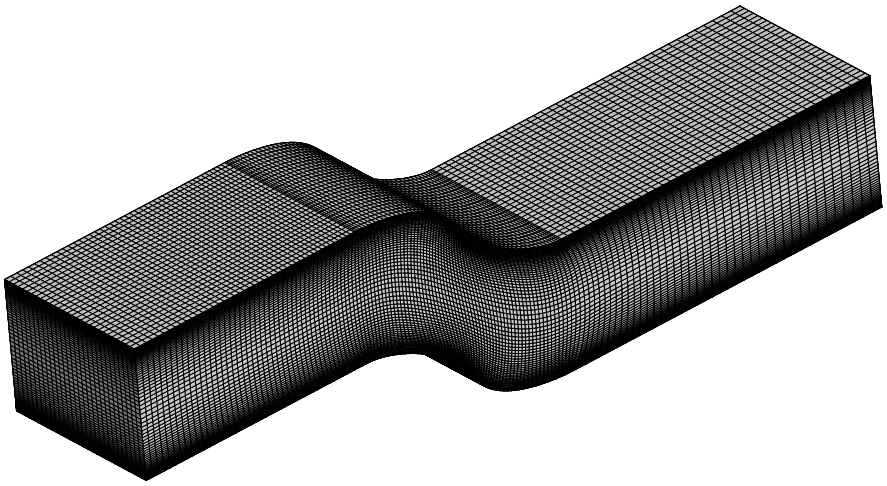
(α΄) Πρόοψη του αγωγού.



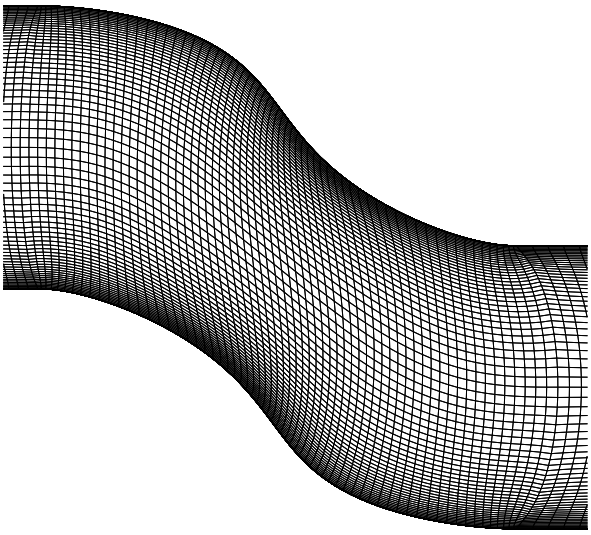
(β΄) Τριδιάστατη απεικόνιση του αγωγού.

Σχήμα 5.1: Απεικόνιση του αγωγού στον οποίο επιλύεται η ροή.

χρήση πλεγμάτων διαφορετικής «καταλληλότητας». Η «καταλληλότητα» του πλέγματος σχετίζεται με τη διδιάστατη κατανομή blocks στο grid που έχει χρησιμοποιηθεί και αναλύεται στην παράγραφο 4.3.11.

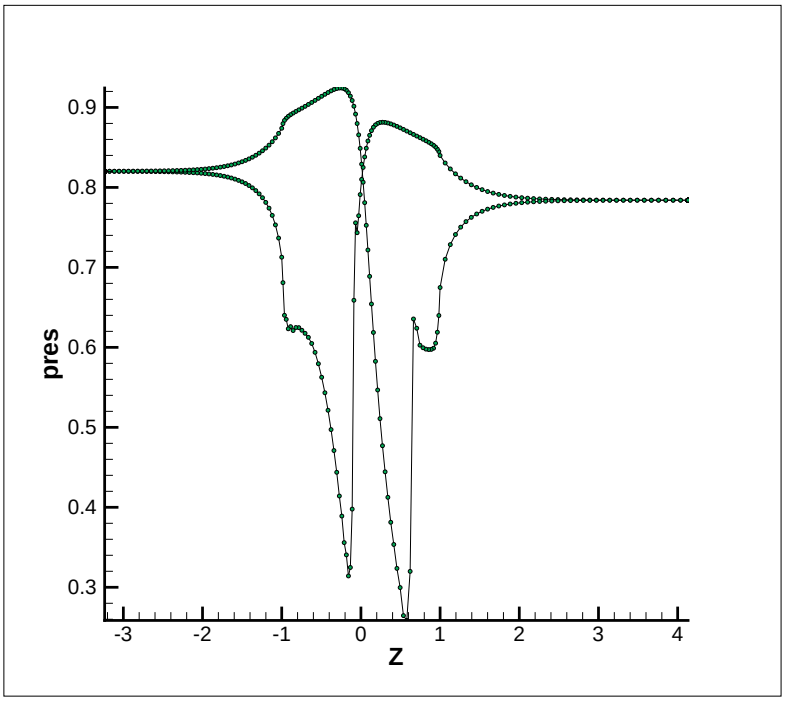


(α΄) Τριδιάστατη απεικόνιση του πλέγματος του αγωγού στον οποίο επιλύεται η ροή.

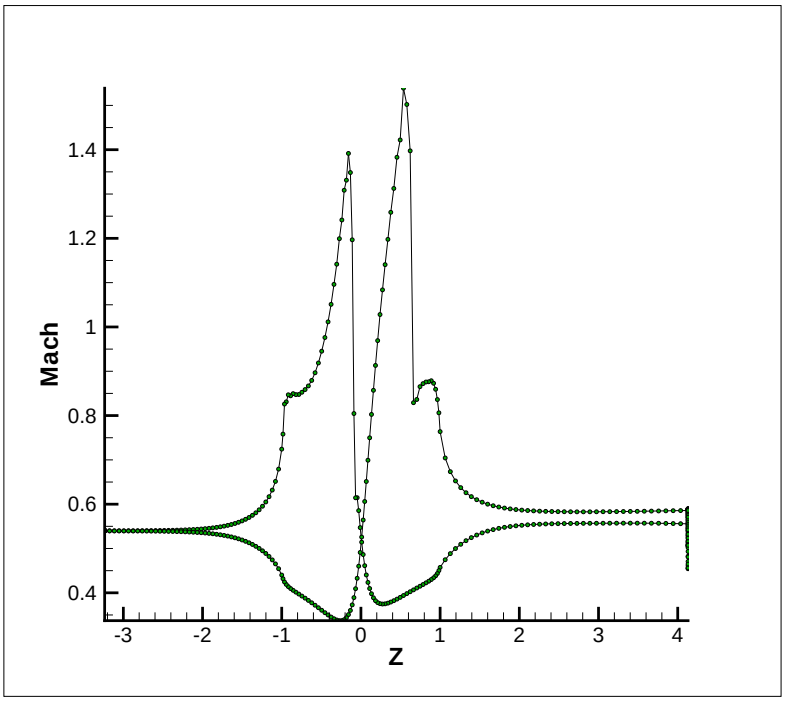


(β΄) Μεγέθυνση στο σημείο στροφής της ροής.

Σχήμα 5.2: Απεικόνιση του πλέγματος του αγωγού στον οποίο επιλύεται η ροή.

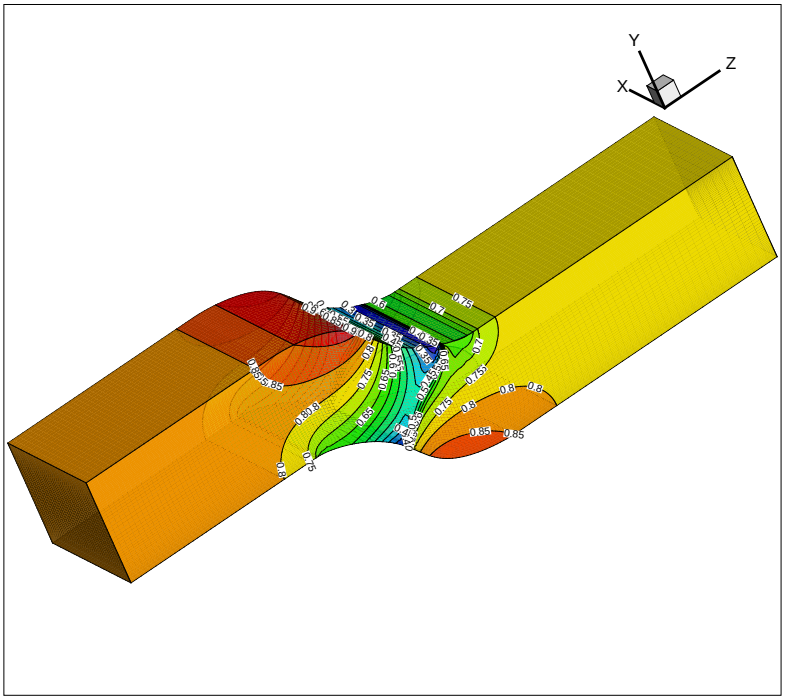


(α΄) Κατανομή αδιάστατης πίεσης κατά μήκος του αγωγού, στην άνω και κάτω πλευρά του αγωγού.

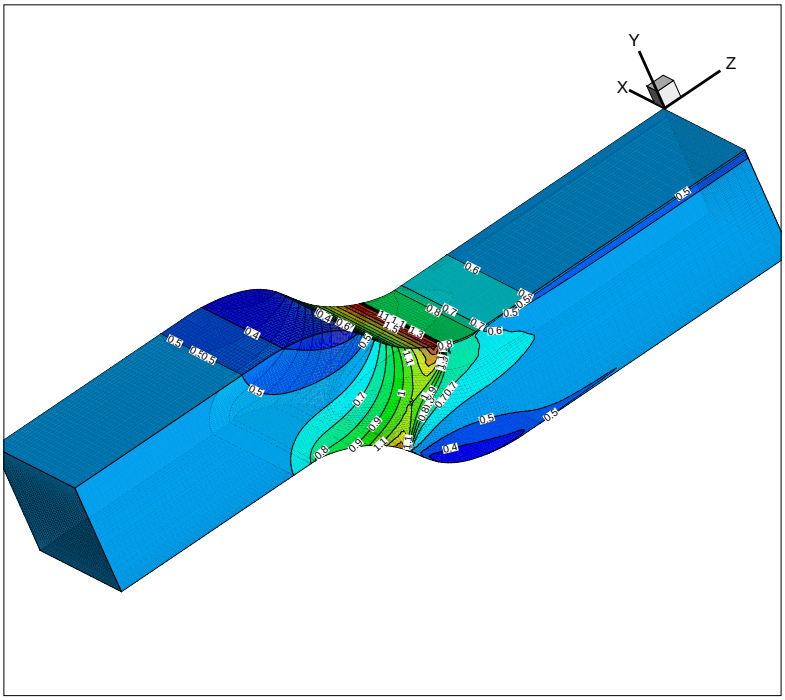


(β΄) Κατανομή αριθμού Mach κατά μήκος του αγωγού, στην άνω και κάτω πλευρά του αγωγού.

Σχήμα 5.3: Κατανομές πίεσης και αριθμού Mach του ρευστού κατά μήκος του αγωγού, στο μέσο του αγωγού.

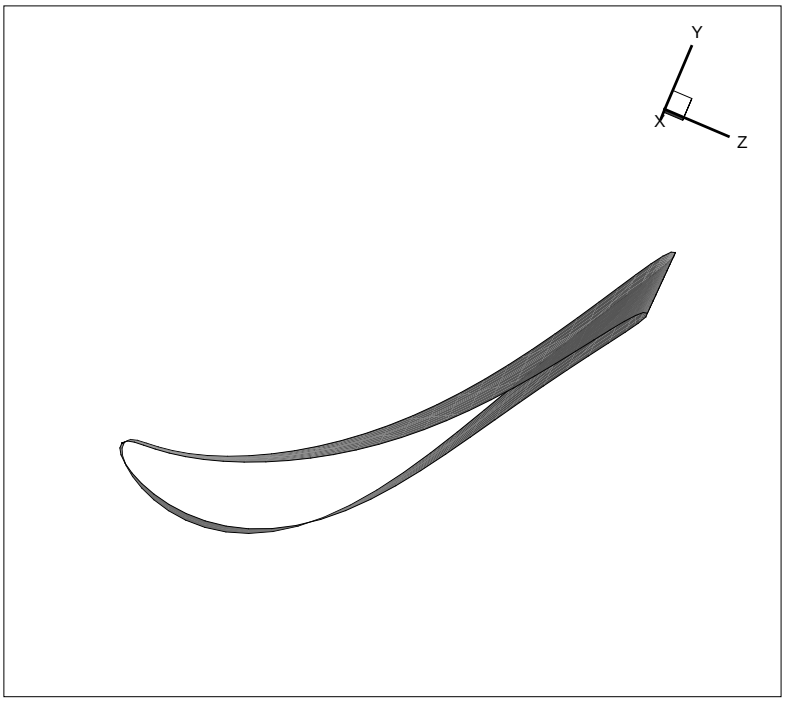


(α΄) Ισογραμμές αδιάστατης πίεσης της ροής.

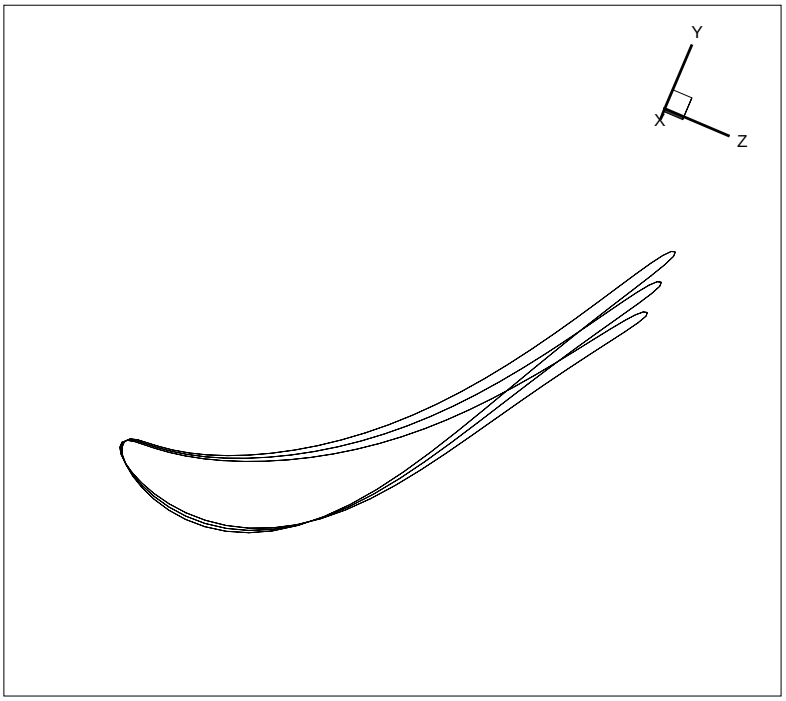


(β΄) Ισογραμμές αριθμού Mach της ροής.

Σχήμα 5.4: Ισογραμμές πίεσης και αριθμού Mach σε τριδιάστατη απεικόνιση.

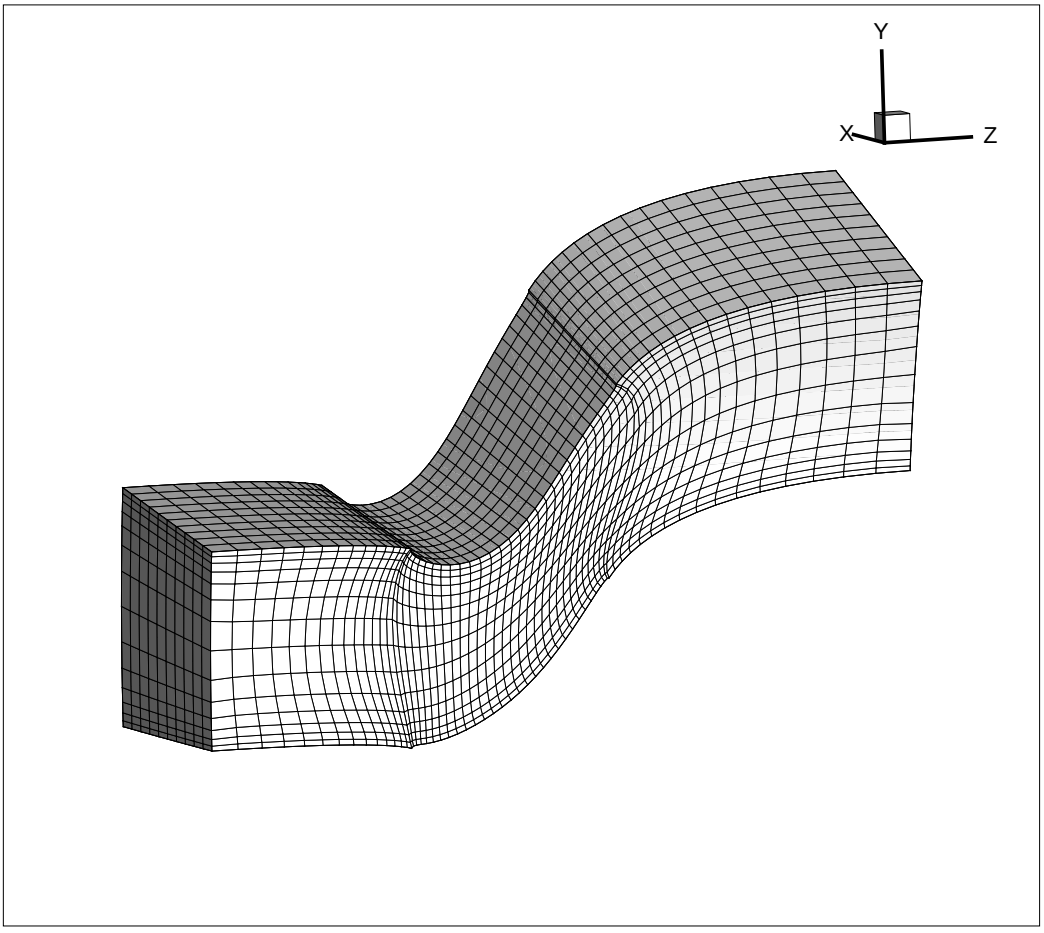


(α΄) Τριδιάστατη απεικόνιση του πτερυγίου.

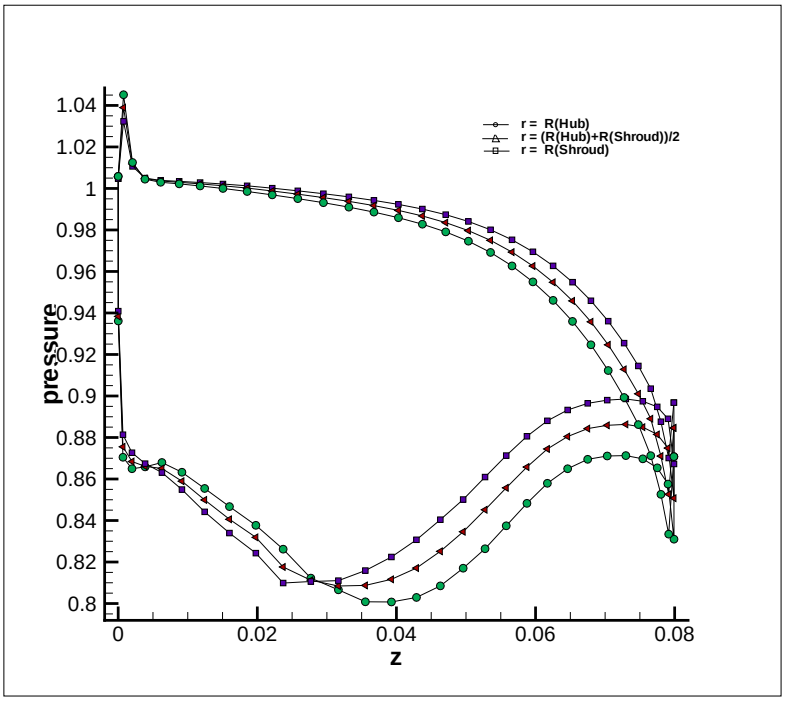


(β΄) Τομές του πτερυγίου και των επιπέδων $r = R(Hub)$, $r = (R(Hub) + R(Shroud))/2$ και $r = R(Shroud)$.

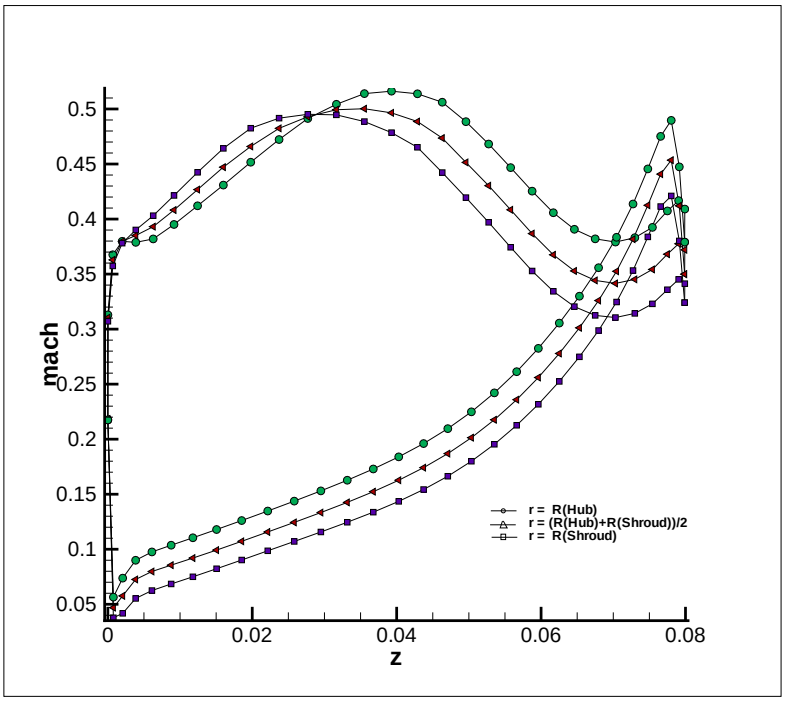
Σχήμα 5.5: Τρείς αεροτομές καθ' ύψος του πτερυγίου του στροβίλου μέσα στον οποίο επιλύεται η ροή.



Σχήμα 5.6: Υπολογιστικό πλέγμα $11 \times 15 \times 61$ πάνω στους κόμβους του οποίου λύθηκε η ροή.

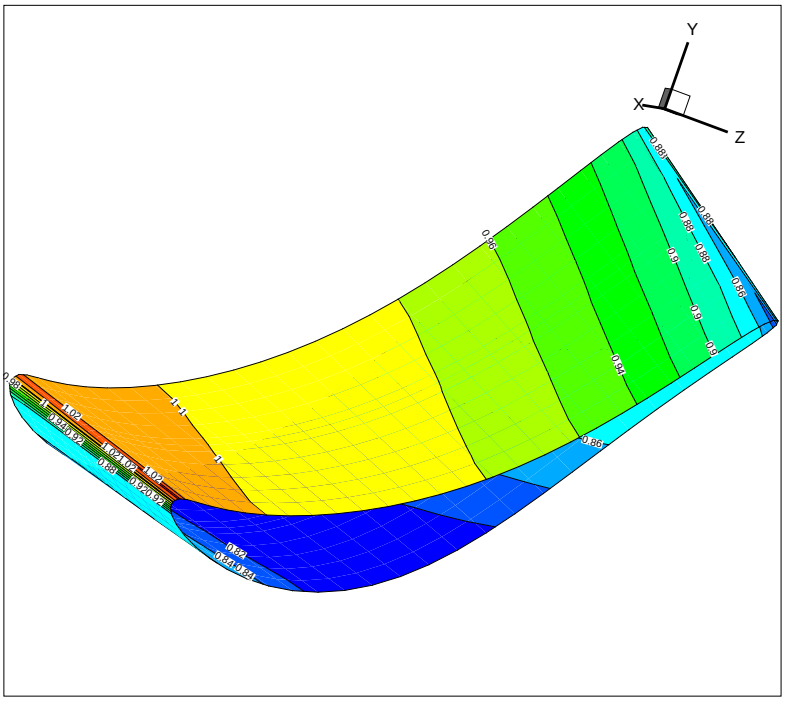


(α΄) Κατανομή αδιάστατης πίεσης πάνω στο πτερύγιο.

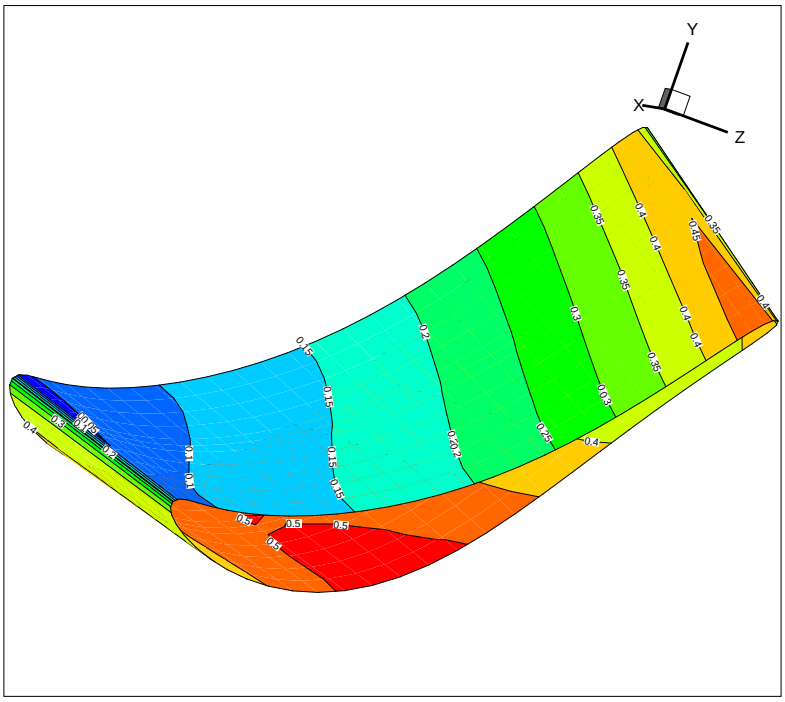


(β΄) Κατανομή αριθμού Mach πάνω στο πτερύγιο.

Σχήμα 5.7: Κατανομές πίεσης και αριθμού Mach πάνω στο πτερύγιο, στις αεροτομές του σχήματος 5.5(β΄).

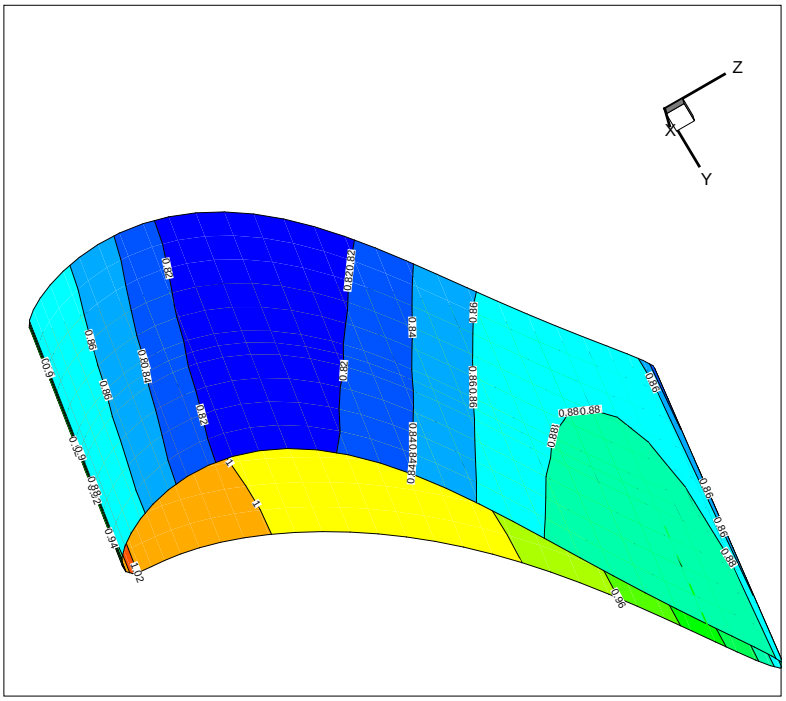


(α΄) Κατανομή αδιάστατης πίεσης πάνω στην πλευρά υπερπίεσης του πτερυγίου.

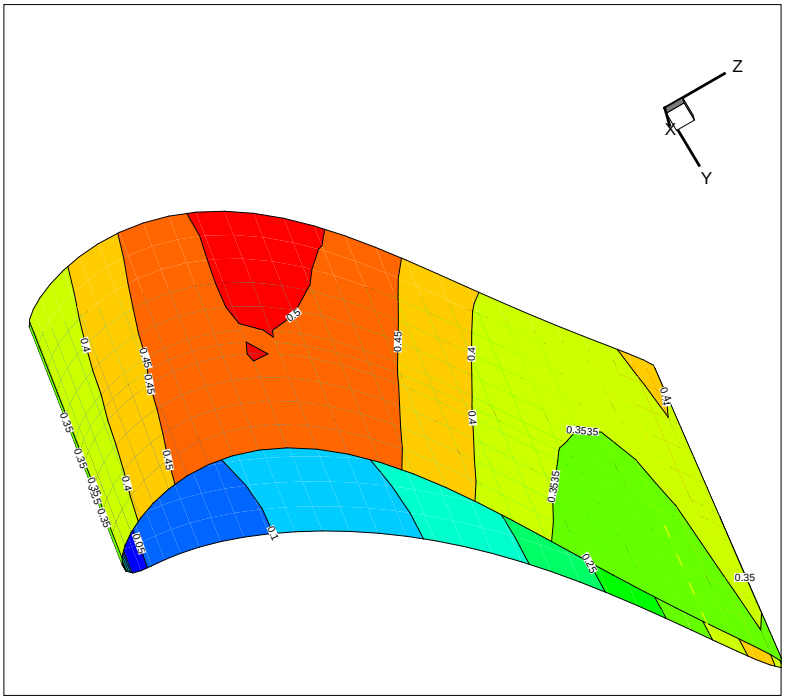


(β΄) Κατανομή αριθμού Mach πάνω στην πλευρά υπερπίεσης του πτερυγίου.

Σχήμα 5.8: Κατανομές πίεσης και αριθμού Mach πάνω στην πλευρά υπερπίεσης του πτερυγίου.

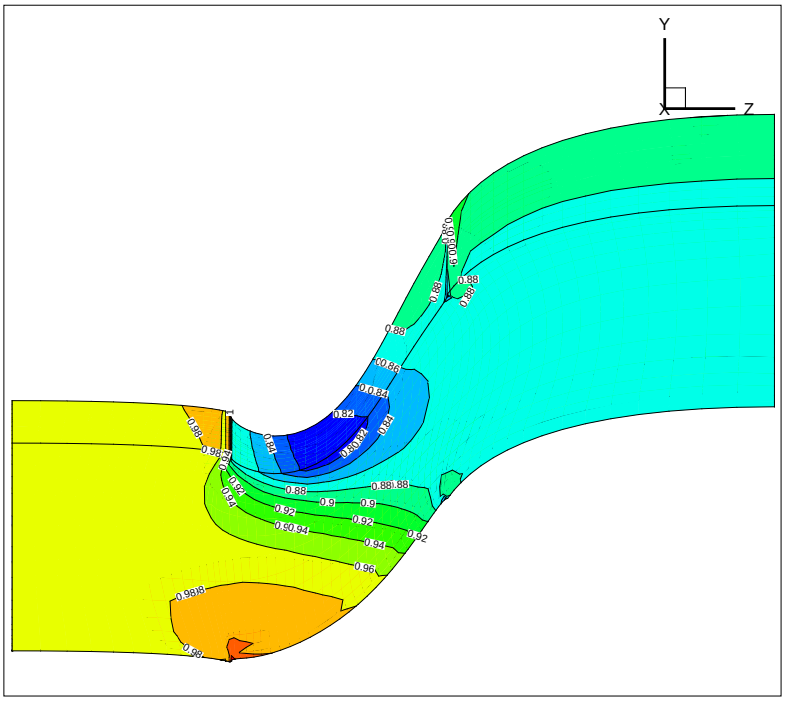


(α΄) Κατανομή πίεσης πάνω στην πλευρά υποπίεσης του πτερυγίου.

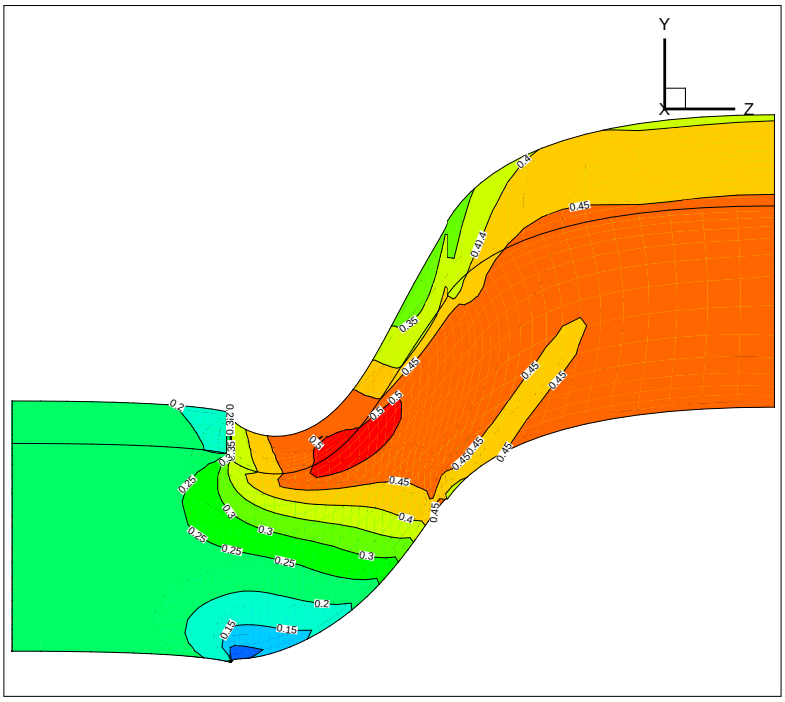


(β΄) Κατανομή αριθμού Mach πάνω στην πλευρά υποπίεσης του πτερυγίου.

Σχήμα 5.9: Κατανομές πίεσης και αριθμού Mach πάνω στην πλευρά υπερπίεσης του πτερυγίου.

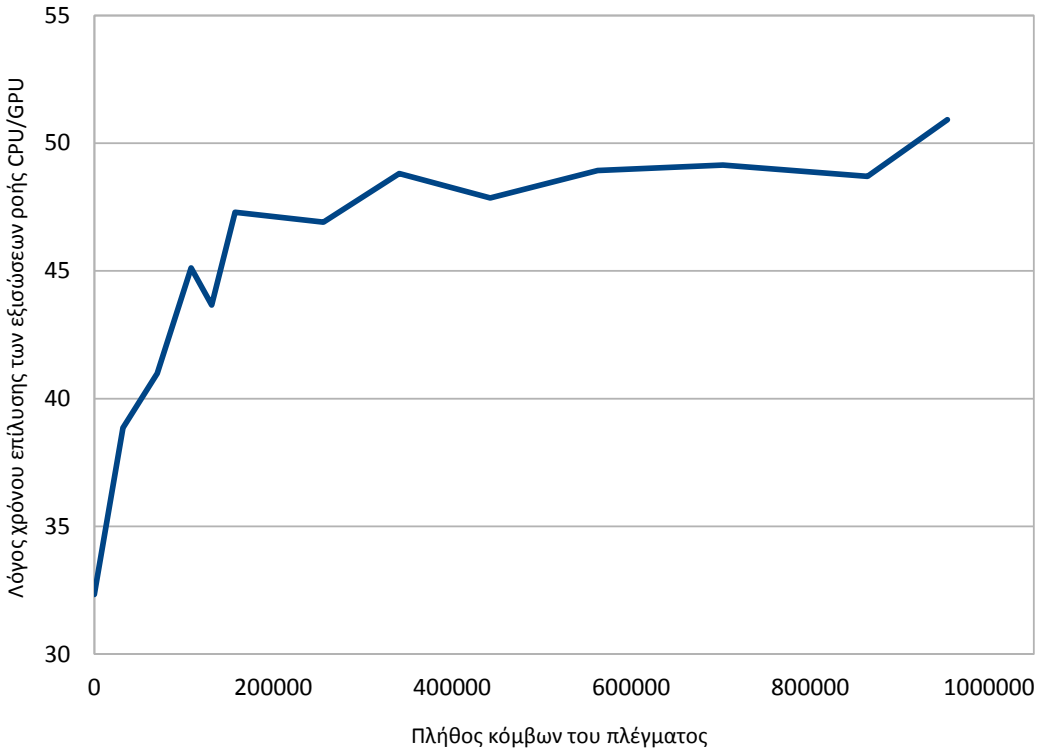


(α΄) Κατανομή αδιάστατης πίεσης ανάμεσα σε δύο διαδοχικά πτερύγια.



(β΄) Κατανομή αριθμού Mach ανάμεσα σε δύο διαδοχικά πτερύγια.

Σχήμα 5.10: Κατανομή πίεσης και αριθμού Mach ανάμεσα σε δύο διαδοχικά πτερύγια.



Σχήμα 5.11: Διάγραμμα επιτάχυνσης επιλύτη εξισώσεων Euler. Για την δημιουργία του παραπάνω διαγράμματος έγινε χρήση και «βέλτιστων» και «μη-βέλτιστων» πλεγμάτων, σύμφωνα με όσα αναφέρθηκαν στα κεφάλαια 2 και 4. Η επίδραση της «καταλληλότητας» του πλέγματος φαίνεται στην «συχνότητα» της καμπύλης, όπου τα τοπικά ελάχιστα αντιστοιχούν στις χείριστες περιπτώσεις και τα τοπικά μέγιστα στις βέλτιστες. Λόγω της διδιάστατης κατανομής threads ανά block που χρησιμοποιήθηκε στον επιλύτη της GPU (παράγραφος 4.3.11), η «καταλληλότητα» του πλέγματος σχετίζεται μόνο με το γινόμενο $ni \times nj$ των διαστάσεων του πλέγματος και όχι με το συνολικό αριθμό κόμβων του πλέγματος, όπου ni είναι το πλήθος των κόμβων κατά την διεύθυνση i και nj είναι το πλήθος των κόμβων κατά την διευύθυνση j.

Κεφάλαιο 6

Ανακεφαλαίωση και Συμπεράσματα.

Τα τελευταία χρόνια, έχει παρουσιαστεί μεγάλο ενδιαφέρον από τη Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής και Βελτιστοποίησης για την εκμετάλλευση των δυνατοτήτων των GPU σε εφαρμογές υπολογιστικής ρευστοδυναμικής και βελτιστοποίησης. Στον τομέα της υπολογιστικής ρευστοδυναμικής, με τον οποίο και ασχολείται η παρούσα διπλωματική εργασία, έχει παρουσιάσει εντυπωσιακά αποτελέσματα σε επιλύτες ροής ατριβούς και συνεκτικού ρευστού με επιταχύνσεις μεγαλύτερες από 40 φορές [12], [13]. Όμως, οι επιλύτες αυτοί κάνουν χρήση μη-δομημένων πλεγμάτων και διακριτοποιούν τις εξισώσεις ροής με ένα σχήμα πεπερασμένων όγκων κεντροκομβικής προσέγγισης.Αυτό το είδος διακριτοποίησης σε συνδιασμό με τη χρήση μη-δομημένων πλεγμάτων αποτελεί τη χειρότερη περίπτωση τρόπου επίλυσης της ροής, από άποψη απόδοση μίας GPU και της τελικής επιτάχυνσης που αυτή επιτυγχάνει.

Από την άλλη πλευρά, η χρήση δομημένων πλεγμάτων προσφέρει δομή σε όλα τα δεδομένα που χρησιμοποιούνται κατά την επίλυση των εξισώσεων της ροής. Η δόμηση αυτή θεωρητικά μπορεί να οδηγήσει σε καλύτερη αξιοποίηση των υπολογιστικών δυνατοτήτων μίας GPU και να καταλήξει σε μεγαλύτερες επιταχύνσεις από τη χρήση μη-δομημένων πλεγμάτων μέσα από καλύτερη διαχείριση της προσπέλαση της μνήμης, κάτι το οποίο είναι πολύ σημαντικό για την καλή απόδοση μίας GPU, καθώς και μέσα από πιο ομαλή παραλληλοποίηση της επίλυσης. Η πιο ομαλή παραλληλοποίηση της επίλυσης προκύπτει από το γεγονός ότι με χρήση δομημένων πλεγμάτων υπάρχει ένας προκαθορισμένος αριθμός γειτόνων σε κάθε κόμβο, ενώ όταν χρησιμοποιείται μη-δομημένο πλέγμα ο αριθμός των γειτόνων είναι μεταβλητός από κόμβο σε κόμβο.

Έτσι, η παρούσα διπλωματική εργασία έγινε με σκοπό την εκμετάλλευση των δυνατοτήτων των GPU τελευταίας τεχνολογίας σε εφαρμογές υπολογιστικής ρευστοδυναμικής με χρήση δομημένων πλεγμάτων. Πιο συγκεκριμένα, αναπτύχθηκε επιλύτης εξισώσεων ροής ατριβούς ρευστού (εξισώσεις Euler), για συμπιεστό ρευστό, απουσία βαρυτικών δυνάμεων, στο καρτεσιανό σύστημα αναφοράς, διακριτοποιημένες με ένα σχήμα πεπερασμένων όγκων κεντροκομβικής προσέγγισης, για δομημένα πλέγματα με χρήση μεθόδου χρονοπροέλασης. Ο επιλύτης προγραμματίστηκε σε γλώσσα CUDA C, η οποία αποτελεί επέκταση της γλώσσας C/C++, με στόχο την, όσο το δυνατόν, πιο αποδοτική εκμετάλλευση των δυνατοτήτων μίας GPU αρχιτεκτονικής

Fermi.

Τα αποτελέσματα που προκύπτουν από την επίλυση μίας ροής με χρήση του επιλύτη που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας είναι ταυτόσημα με τα αποτελέσματα που προκύπτουν από την επίλυση της ίδιας ροής από τον πιστοποιημένο επιλύτη της Μονάδας Παράλληλης Υπολογιστικής Ρευστομηχανικής και Βελτιστοποίησης του Εργαστηρίου Θερμικών Στροβιλομηχανών του Εθνικού Μετσόβιου Πολυτεχνείου προγραμματισμένο σε Fortran και εκτελέσιμο από την CPU, όπως και αναμενόταν καθώς ο επιλύτης του εργαστηρίου είναι και ο κώδικας πάνω στον οποίο βασίστηκε η ανάπτυξη του επιλύτη της GPU. Ακόμα, ο ίδιος επιλύτης είναι και ο επιλύτης με τον οποίο γίνεται η σύγκριση των επιδόσεων του επιλύτη της GPU που αναπτύχθηκε στη παρούσα διπλωματική.

Η σύγκριση των επιδόσεων μεταξύ των GPU και CPU παρουσίασε μέγιστη επιτάχυνση μεγαλύτερη από 50 φορές. Ο αριθμός αυτός είναι εντυπωσιακός καθώς ο επιλύτης της GPU δεν υπολείπεται σε τίποτα συγκριτικά με τον επιλύτη της CPU και ο τελικός χρήστης μπορεί να χρησιμοποιήσει και τους δύο επιλύτες με την ίδια ευκολία, χωρίς να χρειάζεται να γνωρίζει κάτι σχετικό με τη λειτουργία της εκάστοτε GPU.

Ακόμα, επιβεβαιώνεται η αρχική σκέψη πως η χρήση δομημένου πλέγματος για την λύση των εξισώσεων ροής επιτρέπει την καλύτερη αξιοποίηση των δυνατοτήτων μίας GPU. Αυτό συμβαίνει επειδή τα δύο σημαντικά προβλήματα που δεν επιτρέπουν σε μία GPU να λειτουργήσει αποδοτικά είναι η άτακτη προσπέλαση στη global μνήμη και ο διαχωρισμός των warps σε threads με διαφορετικές εντολές προς εκτέλεση. Η χρήση μη-δομημένου πλέγματος έρχεται αντιμέτωπη και με τα δύο προβλήματα καθώς το υπολογιστικό πλέγμα είναι αριθμημένο με άτακτο τρόπο και ο αριθμός των γειτόνων κάθε κόμβου είναι μεταβλητός με αποτέλεσμα όποτε χρειάζεται πληροφορία από όλους τους γείτονες ενός κόμβου τα επιμέρους threads του warp να αναλαμβάνουν να συλλέξουν πληροφορίες από άνισο αριθμό γειτόνων, κάτι το οποίο οδηγεί σε ανενεργά threads και μείωση απόδοσης. Από την άλλη πλευρά, με τη χρήση δομημένων πλεγμάτων, και τη διαχείρισή τους όπως αυτή αναλύθηκε, η προσπέλαση στη global μνήμη γίνεται με βέλτιστο τρόπο, σύμφωνα με την παράγραφο 2.3.5, και ο αριθμός των γειτόνων, τουλάχιστον όλων των εσωτερικών κόμβων, παραμένει σταθερός.

Ολόκληρη η παρούσα διπλωματική εργασία ασχολείται με τη σύγκρηση ενός επιλύτη εκτελέσιμο από μία και μόνο GPU και συγκρίνει τον χρόνο επίλυσης αυτής με έναν επιλύτη εκτελεσμένο από έναν και μόνο πυρήνα της CPU. Όμως, σήμερα σχεδόν κανείς δεν χρησιμοποιεί μόνο ένα πυρήνα της CPU, αλλά οι επιλύτες που χρησιμοποιούνται έχουν τη δυνατότητα να κατακερματίσουν το αρχικό πρόβλημα σε επιμέρους υποπροβλήματα και το κάθε ένα από αυτά να το αναλάβει μία διαφορετική CPU η οποία και με τη σειρά της θα το διασπάσει ξανά σε μικρότερα, ώστε να επωφεληθεί από όλο το πλήθος των πυρήνων της. Έτσι, το επόμενο λογικό βήμα της εκμετάλλευσης των GPU είναι η παραλληλοποίηση του αρχικού προβλήματος σε επιμέρους ώστε η επίλυσή του να μπορεί να ανατεθεί σε περισσότερες από μία GPU. Αρχικά με ανάπτυξη μεθόδου χρήσης δύο ή περισσότερων GPU που ανήκουν στον ίδιο υπολογιστή, σχήμα αντίστοιχο με μία πολυπύρηνη CPU, και στη συνέχεια με παράλληλη εκμετάλλευση των GPU διαφορετικών υπολογιστών του ίδιου δικτύου, σχήμα αντίστοιχο με

παράλληλη επεξεργασία ενός προβλήματος σε πολλές CPU ταυτόχρονα. Η ανάπτυξη της μεθόδου αυτής έχει ήδη ξεκινήσει, από μία εν εξέλιξη διπλωματική εργασία της Μονάδας Υπολογιστικής Ρευστομηχανικής και Βελτιστοποίησης, με στόχο την παραλληλοποίηση του επιλύτη που παρουσιάστηκε στην παρούσα διπλωματική εργασία και την εκμετάλλευση ολόκληρης της παράλληλης υπολογιστικής συστοιχίας καρτών γραφικών της Μονάδας Υπολογιστικής Ρευστομηχανικής και Βελτιστοποίησης.

Αξίζει ακόμα να σημειωθεί πως η χρήση των GPU σε εφαρμογές γενικού προγραμματισμού είναι ένας σχετικά καινούργιος τομέας και αυτό φαίνεται εύκολα να σκεφτεί κανείς πως οι πρώτες GPU αρχιτεκτονικής CUDA κυκλοφόρησαν στην αγορά μόλις το 2006. Έτσι, προσωπική εκτίμηση του γράφοντα είναι πως στο προσεχές μέλλον θα παρουσιαστούν ακόμα μεγαλύτερες επιταχύνσεις, τόσο σε εφαρμογές υπολογιστικής ρευστοδυναμικής όσο και σε άλλους τομείς, καθώς αναπτύσσεται περισσότερο η τεχνολογία αλλά και η τεχνογνωσία γύρω από τη χρήση των GPU σε εφαρμογές γενικού προγραμματισμού.

Τελικά, η μετατροπή ενός ήδη υπάρχοντα κώδικα της CPU σε κώδικα κατάλληλο για εκτέλεση από την GPU είναι χρονοβόρα και απαιτητική εργασία καθώς πρέπει ένας σειριακός κώδικας, προορισμένος για την CPU η οποία εκτελεί πράξεις σειριακά, να μετατραπεί σε έναν πλήρως παράλληλο κώδικα για να είναι αποδοτική η χρήση της GPU, η οποία εκμεταλλεύεται πλήρως παράλληλα τους επεξεργαστές της. Όμως, μία τελική επιτάχυνση όπως η παραπάνω κάνει εμφανές το γιατί όλο και περισσότερο αυξάνεται το ενδιαφέρον της επιστημονικής κοινότητας για το GPGPU σε επιστημονικές εφαρμογές, όπως είναι η επιτάχυνση εφαρμογών υπολογιστικής ρευστοδυναμικής στην παρούσα εργασία εξεταζόμενη με χρήση των GPU αρχιτεκτονικής Fermi.

Βιβλιογραφία

[1] NVIDIA. NVIDIA CUDA C Programming Guide, Version 3.2, October 2010.

[2] Teng-Yi Huang, Yu-Wei Tang, Shiun-Ying Ju. Accelerating image registration of MRI by GPU-based parallel computation, Magnetic Resonance Imaging 29 (2011) 712-716.

[3] Carolyn L. Phillips, Joshua A. Anderson, Sharon C. Glotzer. Pseudo-random number generation for Brownian Dynamics and Dissipative Particle Dynamics simulations on GPU devices, Journal of Computational Physics 230 (2011) 7191-7201.

[4] Alfeus Sunarso, Tomohiro Tsuji, Shigeomi Chono. GPU-accelerated molecular dynamics simulation for study of liquid crystalline flows, Journal of Computational Physics 229 (2010) 5486-5497.

[5] Yan Zhang, Panagiotis Vouzis, Nikolaos V. Sahinidis. GPU simulations for risk assessment in co_2 geologic sequestration, Computers and Chemical Engineering 35 (2011) 1631-1644.

[6] A. Heimlich, A.C.A. Mol, C.M.N.A. Pereira. GPU-based Monte Carlo simulation in neutron transport and finite differences heat equation evaluation, Progress in Nuclear Energy 53 (2011) 229-239.

[7] Erich Elsen, Patrick LeGresley, Eric Darve. Large calculation of the flow over a hypersonic vehicle using a GPU, Journal of Computational Physics 227(2008)10148-10161.

[8] Diego Rossinelli, Michael Bergdorfm Georges-Henri Cottet, Petros Koumoutsakos. GPU accelerated simulations of bluff body flows using vortex particle methods, Journal of Computational Physics 299(2010)3316-3333.

[9] Wangda Zuo, Qingyan Chen. Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit, Building and Environment 45(2010)747-757.

[10] Kazuhiko Komatsu, Takashi Soga, Ryusuke Egawa, Hiroyuki Takizawa, Hiroaki Kobayashi, Shun Takahashi, Daisuke Sasaki, Kazuhiro Nakahashi. Parallel processing of the Building-Cube Method on a GPU platform, Computers & Fluids 45(2011)122-128.

[11] Alfed j. Kalyanapu, Siddharth Shankar, Eric R. Pardyjak, David R. Judi, Steven J. Burian. Assessment of GPU computational enhancement to a 2D flood model, Enviromental Modelling & Software 26(2011)1009-1016.

[12] V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, K. C. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units, International journal for numerical methods in fluids, Int. J. Numer. Meth. Fluids (2010).

[13] Βαλσαμάκης Γεώργιος. Αριθμητική επίλυση μη-μόνιμου πεδίου ροής σε κάρτες γραφικών με απεικόνιση του σε «πραγματικό» χρόνο, Διπλωματική εργασία, Εργαστήριο Θερμικών Στροβιλομηχανών ΕΜΠ, Φεβρουάριος 2010.

[14] Γεώργιος Δ. Ρήγας. Προσομοίωση και χαμηλού κόστους βελτιστοποίηση του ενεργητικού ελέγχου ροής ρευστού γύρω από αεροτομή, σε κάρτες γραφικών, Διπλωματική εργασία, Εργαστήριο Θερμικών Στροβιλομηχανών ΕΜΠ, Ιούλιος 2010.

[15] Ελευθέριος Ι. Φούντης. Προγραμματισμός σε Κάρτες Γραφικών και Εφαρμογη στην Αεροδυναμική Βελτιστοποίηση, Διπλωματική εργασία, Εργαστήριο Θερμικών Στροβιλομηχανών ΕΜΠ, Οκτώβριος 2009.

[16] I. C. Kampolis, X. S. Trompoukis, V. G. Asouti, K. C. Giannakoglou. CFD-based analysis and two-level aerodynamic optimization on graphics processing units, Computer Methods in Applied Mechanics and Engineering, Volume 199, Issues 9-12, 15 January 2010, Pages 712-722.

[17] X.S. Trompoukis, V.G. Asouti, I.C. Kampolis, K.C. Giannakoglou. CUDA implementation of Vertex-Centered, Finite Volume CFD methods on Unstructured Grids with Flow Control Applications, GPU Computing Gems Vol. 2, Editor: Wei-mei Hwu, Morgan Kaufmann, 2011 (to be published).

[18] NVIDIA. NVIDIA CUDA Architecture, Introduction & Overview, Version 1.1, April 2009.

[19] Jason Sanders, Edward Kandrot. NVIDIA, CUDA by example, An introduction to General-Purpose GPU Programming, 2011.

[20] NVIDIA. NVIDIA's Next Generation CUDA ComputeArchitecture: Fermi, 2009.

[21] NVIDIA. NVIDIA CUDA C Programming, Best Practices Guide, CUDA Toolkit 2.3, July 2009.

[22] Ξ. Σ. Τρομπούκης. Υπολογιστική ανάλυση και παραμετρική διερεύνηση της τεχνικής συνεχούς αναρρόφησης για τον έλεγχο οριακών στρωμάτων, Διπλωματική εργασία, Εργαστήριο Θερμικών Στροβιλομηχανών ΕΜΠ, Οκτώβριος 2007.

[23] Θ. Δ. Ζερβογιάννης. Μέθοδοι βελτιστοποίησης στη αεροδυναμική και τις στροβιλομηχανές με χρήση συζυγών τεχνικών, υβριδικών πλεγμάτων και του ακριβούς εσσιανού μητρώου, Διδακτορική Διατριβή, Εργαστήριο Θερμικών Στροβιλομηχανών ΕΜΠ, Αθήνα 2011.

[24] Roe, P. L. Approximate Riemann solvers, parameter vectors and difference schemes, Journal of Computational Physics, 135:250-258, 1997.

[25] van Albada, G. D., van Leer, B., and Roberts, W. W. A comparative study of computational methods in cosmic gas dynamics, Astronomy and Astrophysics, 108:76-84, 1982.

[26] Pulliam, T. H. and Steget, J. L. Recent impovements in efficiency, accuracy and convergence for implicit approximate factorization algorithms, AIAA-85-03060, 1985.

[27] Courant, R., Friedrichs, K. and Lewy, H. über die partiellen differenzengleichungen der mathematischen physik, Mathematische Annalen 1928; 100:32-74.

[28] Steger, P. and Warming, R. F. Flux vector splitting of the inviscid gasdynamic equations with application to the finite-difference methods, Journal of Computational Physics, 40:263-293, 1981.

[29] Fezoui, L. and Stoufflet, B. A class of implicit upwind schemes for Euler simulations with unstructured meshes, Journal of Computational Physics, 84:174-206, 1989.