2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Μηχανολόγων Μηχανικών
Εργαστήριο Θερμικών Στροβιλομηχανών

[ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΚΑΡΤΕΣ ΓΡΑΦΙΚΩΝ ΚΑΙ ΕΦΑΡΜΟΓΗ ΣΤΗΝ ΑΕΡΟΔΥΝΑΜΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ]

Διπλωματική Εργασία του Φουντή Ελευθέριου
Επιβλέπων : Κυριάκος Χ. Γιαννάκογλου

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Εργαστήριο Θερμικών Στροβιλομηχανών

Προγραμματισμός σε Κάρτες Γραφικών και Εφαρμογή στην Αεροδυναμική Βελτιστοποίηση

του **ΕΛΕΥΘΕΡΙΟΥ Ι. ΦΟΥΝΤΗ**

Επιβλέπων : ΚΥΡΙΑΚΟΣ Χ. ΓΙΑΝΝΑΚΟΓΛΟΥ, Αναπληρωτής Καθηγητής ΕΜΠ

ΑΘΗΝΑ , ΟΚΤΩΒΡΙΟΣ 2009

Η διπλωματική αυτή εργασία πραγματεύεται ένα θέμα σημαντικής αξίας στο επάγγελμα του μηχανικού, αυτό της μείωσης του υπολογιστικού κόστους λογισμικού που αναλύει μηχανολογικές εφαρμογές, κυρίως αυτές που είναι μεγάλης κλίμακας.

Μέχρι σήμερα οι κάρτες γραφικών χρησιμοποιούνταν στους ηλεκτρονικούς υπολογιστές για την εκτέλεση υπολογισμών με σκοπό την επεξεργασία γραφικών και την προβολή τους στην οθόνη του υπολογιστή σε εφαρμογές όπως παιχνίδια. Η συνεχής απαίτηση για όλο και περισσότερο αληθοφανή και τριδιάστατα γραφικά σε αυτές τις εφαρμογές, οδήγησε τις προγραμματιζόμενες μονάδες επεξεργασίας γραφικών να εξελίσσονται συνεχώς.

Μέσα από αυτήν την εξέλιξη και εξαιτίας της ραγδαίας αύξησης της υπολογιστικής τους ισχύος σε σχέση με αντίστοιχη των κεντρικών επεξεργαστών, απέκτησαν μια νέα διαφορετική χρήση, στον τομέα της μεγάλης κλίμακας επιστημονικών, παράλληλων υπολογισμών (Large Scale Parallel Computing).

Η παρούσα διπλωματική εργασία, είναι η πρώτη στο Εργαστήριο Θερμικών Στροβιλομηχανών του ΕΜΠ που ασχολείται με αυτήν τη νέα τεχνολογία. Συγκεκριμένα, παρουσιάζεται η λειτουργία της νέας αρχιτεκτονικής της κάρτας γραφικών η οποία επιτρέπει τη χρησιμοποίηση της υπολογιστικής ισχύος της κάρτας για υπολογισμούς μεγάλης κλίμακας, ενώ στη συνέχεια παρουσιάζεται η

γλώσσα προγραμματισμού CUDA, με την οποία ο χρήστης έχει τη δυνατότητα να προγραμματίσει τις εφαρμογές του για εκτέλεση στην κάρτα.

Στη συνέχεια, κάνοντας προετοιμασία για αληθινούς υπολογισμούς, χρησιμοποιείται λογισμικό επίλυσης των εξισώσεων Navier-Stokes[1] έτοιμο για εκτέλεση σε κάρτα γραφικών, με σκοπό τον υπολογισμό της πολικής καμπύλης της αεροτομής ενός επιβατικού υπερηχητικού αεροσκάφους, διαδικασίας με υψηλό υπολογιστικό κόστος. Επιπλέον, προγραμματίζεται σε CUDA ένα υπάρχον μοντέλο προκαταρκτικού σχεδιασμού υπερηχητικού αεροσκάφους[2], το οποίο χρησιμοποιεί την αεροτομή και την πολική καμπύλη του προηγούμενου τμήματος για την εκτέλεση του, ενώ τίθεται το μοντέλο του αεροσκάφους υπό βελτιστοποίηση με χρήση εξελικτικών αλγορίθμων ώστε να γίνει πιο εμφανής η επίδραση της χρήσης κάρτας γραφικών στη μείωση του υπολογιστικού κόστους της εφαρμογής μας.

Τέλος, παρουσιάζεται ένας γενέτης διδιάστατων μη-δομημένων πλεγμάτων, που δημιουργήθηκε ως ουσιαστικό παραπροϊόν της παρούσας διπλωματικής και αποτελεί εργαλείο χρήσιμο για την υπολογιστική ρευστοδυναμική ενώ παράλληλα χρησιμοποιήθηκε για τη δημιουργία του πλέγματος της αεροτομής του πρακτικού τμήματος της εργασίας.

Λέξεις κλειδιά : Κάρτα γραφικών, Compute Unified Device Architecture, Υπολογιστική Ρευστοδυναμική, Αεροδυναμική Βελτιστοποίηση, Εξελικτικοί Αλγόριθμοι, Γένεση Μη-Δομημένου Υπολογιστικού Πλέγματος.

[1] Το πρόγραμμα μας δίνεται έτοιμο από το Εργαστήριο Θερμικών Στροβιλομηχανών του ΕΜΠ και αποτελεί αποτέλεσμα της εργασίας των διδακτόρων Ι.Καμπόλη , Β.Ασούτη και Ξ.Τρομπούκη

[2] Το υπολογιστικό μοντέλο του αεροσκάφους αποτελεί μέρος της διδακτορικής διατριβής της διδάκτορος Βαρβάρας Ασούτη.

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
School of Mechanical Engineering
Department of Fluids
Laboratory of Thermal Turbomachines

Programming on Graphics Processing Units and Implementation in Aerodynamic Optimization

of **Eleftherios I. FOUNTIS**

Advisor : KYRIAKOS C. GIANNAKOGLOU, Associate Professor NTUA

ATHENS , OCTOBER 2009

This Diploma Thesis deals with the reduction of the computational cost of computations related to mechanical engineering applications that involve processing and calculations of scientific data, in particular large scale ones on Graphics Processing Units (GPU).

So far the graphics cards were used in personal computers to perform the calculations required to process graphics and display them on the computer screen in applications such as games. The demand for more realistic and three-dimensional graphics in these applications led programmable graphics processing units to evolve continuously.

Through this development and rapid growth of their computing power, in contrast to those of central processing units, GPUs have acquired a new different use in large-scale parallel scientific computations.

This Diploma Thesis is the first in the Laboratory of Thermal Turbomachines of NTUA on this new technology. In particular, it shows the operation of the new architecture of the GPUs which allows the use of computational power of the card for large scale calculations and then presents the programming language CUDA, used to program applications on GPUs.

In addition, by making preparation for real calculations, a software, ready for execution on GPU, is used to solve the Navier-Stokes [3] equations in order to calculate the polar curve of an airfoil of a supersonic business jet, a process with high

[3] The program is work of I. Kampolis , V. Asouti , X. Trompoukis and given ready for use from LTT

computational cost. In addition, an existing model of preliminary design[4] of a supersonic aircraft, is reprogrammed in CUDA to use the airfoil and the polar curve of the previous section for its execution. The new model is used for optimization by means of evolutionary algorithms.

Finally, a new program for generation of two-dimensional unstructured grids is presented, created as an essential by-product of this diploma thesis, for use with CFD codes for the computation of the flow developed around the airfoil in our essay.

Keywords: Graphics Processing Unit, Compute Unified Device Architecture, Computational Fluid Dynamics, Aerodynamic Optimization, Evolutionary Algorithms, Genesis of Unstructured Grids.

[4] The model of preliminary design of supersonic business jet is work of PhD student V. Asouti.

Αντί Προλόγου..

Θα ήθελα από αυτή τη θέση να ευχαριστήσω τα άτομα που με βοήθησαν και με στήριξαν στα χρόνια της φοιτητικής μου πορείας και που χωρίς αυτά τόσο η ολοκλήρωση των σπουδών μου στη σχολή Μηχανολόγων Μηχανικών, όσο και η παρούσα διπλωματική εργασία δύσκολα θα είχαν ολοκληρωθεί.

Αρχικά την οικογένεια μου, για τις θυσίες τους όλα αυτά τα χρόνια δίνοντας μου τη δυνατότητα να γίνω αυτό που είμαι σήμερα καθώς και τους φίλους μου για την στήριξη τους.

Ιδιαίτερα θα ήθελα να ευχαριστήσω τον καθηγητή μου, κ. Γιαννάκογλου που διέθεσε την πείρα του και τον χρόνο του, καθοδηγώντας με σε νέους ορίζοντες.

Τέλος, θα ήθελα να ευχαριστήσω όλο το προσωπικό του Εργαστηρίου Θερμικών Στροβιλομηχανών του ΕΜΠ και ειδικά τους διδάκτορες ΕΜΠ Βαρβάρα Ασούτη και Ιωάννη Καμπόλη για την συνεχή βοήθεια τους στο πρώτο και δεύτερο τμήμα της διπλωματικής μου.

Φουντής Ελευθέριος
Οκτώβριος 2009

Περιεχόμενα

Κεφάλαιο 1 – Εισαγωγή: 7

1.1 Γενικά Περί Υπολογιστικών Συστημάτων 13

1.2 Η Κάρτα Γραφικών 14

1.3 Χαρακτηριστικά της Κάρτας Γραφικών 16

1.4 Πρόσφατα Αποτελέσματα στην Υπολογιστική Ρευστοδυναμική μέσω GPU 18

1.5 Σύγχρονες Ερευνητικές Δραστηριότητες στο Εργαστήριο Θερμικών Στροβιλομηχανών του ΕΜΠ 18

1.6 Περιεχόμενα και Δομή της Εργασίας 21

Κεφάλαιο 2 – Αρχιτεκτονική CUDA: 23

2.1Βασικές Λειτουργίες της Κάρτας Γραφικών 25

2.2 Βασικές Έννοιες και Παράμετροι 26

2.3 Οργάνωση των Εκτελέσιμων Νημάτων 27

2.4 Πυρήνες-Επεξεργαστές της Κάρτας Γραφικών 29

2.5 Εκτέλεση των Threads 30

2.6 Θέσεις Μνήμης της Κάρτας 31

2.6.1 Constant Memory 34

2.6.2 Shared Memory 34

2.6.3 Registers 39

2.6.4 Global Memory 39

2.6.5 Texture Memory 40

2.7 Συμβατές GPU και Υπολογιστικές δυνατότητες 40

2.8 Το Υπολογιστικό Σύστημα που χρησιμοποιήθηκε 41

2.8.1 Κάρτα Γραφικών GTX285 41

2.8.2 Επεξεργαστής Core 2 Duo E8400 42

Κεφάλαιο 3 – Γλώσσα προγραμματισμού CUDA: 45

3.1 Προέκταση της C 46

3.1.1 Προσδιοριστές Υπορουτινών 47

3.1.2 Προσδιοριστές Μεταβλητών 48

3.1.3 Προσδιοριστικά Κλήσης Υπορουτίνας 49

3.1.4 Μεταβλητές Θέσης και Διαστάσεων 49

3.2 Προσθήκες Βιβλιοθήκης Εκτελέσιμων Εντολών 50
3.2.1 Βιβλιοθήκη Κοινών Εντολών 50
3.2.2 Βιβλιοθήκη Εντολών Συσκευής 51
3.2.3 Βιβλιοθήκη Εντολών Host 51
3.3 Ασύγχρονη Λειτουργία 52
3.4 Μαθηματικές Συναρτήσεις 53
3.4.1 Κοινές Συναρτήσεις Βιβλιοθήκης 53
3.4.2 Συναρτήσεις Βιβλιοθήκης Συσκευής 55
3.5 Εικονική Λειτουργία 56
3.6 Παράδειγμα 58

Κεφάλαιο 4 – Επίλυση της Ροής : 61
4.1 Εφαρμογή 61
4.2 Αποτελέσματα 62
4.2.1 Γωνία αποκόλλησης 62
4.2.2 Πολική καμπύλη 63
4.2.3 Χρόνος εκτέλεσης 63

Κεφάλαιο 5 - Το μοντέλο του Αεροσκάφους : 67
5.1 Λεκτική διατύπωση του προβλήματος 67
5.2 Μεταβλητές σχεδιασμού 67
5.3 Αλγόριθμος Επίλυσης 69
5.4 Ανάλυση του μοντέλου του αεροσκάφους 70
5.4.1 Τυπική Ατμόσφαιρα 70
5.4.2 Υπολογισμός Γεωμετρικών Μεγεθών του Αεροσκάφους 72
5.4.2.1 Άτρακτος 72
5.4.2.2 Πτέρυγα 74
5.4.2.3 Κάθετο Ουραίο Πτερύγιο 75
5.4.3 Υπολογισμός και Διαστασιολόγηση Κινητήρων και Κελυφών 76
5.4.4 Υπολογισμός Βάρους 77
5.4.5 Υπολογισμός Αεροδυναμικών μεγεθών 79
5.4.5.1 Συντελεστής Άνωσης 79
5.4.5.2 Συντελεστής Οπισθέλκουσα 81

5.4.6 Υπολογισμός Επιδόσεων *83*

5.4.6.1 Εμβέλεια *83*

5.4.6.2 Ταχύτητα Προσέγγισης *83*

5.4.6.3 Μήκος Απογείωσης *84*

5.5 Επιτάχυνση Μοντέλου 85

Κεφάλαιο 6 - Βελτιστοποίηση : 87

6.1 Συσκευή Εκτέλεσης 88

6.2 Στόχοι , Περιορισμοί και Εύρος μεταβολής Μεταβλητών Σχεδιασμού 89

6.3 Αποτελέσματα 91

Κεφάλαιο 7 - Συμπεράσματα 95

Παράρτημα Α – Γένεση Πλέγματος: 99

Πρόγραμμα airfoil.exe *100*

Πρόγραμμα ellipsis.exe *101*

Πρόγραμμα makefr.exe *102*

Πρόγραμμα front5.exe *103*

Πρόγραμμα manage.exe *103*

Παράρτημα Β – Εξελικτικοί Αλγόριθμοι: 105

Β.1 Βασικές Διεργασίες 105

Β.2 Αλγόριθμος Εκτέλεσης 106

Β.3 Πρόσθετοι Τελεστές 107

Β.4 Βέλτιστες λύσεις 107

Β.4 Εφαρμογή 108

Βιβλιογραφία: 109

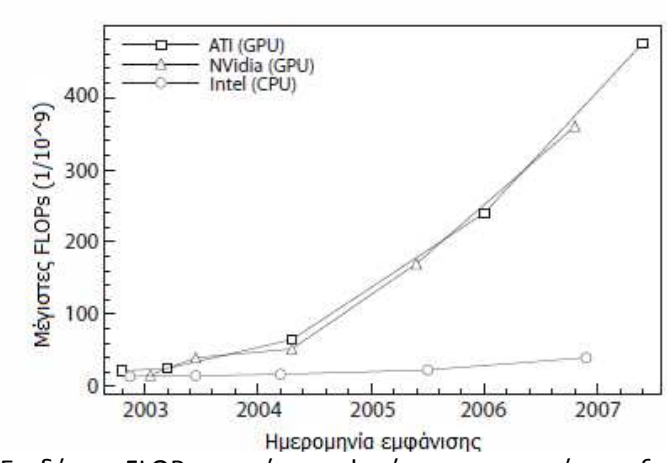
Κεφάλαιο 1ο
Εισαγωγή

1.1 Γενικά Περί Υπολογιστικών Συστημάτων

Η υπολογιστική ισχύς των κοινών εμπορικών κεντρικών επεξεργαστών (CPU) των κυρίαρχων κατασκευαστών INTEL και AMD συνεχίζει διαρκώς να αυξάνεται. Στη δεκαετία του 80 και του 90, ο αριθμός των μέγιστων πράξεων δεκαδικών αριθμών ανά δευτερόλεπτο (FLOPs) που ήταν διαθέσιμοι στους υπολογιστές αυξανόταν ως αποτέλεσμα της εκθετικής αύξησης των τρανζίστορ του επεξεργαστή, σύμφωνα με το νόμο του Moore **[18]**, ενώ παράλληλα ακολουθούσε και η αύξηση της ταχύτητας των ρολογιών τους. Όμως, τα τελευταία χρόνια, αυτή η αύξηση της ταχύτητας έχει περιοριστεί και περαιτέρω βελτιώσεις έρχονται πλέον μέσω των παράλληλων υπολογισμών. Αυτή η παραλληλία μπορεί να επιτευχθεί είτε εξωτερικά με συστοιχίες συνεργαζόμενων υπολογιστών (clusters) ή ακόμα και με σύζευξη πολλών clusters που επιτρέπουν το grid computing, είτε εσωτερικά, με την ανάπτυξη πολυπύρηνων κεντρικών επεξεργαστών.

Αν και η τάση αυτή έχει οδηγήσει στις μέρες μας στη δημιουργία μέχρι και 8πύρηνων επεξεργαστών (14 Giga FLOPs) **[23]**, εντούτοις, σε κάθε σύγχρονο προσωπικό υπολογιστή υπάρχει ένας εγγενώς παράλληλος επεξεργαστής, η κάρτα γραφικών (GPU), με επιδόσεις μακράν καλύτερες από αυτές του κεντρικού επεξεργαστή (1160 Giga FLOPs) **[24]**.

Στο παρακάτω γενικό σχήμα, μπορούμε να δούμε τις επιδόσεις σχετικά με την εκτέλεση πράξεων δεκαδικών αριθμών ανά μονάδα χρόνου για τρείς ξεχωριστές τεχνολογίες, για κάρτες γραφικών της εταιρίας ATI και NVIDIA και για κεντρικούς επεξεργαστές INTEL.



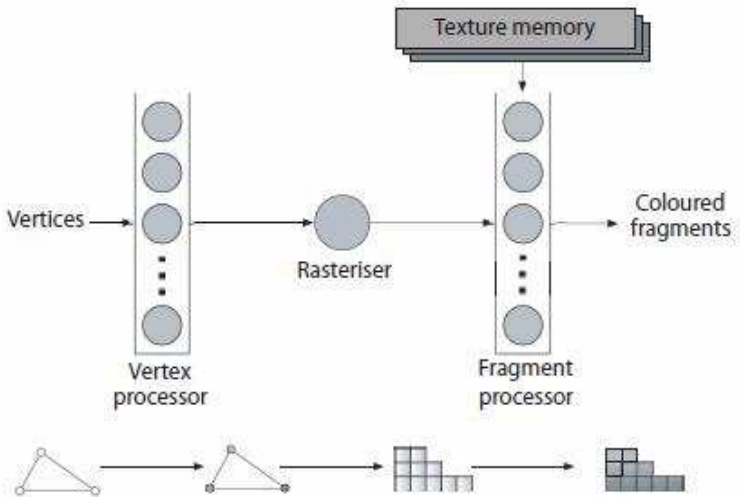
Σχ. 1.1 – Επιδόσεις FLOPs καρτών γραφικών και κεντρικών επεξεργαστών **[10]**.

Επιπλέον, στις μηχανολογικές εφαρμογές και ιδιαίτερα του τομέα της υπολογιστικής ρευστοδυναμικής όπως είναι η επίλυση των εξισώσεων ροής, η ταχύτητα εκτέλεσης των εφαρμογών και η ακρίβεια των αποτελεσμάτων τους, αποτελούν κομβικούς παράγοντες. Συγκεκριμένα, μια μείωση στο στοιχειώδες χρόνο εκτέλεσης κατά τον υπολογισμό ενός κόμβου σε μια επανάληψη, δίνει τη δυνατότητα στον μηχανικό, είτε να χρησιμοποιήσει πιο πυκνά υπολογιστικά πλέγματα για να αυξήσει την ακρίβεια των αποτελεσμάτων στον ίδιο χρόνο, είτε να χρησιμοποιήσει το ίδιο πλέγμα και να επιτύχει ταχύτερη αποκομιδή αποτελεσμάτων.

Τέτοιες εφαρμογές, μέχρι σήμερα, λόγου του υψηλού υπολογιστικού τους κόστους, εκτελούνταν παράλληλα στους κεντρικούς επεξεργαστές συστοιχιών υπολογιστών με σκοπό την ολοκλήρωση τους, εντός αποδεκτού χρόνου. Ο περιορισμός όμως στην αύξηση της ταχύτητας των κεντρικών επεξεργαστών που ήδη αναφέρθηκε, σε συνδυασμό με τις όλο και μεγαλύτερες απαιτήσεις για υπολογιστική ισχύ οδήγησε στην εξερεύνηση της κάρτας γραφικών ως πιθανού μέσου εκτέλεσης των εφαρμογών, της οποίας η υπολογιστική ισχύ ήταν τουλάχιστον μιας τάξης μεγέθους μεγαλύτερη σε σχέση με του κεντρικού επεξεργαστή.

1.2 Η Κάρτα Γραφικών

Η κάρτα γραφικών είναι μια συσκευή που χρησιμοποιείται για την επεξεργασία γραφικών, δηλαδή εικόνων. Συγκεκριμένα είναι σχεδιασμένη να λαμβάνει το πολυγωνικό μοντέλο μιας τριδιάστατης εικόνας από τον κεντρικό επεξεργαστή, να την επεξεργάζεται εκτελώντας υπολογισμούς και να την μετατρέπει σε εικόνα στην οθόνη του υπολογιστή, μέσω μιας διαδικασίας αποτελούμενης από επιμέρους διακριτά βήματα, που ονομάζεται ροή πληροφοριών γραφικών (graphics pipeline) και η οποία φαίνεται στο παρακάτω σχήμα.



Σχ. 1.2 – Ροή πληροφοριών γραφικών **[10]**.

Αναλυτικά η διαδικασία έχει ως εξής: Ο επεξεργαστής κορυφών (vertex processor) **[19]** λαμβάνει τα στοιχειώδη πολύγωνα και τα επεξεργάζεται αλλάζοντας ιδιότητες τους όπως οι θέσεις/συντεταγμένες τους και το χρώμα τους. Στη συνέχεια, ο επεξεργαστής rasteriser **[20]** λαμβάνει τη πληροφορία υπό μορφή πινάκων/ψηφιακών δεδομένων και την μετατρέπει σε κουκίδες, δηλαδή pixels οθόνης. Τέλος, ο επεξεργαστής των στοιχειωδών κουκίδων (fragment processor) **[21]** χρωματίζει τις κουκίδες και τους περνά τις τελευταίες πληροφορίες από την texture θέση μνήμης και προβάλλει τα pixel στην οθόνη του υπολογιστή. Κύριο χαρακτηριστικό είναι ότι κάθε στοιχειώδες πολύγωνο, το οποίο συνθέτει την τριδιάστατη εικόνα, επεξεργάζεται από ξεχωριστή ομάδα επεξεργαστών για να μετασχηματιστεί στο στοιχειώδες pixel στην οθόνη. Κατανοητό είναι πλέον, ότι το κλειδί στην διαδικασία είναι το γεγονός ότι η κάρτα γραφικών έχοντας μεγάλο πλήθος επεξεργαστών έχει τη δυνατότητα να επεξεργάζεται κάθε ένα τμήμα της εικόνας ανεξάρτητα και παράλληλα με όλα τα άλλα.

Η διαδικασία επεξεργασίας μίας εικόνας παρουσιάζει πολλές ομοιότητες ως προς την διεκπεραίωση της σε σχέση με την εκτέλεση ενός προγράμματος επίλυσης των εξισώσεων ροής το οποίο περιλαμβάνει επαναλαμβανόμενες εκτελέσεις των ίδιων εντολών σε διαφορετικά και ανεξάρτητα δεδομένα (κόμβους πλέγματος) αντίστοιχες με τις διεργασίες που γίνονται από τη ροή πληροφοριών γραφικών για το μετασχηματισμό στοιχειωδών πολυγώνων σε στοιχειώδη pixels στην οθόνη.

Επομένως, αν ο προγραμματιστής-μηχανικός είχε τη δυνατότητα να «ξεγελάσει» την κάρτα γραφικών, δηλαδή αντί για πληροφορίες χρώματος, θέσης και άλλων χαρακτηριστικών των τριγώνων να περάσει τα δεδομένα του μοντέλου του (πχ. υπολογιστικό πλέγμα) στη μνήμη της κάρτας και αντί για επεξεργασία των πολυγώνων να χρησιμοποιήσει τη graphics pipeline για εκτέλεση των πράξεων του υπολογιστικού του μοντέλου του (πχ. επίλυση ροής), θα επιτύγχανε να ξεκλειδώσει τον έλεγχο της κάρτας γραφικών και να την μετατρέψει από συσκευή επεξεργασίας εικόνας σε μονάδα επεξεργασίας γενικού σκοπού (General Purpose GPU).

Μέχρι πρόσφατα, οι δίαυλοι ροής πληροφοριών της κάρτας γραφικών εκτελούσαν τη λειτουργία τους και δεν υπήρχε η δυνατότητα επέμβασης σε αυτές. Για το σκοπό αυτό αναπτύχθηκαν αρκετά πακέτα και γλώσσες προγραμματισμού τα οποία ξεκλείδωναν την πρόσβαση σε αυτές και επέτρεψαν τη γενική χρήση της κάρτας γραφικών. Εκτός από τη δυνατότητα που δόθηκε για επαναπρογραμματισμό των εφαρμογών έτσι ώστε οι κλήσεις τους να γίνονται απευθείας στις βιβλιοθήκες της κάρτας όπως οι OpenGL και DirectX και η οποία είχε ως αποτέλεσμα τη δημιουργία πηγαίου κώδικα δυσκολονόητου, δημιουργήθηκαν γλώσσες προγραμματισμού χαμηλού και υψηλού επιπέδου με τις οποίες ο προγραμματισμός της κάρτας γίνεται εύκολη διαδικασία. Τέτοιες είναι οι Sh, Brook, CUDA, CTM, RapidMind, Sequoia **[10].**

Συγκεκριμένα, οι χαμηλού επιπέδου γλώσσες, επιτρέπουν πρόσβαση στις λεπτομέρειες της κάρτας γραφικών χωρίς να γίνεται χρήση της πολύπλοκης ορολογίας της κάρτας. Τέτοια γλώσσα προγραμματισμού είναι και η CUDA, ανεπτυγμένη από την NVIDIA για χρήση στις κάρτες γραφικών της εταιρίας της, η οποία αποτελεί προέκταση της γνωστής γλώσσας C/C++ και η οποία χρησιμοποιήθηκε για τον προγραμματισμό των εφαρμογών στο πλαίσιο της διπλωματικής. Από την άλλη, οι γλώσσες υψηλού επιπέδου, όπως η BrookGPU, αποκρύπτουν τελείως την κάρτα γραφικών από το χρήστη.

1.3 Χαρακτηριστικά της Κάρτας Γραφικών

Όπως αναφέρθηκε στην προηγούμενη ενότητα, ο πρώτος κομβικός παράγοντας που μας ενδιαφέρει σχετικά με την εκτέλεση εφαρμογών υπολογιστικής ρευστοδυναμικής σε κάρτες γραφικών, είναι ο χρόνος εκτέλεσης και μελετώντας το στοιχειώδη τρόπο λειτουργίας της κάρτας συμπεράναμε ότι αυτή η νέα τεχνολογία οδηγεί σε δραματική μείωση του.

Ο δεύτερος παράγοντας που μας απασχολεί είναι η ακρίβεια των αποτελεσμάτων που είναι και το πιο σημαντικό μειονέκτημα της κάρτας γραφικών.

Η χρησιμοποίηση της κάρτας γραφικών, η οποία εκ δημιουργίας της προοριζόταν για επεξεργασία εικόνας, στην οποία η ακρίβεια δεν είναι τόσο σημαντικός παράγοντας, αλλά ο χρόνος εκτέλεσης μιας και αυτό που προέχει είναι η άμεση επεξεργασία μεγάλου πλήθους δεδομένων, για εφαρμογές ευρείας χρήσεως, εισάγει σημαντική μείωση της ακρίβειας των αποτελεσμάτων. Σε μηχανολογικές και γενικότερες αριθμητικές εφαρμογές όπου για να προκύψουν ακριβή αποτελέσματα χρειάζεται μεγάλη ακρίβεια στις πράξεις, η εκτέλεση τους στην κάρτα γραφικών μπορεί να αποτελέσει την Αχίλλειο πτέρνα της διαδικασίας.

Όπως θα δούμε μάλιστα παρακάτω, η γλώσσα προγραμματισμού CUDA που μελετάμε, προσφέρει εναλλακτικές συναρτήσεις που η εκτέλεση τους προσφέρει μεγαλύτερη ακρίβεια και μεγαλύτερο σφάλμα σε σχέση με τις συνηθισμένες.

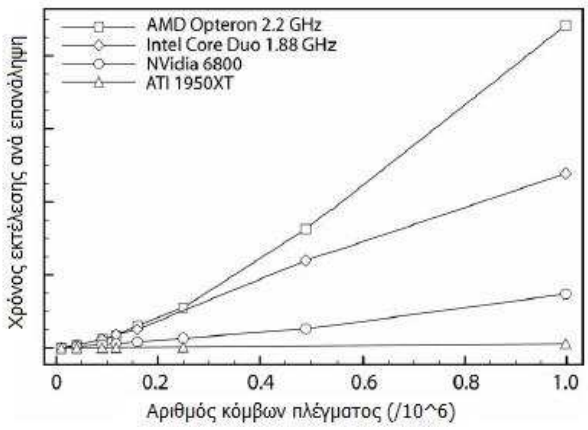
Στα παραπάνω συνεισφέρει το γεγονός ότι οι κάρτες γραφικών μέχρι πρόσφατα εκτελούσαν υπολογισμούς μόνο με μονής ακρίβειας αριθμούς. Και ενώ σε μερικές εφαρμογές που η απόλυτη ακρίβεια δεν είναι το ζητούμενο και η ύπαρξη μονής ακρίβειας πράξεων συνεπάγεται με μεγαλύτερη ταχύτητα (4 φορές μεγαλύτερη από χρήση διπλής ακρίβειας), σε εφαρμογές υπολογιστικής ρευστοδυναμικής τα αποτελέσματα δεν μπορούν να θεωρηθούν αξιόπιστα αν είναι μονής ακρίβειας.

Όσον αφορά όμως τις κάρτες της εταιρίας NVIDIA που εξετάζουμε, πρόσφατα δημιουργήθηκαν οι κάρτες γραφικών της σειράς GT200 οι οποίες δίνουν πλέον την

δυνατότητα στο χρήστη για εκτέλεση πράξεων διπλής ακρίβειας. Σε επόμενη ενότητα υπάρχει λεπτομερής αναφορά.

Επίσης σημαντικό, είναι να αναφερθεί ότι με την εξέλιξη της τεχνολογίας, οι τελευταίες γενιές καρτών γραφικών που υποστηρίζονται από τη βιβλιοθήκη DirectX 10, έχουν τους επιμέρους επεξεργαστές της ροής πληροφοριών γραφικών (vertex, rasteriser,fragment) ενοποιημένους σε έναν ενιαίο, ονομαζόμενο shader. Η αρχιτεκτονική έτσι γίνεται πιο απλή, τα χρησιμοποιούμενα τρανζίστορ γίνονται πιο αποδοτικά μιας και δεν υπάρχει πλέον μεταφορά δεδομένων από τον έναν υποεπεξεραστή στον άλλον, και πλέον ο κάθε shader μπορεί να γράψει σε οποιαδήποτε θέση μνήμης και όχι μονό σε αυτήν του πολυγώνου που επεξεργάζεται. Με αυτήν τη μετατροπή, μια μοντέρνα κάρτα γραφικών πλέον γίνεται υπολογιστική μονάδα της μορφής SIMD (Single Instruction Multiple Data) με την έννοια ότι μπορεί να επεξεργαστεί με την ίδια εντολή, διαφορετικά εισερχόμενα δεδομένα, σε αντίθεση με τον κεντρικό επεξεργαστή που είναι της μορφής MIMD (Multiple Instruction Multiple Data) δηλαδή κάθε πυρήνας εκτελεί διαφορετική λειτουργία σε διαφορετικά δεδομένα.

Επιπλέον, πρέπει να τονίσουμε ότι επειδή το υπολογιστικό κόστος αυξάνεται όσο αυξάνεται και η μεταφορά δεδομένων από και προς την κάρτα, όσο μεγαλύτερη είναι η διάσταση του προβλήματος προς επίλυση και επομένως για την ίδια μεταφορά δεδομένων έχουμε εκτέλεση περισσότερων πράξεων, τόσο μεγαλύτερη επιτάχυνση θα παρατηρηθεί. Χαρακτηριστικά παραθέτουμε το παρακάτω σχήμα.



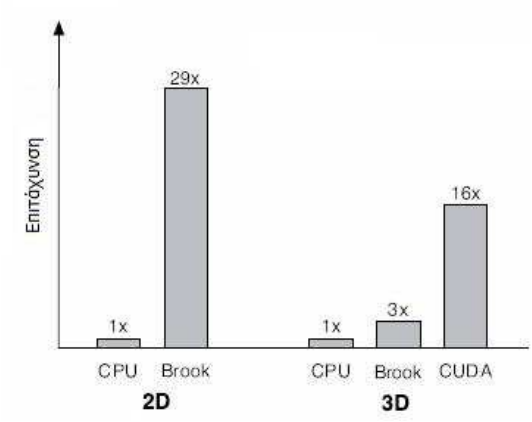
Σχ. 1.3 – Χρόνοι εκτέλεσης για διάφορες μονάδες και μεγέθη πλέγματος **[6].**

Τέλος, θα πρέπει να σημειώσουμε ένα ακόμα από τα πλεονεκτήματα της νέας τεχνολογίας. Οι κάρτες γραφικών είναι συσκευές ευρέως γνωστές, που ήδη υπάρχουν σε κάθε υπολογιστή. Η αξία μιας κάρτας γραφικών, με ικανοποιητική απόδοση, που διαθέτει τη δυνατότητα ευρείας χρήσης δεν ξεπερνά τα 300 ευρώ, σε αντίθεση με τα πολύπλοκα υπολογιστικά συστήματα που δεν είναι προσβάσιμα, λόγο υψηλού κόστους, στο ευρύ κοινό.

1.4 Πρόσφατα Αποτελέσματα στην Υπολογιστική Ρευστοδυναμική μέσω GPU

Όπως έγινε κατανοητό, η κάρτα γραφικών μπορεί να χρησιμοποιηθεί πλέον και ως μέσο εκτέλεσης σε εφαρμογές οι οποίες επιδέχονται παραλληλισμό των εσωτερικών τους διεργασιών, γεγονός που την καθιστά ιδανική για προβλήματα υπολογιστικής ρευστοδυναμικής, συνοριακών συνθηκών και αριθμητικής επίλυσης.

Οι πρώτες δοκιμές γίνονται το 2003 σε εφαρμογή επίλυσης της εξίσωσης Poison από τους N.Goodnight και G.Lewin **[8]** και η επιτευχθείσα επιτάχυνση ήταν της τάξης του 1500%. Το 2006 ο T.R.Hagen **[9]** με εφαρμογή, επιλύτη ροής Euler σε 2 διαστάσεις, σημειώνει επιτάχυνση από 10 έως 25 φορές ανάλογα με το μέγεθος του πλέγματος, ενώ για 3 διαστάσεις σημειώνει επιτάχυνση 10 έως 14 φορές. Τα επόμενα χρόνια οι T.Brandvik και G.Pullan **[10]** μελετούν αναλυτικά την επίλυση των Euler σε 2 και 3 διαστάσεις με επίλυση της ροής σε αεροτομές. Μάλιστα μας δίνουν συγκριτικά αποτελέσματα επιτάχυνσης για 2 διαφορετικές γλώσσες προγραμματισμού όπως φαίνεται στο παρακάτω σχήμα,



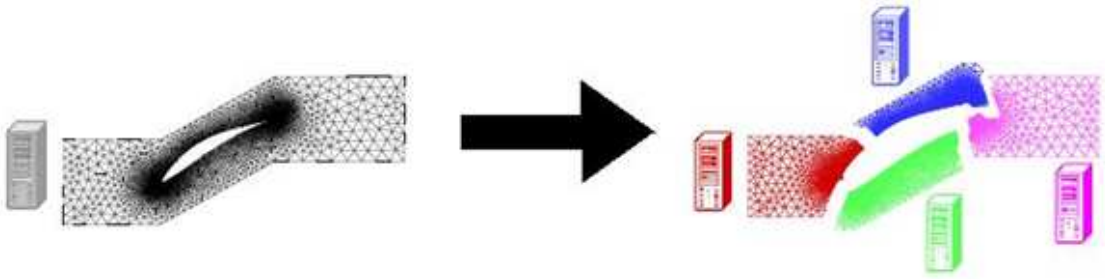
Σχ. 1.4 – Σύγκριση επιτάχυνσης για πλέγμα 300.000 κόμβων **[10].**

ενώ τα τελευταία χρόνια ακολουθούν και άλλες μελέτες σε επιλύτες ροής για διδιάστατα, τριδιάστατα, δομημένα και μη-δομημένα πλέγματα γύρω από αεροτομές, πυραύλους και οχήματα για υπερηχητικές ροές με επιταχύνσεις που φτάνουν μέχρι και τις 40 φορές υπό περιπτώσεις.

1.5 Σύγχρονες Ερευνητικές Δραστηριότητες στο Εργαστήριο Θερμικών Στροβιλομηχανών του ΕΜΠ

Για την υλοποίηση της παρούσας διπλωματικής, χρησιμοποιήθηκε έτοιμο λογισμικό επίλυσης των εξισώσεων Navier-Stokes. Ο επιλύτης ροής για εκτέλεση σε κάρτες γραφικών είναι αποτέλεσμα πρόσφατης μελέτης του εργαστηρίου.

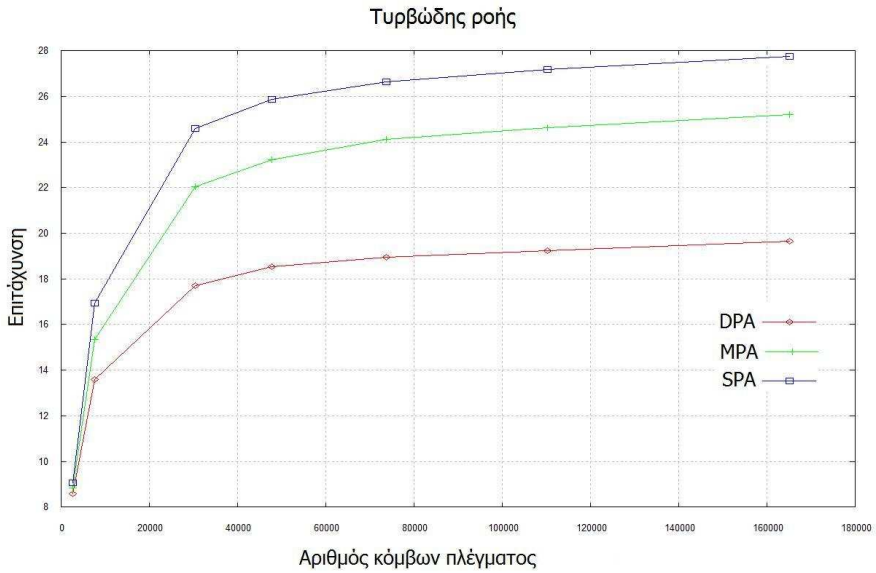
Συγκεκριμένα, το εργαστήριο διαθέτει πείρα ετών σε υπολογιστικές εφαρμογές πολλαπλών υποχωρίων (grid computing) κατά την οποίες το συνολικό πλέγμα ενός προβλήματος αντί να εκτελείται σε μία υπολογιστική μονάδα, διασπάται σε επιμέρους υποχωρία κάθε ένα από τα οποία επεξεργάζεται σε διαφορετική μονάδα επεξεργασίας. Χαρακτηριστικά παρατίθεται το παρακάτω σχήμα:

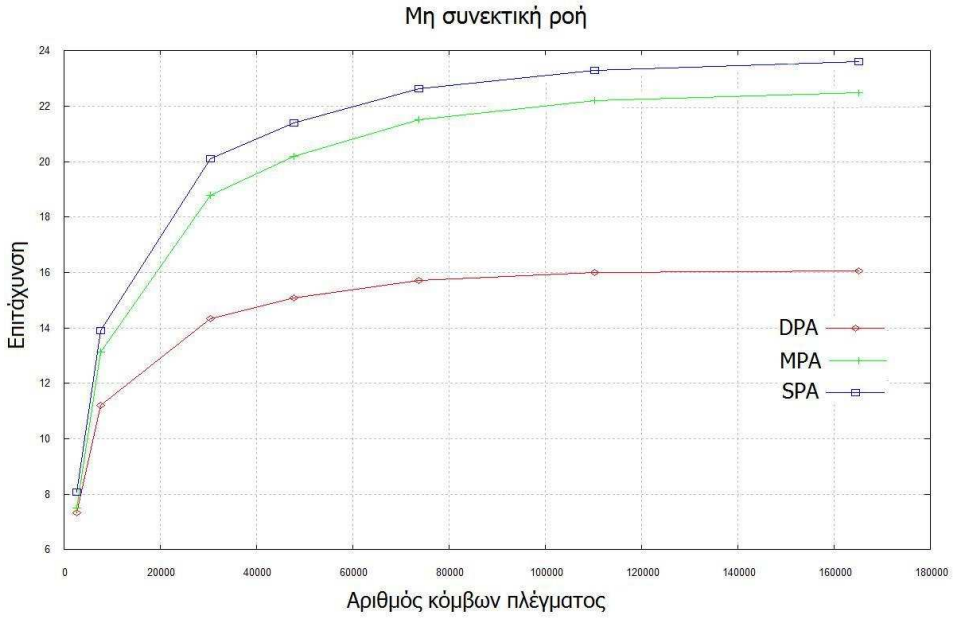


Σχ. 1.5 – Grid computing **[11].**

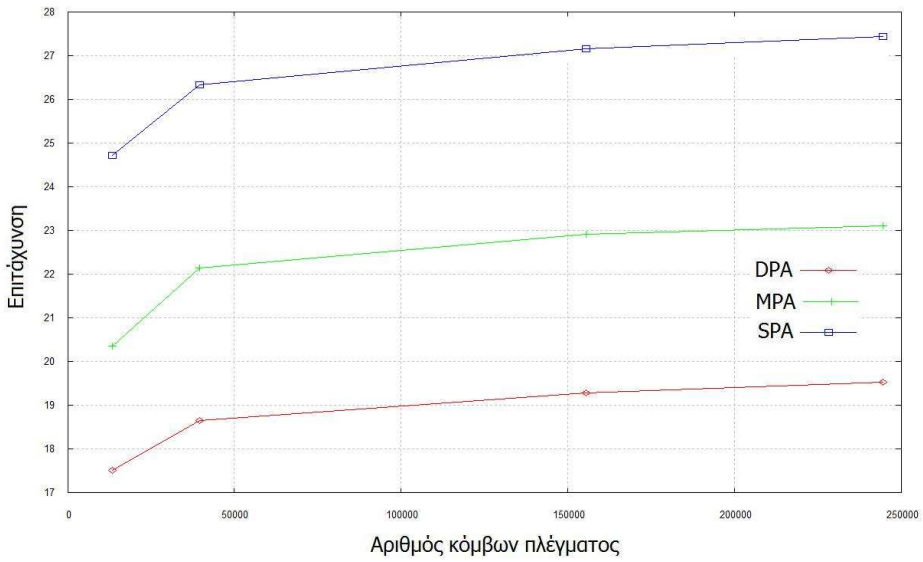
Λόγω της ευρείας εξέλιξης της κάρτας γραφικών, εδώ και ένα χρόνο πλέον, το εργαστήριο έχει αρχίσει να χρησιμοποιεί τις κάρτες γραφικών (NVIDIA GTX280 και GTX285) για την επίλυση τέτοιων προβλημάτων. Αναλυτικά, οι υπάρχοντες κώδικες επίλυσης των εξισώσεων Navier-Stokes και Euler σε δυο και τρείς διαστάσεις, προγραμματισμένοι σε Fortran για εκτέλεση σε κεντρικούς επεξεργαστές, με προσεκτική αναδόμηση που αποσκοπεί στην εκμετάλλευση των δυνατοτήτων της νέας τεχνολογίας, επαναπρογραμματίζονται σε C και CUDA για εκτέλεση στην κάρτα γραφικών.

Ενδεικτικά αποτελέσματα της έρευνας παρουσιάζονται στις εικόνες που ακολουθούν:





Σχ. 1.6 α,β – Επιδόσεις για επίλυση σε δύο διαστάσεις σε αεροτομή για μη συνεκτική και τυρβώδης ροή **[5]**.(DPA=Πράξεις διπλής ακρίβειας στο σύνολο του κώδικα, SPA=Πράξεις μονής ακρίβειας στο σύνολο του κώδικα, MPA=Πράξεις «μικτής» ακρίβειας, δηλαδή διπλής ακρίβειας ως προς τον υπολογισμό των υπολοίπων των εξισώσεων και απλής ακρίβειας ως προς τον υπολογισμό του αριστερού μέλους των εξισώσεων)[5]



Σχ. 1.7 – Επιδόσεις για επίλυση σε τρείς διαστάσεις για μη συνεκτική ροή γύρω από πολιτικό αεροσκάφος με $M_\infty = 0.7$ και $\alpha_\infty = 0.0$ **[5].**

[5] Οι κώδικες DPA και MPA οδηγούν ακριβώς στην ίδια λύση με τον κώδικα που τρέχει στη CPU, με τον MPA να είναι ταχύτερος. Ο κώδικας SPA οδηγεί σε μειωμένη ακρίβεια και πρακτικά δεν θα εξυπηρετούσε το σκοπό της εργασίας. Ο SPA αντανακλά το τι συνέβαινε πριν περίπου ένα έτος, όταν οι κάρτες γραφικών περιορίζονταν σε SPA υπολογισμούς. Πλέον, ευτυχώς, οι κάρτες γραφικών δίνουν την επιθυμητή ακρίβεια δεύτερης τάξης

1.6 Περιεχόμενα και Δομή της Εργασίας

Γενικότερα όμως, παρόλο που οι κάρτες γραφικών έχουν χρησιμοποιηθεί σε πάρα πολλές εφαρμογές στον μαθηματικό και μηχανολογικό τομέα, υπάρχει ακόμα μεγάλη απόσταση από το επιθυμητό αποτέλεσμα. Δηλαδή, από το σημείο που θα χρησιμοποιηθεί η κάρτα γραφικών σε αληθινές εφαρμογές ευρείας κλίμακας όπως σε κώδικες που απαιτούν ημέρες ή και βδομάδες εκτέλεσης για την ολοκλήρωση μιας και μόνο εξομοίωσης.

Στην παρούσα διπλωματική εργασία, επιλύεται στην κάρτα γραφικών η διδιάστατη ροή γύρω από αεροτομή και επαναπρογραμματίζεται ο κώδικας μοντέλου προκαταρκτικού σχεδιασμού αεροσκάφους με βάση τη γλώσσα προγραμματισμού CUDA. Κάνοντας ένα βήμα παραπάνω, συνδυάζονται αυτές οι δυο εφαρμογές, υπό την επίδραση μοντέλου βελτιστοποίησης με σκοπό τη μελέτη της επίδρασης της νέας τεχνολογίας στο χρόνο εκτέλεσης μιας πιο σύνθετης εφαρμογής.

Αναλυτικά στα επτά κεφάλαια αυτής της εργασίας παρουσιάζονται:

- ⇨Η αρχιτεκτονική των καρτών γραφικών της εταιρείας NVIDIA.
- ⇨Η γλώσσα προγραμματισμού CUDA.
- ⇨Η επίλυση της ροής γύρω από την αεροτομή.
- ⇨Το μοντέλο του προκαταρκτικού σχεδιασμού του αεροσκάφους.
- ⇨Τα αποτελέσματα της βελτιστοποίησης του μοντέλου.
- ⇨Τα γενικά συμπεράσματα.

Τέλος σε δυο υπομνήματα παρουσιάζονται:

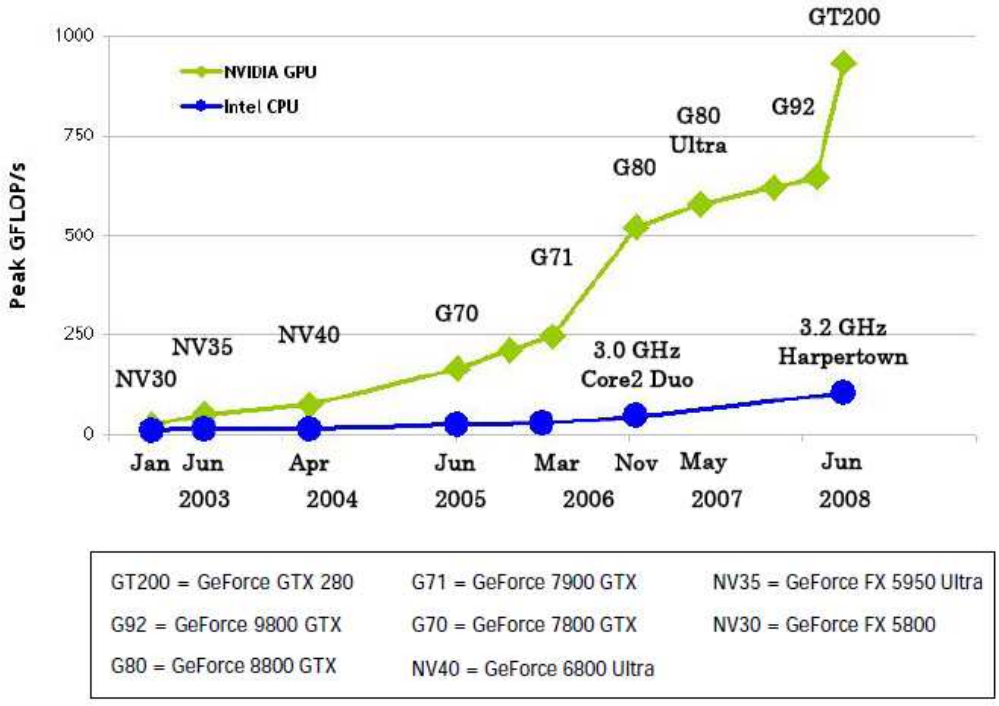
- ⇨Η νέα μέθοδος γένεσης πλέγματος
- ⇨Και το θεωρητικό υπόβαθρο βελτιστοποίησης μέσω εξελικτικών αλγόριθμων.

Κεφάλαιο 2ο
Αρχιτεκτονική CUDA

Εισαγωγή:

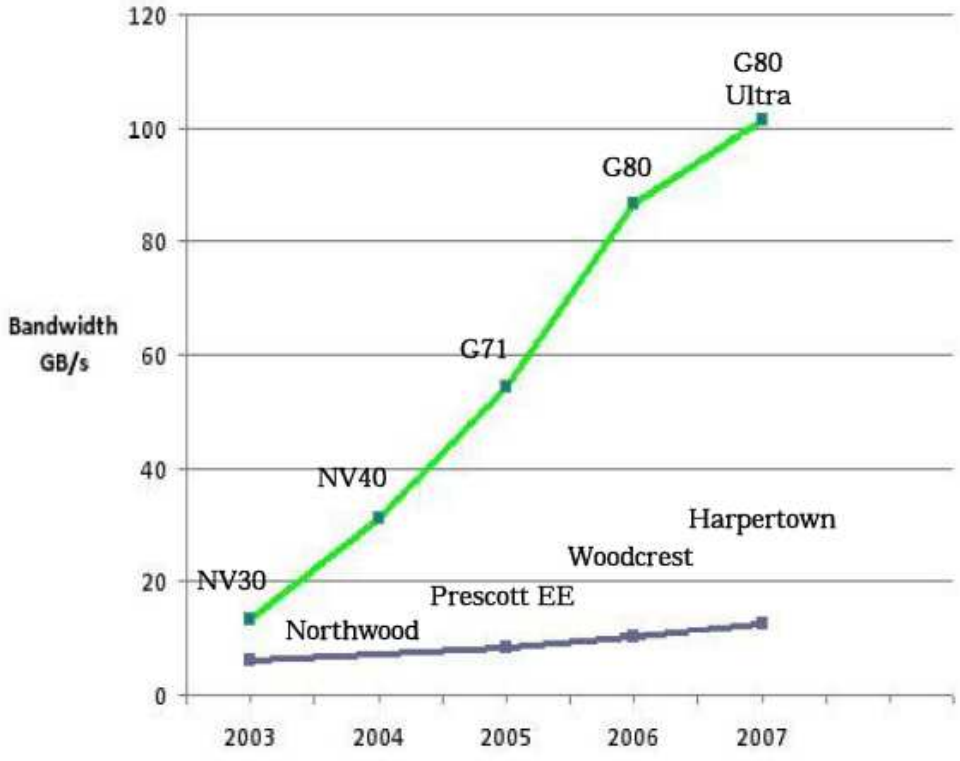
Όπως αναφέραμε, οι κάρτες γραφικών[6] μέσα από τη ραγδαία εξέλιξη τους και την αύξηση της υπολογιστικής τους δύναμης σε σχέση με τους κεντρικούς επεξεργαστές, απέκτησαν χρήση σε εφαρμογές ευρείας κλίμακας παραλλήλων υπολογισμών.

Αυτή τη χαρακτηριστική τους εξέλιξη μπορούμε να τη δούμε στα 2 παρακάτω σχήματα, όπου συγκρίνεται η εξέλιξη των επιδόσεων των καρτών γραφικών (GPU Graphics Processing Unit) της εταιρείας NVIDIA που μελετάμε, με τους κεντρικούς επεξεργαστές (CPU Central Processing Unit) στην πάροδο του χρόνου:



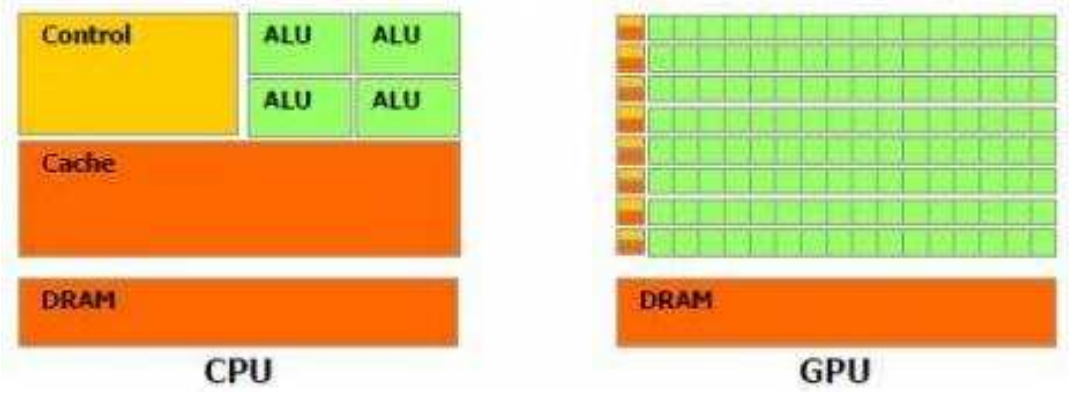
Σχ. 2.1 – Υπολογιστικές δυνατότητες (Floating-Point Operations per Second) **[12]**.

[6] Οι κάρτες γραφικών στις οποίες αναφέρεται η παρούσα διπλωματική εργασία και οι οποίες κατασκευάζονται με βάση την τεχνολογία CUDA, είναι οι κάρτες της εταιρίας Nvidia. Μάλιστα, η συγκεκριμένη κάρτα η οποία χρησιμοποιήθηκε είναι η GTX285 και αναλύεται στην ενότητα 2.8.1



Σχ. 2.2 – Μεταφορά δεδομένων στη μνήμη (Memory Bandwidth) **[12]**.

Ο κύριος λόγος πίσω από αυτήν τη ραγδαία εξέλιξη είναι ότι η κάρτα γραφικών είναι ένα σύστημα εξειδικευμένο σε υπολογιστικά, απαιτητικούς και παράλληλους υπολογισμούς (δηλαδή ότι απαιτεί και η επεξεργασία γραφικών) και αυτό οφείλεται στην αρχιτεκτονική της ίδιας της κάρτας η οποία, σε αντίθεση με τους κεντρικούς επεξεργαστές των υπολογιστών, είναι έτσι σχεδιασμένη ώστε να αφιερώνει περισσότερα τρανζίστορ στην επεξεργασία δεδομένων παρά στην αποθήκευση δεδομένων και στον έλεγχο της ροής. Το σχήμα που ακολουθεί δείχνει χαρακτηριστικά αυτήν την διαφορά:



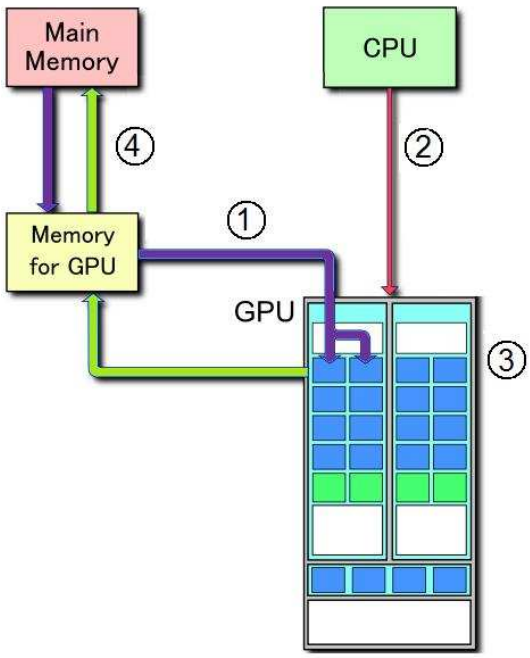
Σχ. 2.3 – Βασική δομή αρχιτεκτονικής **[12].**

Σε αυτό το κεφάλαιο, θα αναπτυχθεί η βασική δομής αυτής της αρχιτεκτονικής και θα αναλυθούν οι βασικές λειτουργίες που γίνονται από την κάρτα γραφικών και με τις οποίες ένα σύνθετο υπολογιστικό πρόβλημα διασπάται σε επιμέρους υποπροβλήματα, καθένα από τα οποία στη συνέχεια εκτελείται ξεχωριστά. Τέλος, θα δοθούν λεπτομερή στοιχεία για τα χαρακτηριστικά του κεντρικού επεξεργαστή και της κάρτας γραφικών που χρησιμοποιήθηκαν για την επίλυση του αλγορίθμου στο πλαίσιο αυτής της διπλωματικής.

Για λόγους συντομίας και λιτότητας στις εκφράσεις, στη συνέχεια του κειμένου, ο κεντρικός επεξεργαστής του υπολογιστή θα αναφέρεται ως **επεξεργαστής** ή με την αγγλική λέξη **host,** καθώς τον μεταχειριζόμαστε ως τη μητρική συσκευή από την οποία ξεκινάει η εκτέλεση του προγράμματος μας και από την οποία θα μεταφέρονται δεδομένα στις επιμέρους συσκευές προς επεξεργασία και οι οποίες είναι οι κάρτες γραφικών (όπως θα δούμε οι κάρτες που μπορεί να διαθέτει ένα υπολογιστικό σύστημα μπορούν να είναι περισσότερες από μια), στις οποίες θα αναφερόμαστε από εδώ και πέρα με την λέξη **συσκευή** ή με την αγγλική λέξη **device.**

2.1Βασικές Λειτουργίες της Κάρτας Γραφικών

Στο παρακάτω σχήμα φαίνονται περιληπτικά οι βασικές λειτουργίες που εκτελεί η κάρτα γραφικών για να επεξεργαστεί μια ομάδα δεδομένων και να επιστρέψει τα αποτελέσματα:



Σχ. 2.4 – Γενική λειτουργία Κάρτας Γραφικών **[16].**

Όπου:

①(Μοβ) : Αντιγραφή δεδομένων από τη μνήμη του συστήματος στη μνήμη της κάρτας γραφικών.
②(Κόκκινο) : Εντολή από τη CPU για εκτέλεση στην GPU.

③(GPU) : Παράλληλη εκτέλεση σε κάθε πυρήνα της κάρτας.

④(Πράσινο) : Αντιγραφή αποτελεσμάτων από τους πυρήνες της κάρτας στη μνήμη της κάρτας και από εκεί στη μνήμη του συστήματος.

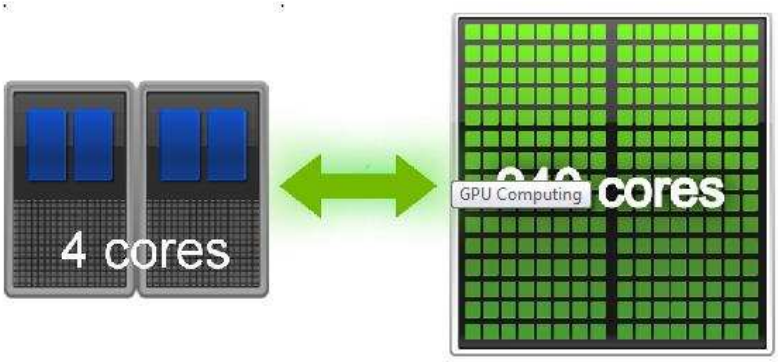
2.2 *Βασικές Έννοιες και Παράμετροι*

Ένα υπολογιστικό πρόβλημα για να μπορέσει να προγραμματιστεί πάνω στη συσκευή πρέπει να αποτελείται από τουλάχιστον μια ακολουθία από εντολές οι οποίες εκτελούνται σειριακά, η μια μετά την άλλη. Η ακολουθία εκτελείται πάρα πολλές φορές κατά τη διάρκεια του προγράμματος αλλά κάθε φορά για διαφορετικά σετ δεδομένων εισόδου ώστε να προκύπτουν αντίστοιχα διαφορετικά σετ αποτελεσμάτων. Αυτή η ακολουθία ενεργειών, από εδώ και πέρα θα αναφέρεται ως **νήμα** ή με την αγγλική λέξη **thread**[7]. Επίσης κάθε υπορουτίνα του αρχικού προγράμματος η οποία θα σταλεί στην συσκευή και η οποία αναλύεται σε πολλά υπολογιστικά threads που θα εκτελούνται ανεξάρτητα ονομάζεται **kernel** ή **πυρήνας εκτέλεσης**.

Το κύριο χαρακτηριστικό της συσκευής και το οποίο προσπαθούμε να εκμεταλλευτούμε είναι η δυνατότητα να αποστέλλονται τα νήματα στη συσκευή παράλληλα (μιας και όπως είπαμε για είσοδο λαμβάνουν διαφορετικά σετ δεδομένων και το ένα είναι ανεπηρέαστο από το άλλο) επιτυγχάνοντας έτσι παράλληλη εκτέλεση τους σε αντίθεση με τη σειριακή που θα υπήρχε όταν το πρόγραμμα ήταν σχεδιασμένο για εκτέλεση στον επεξεργαστή και έτσι να καταφέρουμε να ρίξουμε το χρόνο εκτέλεσης του σε κλάσματα του αρχικού χρόνου.

Έτσι είναι εμφανές σε αυτό το σημείο πόσο σημαντικός είναι ο αριθμός των φυσικών πυρήνων (cores) της συσκευής σε σχέση με του επεξεργαστή και αναφέροντας ότι ενώ το πλήθος των πυρήνων των επεξεργαστών της τελευταίας τεχνολογίας μόλις έφτασε τους 4 έως 8, ο αντίστοιχος των τελευταίων καρτών γραφικών έχει ήδη αγγίζει τους 240 (*Σχήματα 2.5, 2.17, 2.18, 2.19*), δίνοντας έτσι μια μικρή ιδέα για ποίο λόγο οι κάρτες γραφικών έχουν τόσο μεγάλη υπολογιστική ισχύ όπως είδαμε στην εισαγωγή της διπλωματικής εργασίας έναντι των κεντρικών επεξεργαστών.

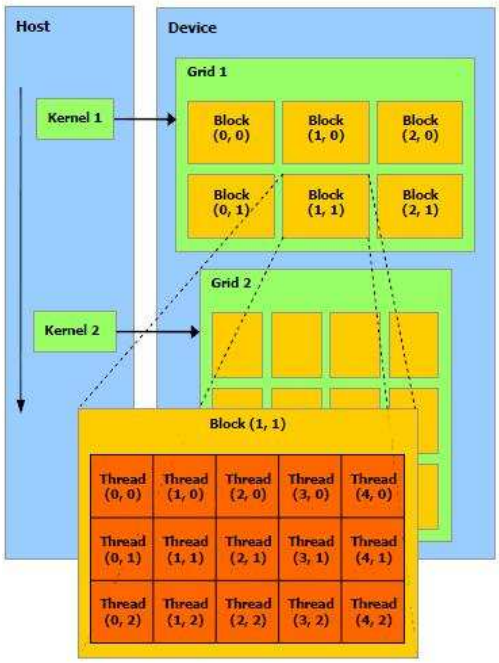
[7] Στα ελληνικά νήμα εκτέλεσης



Σχ. 2.5 – Πυρήνες Κάρτας και Επεξεργαστή **[13].**

2.3 Οργάνωση των Εκτελέσιμων Νημάτων

Το σύνολο των threads που αναφερθήκαμε παραπάνω και που εκτελούνται σε κάθε kernel δεν είναι τυχαία κατανεμημένα αλλά είναι οργανωμένα σε επιμέρους ομάδες (ή μπλοκ) τις λεγόμενες **blocks**. Αυτές, στη συνέχεια, οργανώνονται ως τμήματα του συνολικού υπολογιστικού πλέγματος ή **grid.** Αναπαράσταση της οργάνωσης αυτής βλέπουμε στο σχήμα 2.6 που ακολουθεί .



Σχ. 2.6 – Ομαδοποίηση και Οργάνωση των Threads **[12].**

Η οργάνωση στο παραπάνω πλέγμα εξυπηρετεί 2 σκοπούς :

⇨Το κάθε thread θα χαρακτηρίζεται από 2 αριθμούς που δείχνουν τη θέση του μέσα στο block που ανήκει, το κάθε block θα χαρακτηρίζεται και αυτό από 2 αριθμούς που θα δείχνουν τη θέση του μέσα στο συνολικό πλέγμα (grid). Αυτή η διάταξη των threads και η αρίθμηση τους είναι απαραίτητη ώστε το εκτελέσιμο πρόγραμμα μας να ξέρει σε κάθε στιγμή σε ποιό thread θα αναφέρεται για μεταφορά δεδομένων και για εκτέλεση .

Παράλληλα ορίζεται και ένας αριθμός, ο threadID ο οποίος δείχνει μονοσήμαντα τη θέση του thread μέσα σε block έως και 3 διαστάσεων. Αναλυτικά, για ένα μονοδιάστατο block μήκους Dx, το thread με θέση (x) θα έχει threadID ίσο με (x), για ένα διδιάστατο block μήκους Dx και πλάτος Dy, το thread με θέση (x,y) θα έχει threadID ίσο με (x + y·Dx) και τέλος για ένα τριδιάστατο block με μήκος Dx, πλάτος Dy και ύψος Dz, το thread με θέση (x,y,z) θα έχει threadID ίσο με (x + y·Dx + z·Dx·Dy). Αντίστοιχα μπορεί να οριστεί και ένας αριθμός blockID ο οποίος δείχνει μονοσήμαντα τη θέση ενός block μέσα σε grid έως και 2 διαστάσεων. Αναλυτικά, για μονοδιάστατο και διδιάστατο grid διαστάσεων (Dx) και (Dx,Dy) ο blockID παίρνει τιμές (x) και (x + y·Dx).

Το σχήμα 2.6 αναπαριστά την περίπτωση διδιάστατου grid με διδιάστατα blocks .

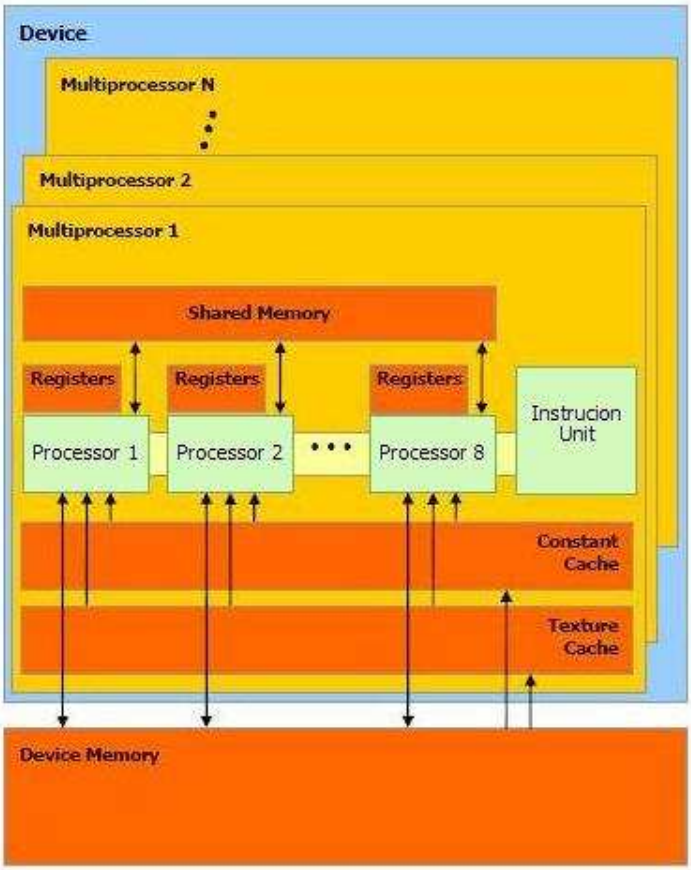
⇨Τα threads που είναι οργανωμένα σε blocks έχουν το πλεονέκτημα να χρησιμοποιούν την κοινή μνήμη του block (shared memory) η οποία προσφέρει πιο γρήγορη μετάδοση δεδομένων και έτσι συνεργαζόμενα και με χρήση της κοινής μνήμης να πετύχουν ακόμα μεγαλύτερη επιτάχυνση της επίλυσης τους (*Σχ. 2.7*). Μια ακόμα δυνατότητα της οργάνωσης των threads σε blocks είναι αυτή του συγχρονισμού τους κατά τη θέληση του χρήστη, μιας και η παράλληλη εκτέλεση διαφορετικών threads σε ένα block δεν συνεπάγεται και με τα ταυτόχρονο τερματισμό της εκτέλεσης τους, γεγονός που μπορεί να δημιουργήσει προβλήματα. Λεπτομερής ανάλυση της ασύγχρονης λειτουργίας των threads ακολουθεί στην ενότητα 3.3 .

Σημαντικό είναι ότι ο αριθμός των διαθέσιμων threads αλλά και των blocks έχει ένα μέγιστο όριο γεγονός που μπορεί να δημιουργήσει προβλήματα στην εκτέλεση της εφαρμογής μας, αν δεν υπάρχει προσεκτικός σχεδιασμός του κώδικα μας. Γενικότερα όμως το πόσα threads ή πόσα blocks θα εκτελεστούν παράλληλα εξαρτάται από τις δυνατότητες της κάρτας γραφικών που χρησιμοποιούμε (Στην ενότητα 2.8.1 παρουσιάζονται αναλυτικά οι δυνατότητες της κάρτας που χρησιμοποιήθηκε). Αν η συσκευή μας δεν έχει μεγάλες δυνατότητες παραλληλίας, το πρόγραμμα μας θα εκτελέσει αυτόματα, ομάδες blocks σειριακά, όπου τα threads του κάθε block θα εκτελούνται παράλληλα.

2.4 Πυρήνες-Επεξεργαστές της Κάρτας Γραφικών

Στο επίκεντρο της αρχιτεκτονικής της κάρτας γραφικών βρίσκονται φυσικά οι πυρήνες της και οι επεξεργαστές της. Αυτοί, χωρισμένοι σε ομάδες πολλών μικρών πολυεπεξεργαστών (Multithreaded Streaming Multiprocessors), έχουν τη δυνατότητα να παραλαμβάνουν ένα πρόβλημα και εκμεταλλευόμενοι το πλήθος τους και τις δυνατότητες τους να το διεκπεραιώνουν ταχύτατα.

Όταν μια εφαρμογή προγραμματισμένη σε CUDA φτάσει στο σημείο να κάνει κλήση ενός kernel για εκτέλεση στη συσκευή, τα blocks του συνολικού πλέγματος, αριθμούνται όπως αναφέραμε στη ενότητα 2.3 και διαμοιράζονται ανά ένα σε κάθε ομάδα πολυεπεξεργαστών που είναι διαθέσιμοι. Στη συνέχεια κάθε ξεχωριστό thread εκτελείται σε κάθε πολυεπεξεργαστή ξεχωριστά και ταυτόχρονα με όλα τα threads του ίδιου block. Από εκεί και πέρα κάθε υπολογιστικό block που τερματίζει την εκτέλεση του, παραχωρεί τη θέση του στο επόμενο που καταλαμβάνει την κενή πλέον θέση της ομάδας των επεξεργαστών.



Σχ. 2.7 – Το Μοντέλο των Επεξεργαστών **[12].**

Κάθε ομάδα πολυεπεξεργαστών (multiprocessors) όπως φαίνεται στο *Σχ. 2.7* αποτελείται από 8 πυρήνες βαθμωτών επεξεργαστών (Scalar Processor Cores), 2

ειδικές μονάδες μνήμης (Constant Cache, Texture Cache), μια κοινή μνήμη (Shared Memory) και μια μονάδα ελέγχου ροής και εντολών που είναι υπεύθυνη για τη δημιουργία, τη διαχείριση και την εκτέλεση των παράλληλων threads στη συσκευή με μηδενική επιβάρυνση στην απόδοση αυτής. Επίσης παρέχει τη δυνατότητα συγχρονισμού των threads που εκτελούνται σε αυτή αλλά αυτό σχολιάζεται στην ενότητα 3.3 .

Επίσης με βάση το *Σχ. 2.7* μπορούμε να κάνουμε μια πρώτη αναφορά στις θέσεις μνήμης που περιέχονται στην κάθε ομάδα πολυεπεξεργαστών και οι οποίες είναι:

⇨Οι τοπικοί registers (των 32-bit) του κάθε βαθμωτό επεξεργαστή.

⇨Η shared memory κοινή και προσβάσιμη και για τους 8 βαθμωτούς επεξεργαστές.

⇨Η constant cache που είναι κοινή για όλους τους βαθμωτούς επεξεργαστές, διαθέσιμη μόνο για ανάγνωση και επιταχύνει τις αναγνώσεις από την constant memory που αποτελεί τμήμα της μνήμης συσκευής (device memory).

⇨Και η texture cache διαθέσιμη μόνο για ανάγνωση και κοινή για όλους τους βαθμωτούς επεξεργαστές και επιταχύνει τις αναγνώσεις από την texture memory.

2.5 Εκτέλεση των Threads

Κατά την εκτέλεση των threads ο πολυεπεξεργαστής αντιστοιχεί καθένα από αυτά σε κάθε βαθμωτό επεξεργαστή όπου πλέον θα εκτελείται ανεξάρτητα, με τη δική του διεύθυνση εντολών και κατάσταση καταχώρησης. Για να μπορέσουν να διαχειριστούν και να οργανωθούν τα εκατοντάδες threads των διάφορων προγραμμάτων που τρέχουν κάθε στιγμή στη συσκευή, οι πολυεπεξεργαστές λειτουργούν κάτω από έναν νέο σύστημα το SIMT (Single Instruction Multiple Thread) που αποτελεί παραλλαγή του SIMD που έχει ήδη περιγραφεί στην εισαγωγή.

Η δουλειά του SIMT είναι να δημιουργεί να διαχειρίζεται να προγραμματίζει και να εκτελεί threads σε ομάδες των 32 παράλληλων threads, οι οποίες ονομάζονται **warps**[8]. Κύριο χαρακτηριστικό είναι ότι κάθε ανεξάρτητο thread που αποτελεί μέλος ενός warp, ξεκινά από την ίδια διεύθυνση του προγράμματος μαζί με τα υπόλοιπα αλλά από εκεί και πέρα είναι ελεύθερο να εκτελεστεί ανεξάρτητα.

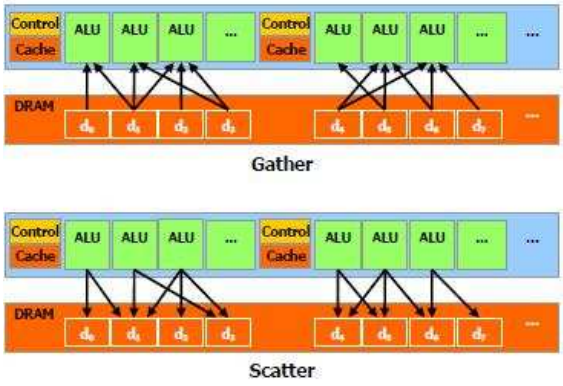
[8] Στα ελληνικά μεταφράζεται στημόνι δηλαδή τα νήματα κατά μήκος του αργαλειού που ανάμεσα τους πλέκεται το υφάδι και ονομάζονται έτσι από την «ύφανση» της πρώτης τεχνολογίας παράλληλων threads

Έτσι όταν ένα thread block στέλνεται σε μια ομάδα πολυεπεξεργαστών, η μονάδα SIMT τα διαχωρίζει σε warps και τα προγραμματίζει προς εκτέλεση. Ο διαχωρισμός αυτός γίνεται σύμφωνα με τον αύξοντα αριθμό του κάθε thread ο οποίος ορίστηκε στην ενότητα 2.3 και είναι πάντα ο ίδιος. Στη συνέχεια, η SIMT επιλέγει ένα warp που είναι ελεύθερο προς χρήση και του δίνει την εντολή εκτέλεσης. Και βέβαια όπως είναι κατανοητό, η εντολή αυτή είναι το ίδιο εφαρμόσιμη για όλα τα threads μέσα στο warp τα οποία θα εκτελεστούν ανεξάρτητα, ακόμα και αν χρησιμοποιούν κοινά κομμάτια του κώδικα όπως για παράδειγμα μια υπορουτίνα.

Πολύ σημαντικό θέμα για την μέγιστη απόδοση της συσκευής κατά την εκτέλεση μιας εφαρμογής είναι το πόσα blocks μπορεί να επεξεργαστεί κάθε ομάδα μικροεπεξεργαστών. Αυτό εξαρτάται από το πόσοι registers και πόση από τη διαθέσιμη shared memory χρειάζεται για το κάθε kernel που στέλνεται στην συσκευή μας. Και αυτό γιατί η διαθέσιμη μνήμη, που έχει συγκεκριμένο μέγεθος για την κάθε συσκευή, θα διασπαστεί και θα χρησιμοποιηθεί ανάλογα στα threads των blocks που θα σταλούν. Έτσι, θα πρέπει κατά τον προγραμματισμό της εφαρμογής μας, να είναι γνωστές οι απαιτήσεις του κάθε kernel που θα στέλνεται στη συσκευή ώστε να επιτυγχάνεται η βέλτιστη λειτουργία του. Σε περίπτωση που δεν υπάρχουν αρκετοί registers ή διαθέσιμη μνήμη για την εκτέλεση έστω ενός block ο πυρήνας θα αποτύχει να εκτελεστεί. Αντίστοιχα μια ομάδα πολυεπεξεργαστών μπορεί να εκτελέσει ταυτόχρονα μέχρι 8 blocks.

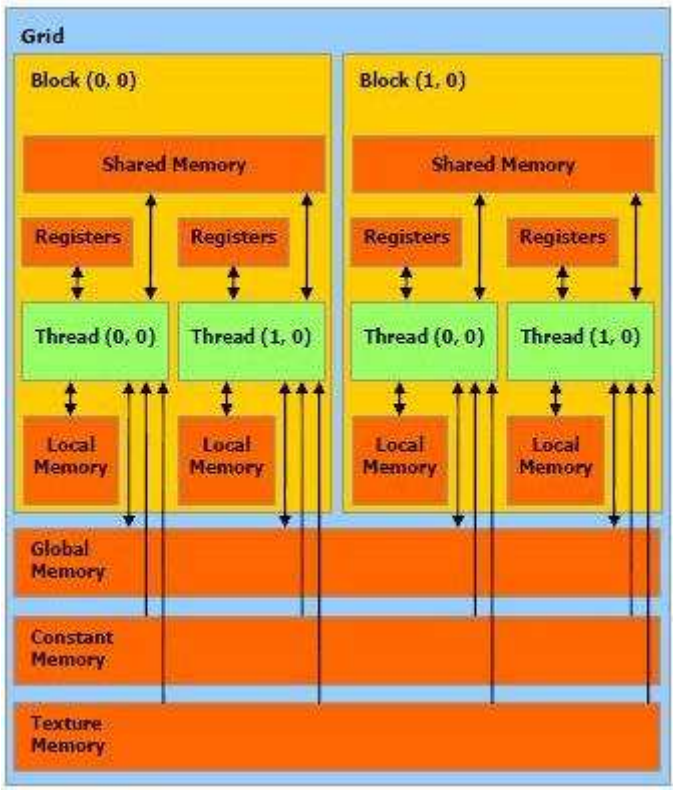
2.6 Θέσεις Μνήμης της Κάρτας

Κύριο μέρος κάθε υπολογιστικού συστήματος και συσκευής αποτελούν οι θέσεις μνήμης του. Οι γενικές μνήμες (DRAM) της κάρτας γραφικών, έχουν σχεδιαστεί για πολύ μεγάλη ευελιξία, αντίστοιχη με αυτές των κεντρικών επεξεργαστών. Αυτό επιτυγχάνεται μέσω της δυνατότητας τους τόσο για ανάγνωση όσο και για εγγραφή πληροφοριών από τη συσκευή προς αυτές σε οποιαδήποτε θέση τους. Αυτή η λειτουργία περιγράφετε στο παρακάτω σχήμα :



Σχ. 2.8 – Λειτουργία DRAM **[12].**

Στο *Σχ. 2.9* που ακολουθεί διακρίνονται οι 6 διαθέσιμες μνήμες που βρίσκονται στην κάρτα γραφικών. Συγκεκριμένα υπάρχουν οι shared memory, local memory, global memory, constant , texture memory και οι registers.



Σχ. 2.9 – Το Μοντέλο της Μνήμης **[12].**

Πολύ βασικό για τον προγραμματισμό στη κάρτα γραφικών είναι να γνωρίζουμε τι μπορεί να έχει πρόσβαση σε κάθε μια από τις παραπάνω μνήμες. Παρατηρώντας το παραπάνω σχήμα βλέπουμε:

⇨στη shared memory μπορεί να διαβάσει και να γράψει το κάθε block

⇨στη local memory μπορεί να διαβάσει και να γράψει το κάθε thread

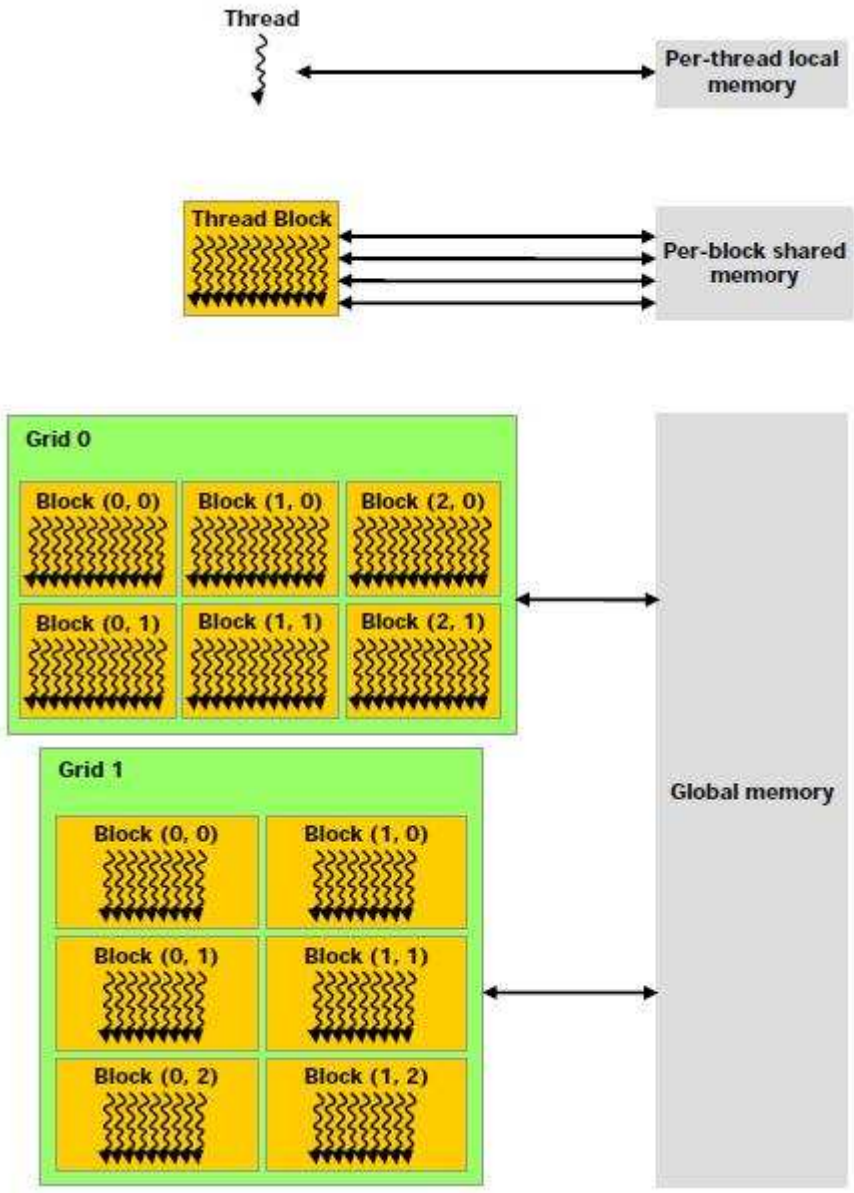
⇨στη global memory μπορεί να διαβάσει και να γράψει το κάθε grid

⇨ στην constant memory μπορεί να διαβάσει μόνο το κάθε grid

⇨ στην texture memory μπορεί να διαβάσει μόνο το κάθε grid

⇨ στους registers μπορεί να διαβάσει και να γράψει το κάθε thread

Τις παραπάνω αλληλοεπιδράσεις μπορούμε να τις αναπαραστήσουμε με το παρακάτω σχήμα :



Σχ. 2.10 – Η Χρήση της κάθε Μνήμης **[12].**

Σημαντικό χαρακτηριστικό είναι ότι στις global, constant και texture μνήμες μπορεί να γράψει και να διαβάσει και ο host, αλλά με μεγάλο overhread[9].

Επίσης πολύ σημαντική για τη βέλτιστη εκτέλεση ενός προγράμματος είναι η ταχύτητα ανάγνωσης της κάθε μνήμης δηλαδή το λεγόμενο memory bandwidth.

Στη συνέχεια, ακολουθεί αναλυτική παρουσίαση για το πώς λειτουργεί η κάθε μνήμη.

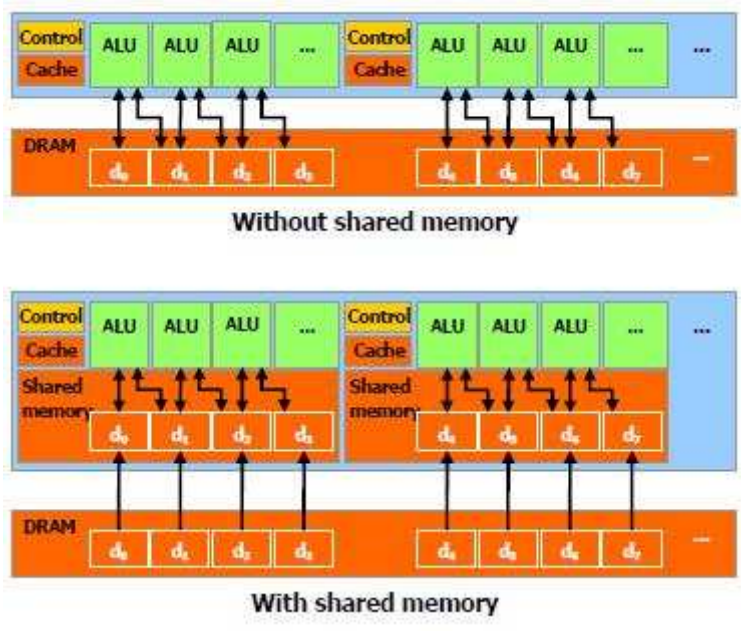
[9] Πρόσθετο Υπολογιστικό Κόστος

2.6.1 Constant Memory

Η Constant Memory αποτελεί λανθάνουσα μνήμη (cached memory) γεγονός που την κάνει πολύ γρήγορη. Ο χρόνος που απαιτούν όλα τα threads ενός μισού warp να διαβάσουν από την constant μνήμη είναι ο ίδιος με το χρόνο που θα χρειάζονταν για να διαβάσουν από έναν register, αρκεί βέβαια όλα τα threads να διαβάζουν στην ίδια διεύθυνση μνήμης. Από εκεί και πέρα ο χρόνος ανάγνωσης αυξάνει γραμμικά με την αύξηση του αριθμού των διευθύνσεων από τις οποίες διαβάζουν τα threads. Έτσι γίνεται κατανοητό ότι, με τον κατάλληλο σχεδιασμό του προγράμματος, μπορούμε να επιτύχουμε τη μέγιστη απόδοση στην ανάγνωση από την Constant Memory.

2.6.2 Shared Memory

Επειδή βρίσκεται πάνω στο τσιπ της κάθε ομάδας πολυεπεξεργαστών, είναι πολύ πιο γρήγορη από τη local και την global μνήμη. Συγκεκριμένα ο χρόνος που χρειάζονται όλα τα threads ενός warp για να αποκτήσουν πρόσβαση στην shared μνήμη είναι ίδιος με όσο θα χρειαζόταν για να αποκτήσουν πρόσβαση στους registers. Η λειτουργία της συσκευής με χρήση της μνήμης αυτής, που έχει ως αποτέλεσμα τη μείωση του απαιτούμενου εύρους μεταφοράς δεδομένων από τη DRAM μνήμη, αλλά και χωρίς αυτή, φαίνεται στη παρακάτω εικόνα:

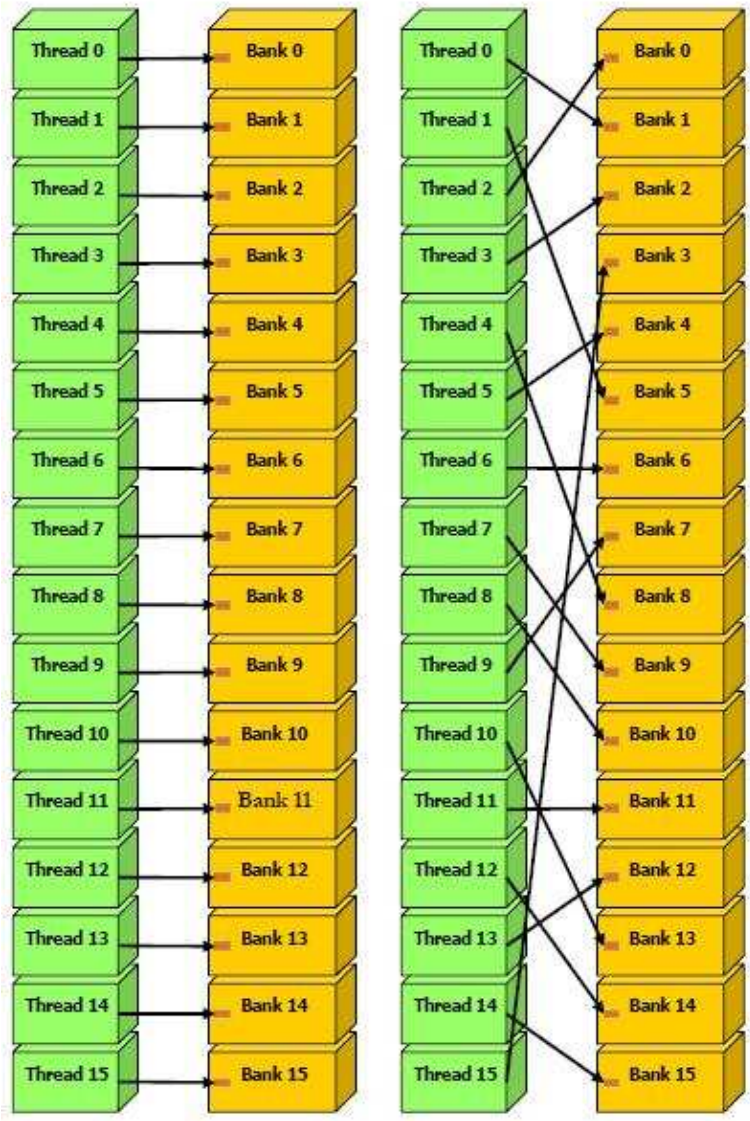


Σχ. 2.11 – Χρήση Κοινής Μνήμης **[12].**

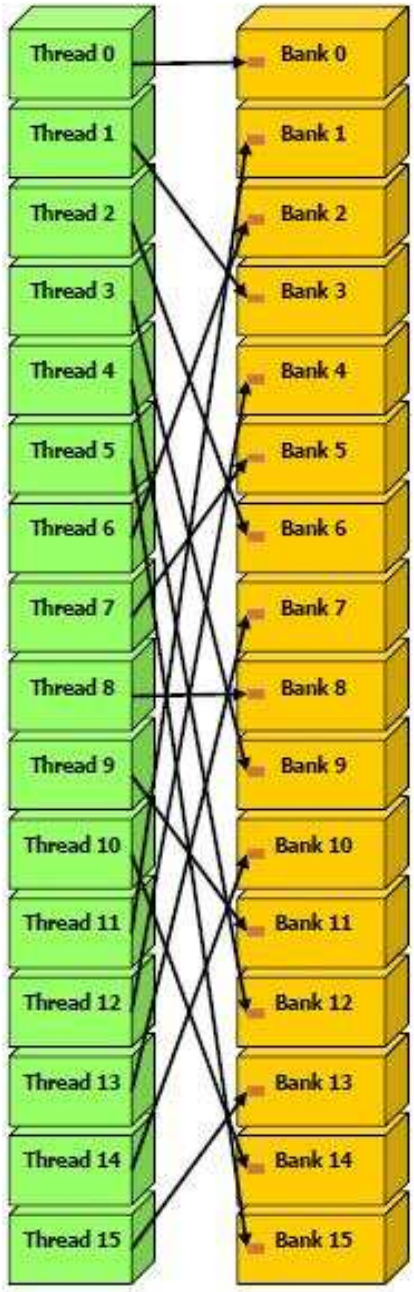
Για να πετύχει η shared μνήμη τόσο μεγάλη ταχύτητα μεταφοράς δεδομένων, έχει χωριστεί σε ίσου μεγέθους κλάσματα μνήμης τα οποία ονομάζονται **banks** (τράπεζες), σε καθένα από τα οποία μπορεί να υπάρχει πρόσβαση ταυτόχρονα. Έτσι

τυχόν απαίτηση για ανάγνωση ή εγγραφή στην μνήμη για N διευθύνσεις που αντιστοιχούν σε N μονάδες banks μπορούν να ικανοποιηθούν ταυτόχρονα, δίνοντας έτσι ταχύτητα μεταφοράς των δεδομένων N φορές μεγαλύτερη από αυτή μιας μονής bank.

Ορίζουμε λοιπόν το μέγεθος του κάθε στοιχειώδη bank έτσι, ώστε διαδοχικές λέξεις των 32-bit να εκχωρούνται σε διαδοχικά banks. Με αυτόν τον τρόπο κάθε bank έχει εύρος ζώνης 32-bit για κάθε 2 κύκλους ρολογιού. Στα σχήματα που ακολουθούν υπάρχουν 3 παραδείγματα τέτοιων καταχωρήσεων:



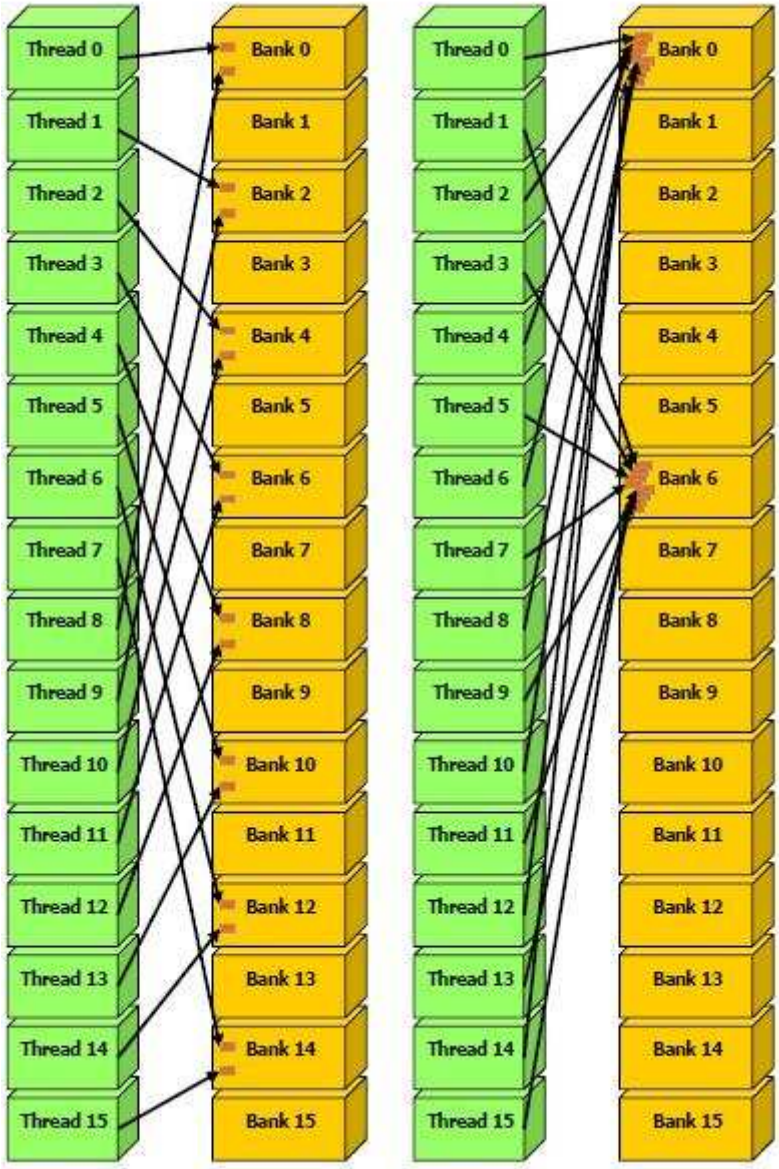
Σχ. 2.12 - Στα αριστερά έχουμε ομαλή γραμμική καταχώρηση μιας λέξης 32-bit για κάθε ένα thread σε κάθε ένα bank, ενώ στα δεξιά έχουμε ομαλή τυχαία καταχώρηση μιας λέξης 32-bit για κάθε ένα thread σε κάθε ένα bank **[12].**



Σχ. 2.13 - Ομαλή γραμμική καταχώρηση 3 λέξεων 32-bit για κάθε thread σε θέσεις μνήμης **[12].**

Όμως, υπάρχει περίπτωση 2 διευθύνσεις μνήμης να πέσουν πάνω στο ίδιο bank μνήμης, δημιουργώντας σύγκρουση και ως αποτέλεσμα η πρόσβαση στον συγκεκριμένο bank πρέπει να γίνει πλέον σειριακά. Έτσι η μνήμη θα διαχωρίσει την αίτηση πρόσβασης στο συγκεκριμένο bank σε επιμέρους αιτήσεις πρόσβασης που δεν αντιμετωπίζουν πρόβλημα σύγκρουσης, μειώνοντας έτσι την ταχύτητα

μεταφοράς δεδομένων κατά ένα ποσοστό ίσο με τον αριθμό των διαχωρισμένων αιτήσεων. Στο σχήμα που ακολουθεί έχουμε δυο τέτοια παραδείγματα.



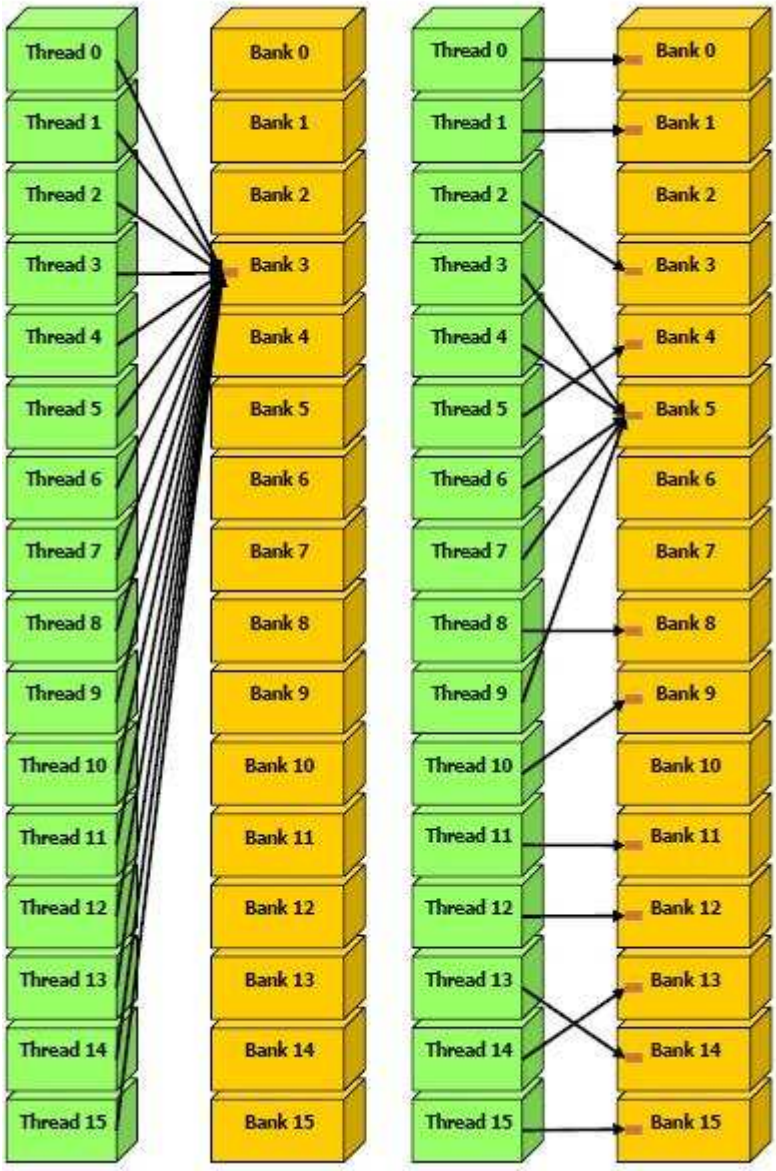
Σχ. 2.14 - Γραμμική καταχώρηση δύο λέξεων 32-bit ανά thread που προκαλεί σύγκρουση διευθύνσεων $2^{η}$ τάξης και γραμμική καταχώρηση 8 λέξεων 32-bit ανά thread που προκαλεί σύγκρουση διευθύνσεων $8^{ης}$ τάξης **[12].**

Για να επιτευχθεί λοιπόν μέγιστη απόδοση, πρέπει να καταλάβουμε πως η μνήμη δίνει διευθύνσεις στα στοιχειώδη banks ώστε να ικανοποιηθούν όλες οι απαιτούμενες προσβάσεις μνήμης με όσο το δυνατό λιγότερες συγκρούσεις.

Για συσκευές λοιπόν με υπολογιστική δύναμη 1.0 (βλέπε ενότητα 2.7) όπου το μέγεθος του κάθε warp είναι 32 και ο αριθμός των banks είναι 16, μια εντολή από την κοινή μνήμη προς το warp διασπάται σε μια εντολή για το πρώτο μισό του warp και μια εντολή για το δεύτερο μισό του warp. Έτσι, ένα thread που ανήκει στο

πρώτο μισό του warp δεν μπορεί να συγκρουστεί με ένα άλλο thread που ανήκει στο δεύτερο μισό του ίδιου warp.

Επίσης, η κοινή μνήμη διαθέτει ένα μηχανισμό «αναμετάδοσης», όπου μια λέξη των 32-bit που απαιτείται να εκχωρηθεί σε μία κλήση μνήμης, μπορεί να διαβαστεί και να αναμεταδοθεί σε πάρα πολλά threads που την χρειάζονται ταυτόχρονα. Με αυτόν το μηχανισμό, μειώνεται ο αριθμός των συγκρούσεων που προκαλούνται με τις εκχωρήσεις θέσεων μνήμης. Στο σχήμα που ακολουθεί, υπάρχουν δύο παραδείγματα για αυτήν τη διαδικασία.



Σχ. 2.15 - Στο αριστερό σχήμα οι καταχωρήσεις της μνήμης γίνονται χωρίς σύγκρουση, μιας και όλα τα threads διαβάζουν από την ίδια θέση μνήμης με χρήση αναμετάδοσης, ενώ στο δεξί δεν θα προκληθεί σύγκρουση αν η λέξη στο bank 5 είναι αυτή που θα αναμεταδοθεί, αλλά σε αντίθετη περίπτωση θα υπάρξει σύγκρουση $2^{ης}$ τάξης **[12].**

2.6.3 Registers

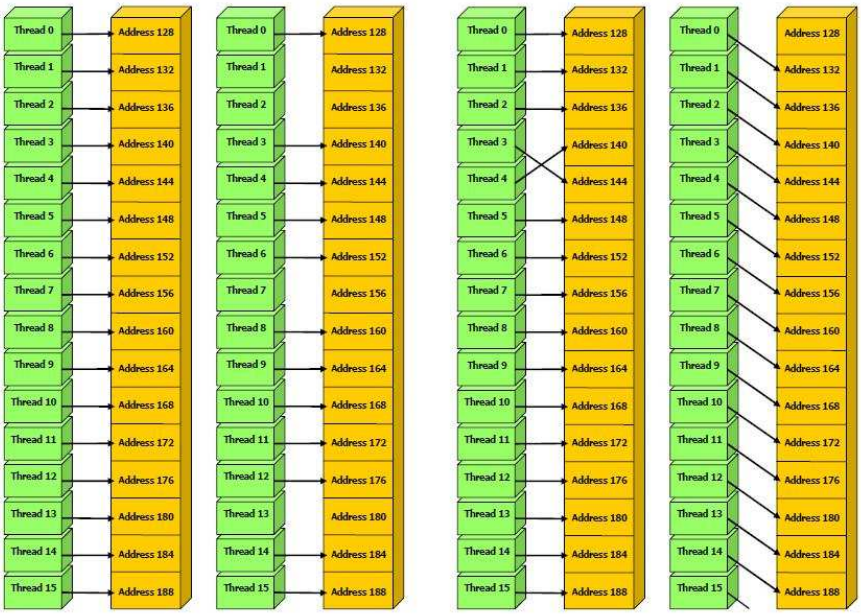
Γενικά η πρόσβαση στους registers δεν επιβαρύνει τους κύκλους του ρολογιού ανά εντολή κλήσης, αλλά καθυστερήσεις μπορεί να προκύψουν για αναγνώσεις που ακολουθούν αμέσως μετά από εγγραφή αλλά και με συγκρούσεις όπως στην περίπτωση της shared μνήμης. Οι μεν καθυστερήσεις εγγραφής ανάγνωσης μπορούν να αγνοηθούν αν τα ενεργά threads για κάθε ομάδα μικροεπεξεργαστών είναι περισσότερα από 192, οι δε συγκρούσεις μπορούν να ελαχιστοποιηθούν όταν τα thread που υπάρχουν σε κάθε block είναι πολλαπλάσια του 64.

2.6.4 Global Memory

Η global μνήμη δεν αποτελεί λανθάνουσα θέση μνήμης (cached memory), γεγονός που κάνει την πρόσβαση σε αυτή να κοστίζει χρονικά, επομένως είναι πολύ σημαντικό να ακολουθούνται τα σωστά πρότυπα καταχώρησης για να επιτυγχάνεται μέγιστο εύρος μεταφοράς δεδομένων.

Αρχικά, η συσκευή έχει την δυνατότητα να διαβάζει λέξεις μεγέθους 32-bit, 64-bit και 128-bit από την global μνήμη και να τις καταχωρεί στους registers με μια μονή εντολή.

Επίσης, οι διευθύνσεις της global μνήμης στις οποίες ταυτόχρονα αποκτούν πρόσβαση κάθε thread ενός μισού warp κατά την εκτέλεση μιας εντολής ανάγνωσης ή εγγραφής, πρέπει να διοργανώνονται με τέτοιο τρόπο ώστε οι προσβάσεις στην μνήμη να μπορούν να συγχωνευτούν σε μια μονή πρόσβαση μνήμης. Στα σχήματα που ακολουθούν παρουσιάζονται χαρακτηριστικά παραδείγματα.



Σχ. 2.16 – Αριστερά: Συγχωνευμένες προσβάσεις μνήμης, Δεξιά: Μη συγχωνευμένες. Σημαντικό είναι ότι οι συγχωνευμένες 64-bit προσβάσεις παρουσιάζουν λίγο μικρότερο εύρος συχνότητας από τις 32-bit, ενώ οι 128-bit παρουσιάζουν αισθητά μειωμένο **[12].**

2.6.5 Texture Memory

Η texture memory είναι λανθάνουσα μνήμη (cached memory) ώστε μια πρόσβαση σε αυτή να κοστίζει όσο μια ανάγνωση από τη μνήμη της συσκευής. Το κύριο χαρακτηριστικό της είναι ότι είναι σχεδιασμένη βέλτιστα για διδιάστατη χωρική κατανομή του πλέγματος υπολογισμού που σημαίνει ότι τα threads ενός warp που διαβάζουν από θέσεις της texture μνήμης, θα πετύχουν βέλτιστη απόδοση όταν αυτές οι θέσεις μνήμης έχουν διευθύνσεις που είναι κοντά η μία στην άλλη. Επίσης, δεν έχει περιορισμούς για το πώς πρέπει να γίνεται η πρόσβαση σε αυτή, για να έχει βέλτιστη απόδοση, όπως οι constant και global μνήμες. Γενικότερα, με τη χρήση της συγκεκριμένης μνήμης πετυχαίνουμε μείωση της απαίτησης για μεγάλο συνολικό εύρος διαμεταγωγής πληροφοριών από και προς τις μνήμες.

Τέλος, ύστερα από την παραπάνω ανάλυση παρατηρείται ότι το εύρος συχνότητας μεταφοράς δεδομένων μεταξύ της συσκευής και της μνήμης της συσκευής είναι πολύ μεγαλύτερο μεταξύ του εύρους συχνότητας της συσκευής και του host. Έτσι για να καταφέρουμε να επιτύχουμε τη βέλτιστη απόδοση της συσκευής, πρέπει να προσπαθούμε να ελαχιστοποιήσουμε τη μεταφορά δεδομένων μεταξύ host και device, ακόμα και αν αυτό συνεπάγεται με εκτέλεση κάποιον kernels σε χαμηλού επιπέδου παραλληλία. Επίσης, η μεταφορά των δεδομένων μας στη συσκευή, εκτελείται ταχύτερα όταν τα δεδομένα αυτά μεταφέρονται όλα μαζί και όχι σε μικρές ομάδες.

2.7 Συμβατές GPU και Υπολογιστικές δυνατότητες

Οι κάρτες γραφικών της NVIDIA που έχουν την δυνατότητα να χρησιμοποιούν την τεχνολογία CUDA χωρίζονται σε 3 κατηγορίες.

⇨Στις παλιότερες κάρτες γραφικών που ανήκουν όλες οι κάρτες των σειρών NVIDIA GeForce 8 και 9 με κύριο χαρακτηριστικό ότι υποστηρίζουν πράξεις μόνο με δεκαδικούς αριθμούς απλής ακρίβειας.

⇨Στις πιο σύγχρονες κάρτες γραφικών της σειράς GT200 (Graphics Tesla) με κύριο χαρακτηριστικό ότι υποστηρίζουν πράξεις με δεκαδικούς διπλής ακρίβειας, γεγονός πολύ σημαντικό για την ακρίβεια των υπολογισμών στις εφαρμογές στον τομέα της Υπολογιστικής Ρευστοδυναμικής.

⇨Στα ολοκληρωμένα υπολογιστικά συστήματα βασιζόμενα σε κάρτα γραφικών και τα οποία είναι τα NVIDIA Tesla και NVIDIA Quadro.

Κάθε κάρτα γραφικών που έχει τη δυνατότητα να εκτελέσει μοντέλο προγραμματισμού σε γλώσσα CUDA, χαρακτηρίζεται από ένα βασικό αριθμό, ανάλογα με το πόσο σύγχρονη είναι, για να διακρίνονται εύκολα οι υπολογιστικές

της δυνατότητες. Οι πρώτες κάρτες χαρακτηρίζονται από τον αριθμό 1.0 , ενώ οι κάρτες τελευταίας τεχνολογίας (μέχρι σήμερα) χαρακτηρίζονται από τον αριθμό 1.3. Ένα άλλο κύριο χαρακτηριστικό που διακρίνει τις κάρτες και δίνει μια πολύ καλή εκτίμηση για τις δυνατότητες τους είναι ο αριθμός των ομάδων πολυεπεξεργαστών που διαθέτουν. Οι πρώτες κάρτες γραφικών, της σειράς 8 είχαν μόνο 2 ομάδες, ενώ οι σημερινές φτάνουν μέχρι και τις 30 .

Εμείς θα ασχοληθούμε αναλυτικά με την κατηγορία καρτών GT200 μιας και σε αυτήν ανήκει η κάρτα γραφικών που χρησιμοποιήσαμε για την εκτέλεση των εφαρμογών στο πλαίσιο της διπλωματικής εργασίας.

2.8 Το Υπολογιστικό Σύστημα που χρησιμοποιήθηκε

Το υπολογιστικό σύστημα που χρησιμοποιήσαμε αποτελείται από τον επεξεργαστή Intel Core 2 Duo E8400 και την κάρτα γραφικών Nvidia GTX285.

2.8.1 Κάρτα Γραφικών GTX285

Η συγκεκριμένη κάρτα γραφικών έχει υπολογιστική δυνατότητα 1.3 και 30 ομάδες μικροεπεξεργαστών.

Αναλυτικά οι δυνατότητες της αυτές , μεταφράζονται ως εξής:

⇨Ο αριθμός των thread σε κάθε block μπορεί να είναι μέχρι 512.

⇨Η μέγιστη διάσταση ενός block μπορεί να είναι κατά x 512 , κατά y 512 και κατά z 64 .

⇨Η μέγιστη τιμή της κάθε διάστασης του συνολικού πλέγματος (grid) μπορεί να είναι 65535.

⇨Το κάθε warp όπως αναφέρθηκε και πιο πάνω μπορεί να περιέχει μέχρι 32 threads.

⇨Ο αριθμός των registers κάθε ομάδας πολυεπεξεργαστών είναι 16384 .

⇨Το μέγεθος της shared μνήμης κάθε ομάδας πολυεπεξεργαστών είναι 16KB .

⇨Το συνολικό μέγεθος της Constant μνήμης είναι 64KB .

⇨Ο μέγιστος αριθμός ενεργών blocks για κάθε ομάδα πολυεπεξεργαστών είναι 8 .

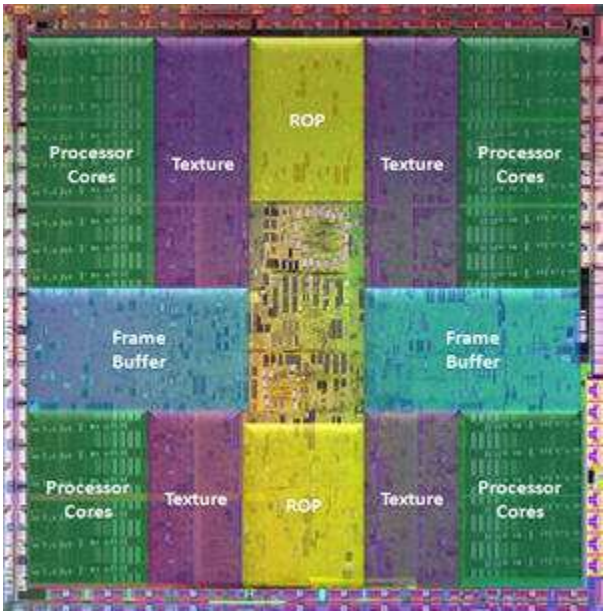
⇨Ο μέγιστος αριθμός ενεργών warps για κάθε ομάδα πολυεπεξεργαστών είναι 32 .

⇨Ο μέγιστος αριθμός ενεργών threads για κάθε ομάδα πολυεπεξεργαστών είναι 1024.

⇨Κάθε ομάδα πολυεπεξεργαστών αποτελείται από 8 βαθμωτούς επεξεργαστές έτσι ώστε όλη η ομάδα να μπορεί να επεξεργαστεί 32 threads σε 4 κύκλους του ρολογιού.

⇨Και τελευταίο αλλά σημαντικότερο, ότι όπως είπαμε υποστηρίζει πράξεις δεκαδικών διπλής ακρίβειας.

Στο σχήμα που ακολουθεί, παρουσιάζεται μια ξεκάθαρη εικόνα του πυρήνα της κάρτας γραφικών, όπου είναι διακριτή η αρχιτεκτονική της και τα επιμέρους τμήματα της.



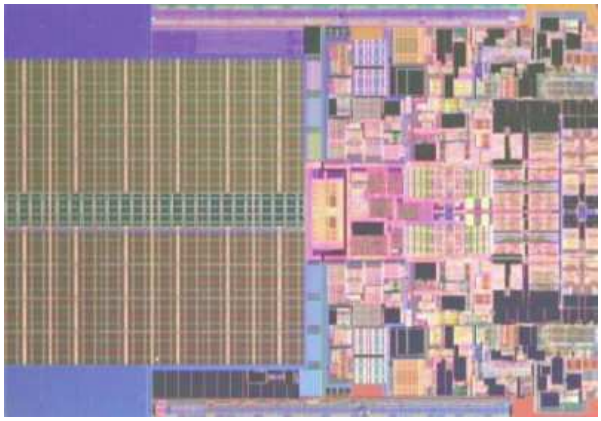
Σχ. 2.17 - GT200 **[22].**

Συγκεκριμένα, διακρίνουμε με πράσινο χρώμα τέσσερις ομάδες επεξεργαστών (Processor Cores), κάθε μια συνδεμένη με ξεχωριστή μονάδα texture (μοβ χρώμα). Κύριο χαρακτηριστικό αποτελεί ότι κάθε ομάδα επεξεργαστών περιλαμβάνει τα ανεξάρτητα TCPs (Texture Processing Clusters) μαζί με την τοπική τους μνήμη.

2.8.2 Επεξεργαστής Core 2 Duo E8400

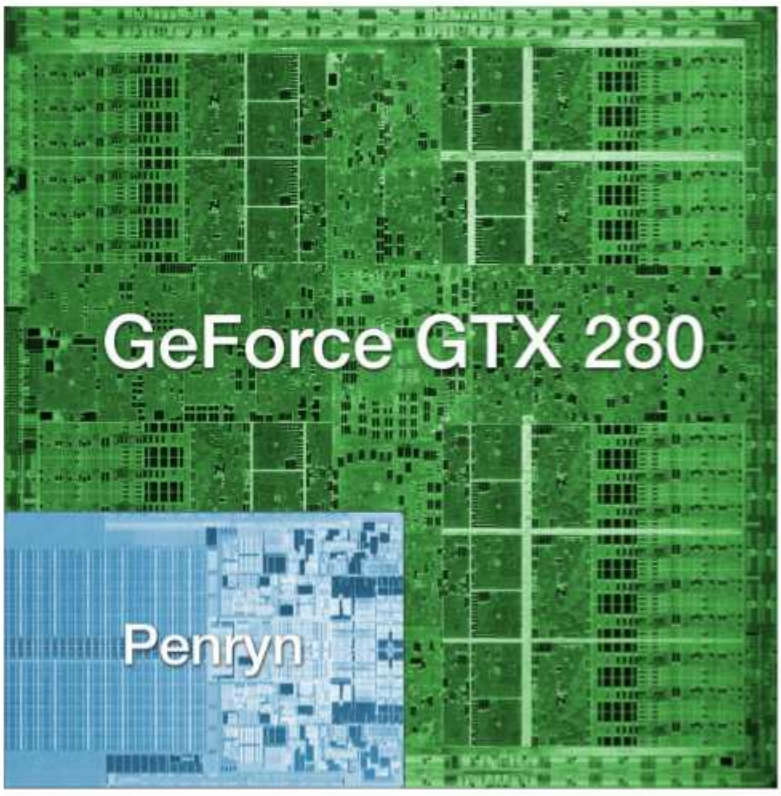
Ο E8400 είναι διπύρηνος επεξεργαστής με συχνότητα 3.000 Mhz .

Στο σχήμα που ακολουθεί, παρουσιάζεται αντίστοιχα μια ξεκάθαρη εικόνα του τσιπ του κεντρικού επεξεργαστή όπου είναι διακριτοί οι δυο πυρήνες του.



Σχ. 2.18 - Core2Duo **[26].**

Στο Σχ. 2.19 που ακολουθεί βλέπουμε το μέγεθος του πυρήνα μιας κάρτας γραφικών της σειράς GT200 σε σχέση με τον πυρήνα ενός επεξεργαστή της γενιάς Core 2 Duo



Σχ. 2.19 - Core2Duo, GTX280 **[17].**

Διακρίνουμε εύκολα ότι το μέγεθος των επιμέρους πυρήνων είναι σχετικά το ίδιο όμως στην κάρτα γραφικών το πλήθος τους είναι μεγαλύτερο από αυτό του επεξεργαστή.

Κεφάλαιο 3ο
Γλώσσα Προγραμματισμού CUDA

Εισαγωγή:

Στο προηγούμενο κεφάλαιο αναλύθηκε η αρχιτεκτονική της κάρτας γραφικών και τον τρόπο με τον οποίο αυτή διαχειρίζεται την εκτέλεση των εφαρμογών.

Γενικότερα, η GPU είναι καλά προσαρμοσμένη για την αντιμετώπιση προβλημάτων που μπορούν να αναλυθούν σε υποπροβλήματα τα οποία δέχονται παράλληλη επεξεργασία και τα οποία εκτελούν αριθμητικές πράξεις σε μεγαλύτερο βαθμό σε σχέση με τις ενέργειες διαχείρισης μνήμης.

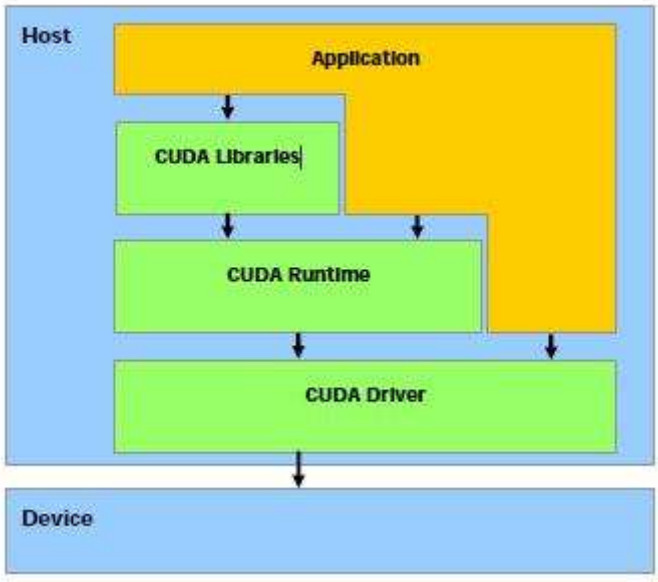
Μέχρι σήμερα ωστόσο, η GPU μπορούσε να προγραμματιστεί μόνο μέσω graphics API [10], γεγονός που μείωνε την ταχύτητα εκτέλεσης των εφαρμογών και ακύρωνε τα πλεονεκτήματα της GPU για χρήση πέραν της επεξεργασίας γραφικών, κάνοντας έτσι την πρόσβαση σε αυτήν την υπολογιστική δύναμη, διαδικασία πολύ δύσκολη.

Σήμερα όμως με την εισαγωγή του μοντέλου προγραμματισμού **CUDA** (Compute Unified Device Architecture) από την NVIDIA που υλοποιείται πάνω στις κάρτες γραφικών της με την αρχιτεκτονική Tesla, Quadro, GeForce και Ion, ξεπερνιούνται τα παραπάνω προβλήματα και παρέχεται η δυνατότητα στο μηχανικό να προγραμματίσει τις εφαρμογές του πάνω στην κάρτα γραφικών και να εκμεταλλευτεί τη μεγάλη της υπολογιστική ισχύ.

Σε αυτό το κεφάλαιο θα μιλήσουμε για το λογισμικό το οποίο συνεργάζεται με την παραπάνω αρχιτεκτονική (CUDA API) και με βάση το οποίο θα επιτύχουμε εκτέλεση των προγραμμάτων μας στην κάρτα γραφικών.

Το πακέτο λογισμικού της CUDA διακρίνεται σε επιμέρους διακριτά τμήματα όπως φαίνεται στο σχήμα 3.1 που ακολουθεί και τα οποία είναι οι οδηγοί (CUDA drivers) της συσκευής, η διεπαφή εφαρμογής προγραμματισμού (API) μαζί με τα εκτελέσιμα της (runtime) και 2 υψηλού επιπέδου βιβλιοθήκες κοινής χρήσης.

[10] Το μοντέλο επικοινωνίας το οποίο παρεμβάλλεται μεταξύ μιας εφαρμογής και του λειτουργικού συστήματος και καθορίζει τον τρόπο με τον οποίο η εφαρμογή αντλεί βιβλιοθήκες από το λειτουργικό.



Σχ. 3.1 – Πακέτο Λογισμικού CUDA **[12].**

Πιο συγκεκριμένα, θα αναλύσουμε τη μέθοδο προγραμματισμού εφαρμογών με βάση τη γλώσσα προγραμματισμού CUDA (η οποία αποτελεί προέκταση της ήδη υπάρχουσας γλώσσας C με την προσθήκη μιας νέας βιβλιοθήκης εκτελέσιμων εντολών), θα παρουσιάσουμε τις νέες εντολές που εισάγονται και με τη χρήση τους επιτυγχάνουμε συνεργασία κεντρικού επεξεργαστή και κάρτας γραφικών και τέλος, θα δώσουμε παραδείγματα για βαθύτερη κατανόηση αυτής της νέας λογικής.

3.1 Προέκταση της C

Το κύριο χαρακτηριστικό και το μεγαλύτερο πλεονέκτημα του μοντέλου προγραμματισμού της CUDA , είναι ότι στηρίζεται στην ήδη υπάρχουσα γλώσσα προγραμματισμού C/C++, της οποίας αποτελεί προέκταση, και με αυτό τον τρόπο, ένα άτομο ήδη εξοικειωμένο στην γλώσσα C, θα μπορέσει να είναι σε θέση να γράφει προγράμματα που θα εκτελούνται στην κάρτα γραφικών του με την προσθήκη ορισμένων παραπάνω γνώσεων.

Οι νέες εντολές που εισάγονται εδώ ως προέκταση της C , δίνουν τη δυνατότητα στο χρήστη να ορίζει τμήματα του κώδικα του προγράμματος του και τα οποία να εκτελούνται στην κάρτα γραφικών.

Αυτές οι εντολές-προεκτάσεις της C χωρίζονται σε 4 βασικές κατηγορίες:

⇨Στους προσδιοριστές των υπορουτινών, με τις οποίες ο χρήστης καθορίζει από πού θα κληθεί μια υπορουτίνα και που θα εκτελεστεί, δηλαδή στον host (CPU) ή στη device (GPU).

⇨Στους προσδιοριστές του είδους των μεταβλητών, με τις οποίες ο χρήστης καθορίζει αν η θέση της μνήμης μιας μεταβλητής βρίσκεται στο host ή στο device.

⇨Στον τρόπο κλήσης μιας υπορουτίνας η οποία καλείται από το host και εκτελείται στο device.

⇨Σε μια κατηγορία 4 μεταβλητών που μας καθορίζουν τη διάσταση του grid (στο οποίο θα εκτελεστεί το πρόγραμμα μας) και του κάθε block μας και μας λένε σε ποίο thread, ποιού block βρισκόμαστε κατά την εκτέλεση του προγράμματος.

Ακολουθεί αναλυτική περιγραφή της κάθε κατηγορίας .

3.1.1 Προσδιοριστές Υπορουτινών

Υπάρχουν 3 είδη προσδιοριστικών που τοποθετούνται πριν από το όνομα της κάθε υπορουτίνας κατά τον ορισμό της και δείχνουν, από πού θα κληθεί η υπορουτίνα και που θα εκτελεστεί.

Έτσι :

⇨Για να κληθεί μια υπορουτίνα από την device και να εκτελεστεί στην device, πρέπει να χρησιμοποιήσουμε το προσδιοριστικό **__device__**.

⇨ Για να κληθεί μια υπορουτίνα από το host και να εκτελεστεί στην device, πρέπει να χρησιμοποιήσουμε το προσδιοριστικό **__global__.**

⇨ Για να κ κληθεί μια υπορουτίνα από το host και να εκτελεστεί στο host, πρέπει να χρησιμοποιήσουμε το προσδιοριστικό **__host__.**

⇨Στην ειδική περίπτωση όπου θέλουμε μια υπορουτίνα να είναι διαθέσιμη για εκτέλεση τόσο στην device όσο και στο host, χρησιμοποιούμε και το προσδιοριστικό **__device__** και το **__host__** ταυτόχρονα.

Τα παραπάνω προσδιοριστικά έχουν κάποιες ξεχωριστές ιδιότητες / περιορισμούς που περιγράφονται ευθύς αμέσως:

⇨Κάθε υπορουτίνα **__global__** καλείται με συγκεκριμένο τρόπο που περιγράφετε στην ενότητα 3.1.3 , ενώ πρέπει να είναι τύπου void όσον αφορά τον τρόπο επιστροφής του αποτελέσματος της .

⇨Κάθε κλήση υπορουτίνας __global__ είναι ασύγχρονη , δηλαδή θα επιστρέψει το αποτέλεσμα της ανεξάρτητα με το αν η συσκευή έχει τελειώσει την εκτέλεση άλλων εντολών ή όχι (Περισσότερα σχετικά με την ασύγχρονη λειτουργία στην ενότητα 3.3)

⇨Οι παράμετροι κάθε συνάρτησης __global__ μεταφέρονται μέσω της shared μνήμης και δεν μπορούν να έχουν συνολικό μέγεθος μεγαλύτερο από 256 bytes.

⇨Οι υπορουτίνες που εκτελούνται στην device δεν επιτρέπεται να καλέσουν τον εαυτό τους , δηλαδή δεν επιτρέπεται η αναδρομική κλήση.

3.1.2 Προσδιοριστές Μεταβλητών

Υπάρχουν 3 είδη προσδιοριστικών που τοποθετούνται πριν το όνομα της μεταβλητής στο σημείο ορισμού της και υποδηλώνουν σε ποια μνήμη είναι αποθηκευμένη η μεταβλητή και ανάλογα, ποια είναι η διάρκεια ζωής της. Στην εικόνα 2.7 που αναφερθήκαμε στο κεφάλαιο 1 έχουμε μια σχηματική αναπαράσταση για τη θέση της κάθε μνήμης μέσα στην κάρτα γραφικών.

Έτσι:

⇨Το προσδιοριστικό **__device__** καθορίζει μια μεταβλητή που είναι αποθηκευμένη στην μνήμη της device και συγκεκριμένα στην global memory, η διάρκεια ζωής της είναι ίση με την διάρκεια ζωής της εφαρμογής και είναι προσβάσιμη από οποιοδήποτε thread στην κάρτα αλλά και από τον host.

⇨Το προσδιοριστικό **__constant__** καθορίζει μια μεταβλητή που είναι αποθηκευμένη στη μνήμη της device και συγκεκριμένα στην constant memory, η διάρκεια ζωής της είναι ίση με τη διάρκεια ζωής της εφαρμογής και είναι προσβάσιμη από οποιοδήποτε thread στην κάρτα αλλά και από το host.

⇨Το προσδιοριστικό **__shared__** καθορίζει μια μεταβλητή που είναι αποθηκευμένη στη μνήμη της device και συγκεκριμένα στην shared memory ενός thread block, η διάρκεια ζωής της είναι ίση με τη διάρκεια ζωής του block και είναι προσβάσιμη μόνο από το ίδιο το block. Η ασύγχρονη εκτέλεση των threads σε ένα block έχει σημαντική επίδραση στις **__shared__** μεταβλητές. (Αναλυτική περιγραφή ακολουθεί στην ενότητα XXXX). Σημαντικό είναι οι **__shared__** μεταβλητές δεν μπορούν να αρχικοποιηθούν κατά τον ορισμό τους.

Κύριο χαρακτηριστικό είναι ότι οι **__device__** μεταβλητές μπορούν να οριστούν παράλληλα και ως **__constant__** ή **__ shared__** και έτσι να αλλάξουν οι ιδιότητες τους, ενώ αν δεν υπάρχει χρήση ενός από τα παραπάνω κατά τον ορισμό μιας μεταβλητής στην device τότε θα τοποθετηθεί αυτόματα σε κάποιον register ή στην local memory.

3.1.3 Προσδιοριστικά Κλήσης Υπορουτίνας

Κάθε υπορουτίνα χαρακτηρισμένη με το προσδιοριστικό __global__ κατά τον ορισμό της, πρέπει να χρησιμοποιεί συγκεκριμένο προσδιοριστικό στην εντολή κλήσης της, το οποίο χρησιμοποιείται για να καθορίσει τις διαστάσεις του grid και των blocks που θα χρησιμοποιηθούν κατά την εκτέλεση της συγκεκριμένης υπορουτίνας.

Για μια συνάρτηση της μορφής __global__ void function(double* parameter); το προσδιοριστικό κλίσης της είναι ως εξής : function **<<<Dg , Db , Ns>>>**(parameter);

Όπου:

⇨Το Dg δείχνει τις διαστάσεις του grid και ισούται με το γινόμενο της διάστασης κατά x με τη διάσταση κατά y

⇨Το Db δείχνει τις διαστάσεις του block και ισούται με το γινόμενο της διάστασης κατά x με τη διάσταση κατά y

⇨Το Ns δείχνει τον αριθμό των bytes της shared memory που είναι δυναμικά δεσμευμένα για κάθε block για τη συγκεκριμένη κλίση της συνάρτησης. Γενικά είναι ένας αριθμός που δεν είναι απαραίτητο να οριστεί, όπου σε αυτή τη περίπτωση έχει την τιμή 0.

Σημαντικό να γνωρίζουμε είναι ότι τα Db και Dg δεν μπορούν να πάρουν τιμές μεγαλύτερες από κάποιες καθορισμένες τιμές που έχουν σχέση με την αρχιτεκτονική της κάρτας και αυτές είναι 512 για κάθε διάσταση για το πλήθος των threads ανά block και 65535 για κάθε διάσταση για το πλήθος των threads ανά grid.

3.1.4 Μεταβλητές Θέσης και Διαστάσεων

Υπάρχουν 4 μεταβλητές τις οποίες τις χρησιμοποιούμε μέσα σε κάθε **__global__** υπορουτίνα οι οποίες δίνουν την διάσταση του συνολικού μας πλέγματος (grid) αλλά και μας προσδιορίζουν ποιο thread εκτελείται κάθε στιγμή ώστε να υπάρχει ανάγνωση των κατάλληλων δεδομένων και επεξεργασία τους.

Έτσι έχουμε:

⇨Την μεταβλητή **gridDim** που όπως μας περιγράφει και το όνομα της, δίνει την διάσταση του grid. (με χρήση του gridDim.x και gridDim.y παίρνουμε αντίστοιχα τις διαστάσεις.

⇨Την μεταβλητή **blockIdx** που δίνει τον αύξοντα αριθμό του block μέσα στο grid . (Με την blockIdx.x παίρνουμε την x θέση και με την blockIdx.y την y).

⇨Την μεταβλητή **blockDim** που δίνει τη διάσταση του block αντίστοιχα με την gridDim.

⇨Την μεταβλητή **threadIdx** που δίνει τον αύξοντα αριθμό του thread μέσα στο block μας αντίστοιχα με την blockIdx.

Προσοχή χρειάζεται διότι δεν πρέπει να χρησιμοποιούμε την διεύθυνση της μνήμης οποιωνδήποτε από τις παραπάνω μεταβλητές και δεν πρέπει να μεταβάλλουμε τις τιμές των ίδιων των μεταβλητών μέσα στο πρόγραμμά μας. Τέτοιες ενέργειες θα ήταν καταστροφικές για τον ομαλό συντονισμό των threads και τη ροή του προγράμματος μας και για αυτό δεν μας επιτρέπονται.

Από τα παραπάνω βλέπουμε ότι οι προεκτάσεις στην C που εισάγονται είναι περιορισμένες, ευκολονόητες και απλές στην εφαρμογή τους, γεγονός που αποδεικνύει αυτό που είχαμε πει αρχικά ότι η εκμάθηση του προγραμματισμού σε CUDA είναι εύκολη διαδικασία για κάθε εξοικειωμένο χρήστη σε C/C++.

3.2 Προσθήκες Βιβλιοθήκης Εκτελέσιμων Εντολών

Η βιβλιοθήκη των εκτελέσιμων εντολών την CUDA χωρίζεται στα εξής 3 τμήματα:

⇨Το τμήμα του host που περιλαμβάνει εντολές που εκτελούνται στο host και χρησιμοποιούνται για να αποκτούμε πρόσβαση και να ελέγχουμε μια ή περισσότερες κάρτες γραφικών από το host.

⇨Το τμήμα της device που περιλαμβάνει εντολές που εκτελούνται και απευθύνονται μόνο στην device.

⇨Το τμήμα που είναι κοινό και για τον host και για την device και υποστηρίζονται και από τα δυο. Σημαντική παρατήρηση είναι ότι οι μόνες εντολές της C που επιτρέπεται να εκτελεστούν στην κάρτα γραφικών είναι αυτές που ανήκουν σε αυτήν τη κατηγορία.

3.2.1 Βιβλιοθήκη Κοινών Εντολών

Σε αυτό το τμήμα ανήκουν :

⇨Οι ορισμοί μεταβλητών διανυσματικού τύπου όπως int1,int2,float1,float2,long1,long2, κυριότερη εκ των οποίων είναι οι μεταβλητές τύπου dim3 που είναι πάντα ακέραιες και σε αυτές ανήκουν οι μεταβλητές που είδαμε στην ενότητα 3.1.4.

⇨Οι γνωστές μαθηματικές συναρτήσεις. Αναλυτικά θα περιγραφούν στην ενότητα 2.4.1).

⇨Η συνάρτηση clock_t clock(); Που χρησιμοποιείται για να δούμε πόσο χρόνο κάνει κάθε thread ξεχωριστά για να εκτελεστεί στην κάρτα.

⇨Και τέλος, οι εντολές που μας επιτρέπουν να αποκτήσουμε πρόσβαση στην texture memory της κάρτας γραφικών και να την εκμεταλλευτούμε στους υπολογισμούς μας. Αναλυτική παρουσίαση για το πώς γίνεται αυτό δεν θεωρείται απαραίτητο μιας και δεν χρησιμοποιήθηκε η συγκεκριμένη λειτουργία στο πλαίσιο αυτής της διπλωματικής.

3.2.2 Βιβλιοθήκη Εντολών Συσκευής

Σε αυτό το τμήμα ανήκουν:

⇨Οι γνωστές μαθηματικές συναρτήσεις οι οποίες όπως είναι εκφρασμένες με άλλο όνομα και εκτελούνται στην κάρτα γραφικών που έχει ως αποτέλεσμα διαφορές στον χρόνο εκτέλεσης και στην ακρίβεια τους και αναλύονται στην ενότητα 3.4.2).

⇨Η εντολή συγχρονισμού των threads που θα αναλυθεί στην ενότητα 3.3 που αναφέρεται στην ασύγχρονη λειτουργία.

⇨Οι συναρτήσεις στρογγυλοποίησης των τιμών των μεταβλητών.

⇨Εντολές διαχείρισης της μνήμης texture.

3.2.3 Βιβλιοθήκη Εντολών Host

Είναι τμήμα στο οποίο ανήκουν μερικές πολύ σημαντικές εντολές όπως:

⇨Εντολές με τις οποίες διαχειριζόμαστε τις διαθέσιμες συσκευές. Δηλαδή ένα υπολογιστικό σύστημα μπορεί να αποτελείται από περισσότερες από μια κάρτες γραφικών. Με τις κατάλληλες εντολές μπορούμε να αριθμήσουμε τις διαθέσιμες συσκευές και να ορίσουμε σε ποια θα γίνει η εκτέλεση του προγράμματος μας. Αυτό είναι πολύ σημαντικό διότι ενώ πολλά threads του host μπορούν να εκτελούνται σε μια συσκευή, το κάθε thread όμως μπορεί να εκτελείται μόνο σε μια συσκευή.

⇨Εντολές με τις οποίες γίνεται η διαχείριση της μνήμης μεταξύ του host και της device. Συγκεκριμένα υπάρχουν οι εντολές **cudaMalloc()** και **cudaFree()** με τις οποίες καταχωρούμε και απελευθερώνουμε θέσεις μνήμης στην device από τον host, καθώς και οι εντολές της οικογένειας **cudaMemcpy()** με τις οποίες αντιγράφουμε τις τιμές των μεταβλητών από τον host στην device αλλά και πολλές άλλες για διάφορες περιπτώσεις.

⇨Και τέλος μια μεγάλη κατηγορία εντολών με τις οποίες μπορούμε να διαχειριστούμε τα streams (και τα events) [11] στην device και ακόμα και να πετύχουμε ασύγχρονη εκτέλεση τους.

[11] Ακολουθία ενεργειών που εκτελούνται σε σειρά.

3.3 Ασύγχρονη Λειτουργία

Είδαμε ότι όταν καλούμε μια **__global__** συνάρτηση με το προσδιοριστικό <<<Dg,Db>>> στην ουσία της δίνουμε εντολή να εκτελέσει το περιεχόμενο της ταυτόχρονα για πολλά παράλληλα και διαφορετικά threads. Όμως ενώ όλα τα threads ξεκινούν την εκτέλεση τους την ίδια χρονική στιγμή , δεν είναι απαραίτητο να τερματίζουν όλα την ίδια χρονική στιγμή. Οι υπορουτίνες μπορεί να περιλαμβάνουν τμήματα επαναλήψεων που απαιτούν αριθμητική σύγκλιση για τον τερματισμό τους και επομένως εξαρτώνται από τις τιμές που παίρνουν ως είσοδο. Πιθανές τιμές εισόδου οδηγούν σε πιο χρονοβόρα εκτέλεση του αντίστοιχου thread.

Όμως υπάρχουν εφαρμογές οι οποίες θα απαιτούσαν τερματισμό όλων των threads ενός block για να σταλεί προς εκτέλεση το επόμενο block. Μια τέτοια εφαρμογή που παραθέτουμε για κατανόηση του προβλήματος αυτού είναι ένα πρόγραμμα το οποίο επιλύει την εξίσωση Laplace. Στο πρόβλημα αυτό για ένα δοσμένο ορθογώνιο καρτεσιανό πλέγμα με ισαπέχοντες κόμβους στην x και y κατεύθυνση, ορίζουμε τις τιμές της μεταβλητής στα άκρα του πλέγματος και θέλουμε να σχεδιάσουμε ένα πρόγραμμα που θα υπολογίζει τις εσωτερικές τιμές της μεταβλητής στο πλέγμα. Για να ικανοποιείται η εξίσωση Laplace θα πρέπει ή μεταβλητή σε κάθε κόμβο να έχει τιμή ίση με το μέσο όρο των τιμών των μεταβλητών των τεσσάρων γειτονικών κόμβων. Ο αλγόριθμος αυτός θα εκτελείται επαναληπτικά χρησιμοποιώντας υποχαλάρωση για να οδηγηθεί στην σύγκλιση του. Όμως έχοντας τον πίνακα με τις αρχικές τιμές του πλέγματος και στην συνέχεια αφού τις επεξεργαστούμε με βάση την εξίσωση Laplace και αφού εφαρμόσουμε την υποχαλάρωση, για να προχωρήσουμε στο επόμενο βήμα που είναι η αθροιστική εύρεση του σφάλματος θα πρέπει να έχουμε εξασφαλίσει ότι έχουν συμπληρωθεί όλα τα στοιχεία του νέου μας πίνακα. Δηλαδή θα πρέπει να έχουμε εξασφαλίσει ότι όλα τα threads του πλέγματος μας έχουν τερματίσει την εκτέλεση τους.

Για το σκοπό αυτό μας δίνεται η επιλογή στην γλώσσα CUDA να ορίσουμε σημεία συγχρονισμού μέσα σε κάθε kernel όπου στα threads κάθε block θα αναστέλλεται η εκτέλεση τους μέχρι όλα να φτάσουν στο σημείο συγχρονισμού. Η λειτουργία αυτή εκτελείται στη συσκευή από τη μονάδα ελέγχου ροής της κάθε ομάδας πολυεπεξεργαστών, όπως αναφέραμε στο κεφάλαιο 1, και για να ενεργοποιηθεί απαιτείται η χρήση της εντολής **__syncthreads()** στον κώδικα μας.

Αναλυτικά, η εντολή **__syncthreads()** ; είναι τύπου void , δηλαδή δεν επιστρέφεται κάποιο αποτέλεσμα στην οθόνη κατά την εκτέλεση της και σκοπός της είναι η αναστολή και ο συγχρονισμός όλων των threads μέσα σε κάθε block . Όταν όλα τα threads φτάσουν στο σημείο της εκτέλεσης της εντολής συγχρονισμού , τότε μόνο η εκτέλεση συνεχίζει κανονικά.

Στην ουσία η εντολή **__syncthreads** συντονίζει την επικοινωνία μεταξύ των threads του ίδιου block. Και αυτό γιατί όταν μερικά threads του block συνδέονται με τις ίδιες θέσεις shared ή global μνήμης, η εγγραφή μετά από ανάγνωση και το ανάποδο σε αυτές τις θέσεις μνήμης είναι πολύ πιθανόν να οδηγήσει σε εσφαλμένες τιμές σε ασύγχρονη λειτουργία.

3.4 Μαθηματικές Συναρτήσεις

Οι μαθηματικές συναρτήσεις της βιβλιοθήκης της CUDA χωρίζονται σε 2 κατηγορίες. Σε αυτή που περιέχει τις μαθηματικές συναρτήσεις που μπορούν να εκτελεστούν τόσο στον host όσο και στη συσκευή και σε αυτήν που περιέχει μαθηματικές συναρτήσεις που μπορούν να εκτελεστούν μόνο στη συσκευή και έχουν ελαφρώς τροποποιημένη λειτουργία. Επίσης κάθε κατηγορία χωρίζεται σε 2 επιμέρους κατηγορίες, μία που περιέχει συναρτήσεις μονής ακρίβειας και μια που περιέχει συναρτήσεις διπλής ακρίβειας.

Αυτή η ανάλυση σχετικά με τις μαθηματικές συναρτήσεις γίνεται διότι στις εφαρμογές των μηχανικών, τέτοιες συναρτήσεις παίζουν μεγάλη σημασία και γι'αυτό θα πρέπει να είναι ξεκάθαρο κάθε συνάρτηση ανάλογα με το αν εκτελείται στην συσκευή ή στον host, τι πλεονεκτήματα και τι προβλήματα δημιουργεί.

3.4.1 Κοινές Συναρτήσεις Βιβλιοθήκης

Οι κοινές μαθηματικές συναρτήσεις μονής ακρίβειας είναι οι γνωστές και έχουν ίδια μορφή όπως και σε κάθε άλλη γλώσσα προγραμματισμού με τη διαφορά ότι στο όνομα τους προστίθεται το γράμμα f.

Έτσι έχουμε ενδεικτικά :

expf(x)
cosf(x)
powf(x,y)
fmaxf(x,y)
fabsf(x)

Οι κοινές μαθηματικές συναρτήσεις διπλής ακρίβειας είναι οι γνωστές και έχουν την ίδια μορφή όπως και σε κάθε άλλη γλώσσα προγραμματισμού.

Έτσι έχουμε αντίστοιχα :

exp(x)
cos(x)
pow(x,y)
fmax(x,y)
fabs(x)

Κύριο χαρακτηριστικό είναι το σφάλμα που εισάγουν αυτές οι δύο διαφορετικές περιπτώσεις. Σφάλμα θα υπάρχει αν ο host δεν υποστηρίζει αυτές τις συναρτήσεις και η τάξη μεγέθους του σφάλματος χαρακτηρίζεται από έναν αριθμό, τον maximum ulp error [12], που είναι διαφορετικός για κάθε συνάρτηση.

Έτσι για τις συναρτήσεις μονής ακρίβειας έχουμε :

Function	Maximum ulp error
sinf(x)	2 (full range)
cosf(x)	2 (full range)
tanf(x)	4 (full range)
sincosf(x,sptr,cptr)	2 (full range)
asinf(x)	4 (full range)
acosf(x)	3 (full range)
atanf(x)	2 (full range)
atan2f(y,x)	3 (full range)
sinhf(x)	3 (full range)
coshf(x)	2 (full range)
tanhf(x)	2 (full range)
asinhf(x)	3 (full range)
acoshf(x)	4 (full range)
atanhf(x)	3 (full range)
powf(x,y)	8 (full range)

Σχ. 3.2 – Σφάλμα Συναρτήσεων μονής ακρίβειας **[12].**

Ενώ για τις αντίστοιχες διπλής ακρίβειας έχουμε:

Function	Maximum ulp error
sin(x)	2 (full range)
cos(x)	2 (full range)
tan(x)	2 (full range)
sincos(x,sptr,cptr)	2 (full range)
asin(x)	2 (full range)
acos(x)	2 (full range)
atan(x)	2 (full range)
atan2(y,x)	2 (full range)
sinh(x)	1 (full range)
cosh(x)	1 (full range)
tanh(x)	1 (full range)
asinh(x)	2 (full range)
acosh(x)	2 (full range)
atanh(x)	2 (full range)
pow(x,y)	2 (full range)

Σχ. 3.3 – Σφάλμα Συναρτήσεων διπλής ακρίβειας **[12].**

[12]Ο ulp είναι ένας αριθμός που δείχνει πόσο αποκλίνει το αποτέλεσμα μιας συνάρτησης από την τιμή που θα έπρεπε να δίνει. Αποτελέσματα με ulp=1 χαρακτηρίζονται από ικανοποιητική ακρίβεια.

Παρατηρούμε ότι συγκεκριμένες συναρτήσεις διπλής ακρίβειας δίνουν μεγαλύτερης ακρίβειας αποτελέσματα.

3.4.2 Συναρτήσεις Βιβλιοθήκης Συσκευής

Οι μαθηματικές συναρτήσεις της συσκευής διακρίνονται από τα εξής χαρακτηριστικά:

⇨Μπορούν να χρησιμοποιηθούν μόνο μέσα στο τμήμα του κώδικα μας που θα τρέξει στην συσκευή, δηλαδή μόνο μέσα σε υπορουτίνες τύπου **__device__** και **__global__**.

⇨Αποτελούν έκδοση περιορισμένου αριθμού των γνωστών συναρτήσεων οι οποίες όμως εκτελούνται ταχύτερα, εισάγοντας παράλληλα όμως και μεγαλύτερο σφάλμα.

Ακολουθούν οι συναρτήσεις διπλής ακρίβειας για εκτέλεση στην συσκευή:

Function	Error bounds
__dadd_[rn,rz,ru,rd](x,y)	IEEE-compliant.
__dmul_[rn,rz,ru,rd](x,y)	IEEE-compliant.
__fma_[rn,rz,ru,rd](x,y,z)	IEEE-compliant.
__double2float_[rn,rz](x)	N/A
__double2int_[rn,rz,ru,rd](x)	N/A
__double2uint_[rn,rz,ru,rd](x)	N/A
__double2ll_[rn,rz,ru,rd](x)	N/A
__double2ull_[rn,rz,ru,rd](x)	N/A
__int2double_rn(x)	N/A
__uint2double_rn(x)	N/A
__ll2double_[rn,rz,ru,rd](x)	N/A
__ull2double_[rn,rz,ru,rd](x)	N/A
__double_as_longlong(x)	N/A
__longlong_as_double(x)	N/A
__double2hiint(x)	N/A
__double2loint(x)	N/A
__hiloint2double(x, y)	N/A

Σχ. 3.4 – Συναρτήσεις Διπλής Ακρίβειας Συσκευής **[12].**

Και οι συναρτήσεις μονής ακρίβειας :

Function	Error bounds
`__fadd_[rn,rz](x,y)`	IEEE-compliant.
`__fmul_[rn,rz](x,y)`	IEEE-compliant.
`__fdividef(x,y)`	For `y` in $[2^{-126}, 2^{126}]$, the maximum ulp error is 2.
`__expf(x)`	The maximum ulp error is `2 + floor(abs(1.16 * x))`.
`__exp10f(x)`	The maximum ulp error is `2 + floor(abs(2.95 * x))`.
`__logf(x)`	For `x` in [0.5, 2], the maximum absolute error is $2^{-21.41}$, otherwise, the maximum ulp error is 3.
`__log2f(x)`	For `x` in [0.5, 2], the maximum absolute error is 2^{-22}, otherwise, the maximum ulp error is 2.
`__log10f(x)`	For `x` in [0.5, 2], the maximum absolute error is 2^{-24}, otherwise, the maximum ulp error is 3.
`__sinf(x)`	For `x` in $[-\pi, \pi]$, the maximum absolute error is $2^{-21.41}$, and larger otherwise.
`__cosf(x)`	For `x` in $[-\pi, \pi]$, the maximum absolute error is $2^{-21.19}$, and larger otherwise.
`__sincosf(x,sptr,cptr)`	Same as `sinf(x)` and `cosf(x)`.
`__tanf(x)`	Derived from its implementation as `__sinf(x) * (1 / __cosf(x))`.
`__powf(x, y)`	Derived from its implementation as `exp2f(y * __log2f(x))`.
`__mul24(x,y)` `__umul24(x,y)`	N/A
`__mulhi(x,y)` `__umulhi(x,y)`	N/A
`__int_as_float(x)`	N/A
`__float_as_int(x)`	N/A
`__saturate(x)`	N/A
`__sad(x,y,z)` `__usad(x,y,z)`	N/A
`__clz(x)`	N/A
`__ffs(x)`	N/A
`__float2int_[rn,rz,ru,rd]`	N/A
`__float2uint_[rn,rz,ru,rd]`	N/A
`__int2float_[rn,rz,ru,rd]`	N/A
`__uint2float_[rn,rz,ru,rd]`	N/A

Σχ. 3.5 – Συναρτήσεις Μονής Ακρίβειας Συσκευής **[12].**

3.5 Εικονική Λειτουργία

Όπως είδαμε μέχρι τώρα οι εντολές και οι συναρτήσεις χωρίζονται σε δύο κατηγορίες. Σε αυτές που εκτελούνται στον επεξεργαστή και σε αυτές που εκτελούνται στη συσκευή. Επομένως. είναι ξεκάθαρο ότι ένα πρόγραμμα που δημιουργήθηκε και μεταγλωττίστηκε για να εκτελείται στη συσκευή δεν έχει τη δυνατότητα να εκτελεστεί στον επεξεργαστή και αντίστροφα. Αυτό όμως συνεπάγεται ότι χρήσιμες εντολές για την εποπτεία κατά τη δημιουργία ενός προγράμματος συσκευής δεν μπορούν να χρησιμοποιηθούν. Χαρακτηριστική εντολή είναι η **printf()** με την οποία εμφανίζονται τα αποτελέσματα κάποιας μεταβλητής στην οθόνη. Κατά την εκτέλεση στην κάρτα αυτή η εντολή του host δεν

μπορεί να χρησιμοποιηθεί και επομένως ο χρήστης δεν μπορεί να επιβλέπει την ορθότητα των προγραμματιζόμενων εντολών.

Γι'αυτό μια πολύ χρήσιμη δυνατότητα που μας παρέχει η CUDA είναι η χρήση εικονικής λειτουργίας (λειτουργία με προσομοίωση) δηλαδή device emulation.

Emulation είναι η διαδικασία με την οποία ένα πρόγραμμα, το οποίο προορίζεται αρχικά για εκτέλεση στην συσκευή, θα μεταγλωττιστεί με χρήση της πρόσθετης εντολής **–deviceemu** και έτσι του δίνεται η οδηγία να εκτελεστεί στον επεξεργαστή. Στην ουσία έχουμε προσομοίωση της λειτουργίας σε ένα διαφορετικό μέσο από αυτό που προοριζόταν.

Έτσι κάθε φορά που μια εφαρμογή καλείται να εκτελεστεί σε emulation, έχουμε προσομοίωση του προγραμματιστικού του μοντέλου. Για κάθε thread του κάθε block του, δημιουργείται ένα αντίστοιχο thread στον επεξεργαστή. Αυτό όμως εισάγει περιορισμούς που πρέπει να είναι γνωστοί στον προγραμματιστή πριν την εφαρμογή αυτής της λειτουργίας, δηλαδή:

⇨Ο επεξεργαστής πρέπει να μπορεί να λειτουργήσει με το μέγιστο αριθμό threads για κάθε block.

⇨Πρέπει να υπάρχει αρκετή μνήμη στον επεξεργαστή για την εκτέλεση όλων των threads με δεδομένο ότι καθένα από αυτά χρησιμοποιεί 256KB μνήμης.

Η χρήση της συγκεκριμένης λειτουργίας εισάγει τα παρακάτω μειονεκτήματα και πλεονεκτήματα :

⇨Κύριο πλεονέκτημα είναι η δυνατότητα χρήσης εντολών και λειτουργιών του επεξεργαστή σε κώδικα που αναφέρεται σε εκτέλεση συσκευής. Κύρια κατηγορία τέτοιων εντολών είναι οι εντολές εξόδου εισόδου δεδομένων στην οθόνη του υπολογιστή, τις οποίες επιτρέπεται πλέον να τις χρησιμοποιήσει ο χρήστης για να επιβλέπει τα ενδιάμεσα αποτελέσματα που προκύπτουν από το πρόγραμμα του και να ελέγχει εύκολα και άμεσα την ορθότητα τους.

⇨Επίσης, η εικονική λειτουργία δίνει τη δυνατότητα σε χρήστες που δεν έχουν κατάλληλη κάρτα γραφικών συμβατή με την τεχνολογία CUDA, να προγραμματίσουν τα προγράμματα τους σε αυτήν και να τα τρέξουν εικονικά στον επεξεργαστή του υπολογιστή τους.

⇨Μειονέκτημα της διαδικασίας, αποτελεί όπως είναι αναμενόμενο το γεγονός ότι εκμηδενίζονται τα πλεονεκτήματα της παραλληλίας που εισάγει η CUDA. Η εκτέλεση ενός προγράμματος σχεδιασμένου για παράλληλη εκτέλεση στους 240 πυρήνες μιας κάρτας γραφικών , σε έναν επεξεργαστή

ενός , δύο ή τεσσάρων πυρήνων συνεπάγεται με δραματική αύξηση του χρόνου εκτέλεσης του. Όσο πιο καλοσχεδιασμένο είναι ένα πρόγραμμα για εκτέλεση σε συσκευή, τόσο μεγαλύτερη θα είναι η επιβράδυνση με εκτέλεση σε επεξεργαστή.

⇨Επίσης τα μειονεκτήματα δεν αφορούν μόνο τον χρόνο εκτέλεσης αλλά και την ακρίβεια των αποτελεσμάτων που παράγονται από το πρόγραμμα. Συγκεκριμένα, οι μαθηματικές συναρτήσεις της συσκευής, όταν με χρήση emulation εκτελεστούν στον επεξεργαστή, ενδέχεται να μην δώσουν ακριβώς τα ίδια αποτελέσματα. Γεγονός κατανοητό αν σκεφτούμε τις διαφορές που υπάρχουν σε συσκευή και επεξεργαστή όσον αφορά τις διαφορές στη διαδικασία μεταγλώττισης, στις διαφορετικές οδηγίες και στη διαφορετική αρχιτεκτονική των δύο μέσων.

Έτσι από τα παραπάνω συμπεραίνουμε ότι η λειτουργία σε emulation είναι ένα πολύ χρήσιμο εργαλείο κατά το στάδιο της δημιουργίας του προγράμματος μας, όπου χρειάζεται όσο το δυνατόν καλύτερη εποπτεία, αλλά δεν θα μπορούσε να χρησιμοποιηθεί για την κανονική εκτέλεση ενός προγράμματος λόγω των μειονεκτημάτων της στο χρόνο εκτέλεσης και της ακρίβειας των αποτελεσμάτων.

3.6 Παράδειγμα

Ακολουθεί ένα παράδειγμα μιας απλής εφαρμογής , της πρόσθεσης 2 πινάκων μαζί με επεξήγηση των βασικών βημάτων που ακολουθούνται για την κατασκευή ενός τέτοιου προγράμματος .

Τμήμα 1ο

```
const int N = 1024;
const int blocksize = 16;
```

Τμήμα 2ο

```
__global__void add_matrix( float* a, float *b, float *c, int N )
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;
   int index = i + j*N;
   if ( i < N && j < N )
      c[index] = a[index] + b[index];
}
```

Τμήμα 3ο

```
int main() {
   float *a = new float[N*N];
   float *b = new float[N*N];
   float *c = new float[N*N];
   for ( int i = 0; i < N*N; ++i ) {
```

```
    a[i] = 1.0f; b[i] = 3.5f; }
```

Τμήμα 4ο

```
  float *ad, *bd, *cd;
  const int size = N*N*sizeof(float);
  cudaMalloc( (void**)&ad, size );
  cudaMalloc( (void**)&bd, size );
  cudaMalloc( (void**)&cd, size );
```

Τμήμα 5ο

```
  cudaMemcpy( ad, a, size, cudaMemcpyHostToDevice );
  cudaMemcpy( bd, b, size, cudaMemcpyHostToDevice );
```

Τμήμα 6ο

```
  dim3 dimBlock( blocksize, blocksize );
  dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
  add_matrix<<<dimGrid, dimBlock>>>( ad, bd, cd, N );
```

Τμήμα 7ο

```
  cudaMemcpy( c, cd, size, cudaMemcpyDeviceToHost );
```

Τμήμα 8ο

```
  cudaFree( ad ); cudaFree( bd ); cudaFree( cd );
  delete[] a; delete[] b; delete[] c;
  return EXIT_SUCCESS;
}
```

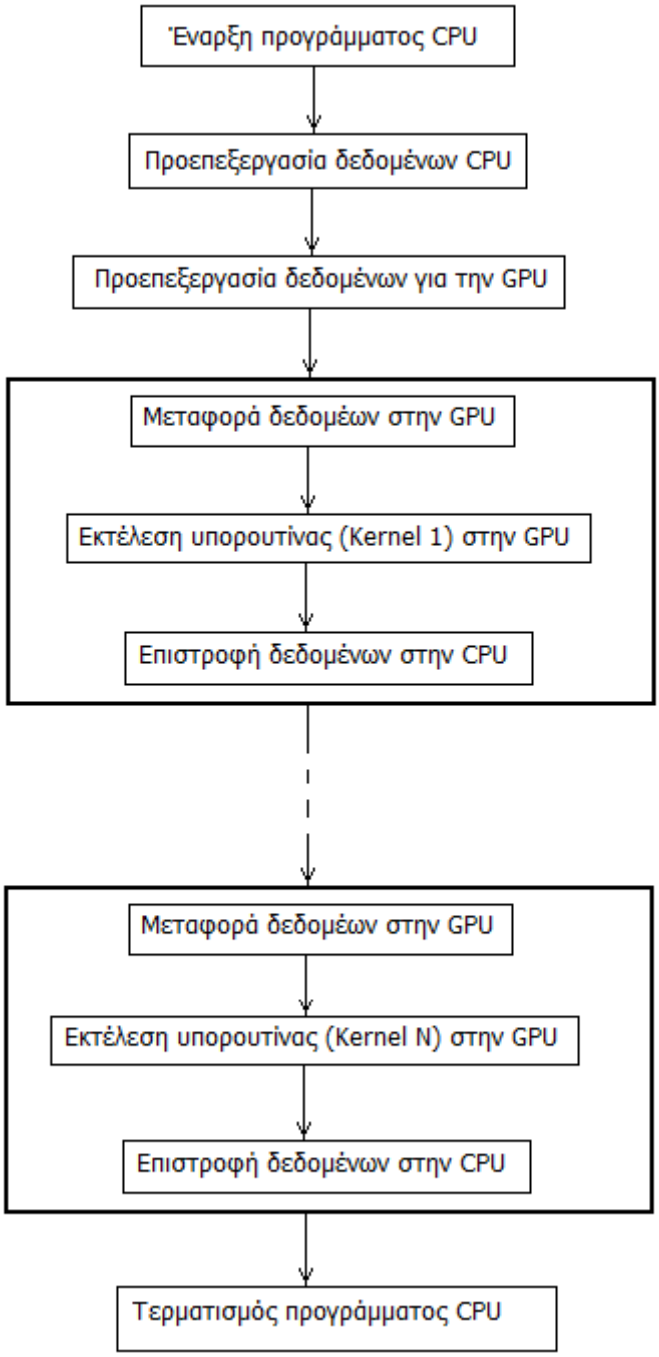
Ανάλυση

Τα βήματα που ακολουθούν είναι θεμελιώδη και υπάρχουν σε κάθε πρόγραμμα γραμμένο σε CUDA .

Αναλυτικά:

⇨Τμήμα 1^{ο}:Ορίζεται το μέγεθος του προβλήματος. Το μέγεθος του προβλήματος μπορεί να έχει οποιαδήποτε τιμή, ενώ το μέγεθος του block θα πρέπει να έχει τέτοια τιμή ώστε τα thread στο εσωτερικό του να μην ξεπερνάμε τα 512.
⇨Τμήμα 2^{ο}:Υπορουτίνα που εκτελεί τους υπολογισμούς στην συσκευή (χρήση **__global__** προσδιοριστικού).
⇨Τμήμα 3^{ο}:Καταχώρηση μνήμης στον επεξεργαστή
⇨Τμήμα 4^{ο}:Καταχώρηση μνήμης στη συσκευή με βάση το μέγεθος του προβλήματος που είχαμε ορίσει (**cudaMalloc()**)
⇨Τμήμα 5^{ο}:Αντιγραφή δεδομένων από τον επεξεργαστή στην συσκευή (**cudaMemcpy()**)
⇨Τμήμα 6^{ο}:Εκτέλεση υπορουτίνας υπολογισμών (**add_matrix<<<dimGrid, dimBlock>>>()**)
⇨Τμήμα 7^{ο}:Αντιγραφή αποτελεσμάτων πίσω από τη συσκευή στον επεξεργαστή (**cudaMemcpy()**)
⇨Τμήμα 8^{ο}:Άδειασμα μνήμης στη συσκευή και στον επεξεργαστή και τερματισμός προγράμματος (**cudaFree( ) , delete[]**)

Ακολουθεί το γενικότερο διάγραμμα ροής της εκτέλεσης μιας εφαρμογής στην κάρτα γραφικών.



Σχ. 3.6 – Διάγραμμα ροής γενικής εφαρμογής

Κεφάλαιο 4ο
Επίλυση της Ροής

Εισαγωγή :

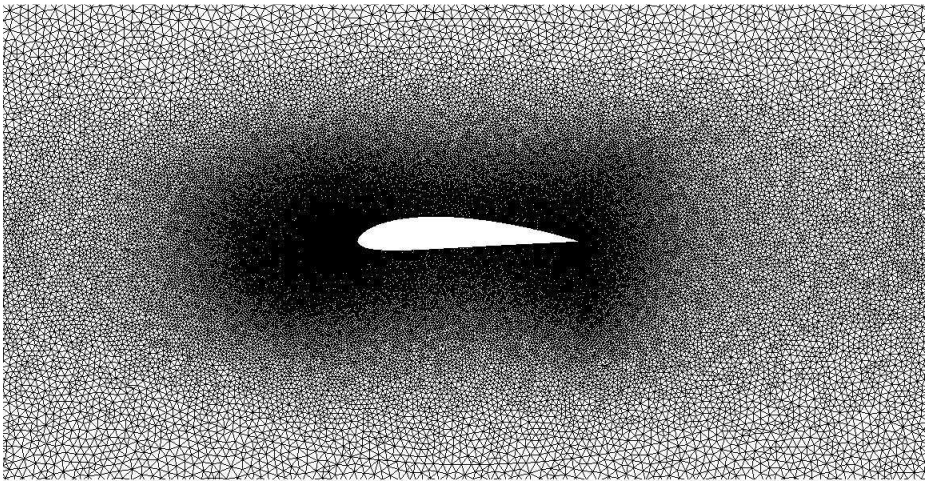
Η πρώτη εφαρμογή στην κάρτα γραφικών έχει σχέση με την επίλυση της ροής γύρω από μία μεμονωμένη αεροτομή.

Συγκεκριμένα, όπως αναφέρθηκε στην εισαγωγή, στο πλαίσιο αυτής της διπλωματικής, δεν έγινε ο προγραμματισμός του επιλύτη ροής Navier-Stokes[13] σε CUDA, αλλά το πρόγραμμα δόθηκε έτοιμο για χρήση από ερευνητές του Εργαστήριου Θερμικών Στροβιλομηχανών του ΕΜΠ **[5].**

Ωστόσο, επιλέγεται μια χαρακτηριστική εφαρμογή με μεγάλο υπολογιστικό κόστος στην οποία θα κάνουμε χρήση του παραπάνω προγράμματος για να μελετήσουμε την επίδραση στο χρόνο εκτέλεσης, της νέας τεχνολογίας.

4.1 Εφαρμογή

Συγκεκριμένα, στόχος μας είναι η επίλυση της ροής (προφανώς με υπολογισμούς διπλής ακρίβειας) γύρω από αεροτομή, σε διάφορες γωνίες τυρβώδους ροής, με σκοπό την εύρεση της γωνίας απώλειας στήριξης μέσω δημιουργίας της πολικής καμπύλης της αεροτομής. Για το σκοπό αυτό επιλέγεται η NACA4415, ενώ θα μπορούσε να είχε επιλεγεί οποιαδήποτε αεροτομή, ακόμα και υπερηχητικής ροής. Επίσης επιλέγεται αριθμός Mach=0.5 και Reynolds=$9 \cdot 10^5$, ενώ χρησιμοποιείται μη-δομημένο πλέγμα 113.888 κόμβων και 226.825 τριγώνων.



Σχ. 4.1 – Χρησιμοποιηθέν υπολογιστικό πλέγμα σε μεγέθυνση

[13] Χρησιμοποιήθηκε το μοντέλο Spalart–Allmaras για την μοντελοποίηση της τύρβης και MPA για την ακρίβεια των πράξεων **[25].**

4.2 Αποτελέσματα

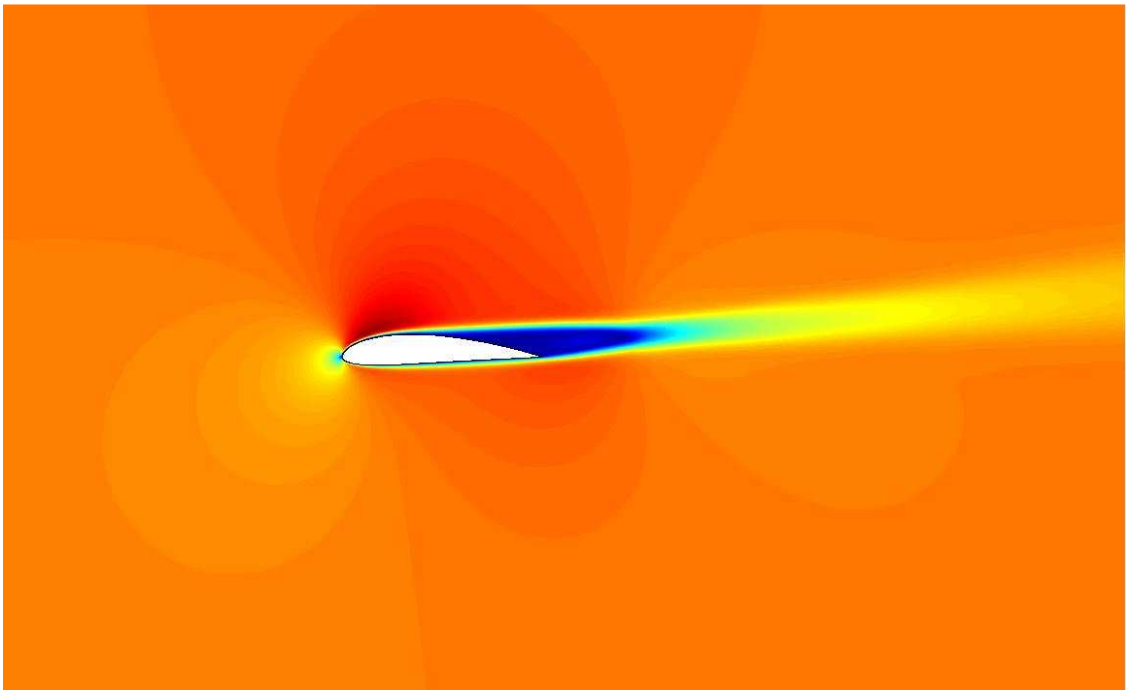
4.2.1 Γωνία απώλειας στήριξης

Ύστερα από εκτέλεση για γωνίες ροής 0°, 1°, 2°, 3°, 4°, 5°, 6°, 7°, 7.1°, 7.2°, 7.5° παρατηρήθηκε αποκόλληση στις a_{max}=7.1° (μέγιστη γωνία)

Ακολουθούν πινακοποιημένα τα αποτελέσματα:

Γωνία Ροής (°)	Συντελεστής Άνωσης
0.0	-0.0263491
1.0	0.0317805
2.0	0.0916066
3.0	0.152281
4.0	0.208282
5.0	0.26301
6.0	0.319858
7.0	0.395195
7.1	0.405517
7.2	0.401039
7.5	0.290137

Στο σχήμα που ακολουθεί παραθέτουμε εικόνα[14] της ροής στις a_{max} μοίρες.

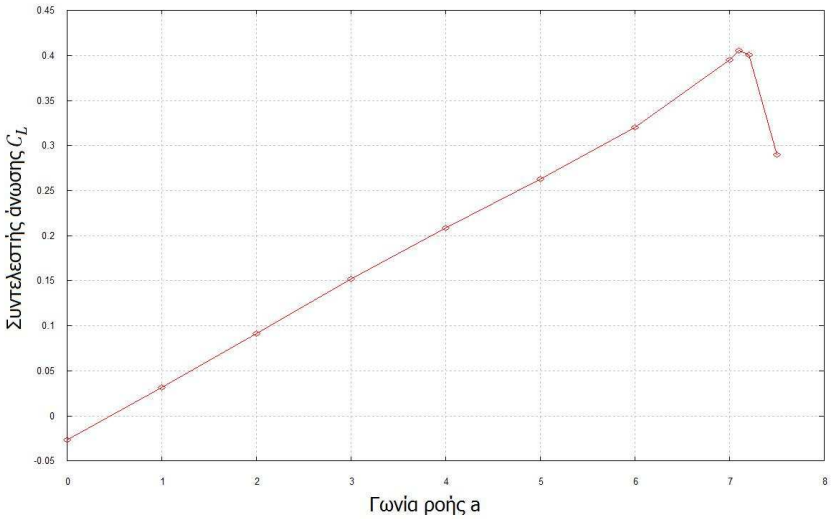


Σχ. 4.2 – Εικόνα ροής κατά την απώλεια στήριξης στις 7.1°

[14] Για την απεικόνιση χρησιμοποιήθηκε το λογισμικό Mesh του ΕΘΣ του ΕΜΠ

4.2.2 Πολική καμπύλη

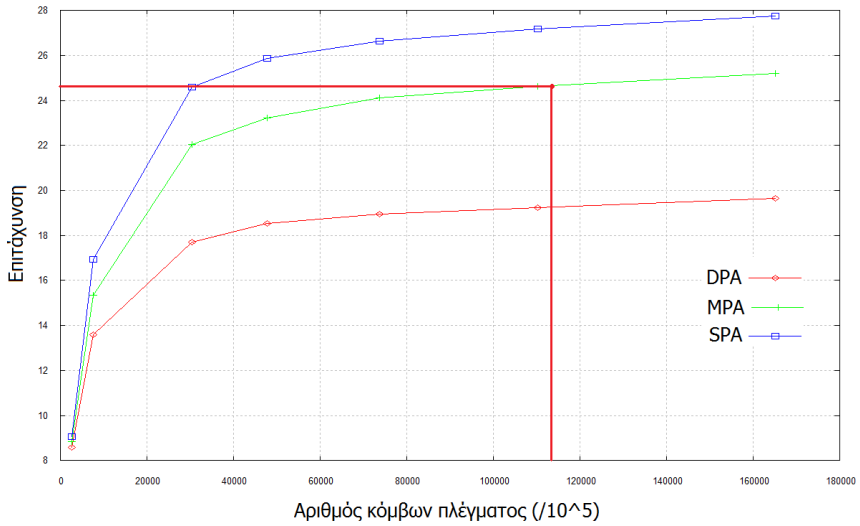
Στο παρακάτω σχήμα ακολουθεί η πολική καμπύλη της αεροτομής.



Σχ. 4.3 – Πολική καμπύλη

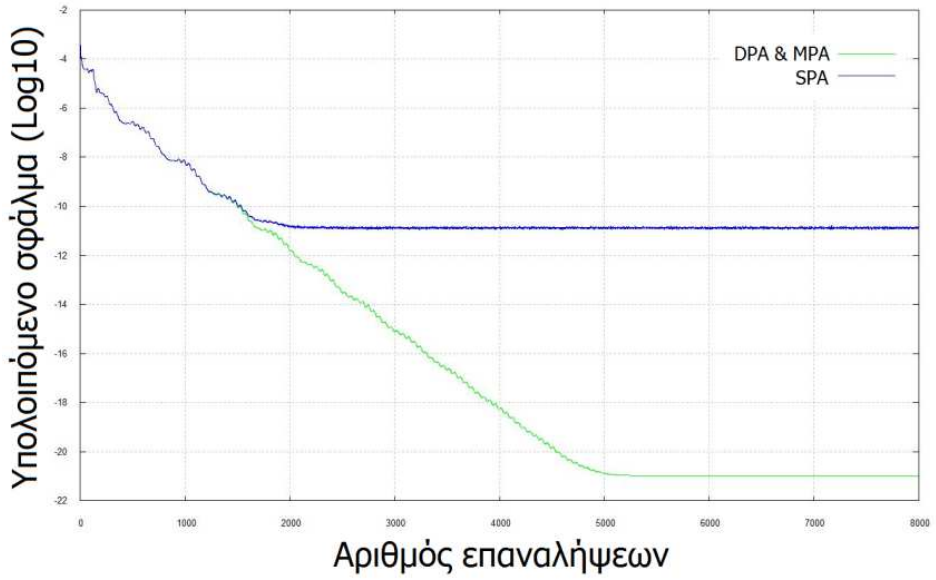
4.2.3 Χρόνος εκτέλεσης

Για τη μέγιστη γωνία a_{max}, ο χρόνος εκτέλεσης στην κάρτα γραφικών GTX285 είναι: 3903sec, ενώ με εκτέλεση στον κεντρικό επεξεργαστή ο χρόνος εκτέλεσης είναι: 93674sec**.** Επομένως παρατηρείται επιτάχυνση x24 ενώ τα αποτελέσματα έχουν την ίδια ακρίβεια με τα αντίστοιχα στον κεντρικό επεξεργαστή. Ίδιο ποσοστό επιτάχυνσης παρατηρείται και για τη συνολική διαδικασία δημιουργίας της πολικής καμπύλης. Τα αποτελέσματα ήταν συμβατά με τα αποτελέσματα του Εργαστηρίου Θερμικών Στροβιλομηχανών του ΕΜΠ πάνω στην ίδια αεροτομή με τις ίδιες συνθήκες ροής. Παραθέτουμε ενδεικτικά το παρακάτω σχήμα από την εργασία **[5]** .



Σχ. 4.4 – Επιτάχυνση για πράξεις DPA, MPA, SPA (αναλύθηκαν στην εισαγωγή) . Με κόκκινο έχει τονιστεί το μέγεθος του πλέγματος μας και η επιτευχθείσα επιτάχυνση **[5]**.

Επίσης να τονιστεί ότι λόγο χρήσης «μικτής»[15] ακρίβειας στις πράξεις, τα αποτελέσματα παρουσιάζουν την ίδια ακρίβεια με τα αντίστοιχα του κεντρικού επεξεργαστή. Επιπλέον, με χρήση της «μεικτής» ακρίβειας επιτυγχάνουμε την επιτάχυνση που θα είχαμε με χρήση μονής ακρίβειας. Χαρακτηριστικά παρατίθεται διάγραμμα από την μελέτη του εργαστηρίου στο οποίο παρουσιάζεται ο απαιτούμενος αριθμός επαναλήψεων που χρειάζεται μέχρι να οδηγηθούμε στην σύγκλιση , συναρτήσει της ακρίβειας των πράξεων.

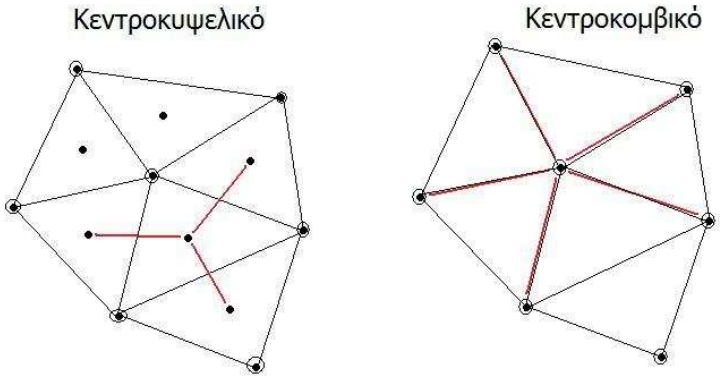


Σχ. 4.5 – Υπολειπόμενο σφάλμα σε σχέση με τον αριθμό των επαναλήψεων **[5].**

Βλέπουμε λοιπόν ότι χρησιμοποιήθηκε η βέλτιστη δυνατή επιλογή, με την οποία εκμεταλλευόμαστε το πλεονέκτημα της ακρίβειας της διπλής και το πλεονέκτημα της ταχύτητας της μονής ακρίβειας.

Τέλος, θα πρέπει να τονιστεί ότι σε αντίθεση με τους κώδικες επίλυσης άλλων ελαχίστων εργαστηρίων που χρησιμοποιούν κεντροκυψελικό μοντέλο για την επεξεργασία των δεδομένων, το πρόγραμμα που χρησιμοποιήθηκε χρησιμοποιεί ένα νέο κεντροκομβικό μη-δομημένο μοντέλο. Χαρακτηριστικά παρουσιάζεται το παρακάτω σχήμα:

[15] Κατά την αριθμητική επίλυση γραμμικών πινάκων η τρέχουσα λύση Φ^n και η λύση της επόμενης επανάληψης Φ^{n+1} συνδέονται με τη σχέση: $\Phi^n + \Delta\Phi \rightarrow \Phi^{n+1}$. Η εύρεση του βήματος ΔΦ προκύπτει από την επίλυση του συστήματος $A \cdot \Delta\Phi = b - A \cdot \Phi^n$. Σε αυτήν την εξίσωση το πρώτο μέλος αποτελεί το υπολογιστικά πιο δαπανηρό κομμάτι (παραγοντοποίηση, εμπρός και πίσω αντικατάσταση του πίνακα Α) , ενώ το δεύτερο είναι ελαφρύ υπολογιστικά, αλλά απαιτεί μεγαλύτερη ακρίβεια επειδή εμπεριέχει τον υπολογισμό του υπολειπόμενου σφάλματος (residual). Επομένως με εφαρμογή μονής ακρίβειας στο πρώτο κομμάτι και διπλής στο δεύτερο, δηλαδή συνολικά μεικτής, εκμεταλλευόμαστε το πλεονέκτημα της διπλής στην ακρίβεια και της μονής στο χρόνο εκτέλεσης.



Σχ. 4.6 – Κεντροκυψελικό και κεντροκομβικό μοντέλο

Στο κεντροκυψελικό μοντέλο, τα δεδομένα που επεξεργάζονται είναι του «εικονικού» κόμβου που προκύπτει στο κέντρο βάρους κάθε τριγωνικού στοιχείου ο οποίος αποτελεί συνάρτηση των τριών κορυφών του. Κάθε πληροφορία που θα χρειαστεί για αυτό το σημείο προκύπτει με ανάγνωση μνήμης των κεντροβαρικών σημείων των τριών γειτονικών τριγώνων.

Αντίθετα, στο κεντροκομβικό μοντέλο, κάθε πληροφορία που θα χρειαστεί για το σημείο που επεξεργάζεται, ναι μεν προκύπτει άμεσα από τους γειτονικούς κόμβους χωρίς την ανάγκη υπολογισμού κέντρου βάρους, αλλά συνεπάγεται με τόσες αναγνώσεις μνήμης όσο και το πλήθος των γειτονικών στοιχείων, το πλήθος των οποίων δεν είναι σταθερό. Τέτοιου είδους αναγνώσεις μνήμης εισάγουν πρόσθετο υπολογιστικό κόστος.

Επομένως με την χρήση τέτοιου μοντέλου, εισάγεται μια σημαντική πρόσθετη επιβράδυνση στην εκτέλεση του κώδικα, γεγονός που κάνει το μοντέλο ικανό για συγκεκριμένο εύρος εφαρμογών, αλλά πιο αργό σε σχέση με τις δυνατότητες του κεντροκυψελικού μοντέλου.

Κεφάλαιο 5ο
Το Μοντέλο του Αεροσκάφους

Εισαγωγή :

Το παρακάτω μοντέλο **[2]**, προκαταρκτικού σχεδιασμού υπερηχητικού αεροσκάφους, που θα παρουσιαστεί επαναπρογραμματίστηκε σε CUDA. Συγκεκριμένα η κύρια δομή που παρουσιάζεται παρακάτω εκτελείται στον επεξεργαστή του υπολογιστή ενώ οι επιμέρους υπολογισμοί γίνονται καλώντας υπορουτίνες οι οποίες εκτελούνται στην κάρτα γραφικών.

5.1 Λεκτική διατύπωση του προβλήματος

Στόχος μας είναι ο σχεδιασμός ενός μικρού υπερηχητικού αεροσκάφους (Small Business Jet) με δυνατότητα μεταφοράς έως 16 ατόμων, το οποίο θα είναι ικανό να εκτελεί υπερατλαντικές πτήσεις. Μικρό αεροσκάφος σημαίνει αεροσκάφος μικρού σχετικά μεγέθους και βάρους.

Ο στόχος αυτός εισάγει πολύπλευρες σχεδιαστικές δυσκολίες καθώς ο προφανής τρόπος μείωσης του βάρους είναι η μείωση της ποσότητας των καυσίμων του αεροσκάφους, γεγονός που έρχεται σε αντίθεση με τον στόχο της μεγάλης εμβέλειας που θα θέσουμε στη συνέχεια.

Επίσης επιβάλλονται περιορισμοί σχετικά με το μήκος απογείωσης αλλά και την ταχύτητα προσέγγισης η οποία εμμέσως καθορίζει και το απαιτούμενο μήκος απογείωσης, με σκοπό τα αποτελέσματα μας να προσεγγίζουν όσο γίνεται αληθινά δεδομένα.

5.2 Μεταβλητές σχεδιασμού

Το μοντέλο του αεροσκάφους που θα παρουσιαστεί αποτελεί μια συνάρτηση πολλών μεταβλητών. Τα ορίσματα αυτής της συνάρτησης ονομάζονται μεταβλητές σχεδιασμού (design variables) και θα μας απασχολήσουν ευθύς αμέσως.

Η μεταβλητή σχεδιασμού μπορεί να είναι οποιοδήποτε μέγεθος από κάποιο γεωμετρικό ή αεροδυναμικό χαρακτηριστικό μέχρι και το βάρος του καυσίμου του αεροσκάφους ή την ταχύτητα πτήσης του. Βέβαια, είναι προφανές ότι είναι σημαντική η προσεκτική επιλογή των μεταβλητών αυτών έτσι ώστε να πληρούν δύο βασικές προϋποθέσεις:

⇨Να επηρεάζουν τις επιδόσεις του αεροσκάφους αρκετά ώστε να αξίζει το πρόσθετο υπολογιστικό κόστος της διερεύνησης της μεταβολής της τιμής τους.

⇨Να παραμετροποιούν όσο το δυνατόν περισσότερα σημαντικά, για τις επιδόσεις, μέρη του αεροσκάφους (όπως η πτέρυγα, το ουραίο πτερύγιο και η άτρακτος)

Έτσι, έχοντας υπόψη τα παραπάνω, με βάση τον αλγόριθμο επίλυσης που θα χρησιμοποιηθεί, επιλέγονται οι παρακάτω μεταβλητές σχεδιασμού που παραμετροποιούν τα πρόβλημα μας:

⇨Για την ευθεία πτήση (δείκτης cr {cruise}):

M_{cr}**{**Mach cruise}	αριθμού Mach
h_{cr}	υψομέτρου πτήσεως
$alfa$	γωνίας πρόσπτωσης κατά την ευθεία πτήση

⇨Για την άτρακτο (δείκτης fus {fuselage}):

D_f**{**fuselage Diameter}	η διάμετρος της ατράκτου

⇨ Για την πτέρυγα (δείκτης w {wing}):

S_w	επιφάνειά της σε κάτοψη
Λ_{LE_w}**{**Leading Edge Sweep Angle}	η γωνία οπισθόκλισης στην ακμή προσβολής
AR_w**{**Wing Aspect Ratio}	ο λόγος επιμήκους της πτέρυγας
λ_w**{**Wing Taper Ratio}	η εκλέπτυνση
t/c_w**{**Wing Relative Thickness}	ο λόγος μέγιστου πάχους προς χορδή

⇨Για το ουραίο πτερύγιο (δείκτης v {vertical wing}):

Λ_{LE_v} {Leading Edge Sweep Angle}	η γωνία οπισθόκλισης στην ακμή προσβολής
AR_v ({Vertical Wing Aspect Ratio}	ο λόγος επιμήκους
λ_v {Vertical Wing Taper Ratio}	η εκλέπτυνση
t/c_v**{**Vertical Wing Relative Thickness}	ο λόγος μέγιστου πάχους προς χορδή

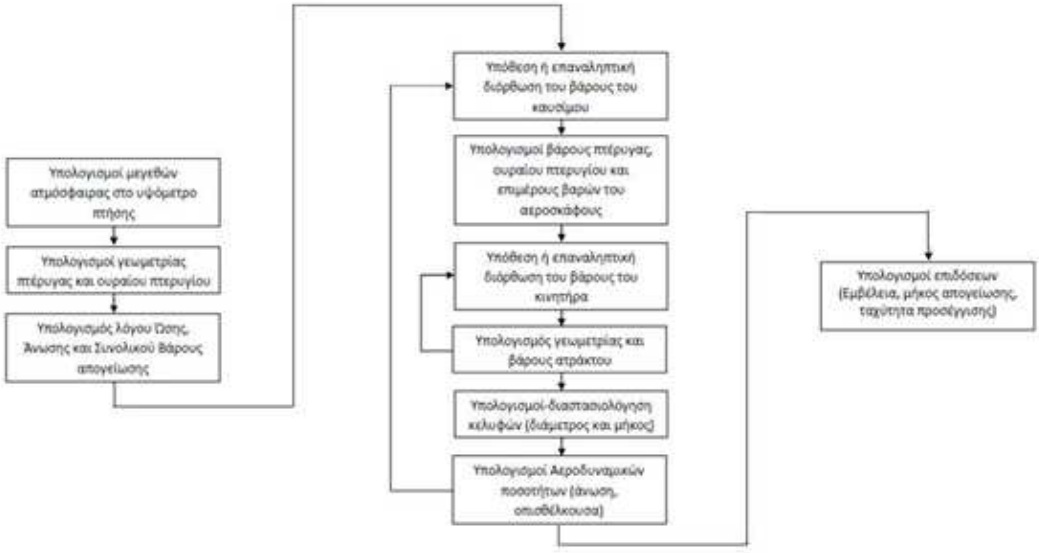
⇨ Για τον έλεγχο του μήκους απογείωσης και της ταχύτητας προσέγγισης:

N_{ult}**{**Ultimate Load Factor}	ο “απόλυτος” συντελεστής φόρτισης

Συνολικά λοιπόν επιλέγονται 14 μεταβλητές σχεδιασμού. Κατά τη διαδικασία της βελτιστοποίησης (ανάλυση στο επόμενο κεφάλαιο), θα αναζητηθεί ο συνδυασμός των μεταβλητών αυτών που εξασφαλίζουν τη βέλτιστη επίδοση του αεροσκάφους.

5.3 Αλγόριθμος Επίλυσης

Ακολουθεί ο αλγόριθμος επίλυσης του μοντέλου του προκαταρκτικού σχεδιασμού του αεροσκάφους σε μορφή διαγράμματος ροής και στην συνέχεια επεξήγηση του:



Σχ. 5.2 – Αλγόριθμος Επίλυσης

Όπως εικονίζεται και στο παραπάνω σχήμα, αρχικά υπολογίζουμε τα χαρακτηριστικά της τυπικής ατμόσφαιρας στο συγκεκριμένο υψόμετρο ευθείας πτήσης του αεροσκάφους.

Στη συνέχεια, υπολογίζονται τα γεωμετρικά χαρακτηριστικά της ατράκτου (μήκος, διάμετρος και επιφάνεια), της πτέρυγας (εκτιθέμενης επιφάνεια, εύρος {span} και κατασκευαστικές γωνίες) και του ουραίου πτερυγίου.

Υπάρχουν πλέον τα δεδομένα για να υπολογισθεί ο λόγος ώσης F/Fo όπου Fo η ώση στο επίπεδο της θάλασσας και F η ώση στο υψόμετρο ευθείας πτήσης.

Σε αυτό το σημείο, με γνωστή τη γωνία πρόσπτωσης, την ταχύτητα πτήσης και το ύψος πτήσης με όλα τα χαρακτηριστικά της ατμόσφαιρας σε αυτό το ύψος, με χρήση του επιλύτη ροής για τη γνωστή πτέρυγα του αεροσκάφους μας, υπολογίζουμε εξωτερικά από τον αλγόριθμο, την άνωση του αεροσκάφους και στην συνέχεια την εισάγουμε και τη χρησιμοποιούμε για τον υπολογισμό του συνολικού βάρους απογείωσης (TOW).

Στο σημείο αυτό γίνεται υπόθεση του βάρους του καυσίμου και εκκινείται επαναληπτική διαδικασία για τον υπολογισμό του.

Το επόμενο βήμα είναι να υπολογισθούν τα βάρη της πτέρυγας και του ουραίου πτερυγίου. Το βάρος της ατράκτου δεν είναι γνωστό ακόμη καθώς, όπως

αναφέρεται σε επόμενη παράγραφο, στο βάρος της ατράκτου συνυπολογίζεται και το βάρος του καυσίμου.

Το βάρος της ατράκτου περιλαμβάνει, όπως προελέχθη, το βάρος του καυσίμου αλλά είναι και συνάρτηση του βάρους του κινητήρα, το οποίο είναι ακόμη άγνωστο. Στο σημείο αυτό λοιπόν εκκινείται ο δεύτερος επαναληπτικός βρόχος στον οποίο γίνεται υπόθεση και διόρθωση του βάρους του κινητήρα.

Αφού γίνει η υπόθεση, υπολογίζεται η γεωμετρία και το βάρος της ατράκτου. Όμως, όπως ανεφέρθη και στον προηγούμενο αλγόριθμο, το συνολικό βάρος απογείωσης του αεροσκάφους είναι το άθροισμα των βαρών των επιμέρους δομικών στοιχείων του (πτέρυγα, άτρακτος κλπ.) και του βάρους των καυσίμων. Έτσι, υπολογίζεται εκ νέου το βάρος του κινητήρα ως η διαφορά του Take-off Weight από τα υπόλοιπα βάρη μέχρι σύγκλισης της διαδικασίας.

Με το βάρος του κινητήρα γνωστό, δύνανται πλέον να υπολογισθούν η ώση, οι διαστάσεις και το κέλυφος του κινητήρα.

Ακολουθούν οι αεροδυναμικοί υπολογισμοί κατά τους οποίους βρίσκεται ο συνολικός συντελεστής αντίστασης του αεροσκάφους. Έτσι, υπολογίζεται εκ νέου η ώση, από την καινούργια ώση το νέο βάρος του κινητήρα και το νέο βάρος των καυσίμων είναι η διαφορά του Take-off Weight από τα υπόλοιπα βάρη.

Μετά τη σύγκλιση του βάρους των καυσίμων είναι γνωστά όλα τα απαραίτητα δεδομένα για τον υπολογισμό των επιδόσεων του αεροσκάφους. Τα μεγέθη που θα χαρακτηρίζουν τις επιδόσεις του αεροσκάφους είναι 4 και είναι τα εξής:

- ⇨Η εμβέλεια του αεροσκάφους
- ⇨Το βάρος καυσίμου
- ⇨Το μήκος απογείωσης του αεροδιαδρόμου
- ⇨Η ταχύτητα προσέγγισης στον αεροδιάδρομο

5.4 Ανάλυση του μοντέλου του αεροσκάφους

Οι υπολογισμοί τόσο στο κύριο αλγόριθμο όσο και στις υπορουτίνες γίνονται με βάση εμπειρικούς τύπους και παρατίθενται αναλυτικά στις ενότητες που ακολουθούν με βάση την κατηγορία τους.

5.4.1 Τυπική Ατμόσφαιρα

Η μοντελοποίηση της ατμόσφαιρας ξεκινά με την προϋπόθεση ότι ο αέρας συμπεριφέρεται ως τέλειο αέριο και για τον οποίον ισχύει η καταστατική εξίσωση των τέλειων αερίων:

$$p = \rho RT \qquad (1.1)$$

Όπου p η πίεση, ρ η πυκνότητα, T η θερμοκρασία και R η σταθερά των τελείων αερίων $R = 287.04 \frac{J}{Kg \cdot K}$

Η γήινη ατμόσφαιρα χωρίζεται σε πέντε ζώνες ανάλογα με το υψόμετρο. Έτσι έχουμε τις εξής ζώνες:

⇨*Τροπόσφαιρα (έως 20 χιλιόμετρα)*

⇨Στρατόσφαιρα (έως 55 χιλιόμετρα)

⇨Μεσόσφαιρα (έως 80 χιλιόμετρα)

⇨Θερμόσφαιρα (έως 800 χιλιόμετρα)

⇨Εξώσφαιρα (μεγαλύτερο από 800 χιλιόμετρα)

Επιπλέον τα υπερηχητικά αεροσκάφη ταξιδεύουν σε μεγάλα υψόμετρα για δύο λόγους:

⇨Διεθνείς κανονισμοί για τη στάθμη του εκπεμπόμενου θορύβου.

⇨Εκτελούν πτήσεις μεγάλων αποστάσεων (μεγαλύτερη οικονομία καυσίμου.

Συγκεκριμένα, εμάς θα απασχολήσουν οι περιοχές της τροπόσφαιρας και της στρατόσφαιρας στις οποίες θα κινηθεί το αεροσκάφος μας.

Αναλυτικά θεωρούμε ότι η γήινη τροπόσφαιρα εκτείνεται έως υψόμετρο 11000m. όπου η θερμοκρασία της είναι γραμμική συνάρτηση του ύψους. Δηλαδή:

$$T = T_o - B_z \quad (1.2)$$

Όπου $T_o = 288.15\, K$ η απόλυτη θερμοκρασία στο επίπεδο της θάλασσας και $B = -\frac{dT}{dz} = 0.0065\, \frac{K}{m}$ είναι ο ρυθμός της μεταβολής της θερμοκρασίας με το υψόμετρο. Δεδομένου ότι η μεταβολή της πίεσης με το υψόμετρο δίνεται από τη σχέση:

$$\frac{dp}{dz} = -\rho g \qquad (1.3)$$

Προκύπτει ότι:

$$p = p_o(1 - \frac{B_z}{T_o})^{\frac{g}{RB}} = p_o(\frac{T}{T_o})^{\frac{g}{RB}} \quad (1.4)$$

Όπου $p_o = 101325\,Pa$ είναι η πίεση στο επίπεδο της θάλασσας και $g = 9.80665\ m/s^2$ η επιτάχυνση της βαρύτητας. Ο λόγος της πυκνότητας προς την πυκνότητα στο επίπεδο της θάλασσας είναι :

$$\sigma = \frac{\rho}{\rho_o} = \left(\frac{T}{T_o}\right)^{\left(\frac{g}{RB}-1\right)} \quad (1.5)$$

Την τροπόσφαιρα ακολουθεί η στρατόσφαιρα σε υψόμετρο από 55000 m έως 80000 m. Στη στρατόσφαιρα η θερμοκρασία είναι σταθερή και ίση με $T = 216.5\,K$. Η πίεση εκεί προκύπτει ως εξής :

$$p = p_{55} e^{-\left(\frac{g}{RT}\right)(z-z_{11})} \quad (1.6)$$

Όπου p_{11} η πίεση σε υψόμετρο $z_{55} = 55000\,m$ όπως υπολογίζεται από τη σχέση (2.4).

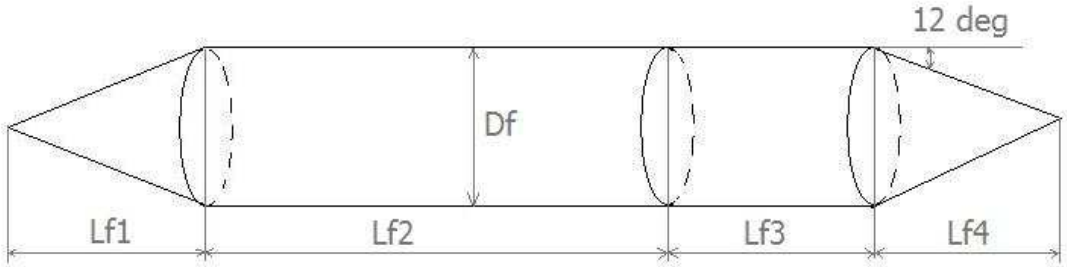
5.4.2 Υπολογισμός Γεωμετρικών Μεγεθών του Αεροσκάφους

Τα τμήματα του αεροσκάφους στα οποία θα επιτελέσουμε υπολογισμούς είναι η άτρακτος , η πτέρυγα και το κάθετο ουραίο πτερύγιο .

5.4.2.1 Άτρακτος

Η διαστασιολόγηση της ατράκτου γίνεται λαμβάνοντας υπόψη τα χαρακτηριστικά άλλων SBJ που παρουσιάζονται στη βιβλιογραφία.

Η άτρακτος χωρίζεται σε τέσσερα τμήματα με βάση το παρακάτω σχήμα.



Σχ. 5.3 – Διαστασιολόγηση ατράκτου

Το κυρίως τμήμα αποφασίζεται πως θα έχει κυλινδρική διατομή, η τιμή της οποίας αποτελεί και μεταβλητή σχεδιασμού. Το εμπρόσθιο και το οπίσθιο τμήμα αποφασίζεται πως θα έχουν κωνική μορφή. Επιπλέον το εμπρόσθιο τμήμα υπολογίζεται έτσι ώστε το αεροσκάφος να βρίσκεται πλήρως μέσα στον κώνο Mach ενώ για το οπίσθιο τμήμα η γωνία κλίσης δεν πρέπει να ξεπερνάει τις 12^o προκειμένου να αποφευχθούν φαινόμενα αποκόλλησης της ροής.

Με βάση τα παραπάνω, τα μήκη του πρόσθιου και του οπίσθιου τμήματος υπολογίζονται από τις παρακάτω σχέσεις:

$$L_{f_1} = \frac{D_f/2}{\tan\{0.3 \cdot \arcsin(\frac{1}{M_{cr}})\}} \qquad (1.7)$$

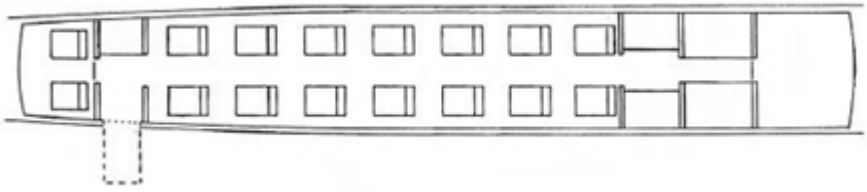
$$L_{f_4} = \frac{D_f/2}{\tan(12^o)} \qquad (1.8)$$

Επίσης το κυλινδρικό τμήμα της ατράκτου χωρίζεται σε δύο τμήματα. Το πρώτο αποτελεί το χώρο για τους επιβάτες (καμπίνα, θάλαμος επιβατών) ενώ το δεύτερο τμήμα χρησιμοποιείται για την αποθήκευση του καυσίμου (ή μέρος αυτού εάν η μοντελοποίηση προβλέπει την αποθήκευση καυσίμων και στην πτέρυγα όπως συχνά συμβαίνει).Εμείς υποθέτουμε ότι το τμήμα αυτό θα περιλαμβάνει όλο το καύσιμο και επομένως το μήκος του τμήματος αυτού δίνεται από τη σχέση:

$$L_{f_3} = \frac{4 \cdot V_{fuel}}{\pi \cdot D_f^2} \qquad (1.9)$$

όπου ο όγκος του καυσίμου δίνεται από τον τύπο: $V_{fuel} = \frac{w_{fuel}}{\rho_{fuel}}$. Το καύσιμο, εξαρτάται από τον τύπο του κινητήρα ενώ συνήθεις τιμές πυκνότητας ρ_{fuel} κυμαίνονται μεταξύ 770 και 820 kg/m^3 . Εμείς επιλέγουμε ένα κάυσιμο τυπικό για αεροσκάφη τέτοιου είδους το Jet A-1 το οποίο έχει πυκνότητα $\rho_{fuel} = 807.5\, kg/m^3$.

Το μήκος θαλάμου των επιβατών καθορίζεται από την απαίτηση μεταφοράς 16 επιβατών αλλά και για προδιαγραφές άνεσης υπηρεσιών πρώτης θέσης ανά επιβάτη. Για τους 16 επιβάτες προβλέπονται δύο σειρές 8 καθισμάτων πρώτης θέσης (με βήμα 1m.) . Επιπλέον με βάση τους κανονισμούς , για αριθμό επιβατών μικρότερο του 20, επιβάλλεται η ύπαρξη μιας εξόδου κινδύνου [16], ενώ στο συνολικό χώρο, συνεκτιμάται η ύπαρξη χώρου αποσκευών και WC. Έτσι προκύπτει $L_{f_2} = 10m$, ενώ μια πιθανή διαμόρφωση του θαλάμου των επιβατών, με βάση τη σχετική βιβλιογραφία, θα μπορούσε να είναι η παρακάτω:



Σχ. 5.4 – Το εσωτερικό της ατράκτου.

Αν και η εικονιζόμενη στο σχήμα άτρακτος έχει ελλειπτική διατομή , η διαφορά της από την προτεινόμενη κυκλική είναι ουσιαστικά μικρή.

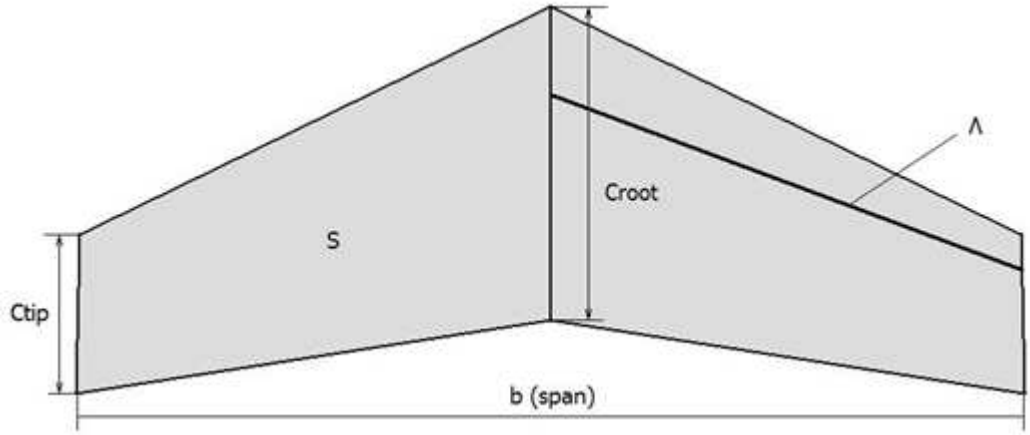
[16] Type III 510 x 915 mm (FAR 25 section 25:807)

Τέλος, ύστερα από τον ορισμό των μηκών της ατράκτου η διάμετρος της ορίζεται από την σχέση:

$$S_f = \pi \cdot D_f \cdot \left[L_{f_2} + L_{f_3} + \frac{1}{2} \cdot \left(\sqrt{\frac{D_f^2}{4} + L_{f_1}} + \sqrt{\frac{D_f^2}{4} + L_{f_4}} \right) \right] \qquad (1.10)$$

5.4.2.2 Πτέρυγα

Οι μεταβλητές σχεδιασμού οι οποίες έχουν ήδη οριστεί και αφορούν την πτέρυγα (επιφάνεια, γωνία οπισθόκλισης στην ακμή προσβολής, λόγος επιμήκους, εκλέπτυνση και λόγος μέγιστου πάχους προς χορδή), καθορίζουν και τα υπόλοιπα χαρακτηριστικά της , μέσω των γεωμετρικών σχέσεων που ακολουθούν.



Σχ. 5.5 – Πτέρυγα αεροσκάφους

Όπου : S=εμβαδόν πτέρυγας , AR=λόγος επί μήκους=b^2/s και Λ=γωνία οπισθόκλισης

Έτσι για τη χορδή στην ρίζα ($C_{root,w}$) ισχύει :

$$C_{rw} = \sqrt{\frac{S_w \cdot [tan(\Lambda_{LE_w}) - tan(\Lambda_{TE_w})]}{1 - \lambda_w^2}} \qquad (1.11)$$

$$\text{Όπου } \lambda_w = \frac{c_{tip_w}}{c_{root_w}}$$

για το εκπέτασμα:

$$b_w = \frac{2 \cdot S_w}{c_{r_w} \cdot (1 + \lambda_w)} \qquad (1.12)$$

και για το λόγο επιμήκους:

$$AR_w = \frac{b_w^2}{S_w} \qquad (1.13)$$

Η γωνία οπισθόκλισης στο 25% του μήκους της χορδής, μέγεθος που χρησιμοποιείται κατά τον υπολογισμό του βάρους της πτέρυγας, ορίζεται από τη σχέση:

$$\tan\Lambda_{0.25_w} = \tan\Lambda_{TE_w} - 0.75 \cdot \frac{4}{AR_w} \cdot \frac{1-\lambda_w}{1+\lambda_w} \quad (1.14)$$

Επιπλέον για τους υπολογισμούς αεροδυναμικών μεγεθών επιβάλλεται ο υπολογισμός της βρεχόμενης επιφάνειας της πτέρυγας (S_{wet_w}), (δηλαδή της συνολικής επιφάνειας της πτέρυγας που «βρέχεται» από τον αέρα) :

$$S_{wet_w} = 2 \cdot \left[1 + 0.2 \cdot \left(\frac{t}{c}\right)_w\right] \cdot S_{exp_w} \quad (1.15)$$

όπου η εκτεθειμένη επιφάνεια (S_{exp_w}) , (εμβαδόν κάτοψης του τμήματος της πτέρυγας που εξέχει από την άτρακτο) είναι :

$$S_{exp_w} = S_w - \frac{D_f}{2} \cdot \left[2 \cdot c_{r_w} - \frac{D_f}{2} \cdot \tan(\Lambda_{LE_w}) + \frac{D_f}{2} \cdot \tan(\Lambda_{TE_w})\right] \quad (1.16)$$

5.4.2.3 Κάθετο Ουραίο Πτερύγιο

Για το κάθετο ουραίο πτερύγιο , όπως και για την πτέρυγα του αεροσκάφους, από τις μεταβλητές σχεδιασμού που έχουν ήδη οριστεί και αφορούν στο ουραίο πτερύγιο, (επιφάνεια, γωνία οπισθόκλισης στην ακμή προσβολής, λόγος επιμήκους, εκλέπτυνση και λόγος μέγιστου πάχους προς χορδή), καθορίζονται και τα υπόλοιπα χαρακτηριστικά της , μέσω των γεωμετρικών σχέσεων που ακολουθούν.

Αρχικά γίνεται η παραδοχή ότι η επιφάνεια της πλάγιας όψης ισούται με το ένα δέκατο της κάτοψης της επιφάνειας της πτέρυγας. Έχουμε δηλαδή :

$$S_v = 0.1 \cdot S_w \quad (1.16)$$

Έτσι για τη χορδή στη ρίζα του κάθετου πτερυγίου ισχύει :

$$C_{rv} = \sqrt{\frac{2 \cdot S_v \cdot [tan(\Lambda_{LE_v}) - tan(\Lambda_{TE_v})]}{1-\lambda_v^2}} \quad (1.18)$$

Για το ύψος ισχύει :

$$b_v = \frac{2 \cdot S_v}{c_{r_v} \cdot (1-\lambda_v^2)} \quad (1.19)$$

Και τέλος, για τη βρεχόμενη επιφάνεια :

$$S_{wet_v} = 2 \cdot \left[1 + 0.2 \cdot \left(\frac{t}{c}\right)_v\right] \cdot S_{exp_v} \qquad (1.20)$$

5.4.3 Υπολογισμός και Διαστασιολόγηση Κινητήρων και Κελυφών

Για τον υπολογισμό και τη διαστασιολόγηση του κινητήρα όπως προαναφέρθηκε στην περιγραφή του αλγόριθμου, αρχικά υποτίθεται μια τιμή για την ώση F_o [N] στο επίπεδο της θάλασσας. Ο υπολογισμός της ώσης στις συνθήκες της ευθείας πτήσης γίνεται από εμπειρικές σχέσεις οι οποίες συνδέουν το λόγο της ώσης F [N] προς την ώση στο επίπεδο της θάλασσας με το υψόμετρο και τη ταχύτητα πτήσης. Ειδικότερα, για κινητήρα διπλού ρεύματος (turbofan) με μικρό λόγο παράκαμψης, ισχύει :

$$\frac{F}{F_o} = 0.6 \cdot \left(1 + \frac{\gamma-1}{2} \cdot M_{cr}^2\right)^{\frac{\gamma}{\gamma-1}} \qquad (1.21)$$

Αντίστοιχα, εμπειρική σχέση χρησιμοποιείται και για την ειδική κατανάλωση καυσίμου :

$$SFC = 2.83 \cdot 10^{-5} \cdot (0.9 + 0.3 \cdot M_{cr}) \cdot \sqrt{\frac{T_{cr}}{T_o}} \quad \left[\frac{Kg}{N \cdot sec}\right] \qquad (1.22)$$

Τα γεωμετρικά χαρακτηριστικά του κινητήρα υπολογίζονται και αυτά από εμπειρικές σχέσεις και εξαρτώνται από την ώση στο επίπεδο της θάλασσας . Έτσι έχουμε :

$$D_{eng} = c_1 \cdot T_o^{k_1} \qquad (1.23)$$

$$L_{eng} = c_2 \cdot T_o^{k_2} \qquad (1.24)$$

όπου c_1, c_2, k_1, k_2 σταθερές.

Τα γεωμετρικά χαρακτηριστικά του κελύφους του κινητήρα και του αγωγού εισόδου υπολογίζονται από τα αντίστοιχα μεγέθη του κινητήρα:

$$\left.\begin{aligned} L_{in} &= 4 \cdot D_{eng} \\ L_{nac} &= L_{in} + L_{eng} \\ D_{nac} &= 1.1 \cdot D_{eng} \end{aligned}\right\} \qquad (1.25)$$

5.4.4 Υπολογισμός Βάρους

Το συνολικό βάρος [17] απογείωσης (TOW) του αεροσκάφους υπολογίζεται αθροίζοντας το βάρος του άδειου από καύσιμα αεροσκάφους (ZFW) και το βάρος του καυσίμου (W_{fuel}), δηλαδή :

$$TOW = ZFW + W_{fuel} \qquad (1.26)$$

Το βάρος του άδειου από καύσιμα αεροσκάφους, αποτελείται από τα βάρη των επιβατών (W_{payl}), του πληρώματος (W_{crew}), το λειτουργικό βάρος (W_{op}) και το βάρος άδειου αεροσκάφους (W_{empty}) . Έτσι έχουμε:

$$ZFW = W_{payl} + W_{crew} + W_{op} + W_{empty} \qquad (1.27)$$

Για τον υπολογισμό των βαρών πληρώματος και επιβατών πρέπει να υποτεθεί ένα μέσο βάρος για κάθε άτομο. Ο αριθμός των μελών του πληρώματος καθορίζεται από κανονισμούς, αλλά και από το είδος της εργασίας για την οποία προορίζεται . Έτσι επειδή πρόκειται για αεροσκάφος μεγάλης εμβέλειας και διάρκειας πτήσης , πρέπει να υπάρχουν δύο πιλότοι . Όσο για τους ιπτάμενους φροντιστές , αρκεί ένας ανά 10-16 επιβάτες, άρα εδώ απαιτείται μόνο ένας . Για το πλήρωμα λοιπόν, έχουμε $N_{crew} = 3$ (υπολογίζουμε έναν κυβερνήτη, έναν συγκυβερνήτη και έναν ιπτάμενο φροντιστή). Για τους επιβάτες, όπως έχουμε ήδη αναφέρει, ισχύει $N_{pas} = 16$. Για τον σχεδιασμό μεγάλων αεροσκαφών, κάθε άτομο υπολογίζεται ότι ζυγίζει 80 kg και μεταφέρει χειραποσκευές βάρους 7 kg και αποσκευές βάρους 20 kg αντίστοιχα. Εδώ, κάθε επιβάτης ή μέλος του πληρώματος υπολογίζεται χοντρικά με συνολικό βάρος 100kg. Έτσι έχουμε:

$$W_{crew} + W_{payl} = 100 \cdot (N_{crew} + N_{pas}) = 1900\ [kg] \qquad (1.28)$$

Το λειτουργικό βάρος που ουσιαστικά αποτελεί τις προμήθειες νερού , φαγητού και εξοπλισμού ασφαλείας, αντιστοιχεί σε περίπου 17kg ανά επιβάτη και 10kg για όλο το πλήρωμα. Έτσι έχουμε $W_{op} = 282\ [kg]$.

Το βάρος άδειου αεροσκάφους αποτελείται από το βάρος του αεροδυναμικού πλαισίου (άτρακτος $\{W_f\}$, πτέρυγα $\{W_w\}$, ουραίο πτερύγιο $\{W_v\}$) , το βάρος του συστήματος προσγείωσης (W_{gear}) , το βάρος του συστήματος πρόωσης (με βάση το βάρος και τον αριθμό των κινητήρων) $W_{prop} = N_{eng} \cdot W_{eng}$ και το βάρος του σταθερού εξοπλισμού του αεροσκάφους (W_{fe}, ηλεκτρικά, υδραυλικά συστήματα, συστήματα πλοήγησης). Έχουμε :

$$W_{empty} = W_w + W_v + W_f + W_{gear} + N_{eng} \cdot W_{eng} + W_{fe} \qquad (1.29)$$

[17] Το βάρος , όπως παρουσιάζεται σε αυτή την ενότητα μετράται σε kg και ουσιαστικά ταυτίζεται με τη μάζα . Προτιμάται , παρόλα αυτά , η χρησιμοποίηση του όρου βάρους προκειμένου η ανάλυση να είναι συμβατή με την σχετική βιβλιογραφία .

Από αυτά, τα βάρη του συστήματος προσγείωσης και του σταθερού εξοπλισμού, με βάση δεδομένα από αντίστοιχα αεροσκάφη και σχετική βιβλιογραφία **[3]** υπολογίζονται κατ' αναλογία προς το συνολικό βάρος απογείωσης . Έτσι έχουμε :

$$W_{gear} = 0.04 \cdot TOW \qquad (1.30)$$

και

$$W_{fe} = 0.08 \cdot TOW \qquad (1.31)$$

Τα βάρη του αεροδυναμικού πλαισίου υπολογίζονται χρησιμοποιώντας σχέσεις που βρίσκονται στη βιβλιογραφία **[14]**.

Έτσι για την πτέρυγα θα έχουμε:

$$W_w = 20.6 \cdot S_w + 5.387 \cdot 10^{-6} \cdot \frac{N_{ult} \cdot b_w^3 \cdot \sqrt{TOW \cdot ZFW} \cdot (1+2 \cdot \lambda)}{\left(\frac{t}{c}\right)_w \cdot (\cos \Lambda_{0.25_w})^2 \cdot S_{exp_w} \cdot (1+2 \cdot \lambda)} \qquad (1.32)$$

με τον «απόλυτο» συντελεστή φόρτισης[18] $N_{ult} = 4.5$

για το ουραίο πτερύγιο θα έχουμε:

$$W_v = 12.8 \cdot S_v + 24 \cdot 10^{-5} \cdot \frac{N_{ult} \cdot b_v^3 \cdot \left(8+0.09 \cdot \frac{TOW}{S_{exp_v}}\right)}{\left(\frac{t}{c}\right)_v \cdot (cos \Lambda_{0.25_v})^2} \qquad (1.33)$$

και για την άτρακτο :

$$W_f = \left(5.1314 + 0.498 \cdot I_f\right) \cdot S_f \qquad (1.34)$$

όπου :

$$I_f = \begin{cases} I_p & \alpha\nu \;\; I_p > I_b \\ \frac{I_p^2 + I_b^2}{2 \cdot I_b} & \alpha\nu \;\; I_p \leq I_b \end{cases} \qquad (1.35)$$

όπου οι δείκτες πίεσης $\left(\{I_p\}\ pressure\ index\right)$ και κάμψης $(\{I_b\}\, bending\ \ index)$ υπολογίζονται από τις σχέσεις :

[18] Απόλυτος συντελεστής φόρτισης είναι ο οριακός συντελεστής φόρτισης (αναλογία άνωση προς βάρος) πολλαπλασιασμένος επί 1.5 για μεγαλύτερη ασφάλεια των δοκιμών στοιχείων του αεροσκάφους . Ειδικότερα , τα δομικά στοιχεία αρχίζουν να υφίστανται φθορές στον οριακό συντελεστή φόρτισης ενώ αποτυγχάνουν πλήρως στον απόλυτο συντελεστή φόρτισης .

$$I_p = 10^{-4} \cdot (p_{cr} + p_{cab}) \cdot D_f \qquad (1.36)$$

$$I_b = 1.3 \cdot 10^{-4} \cdot \frac{W_{lb} \cdot N_{ult} \cdot L_f}{D_f^2} \cdot D_f \qquad (1.37)$$

$$W_{lb} = ZFW - W_w - N_{we} \cdot W_{eng} \qquad (1.38)$$

όπου p_{cab} η πίεση στην καμπίνα των επιβατών υπολογισμένη[19] ίση με αυτή σε υψόμετρο 2000m και N_{we} είναι ο αριθμός των κινητήρων που είναι «δεμένοι» στην πτέρυγα. Τέλος, το βάρος κάθε κινητήρα εξαρτάται από την ώση του στο επίπεδο της θάλασσας (Fo) , ως :

$$W_{eng} = 3.5 \cdot 10^{-2} \cdot F_o^{0.9255} \qquad (1.39)$$

5.4.5 Υπολογισμός Αεροδυναμικών μεγεθών

5.4.5.1 Συντελεστής Άνωσης

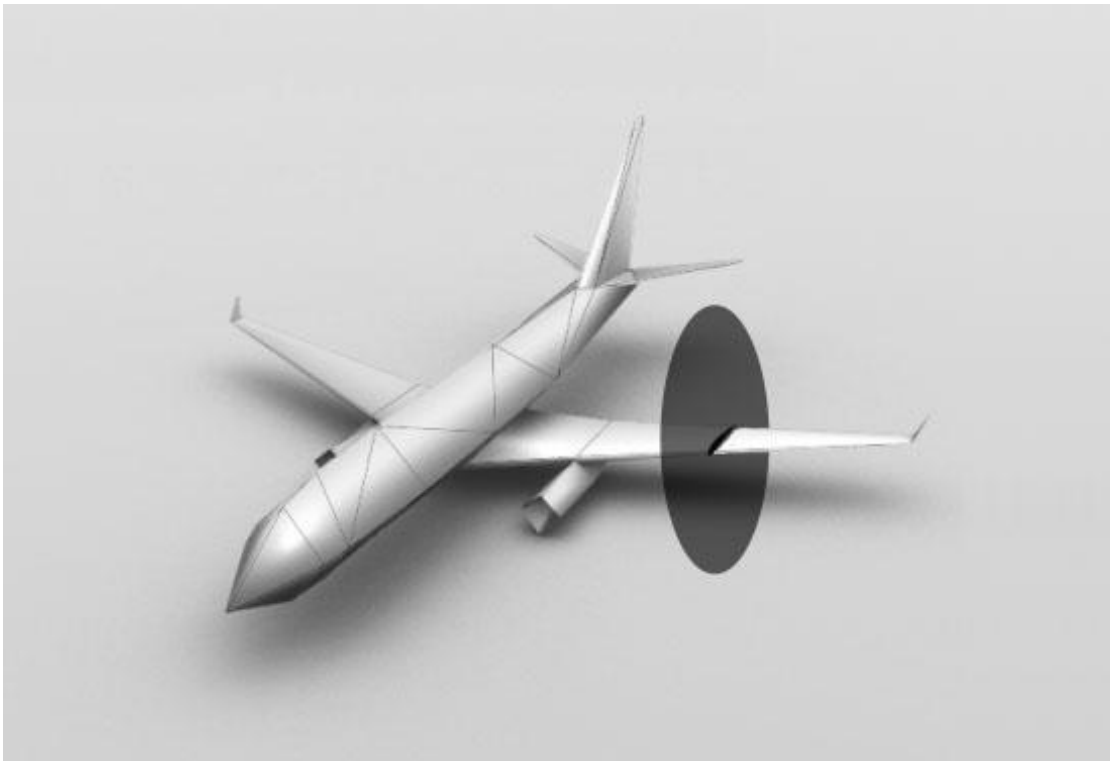
5.4.5.1.1 Υπολογισμός με αριθμητική επίλυση ροής

Ο συντελεστής άνωσης του αεροσκάφους τόσο για γωνία πτήσης όσο και για τη μέγιστη γωνία υπολογίζεται με επίλυση της ροής γύρω από την αεροτομή και επομένως μας δίνεται έτοιμος από το πρόγραμμα επίλυσης ροής που περιγράφηκε στο κεφάλαιο 3 για χρήση στον αλγόριθμο μας.

Συγκεκριμένα, γνωρίζοντας τα γεωμετρικά χαρακτηριστικά της πτέρυγας του αεροσκάφους, επιλέγεται η θέση στο μέσο του εκπετάσματός της, στην οποία θεωρούμε αεροτομή γνωστής γεωμετρίας, όπως στο δίπλα σχήμα, κατάλληλη για την μελέτη μας.

Σε εκείνη τη θέση, με χρήση του βοηθητικού προγράμματος **mach.exe** που δημιουργήθηκε, υπολογίζονται οι συνθήκες ροής του αέρα που «βλέπει» η αεροτομή και με χρήση του επιλύτη ροής υπολογίζεται η άνωση στη θέση αυτή. Τέλος, με ολοκλήρωση του αποτελέσματος αυτού για όλη την πτέρυγα, υπολογίζεται η συνολική άνωση του αεροσκάφους που είναι και το ζητούμενο μας. Βέβαια με αυτή τη μέθοδο επίλυσης αγνοούνται τριδιάστατα φαινόμενα και επιδράσεις της ροής, όμως ο σκοπός μας δεν είναι η εκτέλεση ενός ορθού μοντέλου σχεδιασμού, αλλά η ενσωμάτωση σε αυτό υπολογιστικής επίλυσης σε κάρτα γραφικών.

[19] FAR 25 (section 25:841)



Σχ. 5.6 - Αεροτομή πτέρυγας σε ενδεικτική θέση

Σημειώνεται ότι ενώ η γωνία πτήσης $alfa_{cr}$ αποτελεί μεταβλητή σχεδιασμού και μεταβάλλεται συνεχώς, η μέγιστη γωνία πτήσης $alfa_{max}$ ισούται με τη γωνία απώλειας στήριξης της αεροτομής, υπολογίζεται με βάση το κεφάλαιο 4 και παραμένει αμετάβλητη για το πρόβλημα μας.

Επομένως, με χρήση της αριθμητικής επίλυσης της ροής, πριν την εκτέλεση του μοντέλου, υπολογίζουμε την πολική καμπύλη της αεροτομής η οποία είναι δεδομένη πλέον για το πρόβλημα μας. Στη συνέχεια, σε κάθε εκτέλεση του μοντέλου η ζητούμενη άνωση προκύπτει με αριθμητική παρεμβολή στην πολική καμπύλη για την δεδομένη γωνία πτήσης $alfa_{cr}$, ενώ η γωνία $alfa_{max}$ που έχει ήδη υπολογιστεί αντιστοιχεί στην μέγιστη (γνωστή επίσης από την πολική) άνωση του αεροσκάφους.

5.4.5.1.2 Υπολογισμός με εμπειρικό μοντέλο

Εναλλακτικά μπορεί να χρησιμοποιηθεί εμπειρικό μοντέλο υπολογισμού της άνωσης του αεροσκάφους.

Συγκεκριμένα, ο συντελεστής άνωσης δίνεται από την αναλογία Polhamus, σύμφωνα με την οποία η άνωση υπολογίζεται ως άθροισμα της δυναμικής άνωσης (potential lift) και της άνωσης που σχετίζεται με την ύπαρξη αποκολλημένων στροβίλων στην ακμή προσβολής (vortex lift).

Σύμφωνα με αυτή είναι:

$$c_L = \frac{L}{\frac{1}{2}\cdot \rho_{cr}\cdot V_{cr}^2 \cdot S_w} = \frac{\pi \cdot AR_w}{2}\cdot sina \cdot cosa \cdot \left(cosa + \frac{sina \cdot cosa}{cos\Lambda_{LE_w}} - \frac{sina}{2\cdot cos\Lambda_{LE_w}}\right)$$

όπου a η γωνία προσβολής.

5.4.5.2 Συντελεστής οπισθέλκουσας

Ο συντελεστής οπισθέλκουσας του αεροσκάφους σχηματίζεται από το συντελεστή οπισθέλκουσας μηδενικής άνωσης στον οποίο υπερτίθεται η επαγόμενη οπισθέλκουσα λόγο άνωσης ως:

$$c_D = \frac{D}{\rho_{cr}\cdot V_{cr}^2 \cdot S_w} = c_{D_o} + K\cdot c_L^2 \qquad (1.40)$$

όπου D η οπισθέλκουσα και K συντελεστής που εξαρτάται από τον αριθμό Mach πτήσης ως εξής :

$$K = \begin{cases} \frac{1}{\pi\cdot AR_w\cdot e} & για\ M_{cr} < 1 \\ \frac{AR_w\cdot(M_{cr}^2-1)}{4\cdot AR_w\cdot\sqrt{M_{cr}^2-1}-2}\cdot \cos\Lambda_{LE_w} & για\ M_{cr} > 1 \end{cases} \qquad (1.41)$$

όπου e ο συντελεστής Oswald **[3]**:

$$e = 4.61\cdot(1-0.045\cdot AR_w^{0.68})\cdot \cos\Lambda_{LE_w}^{0.15} - 3.1 \qquad για\ \Lambda_{LE_w} > 30^o \qquad (1.42)$$

Ο συντελεστής οπισθέλκουσας μηδενικής άνωσης, ανάλογα με τις συνθήκες πτήσης, εξαρτάται από την οπισθέλκουσα λόγω κατεβασμένων τροχών ($c_{D_{gear}}$), κυμάτων ($c_{D_{wave}}$) και διατμητικών τάσεων ($c_{D_{visc}}$) ως εξής:

$$C_{D_o} = \begin{cases} c_{D_{visc}} + c_{D_{gear}} & ,για\ κατεβασμένους\ τροχούς\ και\ M_{cr} < 1 \\ c_{D_{visc}} + c_{wave} & ,για\ M_{cr} \geq 1 \end{cases} \qquad (1.43)$$

όπου ο συντελεστής οπισθέλκουσας λόγω των κατεβασμένων τροχών έχει σταθερή τιμή $c_{D_{gear}} = 0.02$.

Η οπισθέλκουσα λόγω κυμάτων είναι η δύναμη πίεσης από την ύπαρξη κρουστικών κυμάτων, καθορίζεται από την κατανομή όγκου του αεροσκάφους και για ένα αεροσκάφος υπολογίζεται από τη σχέση :

$$c_{D_{wave}} = \frac{9\cdot\pi}{2\cdot S_w}\cdot\left(\frac{A_{max}}{L_f}\right)^2 \cdot E_{wd}\cdot\left[1 - 0.386\cdot(M_{cr}-1.2)^{0.57}\cdot\left(1-\frac{\pi\cdot\Lambda_{LE_w}^{0.77}}{100}\right)\right] \qquad (1.44)$$

Όπου A_{max} το εμβαδόν της μέγιστης τομής του αεροσκάφους και E_{wd} ένας εμπειρικός συντελεστής που λαμβάνει τιμές στο διάστημα 1.8 – 2.2 (εδώ τίθεται $E_{wd} = 2$) . Ο συντελεστής οπισθέλκουσας λόγω διατμητικών τάσεων δίνεται από το άθροισμα :

$$c_{D_{visc}} = \sum_i \left(\frac{S_{wet_i}}{S_i} \cdot c_{f_i} \right) \quad (1.45)$$

Όπου ο συντελεστής τριβής c_{f_i} για κάθε στοιχείο (άτρακτο, πτέρυγα, ουραίο πτερύγιο, κέλυφος κινητήρων) είναι υπολογισμένος για τυρβώδη ροή. Ειδικότερα, ο υπολογισμός του συντελεστή τριβής γίνεται από το συντελεστή τριβής για ασυμπίεστες ροές λαμβάνοντας επιπλέον υπόψη την επίδραση του αριθμού Mach[20] σύμφωνα με τη σχέση:

$$\frac{c_f}{c_f^{inc}} = \frac{r_R^{0.2}}{r_T} \quad (1.46)$$

όπου ο συντελεστής τριβής για ασυμπίεστες ροές ισούται με

$$c_f'^{inc} = \frac{0.074}{Re'^{0.2}} \quad (1.47)$$

Ο αριθμός Reynolds $Re_i = Re_i(L_{char})$ της ροής υπολογίζεται από τη σχέση $Re_i = \frac{\rho \cdot V \cdot L_{char}}{\mu}$ όπου το χαρακτηριστικό μήκος κατά περίπτωση είναι το L_f για την άτρακτο , το L_{nac} για το κέλυφος του κινητήτρα και η μέση αεροδυναμική χορδή για την πτέρυγα και το ουραίο πτερύγιο. Η γενική σχέση υπολογισμού της μέσης αεροδυναμικής χορδής είναι:

$$\overline{c_j} = \frac{2 \cdot c_{r_j} \cdot \left(1 + \lambda_j + {\lambda_j}^2\right)}{3 \cdot \left(1 + \lambda_j\right)} \quad (1.48)$$

όπου $j \equiv w$ για την πτέρυγα και $j \equiv v$ για το κάθετο ουραίο πτερύγιο.

Για την ευθεία υπερηχητική πτήση, ο συντελεστής οπισθέλκουσας μηδενικής άνωσης υπολογίζεται από την δεύτερη σχέση της εξίσωσης (1.43) και ο συνολικός συντελεστής οπισθέλκουσας από την εξίσωση (1.40). Στη συνέχεια, η οπισθέλκουσα D εξισώνεται με την συνολική ώση F_{tot} του αεροσκάφους:

[20] Η εξάρτηση του συντελεστή τριβής από τον αριθμό Mach είναι μικρή στους χαμηλούς αριθμούς Mach, αλλά γίνεται σημαντική σε υψηλούς αριθμούς Mach . Ο τρόπος υπολογισμού του λόγου c_f/c_f^{inc} παρουσιάστηκε από τους Sommer και Short , ονομάζεται μέθοδος T' και στηρίζεται στο ότι η «αεροδυναμική θέρμανση» (aerodynamic heating) μεταβάλλει τις ιδιότητες του ρευστού. Θεωρώντας πλήρως τυρβώδη ροή, η θερμοκρασία του τοιχώματος υπολογίζεται από τη θερμοκρασία στο υψόμετρο πτήσης ως $\frac{T_w}{T_{cr}} = (1 + 0.178 \cdot M_{cr}^2)$. Ο λόγος της φαινόμενης ασυμπίεστης θερμοκρασίας ορίζεται ως: $r_T = \frac{T'}{T_{cr}} = 1 + 0.035 \cdot M_{cr}^2 + 0.45 \cdot \left(\frac{T_w}{T_{cr}} - 1\right)$ και αντίστοιχα ο λόγος του φαινόμενου αριθμού Reynolds $r_R = \frac{Re'}{Re_{cr}} = \frac{\rho'}{\rho_{cr}} \cdot \frac{\mu_{cr}}{\mu'} = \frac{T_{cr}}{T'} \cdot \frac{\mu_{cr}}{\mu'}$ με $\frac{\rho'}{\rho_{cr}} = \frac{T_{cr}}{T'}$ θεωρώντας σταθερή πίεση στο οριακό στρώμα, ενώ η συνεκτικότητα υπολογίζεται από τον τύπο του Sutherland. Έτσι ο συντελεστής τριβής για ασυμπίεστες ροές δίνεται από τη σχέση $c_f = \frac{T_{cr}}{T'} \cdot c_f'^{inc}$ με $c_f'^{inc}$ τον ασυμπίεστο συντελεστή τριβής υπολογισμένος σε αριθμό Reynolds Re' ως $c_f'^{inc} = \frac{0.074}{Re'^{0.2}}$. Άρα ο λόγος του συμπιεστού προς τον ασυμπίεστο συντελεστή τριβής είναι $\frac{c_f}{c_f^{inc}} = \frac{r_R^{0.2}}{r_T}$.

$$\frac{1}{2} \cdot \rho_{cr} \cdot V_{cr}^2 \cdot S_w \cdot c_D = N_{eng} \cdot F \qquad (1.49)$$

όπου $N_{eng} = 2ή3ή4$ όπως προαναφέρθηκε. Από την παραπάνω σχέση υπολογίζεται εκ νέου η ώση για κάθε κινητήρα, άρα και η ώση στο επίπεδο της θάλασσας της οποίας η τιμή διορθώνεται και οι υπολογισμοί βάρους και αεροδυναμικών μεγεθών επαναλαμβάνονται σύμφωνα με τον αλγόριθμο που παρουσιάστηκε παραπάνω .

5.4.6 Υπολογισμός Επιδόσεων

5.4.6.1 Εμβέλεια

Η εμβέλεια του αεροσκάφους (Range) υπολογίζεται από την εξίσωση Breguet :

$$R = \frac{V}{g \cdot SFC \cdot c_D} \cdot \frac{c_L}{c_D} \cdot \ln\left(\frac{W_{bc}}{W_{ec}}\right) \qquad (1.50)$$

όπου το βάρος κατά την έναρξη της πτήσης υπολογίζεται από τη σχέση :

$$W_{bc} = TOW - W_{fuel}^{0-2} \approx 0.95 \cdot TOW \qquad (1.51)$$

αφού γίνεται η προσέγγιση ότι $W_{fuel}^{0-2} \approx 0.05 \cdot TOW$ (είναι το βάρος του καυσίμου που απαιτείται για το τμήμα 0-2 της αποστολής του αεροσκάφους, δηλαδή την εκκίνηση, την τροχοδρόμηση, την απογείωση και την άνοδο στο υψόμετρο πτήσης)

ενώ το βάρος στο τέλος της πτήσης εκτιμάται ίσο με :

$$W_{\varepsilon c} \approx TOW - 0.95 \cdot W_{fuel} \qquad (1.52)$$

5.4.6.2 Ταχύτητα Προσέγγισης

Η ταχύτητα προσέγγισης (V_{appr}) , η οποία αποτελεί σημαντική παράμετρο που καθορίζει το μήκος προσγείωσης , εξαρτάται από την ταχύτητα αποκόλλησης (V_s) . Ειδικότερα , με βάση τους κανονισμούς[21] αποτελεί πολλαπλάσιο της με βάση τη σχέση :

$$V_{appr} = 1.3 \cdot V_s \qquad (1.53)$$

Ως ταχύτητα αποκόλλησης (για δεδομένη γωνία προσβολής) ορίζεται η ελάχιστη ταχύτητα για την οποία η πτέρυγα δημιουργεί επαρκή άνωση για την πτήση και εξαρτάται από τη φόρτιση της πτέρυγας $\left(\frac{w}{s_w}\right)$ και από το μέγιστο συντελεστή άνωσης $c_{L_{max}}$. Για τον υπολογισμό της ταχύτητας αποκόλλησης θεωρείται ότι η άνωση ισούται με το βάρος και ο συντελεστής άνωσης έχει τη μέγιστη τιμή του (με βάση το κεφάλαιο 3.1 για γωνία). Στην προκειμένη περίπτωση, ως βάρος λαμβάνεται το μέγιστο βάρος προσγείωσης (MLW). Αυτό υπολογίζεται ως το ποσοστό του συνολικού βάρους απογείωσης και ειδικότερα αποτελεί το 80-100%

[21] FAR 23 (section 23:73)

αυτού ώστε να είναι εφικτή ενδεχόμενη αναγκαστική προσγείωση αμέσως μετά την απογείωση [Raymer] . Έτσι έχουμε :

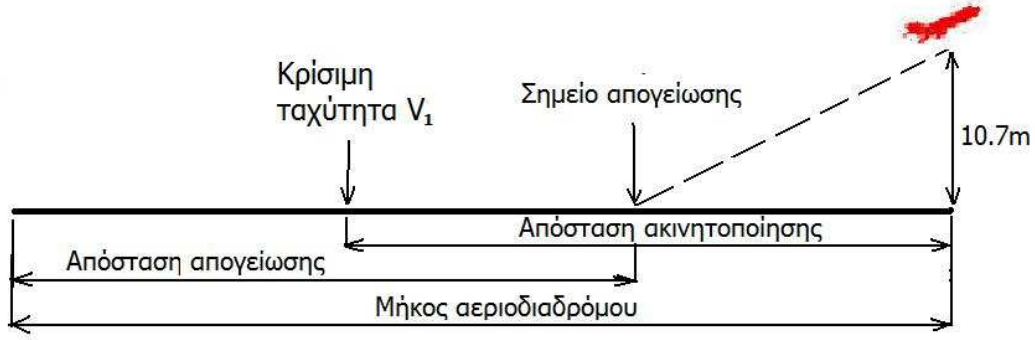
$$V_s = \sqrt{\frac{g \cdot MLW}{\frac{1}{2} \cdot \rho_o \cdot S_w \cdot c_{L_{max}}}} \qquad (1.54)$$

όπου για το δικό μας πρόβλημα θέτουμε $MLW = 0.8 \cdot TOW$.

5.4.6.3 Μήκος Απογείωσης

Στη βιβλιογραφία εμφανίζονται διάφοροι τρόποι υπολογισμού του μήκους απογείωσης [Bergel], [Raymer], [Jenkin] κ.α.. Στη συγκεκριμένη ανάλυση χρησιμοποιείται η θεωρία του «ισορροπημένου μήκους απογείωσης» (BFL {Balanced Field Length}), όπως παρουσιάζεται από τον Raymer **[3]**. Έτσι, αρχικά ορίζεται το μήκος τροχοδρόμησης (ground roll), δηλαδή η απόσταση που διανύει το αεροσκάφος μέχρι τη στιγμή που αποκολλάται από το έδαφος (σε ταχύτητα ίση με το 110% - 120% της ταχύτητας αποκόλλησης) και το μήκος υπερπήδησης εμποδίου (obstacle clearance distance), δηλαδή η απόσταση που απαιτείται από την αποκόλληση εωσότου το αεροσκάφος φτάσει σε ύψος 35ft (=10.7m) .

«Ισορροπημένο μήκος απογείωσης» είναι το μήκος που απαιτείται για την απογείωση στην περίπτωση αστοχίας ενός κινητήρα (για αεροσκάφη με δύο ή περισσότερους κινητήρες) τη χειρότερη, για την απογείωση, χρονική στιγμή και ανταποκρίνεται στους κανονισμούς του FAR 25. Ειδικότερα, κατά τη φάση της τροχοδρόμησης υπάρχει μια κρίσιμη ταχύτητα «απόφασης» (decision speed V_1) στην οποία, αν αστοχήσει ένας κινητήρας, το αεροσκάφος μπορεί είτε να φρενάρει και να σταματήσει είτε να συνεχίσει την απογείωση. Αν υπάρξει αστοχία κινητήρα πριν την ταχύτητα V_1 τότε το αεροσκάφος σταματά, ενώ διαφορετικά ο πιλότος υποχρεούται να το απογειώσει. Για τον υπολογισμό του BFL η απογείωση χωρίζεται σε δυο φάσεις, τη φάση της επιτάχυνσης από μηδενική ταχύτητα μέχρι την ταχύτητα αστοχίας ενός κινητήρα και τη φάση της κίνησης ως ύψος 10.7m.



Σχ. 5.7 – Φάση απογείωσης και μήκος αεροδιαδρόμου.

Η αναλυτική σχέση υπολογισμού του BFL είναι :

$$BFL = \frac{0.863}{1+2.3\cdot\Delta\gamma}\cdot\left(\frac{TOW}{S_w\cdot\rho\cdot c_{L_{climb}}}+h_{to}\right)\cdot\left(\frac{1}{\frac{\bar{F}}{g\cdot TOW}-U}+2.7\right)+\frac{\Delta S_{to}}{\sqrt{\sigma}} \quad (1.55)$$

όπου $h_{to}=10.7m$, $\Delta S_{to}=200m$, $U=0.01\cdot c_{L_{max}}+0.02$ για κατεβασμένα flap, $\bar{F}=0.75\cdot N_{eng}\cdot F_o$ η μέση ώση κατά την απογείωση και $\Delta\gamma=\gamma_{climb}-\gamma_{min}$, ενώ:

$$\gamma_{climp}=\sin^{-1}\left(\frac{(N_{eng}-1)\cdot F_o-D}{g\cdot TOW}\right) \text{ και } \gamma_{min}=\begin{cases}0.024\ \alpha\nu\ N_{eng}=2\\0.027\ \alpha\nu\ N_{eng}=3\\0.030\ \alpha\nu\ N_{eng}=4\end{cases} \quad (1.58)$$

είναι οι γωνίες ανόδου και μια ελάχιστη γωνία, αντίστοιχα. Η οπισθέλκουσα (D) και ο συντελεστής άνωσης ($c_{L_{climb}}$) υπολογίζονται με βάση τα μεγέθη της απογείωσης. Ειδικότερα, η ταχύτητα κατά την απογείωση είναι $V_{to}=1.2\cdot V_{to_s}$, με την ταχύτητα αποκόλλησης κατά τη απογείωση ίση με :

$$V_{to_s}=\sqrt{\frac{g\cdot TOW}{\frac{1}{2}\cdot\rho_o\cdot S_w\cdot c_{L_{max}}}} \quad (1.59)$$

Επιπλέον, για τον υπολογισμό της οπισθέλκουσας χρησιμοποιείται η σχέση $D=\frac{1}{2}\cdot\rho_o\cdot V_{to}^2\cdot S_w\cdot c_D$ όπου ο συντελεστής οπισθέλκουσας υπολογίζεται για υποηχητική ταχύτητα και περιλαμβάνει την οπισθέλκουσα λόγω κατεβασμένων τροχών.

5.5 Επιτάχυνση μοντέλου

Σκοπός του επαναπρογραμματισμού του μοντέλου, για εκτέλεση στην κάρτα γραφικών, ήταν η μείωση του χρόνου εκτέλεσης τους. Όμως το μοντέλο από μόνο του, απαιτεί ελάχιστη υπολογιστική ισχύ και με μέσο εκτέλεσης τον κεντρικό επεξεργαστή, απαιτεί ελάχιστο χρόνο για να διεκπεραιωθεί. Επομένως η εκτέλεση του στην κάρτα γραφικών επέφερε μηδαμινή επιτάχυνση. Παρόλα αυτά για εκπαιδευτικούς λόγους ήταν μια πολύ καλή εφαρμογή για γνωριμία με την νέα τεχνολογία.

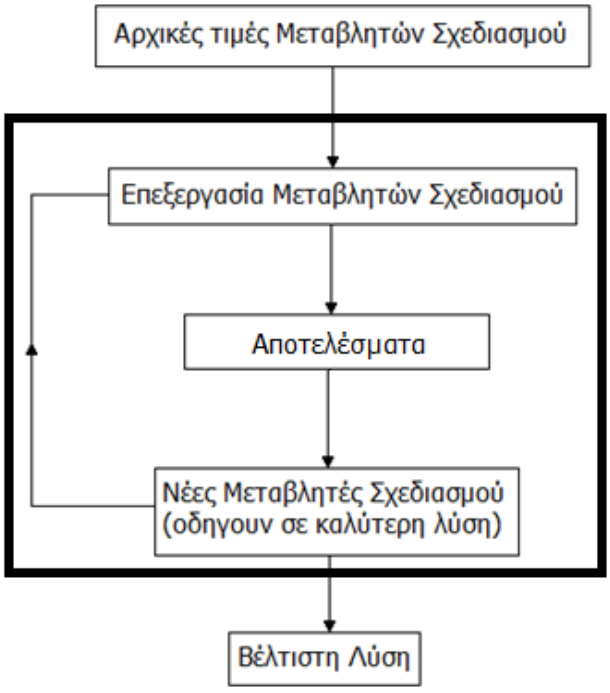
Βέβαια, με την προσθήκη επίλυσης ροής, για την υπολογισμό της πολικής καμπύλης κατά την αρχή του μοντέλου, διαδικασίας που εισάγει σημαντικό υπολογιστικό κόστος και εκτέλεση στην κάρτα γραφικών, επιτυγχάνεται η αναμενόμενη επιτάχυνση με βάση το κεφάλαιο 4.

Κεφάλαιο 6ο
Βελτιστοποίηση

Εισαγωγή :

Στην προηγούμενη ενότητα παρουσιάστηκε η εφαρμογή μας, δηλαδή το μοντέλο προκαταρκτικού σχεδιασμού του αεροσκάφους. Σε αυτό το κεφάλαιο, στόχος είναι η βελτιστοποίηση του προβλήματος μας.

Βελτιστοποίηση είναι η διαδικασία κατά την οποία ανιχνεύονται οι υποψήφιες λύσεις ενός προβλήματος και εντοπίζεται η βέλτιστη από αυτές, ενώ μέθοδος βελτιστοποίησης είναι το εργαλείο που ανιχνεύει το χώρο των υποψήφιων λύσεων και οδηγεί στην βέλτιστη από όλες τις υποψήφιες. Στο παρακάτω σχήμα παρουσιάζεται ο γενικός αλγόριθμος βελτιστοποίησης.



Σχ. 6.1 – Γενικός Αλγόριθμος Βελτιστοποίησης

Το τι μέθοδος θα χρησιμοποιηθεί εξαρτάται από τι είδους προβλήματα καλούμαστε να αντιμετωπίσουμε και ποιοι είναι οι πιθανοί-συνηθισμένοι στόχοι. Στην παρούσα διπλωματική εργασία, επιζητούμε την επιλογή των κατάλληλων παραμέτρων και μεταβλητών σχεδιασμού που οδηγούν στην επίτευξη των βέλτιστων επιδόσεων του αεροσκάφους, με σκοπό την μελέτη της επιτάχυνσης της εκτέλεσης του προβλήματος μας. Γι'αυτό επιλέγεται η χρήση Εξελικτικών Αλγόριθμων (Evolutionary Algorithms) που είναι μια μέθοδο η οποία απαιτεί μεγάλο αριθμό

αξιολογήσεων, γεγονός που την καθιστά ιδανική για το πρόβλημα. Αναφορές σε προηγούμενες μελέτες του ΕΘΣ σε προβλήματα αεροδυναμικής βελτιστοποίησης με χρήση ΕΑ βρίσκονται στα **[27,28,29].**

Σε αυτήν την ενότητα δεν θα μιλήσουμε για το θεωρητικό υπόβαθρο των εξελικτικών αλγορίθμων[22] καθώς αυτό έχει παρουσιαστεί ήδη αρκετές φορές στα πλαίσια παλαιοτέρων διπλωματικών εργασιών του τομέα, αλλά θα θέσουμε τις παραμέτρους, τους στόχους και τους περιορισμούς για την λειτουργία του λογισμικού αξιολόγησης[23].

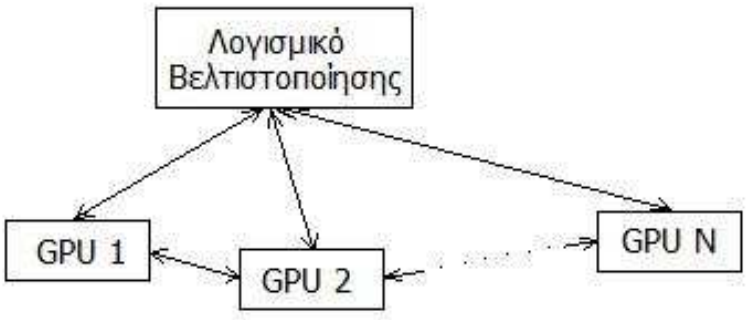
6.1 Συσκευή Εκτέλεσης

Σημαντικό σε αυτή τη διαδικασία, είναι να ξεκαθαριστεί το γεγονός ότι το λογισμικό βελτιστοποίησης δεν είναι προγραμματισμένο για εκτέλεση στην κάρτα γραφικών και ούτε είναι απαραίτητο κάτι τέτοιο.

Η κάρτα γραφικών είναι το μέσο στο οποίο θα εκτελείται το πρόγραμμα που θα αξιολογεί τις υποψήφιες λύσεις (στην περίπτωση μας το πρόγραμμα του προκαταρκτικού σχεδιασμού του αεροσκάφους).Θα δέχεται τις υποψήφιες λύσεις από το λογισμικό βελτιστοποίησης και θα επιστρέφει την αξιολόγηση τους.

Επίσης, πρέπει να τονιστεί το γεγονός ότι χρησιμοποιούμε κάρτες γραφικών οι οποίες επικοινωνούν μόνο με το λογισμικό βελτιστοποίησης και όχι μεταξύ τους καθώς και ο λόγος για τον οποίο γίνεται αυτό.

Αν είχαμε πλήθος N καρτών γραφικών όπως φαίνεται στο παρακάτω σχήμα και εφαρμόζαμε παραλληλία μεταξύ τους, πέραν της παράλληλης εκτέλεσης που επιτελεί η κάθε μια στο εσωτερικό της, θα είχαμε ως αποτέλεσμα την μεταφορά δεδομένων από την μία στην άλλη, ενέργεια χρονοβόρα που θα αύξανε το υπολογιστικό κόστος και θα μείωνε το βαθμό απόδοσης της διαδικασίας.

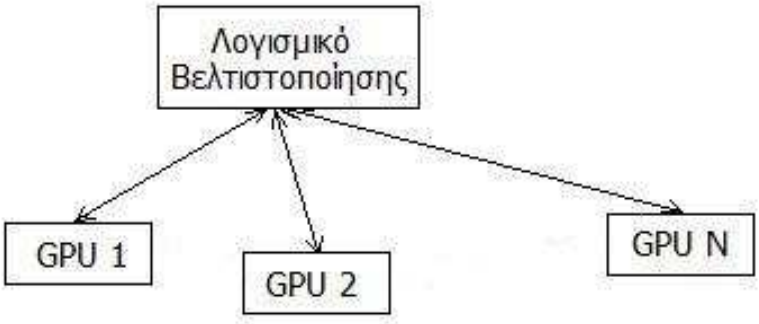


Σχ. 6.2 – Συνεργασία υπολογιστικών μονάδων

[22] Στο παράρτημα Β υπάρχει σύντομη αναφορά για τους εξελικτικούς αλγόριθμους

[23] Χρησιμοποιήθηκε το έτοιμο πακέτο λογισμικού EASY του Εργαστηρίου Θερμικών Στροβιλομηχανών του ΕΜΠ **[15].**

Αντιθέτως, αν οι κάρτες δεν επικοινωνούν μεταξύ τους, παρά μόνο δέχονται και επιστρέφουν δεδομένα από το λογισμικό αξιολόγησης όπως στο παρακάτω σχήμα, τότε επιτυγχάνουμε μέγιστη απόδοση της εφαρμογής μας.



Σχ. 6.3 - Συνεργασία υπολογιστικών μονάδων.

Έτσι σε ένα πρόβλημα όπως το δικό μας, που απαιτεί N επιλύσεις ροής για τον υπολογισμό της πολικής καμπύλης πριν την εκτέλεση του μοντέλου του αεροσκάφους, η βέλτιστη επιλογή θα ήταν η εκτέλεση κάθε επίλυσης ροής σε διαφορετική κάρτα γραφικών.

Επίσης, αν το λογισμικό βελτιστοποίησης στηρίζεται σε εξελικτικούς αλγόριθμους, όπως στην περίπτωση μας, και υπάρχουν N διαθέσιμα χρωμοσώματα για αξιολόγηση σε κάθε γενιά, είναι προτιμότερη η αποστολή για αξιολόγηση κάθε ενός σε διαφορετική κάρτα γραφικών, από τη συνεργασία των καρτών για την αξιολόγηση του καθενός ξεχωριστά.

Στη παρούσα διπλωματική χρησιμοποιείται μία κάρτα βέβαια, για εκτέλεση των εφαρμογών μας, όμως υπάρχει η δυνατότητα χρησιμοποίησης περισσοτέρων.

6.2 Στόχοι , Περιορισμοί και Εύρος μεταβολής Μεταβλητών Σχεδιασμού

Το πρόβλημα μας είναι πολυκριτιριακό και επιζητούμε τους εξής δυο στόχους:

⇨Αύξηση της εμβέλειας του αεροσκάφους (Καθώς αυτό προορίζεται για υπερατλαντικά ταξίδια).

⇨Μείωση του βάρους καυσίμων του (Καθώς επιζητούμε μικρό μέγεθος αεροσκάφους).

Επίσης επιβάλλουμε στο πρόβλημα μας περιορισμούς στις εξής μεταβλητές:

⇨Ισορροπημένο μήκος απογείωσης (BFL {Balanced Field Length}). Για να είναι λειτουργικό το αεροσκάφος και επειδή το μήκος απογείωσης είναι μια

παράμετρος η οποία καθορίζει τον αριθμό των αεροδρομίων που μπορεί να χρησιμοποιήσει ένα αεροσκάφος, επιβάλουμε ένα άνω όριο στο μήκος απογείωσης. Με γνώμονα ότι τα μεγαλύτερη αεροδρόμια του κόσμου έχουν μήκος απογείωσης που φτάνει έως και τα 4500m **[1]** (JFK: 4442, Charles de Gaulle: 4215, San Francisco International Airport: 3618, Ελ. Βενιζέλος: 4000) επιλέγουμε το μήκος ισορροπημένης απογείωσης να είναι μικρότερο από τα 4000m.

⇨Ταχύτητα προσέγγισης (Vappr). Συνήθης τιμή ταχύτητας προσέγγισης υπερηχητικών αεροσκαφών είναι τα 80 m/s. Λαμβάνοντας υπόψη ότι χρησιμοποιούμε σταθερή αεροτομή κατά την διάρκεια της βελτιστοποίηση επιλέγουμε ανεκτικότερη τιμή ορίου της ταχύτητας. Έτσι επιλέγουμε η ταχύτητα προσέγγισης να είναι μικρότερη των 100M/s.

Έτσι έχουμε:

Μεταβλητή	Άνω όριο	Μονάδες
BFL	4000	m
Vappr	100	m/sec

Επίσης, για κάθε μία από τις δεκαπέντε μεταβλητές σχεδιασμού, ορίζουμε ένα συγκεκριμένο εύρος μεταβολής τους που να εξυπηρετεί τους σχεδιαστικούς στόχους μας.

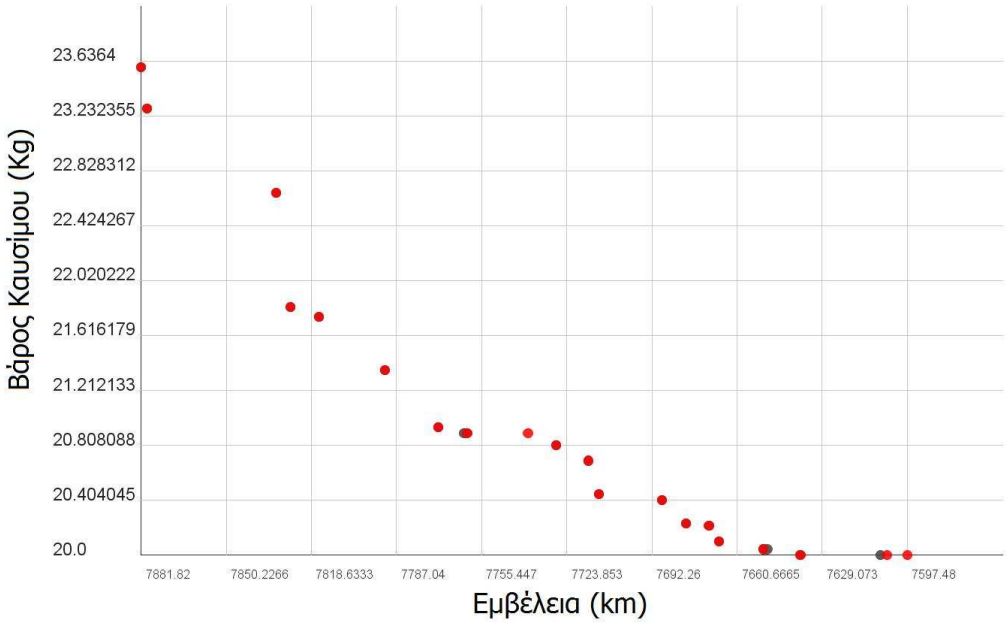
Τα όρια μεταβολής των ανωτέρω μεταβλητών παραθέτονται παρακάτω :

Μεταβλητή	Κάτω όριο	Άνω όριο	Μονάδες
hcr	13000	18000	m
Mcr	1.5	2.0	
alfa	1.0	6.0	o
Dfus	2.0	2.3	m
Sw	120.0	220.0	m2
Λlev	40.0	70.0	o
ARv	1.0	3.5	
λv	0.04	0.5	
t/cv	0.04	0.06	
Λlev	45.0	60.0	o
ARv	0.8	2.5	
λv	0.05	0.5	
t/cv	0.06	0.08	
Nult	0.75	0.90	

6.3 Αποτελέσματα

Ορίζουμε κριτήριο τερματισμού τις 17.000 αξιολογήσεις στις οποίες θεωρούμε ότι η λύση θα έχει συγκλίνει. Πράγματι, παρακολουθώντας την εκτέλεση του λογισμικού EASY, λίγο πριν τις 17.000 αξιολογήσεις παρατηρούμε ότι οι μεταβλητές σχεδιασμού παρουσιάζουν πολύ μικρή μεταβολή.

Αναλυτικά οι είκοσι βέλτιστες λύσεις που προέκυψαν παρουσιάζονται στο παρακάτω σχήμα :



Σχ. 6.4 – Μέτωπο Pareto βέλτιστων λύσεων

Ενδεικτικά παραθέτονται οι τιμές των μεταβλητών σχεδιασμού των δυο ακραίων λύσεων που προέκυψαν από τη βελτιστοποίηση:

Μέγιστο βεληνεκές		
Μεταβλητή	Βέλτιστη τιμή	Μονάδες
hcr	16103	m
Mcr	1.504	
alfa	4.949	°
Dfus	2.0	m
Sw	131.828	m^2
Λlew	69.648	°
ARw	1.672	
λw	0.050	
t/cw	0.040	
Λlev	52.976	°
ARv	1.604	
λv	0.472	
t/cv	0.060	
Nult	0.752	

Ελάχιστο βάρος καυσίμου		
Μεταβλητή	Βέλτιστη τιμή	Μονάδες
hcr	16069	m
Mcr	1.503	
alfa	5.045	°
Dfus	2.0	m
Sw	120.195	m^2
Λlew	69.970	°
ARw	1.439	
λw	0.051	
t/cw	0.040	
Λlev	57.214	°
ARv	1.358	
λv	0.483	
t/cv	0.064	
Nult	0.754	

Επίσης παραθέτονται οι τιμές των μεταβλητών των κύριων χαρακτηριστικών του αεροσκάφους με βάση αυτές τις δυο λύσεις:

Μέγιστο βεληνεκές		
Παράμετρος	Τιμή	Μονάδες
Ειδική κατανάλωση καυσίμου	$2.938 \cdot 10^{-5}$	Kg/N/s
Διάμετρος κινητήρα	1.268	m
Μήκος κινητήρα	4.69	m
Μήκος ατράκτου	38.719	m
AR πτέρυγας	1.672	
Εκπέτασμα πτέρυγας	14.846	m
AR ουραίου πτερυγίου	1.604	
Εκπέτασμα ουραίου πτερυγίου	4.598	m
Γωνία πτήσης $alfa_{cr}$	4.949	°
Άνωση $C_{L_{alfa}}$	0.252	
Αντίσταση $C_{D_{alfa}}$	0.022	
Βάρος πτέρυγας	7211	Kg
Βάρος ουραίου πτερυγίου	570	Kg
Βάρος ατράκτου	6179	Kg
Βάρος μηχανισμού προσγείωσης	2293	Kg
Βάρος προωθητικού συστήματος	2595	Kg
Βάρος κινητήρα	1622	Kg
Βάρος βοηθητικών συστημάτων	2500	Kg
Βάρος υπόλοιπου εξοπλισμού	9745	Kg
Βάρος αεροσκάφους χωρίς καύσιμα	33692	Kg
Βάρος απογείωσης	57328	Kg

Ελάχιστο βάρος καυσίμου		
Παράμετρος	Τιμή	Μονάδες
Ειδική κατανάλωση καυσίμου	$2.938 \cdot 10^{-5}$	Kg/N/s
Διάμετρος κινητήρα	1.159	m
Μήκος κινητήρα	4.408	m
Μήκος ατράκτου	37.269	m
AR πτέρυγας	1.493	
Εκπέτασμα πτέρυγας	13.398	m
AR ουραίου πτερυγίου	1.358	
Εκπέτασμα ουραίου πτερυγίου	4.040	m
Γωνία πτήσης $alfa_{cr}$	5.045	°
Άνωση $C_{L_{alfa}}$	0.230	
Αντίσταση $C_{D_{alfa}}$	0.021	
Βάρος πτέρυγας	5353	Kg
Βάρος ουραίου πτερυγίου	481	Kg
Βάρος ατράκτου	5033	Kg
Βάρος μηχανισμού προσγείωσης	1919	Kg
Βάρος προωθητικού συστήματος	2273	Kg
Βάρος κινητήρα	1420	Kg
Βάρος βοηθητικών συστημάτων	2500	Kg
Βάρος υπόλοιπου εξοπλισμού	8158	Kg
Βάρος αεροσκάφους χωρίς καύσιμα	27993	Kg
Βάρος απογείωσης	47993	Kg

Ακολουθούν οι τιμές των στόχων που είχαμε θέσει:

Μέγιστο βεληνεκές		
Παράμετρος	Τιμή	Μονάδες
Βάρος καυσίμου	23636	Kg
Βεληνεκές	7881	km

Ελάχιστο βάρος καυσίμου		
Παράμετρος	Τιμή	Μονάδες
Βάρος καυσίμου	20000	Kg
Βεληνεκές	7637	km

Και τέλος των περιορισμών:

Μέγιστο βεληνεκές		
Παράμετρος	Τιμή	Μονάδες
Μήκος BFL	3974	m
Ταχύτητα προσέγγισης	97.902	m/sec

Ελάχιστο βάρος καυσίμου		
Παράμετρος	Τιμή	Μονάδες
Μήκος BFL	3969	m
Ταχύτητα προσέγγισης	99.880	m/sec

Ενδιαφέρον όμως παρουσιάζει ο χρόνος εκτέλεσης της διαδικασίας.

Με χρήση του εμπειρικού μοντέλου της άνωσης δεν παρατηρήθηκε μείωση του χρόνου εκτέλεσης και είναι φυσιολογικό καθώς το μοντέλο του αεροσκάφους από μόνο του, είναι πολύ «ελαφρύ» υπολογιστικά και η εκτέλεση του δεν είχε να προσφέρει κάποιο όφελος. Με την προσθήκη του υπολογιστικά κοστίζοντος επιλύτη, παρατηρήθηκε επιτάχυνση της εκτέλεσης, μικρότερης από την περίπτωση του κεφαλαίου 4, που αναμενόταν λόγο της μεγάλης μεταφοράς δεδομένων από και προς την κάρτα γραφικών σε συνδυασμό με την μηδενική δυνατότητα παραλληλίας του μοντέλου.

Κεφάλαιο 7ο
Συμπεράσματα

Στόχος της παρούσας διπλωματικής ήταν η ανάλυση της λειτουργίας της κάρτας γραφικών, η μελέτη της νέας τεχνολογίας ενοποιημένης αρχιτεκτονικής τους, η εκπαίδευση στο μοντέλο προγραμματισμού CUDA, η χρησιμοποίηση του για τον επαναπρογραμματισμό εφαρμογών με σκοπό την εκτέλεση τους στην κάρτα γραφικών και η διερεύνηση της επιτάχυνσης που εισάγει αυτή η νέα τεχνολογία στο τομέα της μεγάλης κλίμακας παράλληλων υπολογισμών.

Με βάση τα παραπάνω, επιλέχθηκε ως εφαρμογή ένα μοντέλο προκαταρκτικού σχεδιασμού μικρού υπερηχητικού αεροσκάφους, το οποίο ήδη υπήρχε και επαναπρογραμματίστηκε για εκτέλεση στην κάρτα γραφικών και το οποίο τέθηκε υπό βελτιστοποίηση με χρήση εξελικτικών αλγορίθμων. Μάλιστα, στο μοντέλο προστέθηκε ο υπολογισμός της πολικής καμπύλης με βάση την αεροτομή του αεροσκάφους του μοντέλου και χρήση έτοιμου λογισμικού επίλυσης ροής, διαδικασίας κοστίζουσας που δικαιολογεί το προγραμματισμό στην κάρτα γραφικών.

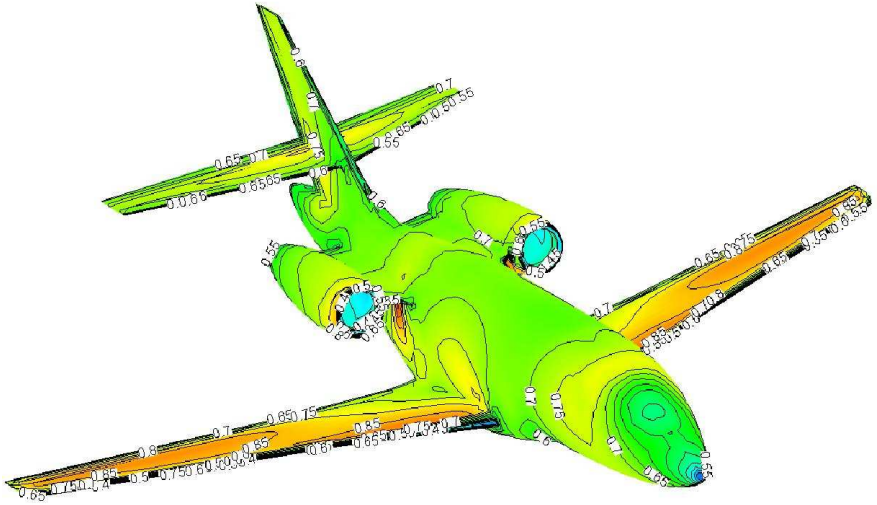
Με την ολοκλήρωση της διπλωματικής εργασίας συνοψίζονται τα παρακάτω συμπεράσματα:

1. Ο επαναπρογραμματισμός εφαρμογών, που χειρίζονται πίνακες δεδομένων και εκτελούν τις εσωτερικές τους διεργασίες παράλληλα όπως οι κώδικες υπολογιστικής ρευστοδυναμικής, με χρήση της γλώσσας CUDA, επιφέρει δραματική μείωση στο χρόνο εκτέλεσης τους.

2. Αντιθέτως, ο προγραμματισμός εφαρμογών οι οποίες εκτελούν σειριακά ενέργειες με σκοπό την επεξεργασία απλών βαθμωτών μεταβλητών, ακόμα και αν το πλήθος αυτών είναι πολύ μεγάλο, δεν επιφέρει κέρδη και μάλιστα ενδέχεται αν η μεταφορά δεδομένων μεταξύ κεντρικού επεξεργαστή και κάρτας γραφικών είναι συχνή, να οδηγηθούμε σε επιβράδυνση της εκτέλεσης.

3. Με βάση τα δύο παραπάνω συμπεράσματα, καταλήγουμε ότι η νέα τεχνολογία είναι κατάλληλη για χρήση σε εφαρμογές παράλληλου προγραμματισμού με μεγάλο υπολογιστικό κόστος. Επομένως στον τομέα της ευρείας κλίμακας παράλληλων υπολογισμών και ειδικά σε εφαρμογές υπολογιστικής ρευστοδυναμικής η χρήση της δεν ενδείκνυται απλά, αλλά επιβάλλεται. Χαρακτηριστικά συνοψίζουμε ότι με βάση την πρόσφατη εργασία του ΕΘΣ του ΕΜΠ επιτεύχθηκε επιτάχυνση της επίλυσης των εξισώσεων Navier-Stokes

(τυρβώδης ροή) σε δυο διαστάσεις έως και 28 φορές και των εξισώσεων Euler (ατριβής ροή) σε δυο διαστάσεις έως και 23 φορές και σε τρείς διαστάσεις έως και 28 φορές. Χαρακτηριστικά παρουσιάζεται το αεροσκάφος το οποίο χρησιμοποιήθηκε για την μελέτη της 3Δ ροής:



Σχ. 7.1 – Ατριβής ροή που αναπτύσσεται γύρω από πολιτικό αεροσκάφος. Προβάλλονται οι αριθμοί Mach στο περίγραμμά του.

4. Η νέα τεχνολογία της ενοποιημένης αρχιτεκτονικής είναι διαθέσιμη για μεγάλο εύρος καρτών γραφικών, αν και μόνο οι κάρτες της τελευταίας γενιάς της NVIDIA (GT200) έχουν την δυνατότητα εκτέλεσης πράξεων διπλής ακριβείας. Επομένως το πρόβλημα της ακρίβειας των αποτελεσμάτων με χρήση των σύγχρονων καρτών έχει ξεπεραστεί ενώ μοναδικό μειονέκτημα παραμένει η μειωμένη διαθέσιμη μνήμη της συσκευής, γεγονός που περιορίζει το μέγεθος του προβλήματος το οποίο μπορεί να διαχειριστεί. Τέλος, η γλώσσα προγραμματισμού CUDA είναι διαθέσιμη ελεύθερα στο κοινό, ενώ προγραμματιστές εξοικειωμένοι στη γλώσσα προγραμματισμού C/C++ δεν θα αντιμετωπίσουν πρόβλημα στη χρήση της για την κατασκευή των προγραμμάτων τους.

5. Το μοντέλο του προκαταρκτικού σχεδιασμού του αεροσκάφους, παρόλο που χρησιμοποιεί την επίλυση ροής γύρω από αεροτομή, δεν ελέγχει τη μορφή της μέσω μεταβλητών σχεδιασμού και επομένως κατά τη βελτιστοποίηση, η μορφή της παραμένει σταθερή και δεν βελτιστοποιείται. Για να επιτυγχανόταν κάτι τέτοιο θα έπρεπε να προγραμματιστεί το λογισμικό δημιουργίας πλέγματος που αναπτύχθηκε, για εκτέλεση στην κάρτα γραφικών. Όμως η λειτουργία του στηρίζεται σε εμπορικό λογισμικό που εφαρμόζει την τεχνική προελαύνοντος μετώπου για τη δημιουργία του πλέγματος, το οποίο διατίθεται αποκλειστικά για εκτέλεση στον κεντρικό επεξεργαστή. Η επαναλαμβανόμενη μεταφορά πληροφοριών από την κάρτα προς τον επεξεργαστή σε κάθε επανάληψη για τη

δημιουργία νέου πλέγματος για κάθε μορφή της αεροτομής, είναι διαδικασία με μεγάλο υπολογιστικό κόστος και απαγορευτική για την περίπτωση μας. Πρόταση για μελλοντική μελέτη θα ήταν η δημιουργία νέου λογισμικού δημιουργίας πλέγματος στην κάρτα γραφικών και η συνεργασία του με λογισμικό επίλυσης ροής που θα επέτρεπε τον υπολογισμό της βέλτιστης αεροτομής σε μοντέλα προκαταρκτικού σχεδιασμού αεροσκάφους όπως αυτό που μελετήθηκε.

Παράρτημα Α
Γένεση Πλέγματος

Εισαγωγή

Στο πλαίσιο αυτής της διπλωματικής, χρειάστηκε, για τις ανάγκες της εκτέλεσης του επιλύτη ροής, να αναπτύξουμε ένα νέο πρόγραμμα δημιουργίας διδιάστατου υπολογιστικού μη δομημένου πλέγματος (Grid Generator) γύρω από αεροτομές.

Για το πρόγραμμα χρησιμοποιήθηκε η γλώσσα προγραμματισμού FORTRAN, ενώ να σημειώσουμε ότι το πρόγραμμα μας «πατάει» πάνω στο ήδη υπάρχον λογισμικό δημιουργίας πλέγματος **front5**.

Στην συνέχεια ακολουθεί μια σύντομη περιγραφή της λειτουργίας του προγράμματος.

Το λογισμικό μας λειτουργεί με την εκτέλεση του προγράμματος **grid_generator.exe** το οποίο είναι υπεύθυνο για την διαχείριση έξι υποπρογραμμάτων, τα οποία συνεργαζόμενα μας δίνουν το αποτέλεσμα, ενώ η είσοδος δεδομένων από τον χρήστη για τον έλεγχο του προγράμματος, γίνεται μέσω τριών βοηθητικών αρχείων.

Κύριο χαρακτηριστικό του προγράμματος είναι ότι το τελικό πλέγμα θα προκύψει από τη σύνθεση επιμέρους μικρότερων πλεγμάτων τα οποία χωρίζονται σε δυο κατηγορίες: στα πλέγματα που έρχονται σε επαφή με την αεροτομή μας και με τη σύνθεση τους θα προκύψει το πυκνό και κοντινό στο αεροσκάφος πλέγμα και τα οποία θα τα ονομάζουμε εσωτερικά, και στα πλέγματα που θα περιβάλλουν τα προηγούμενα και θα είναι πιο αραιά και θα τα ονομάσουμε εξωτερικά πλέγματα.

Με την εκτέλεση του **grid_generator.exe** η ροή της λειτουργίας του λογισμικού έχει ως εξής:

Η διαδικασία δημιουργίας πλέγματος ξεκινά από το αρχείο **cp.bzp** το οποίο περιέχει τα σημεία ελέγχου Bezier **[32]** της αεροτομής. Το αρχείο διαβάζεται από το πρόγραμμα **airfoil.exe** του οποίου σκοπός είναι να δημιουργήσει το περίγραμμα της αεροτομής με κατάλληλες πυκνώσεις ανά τμήματα της αεροτομής ώστε να προκύψει στη συνέχεια πυκνό πλέγμα σε συγκεκριμένες θέσεις της αεροτομής που χρειαζόμαστε μεγαλύτερη ακρίβεια όπως το σημείο πρόσπτωσης και το σημείο εκφυγής.

Στη συνέχεια, κατασκευάζεται η εξωτερική καμπύλη με βάση την εξίσωση της έλλειψης και αν χρειαστεί για τις ανάγκες του προβλήματος μας, πυκνώνεται και αυτή κατά θέσεις με τη χρήση του προγράμματος **elipsis.exe.**

Στη συνέχεια, το πρόγραμμα **makefr.exe** είναι αυτό που αναλαμβάνει να δημιουργήσει τα χωρία τα οποία, στη συνέχεια το **front5.exe** θα τα γεμίσει με τρίγωνα δημιουργώντας το πλέγμα.
Τέλος η συρραφή των επιμέρους χωρίων που θα δημιουργηθούν γίνεται με την εκτέλεση των προγραμμάτων **prematch.exe** και **matchkirk.exe.**

Ακολουθεί περιγραφή της λειτουργίας του κάθε υποπρογράμματος ξεχωριστά :

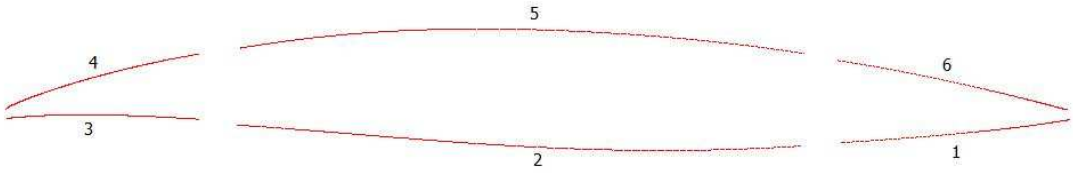
Πρόγραμμα airfoil.exe

Όπως είπαμε, το πρόγραμμα θα διαβάσει τα σημεία ελέγχου Bezier της αεροτομής και θα δίνει την τελική μορφή της αεροτομής.

Η ίδια η υπορουτίνα Bezier θα μπορούσε να δώσει απευθείας τη μορφή της αεροτομής χρησιμοποιώντας τα σημεία ελέγχου, όμως το αποτέλεσμα δεν μας ικανοποιεί, γιατί όλα τα σημεία της αεροτομής θα ακολουθούν την ίδια πύκνωση γεγονός που θα οδηγήσει σε ένα ομοιόμορφο πλέγμα.

Εμείς όμως θέλουμε να δώσουμε μεγαλύτερη βαρύτητα στα κρίσιμα τμήματα της αεροτομής και γι'αυτό χρησιμοποιούμε υπορουτίνες παρεμβολής σημείων για να πυκνώσουμε κατάλληλα τις περιοχές που θέλουμε, προσέχοντας παράλληλα η κατανομή των σημείων γύρω από τα σημεία ένωσης να είναι ομαλή.

Για να το κάνουμε αυτό χωρίζουμε την αεροτομή μας σε 6 τμήματα όπως φαίνεται στο παρακάτω σχήμα και επεξεργαζόμαστε ξεχωριστά κάθε ένα από αυτά. Αρχικά τη χωρίζουμε στην πλευρά πίεσης και στην πλευρά υποπίεσης και στη συνέχεια το κάθε τμήμα σε 3 τμήματα, το μπροστά το πίσω και το κεντρικό.



Σχ. ΠΑ.1 – Τμήματα αεροτομής

Σε καθένα από αυτά τα τμήματα ο χρήστης με χρήση υπορουτινών αριθμητικής παρεμβολής[24], μπορεί να μεταβάλει την κατανομή των σημείων παρεμβάλλοντας και κατανέμοντας νέα σημεία ανάλογα με την ποιότητα της καμπύλης που επιθυμεί.

Για να εισαχθούν τα δεδομένα για τέτοια επεξεργασία το πρόγραμμα διαβάζει το αρχείο **air.in** της οποίας η μορφή παρουσιάζεται παρακάτω και επεξηγείται.

[24] Οι υπορουτίνες παρεμβολής που χρησιμοποιήθηκαν προέρχονται από βιβλιοθήκη υπορουτινών του ΕΘΣ.

$$
\begin{array}{llll}
300 & 700\} & \textit{Σημεία Διαχωρισμού} & \textit{Πλευράς Υποπίεσης} \\
1 & & & \\
316 \quad 1 & 0.987\} & \textit{Πίσω τμήμα} & \\
1 & & & \\
501 \quad 1 & 1.001\} & \textit{Μεσαίο τμήμα} & \Big\} \textit{Πλευρά Υποπίεσης} \\
1 & & & \\
790 \quad 1 & 1.0055\} & \textit{Μπροστά τμήμα} & \\
250 & 700\} & \textit{Σημεία Διαχωρισμού} & \textit{Πλευράς Πίεσης} \\
1 & & & \\
700 \quad 1 & 1.0056\} & \textit{Πίσω τμήμα} & \\
1 & & & \\
551 \quad 1 & 1.0010\} & \textit{Μεσαίο τμήμα} & \Big\} \textit{Πλευρά Πίεσης} \\
1 & & & \\
287 \quad 1 & 0.985\} & \textit{Μπροστά τμήμα} &
\end{array}
$$

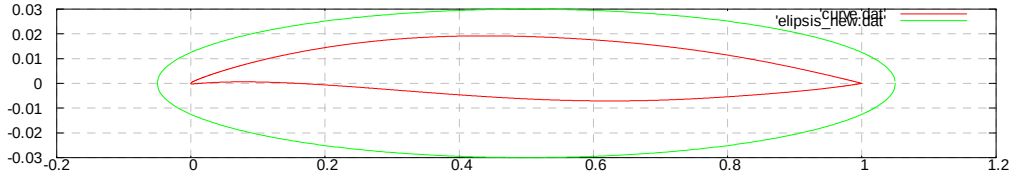
Οι πρώτες 7 γραμμές αναφέρονται στην πλευρά πίεσης και οι επόμενες 7 στην πλευρά υποπίεσης. Συγκεκριμένα η $1^{η}$ γραμμή αναφέρει σε ποια σημεία θα χωριστεί η κάτω πλευρά της αεροτομής για να δημιουργηθούν τα 3 τμήματα, ενώ για τις γραμμές που ακολουθούν αρχικά το νούμερο δείχνει ποια μέθοδο παρεμβολής θα χρησιμοποιηθεί (1 για geosimple) ενώ τα 3 νούμερα στην επόμενη γραμμή δείχνουν τα σημεία που θέλουμε να έχουμε τελικά (316), το είδος της κατανομής που θέλουμε να προκύψει (πύκνωση σε ένα άκρο ή στο κέντρο) και τέλος το λόγο που θα ακολουθήσει αυτή η πύκνωση.

Έτσι με την εκτέλεση αυτού του προγράμματος προκύπτει το αρχείο **curve.dat** στο οποίο περιέχονται όλα τα σημεία που περιγράφουν την αεροτομή σε (x,y) συντεταγμένες.

Πρόγραμμα ellipsis.exe

Η μορφή του προγράμματος αυτού είναι παρόμοια με το **airfoil.exe** με κύρια διαφορά ότι αντί να λαμβάνει σημεία Bezier αεροτομής για να ξεκινήσει, χρησιμοποιεί την εξίσωση της έλλειψης για να παράγει τα αρχικά σημεία, τα οποία στη συνέχεια θα πυκνώσει.

Ακολουθεί εικόνα με την αεροτομή μας και την εξωτερική έλλειψη:



Σχ. ΠΑ.2 – Αεροτομή /Εξωτερική Έλλειψη

Πρόγραμμα makefr.exe

Το πρόγραμμα αυτό είναι υπεύθυνο για τη δημιουργία του κάθε χωρίου στο οποίο στη συνέχεια θα δημιουργηθεί το κάθε εσωτερικό πλέγμα.

Το κάθε χωρίο από αυτά, (ο συνολικός αριθμός τους είναι 4, αλλά ο κώδικας δίνει τη δυνατότητα στο χρήστη να δημιουργήσει ότι πλήθος χωρίων χρειαστεί.) αποτελείται από 4 τμήματα. Το τμήμα της αεροτομής, το τμήμα της εξωτερικής καμπύλης (στην περίπτωση μας έλλειψη), και από δυο τμήματα τα οποία συνδέουν τα δύο άκρα των δυο προηγούμενων τμημάτων.

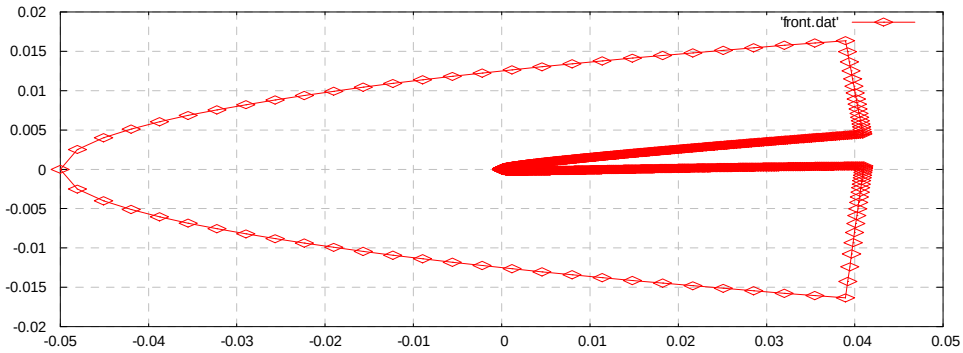
Το τμήμα της αεροτομής και της έλλειψης χρησιμοποιεί τα σημεία που έχουν προκύψει από τα προηγούμενα προγράμματα, ενώ τα δύο τμήματα σύνδεσης χρησιμοποιούν και αυτά σχήματα παρεμβολής για να εισάγουν όσα σημεία επιθυμεί ο χρήστης και στην επιθυμητή κατανομή.

Έτσι ο χρήστης θέτοντας τις παραμέτρους που χρειάζονται για την παραπάνω διαδικασία στο αρχείο **mfr.in** το οποίο διαβάζει το πρόγραμμα **makefr.exe**, κατασκευάζει το χωρίο στο οποίο θα δημιουργηθεί το πλέγμα.
Το αρχείο **mfr.in** έχει την εξής μορφή :

```
   4                  } Αριθμός Χωρίων
 100   1130           } Τμήμα Αεροτομής
 407    675           } Τμήμα Ελλειψης
  22    1    1.13     } Κάθετα Τμήματα
  30    1    1.1      } 1ο Χωρίο
1130   2049           } Τμήμα Αεροτομής
 353    407           } Τμήμα Ελλειψης
  18    1    1.11     } Κάθετα Τμήματα
  22    1    1.13     } 2ο Χωρίο
2049   3042           } Τμήμα Αεροτομής
  85    353           } Τμήμα Ελλειψης
  30    1    1.1      } Κάθετα Τμήματα
  18    1    1.11     } 3ο Χωρίο
3042    100           } Τμήμα Αεροτομής
 675     85           } Τμήμα Ελλειψης
  30    1    1.1      } Κάθετα Τμήματα
  30    1    1.1      } 4ο Χωρίο
```

Η πρώτη γραμμή δείχνει από πόσα χωρία θα αποτελείται το εσωτερικό πλέγμα.
Παρακάτω για κάθε ένα από τα χωρία αυτά δίνεται στην 1η γραμμή από ποιος μέχρι ποιον κόμβο της αεροτομής θα χρησιμοποιηθεί (από τον 100 έως τον 1130), στην 2η γραμμή από ποιος μέχρι ποιος κόμβος της εξωτερικής καμπύλης θα χρησιμοποιηθεί (407 675) ενώ οι άλλες 2 περιέχουν πληροφορίες για τα τμήματα σύνδεσης και συγκεκριμένα τον αριθμό των σημείων το τύπο της κατανομής και το λόγο που θα ακολουθήσει η πύκνωση.
Έτσι με την εκτέλεση του προγράμματος αυτού προκύπτει το αρχείο **front.dat** στο οποίο περιέχονται τα σημεία του χωρίου σε συντεταγμένες (x,y).

Ακολουθεί εικόνα ενός από τα εσωτερικά μας χωρία που δημιουργήθηκαν.



Σχ. ΠΑ.3 – Εσωτερικό Χωρίο

Πρόγραμμα front5.exe

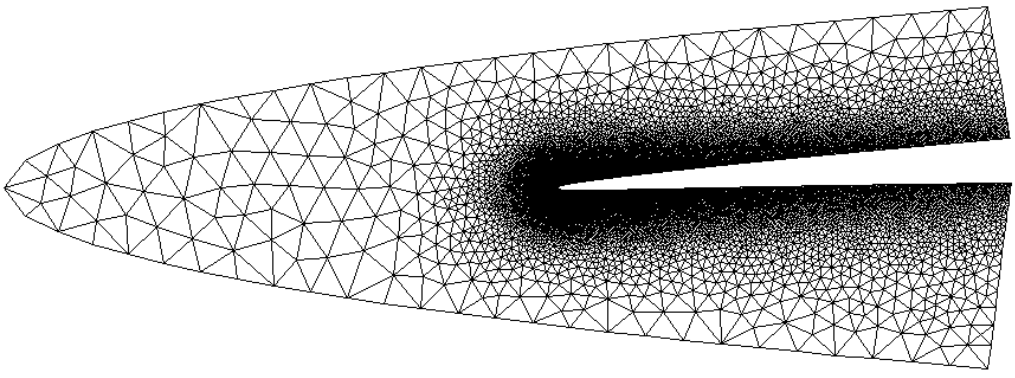
Το πρόγραμμα αυτό προϋπήρχε στο Εργαστήριο Θερμικών Στροβιλομηχανών του ΕΜΠ από το εργαστήριο. Ως είσοδο λαμβάνει το αρχείο front.dat που περιλαμβάνει το χωρίο και δημιουργεί το πλέγμα σε αρχεία της μορφής grid.ele και grid.nod.

Πρόγραμμα manage.exe

Τέλος το πρόγραμμα αυτό συντονίζει τις παραπάνω ενέργειες .

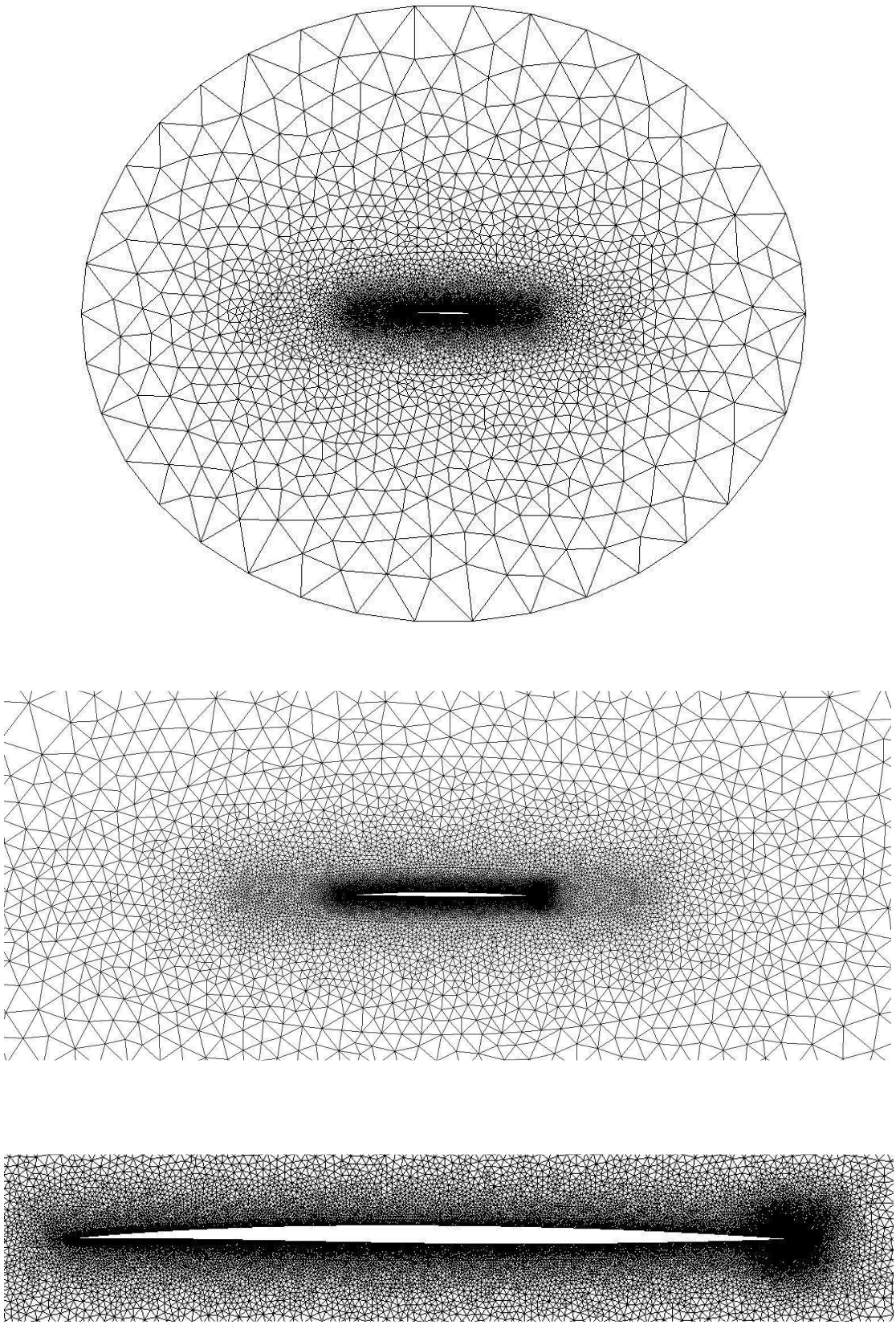
Αρχικά δημιουργεί το κάθε ένα από τα εσωτερικά πλέγματα, στη συνέχεια τα ενώνει μεταξύ τους δημιουργώντας το ενιαίο εσωτερικό πλέγμα και τέλος δημιουργεί τα εξωτερικά πλέγματα και τα συνδέει με τα υπάρχοντα, φτιάχνοντας έτσι το τελικό ενιαίο υπολογιστικό μας πλέγμα.

Ακολουθεί εικόνα από το πλέγμα του παραπάνω χωρίου:



Σχ. ΠΑ.4 – Εσωτερικό Πλέγμα

Τέλος παραθέτουμε το τελικό πλέγμα που προέκυψε σε τρείς διαφορετικές μεγεθύνσεις:



Σχ. ΠΑ.5 – Υπολογιστικό Πλέγμα

Παράρτημα Β
Εξελικτικοί Αλγόριθμοι

Εισαγωγή

Από την δεκαετία του 1990 και έκτοτε, η ραγδαία αύξηση της υπολογιστικής ισχύος και η μείωση του κόστους μεγάλων και γρήγορων υπολογιστικών συστημάτων οδήγησε στην ανάπτυξη στοχαστικών μεθόδων βελτιστοποίησης με κυριότερους εκπροσώπους, τους εξελικτικούς αλγόριθμους **[4,30,31].**

Κύριο χαρακτηριστικό τους αποτελεί το μη-μαθηματικό υπόβαθρο τους, η ευκολία με την οποία προσαρμόζονται σε κάθε νέο πρόβλημα αρκεί να υπάρχει προγραμματισμένο λογισμικό αξιολόγησης κάθε υποψήφιας λύσης, η δυνατότητα τους να μην εγκλωβίζονται σε τοπικά ακρότατα αλλά και το μειονέκτημα τους να απαιτούν μεγάλο αριθμό αξιολογήσεων για τον εντοπισμό της βέλτιστης λύσης.

Βασικά γνώρισμα των εξελικτικών αλγόριθμων (ΕΑ) είναι ο χειρισμός πληθυσμού υποψηφίων λύσεων και όχι μιας μεμονωμένης λύσης σε κάθε επανάληψη, ενώ έχουν την δυνατότητα να αντιμετωπίζουν τόσο προβλήματα μονοκριτιριακά όσο και πολυκριτιριακά.

Β.1 Βασικές Διεργασίες

Η λειτουργία των εξελικτικών, με σκοπό την αναζήτηση της βέλτιστης λύσης, στηρίζεται στην διαδικασία της εξέλιξης. Σύμφωνα με αυτή, ένα πληθυσμός **μ** υποψηφίων λύσεων, με χρήση συγκεκριμένων τελεστών, εξελίσσεται στον πληθυσμό των **λ** απογόνων που αποτελούν τις νέες λύσεις και έχουν ενδεχομένως καλύτερα χαρακτηριστικά. Από αυτούς με κριτήριο την καταλληλότητα τους επιλέγονται οι **μ** γονείς της επόμενης γενιάς και η διαδικασία επαναλαμβάνεται, γενιά με γενιά, μέχρι την ικανοποίηση κάποιου κριτηρίου σύγκλισης, όπου και θεωρούμε ότι έχουμε οδηγηθεί στην βέλτιστη λύση.

Με αυστηρότερη θεμελίωση, θεωρούμε ότι σε κάθε γενιά υπάρχουν τρία διακριτά σύνολα πληθυσμών. Το σύνολο των γονέων $S^{g,\mu}$ με μ μέλη, το σύνολο των απογόνων $S^{g,\lambda}$ με λ μέλη και το σύνολο των επίλεκτων ατόμων $S^{g,e}$ όπου g είναι ο αύξων αριθμός της γενιάς. Το σύνολο $S^{g,e}$ αποθηκεύει τα επίλεκτα-καλύτερα άτομα που έχουν προκύψει μέχρι τη συγκεκριμένη γενιά και επομένως σε οποιαδήποτε στιγμή τερματισμού της διαδικασίας περιέχει της βέλτιστες λύσεις.

Β.2 Αλγόριθμος Εκτέλεσης

Θεωρώντας πρόβλημα ενός στόχου, ο εξελικτικός αλγόριθμος βελτιστοποίησης περιγράφεται από τα παρακάτω βήματα.

Βήμα 1^{ο}:
Επιλέγεται το μέγεθος των πληθυσμών μ και λ ενώ αρχικοποιείται η πρώτη γενιά με g=0 και επιλέγονται τυχαία τα μέλη του αρχικού της πληθυσμού.

Βήμα 2^{ο}:
Αξιολογούνται τα λ άτομα του συνόλου, με χρήση εξωτερικού λογισμικού αξιολόγησης. Στην περίπτωση του προβλήματος σχεδιασμού του αεροσκάφους, το λογισμικό αξιολόγησης είναι το ίδιο το μοντέλο του προκαταρκτικού σχεδιασμού. Μάλιστα όταν αυτό εκτελεστεί με χρήση επίλυσης ροής τότε το βήμα 2 είναι που φέρει το μεγαλύτερο υπολογιστικό κόστος.

Βήμα 3^{ο}:
Ανανεώνεται το σύνολο των επίλεκτων με κριτήριο την τιμή της αντικειμενικής τους συνάρτησης. Έτσι κατά το βήμα αυτό είναι δυνατόν άτομα του $S^{g,\lambda}$ να αντικαταστήσουν μέλη του $S^{g,e}$ αν αποκτήσουν καλύτερη τιμή της αντικειμενικής συνάρτησης. Για το βήμα αυτό, χρησιμοποιείται ο τελεστής εντοπισμού των επίλεκτων μελών T_e.

Βήμα 4^{ο}:
Εφαρμόζεται ο τελεστής επιλεκτικότητας T_{e2} , με τον οποίο άτομα από το σύνολο $S^{g,e}$ αντικαθιστούν άτομα του$S^{g,\lambda}$. Έτσι γίνεται εμπλουτισμός των πιθανών λύσεων με τις καλύτερες που έχουν υπάρξει μέχρι την συγκεκριμένη γενιά, διαδικασία που οδηγεί σταδιακά στην βέλτιστη λύση.

Βήμα 5^{ο}:
Εφαρμόζεται ο τελεστής επιλογής γονέων T_μ, με την βοήθεια του οποίου γίνεται η επιλογή του συνόλου των μελών του νέου πληθυσμού γονέων της επόμενης γενιάς $S^{g+1,\mu}$, δίνοντας μεγαλύτερες πιθανότητες στα άτομα του $S^{g,\lambda}$ με καλύτερη τιμή αντικειμενικής συνάρτησης να συμμετέχουν στην δημιουργία απογόνων. Για τον σχηματισμού του νέου πληθυσμού μπορούν να χρησιμοποιηθούν είτε οι τρέχοντες απόγονοι λ , είτε οι απόγονοι λ σε συνδυασμό με τους γονείς μ.

Βήμα 6^{ο}:
Ακολουθεί η διαδικασία αναπαραγωγής, δηλαδή η δημιουργία της επόμενης γενιάς η οποία επιτυγχάνεται με την εφαρμογή τελεστών μετάλλαξης T_m και διασταύρωσης T_r στο σύνολο των γονέων της τρέχουσας γενιάς.

Βήμα 7^{ο}:
Εφαρμόζεται κατάλληλο κριτήριο σύγκλισης όπως το να μην βελτιώνεται περαιτέρω η λύση για έναν συγκεκριμένο αριθμό αξιολογήσεων ή το να έχει ομογενοποιηθεί ο

πληθυσμός ή η ανάλωση του υπολογιστικού χρόνου που επέτρεπε ο χρήστης, και αν η μέθοδος θεωρείται ότι δεν έχει συγκλίνει αρχίζει μια νέα γενιά με βάση τα βήματα 2 έως 6 θέτοντας g←g+1.

Β.3 Πρόσθετοι Τελεστές

Με βάση τον παραπάνω αλγόριθμο, εκτός από τους βασικούς για την λειτουργία τελεστές επίλεκτων μελών, επιλεκτικότητας και επιλογής γονέων, όπως αναφέρθηκε μπορούν να χρησιμοποιηθούν και δυο επιπλέον τελεστές, που οδηγούν στην διεύρυνση του πεδίου ανίχνευσης των λύσεων και οι οποίοι είναι:

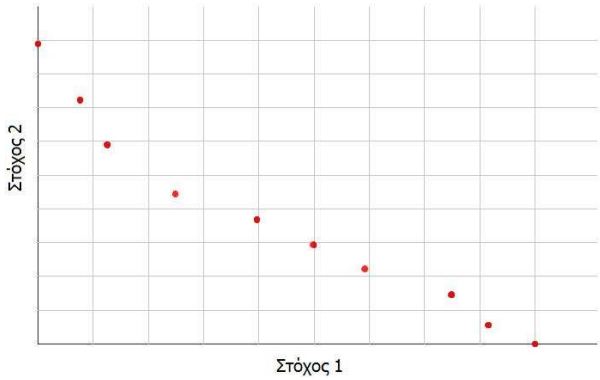
⇨**Τελεστές διασταύρωσης**, οι οποίοι χρησιμοποιούνται κατά την διασταύρωση, κατά την οποία δύο ή περισσότεροι γονείς δημιουργούν δύο απογόνους.

⇨**Τελεστές μετάλλαξης**, που χρησιμοποιούνται κατά την μετάλλαξη, διαδικασία η οποία στοχεύει στην εισαγωγή νέου γενετικού υλικού στον πληθυσμό των απογόνων, με (συνήθως πολύ μικρή) συγκεκριμένη πιθανότητα.

Β.4 Βέλτιστες λύσεις

Όπως αναφέραμε παραπάνω το σύνολο $S^{g,e}$ περιέχει τις βέλτιστες λύσεις κάθε γενιάς και η διάσταση του μπορεί να είναι οποιοσδήποτε αριθμός. Επομένως όταν ικανοποιηθεί το κριτήριο σύγκλισης και τερματιστεί ο αλγόριθμος, θα περιέχει και τις τελικές λύσεις που προέκυψαν.

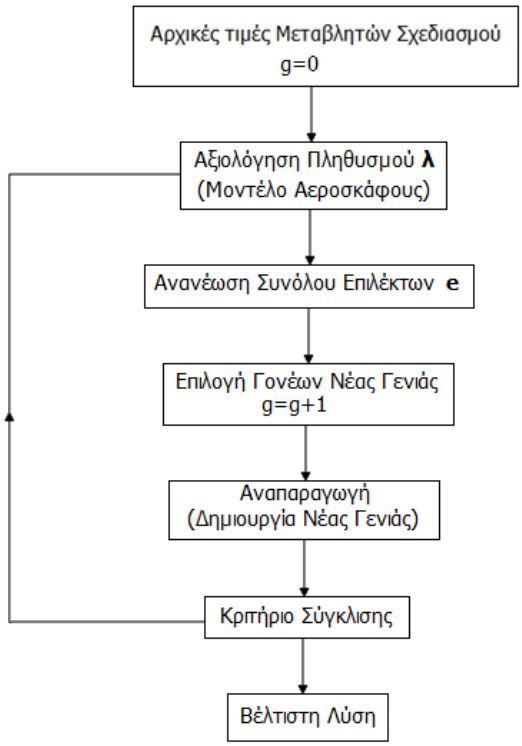
Για μονοκριτιριακό πρόβλημα, ανεξάρτητα από το μέγεθος του συνόλου μία θα είναι η βέλτιστη του λύση. Αντίθετα, για πρόβλημα δυο στόχων όλες οι λύσεις του συνόλου θα είναι βέλτιστες, με μόνη διαφορά μεταξύ τους ότι κάθε μια θα ικανοποιεί περισσότερο τον έναν στόχο από τον άλλο. Το διάγραμμα με άξονες τους δύο στόχους και σημεία τις βέλτιστες λύσεις ονομάζεται μέτωπο Pareto και στο παρακάτω σχήμα παρουσιάζεται χαρακτηριστικό παράδειγμα με $N_e = 10$.



Σχ. Β.1 – Μέτωπο Pareto

Β.4 Εφαρμογή

Ακολουθεί ο αλγόριθμος βελτιστοποίησης της εφαρμογής μας:



Σχ. Β.2 – Αλγόριθμος Βελτιστοποίησης με ΕΑ

Η χρήση βελτιστοποίησης με εξελικτικούς αλγόριθμους στο πλαίσιο της διπλωματικής έγινε μέσω του έτοιμου λογισμικού EASY (Evolutionary Algorithm System) **[15]**, στο οποίο προσαρμόστηκε το δικό μας λογισμικό αξιολόγησης του μοντέλου του αεροσκάφους.

Βιβλιογραφία

[1] Νικόλαος Σ.Καζαζάκη:, **Βελτιστοποίηση Υπερηχητικού Επιβατικού Αεροσκάφους με Χρήση της Μεθόδου των Μιγαδικών Μεταβλητών**, Διπλωματική Εργασία , Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Μηχανολόγων Μηχανικών.

[2] Βαρβάρα Ασούτη: **Ανάλυση και Σχεδίαση Πτερυγώσεων και Πτερύγων σε Πολυεπεξεργαστικό Περιβάλλον**, Διδακτορική Διατριβή, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Μηχανολόγων Μηχανικών.

[3] D. Raymer: **Aircraft Design – A Conceptual Approach 3nd edition**, AIAA, 1992.

[4] Κ. Χ. Γιαννάκογλου: **Μέθοδοι Βελτιστοποίησης στην Αεροδυναμική**, Εκδόσεις ΕΜΠ, 2006.

[5] I.C.Kampolis, X.S. Trompoukis, V.G. Asouti, K.C. Giannakoglou: **CFD-based analysis and two-level aerodynamic optimization on GPU**, Computer Methods in Applied Mechanics and Engineering, Submitted, 2009.

[6] T. Brandvik, G. Pullan: **Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware** . J. Mechanical Engineering Science, Vol. 221 Part C, 2007.

[7] E. Elsen, P. LeGresley, E. Darve**: Large calculation of the flow over a hypersonic vehicle using a GPU**, Journal of Computational Physics 227 pg.10148-10161, 2008.

[8] N. Goodnight, G. Lewin, D. Luebke, K. Skadron: **A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware**, University of Virginia Technical Report CS-2003-03.

[9] T.R. Hagen, K.-A. Lie, J.R. Natvig: **Solving the Euler Equations on GPUs**. Computaional Science – ICCS 2006, Volume 3994, 2006

[10] T. Brandvik, G. Pullan: **Acceleration of a 3D Euler Solver Using GPUs,** Presented at the 46th AIAA Aerospace Sciences Meeting, Reno, NV, 2008

[11] K.C. Giannakoglou, I.C. Kampolis, P.I.K. Liakopoulos, M.K. Kakarasis, D.I.Papadimitriou, T. Zervogiannis, V.G. Asouti: **Aerodynamic Shape Optimization Methods on Multiprocessor Platforms, submitted to Computational Methods in Applied Mechanics and Engineering, 2009**

[12] CUDA Programming Guide 2.0

[13] www.workstationspecialist.com/hpc/gpu_computing

[14] www.aoe.vt.edu/~mason/Mason/ACinfoTOC.html

[15] http://velos0.ltt.mech.ntua.gr/EASY/

[16] www.hbeonlabs.com/detailcuda.htm

[17] http://stormeffect.com/blog/?p=58

[18] http://en.wikipedia.org/wiki/Moore's_law

[19] http://en.wikipedia.org/wiki/Vertex_shader

[20] http://en.wikipedia.org/wiki/Rasterizer

[21] http://en.wikipedia.org/wiki/Fragment_processing

[22] http://www.hardwarecanucks.com/forum/hardware-canucks-reviews/11907-bfg-gtx-260-ocx-maxcore-216sp-896mb-video-card-review-3.html

[23] http://www.maxxpi.net/pages/result-browser/top10---flops.php

[24] http://en.wikipedia.org/wiki/GeForce_200_Series

[25] http://en.wikipedia.org/wiki/Spalart%E2%80%93Allmaras_turbulence_model

[26] http://www.behardware.com/articles/691-1/intel-core-2-extreme-qx9650.html

[27] P.I.K. Liakopoulos, I.C. Kampolis, K.C. Giannakoglou: **Grid-Enabled, Hierarchical Distributed Metamodel-Assisted Evolutionary Algorithms for Aerodynamic Shape Optimization**, Future Generation Computer Systems, Vol. 24, pp. 701-708, 2008.

[28] M. Karakasis, D.G. Koubogiannis, K.C. Giannakoglou: **Hierarchical Distributed Evolutionary Algorithms in Shape Optimization**, International Journal for Numerical Methods in Fluids, Vol. 53, pp. 455-469, 2007.

[29] K.C. Giannakoglou, D.I. Papadimitriou, I.C. Kampolis: **Aerodynamic Shape Design Using Evolutionary Algorithms and New Gradient-Assisted Metamodels**, Computer Methods in Applied Mechanics and Engineering, Vol. 195, pp. 6312-6329, 2006.

[30] I.C. Kampolis, D.I. Papadimitriou, K.C. Giannakoglou: **Evolutionary Optimization Using a New Radial Basis Function Network and the Adjoint Formulation**, Journal of Inverse Problems in Science & Engineering, Vol. 14, No. 4, 397-410, 2006

[31] V.G. ASOUTI, K.C. Giannakoglou: **Aerodynamic Optimization Using a Parallel Asynchronous Evolutionary Algorithm Controlled by Strongly Interacting Demes**, Engineering Optimization, Vol. 41, No. 3, pp. 241-257, 2009

[32] http://en.wikipedia.org/wiki/B%C3%A9zier_curve