Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή εργασία

Τίτλος: Ανάπτυξη εφαρμογής για εκτίμηση επιδόσεων της κάρτας γραφικών σε περιβάλλον CUDA (Massively Parallel Computing on GPUs)

Καπίρης Ιωάννης (ΑΜ: 686)

Μαργαρίτης Αθανάσιος(ΑΜ : 737)

Επιβλέπων καθηγητής: Γεώργιος Κορνάρος

Επιτροπή Αξιολόγησης :

Ημερομηνία παρουσίασης:

Abstract

This paper deals with the lack of a complete solution that would assess the performance of a CUDA Enabled GPU on a CUDA Programming Enviroment

In the last years there has been a change from frequency driven updates on CPU , to more multicore based systems. To take advantage of the multicore and multithreaded hardware, software must turn into multithreaded parallel programming strategies. That gave birth to the concept of GPGPU.

GPGPU or General Processing on Graphics Processing Units, is the technology and programming tactics that enable software to be run on Graphics Card instead of the CPU. The disadvantage of that is that the Data being manipulated by the GPU must be in a form that is already compatible with GPU's, and the data manipulation must be in a similar way as the one the GPU calculates graphics.

A solution to that disadvantage was given by Nvidia in 2006, with the Introductioni of CUDA. Cuda is a GPGPU programming platform, that gives users a much greater flexibility when it comes to dealing with data , and also provides the programmer with a High Level Programming Language (Cuda C) similar to the already popular C.

Our program is a Benchmark written partly on CUDA C, that can be run on Moden Cuda Enabled GPU's, and assess their performance in real life situations. It does a number of calculations such as Memory Bandwidth , Matrix multiplication and Line of Sight Calculations and it provides the user with an easy way to Visualize those results and keep an archive or previous runs for later consulting.

We run our Benchmark on a number of different cards and systems, to make sure that our results are consistend with the Graphics Cards characteristics.Our numbers made sense when it became obvious that the Overall Performance of a GPU is not based solely on clocks and number or processors, but other factors such as PCI-E bandwidth, and Memory Bus Width have to be taken into account.

Overall we consider that we offer a complete solution when it comes to measuring a GPU's performance in Cuda, but the most important lesson taught, was that the future of Processing is taking a turn from Serial To Massively Parallel

Σύνοψη

Η εργασία αυτή ασχολείται με την ελλείψη μία πλήρους λυσης που να μετράει τις επιδόσεις μιας κάρτας γραφικών CUDA σε περιβάλον CUDA

Τα τελευταία χρόνια παρατηρείται μια αλλαγή από την τάση να αυξάνονται συνεχώς οι συχνότητες των επεξεργαστων, σε συστήματα με πολλαπλούς επεξεργαστές η πολλαπλούς πυρήνες. Για να μπορέσει το λογισμικό να ακολουθήσει αυτή την τάση, πρέπει να ακολουθήσει νέες τακτικές ώστε ο κώδικας μας να είναι παράλληλος και πολυνηματικός. Αυτή η πρακτική γέννησε την GPGPU

GPGPU ή Υπολογισμοί Γενικού Σκοπού σε Επεξεργαστές Γραφικών είναι η τεχνολογία που επιτρέπει σε λογισμικό να τρέχει σε κάρτες γραφικών αντί για τον επεξεργαστή. Το μεγάλο μειονέκτημα είναι ότι τα προς επεξεργασία δεδομένα, θα πρέπει να μιάζουν με τα δεδομένα που συνήθως επεξεργάζεται μια κάρτα γραφικών, και το είδος της επεξεργάσιας να είναι παρόμοιο με τον τρόπο που μια καρτα γραφικών επεξεργάζεται τα γραφικά.

Λύση στο πρόβλημα αυτό έδωσε η Nvidia το 2006 με την παρουσίαση της CUDA. Η Cuda είναι μια πλατφόρμα προγραμματισμού βασισμένη στις αρχές του GPGPU που δίνει όμως στους προγραμματιστές μια μεγαλύτερη ευελιξία όσον αφορά τα δεδομενα που θα επεξεργαστούν, αλλα και στο είδος της επεξεργασίας που θα τους γίνει. Προσφέρει επείσης ένα περιβάλον Γλώσσας Υψηλού Επιπέδου (CUDA C) που είναι παρόμοια με την γνωστή C

Το πρόγραμμά μας είναι ένα μετροπρόγραμμα, γραμένο εν μέρει σε Cuda C, που μπορεί να εκτελειται σε κάρτες Γραφικών με δυνατότητες CUDA και να υπολογίζει την απόδοση τους σε πραγματικές συνθήκες. Κάνει διάφορες μετρήσεις όπως μέτρηση ταχύτητας μνήμης, πολλαπλασιασμό πινάκων και Ελεγχο ορατότητας, και παρέχει στον χρήση έναν εύκολο τρόπο να διαβάσει αυτά τα αποτελέσμα και να μπορεί να τα αποθυκευσει σαν ιστορικό για μελοντική σύγκριση

Δοκιμάσαμε το πρόγραμμα μας σε διαφορετικές κάρτες και διαφορετικά συστήματα ώστε να επιβεβαιώσουμε πως τα αποτελέσματα μας αντοιστοιχούν στα χαρακτηριστικά της εκάστοτε κάρτας. Εγινε αμέσως προφανές ότι για να βγάζουν νόημα τα νούμερα, θα πρέπει να υπολογίσουμε και άλλα χαρακτηριστικά μια κάρτας γραφικών πέραν την συχνότητας των επεξεργαστών της και τον αριθμό τους. Τετοια χαρακτηριστικά είναι η ταχύτητα της μνήμης, η ταχύτητα της διεπαφής της με τον υπολογιστή (διαυλος PCI-Express) και το ευρος του Δίαυλου της μνήμης (Memory Bus Width)

Συνοψίζοντας, θεωρούμε ότι μπορέσαμε να προσφέρουμε μια ολοκληρώμενη λύση όσον αφορά στην μέτρηση επιδόσεων μιας κάρτας σε περιβάλλον CUDA, αλλά το ποιο σημαντικό μάθημε που πήραμε, είναι ότι πια είναι εμφανές πως το μέλον των επεξεργαστών έχει φύγει από την γραμμική επεξεργασία και οδεύει προς την Μαζικά Παράλληλη

Πίνακας Περιεχομένων

Παραρτήματα

Πίνακας Εικόνων

Λίστα Πινάκων

1 Εισαγωγή

Νόμος τουMoore

"The number of transistors on an integrated
Circuit doubles every two years."
– GordonE. Moore

1.1 Περίληψη

Έχουμε φτάσει σε σημείο όπου η συχνότητα δεν μπορεί να αυξηθεί .
Μόνη λύση είναι η χρησιμοποίηση περισσότερων επεξεργαστών ταυτόχρονα, (συνεπώς περισσότερα τρανζίστορ) αντί για πιο γρήγορους επεξεργαστές
Αυτό οδηγεί στην ανάγκη αλλαγής τρόπου προγραμματισμού
Τα προγράμματα πρέπει να γράφονται με γνώμονα την παραλληλία όπου αυτό είναι δυνατό, και τα δεδομένα που επεξεργάζονται θα πρέπει να είναι όσο το δυνατό πιο ανεξάρτητα μεταξύ τους.

Η έννοια της μαζικής παράλληλης επεξεργασίας δεν είναι καινούρια.
Μία παράγωγό της είναι το GPGPU. Ουσιαστικά πρόκειται για API's που μας επιτρέπουν να χρησιμοποιούμε την κάρτα γραφικών για εκτέλεση υπερπαράληλων εφαρμογών.
Αυτή την στιγμή η πιο προσιτή και εύχρηστη λύση GPGPU είναι η τεχνολογία CUDA (**Compute Unified Device Architecture)** της εταιρίας Nvidia, και υποστηρίζεται από όλες της κάρτες γραφικών της εταιρίας από την σειρά GeForce 8 και έπειτα.

Απόρροια της παράλληλης αυτής επεξεργασίας είναι και η ανάγκη για μέτρηση των επιδόσεων των καρτών αυτών .
Παρόλο που ήδη έχουν κυκλοφορήσει αρκετές εφαρμογές που να χρησιμοποιούν την αρχιτεκτονική cuda , δυστυχώς δεν υπάρχουν αξιόπιστα μετροπρογράμματα , που να καλύπτουν μεγάλο εύρος υποστηριζόμενων καρτών και εφαρμογών, που να αντιστοιχούν σε πραγματικές επιδόσεις της κάρτας.

Γι' αυτό τον λόγο δημιουργήσαμε ένα μετροπρόγραμμα, που χρησιμοποιεί της πιο συχνά χρησιμοποιούμενες ρουτίνες που μπορεί να βρει κάποιος σε προγράμματα που χρησιμοποιούν την αρχιτεκτονική cuda.

Γραμμένο σε δύο σκέλη, μία εφαρμογή κονσόλας γραμμένη σε C με επεκτάσεις CUDA και ένα γραφικό περιβάλλον γραμμένο σε VisualBasic .NET , το πρόγραμμα αυτό προσπαθεί να εξομοιώσει υπολογισμούς που θα έκανε μία κάρτα γραφικών σε πραγματικές συνθήκες.

Προσπαθήσαμε να φτιάξουμε ένα πρόγραμμα που να λαμβάνει υπ' όψιν διαφορικές παραμέτρους των επιδόσεων της κάρτας γραφικών, όπως η ταχύτητα επεξεργασίας, ο αριθμός των πυρήνων και το εύρος ζώνης της μνήμης αλλά και τις διασύνδεσης του υπολογιστή με την κάρτα γραφικών.

1.2 Σκοπός και στόχοι Εργασίας

Σκοπός μας είναι να προσφέρουμε ένα εργαλείο στον χρήστη που να του δίνει μια απλοϊκή και ταυτόχρονα πραγματική απεικόνιση των δυνατοτήτων της εκάστοτε κάρτας γραφικών σε πραγματικές συνθήκες, ώστε να μπορεί να συγκρίνει διαφορικές κάρτες γραφικών μεταξύ τους, διαφορετικές ρυθμίσεις στην ίδια κάρτα, όπως για παράδειγμα πόση απόδοση μπορεί να κερδίσει από υπερχρονισμό μιας κάρτας γραφικών CUDA, ακόμα και να υπολογίσει το κέρδος που θα έχει, αν το πρόγραμμα του θα τρέξει σε περιβάλλονCUDA.

1.3Δομή της Εργασίας

Αυτή η παρουσίαση χωρίζεται σε 6 κεφάλαια. Στο πρώτο κεφάλαιο αναφερόμαστε γενικά στο θέμα της εργασίας, το πρόβλημα που προσπαθούμε να λύσουμε και έχουμε και μια γενική εισαγωγή στην εργασία

Το δέυτερο κεφάλαιο αναφαίρετε στην έννοια του GPGPU και την σπουδαιότητα του για την ανάπτυξη της πληροφορικής.

Στο τρίτο κεφάλαιο αναφερόμαστε πιο συγκεκριμένα στην τεχνολογία CUDA, στην ιστορία της, σε βασικές της έννοιες και στα προβλήματα τα οποία μπορεί να λύσει.

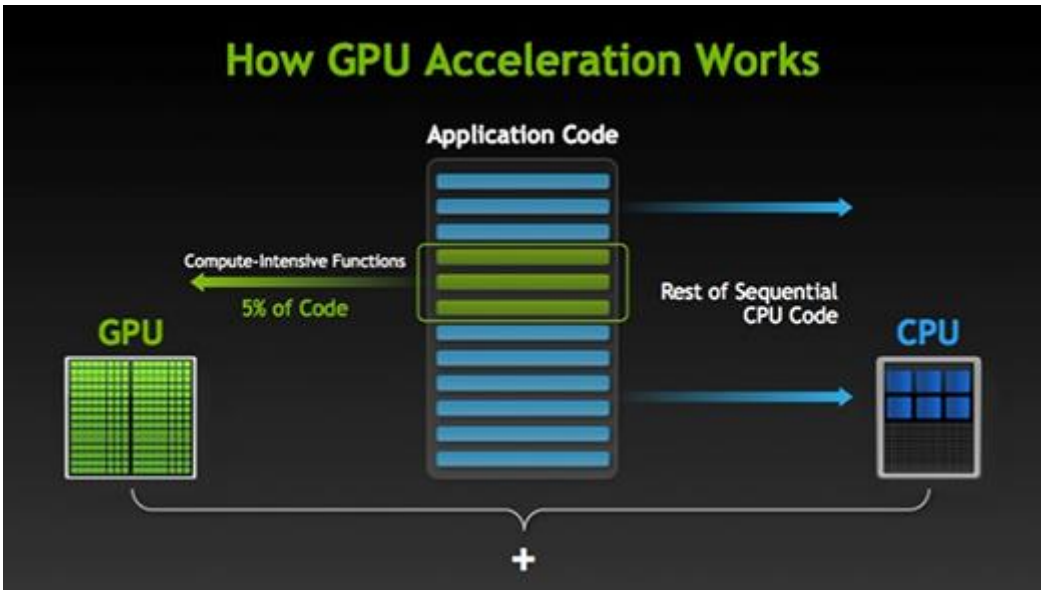
Το τέταρτο κεφάλαιο περιγράφει με λεπτομέρεια την εφαρμογή μας ενώ στο έκτο κεφάλαιο περιγράφουμε τα εργαλεία που χρησημοποιήσαμε κατά την εκπόνηση αυτής της εργασίας

Το έκτο κεφάλαιο αποτελεί μια σύνοψη τον αποτελεσμάτων και τον εμπειριών μας από την εργασία αυτή.

2 GPGPU

General Purpose Computing on Graphics Processing Units

Η GPGPU (Υπολογισμοί Γενικού Σκοπού σε Επεξεργαστές Γραφικών) είναι μια τεχνική παράλληλης επεξεργασίας που επιτρέπει την χρήση μιας GPU (Μονάδα Επεξεργασίας Γραφικών) , η οποία συνήθως χειρίζεται υπολογισμούς γραφικών, για τον υπολογισμό εφαρμογών που συνήθως διαχειρίζονται από τον επεξεργαστή. Έτσι μας δίνεται η δυνατότητα χρήσης ενός συστήματος με ιδιαίτερα μεγάλες δυνατότητες παράλληλης επεξεργασίας, που υπάρχει ήδη στους περισσότερους υπολογιστές , παράγεται μαζικά και σε χαμηλό κόστος σε σχέση με άλλες λύσεις.



Εικόνα 1 : Εκτέλεση κώδικα σε ΕΓ

2.1 Προγραμματιστικές Έννοιες GPGPU

Οι Επεξεργαστές Γραφικών είναι σχεδιασμένοι ειδικά για να επεξεργάζονται γραφικά, και συνεπώς είναι ιδιαιτέρως περιοριστικές όσο αναφορά χειρισμούς και προγραμματισμό. Λόγο της φύσης τους, οι ΕΓ είναι αποδοτικοί μόνο στην αντιμετώπιση προβλημάτων που λύνονται με την χρήση «Επεξεργασίας Ροής» (stream processing).

2.1.1 Επεξεργασία Ροής(stream processing)

Οι ΕΓ από κατασκευής είναι σχεδιασμένες να υπολογίζουν μόνο δεδομένα διανυσματικής μορφής η κλάσματα (πράξεις κινητής υποδιαστολής)

Κατ' αυτή την έννοια οι ΕΓ θεωρούνται επεξεργαστές ροής, δηλαδή επεξεργαστές που λειτουργούν παράλληλα, τρέχοντας ταυτόχρονα έναν κοινό πυρήνα(πρόγραμμα) σε πολλαπλές ¨ροές¨.

2.1.2 Ροή θεωρείται μια ομάδα καταχωρήσεων που απαιτούν πανομοιότυπους υπολογισμούς, και παρέχουν υψηλό παραλληλισμό δεδομένων.

2.1.3 Οι πυρήνες είναι οι λειτουργίες που εφαρμόζονται σε κάθε στοιχείο στην ροή.

Επειδή τα δεδομένα σε μία ροή , επεξεργάζονται ανεξάρτητα το ένα με το άλλο, είναι αδύνατο να έχουν ταυτόχρονη πρόσβαση σε στατικά η κοινόχρηστα δεδομένα.

2.1.4 Αριθμητική Ένταση.

Αριθμητική ένταση είναι ο αριθμός των λειτουργιών που τελούνται ανά μεταφερόμενη λέξη μνήμης.

Είναι σημαντικό για τις εφαρμογές GPGPU να έχουν υψηλή αριθμητική ένταση, γιατί αλλιώς οι καθυστερήσεις από την πρόσβαση μνήμης, καθιστούν το κέρδος από την παράλληλη επεξεργασία μηδαμινό.

Ιδεατά, οι εφαρμογές GPGPU χαρακτηρίζονται από μεγάλα σετ δεδομένων, υψηλό παραλληλισμό και μηδαμινή εξάρτηση μεταξύ των στοιχείων δεδομένων.

3 CUDA (Compute Unified Device Architecture)

Το μεγαλύτερο μειονέκτημα των GPGPU αρχιτεκτονικών, είναι ότι τα δεδομένα που πρόκειται να επεξεργαστούν, θα πρέπει να μοιάζουν με τα δεδομένα που συνήθως επεξεργάζεται ένας επεξεργαστής γραφικών, και οι επεξεργασίες που γίνονται πάνω σε αυτά τα δεδομένα , θα πρέπει να ακολουθούν την δομή των εργασιών που κάνει συνήθως ένας επεξεργαστής γραφικών.

3.1 CUDA Basics

Σε αυτό έρχεται να δώσει λύση η αρχιτεκτονική CUDA.

Τον Νοέμβριο του 2006 η NVidia ανακοίνωσε την CUDA, μία πλατφόρμα παράλληλης επεξεργασίας που χρησιμοποιεί την μηχανή παράλληλης επεξεργασίας των καρτών γραφικών NVIDIA , για να λύσει διάφορα υπολογιστικά προβλήματα με έναν πιο αποδοτικό τρόπο απ’ ότι ένας επεξεργαστής

Η πλατφόρμα της CUDA παρέχει ένα περιβάλλον προγραμματισμού που επιτρέπει στους προγραμματιστές να χρησιμοποιούν την C σαν γλώσσα υψηλού επιπέδου.

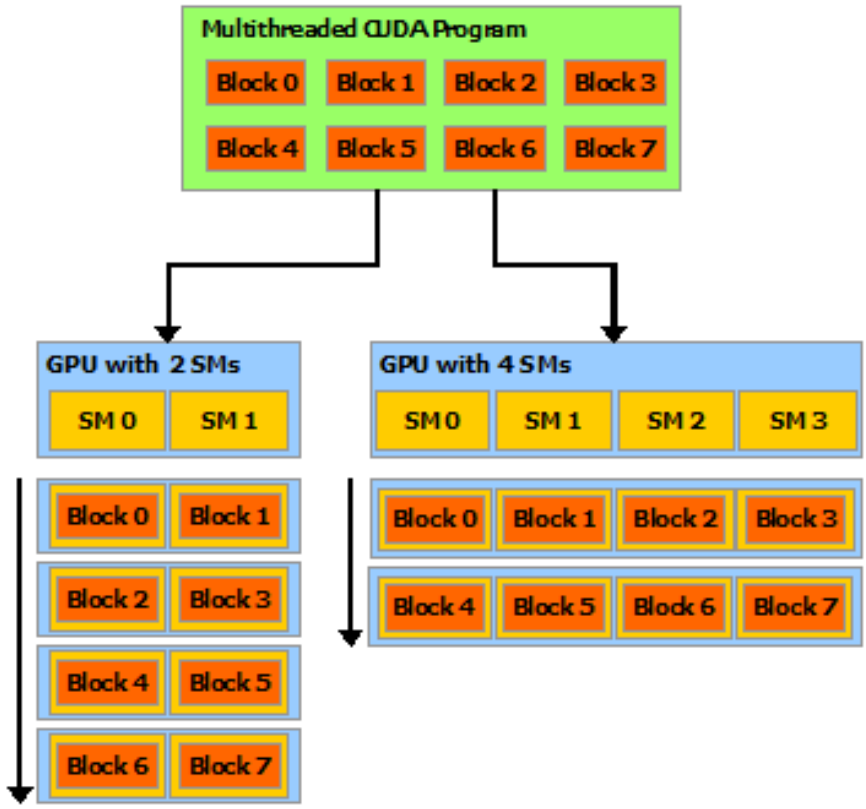
Η άφιξη των πολυπύρηνων επεξεργαστών και καρτών γραφικών συνεπάγεται ότι οι τα ολοκληρωμένα που βρίσκονται πια σε μαζική παραγωγή, αποτελούν παράλληλα συστήματα, που ταυτόχρονα η επεξεργαστική ισχύς τους μέσω της παραλληλίας αυξάνεται σύμφωνα με το νόμο του Moore. Η πρόκληση έγκειται στην ανάπτυξη εφαρμογών που να μπορούν να χρησιμοποιήσουν αυτή την παραλληλία και με διαφανή τρόπο να αλλάζουν το επίπεδο παραλληλίας τους ώστε να χρησιμοποιούν στο έπακρο τον αριθμό των αυξανόμενων επεξεργαστικών πυρήνων.

Το μοντέλο παράλληλης επεξεργασίας της CUDA είναι σχεδιασμένο να ξεπερνάει αυτό το εμπόδιο, ενώ ταυτόχρονα να είναι εύκολο στην εκμάθηση από προγραμματιστές που έχουν ήδη γνώσεις πάνω σε γλώσσες προγραμματισμού όπως η C.

Στον πυρήνα του έχει 3 βασικές έννοιες. Μία ιεραρχία από ομάδες προγραμματιστικών νημάτων , διαμοιραζόμενες μνήμες, και συγχρονισμό ορίων, που εμφανίζονται στον προγραμματιστή ως ένα ελάχιστο σετ από επεκτάσεις στην γλώσσα προγραμματισμού

Αυτές οι έννοιες παρέχουν έναν υψηλής ακρίβειας παραλληλισμό δεδομένων και νημάτων, εμφωλευμένα σε χαμηλότερης ακρίβειας δεδομένα και εργασίες παραλληλισμού. Οδηγούν τον προγραμματιστή να διαχωρίσει το πρόβλημα σε μικρότερα υποπροβλήματα που λύνονται ανεξάρτητα και σε παραλληλία, από κομμάτια (blocks) νημάτων, και κάθε υποπρόβλημα σε ακόμα μικρότερα τμήματα που μπορούν με την σειρά τους να λυθούν

Αυτός ο διαμερισμός εργασιών, επιτρέπει στην γλώσσα να διατηρεί την μορφή της επιτρέποντας στα νήματα να συνεργάζονται όταν λύνουν κάθε υποπρόβλημα, και την ίδια στιγμή να επιτρέπει αυτόματη διακλιμάκωση των εργασιών. Κάθε ομάδα (block) από νήματα μπορεί να τρέξει σε οποιονδήποτε από τους ελεύθερους πολύ-επεξεργαστές σε έναν ΕΓ , σε οποιαδήποτε σειρά, παράλληλα η σειριακά, ώστε κάθε πρόγραμμα που έχει γραφεί σε CUDA να μπορεί να τρέξει ανεξάρτητα κάθε φορά από τον διαθέσιμο αριθμό πολυεπεξεργαστών , και μόνο το σύστημα κατά την εκτέλεση να πρέπει να ξέρει τον φυσικό αριθμό πολυεπεξεργαστών σε κάθε ΕΓ.



Εικόνα 2: Αυτόματη Διακλυμάκωση

Σημείωση: Κάθε ΕΓ είναι χτισμένος γύρω από μία συστοιχία Επεξεργαστών Ροής (ΕΡ). Ένα πολυνημάτικο πρόγραμμα διαχωρίζεται σε μπλοκ(blocks) από νήμα που εκτελούνται ανεξάρτητα μεταξύ τους ώστε ένα ΕΓ με περισσότερους πολυεπεξεργαστές να μπορεί αυτόματα να εκτελεί το πρόγραμμα σε λιγότερο χρόνο απ' ότι ένας ΕΓ με λιγότερους πολυεπεξεργαστές

3.2 Προγραμματισμός σε CUDA.

Ο προγραμματισμός στην Cuda γίνεται μέσω του παρεχόμενου API και SDK που παρέχει η Nvidia και επιτρέπει σε προγραμματιστές, μέσω της γλώσσας C να δημιουργούν κώδικα ο οποίος έχει την δυνατότητα , κομμάτια του να εκτελούνται στον ΕΓ.

Αυτό γίνεται με την χρήση των πυρήνων CUDA (Cuda Kernels) , δηλαδή κομματιών κώδικα τα οποία γίνονται Compile από τον compiler της Cuda (nvcc) και όχι από κλασικό Compiler C , και ο κώδικας αυτός τρέχει στον ΕΓ και όχι στον κανονικό επεξεργαστή. Το γεγονός ότι ένα κομμάτι κώδικα εκτελείται από τον κανονικό επεξεργαστή και ένα άλλο κομμάτι του ίδιου προγράμματος τρέχει σε άλλη συσκευή, στην συγκεκριμένη περίπτωση σε έναν ΕΓ, λέγεται ετερογενής προγραμματισμός. Λεπτομέρειες για τον ετερογενή προγραμματισμό θα δούμε παρακάτω.

Ένα παράδειγμα Cuda Kernel είναι το εξής:

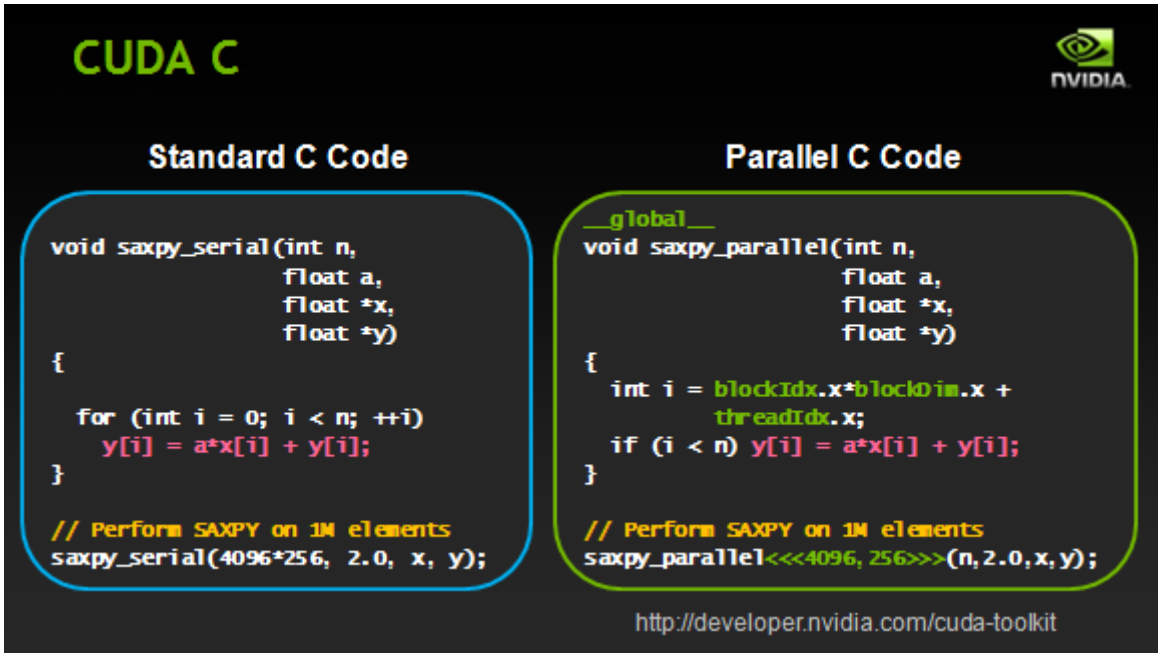
```
__global__void mat_mul2(float *a, float *b, float *ab){
int idx = blockIdx.x * blockDim.x + threadIdx.x;

ab[idx]=a[idx]*b[idx];
}
```

Αυτό το κομμάτι κώδικα θα τρέξει εν παραλλήλο σε όλα τα threads ταυτόχρονα, και κάθε thread θα χαρακτηρίζεται από τα ξεχωριστά γι' αυτό blockIdx.xblockDim.x και threadIdx.x

Όσο περισσότερα threads μπορούν να τρέξουν ταυτόχρονα, τόσο πιο αποδοτικό θα είναι το πρόγραμμά μας (θα έχει υψηλή παραλληλία και συνεπώς υψηλή Αριθμητική Ένταση)

3.3 Προγραμματιστικό Μοντέλο

HCUDAακολουθεί στενά το προγραμματιστικό μοντέλο των GPGPUεφαρμογών, παρέχοντας όμως μεγαλύτερη ευελιξία κατά τον προγραμματισμό. Σαν αποτέλεσμα, έχει αρκετές κοινές έννοιες με το GPGPUμοντέλο, τις οποίες θα δούμε παρακάτω.



Εικόνα 3: Σύγκριση κανονικής C με CUDA C

3.3.1 Πυρήνες CUDA(kernels)

Η CUDAC επεκτείνει την C επιτρέποντας στον προγραμματιστή να χρησιμοποιεί ρουτίνες C , ονομαζόμενες Πυρήνες (kernels) , που όταν καλούνται, εκτελούνται N φορές παράλληλα από N διαφορετικά νήματα CUDA, εν αντίθεση με το να εκτελούνται μόνο μία φορά όπως θα συνέβαινε στην κανονική C

Ένας πυρήνας ορίζεται χρησιμοποιώντας την δήλωση __global__ και ο αριθμός των νημάτων στα οποία θα εκτελεστεί αυτός ο πυρήνας καθορίζεται χρησιμοποιώντας ένα την σύνταξη <<<…>>> Σε κάθε νήμα που εκτελεί τον πυρήνα, δίνεται ένα μοναδικό ID νήματος που είναι διαθέσιμο στον πυρήνα μέσω της μεταβλητής ThreadIDx

Ένα παράδειγμα πυρήνα είναι το εξής:

```
// Kernel definition

__global__ void VecAdd(float* A, float* B, float* C)

{

int i = threadIdx.x;

   C[i] = A[i] + B[i];

}
```

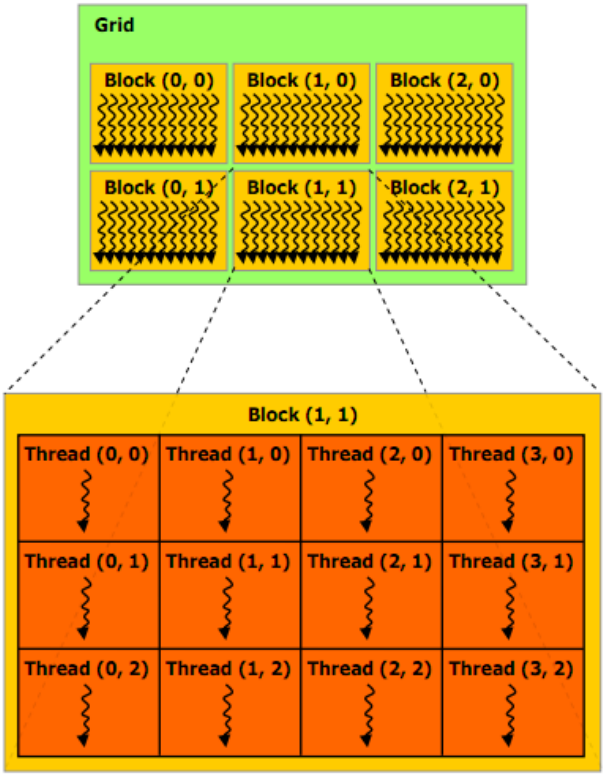
Και καλείται μέσω από το πρόγραμμά μας έτσι:

```
Vecadd<<< n_blocks, block_size >>> (A,B,C );
```
Που εκτελείται παράλληλα σε n_blocks*block_sizethreads

3.3.2 ΙεραρχίαThreads

Τα Threads ομαδοποιούνται σε ισομεγέθη Blocks που περιέχουν ομογενοποιημένα threads (δηλαδή εκτελούν τον ίδιο kernel). Τα blocks με την σειρά τους ομαδοποιούνται σε Grids. Το κάθε thread μπορεί να είναι μέχρι 3 διαστάσεων, δηλαδή να μπορεί να χαρακτηριστεί από 1, 2 η 3 δείκτες (threadId x , y , z) το οποίο με την σειρά του ομαδοποιείται σε Blocks με 1 , 2 η και 3 διαστάσεις (blockIdx, y, z). Τα Grids μπορεί να είναι μέχρι 2 διαστάσεων.

Πχ ένα 2διάστατο thread σε ένα 2διάστατο block που υπάρχει σε ένα μονοδιάστατο grid μπορεί να χαρακτηριστεί από τα threadId.x , threadId.y , blockId.x , blockId.y,



Εικόνα 4: Ιεραρχία Threads

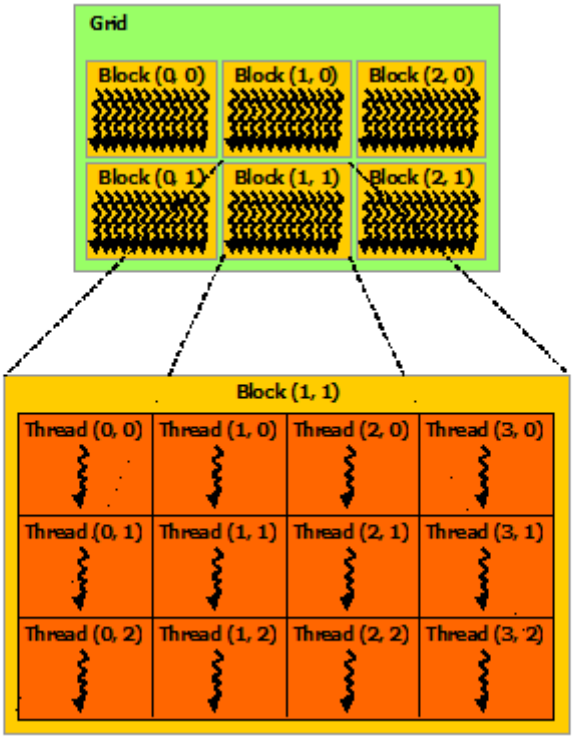
Για χάριν ευκολίας, η threadIDx είναι ένα διάνυσμα 3 συνιστωσών, ώστε τα νήματα να προσδιορίζονται χρησιμοποιώντας μονοδιάστατους, δυσδιάστατους ή τρισδιάστατους δείκτες σχηματίζοντας ένα μονοδιάστατο, δυσδιάστατο η τρισδιάστατο μπλοκ νημάτων. Αυτό προσφέρει έναν φυσικό τρόπο να καλούνται υπολογισμοί σε κοινά προγραμματιστικά στοιχεία όπως διανύσματα, πίνακες, πλέγματα η όγκους (volume).

Ο δείκτης ενός νήματος και το ThreadID συσχετίζονται μεταξύ τους με έναν ξεκάθαρο τρόπο: Για μονοδιάστατα μπλοκ είναι παρόμοια, για δυσδιάστατα μπλοκ μεγέθους (A,B) το threadID ενός νήματος με δείκτη (χ,y) είναι (x+ yA) , και για ένα τρισδιάστατο μπλοκ μεγέθους (A,B,Γ), το threadID ενός νήματος με δείκτη (x,y,z) είναι (X+ yA+zAB)

Υπάρχει όριο στον αριθμό νημάτων ανά μπλοκ, μιας και όλα τα νήματα ενός μπλοκ πρέπει να βρίσκονται στον ίδιο πυρήνα επεξεργαστή και να μοιράζονται την περιορισμένη σε πόρους μνήμη αυτού του πυρήνα. Στην τρέχουσα γενιά ΕΓ ένα μπλοκ από νήματα μπορεί να περιέχει μέχρι και 1024 νήματα

Παρ όλα αυτά , ένας πυρήνας μπορεί να εκτελείται από πολλαπλά ισομεγέθη μπλοκ , ώστε ο συνολικός αριθμός των νημάτων να ισούται με τον αριθμό των νημάτων, επί τον αριθμό των μπλοκ.

Τα μπλοκ μπορούν να οργανώνονται σε μονοδιάστατα, δισδιάστατα και τρισδιάστατα πλέγματα (GRID) από μπλοκ αριθμός των μπλοκ ανά πλέγμα , συνήθως υπαγορεύεται από το μέγεθος των δεδομένων που επεξεργάζονται , ή από τον αριθμό τον επεξεργαστών στο σύστημα, τους οποίους μπορεί να υπερβαίνει κατά πολύ.



Εικόνα 5: Πλέγμα νηματικών μπλοκ

Ο αριθμός των νημάτων ανά μπλοκ , και ο αριθμός τον μπλοκ ανά πλέγμα που ορίζονται στην σύνταξη <<<…>>> μπορεί να είναι είτε σε μορφή ακεραίας μεταβλητής (int) είτε μορφής dim3.

Κάθε μπλοκ μέσα στο πλέγμα μπορεί να ταυτοποιηθεί μέσω ενός μονοδιάστατου, δισδιάστατου η τρισδιάστατου πίνακα (index) προσβάσιμου μέσα απ' τον πυρήνα διαμέσου της μεταβλητής blockIdX. Η διάσταση του νηματικού μπλοκ είναι προσβάσιμη απ' τον πυρήνα μέσω της εγγενής μεταβλητής blockDim

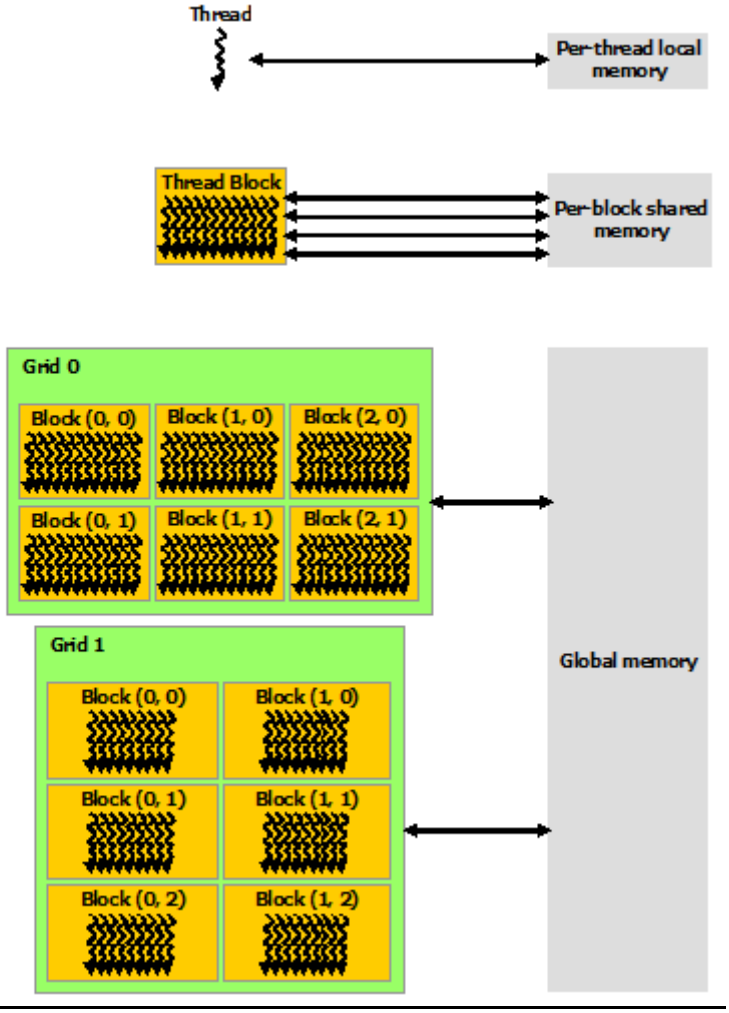
Τα νηματικά μπλοκ είναι υποχρεωμένα να εκτελούνται ανεξάρτητα. Πρέπει να είναι δυνατό να εκτελούνται σε οποιαδήποτε σειρά, παράλληλα η σειριακά. Αυτή η αναγκαιότητα επιτρέπει τα νηματικά μπλοκ να μπορούν να εκτελούνται σε οποιοδήποτε αριθμό επεξεργαστικών πυρήνων, επιτρέποντας στον προγραμματιστή να γράφει κώδικα που να μπορεί να προσαρμοστεί στον αριθμό των διαθέσιμων επεξεργαστικών πυρήνων

Τα νήματα που περιέχονται σε ένα μπλοκ , μπορούν να συνεργαστούν μεταξύ τους μέσω μιας διαμοιρασμένης μνήμης (shared memory), και συγχρονίζοντας την εκτέλεση τους ώστε να συντονίζεται η προσπέλαση μνήμης.

3.3.3 Ιεραρχία μνήμης

Τα νήματα CUDA μπορούν να έχουν πρόσβαση σε δεδομένα από διαφορετικές μνήμες κατά την εκτέλεση τους. Κάθε νήμα έχει την ιδιωτική τοπική μνήμη (private local memory). Κάθε νηματικό μπλοκ έχει την δικιά του κοινόχρηστη μνήμη (shared memory) , που είναι ορατή σε όλα τα νήματα του μπλοκ και έχει την ίδια διάρκεια ζωής με το μπλοκ. Όλες τα νήματα έχουν επίσης πρόσβαση στην οικουμενική μνήμη (global memory)

Μελλοντικές εκδόσεις CUDAενδέχεται να επιτρέπουν την άμεση προσπέλαση της Κύριας μνήμης του υπολογιστή από τους πυρήνες, επιταχύνοντας κατά πολύ τον χρόνο εκτέλεσης , κυρίος σε προγράμματα με μικρή Αριθμητική Ένταση

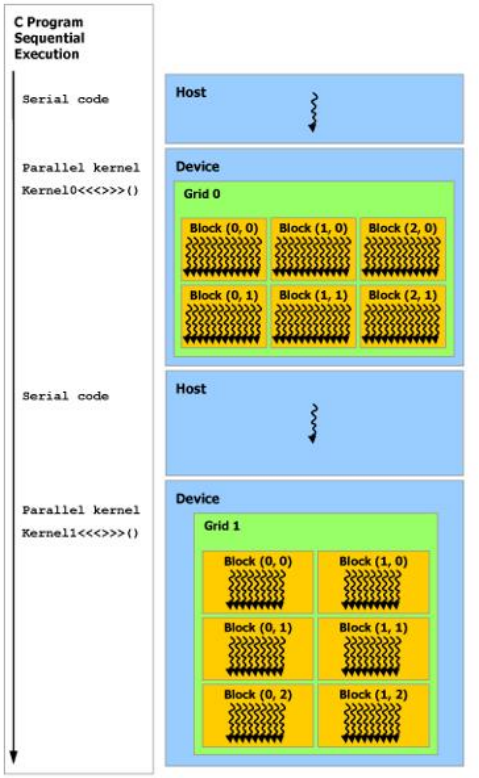


Εικόνα 6: Ιεραρχία μνήμης

3.4 Ετερογενής Προγραμματισμός

Το προγραμματιστικό μοντέλο της CUDA θεωρεί ότι τα νήματα CUDA εκτελούνται σε μια φυσικά ξεχωριστή συσκευή (device) που λειτουργεί σαν συνεπεξεργαστής στον υπολογιστή (Host) που τρέχει το πρόγραμμα C. Δηλαδή τα νήματα CUDA εκτελούνται στον ΕΓ και το υπόλοιπο πρόγραμμα C εκτελείται στον Επεξεργαστή.

Το προγραμματιστικό μοντέλο επίσης θεωρεί ότι και ο Host και η συσκευή Cuda, έχουν την δικιά τους ξεχωριστή μνήμη, που αναφέρονται σαν μνήμη HOST και μνήμη DEVICE. Συνεπώς το πρόγραμμα διαχειρίζεται τις Global, constant και texture μνήμες που είναι ορατές στους πυρήνες CUDA. Αυτή η διαχείριση περιλαμβάνει ανάθεση και αποανάθεση μνήμης, όπως και μεταφορά δεδομένων από την HOST στην DEVICE μνήμη και αντίστροφα.



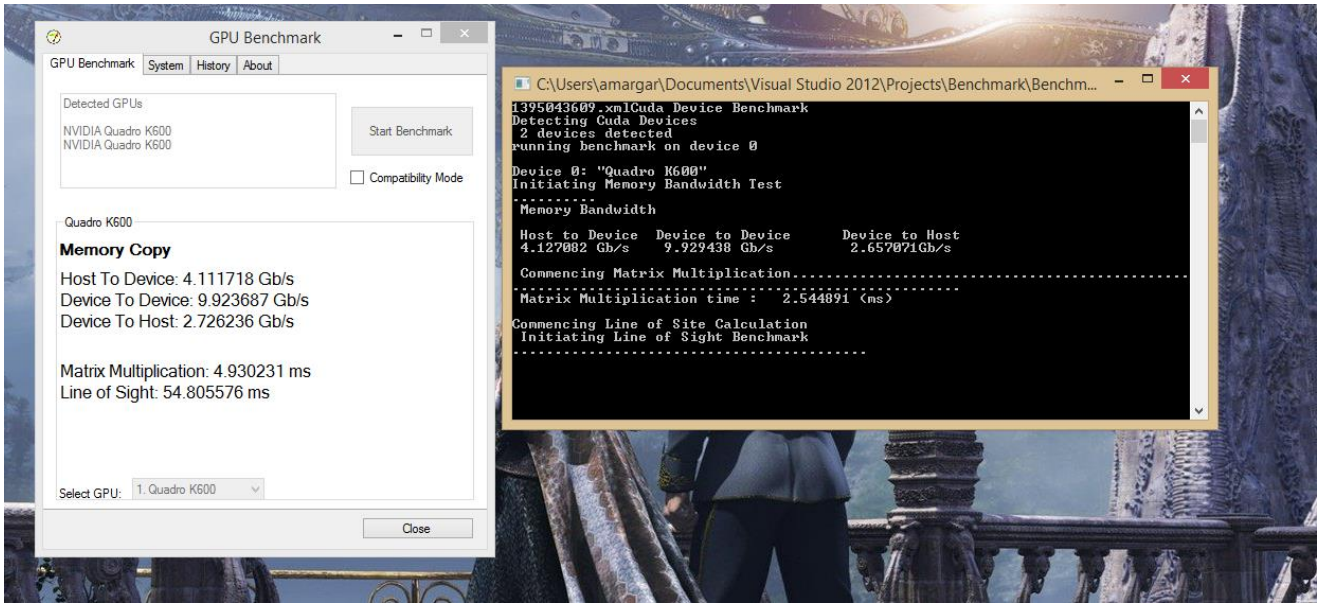
Εικόνα 7: Ετερογενής Προγραμματισμός

Σημείωση: Ο σειριακός κώδικας εκτελείται στον HOST ενώ ο παράλληλος κώδικας εκτελείτε στον ΕΓ

4 Μετροπρόγραμμα για χρήση σε περιβάλλον CUDA

Η ύπαρξη της αρχιτεκτονικήςCUDA είναι συνυφασμένη με την ανάγκη για μεγαλύτερη επεξεργαστική ισχύ, και για γρηγορότερους υπολογισμούς. Συνεπώς δημιουργείται και η ανάγκη για ένα πρόγραμμα που να μπορεί να μετράει την επίδοση αυτή των καρτών γραφικών. Ένα τέτοιο μετροπρόγραμμα (benchmark) θα πρέπει να δύναται να τρέχει κατά προτίμηση σε όλες τις κάρτες που υποστηρίζουν τις διάφορες εκδόσεις της CUDA και να μπορεί να δίνει αποτελέσματα που να αντιστοιχούν με τις πραγματικές επιδόσεις της κάρτας, σε εφαρμογές καθημερινής χρήσης.

Για να καλυφθεί το κενό που περιγράψαμε πιο πάνω, δημιουργήσαμε ένα μετροπρόγραμμα βασισμένο σε CUDA που εκτελεί μερικές από τις πιο συνηθισμένες ρουτίνες που μπορεί να υπάρχουν σε ένα πρόγραμμα CUDA. Το πρόγραμμα αυτό χωρίζεται σε 2 διακριτά κομμάτια. Μία console εφαρμογή γραμμένη σε C που περιέχει τις απαραίτητες βιβλιοθήκες για CUDA , και τρέχει τους υπολογισμούς στην κάρτα γραφικών, και ένα γραφικό περιβάλλον που κάνει την εκτέλεση της console εφαρμογής πιο εύχρηστη, ενώ προσφέρει και παραπάνω λειτουργίες όπως ιστορικό των μετρήσεων και πληροφορίες για τον υπολογιστή.

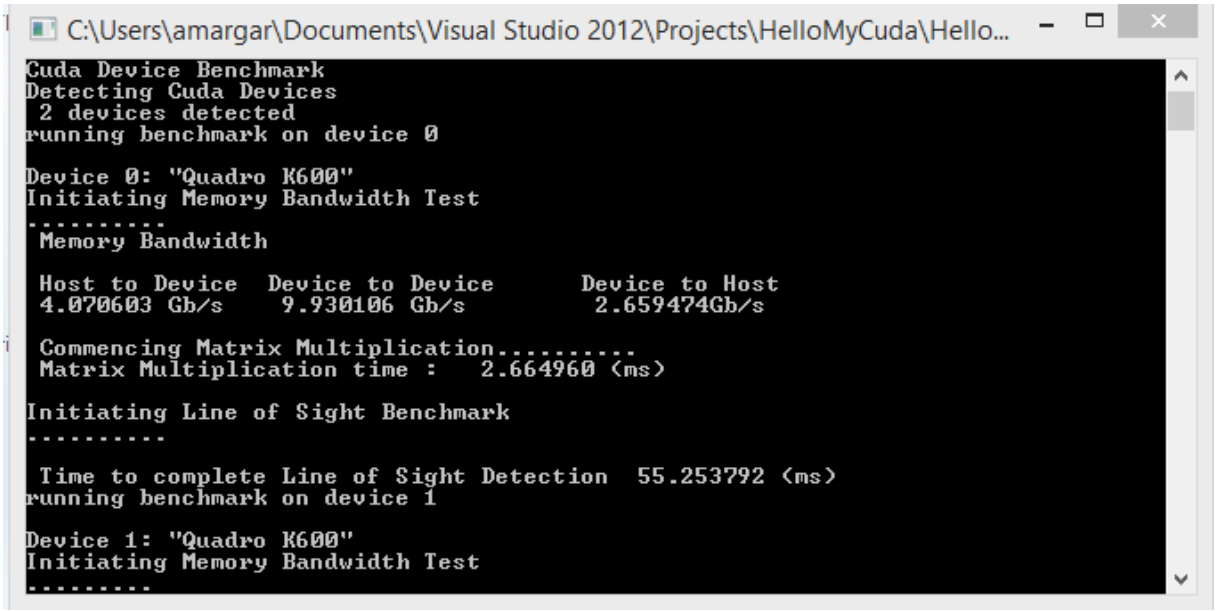


Εικόνα 8: Η εφαρμογή κατά την εκτέλεση

4.1 Η εφαρμογή κονσόλας

Πρόκειται για μία console εφαρμογή, γραμμένη σε C και χρησιμοποιεί τις βιβλιοθήκες CUDA που παρέχονται από την NVidia στο SDK της. Τρέχει τρείς διαφορετικές μετρήσεις. Μέτρηση Bandwidth μνήμης, έναν τυπικό πολλαπλασιασμό πλεγμάτων (πίνακα δεδομένων 2 διαστάσεων) και μία ρουτίνα υπολογισμού οπτικής επαφής (line of sight) σε επίπεδο 2 διαστάσεων.

Μπορεί και ανιχνεύει από μόνο του της εγκατεστημένες κάρτες γραφικών με δυνατότητες CUDA και τρέχει τις μετρήσεις σε κάθε κάρτα ξεχωριστά. Τα αποτελέσματα του κάθε benchmark αποθηκεύονται σε ένα xml αρχείο, το όνομα του οποίου μπορεί να περαστεί σαν όρισμα κατά την εκτέλεση του.

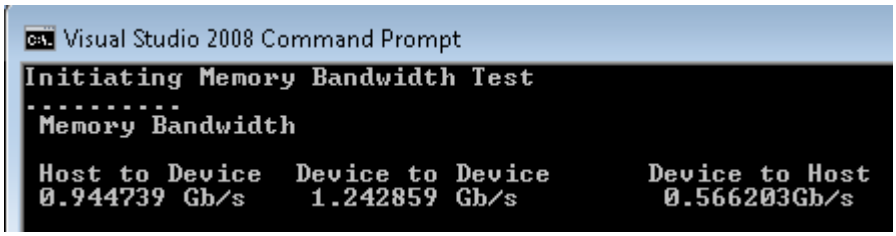


Εικόνα 9: Η εφαρμογή κονσόλας κατά την εκτέλεση

4.1.1 Έλεγχος ταχύτητας μνήμης.

Συγκεκριμένα εκτελούνται και μετριούνται 3 μεταφορές μνήμης.

- Από τον μνήμη του υπολογιστή (HOST) στην global μνήμη του ΕΓ (device) – Host to device
- Από ένα σημείο της μνήμης του ΕΓ σε ένα άλλο – Device to Device
- Από την μνήμη του ΕΓ πίσω στην μνήμη του υπολογιστή – Device to Host



Εικόνα 10: Η μέτρηση ταχύτητας μνήμης

Από προγραμματιστικής πλευράς, για τον έλεγχο αυτό έχουμε 1 ρουτίνα την memcopy() που επιστρέφει μία δομή τύπου Bwidth που περιέχει τα αποτελέσματα. Δομή Bwidth αποτελείται από τρείς μεταβλητέςfloat.

```
typedefstruct {
floatd2h;
float d2d;
floath2d;
  } BWidth;
```

Κατ' άρχήν αρχικοποιούνται στον HOST τα δεδομένα που θα αντιγραφούν στην κάρτα γραφικών, σε έναν πίνακα float με μέγεθος tablesize, και γεμίζει αυτός ο πίνακας με στατικά δεδομένα.

```
reiterations = COPYSIZE/sizeof(float);

//populating data to be copied...
//calculating data size
      tablesize=reiterations*sizeof(float);
//allocating data on HOST
      data_h=(float *)malloc(tablesize);
//initializing data on HOST
      for (i=0;i<reiterations;i++)
            {
                  data_h[i]=1.1f ;
            }
```

Αμέσως μετά δεσμεύεται ο ίδιος χώρος στην κάρτα γραφικών μέσω της εντολής cudaMalloc

```
//alocating data on device
      cudaMalloc((void **) &data_d, tablesize);
```

Έπειτα δημιουργείται και ξεκινάει ένας εσωτερικός timer και ξεκινάει η μεταφορά των αρχείων μέσω της εντολής cudaMemcpy. Μόλις ολοκληρωθεί η διαδικασία , σταματάει ο timer , παίρνουμε την χρόνο εκτέλεσης σε ms, και ανακυκλώνουμε τον timer Προτιμάμε να χρησιμοποιούμε τον εσωτερικό timer στην κάρτα γραφικών, γιατί δίνει καλύτερα αποτελέσματα και πιο ακριβή, ανεξάρτητα με την ταχύτητα του HOST.

```
      //creating and starting timer
      //creating timer and event
sdkCreateTimer(&timer1);
checkCudaErrors(cudaEventCreate(&start));
```

```
checkCudaErrors(cudaEventCreate(&stop));

      //starting timer
      sdkStartTimer(&timer1);
      checkCudaErrors(cudaEventRecord(start, 0));

      //copy data from host to device
      cudaMemcpy(data_d, data_h, tablesize, cudaMemcpyHostToDevice);
      checkCudaErrors(cudaEventRecord(stop, 0));

      //cutilSafeCall( cudaThreadSynchronize() );
      checkCudaErrors(cudaDeviceSynchronize());

//stoping timer,getting data,  printing and deleting timer
      sdkStopTimer(&timer1);
      checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));
      sdkDeleteTimer(&timer1);
```

Τέλος υπολογίζουμε την ταχύτητα μεταφοράς σε GB/s

```
bandwidth1=(COPYSIZE/1024/1024)/ (elapsedTimeInMs/1000); //(gbytes/s)
```

Έτσι έχουμε υπολογίσει την ταχύτητά μεταφοράς από τον HOST προς τον ΕΓ(Host to Device). Κάνουμε το ίδιο 2 ακόμα φορές για να υπολογίσουμε το Bandwidth εσωτερικά στον ΕΓ (Device to Device) και από τον ΕΓ στον HOST (Device to Host). Η μόνη διαφορά στον κώδικα είναι ο τρόπος που καλείται η εντολή cudaMemcpy, όπου καλείται μια φορά σαν

```
cudaMemcpy(datacopy_d, data_d, tablesize, cudaMemcpyDeviceToDevice);
```

και μία φορά σαν

```
cudaMemcpy(datacopy_d, data_d, tablesize, cudaMemcpyDeviceToHost);
```

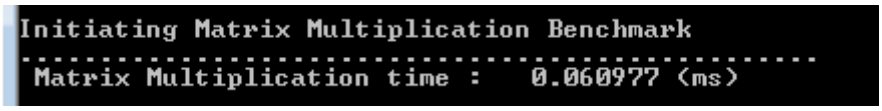
Τέλος , από τα νούμερα που έχουμε ήδη, φτιάχνουμε μία δομή Bwidth , ελευθερώνουμε την μνήμη και επιστρέφουμε την δομή.

```
      bwidth.h2d=bandwidth1;
      bwidth.d2d=bandwidth2;
      bwidth.d2h=bandwidth3;


free(data_h);
cudaFree(data_d);
free(datacopy_h);
```

```
cudaFree(datacopy_d);

return(bwidth);
```

4.1.2 Πολλαπλασιασμός Πλεγμάτων (Matrix Multiplication)

Εκτελείται πολλαπλασιασμός των στοιχείων 2 ισομεγεθών μονοδιάστατων πινάκων, και το αποτέλεσμα τους αποθηκεύεται σε 3ο πίνακα



Εικόνα 11: Η μέτρηση πολλαπλασιασμού πλεγμάτων

Συγκεκριμένα αυτό επιτυγχάνεται από την υπορουτίνα matrixmul() που επιστρέφει τον χρόνο εκτέλεσης της σε float. Η ρουτίνα αυτή καλεί τον πυρήνα CUDA mat_mul2

4.1.2.1 matrixmul()

Αμέσως μετά την αρχικοποίηση των μεταβλητών της, η matrixmul() δεσμεύει στην μνήμη χώρο για 3 πίνακες float μεγέθους matrixsize, και αμέσως μετά δεσμεύει τον ίδιο ακριβώς χώρο στην μνήμη του ΕΓ

```
//allocate host matrices
//calculate size of matrix
//printf("size of float is %d", sizeof(float));
matrixsize=matrixlen*sizeof(float);
matrixhost_a=(float *)malloc(matrixsize);
matrixhost_b=(float *)malloc(matrixsize);
matrixhost_ab=(float *)malloc(matrixsize);

//alocate device matrices

cudaMalloc((void **) &matrixdevice_a, matrixsize);
cudaMalloc((void **) &matrixdevice_b, matrixsize);
cudaMalloc((void **) &matrixdevice_ab, matrixsize);
```

Στην συνέχεια γεμίζουν οι πίνακες matrixhost_a και matrixhost_b με τυχαία δεδομένα και αντιγράφονται από τον Host στον ΕΓ

```
//initiate host matrices

for(i=0;i<matrixlen;i++){
      matrixhost_a[i]=2*i;
      matrixhost_b[i]=i/10;
      }
```

```
// copy matrices from host to device
cudaMemcpy(matrixdevice_a, matrixhost_a,matrixsize,cudaMemcpyHostToDevice);
cudaMemcpy(matrixdevice_b, matrixhost_b,matrixsize,cudaMemcpyHostToDevice);
```

Ενεργοποιείται ο timer και καλείται η mat_mul2 για threadsόσα και ο αριθμός των στοιχείων του πίνακα. Για μέγιστη συμβατότητα θέτουμε ότι κάθε μπλοκ θα έχει 4 threads, οπότε καλούμε την mat_mul για matrixlen/4(+1 αν η διαίρεση έχει υπόλοιπο)

```
sdkCreateTimer(&timer);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));


//calling the cuda kernel


      int block_size = 4;
int n_blocks = matrixlen/block_size + (matrixlen%block_size == 0 ? 0:1);
    mat_mul2<<< n_blocks, block_size >>>
(matrixdevice_a,matrixdevice_b,matrixdevice_ab );

//stop timer

      checkCudaErrors(cudaEventRecord(stop, 0));
      checkCudaErrors(cudaDeviceSynchronize());
      sdkStopTimer(&timer);
//); get time from timer
checkCudaErrors(cudaEventElapsedTime(&GPU_time, start, stop));
//destroy timer;
sdkDeleteTimer(&timer);
//  retrieve multiplied matrix from device
cudaMemcpy(matrixhost_ab,matrixdevice_ab,matrixsize,cudaMemcpyDeviceToHost);
//free used memory

cudaFree(matrixdevice_a);
cudaFree(matrixdevice_b);
cudaFree(matrixdevice_ab);

free(matrixhost_a);
free(matrixhost_b);
free(matrixhost_ab);
//return time in ms
```

```
return(GPU_time);
```

4.1.2.2 Ο πυρήνας cuda mat_mul2

Ο πυρήνας mat_mul2 είναι ουσιαστικά ένα κομμάτι κώδικα που τρέχει παράλληλα και ανεξάρτητα σε κάθε νήμα ξεχωριστά. Δέχεται σαν ορίσματα 3 δείκτες (Pointers) με θέσεις μνήμης στον ΕΓ που περιέχουν 3 ισομεγέθεις πίνακες τύπου float, και πολλαπλασιάζει κάθε θέση idx του πίνακα ab , με το αποτέλεσμα του πολλαπλασιασμού a[idx]*b[idx].

Ουσιαστικά ο αντίστοιχος κώδικας γραμμένος σε C , που θα έτρεχε σειριακά θα ήταν ο

```
For(idx=0;idx<matrixlen;idx++){ab[idx]=a[idx]*b[idx]};
```

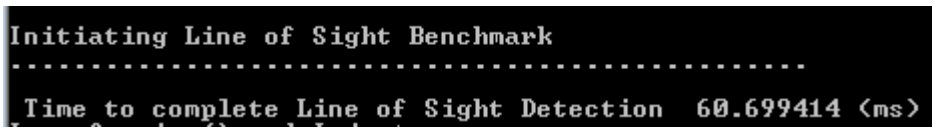
Σαν cuda πυρήνας όμως αυτό ο κώδικας γίνεται

```
__global__void mat_mul2(float *a, float *b, float *ab){
int idx = blockIdx.x * blockDim.x + threadIdx.x;

ab[idx]=a[idx]*b[idx];
}
```

ο οποίος και τρέχει παράλληλα και ανεξάρτητα για κάθε idx , όπου το idx είναι το μοναδικό id κάθε thread

4.1.3 Έλεγχος ορατότητας(Line of Sight)

Δημιουργείται απ' τον ΕΓ ένα μονοδιάστατο heightmap 200.000 στοιχείων με ημιτονική κατανομή, και ελέγχεται η ορατότητα του πρώτου με το τελευταίο στοιχείο. Επιστρέφεται πίνακας με όλα τα στοιχεία που κόβουν την ορατότητα χαρακτηρισμένα με 1 , και όλα τα στοιχεία που δεν κόβουν την ορατότητα χαρακτηρισμένα με 0



Εικόνα 12: Η μέτρηση Ορατότητας

Στην θεωρία , έχουμε ένα δισδιάστατο επίπεδο, μήκους 200.000 στοιχείων με διαφορετικό ύψος κάθε στοιχείο. Υπολογίζεται η νοητή γραμμή μεταξύ του πρώτου και του τελευταίου και όσα στοιχεία βρίσκονται κάτω από αυτή την γραμμή θεωρούμε ότι δεν κόβει την ορατότητα και όσα βρίσκονται πάνω από την γραμμή αυτή θεωρούμε ότι κόβουν την ορατότητα.

Για να γίνει αυτό θα πρέπει να υπολογίζεται το ύψος της νοητής γραμμής που ενώνει τα δυο σημεία, σε κάθε μέρος του επιπέδου. Για να το κάνουμε αυτό πρέπει να γνωρίζουμε την απόσταση του σημείου που ελέγχουμε από το αρχικό σημείο, και την κλήση της γωνίας που έχει η νοητή μας γραμμή. Την κλήση της γωνίας μας την δίνει η εφαπτόμενη της γωνίας, ενώ η απόσταση είναι η ίδια με τον αριθμό του στοιχείου του πίνακα.

Το συγκεκριμένο benchmark γίνεται με την χρήση 2 πυρήνων CUDA, τωνcreate_hmap και calc_los και της υπορουτίνας los() η οποία επιστρέφει float με τον χρόνο εκτέλεσης της σε ms.Πρόκειται για ένα benchmark που υπολογίζει και τον χρόνο μεταφοράς δεδομένων στην μνήμη, αλλά και τον χρόνο υπολογισμού των δεδομένων αυτών, και συνεπώς δίνει μια πιο σφαιρική άποψη για τις επιδόσεις. Επηρεάζεται δηλαδή και από την ταχύτητα της μνήμης ενός ΕΓ , και από τον δίαυλο επικοινωνίας HOST και DEVICE , αλλά και από τις επεξεργαστικές δυνατότητες του ΕΓ.

4.1.3.1 Los()

Η βασική ρουτίνα απ' την οποία καλούνται και οι 2 kernels create_hmap και calc_los. Ξεκινάει ενεργοποιώντας τον εσωτερικό timer του ΕΓ. Αμέσως μετά δεσμεύει χώρο στην μνήμη του HOST και της DEVICE για έναν πίνακα float μεγέθους tablesize , και τρέχει τον πυρήνα create_hmap που θα γεμίσει τον πίνακα αυτό με δεδομένα. Τέλος αντιγράφει τον πίνακα αυτόν πίσω στο HOST

```
//creating and starting timer
sdkCreateTimer(&timer);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));

tablesize=mapsize*sizeof(float);
resultsize=mapsize*sizeof(int);

//create table
//run 200.000 sin table using cuda createhmap
hmap_h=(float *)malloc(tablesize);
//alocate heightmap space on device
cudaMalloc((void **) &hmap_d,tablesize);
//call height map creation routing on device
int block_size = 4;
int n_blocks = mapsize/block_size + (mapsize%block_size == 0 ? 0:1);
create_hmap <<< n_blocks, block_size >>> (hmap_d,mapsize);
//copy result height map to host for refference
cudaMemcpy(hmap_h,hmap_d,tablesize,cudaMemcpyDeviceToHost);
checkCudaErrors(cudaDeviceSynchronize());
```

Στην συνέχεια δεσμεύεται στην κάρτα γραφικών και στον HOSTο χώρος για τον πίνακα που περιέχει το αποτέλεσμα. Αυτός ο πίνακας είναι τύπου int και μεγέθους tablesize.Έπειτα υπολογίζεται η κλήση της γωνίας της οπτικής επαφής (theta) και καλείται ο δεύτερος πυρήνας που θα υπολογίσει αν κάθε στοιχείο του πίνακα hmap είναι πάνω απ' την γραμμή οπτικής επαφής ή όχι ενώ μετά αντιγράφεται ο πίνακας των αποτελεσμάτων από την DEVICE στον HOST.

```
//allocate result map on device
cudaMalloc((void **) &dvmap,resultsize);
//allocate result map on host
vlmap=(int *)malloc(resultsize);
//calculate theta
```

```
theta=(hmap_h[0]-hmap_h[mapsize-1])/mapsize;
//call los detection routine
 block_size = 4;
 n_blocks = mapsize/block_size + (mapsize%block_size == 0 ? 0:1);
 calc_los <<< n_blocks, block_size >>>
(hmap_d,dvmap,mapsize,theta,hmap_h[mapsize-1]);
//recover result map from device
cudaMemcpy(vlmap,dvmap,resultsize,cudaMemcpyDeviceToHost);
```

Τέλος σταματάει ο timer και καταστρέφεται, ο χρόνος σώζεται σε μία μεταβλητή gpu_time , όλες οι δεσμευμένες μνήμες ελευθερώνονται και επιστρέφεται ο χρόνος εκτέλεσης σε ms

```
checkCudaErrors(cudaEventRecord(stop, 0));
checkCudaErrors(cudaDeviceSynchronize());
sdkStopTimer(&timer);
checkCudaErrors(cudaEventElapsedTime(&GPU_time, start, stop));



free(hmap_h);
cudaFree(hmap_d);
free(vlmap);
cudaFree(dvmap);


return(GPU_time);
```

4.1.3.2 Πυρήνας create_hmap

```
__global__void create_hmap(float *a,intdistance){
int idx= blockIdx.x * blockDim.x + threadIdx.x;
a[idx]=(sinf(idx)+1)*40;
}
```

4.1.3.3 Πυρήνας calc_los

```
__global__ void calc_los(float *a,int *dvmap, int Totaldistance, float
theta,float objh)
{
float ah,b;
int idx = blockIdx.x * blockDim.x + threadIdx.x;
ah=((Totaldistance-idx)*theta)+objh;
   b=a[idx];
```

```
if (ah>b){dvmap[idx]=1;}
elsedvmap[idx]=0;
 }
```

4.1.4 Main()

Η main φροντίζει ώστε να εκτελούνται τα κάθε benchmark σε κάθε ΕΓ και να φτιάχνει ένα xml με το αποτέλεσμα. Εκτελεί κάθε benchmark αρκετές φορές και βγάζει τον μέσο όρο των αποτελεσμάτων, ώστε να ελαχιστοποιείται η πιθανότητα να υπάρξει λάθος αποτέλεσμα από προσωρινή αστοχία υλικού.

Ξεκινάει ελέγχοντας αν το εκτελέσιμο , εκτελείται με ορίσματα. Αν υπάρχουν ορίσματα, το πρώτο όρισμα περνάει σαν filename του xml αρχείου που θα σωθούν τα αποτελέσματα και ανοίγει το αρχείο , ενώ αν δεν έχει κανένα όρισμα, τότε χρησιμοποιεί σαν default όνομα αρχείου το result.xml και το ανοίγει.

```
if(argc>1){
      myFile = fopen(argv[1], "wb+");
printf(argv[1]);
}
else {
myFile= fopen("result.xml","wb+");
printf("no arguments where given, using result.xml instead \n");
}
printf("Cuda Device Benchmark  \n");
fprintf(myFile,"<?xml version=\"1.0\"?>\n");
fprintf(myFile,"<benchmark>\n");
```

Αμέσως μετά ελέγχεται ο αριθμός των καρτών γραφικών που υποστηρίζουν CUDA,και ξεκινάει να τρέχει τα benchmarks για κάθε κάρτα ξεχωριστά.

```
//get cuda device count
printf("Detecting Cuda Devices  \n");
cudaGetDeviceCount(&devicecount);
// Device Selection
printf(" %d devices detected  \n",devicecount);
```

Ξεκινάει το benchmarkγια κάθε κάρτα ξεχωριστά και γράφονται τα αρχικά κομμάτια του XMLαρχείου. Σε ποια κάρτα θα τρέξει το benchmark το δηλώνουμε με την εντολή status = cudaSetDevice();

```
//starting benchmark loop for every detected device
for (dev=0;dev<devicecount;dev++){
      printf("running benchmark on device %d \n",dev);
error_id = cuDeviceGetName(deviceName, 256, dev);
if (error_id != CUDA_SUCCESS)
       {
printf("cuDeviceGetName returned %d\n-> %s\n", (int)error_id, getCudaDrvErrorString(error_id));
printf("Result = FAIL\n");
exit(EXIT_FAILURE);
       }
```

```
        printf("\nDevice %d: \"%s\"\n", dev, deviceName);

//setting device
status = cudaSetDevice(dev);
if (status != cudaSuccess) {
printf ("!!!! Set Device error\n");
returnEXIT_FAILURE;
}

//starting result file
fprintf(myFile,"<gpu>\n");
fprintf(myFile,"<gpuid>%d</gpuid>\n",dev);
fprintf(myFile,"<gpuName>%s</gpuName>\n",deviceName);
```

Τρέχουν πρώτα οι μετρήσεις για την μνήμη. Κάθε μέτρηση θα τρέξει 10 φορές για ποιο καθαρά αποτελέσματα. (MEMCOPYITTERATIONS=10)

```
//starting memory copy benchmark
printf("Initiating Memory Bandwidth Test\n");
//Initializing values
mmcpresult.d2h=0.0;
mmcpresult.d2d=0.0;
mmcpresult.h2d=0.0;

//Running memory copy MEMCOPYITERATIONS times
for(i=0;i<MEMCOPYITTERATIONS;i++){
mmcptemp=memcopy();
mmcpresult.d2h=mmcpresult.d2h+mmcptemp.d2h;
mmcpresult.d2d=mmcpresult.d2d+mmcptemp.d2d;
mmcpresult.h2d=mmcpresult.h2d+mmcptemp.h2d;
printf(".");
}

//extrapolating result
mmcpresult.d2h=mmcpresult.d2h/MEMCOPYITTERATIONS;
mmcpresult.d2d=mmcpresult.d2d/MEMCOPYITTERATIONS;
mmcpresult.h2d=mmcpresult.h2d/MEMCOPYITTERATIONS;

//Printing results to console and file
printf("\n Memory Bandwidth   \n\n Host to Device  Device to Device      Device to Host\n %f Gb/s
%f Gb/s         %fGb/s \n\n",mmcpresult.h2d/1024,mmcpresult.d2d/1024,mmcpresult.d2h/1024);
fprintf(myFile,"<h2d>%f</h2d>\n<d2d>%f</d2d>\n<d2h>%f</d2h>\n",mmcpresult.h2d/1024,mmcpresult.d2d/1
024,mmcpresult.d2h/1024);
```

Μετά καλείται η mat_mul() 10 φορές και αυτή και τα αποτελέσματα της εκτυπώνονται.

```
//starting Matrix Multiplication Benchmark

printf(" Commencing Matrix Multiplication" );
//running the benchmark MATMULITTERATIONS times
for(i=0;i<MATMULITTERATIONS;i++){
printf(".");
matrixresult=matrixresult+matrixmul();
```

```
}
//printing results on file and console
printf("\n Matrix Multiplication time :   %f (ms) \n\n",matrixresult/MATMULITTERATIONS);
fprintf(myFile,"<mmul>%f</mmul>\n",matrixresult/MATMULITTERATIONS);
```

Στην συνέχεια καλείται η LOS()

```
//running benchmark LOSITTERATIONS times
for(i=0;i<LOSITTERATIONS;i++){
printf(".");
losresult=losresult+los();
}
//printing results on file and console
printf("\n\n Time to complete Line of Sight Detection  %f (ms)\n",losresult/LOSITTERATIONS);
fprintf(myFile,"<los>%f</los>\n",losresult/LOSITTERATIONS);
```

Τέλος κλείνουν τα TAGS για το XML αρχείο, καθαρίζονται όλα τα CUDA νήματα , και η εφαρμογή σταματάει

```
//close tags for GPU on XML
fprintf(myFile,"</gpu>\n");
}
//close tags for BENCHMARK on XML and close file
fprintf(myFile,"</benchmark>");
fclose(myFile);
//kill all stray cuda threads
cudaThreadExit();
	}
```

4.2 Το γραφικό περιβάλλον

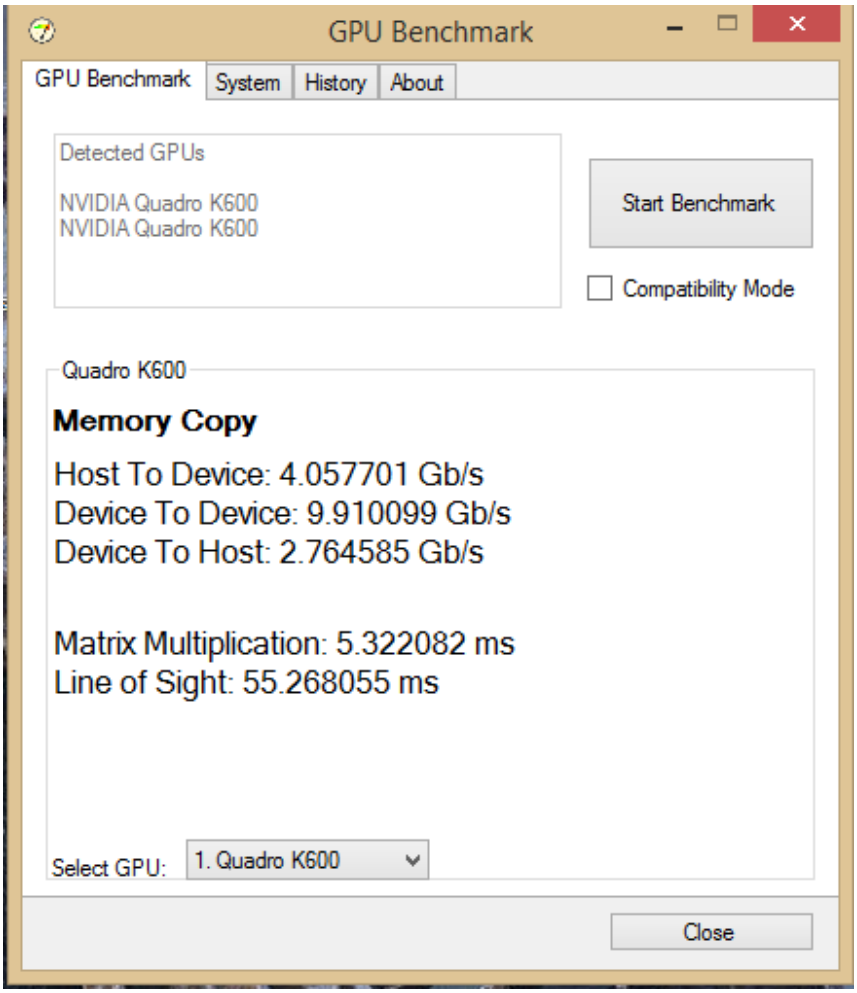
Ένα γραφικό περιβάλλον γραμμένο σε Visual Basic που παρέχει μια ευκολότερη και χρηστικότερη δυνατότητα διαχείρισης του μετροπρογράμματος.

Το γραφικό περιβάλλον αποτελείται από μία φόρμα με 4 καρτέλες (tabs).

Στην πρώτη καρτέλα, που είναι και η κύρια του προγράμματος, έχουμε textbox που περιέχει μια λίστα με τις κάρτες γραφικών (CUDA και μη) που υπάρχουν εγκατεστημένες στον υπολογιστή. Την λίστα αυτή την παίρνουμε από το WMI του συστήματος. Ακριβώς δεξιά του υπάρχει το κουμπί εκτέλεσης του Benchmark. Κάνοντας κλικ στο κουμπί, θα τρέξει αυτόματα η console εφαρμογή, και μόλις τελειώσει, το γραφικό περιβάλλον θα διαβάσει το παραγόμενο XML και θα εμφανίσει τα αποτελέσματα ακριβώς από κάτω σε ένα groupbox που θα περιέχει τα αποτελέσματα από όλα τα benchmarks.

Για καθαρά λόγους επίδειξης, υπάρχει και ένα checkbox με όνομα Compatibility Mode, που όταν είναι επιλεγμένο , όταν ο χρήστης πατήσει το κουμπί Start Benchmark, αντί να τρέξει το κανονικό μετροπρόγραμμα, απλά θα δημιουργηθεί ένα DEMO XML αρχείο με προεπιλεγμένα αποτελέσματα.

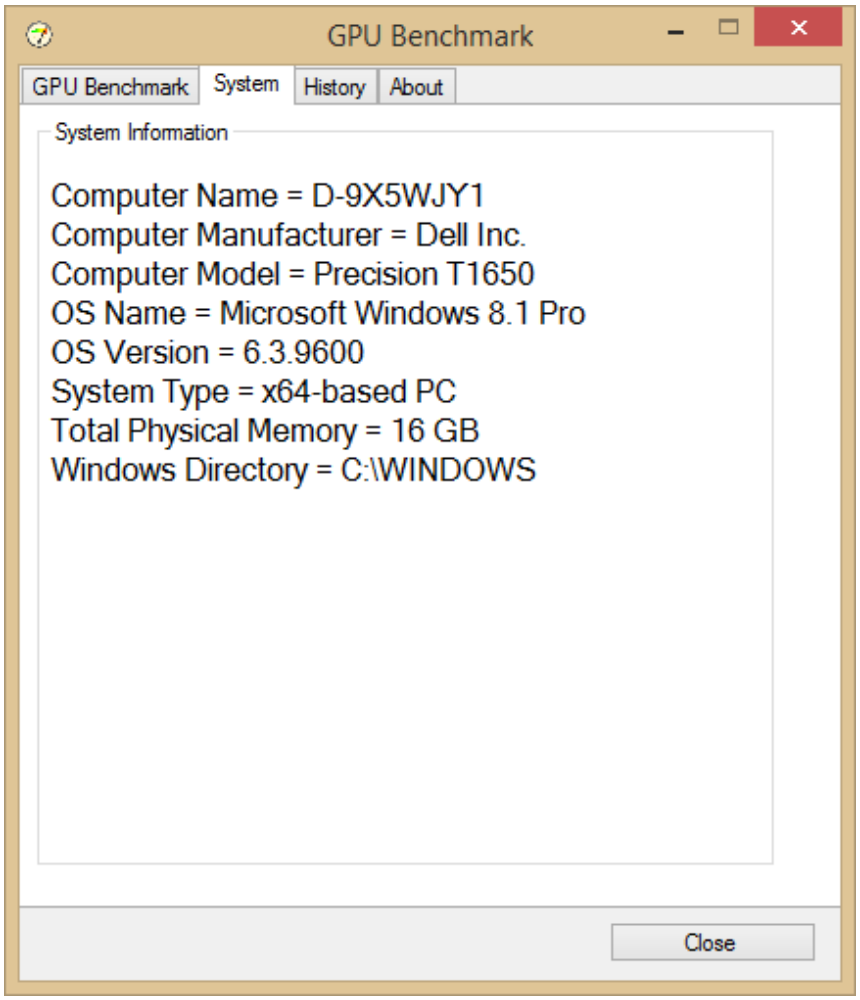
Τέλος υπάρχει και ένα dropdown μενού, που ενεργοποιείται όταν υπάρχει παραπάνω από μία CUDA ENABLED κάρτα γραφικών, και μας επιτρέπει να επιλέγουμε σε πιά κάρτα θέλουμε να δούμε τα αποτελέσματα.



Εικόνα 13: Το γραφικό Περιβάλλον (καρτέλα GPUBenchmark)

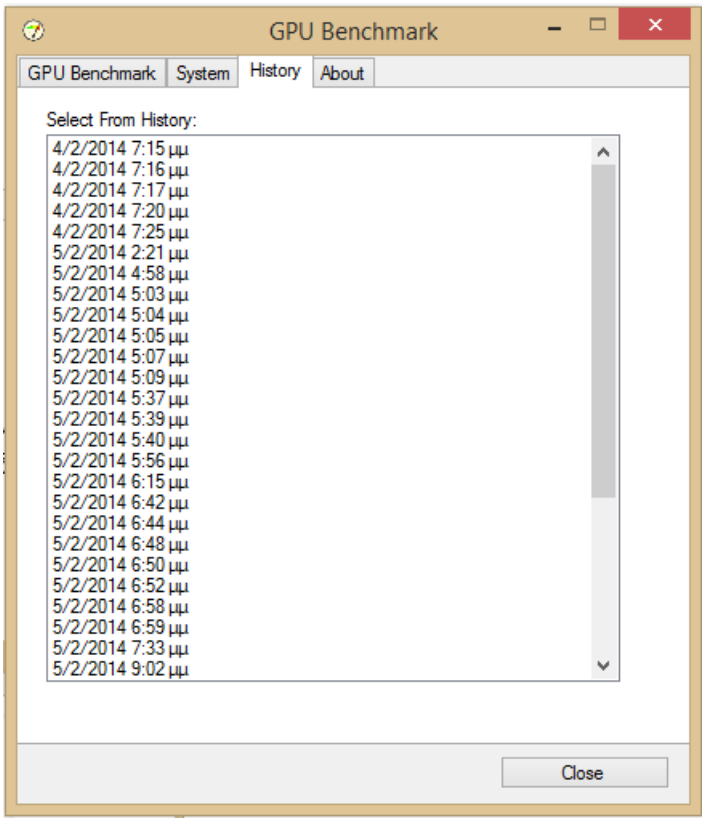
Στην δεύτερη καρτέλα , με το όνομα System, έχουμε ένα groupbox που ονομάζεται System Information, και περιέχει μερικές βασικές πληροφορίες για το HOST σύστημα μας, όπως Όνομα υπολογιστή, Κατασκευαστή, Μοντέλο, Όνομα και έκδοση λειτουργικού, είδος συστήματος, εγκατεστημένη μνήμη RAM και το Path του WINDOWS DIRECTORY.

Όλα αυτά τα στοιχεία τα παίρνουμε από το WMI του συστήματος



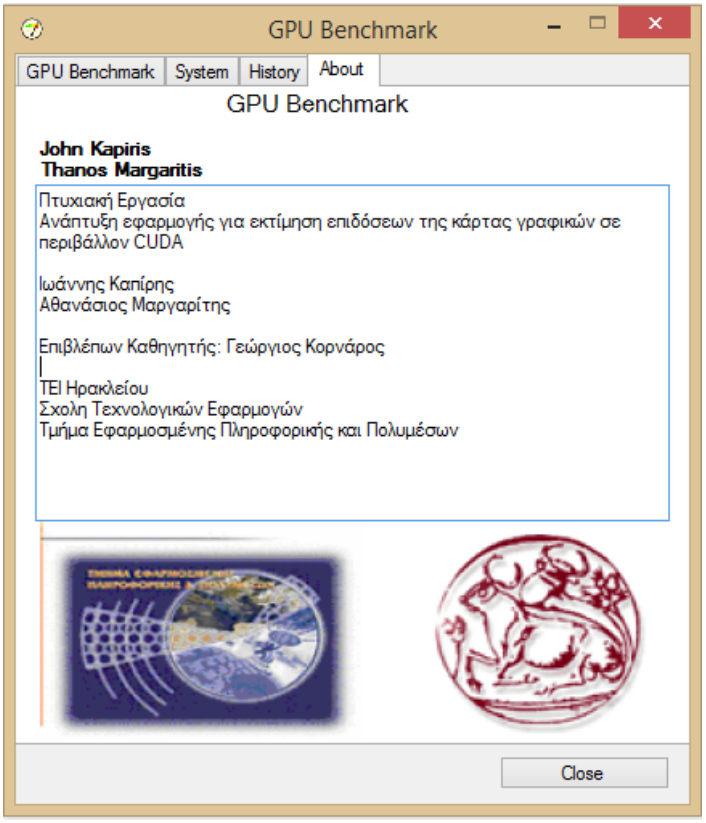
Εικόνα 14: Το γραφικό περιβάλλον (καρτέλα System)

Στην Τρίτη καρτέλα υπάρχει το ιστορικό με τις παλαιότερες εκτελέσεις του μετροπρογράμματος. Κάθε φορά που τρέχει το μετροπρόγραμμα, το XML αρχείο αντί να διαγράφεται , αποθηκεύεται, και μπορεί να ανακαλεστεί από την καρτέλα History. Έτσι ο χρήστης μπορεί να έχει ένα πλήρες ιστορικό από της επιδόσεις της κάρτας γραφικών του, και να συγκρίνει για παράδειγμα τις επιδόσεις πριν και μετά από υπερχρονισμό, η από εγκατάσταση νέων οδηγών.



Εικόνα 15: Το γραφικό περιβάλλον (καρτέλα History)

Τέλος, η τελευταία καρτέλα (About) περιέχει γενικές πληροφορίες για το πρόγραμμα και τους προγραμματιστές του.



Εικόνα 16: Το γραφικό περιβάλλον (καρτέλα About)

4.3 Διεπικοινωνία (interconnectivity)

Η επικοινωνία μεταξύ του GUIκαι της console εφαρμογής, γίνεται μέσω ενός XML αρχείου.

Το XML αυτό αρχείο είναι συμβατό και ακολουθεί τα πρότυπα του XMLVersion 1.0 [1]

Κατά την εκτέλεση της η ConsoleΕφαρμογή δημιουργεί ένα XMLαρχείο που περιέχει όλα τα αποτελέσματα του Benchmarkπου έχει εκτελεστεί. Έχει ιεραρχική δομή. Στην κορυφή του σχήματος είναι το Benchmark, το οποίο μπορεί να περιέχει τα αποτελέσματα για κάθε κάρτα γραφικών που μετρήθηκε στην εκτέλεση. Ακριβώς από κάτω είναι οι GPU. Φτιάχνεται ένα αντικείμενο GPUγια κάθε κάρτα που μετριέται. Σε κάθε αντικείμενο GPU υπάρχουν 7 τιμές. Το gpuid που έχει το CudaID της κάρτας, και είναι και ο αύξων αριθμός της, το gpuName που περιέχει το όνομα της κάρτας γραφικών, όπως το επιστρέφει ο Drivertης κάρτας γραφικών, τα h2d d2d d2h που είναι οι μετρήσεις από τον έλεγχο της ταχύτητας μνήμης, δηλαδή το Host to Device, Device to Device και Device to host, το mmul που είναι το αποτέλεσμα της μέτρησης πολλαπλασιασμού πινάκων, και τέλος το los που είναι το αποτέλεσμα της μέτρησης της Οπτικής επαφής (Line of sight)

Όταν τελειώνει η εκτέλεση της Consoleεφαρμογής , το παραγόμενο XMLγίνεται PARSEαπό το γραφικό περιβάλλον και εμφανίζονται τα αποτελέσματα. Κάθε XMLαρχείο που φτιάχνεται , έχει ξεχωριστό όνομα που προέρχεται από την ημερομηνία και ώρα εκτέλεσης του Benchmark

[1]http://www.w3.org/XML/#intro

Παράδειγμα XML αρχείου

```
<?xmlversion="1.0"?>
<benchmark>
<gpu>
<gpuid>0</gpuid>
<gpuName>Quadro K600</gpuName>
<h2d>4.110348</h2d>
<d2d>9.916353</d2d>
<d2h>2.792971</d2h>
<mmul>5.265943</mmul>
<los>55.766850</los>
</gpu>
<gpu>
<gpuid>1</gpuid>
<gpuName>Quadro K600</gpuName>
<h2d>1.254791</h2d>
<d2d>9.616291</d2d>
<d2h>1.218491</d2h>
<mmul>10.216842</mmul>
<los>179.192795</los>
</gpu>
</benchmark>
```

5. Εργαλεία-Πόροι

Και για τις δύο εφαρμογές, το compile έγινε με Microsoft Visual Studio 2012.
Στην περίπτωση του κώδικα που περιέχει και κώδικα Cuda , ήταν αναγκαία η χρήση του Nvidia Cuda SDK που παρέχει τον compiler nvcc για CUDA , και του NVIDIA Nsight Visual Studio Edition που βοηθάει στην δημιουργεί Profile στο Visual Studio που να υποστηρίζει κώδικα γραμμένο σε Cuda.Τέλος, χρησιμοποιήσαμε την εφαρμογή TechPowerup GPU-Zγια να παίρνουμε πληροφορίες για τις κάρτες γραφικών και να συγκρίνουμε τα χαρακτηριστικά τους, με τις επιδόσεις τους και τα αποτελέσματα των μετρήσεων μας

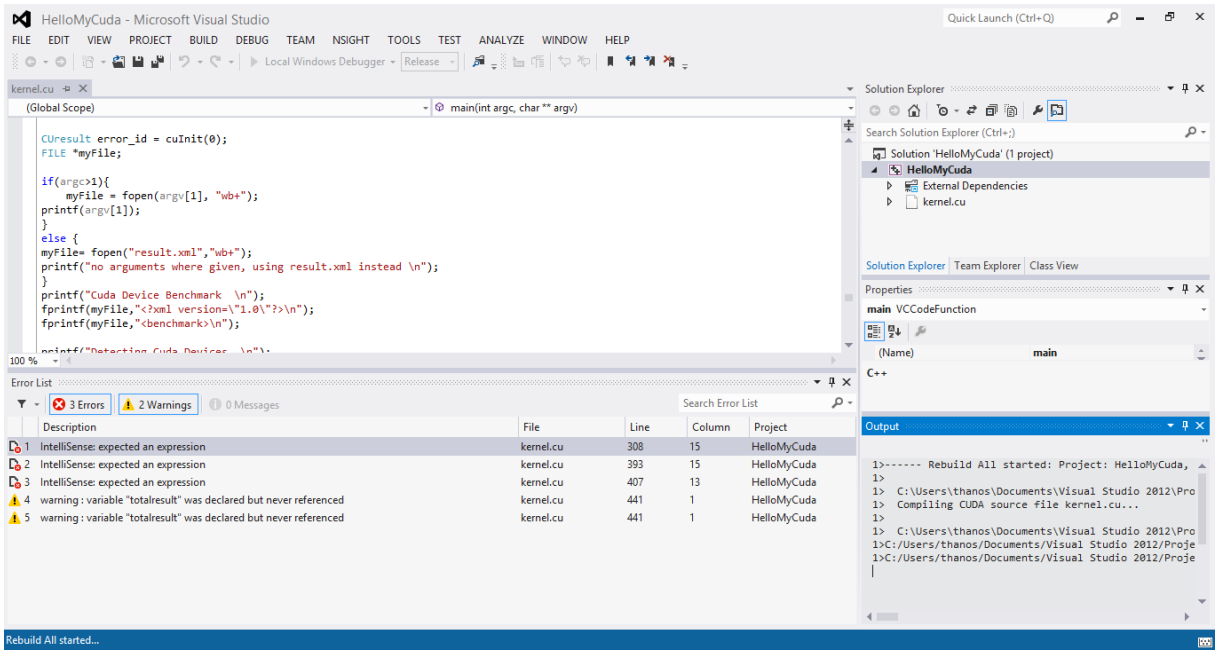
5.1 Visual Studio 2012

Το Microsoft Visual Studio είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment – IDE) από την Microsoft. Χρησιμοποιείται για την ανάπτυξη προγραμμάτων για Microsoft Windows, και επίσης για Ιστοσελίδες, εφαρμογές WEB και υπηρεσίες WEB. Χρησιμοποιεί πλατφόρμες ανάπτυξης λογισμικού όπως τα Windows API, Windows Forms, Windows Presentation Foundation, Windows Store και Microsoft Silverlight. Μπορεί να παράγει και εγγενή και διαχειριζόμενο κώδικα.

Περιλαμβάνει ένα επεξεργαστή κώδικα (code editor) που υποστηρίζει την τεχνολογία IntelliSense.Ο ενσωματωμένος debugger, δουλεύει και σε πηγαίο επίπεδο, αλλά και σε επίπεδο μηχανής.

Άλλα ενσωματωμένα εργαλεία περιλαμβάνουν ένα εργαλείο σχεδιασμού φορμών για την ανάπτυξη εφαρμογών γραφικού περιβάλλοντος, σχεδιαστές WEB, σχεδιαστές κλάσεων , και έναν σχεδιαστή σχήματος βάσεων δεδομένων.

Δέχεται plug-ins που αυξάνουν την χρηστικότητα σε κάθε σχεδόν επίπεδο, και που προσθέτουν νέα εργαλεία όπως επεξεργαστές και σχεδιαστικά προγράμματα για γλώσσες domain-specific ή και εργαλεία για άλλα κομμάτια της ανάπτυξης λογισμικού.



Εικόνα 17. Το περιβάλλον λειτουργίας του Microsoft Visual Studio 2012

Το Visual Studio υποστηρίζει διάφορες γλώσσες προγραμματισμού και επιτρέπει στον επεξεργαστή κώδικα και στον debugger να υποστηρίζουν (σε διαφορετικό βαθμό) σχεδόν κάθε γλώσσα προγραμματισμού, αν υπάρχει

μια σχετική με την γλώσσα υπηρεσία. Οι ενσωματωμένες γλώσσες περιλαμβάνουν C/C++ , VB.net, C# και F#. Αλλες γλώσσες όπως οι M, Python, και Ruby, μπορούν να προστεθούν μέσων υπηρεσιών γλώσσας που εγκαθίστανται ξεχωριστά. Παρέχει υποστήριξη επίσης και σε XML/XSLT, HMTL/XHTML, JavaScript και CSS.

Σύμφωνα με τον οδηγό προϊόντος της Microsoft

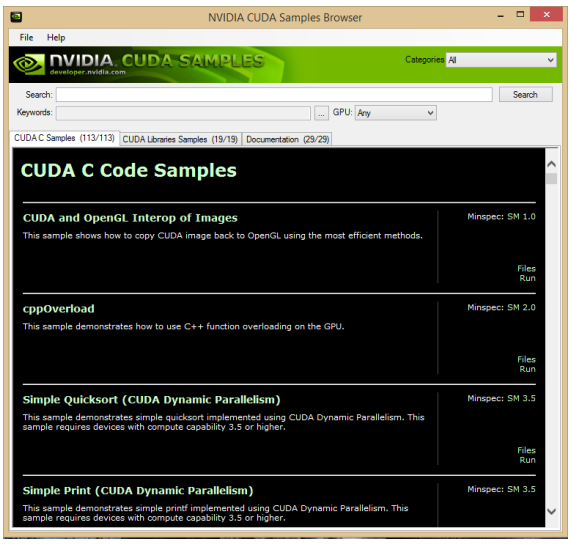
"Το Microsoft Visual Studio 2012 είναι μια ολοκληρωμένη λύση που επιτρέπει σε προγραμματιστές και ομάδες προγραμματιστών ανεξαρτήτου μεγέθους να μετατρέπουν τις ιδέες τους σε εξαιρετικές και εντυπωσιακές εφαρμογές. Δίνει την δυνατότητα σε όσους ασχολούνται με την παραγωγή λογισμικού να επωφεληθούν από τελευταίας τεχνολογίας εργαλεία, ώστε να δημιουργούν εμπειρίες που να ενθουσιάζουν τον τελικό χρήστη, χρησιμοποιώντας απλοποιημένες λύσης ανάπτυξης που επιτρέπουν την ποιοτική εκτέλεση όλων των εργασιών και ρόλων που περιλαμβάνονται σε ένα έργο λογισμικού.

Έχει σχεδιαστεί ώστε να φροντίζει ότι οι προγραμματιστές μπορούν να προσφέρουν μια συνεχής ροή αξίας στην επιχείρηση. Το περιβάλλον έχει ξανασχεδιαστεί ώστε να μειωθεί το μέγεθος των άχρηστων δεδομένων από την οθόνη, ενώ ταυτόχρονα να προσφέρει γρήγορα πρόσβαση στα εργαλεία που χρησιμοποιούνται πιο συχνά. "

5.2 Nvidia CUDA SDK, Nvidia Cuda Toolkit και Nvidia Nsight Visual Studio Edition

Η Nvidia παρέχει σε όλους όσους θέλουν να αναπτύξουν εφαρμογές σε CUDA περιβάλλον, κάποια εργαλεία τα οποία επιτρέπουν στον προγραμματιστή να αναπτύξει εφαρμογές CUDA.

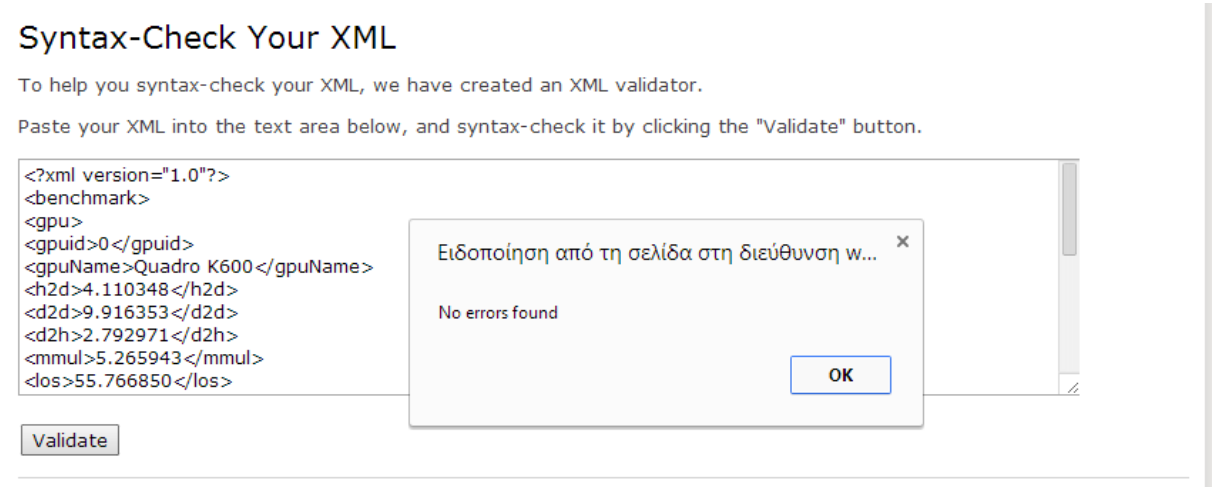
Το Nvidia Cuda Toolkit παρέχει στον χρήστη τα απαραίτητα εργαλεία για να ξεκινήσει προγραμματισμό σε CUDA. Προσφέρει έναν CUDA Enabled οδηγό για την κάρτα γραφικών, το CUDASDKπου περιέχει της απαραίτητες βιβλιοθήκες , και τον compilerγια κώδικα CUDA (NVCC) . Παρέχει επίσης και αρκετές DEMOεφαρμογές, μαζί με τον sourceκώδικα(Samples Browser) τους, που δείχνουν τις δυνατότητες της CUDA.



Εικόνα 18: Το περιβάλλον του Samples Browserμε τις εφαρμογές επίδειξης της Cuda

5.3 XML - XMLValidator

Ο XMLValidator είναι ένα WEB Application που ελέγχει αν τα παραγόμενα XMLαπό μία εφαρμογή είναι απόλυτα συμβατή με τις προδιαγραφές του XML. Υπάρχει στο http://www.w3schools.com/xml/xml_validator.asp



Εικόνα 19: Η σελίδα του XMLValidator εν δράση

5.4 TechPowerup GPU-Z

Το GPU-Zείναι μια εφαρμογή της TechPowerup , που προσφέρει λεπτομερείς πληροφορίες για κάθε κάρτα γραφικών εγκατεστημένη στο σύστημα μας, όπως Bus Interface, Memory Type και Bus Width, χρονισμούς, αριθμό Shaders και πολλά άλλα

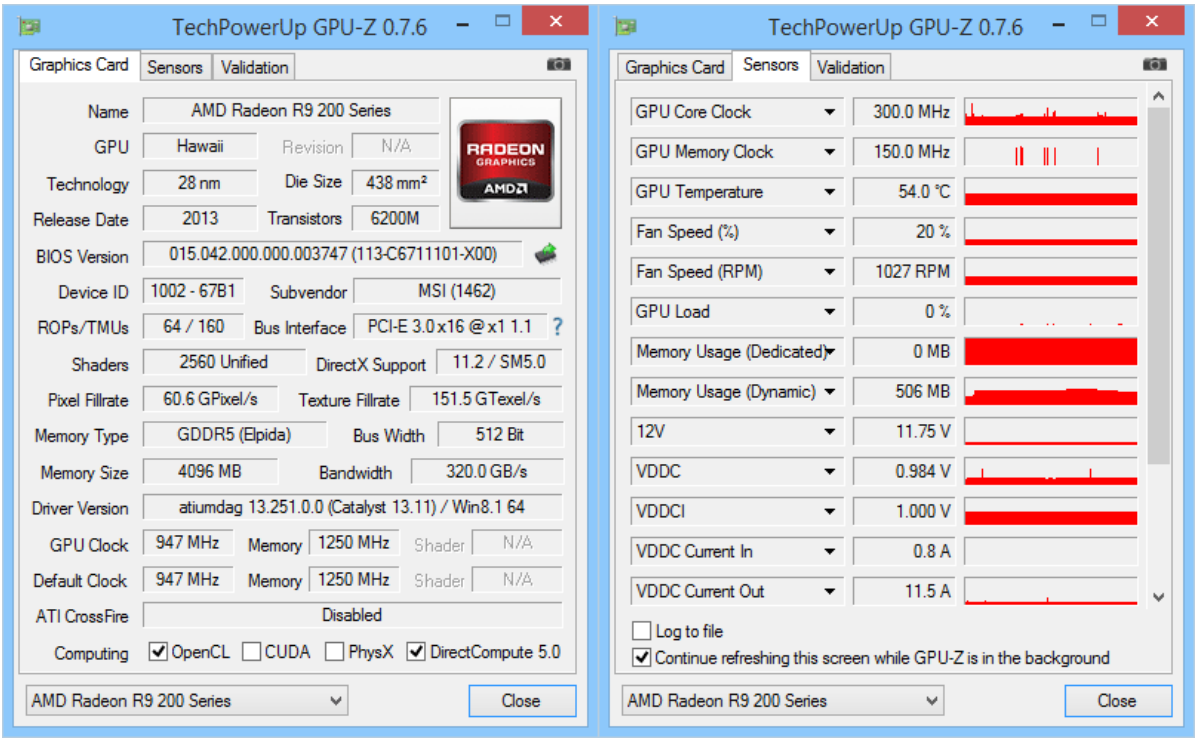
Από την ιστοσελίδα της εταιρίας:

«Το GPU-Z είναι ένα ελαφρύ εργαλείο συστήματος σχεδιασμένο να προσφέρει ζωτικής σημασίας πληροφορίες για την Κάρτα και τον Επεξεργαστή Γραφικών σας

Κύρια χαρακτηριστικά

- Υποστηρίζει Nvidia, Ati και Intel κάρτες γραφικών
- Δείχνει πληροφορίες για προσαρμογέα , ΓΕ , και ρυθμίσεων απεικόνισης
- Δείχνει πληροφορίες για συχνότητες υπερχρονισμού, defaultσυχνότητες και συχνότητες 3D (όπου είναι εφικτό)
- Περιέχει τεστ υπερφόρτωσης του ΕΓ για την επιβεβαίωση της διαμόρφωσης των Γραμμών PCI-Express
- Επιβεβαίωση αποτελεσμάτων
- Μπορεί να κρατήσει αντίγραφο ασφαλείας του BIOSτης κάρτας γραφικών σας
- Δεν χρειάζεται εγκατάσταση

- Υποστήριξη σεWindowsXP / Vista /Windows Windows 8 (32 και 64 bit)»



Εικόνα 20: Το GPU-Z

6. Αποτελέσματα-Συμπεράσματα

Με την ενασχόληση μας με την παρούσα εργασία, αποκτήσαμε πολύ εξειδικευμένες γνώσεις, και πάνω στον σχεδιασμό των Καρτών Γραφικών, όπως πχ τα ήδη μνήμης, η ταυτότητά τους, και στο πως μπορούμε να σχεδιάζουμε καλύτερα προγράμματα με υψηλή παραλληλία και μεγάλη Αριθμητική Ένταση, έννοιες που δεν υπάρχουν στον κλασικό προγραμματισμό, άλλα αναπτύχθηκαν ταυτόχρονα με την ανάπτυξη πολυπυρηνικών επεξεργαστών.

Όπως αναφέραμε και στην αρχή αυτής της παρουσίασης, ο στόχος μας ήταν να ετοιμάσουμε μια ολοκληρωμένη λύση που να μετράει τις πραγματικές επιδόσεις μιας κάρτας γραφικών σε περιβάλλον CUDA. Αυτό σημαίνει ότι θα έπρεπε να προσφέρουμε μετρήσεις που να επηρεάζονται από τα χαρακτηριστικά της Κάρτας Γραφικών, και 2 διαφορετικές κάρτες να μην έχουν τα ίδια αποτελέσματα.

Θα έπρεπε λοιπόν τα αποτελέσματα που μας δίνει το πρόγραμμα μας, να έχουν συνάφεια και με τα χαρακτηριστικά των καρτών.

Γι' αυτό λοιπόν ετοιμάσαμε έναν πίνακα με τα αποτελέσματα κάθε μέτρησης , αλλά και με τα χαρακτηριστικά τις κάθε κάρτας.

GPU	Αριθμός Shaders	GPU Clock	Memory Clock	Bus Interface	PCI Express Lanes	Bus Width	Theoretical Memory Bandwidth		Host To Device (Gb/s)	Device to Device (Gb/s)	Device to Host (Gb/s)	Matrix Multiplication (ms)	Line of Sight(ms)
Quadro K600 (1)	192	876	891	PCI-E 2.0 x	16	128	28.5	RESULTS	4,03	9,89	2,83	5,06	54,05
Quadro K600 (2)	192	876	891	PCI-E 2.0 x	4	128	28.5		1,24	9,49	1,22	10,03	176,77
Geforce GT740M	384	980	900	PCI-E 2.0 x	8	64	14,4		1,14	5,9	1,14	2,06	133,2
Geforce 9600GT	64	650	900	PCI-E 2.0 x	16	256	57,6		1,99	14,01	1,53	1,1	101,43
Geforce GT440	96	780	800	PCI-E 2.0 x	16	128	28.5		1,85	10,2	1,5	4,6	91
Geforce GT620	48	810	897	PCI-E 2.0 x	16	64	14,4		3,9	5,77	2,52	7,8	59,73
Geforce GTX760	1152	1020	1502	PCI-E 3.0x	16	256	192,3		3,05	69,72	2,7	0,6	53,11
Quadro K2000	384	954	1000	PCI-E 2.0 x	16	128	64		2,3	23,15	1,69	2,1	89,26
Quadro K4000	768	811	1404	PCI-E 2.0 x	16	192	134,8		1,98	42,8	1,05	1,46	118

Πίνακας 1: Χαρακτηριστικά και μετρήσεις σε κάρτες

Όπως βλέπουμε τα αποτελέσματα αντικατοπτρίζουν τα χαρακτηριστικά της κάθε κάρτας.

Για παράδειγμα το Host to Device όπως ήταν αναμενόμενα, επηρεάζεται πιο πολύ από την διεπαφή με τον υπολογιστή. Δηλαδή το Bandwidth του δίαυλού PCI-Express. Πιο συγκεκριμένα, για PCI-E 2.0 , κάθε 4 lanesμας δίνουν περίπου από 1Gb/s Bandwidth. Λιγότερο επηρεάζει ο χρονισμός της μνήμης ,και το πλάτος του δίαυλου σε bit. Σχεδόν ακριβός το ίδιο επηρεάζεται και το DevicetoHost, αλλά εδώ έχουμε πιο εμφανές το bottlenecktου επεξεργαστή και της μνήμης του υπολογιστή.

Το DevicetoDevice, δηλαδή η αντιγραφή εσωτερικά της κάρτας, είναι απόλυτα συνυφασμένη με τον χρονισμό της μνήμης και με το πλάτος του δίαυλου (bus width). Και αυτή η συμπεριφορά είναι αναμενόμενη.

Το matrix multiplication, επηρεάζεται πιο πολύ από τον χρονισμό των shaders, και λιγότερο από την ταχύτητα της μνήμης, παρ όλα αυτά επηρεάζεται εμφανώς και από τα δυο.

Η μέτρηση για το Line of Sight από την άλλη, επηρεάζεται πιο πολύ από τον αριθμό των Sharedπου υπάρχουν, και από την ταχύτητα της μνήμης και του δίαυλου PCI-E, και λιγότερο από τον χρονισμό των Shaders και της μνήμης.

Και οι 3 μετρήσεις μαζί, ελέγχουν κάθε χαρακτηριστικό της Κάρτας Γραφικών , και προσφέρουν μια ολοκληρωμένη μέτρηση των επιδόσεων. Τα αποτελέσματα συμφωνούν απόλυτα με τις προσδοκίες μας, και το πρόγραμμα μας δίνει πια μια ξεκάθαρη εικόνα της απόδοσης της κάθε κάρτας.

Κλείνοντας, καλό θα ήταν να αναφερθούμε και στα προτερήματα της CUDA στον πραγματικό κόσμο. Οι δυνατότητες είναι πραγματικά τεράστιες. Το σύνολο τον εφαρμογών που υπάρχουν για να επεξεργάζονται μεγάλο αριθμό δεδομένων, ξεχωριστών μεταξύ τους μπορούν να έχουν πολύ μικρότερους χρόνους εκτέλεσης. Εφαρμογές όπως πχ η επεξεργασία είκονας, που κάθε εικόνα θεωρείται ένας πίνακας από ξεχωριστά μεταξύ τους pixel, επεξεργασία βίντεο, εφαρμογές γεωγραφικών συστημάτων , εφαρμογές κινηματικής κ.α. μπορουν να εκτέλεστουν από 2 μέχρι και 150 φόρες πιο γρήγορα.

Διάφορες γνωστές εφαρμογές και το όφελος σε ταχύτητα περιλαμβάνουν τις εξής (σύμφωνα με την ιστοσελίδα της NVidia το 2014)

- Adobe Photoshop (Image manipulation effects)
- Amira (3d Visualization of volumetric data and surfaces) 70x αύξηση ταχύτητας
- ArcVideo Core (encoding/decoding video) 10~40x αύξηση ταχύτητας\
- MUMmet GPU : Επεξεργασία αλληλουχίας DNA 3,4~3,8x αύξηση ταχύτητας
- Matlab
- Ανίχνευση εικόνας με χρήση νευρωνικών δυκτίων (Dr. Dan Ciresan, Swiss Al Lab IDSIA, Switzerland) 10x αύξηση ταχύτητας

Με τέτοιο μεγάλο κέρδος , το μόνο σίγουρο είναι ο η CUDA και η GPGPU γενικότερα θα αποτελέσουν μεγάλο μέρος του πως γράφουμε πια προγράμματα.

Βιβλιογραφία

[1] Nvidia Cuda Getting Started Guide http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/index.html

[2] Nvidia Cuda Programming Guide http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[3] Nvidia Cuda Best Practices Guide http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[4] Nvidia Cuda Runtime API http://docs.nvidia.com/cuda/cuda-runtime-api/index.html

[5] Unified Memory Reports for CUDA 6 http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/

[6] Microsoft Visual Studio 2012 Product Guide http://download.microsoft.com/download/C/D/3/CD39BB69-35CC-458A-B4EB-2F928B58FB4B/visual-studio-2012/Visual-Studio-2012-Product-Guide.pdf

[7] W3C Extensive Markup Language XML http://www.w3.org/XML/#intro

[8] W3 Schools XML Tutorial http://www.w3schools.com/xml/

[9] W3 Schools XML Validator http://www.w3schools.com/xml/xml_validator.asp

[10] Nsight Visual Studio Edition User Guide http://docs.nvidia.com/gameworks/index.html#developertools/desktop/nvidia_nsight.htm

Ανάπτυξη εφαρμογής για εκτίμηση επιδόσεων της κάρτας γραφικών σε περιβάλλον CUDA

Πτυχιακή εργασία Τμήματος Εφαρμοσμένης πληροφορικής και Πολυμέσων - Σχολής Τεχνολογικών Εφαρμογών - ΑΤΕΙ Ηρακλείου

(Επιβλέπων καθηγητής Γ.Κορνάρος)

Ιωάννης Καπίρης (*Author*) ΑΜ 686
Σχολή: ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ
Σχολή Τεχνολογικών Εφαρμογών ΑΤΕΙ ΗΡΑΚΛΕΙΟΥ
epp697@epp.teiher.gr

Αθανάσιος Μαργαρίτης (*Author*) ΑΜ 737
Σχολή: ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ
Σχολή Τεχνολογικών Εφαρμογών ΑΤΕΙ ΗΡΑΚΛΕΙΟΥ
epp737@epp.teiher.gr

***Σύνοψη*—Η παρουσίαση αυτή διαπραγματευται την έρευνα και υλοποίηση μιας εφαρμογής , η οποία μετράει τις επιδόσεις καρτών γραφικών Nvidia στο προγραμματιστικό Περιβάλλον CUDA.**

Keywords—gpgpu; cuda; benchmark; kernels;

Εισαγωγή

Νόμος του Moore
"The number of transistors on an integrated Circuit doubles every two years."

– GordonE. Moore

Οι σύγχρονοι πια επεξεργαστές έχουν φτάσει πια σε σημείο που δεν είναι εφικτή η μεγαλύτερη συχνότητα λειτουργίας του. Έτσι, για έχουμε μεγαλύτερη υπολογιστική ισχύ αναγκαζόμαστε να χρησιμοποιούμε παραπάνω επεξεργαστικούς πυρήνες. Αυτή η πρακτική οδήγησε στην ανάπτυξη της προγραμματιστικής πρακτικής της παράλληλης επεξεργασίας. Τα προγράμματά μας πρέπει να είναι σχεδιασμένα έτσι ώστε να μπορούν να χρησιμοποιήσουν τους παραπάνω επεξεργαστικούς πυρήνες. Θα πρέπει να φροντίζουμε ώστε να τα δεδομένα μας να υπολογίζονται με τέτοιο τρόπο (οπου είναι εφικτό) ώστε οι υπολογισμοί σε αυτά να είναι διακριτοί, ξεχωριστοί μεταξύ τους, και να μπορούν να εκτελούνται ανεξάρτητα.

Με την άνθηση αυτής της προγραμματιστικής παραλληλίας, ήρθε στο προσκήνιο η ιδέα του GPGPU. Δηλαδή να γίνεται χρήση της κάρτας γραφικών, η οποία είναι ένα σύστημα με πάρα πολλούς επεξεργαστικούς πυρήνες (έως και εκατοντάδες) , για την εκτέλεση προγραμμάτων γενικής χρήσης.
Έτσι γεννήθηκε η CUDA, μία προγραμματιστική πλατφόρμα της εταιρίας Nvidia που επιτρέπει στους προγραμματιστές να γράφουν και να εκτελούν προγράμματα στην κάρτα γραφικών, και να μπορούν να επωφεληθούν την υψηλή αυτή παραλληλία.

Με την εισαγωγή της CUDA στην ζωή μας, και με τις πάρα πολλές και διαφορετικών δυνατοτήτων κάρτες γραφικών, γεννήθηκε η ανάγκη να γνωρίζουμε πως η κάρτα γραφικών μας, αποδίδει σε προγράμματα CUDA.
Σκοπός αυτής της εργασίας είναι η έρευνα και η υλοποίηση ενός τέτοιου προγράμματος, που θα μπορεί να μετρά με ακρίβεια, τις επεξεργαστικές δυνατότητες κάθε κάρτας που υποστηρίζει την πλατφόρμα CUDA.

Στην παρουσίαση αυτή θα πούμε λίγα λόγια για την GPGPU, θα αναπτύξουμε πάνω στην CUDA , και μετά θα παρουσιάσουμε την εφαρμογή μας, και θα κλείσουμε με τα συμπεράσματα και με τα αποτελέσματα που είχαμε από την εργασία αυτή

GPGPU

GPGPU (Υπολογισμοί Γενικού Σκοπού σε Επεξεργαστές Γραφικών)

Η GPGPU(Υπολογισμοί Γενικού Σκοπού σε Επεξεργαστές Γραφικών) είναι μια τεχνική παράλληλης επεξεργασίας που επιτρέπει την χρήση μιας GPU (Μονάδα Επεξεργασίας Γραφικών) , η οποία συνήθως χειρίζεται υπολογισμούς γραφικών, για τον υπολογισμό εφαρμογών που συνήθως διαχειρίζονται από τον επεξεργαστή. Έτσι μας δίνεται η δυνατότητα χρήσης ενός συστήματος με ιδιαίτερα μεγάλες δυνατότητες παράλληλης επεξεργασίας, που υπάρχει ήδη στους περισσότερους υπολογιστές , παράγεται μαζικά και σε χαμηλό κόστος σε σχέση με άλλες λύσεις.

Προγραμματιστικές Έννοιες GPGPU
Οι Επεξεργαστές Γραφικών είναι σχεδιασμένοι ειδικά για να επεξεργάζονται γραφικά, και συνεπώς είναι ιδιαιτέρως περιοριστικές όσο αναφορά χειρισμούς και προγραμματισμό. Λόγο της φύσης τους, οι ΕΓ είναι αποδοτικοί μόνο στην αντιμετώπιση προβλημάτων που λύνονται με την χρήση «Επεξεργασίας Ροής» (stream processing).

Επεξεργασία Ροής(stream processing)

Οι ΕΓ από κατασκευής είναι σχεδιασμένες να υπολογίζουν μόνο δεδομένα διανυσματικής μορφής η κλάσματα (πράξεις κινητής υποδιαστολής)
Κατ' αυτή την έννοια οι ΕΓ θεωρούνται επεξεργαστές ροής, δηλαδή επεξεργαστές που λειτουργούν παράλληλα, τρέχοντας ταυτόχρονα έναν κοινό πυρήνα(πρόγραμμα) σε πολλαπλές ¨ροές¨.

- **Ροή(stream)** Ροή θεωρείται μια ομάδα καταχωρήσεων που απαιτούν πανομοιότυπους υπολογισμούς, και παρέχουν υψηλό παραλληλισμό δεδομένων.
- **Πυρήνες** Οι πυρήνες είναι οι λειτουργίες που εφαρμόζονται σε κάθε στοιχείο στην ροή.
  Επειδή τα δεδομένα σε μία ροή , επεξεργάζονται ανεξάρτητα το ένα με το άλλο, είναι αδύνατο να έχουν ταυτόχρονη πρόσβαση σε στατικά η κοινόχρηστα δεδομένα.
- **Αριθμητική Ένταση**. Αριθμητική ένταση είναι ο αριθμός των λειτουργιών που τελούνται ανά μεταφερόμενη λέξη μνήμης. Είναι σημαντικό για τις εφαρμογές GPGPU να έχουν υψηλή αριθμητική ένταση, γιατί αλλιώς οι καθυστερήσεις από την πρόσβαση μνήμης, καθιστούν το κέρδος από την παράλληλη επεξεργασία μηδαμινό.

Αρχιτεκτονική CUDA

Το μεγαλύτερο μειονέκτημα των GPGPU αρχιτεκτονικών, είναι ότι τα δεδομένα που πρόκειται να επεξεργαστούν, θα πρέπει να μοιάζουν με τα δεδομένα που συνήθως επεξεργάζεται ένας επεξεργαστής γραφικών, και οι επεξεργασίες που γίνονται πάνω σε αυτά τα δεδομένα , θα

πρέπει να ακολουθούν την δομή των εργασιών που κάνει συνήθως ένας επεξεργαστής γραφικών.

Σε αυτό έρχεται να δώσει λύση η αρχιτεκτονική CUDA.
Τον Νοέμβριο του 2006 η NVidia ανακοίνωσε την CUDA, μία πλατφόρμα παράλληλης επεξεργασίας που χρησιμοποιεί την μηχανή παράλληλης επεξεργασίας των καρτών γραφικών NVIDIA , για να λύσει διάφορα υπολογιστικά προβλήματα με έναν πιο αποδοτικό τρόπο απ' ότι ένας επεξεργαστής

Η πλατφόρμα της CUDA παρέχει ένα περιβάλλον προγραμματισμού που επιτρέπει στους προγραμματιστές να χρησιμοποιούν την C σαν γλώσσα υψηλού επιπέδου γράφοντας κώδικα που να τρέχει στην κάρτα γραφικών.

Η άφιξη των πολυπύρηνων επεξεργαστών και καρτών γραφικών συνεπάγεται ότι οι τα ολοκληρωμένα που βρίσκονται πια σε μαζική παραγωγή, αποτελούν παράλληλα συστήματα, που ταυτόχρονα η επεξεργαστική ισχύς τους μέσω της παραλληλίας αυξάνεται σύμφωνα με το νόμο του Moore. Η πρόκληση έγκειται στην ανάπτυξη εφαρμογών που να μπορούν να χρησιμοποιήσουν αυτή την παραλληλία και με διαφανή τρόπο να αλλάζουν το επίπεδο παραλληλίας τους ώστε να χρησιμοποιούν στο έπακρο τον αριθμό των αυξανόμενων επεξεργαστικών πυρήνων.
Το μοντέλο παράλληλης επεξεργασίας της CUDA είναι σχεδιασμένο να ξεπερνάει αυτό το εμπόδιο, ενώ ταυτόχρονα να είναι εύκολο στην εκμάθηση από προγραμματιστές που έχουν ήδη γνώσεις πάνω σε γλώσσες προγραμματισμού όπως η C.

Στον πυρήνα του έχει 3 βασικές έννοιες.

- Μία ιεραρχία από ομάδες προγραμματιστικών νημάτων
- Διαμοιραζόμενες μνήμες με συγκεκριμένη ιεραρχία
- Συγχρονισμό ορίων, που εμφανίζονται στον προγραμματιστή ως ένα ελάχιστο σετ από επεκτάσεις στην γλώσσα προγραμματισμού

Αυτές οι έννοιες παρέχουν έναν υψηλής ακρίβειας παραλληλισμό δεδομένων και νημάτων, εμφωλευμένα σε χαμηλότερης ακρίβειας δεδομένα και εργασίες παραλληλισμού. Οδηγούν τον προγραμματιστή να διαχωρίσει το πρόβλημα σε μικρότερα υποπροβλήματα που λύνονται ανεξάρτητα και σε παραλληλία, από κομμάτια (blocks) νημάτων, και κάθε υποπρόβλημα σε ακόμα μικρότερα τμήματα που μπορούν με την σειρά τους να λυθούν

Αυτός ο διαμερισμός εργασιών, επιτρέπει στην γλώσσα να διατηρεί την μορφή της επιτρέποντας στα νήματα να συνεργάζονται όταν λύνουν κάθε υποπρόβλημα, και την ίδια στιγμή να επιτρέπει αυτόματη διακλιμάκωση των εργασιών. Κάθε ομάδα (block) από νήματα μπορεί να τρέξει σε οποιονδήποτε από τους ελεύθερους πολύ-επεξεργαστές σε έναν ΕΓ , σε οποιαδήποτε σειρά, παράλληλα η σειριακά, ώστε κάθε πρόγραμμα που έχει γραφεί σε CUDA να μπορεί να τρέξει ανεξάρτητα κάθε φορά από τον διαθέσιμο αριθμό πολυεπεξεργαστών , και μόνο το σύστημα κατά την εκτέλεση να πρέπει να ξέρει τον φυσικό αριθμό πολυεπεξεργαστών σε κάθε ΕΓ.

Ο προγραμματισμός στην Cuda γίνεται μέσω του παρεχόμενου API και SDK που παρέχει η Nvidia και επιτρέπει σε προγραμματιστές, μέσω της γλώσσας C να δημιουργούν κώδικα ο οποίος έχει την δυνατότητα , κομμάτια του να εκτελούνται στον ΕΓ.

Αυτό γίνεται με την χρήση των πυρήνων CUDA (Cuda Kernels) , δηλαδή κομματιών κώδικα τα οποία γίνονται Compile από τον compiler της Cuda (nvcc) και όχι από κλασικό CompilerC , και ο κώδικας αυτός τρέχει στον ΕΓ και όχι στον κανονικό επεξεργαστή. Το γεγονός ότι ένα κομμάτι κώδικα εκτελείται από τον κανονικό επεξεργαστή και ένα άλλο κομμάτι του ίδιου προγράμματος τρέχει σε άλλη συσκευή, στην συγκεκριμένη περίπτωση σε έναν ΕΓ, λέγεται ετερογενής προγραμματισμός.

Το προγραμματιστικό μοντέλο της Cuda βασίζεται στης παρακάτω έννοιες, που έχουν πολλές ομοιότητες με την GPGPU

- **CUDA Kernels Δ**ηλαδή το σύνολο εντολών που θα πρέπει να τρέξουν παράλληλα για κάθε κομμάτι των δεδομένων μας που πρέπει να επεξεργαστεί.
- **Thread** Κάθε εκτέλεση του kernel για κάθε κομμάτι δεδομένων (πχ ενός στοιχείου πίνακα) λέγεται Thread (νήμα)
- **Ιεραρχία Threads** Κάθε νήμα ομαδοποιήται σε ισομεγέθη ομάδες που λέγονται μπλοκ (**Blocks)** και κάθε μπλοκ ομαδοποιήται σε πλέγματα (**Grids**). Κάθε thread έχει το δικό του **Thread ID** που λειτουργεί σαν αύξων αριθμός και βασίζεται στα **Block ID** και **Grid ID**
- **Ιεραρχία μνήμης** Ολα τα νήματα έχουν πρόσβαση σε διαφορετικές μνήμες. Κάθε νήμα έχει την προσωπική του μνήμη **(Private Local Memory),** έχει πρόσβαση στην κοινή μνήμη για κάθε μπλοκ (**shared memory**) ενώ όλα τα νήματα ανεξάρτητα από το Block έχουν πρόσβαση στην οικουμενική μνήμη (**global memory**)

Η εφαρμογή μας

Η ύπαρξη της αρχιτεκτονικής CUDA είναι συνυφασμένη με την ανάγκη για μεγαλύτερη επεξεργαστική ισχύ, και για γρηγορότερους υπολογισμούς. Συνεπώς δημιουργείται και η ανάγκη για ένα πρόγραμμα που να μπορεί

να μετράει την επίδοση αυτή των καρτών γραφικών. Ένα τέτοιο μετροπρόγραμμα (benchmark) θα πρέπει να δύναται να τρέχει κατά προτίμηση σε όλες τις κάρτες που υποστηρίζουν τις διάφορες εκδόσεις της CUDA και να μπορεί να δίνει αποτελέσματα που να αντιστοιχούν με τις πραγματικές επιδόσεις της κάρτας, σε εφαρμογές καθημερινής χρήσης.

- Για να καλυφθεί το κενό που περιγράψαμε πιο πάνω, δημιουργήσαμε ένα μετροπρόγραμμα βασισμένο σε CUDA που εκτελεί μερικές από τις πιο συνηθισμένες ρουτίνες που μπορεί να υπάρχουν σε ένα πρόγραμμα CUDA. Το πρόγραμμα αυτό χωρίζεται σε 2 διακριτά κομμάτια. Μία console εφαρμογή γραμμένη σε C που περιέχει τις απαραίτητες βιβλιοθήκες για CUDA , και τρέχει τους υπολογισμούς στην κάρτα γραφικών, και ένα γραφικό περιβάλλον που κάνει την εκτέλεση της console εφαρμογής πιο εύχρηστη, ενώ προσφέρει και παραπάνω λειτουργίες όπως ιστορικό των μετρήσεων και πληροφορίες για τον υπολογιστή.

Η Console εφαρμογή

Η εφαρμογή κονσόλας εκτελεί 3 ήδη υπολογισμών.

- **Μέτρηση ταχύτητας μνήμης** όπου υπολογίζεται το bandwidth κατά την μεταφορά δεδομένων σε 3 διαφορετικές περιπτώσεις
  1. **Από τον υπολογιστή στην κάρτα γραφικών (Host to Device)**
  2. **Εσωτερικά στην κάρτα γραφικών (Device to Device)**
  3. **Από την κάρτα γραφικών στον υπολογιστή (Device to Host)**
- **Πολλαπλασιασμός δισδιάστατων πινάκων (matrix mul).** 2 ισομεγέθεις πίνακας πολλαπλασιάζονται μεταξύ τους , και το αποτέλεσμα αποθηκεύεται σε έναν τρίτο πίνακα. Είναι ίσως ο ποιο συνηθισμένος τύπος υπολογισμού που υπάρχει σε προγράμματα CUDA
- **Έλεγχος Οπτικής Επαφής.** Αρχικοποιήται ένα heightmap 200.000 στοιχείων και ελέγχεται η οπτική επαφή του πρώτου στοιχείου με το τελευταίο. Το αποτέλεσμα είναι ένας πίνακας 200.000 στοιχείων όπου αποθηκεύονται με 1 όσα στοιχεία του heightmap κόβουν την οπτική επαφή των 2 σημείων και με 0 οσα στοιχεία δεν διακόπτουν την οπτική επαφή.

Τα αποτελέσματα των μετρήσεων αυτών αποθηκεύονται σε ένα XML αρχείο το οποίο μετά το διαχειρίζεται η εφαρμογή του Γραφικού Περιβάλλοντος.

Το Γραφικό Περιβάλλον

Το Γραφικό Περιβάλλον φροντίζει για την ευκολότερη απεικόνιση των δεδομένων του Benchmark. Είναι γραμμένη σε Visual Basic .Net και προσφέρει στον χρήστη τις εξής ευκολίες

- **Εύκολη εκτέλεση της εφαρμογής κονσόλας, χωρίς να χρειάζεται ο χρήστης να περνάει ορίσματα και παραμέτρους.**
- **Οπτικοποίηση των αποτελεσμάτων.** Ο χρήστης μπορεί να δει σε καθαρή μορφή και σε παραθυρικό περιβάλλον τα αποτελέσματα από την εκτέλεση των μετρήσεων, χωρίς να χρειάζεται να έρχεται σε επαφή με την δύσχρηστη μορφή της εφαρμογής κονσόλας. Του παρέχονται επίσης και παραπάνω πληροφορίες για τον υπολογιστή που τρέχει το πρόγραμμα, όπως Μέγεθος μνήμης, Τύπος λειτουργικού κ.α.
- **Ιστορικό των αποτελεσμάτων.** Η εφαρμογή κρατάει ιστορικό με όλες τις προηγούμενες εκτελέσεις ώστε ο χρήστης να μπορεί να συγκρίνει τα παλιά του αποτελέσματα με τα νεότερα, για να μπορεί πχ να βλέπει πόση παραπάνω ισχύ του δίνει ο υπερχρονισμός της κάρτας του, ή μια νεότερη κάρτα.

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

Εργαλεία

Για την εκπόνηση μας χρησιμοποιήσαμε τα εξής εργαλεία/προγράμματα

- **Windows Visual Studio 2012** Η πλατφόρμα ανάπτυξης εφαρμογών της Microsoft για περιβάλλον Windows. Παρέχει υποστήριξη σχεδόν σε όλες τις γλώσσες προγραμματισμού, είτε εγγενώς, είτε με την προσθήκη Plug-Ins. Την χρησιμοποιήσαμε για να γράψουμε το σύνολο του κώδικα μας.
- **Nvidia Cuda Toolkit and SDK** Το πακέτο της Nvidia με τα απολύτως αναγκαία για την ανάπτυξη εφαρμογών σε CUDA. Περιέχει τις απαραίτητες βιβλιοθήκες, CUDA ENABLED DRIVERS για την κάρτα γραφικών και πλήθος demo εφαρμογών που δείχνουν τους διάφορους τρόπου ανάπτυξης και τις διάφορες δυνατότητες της CUDA. Παρέχει επίσης τον Compiler για τον κώδικα CUDA (NVCC)
- **Nvidia Nsight(Visual Studio Edition)** Προγραμματιστικό βοήθημα για ανάπτυξη σε CUDA απο την Nvidia που παρέχει αρκετές επιλογές debugging, και φροντίζει για την εύκολη παραμετροποίηση του Visual Studio ώστε να υποστηρίζει Cuda C
- **XML Validator** Web εφαρμογή της w3schools που ελέγχει XML αρχεία για την ορθότητα τους και για την συμβατότητα τους με το πρότυπο XML 1.0
- **TechPowerup GPU-Z** Πρόγραμμα που προσφέρει πληθώρα πληροφοριών για κάθε κάρτα γραφικών στο σύστημα μας, όπως μέγεθος μνήμης, χρονισμοί, είδος μνήμης, αριθμό shaders και ROPS, θερμοκρασίες και πολλά άλλα. Το χρησιμοποιήσαμε για να δούμε τις ακριβείς προδιαγραφές κάθε κάρτας ώστε να μπορέσουμε να συγκρίνουμε τα αποτελέσματα των μετρήσεων μας, με της δυνατότητες της κάθε κάρτας

Αποτελέσματα/ Συμπεράσματα

Με την ενασχόληση μας με την παρούσα εργασία, αποκτήσαμε πολύ εξειδικευμένες γνώσεις, και πάνω στον σχεδιασμό των Καρτών Γραφικών, όπως πχ τα ήδη μνήμης, η ταχύτητά τους, και στο πως μπορούμε να σχεδιάζουμε καλύτερα προγράμματα με υψηλή παραλληλία και μεγάλη Αριθμητική Ένταση, έννοιες που δεν υπάρχουν στον κλασικό προγραμματισμό, άλλα αναπτύχθηκαν ταυτόχρονα με την ανάπτυξη πολυπυρηνικών επεξεργαστών.

Όπως αναφέραμε και στην αρχή αυτής της παρουσίασης, ο στόχος μας ήταν να ετοιμάσουμε μια ολοκληρωμένη λύση που να μετράει τις πραγματικές επιδόσεις μιας κάρτας γραφικών σε περιβάλλον CUDA. Αυτό σημαίνει ότι θα έπρεπε να προσφέρουμε μετρήσεις που να επηρεάζονται από τα χαρακτηριστικά της Κάρτας Γραφικών, και 2 διαφορετικές κάρτες να μην έχουν τα ίδια αποτελέσματα.

Απο μετρήσεις που κάναμε σε διάφορες κάρτες γραφικών είδαμε οτι τα δεδομένα που πήραμε ήταν τα αναμενόμενα .Για παράδειγμα το HosttoDevice, επηρεάζεται πιο πολύ από την διεπαφή με τον υπολογιστή. Δηλαδή το Bandwidthτου δίαυλού PCI-Express. Πιο συγκεκριμένα, για PCI-E 2.0 , κάθε 4 lanesμας δίνουν περίπου από 1Gb/sBandwidth. Λιγότερο επηρεάζει ο χρονισμός της μνήμης ,και το πλάτος του δίαυλου σε bit. Σχεδόν ακριβός το ίδιο επηρεάζεται και το DevicetoHost, αλλά εδώ έχουμε πιο εμφανές το bottlenecktου επεξεργαστή και της μνήμης του υπολογιστή.

Το DevicetoDevice, δηλαδή η αντιγραφή εσωτερικά της κάρτας, είναι απόλυτα συνυφασμένη με τον χρονισμό της μνήμης και με το πλάτος του δίαυλου (bus width). Και αυτή η συμπεριφορά είναι αναμενόμενη.

Το matrix multiplication, επηρεάζεται πιο πολύ από τον χρονισμό των shaders, και λιγότερο από την ταχύτητα της μνήμης, παρ όλα αυτά επηρεάζεται εμφανώς και από τα δυο.

Η μέτρηση για το Line of Sight από την άλλη, επηρεάζεται πιο πολύ από τον αριθμό των Shaders που υπάρχουν, και από την ταχύτητα της μνήμης και του δίαυλου PCI-E, και λιγότερο από τον χρονισμό των Shaderskαι της μνήμης.

Και οι 3 μετρήσεις μαζί, ελέγχουν κάθε χαρακτηριστικό της Κάρτας Γραφικών , και προσφέρουν μια ολοκληρωμένη μέτρηση των επιδόσεων. Τα αποτελέσματα συμφωνούν απόλυτα με τις προσδοκίες μας, και το

πρόγραμμα μας δίνει πια μια ξεκάθαρη εικόνα της απόδοσης της κάθε κάρτας.

Βιβλιογραφία

[1] Nvidia Cuda Getting Started Guide
http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/index.html

[2] Nvidia Cuda Programming Guide
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[3] Nvidia Cuda Best Practices Guide
http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[4] Nvidia Cuda Runtime API http://docs.nvidia.com/cuda/cuda-runtime-api/index.html

[5] Unified Memory Reports for CUDA 6
http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/

[6] Microsoft Visual Studio 2012 Product Guide
http://download.microsoft.com/download/C/D/3/CD39BB69-35CC-458A-B4EB-2F928B58FB4B/visual-studio-2012/Visual-Studio-2012-Product-Guide.pdf

[7] W3C Extensive Markup Language XML
http://www.w3.org/XML/#intro

[8] W3 Schools XML Tutorial http://www.w3schools.com/xml/

[9] W3 Schools XML Validator
http://www.w3schools.com/xml/xml_validator.asp

[10] Nsight Visual Studio Edition User Guide
http://docs.nvidia.com/gameworks/index.html#developertools/desktop/nvidia_nsight.htm

ΠΑΡΑΡΤΗΜΑ 2

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΦΗΣ ΓΙΑ ΕΚΤΙΜΗΣΗ ΕΠΙΔΟΣΕΩΝ ΤΗΣ ΚΑΡΤΑΣ ΓΡΑΦΙΚΩΝ ΣΕ ΠΕΡΙΒΑΛΟΝ CUDA

Ιωάννης Καπίρης(686)-Μαργαρίτης Αθανάσιος(737)

Επιβλέπων Καθηγητής : Γ. Κορνάρος

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

ΑΤΕΙ ΚΡΗΤΗΣ

ΕΙΣΑΓΩΓΗ

- Νόμος του Moore

"The number of transistors on an integrated

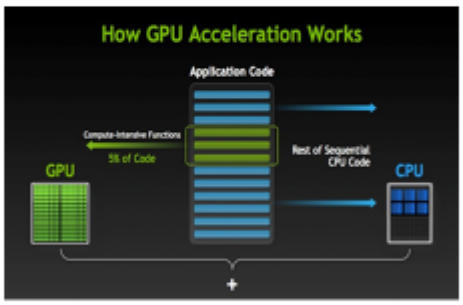
Circuit doubles every two years."

– Gordon E. Moore

- Οι Επεξεργαστές δεν μπορούν πια να αυξήσουν την συχνότητα τους ώστε να ακολουθήσουν τον νόμο του Moore
- Αποτέλεσμα: Παράγοντες επεξεργαστές με περισσότερους πυρήνες.
- Συνεπώς τα προγράμματα μας γίνονται παράλληλα αντί για γραμμικά
- Έρχονται στο προσκήνιο νέες τεχνολογίες όπως GPGPU

GPGPU

- General Purpose Computing on Graphics Processing Units η αλλιώς «Υπολογισμοί Γενικού Σκοπού σε Επεξεργαστές Γραφικών»
- Τεχνολογία που επιτρέπει σε κώδικα που κανονικά θα έτρεχε σε CPU να τρέχει σε κάρτα γραφικών (GPU)
- Οι κάρτες γραφικών αποτελούνται από πάρα πολλούς επεξεργαστές που είναι βελτιστοποιημένοι για συγκεκριμένες λειτουργίες (πχ pixel shading)



GPGPU

ΠΛΕΟΝΕΚΤΗΜΑΤΑ

- Πολύ μεγάλη επιτάχυνσή της εκτέλεσης του κώδικα, ακόμα και της τάξης του 100x

ΜΕΙΟΝΕΚΤΗΜΑΤΑ

- Ο κώδικας για να μπορεί να τρέξει σε GPU θα πρέπει να έχει υψηλή παραλληλία
- Τα δεδομένα θα πρέπει να μπορούν να επεξεργαστούν ανεξάρτητα μεταξύ τους
- Τα δεδομένα και οι επεξεργασία τους θα πρέπει να μοιάζουν με δεδομένα που συνήθως επεξεργάζεται η κάρτα γραφικών

NVidia CUDA

Λύση σε αυτό το πρόβλημα δίνει η τεχνολογία CUDA της εταιρία Nvidia

- Ανακοινώθηκε τον Νοέμβριο του 2006
- Επιτρέπει πολύ μεγαλύτερη ευελιξία στους υπολογισμούς απ' ότι οι κλασικές λύσεις GPGPU

ΤΙ ΕΙΝΑΙ Η CUDA

- ΠΛΑΤΦΟΡΜΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΠΟΥ ΔΙΝΕΙ ΤΗΝ ΔΥΝΑΤΟΤΗΤΑ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ ΝΑ ΓΡΑΦΕΙ ΚΩΔΙΚΑ ΥΨΗΛΟΥ ΕΠΙΠΕΔΟΥ ΠΟΥ ΝΑ ΤΡΕΧΕΙ ΣΤΗΝ ΚΑΡΤΑ ΓΡΑΦΙΚΩΝ

NVidia CUDA

ΠΡΟΤΕΡΗΜΑΤΑ

- Προσφέρει προγραμματισμό υψηλού επιπέδου με πολύ μεγαλύτερη ευελιξία στα δεδομένα που μπορούν να επεξεργαστούν, αλλά και στην επεξεργασία που μπορεί να γίνει
- Ακολουθεί τις βασικές αρχές GPGPU , άρα τα προγράμματα που είναι φτιαγμένα για GPGPU μπορούν εύκολα να μετατραπούν για CUDA
- Υποστηρίζεται από πολλές και χαμηλού κόστους κάρτες γραφικών που υπάρχουν ήδη στο εμπόριο

NVidia CUDA

ΠΡΟΤΕΡΗΜΑΤΑ

- Είναι εύκολη στην χρήση, αφού ουσιαστικά πρόκειται για επέκταση της C (αν και σιγά σιγά γίνεται μεταφορά και σε άλλες γλώσσες)

Πρόβλημα!

Υπάρχουν πολλές κάρτες που υποστηρίζουν CUDA με πολύ διαφορετικά χαρακτηριστικά η κάθε μία

Δεν υπάρχει εύκολος τρόπος να μετρήσουμε τις επιδόσεις μίας κάρτας γραφικών σε περιβάλλον CUDA.

Γι αυτό αναγκαστήκαμε να φτιάξουμε μια εφαρμογή που να μετράει τις πραγματικές επιδόσεις των καρτών αυτόν

Η εφαρμογή μας

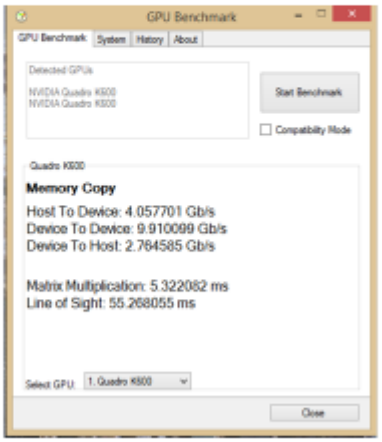
- Δημιουργήθηκε για να μετράει τις πραγματικές επιδόσεις των καρτών γραφικών σε περιβάλλον CUDA
- Προσπαθήσαμε ώστε να υπολογίσουμε και να μετρήσουμε όλα τα χαρακτηριστικά μιας κάρτας όπως ταχύτητα μνήμης, ταχύτητα επεξεργαστών (shaders) και αριθμό επεξεργαστών
- Χωρίζεται σε 2 κομμάτια. Το γραφικό περιβάλλον γραμμένο σε VB.NET και σε μια εφαρμογή κονσόλας γραμμένη σε CUDA C
- Η επικοινωνία μεταξύ τους γίνεται μέσω XML αρχείου

Η εφαρμογή μας

ΟΙ ΜΕΤΡΗΣΕΙΣ

Για να βγάλουμε το αποτέλεσμα κάνουμε 3 είδη μετρήσεων

- Μέτρηση ταχύτητας μνήμης
  1. Από τον υπολογιστή στην Κάρτα (Host to Device)
  2. Εσωτερικά στην Κάρτα (Device to Device)
  3. Από την Κάρτα στον υπολογιστή
- Πολλαπλασιασμό 2 πινάκων 200.000 στοιχείων (Matrix Multiplication)
- Έλεγχος Ορατότητας σε Height map 200.000 στοιχείων



Η εφαρμογή μας

ΤΡΟΠΟΣ ΧΡΗΣΗΣ

Όταν τρέχουμε την εφαρμογή βλέπουμε την αρχική εικόνα της.

- Αυτόματα εμφανίζονται όλες οι διαθέσιμες κάρτες γραφικών
- Κάνοντας κλικ στο κουμπί **Start Benchmark** τρέχει αυτόματα η Console εφαρμογή.
- Όταν τελειώσει και κλείσει το παράθυρο της, το γραφικό περιβάλλον εμφανίζει τα αποτελέσματα τα οποία τα παίρνει από το XML που παράχθηκε από την εφαρμογή κονσόλας
- Το κουμπί **Select GPU** μας επιτρέπει να δούμε τα αποτελέσματα από την δεύτερη μας κάρτα(αν υπάρχει)

Η εφαρμογή μας

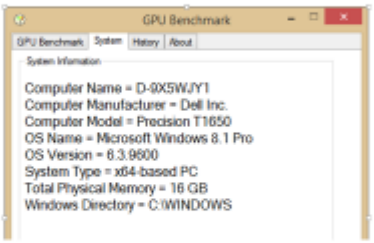
ΤΡΟΠΟΣ ΧΡΗΣΗΣ

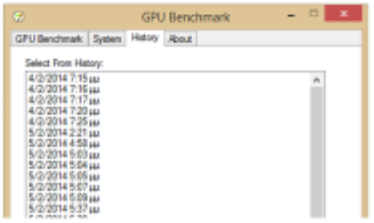
Καρτέλα System

- Στην καρτέλα System βλέπουμε πληροφορίες για το σύστημα μας
- Τις πληροφορίες αυτές της παίρνουμε απ' το WMI του συστήματος

Καρτέλα History

- Στην καρτέλα History μπορούμε να δούμε λίστα με της προηγούμενες μετρήσεις
- Κάνοντας διπλό κλικ σε οποιαδήποτε ημερομηνία, αυτόματα φορτώνονται τα αποτελέσματα από εκείνη την μέτρηση τα οποία μπορούμε να τα δούμε στην καρτέλα GPU Benchmark





Η εφαρμογή μας

ΤΡΟΠΟΣ ΧΡΗΣΗΣ

Καρτέλα About

- Στην καρτέλα About βλέπουμε γενικές πληροφορίες για το πρόγραμμα

Το πρόγραμμα μπορούμε οποιαδήποτε στιγμή να το κλείσουμε πατώντας στο κουμπί **Close**

Μετρήσεις/Αποτελέσματα

- Τρέξαμε το benchmark μας σε μία πλειάδα καρτών γραφικών nvidia
- Επιβεβαιώσαμε τα αποτελέσματα με τα χαρακτηριστικά κάθε κάρτας
- Κάρτες με γρηγορότερη διεπαφή και γρηγορότερη μνήμη, αποδίδουν καλύτερα στις μετρήσεις που απαιτούν memory bandwidth
- Ομοίως κάρτες με γρηγορότερους/περισσότερους πυρήνες αποδίδουν υψηλότερα στις μετρήσεις που θέλουν επεξεργαστική ισχύ

GPU	Αριθμός Shaders	GPU Clock	Memory Clock	Bus Interface	PCI Express Lanes	Bus Width	Theoretical Memory Bandwidth	RESULTS	Host To Device (GB/s)	Device to Device (GB/s)	Device to Host (GB/s)	Matrix Multiplication (ms)	Line of Sight(ms)
Quadro K600 (1)	192	876	891	PCI-E 2.0 x1	16	128	28.5		4.03	5.89	2.83	5.06	54.95
Quadro K600 (2)	192	876	891	PCI-E 2.0 x1	4	128	28.5		4.24	5.49	4.22	10.03	476.77
Geforce GT740M	384	980	900	PCI-E 2.0 x1	8	64	14.4		2.24	5.9	2.24	2.06	133.2
Geforce 9600GT	64	650	900	PCI-E 2.0 x1	16	256	57.6		4.99	44.04	4.52	4.2	102.42
Geforce GT440	96	700	800	PCI-E 2.0 x1	16	128	28.5		2.85	10.1	1.5	4.6	52
Geforce GT620	48	810	897	PCI-E 2.0 x1	16	64	14.4		2.9	5.77	3.52	7.8	58.72
Geforce GTX760	1152	1020	1502	PCI-E 3.0 x1	16	256	192.3		3.05	69.72	2.7	0.6	53.22
Quadro K2000	384	954	1000	PCI-E 2.0 x1	16	128	64		2.3	23.45	4.69	2.4	89.26
Quadro K4000	768	811	1404	PCI-E 2.0 x1	16	192	134.8		2.98	42.8	2.05	2.46	118

Εργαλεία

Για την εκπόνηση της εργασίας αυτής χρησιμοποιήσαμε τα παρακάτω εργαλεία

- Microsoft Visual Studio 2012
- NVidia Cuda Toolkit & NVidia SDK
- NVidia Nsight Visual Studio Edition
- W3schools XML Validator
- TechPowerup GPU-Z

Ευχαριστούμε για την Προσοχή σας

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΕΚΤΙΜΙΣΗ ΕΠΙΔΟΣΕΩΝ ΚΑΡΤΑΣ ΓΡΑΦΙΚΩΝ ΣΕ ΠΕΡΙΒΑΛΛΟΝ CUDA

Σχολή Τεχνολογικών Εφαρμογών ΑΤΕΙ Ηρακλείου

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Συγγραφή:

Ιωάννης Καπίρης

Αθανάσιος Μαργαρίτης

Επιβλέπων Καθηγητής: Γ Κορνάρος

ΠΑΡΑΡΤΗΜΑ3

Πηγαίος Κώδικας των 2 Εφαρμογών

GUI (VB.NET)

```
Imports System.Threading
Imports System.IO
Imports System.IO.File
Imports System.Globalization
Imports System.Management

PublicClassbenchmark

Dim startDate AsNewDateTime(1970, 1, 1)
Dim argument AsInteger
Dim result_xml AsString
Dim result_gpu1 AsNewArrayList()
Dim result_gpu2 AsNewArrayList()
Dim history AsNewArrayList()
Dim total_gpus AsInteger

PrivateSub benchmark_Load(sender AsObject, e AsEventArgs) HandlesMyBase.Load

        view_history()
getGPUfromWMI()


Dim objWMI AsNewclsWMI()
With objWMI
            computer_name.Text = "Computer Name = "& .ComputerName
            computer_manufacturer.Text    =    "Computer    Manufacturer    =    "&
.Manufacturer
            computer_model.Text = "Computer Model = "& .Model
Dim parts AsString() = .OsName.Split(NewChar() {"|"c})
            os_name.Text = "OS Name = "&parts(0)
            os_version.Text = "OS Version = "& .OSVersion
            system_type.Text = "System Type = "& .SystemType
            total_memory.Text      =      "Total      Physical      Memory      =
"&Math.Round((((.TotalPhysicalMemory / 1024) / 1024) / 1024)) &" GB"
            windows_directory.Text = "Windows Directory = "& .WindowsDirectory
EndWith

EndSub


PrivateSubgetGPUfromWMI()
Dim GraphicsCardName = String.Empty
Try
Dim wmiSelect AsNewManagementObjectSearcher _
```

```
                ("root\CIMV2", "SELECT * FROM Win32_VideoController")
Me.TextBox1.Text = "Detected GPUs"
Me.TextBox1.Text += Environment.NewLine
Me.TextBox1.Text += Environment.NewLine
ForEach WmiResults AsManagementObjectInwmiSelect.Get()
                GraphicsCardName = WmiResults.GetPropertyValue("Name").ToString
Me.TextBox1.Text += GraphicsCardName
Me.TextBox1.Text += Environment.NewLine
Next

Catch ex AsManagementException
MessageBox.Show(ex.Message)
EndTry

EndSub

PrivateSub run_Click(sender AsObject, e AsEventArgs) Handles run.Click
If cmode_box.Checked = FalseThen

            run.Enabled = False
            select_gpu.Enabled = False
            history_list.Enabled = False

Try

argument = (DateTime.Now - startDate).TotalSeconds
Process.Start("HelloMyCuda.exe", argument.ToString + ".xml")
                result_xml = argument.ToString + ".xml"

                xmlopen.Enabled = True

Catch ex AsException
MsgBox(ex.Message)
                run.Enabled = True
                select_gpu.Enabled = True
                history_list.Enabled = True
EndTry
Else
MsgBox("Running In Compatibility Mode for NON CUDA Systems")

            run.Enabled = False
            select_gpu.Enabled = False
            history_list.Enabled = False

Try

argument = (DateTime.Now - startDate).TotalSeconds
Process.Start("compmode.exe", argument.ToString + ".xml")
                result_xml = argument.ToString + ".xml"
```

```
                xmlopen.Enabled = True

Catch ex AsException
MsgBox(ex.Message)
                run.Enabled = True
                select_gpu.Enabled = True
                history_list.Enabled = True
EndTry


EndIf


EndSub

PrivateSub xmlopen_Tick(sender AsObject, e AsEventArgs) Handles xmlopen.Tick

Try

' Load XML
Dim doc = XDocument.Load(result_xml)

            xmlopen.Enabled = False

readxml()

Catch ex AsException

EndTry

EndSub

PrivateSub view_history()

Try

Dim directory AsNewIO.DirectoryInfo(Environment.CurrentDirectory)
Dim allfiles AsIO.FileInfo() = directory.GetFiles("*.*")
Dim file_info As IO.FileInfo

ForEach file_info In allfiles

If file_info.Extension = ".xml"Then
If file_info.Name <>"Benchmark.xml"Then
                        history_list.Items.Add(file_info.LastWriteTime)
history.Add(file_info.Name)
EndIf
EndIf

Next
```

```
Catch ex AsException
MsgBox(ex.Message)
EndTry

EndSub

PrivateSubreadxml()

        total_gpus = 0
        result_gpu1.Clear()
        result_gpu2.Clear()
        select_gpu.Items.Clear()

Try

Dim doc = XDocument.Load(result_xml)

Dim read_gpu AsInteger

' Count gpus and enable counters
Dim gpus = From v Indoc.Element("benchmark").Elements("gpu")
Select gpuid = v.Element("gpuid").Value
ForEach gpu In gpus
                total_gpus += 1
Next

' GPU Names to Panels
Dim gpuN = From v Indoc.Element("benchmark").Elements("gpu")
Select gpuName = v.Element("gpuName").Value
            read_gpu = 1
ForEach gpu In gpuN

If read_gpu = 1 Then
                    result_gpu1.Add(gpu)
                    select_gpu.Items.Add("1. " + gpu)
EndIf

If read_gpu = 2 Then
                    result_gpu2.Add(gpu)
                    select_gpu.Items.Add("2. " + gpu)
EndIf

                read_gpu = 2

Next

' h2d
Dim h2dv = From v Indoc.Element("benchmark").Elements("gpu")
Select h2d = v.Element("h2d").Value
```

```
            read_gpu = 1
ForEach gpu In h2dv
If read_gpu = 1 Then result_gpu1.Add(gpu)
If read_gpu = 2 Then result_gpu2.Add(gpu)
                read_gpu = 2
Next

' d2d
Dim d2dv = From v Indoc.Element("benchmark").Elements("gpu")
Select d2d = v.Element("d2d").Value
            read_gpu = 1
ForEach gpu In d2dv
If read_gpu = 1 Then result_gpu1.Add(gpu)
If read_gpu = 2 Then result_gpu2.Add(gpu)
                read_gpu = 2
Next

' d2h
Dim d2hv = From v Indoc.Element("benchmark").Elements("gpu")
Select d2h = v.Element("d2h").Value
            read_gpu = 1
ForEach gpu In d2hv
If read_gpu = 1 Then result_gpu1.Add(gpu)
If read_gpu = 2 Then result_gpu2.Add(gpu)
                read_gpu = 2
Next

' mmul
Dim mmulv = From v Indoc.Element("benchmark").Elements("gpu")
Select mmul = v.Element("mmul").Value
            read_gpu = 1
ForEach gpu In mmulv
If read_gpu = 1 Then result_gpu1.Add(gpu)
If read_gpu = 2 Then result_gpu2.Add(gpu)
                read_gpu = 2
Next

' los
Dim losv = From v Indoc.Element("benchmark").Elements("gpu")
Select los = v.Element("los").Value
            read_gpu = 1
ForEach gpu In losv
If read_gpu = 1 Then result_gpu1.Add(gpu)
If read_gpu = 2 Then result_gpu2.Add(gpu)
                read_gpu = 2
Next

Catch ex AsException
MsgBox(ex.Message)
EndTry
```

```
        run.Enabled = True
        select_gpu.Enabled = True
        history_list.Enabled = True

        history_list.Items.Clear()
        view_history()

        select_gpu.SelectedIndex = 0

EndSub

PrivateSub  select_gpu_SelectedIndexChanged(sender  AsObject,  e  AsEventArgs)
Handles select_gpu.SelectedIndexChanged

If select_gpu.SelectedIndex = 0 Then
readgpu1()
EndIf

If select_gpu.SelectedIndex = 1 Then
readgpu2()
EndIf

EndSub

PrivateSubreadgpu1()

        gpu_board.Text = result_gpu1.Item(0)
        h2d_txt.Text = "Host To Device: " + result_gpu1.Item(1) + " Gb/s"
        d2d_txt.Text = "Device To Device: " + result_gpu1.Item(2) + " Gb/s"
        d2h_txt.Text = "Device To Host: " + result_gpu1.Item(3) + " Gb/s"
        mmul_txt.Text = "Matrix Multiplication: " + result_gpu1.Item(4) + " ms"
        los_txt.Text = "Line of Sight: " + result_gpu1.Item(5) + " ms"

EndSub

PrivateSubreadgpu2()

        gpu_board.Text = result_gpu2.Item(0)
        h2d_txt.Text = "Host To Device: " + result_gpu2.Item(1) + " Gb/s"
        d2d_txt.Text = "Device To Device: " + result_gpu2.Item(2) + " Gb/s"
        d2h_txt.Text = "Device To Host: " + result_gpu2.Item(3) + " Gb/s"
        mmul_txt.Text = "Matrix Multiplication: " + result_gpu2.Item(4) + " ms"
        los_txt.Text = "Line of Sight: " + result_gpu2.Item(5) + " ms"

EndSub

PrivateSub   close_app_Click(sender   AsObject,   e   AsEventArgs)   Handles
close_app.Click
Application.Exit()
```

```
EndSub

PrivateSub history_list_DoubleClick(sender AsObject, e AsEventArgs) Handles
history_list.DoubleClick
Try
            result_xml = history.Item(history_list.SelectedIndex)
readxml()
            TabControl.SelectedTab = TabPage1
Catch ex AsException

EndTry

EndSub




EndClass
```

Console (Cuda C)

```
#include<stdio.h>
#include<cuda.h>
#include<cuda_runtime.h>
#include<time.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include"cuda_runtime.h"
#include"device_launch_parameters.h"
#include"helper_functions.h"// helper for shared functions common to CUDA SDK samples
#include"helper_cuda.h"// helper functions for CUDA error checking and initialization
#include"helper_timer.h"
#include"helper_cuda_drvapi.h"
#include"drvapi_error_string.h"
#include<cufft.h>
#include<cuda.h>
#include<memory>
#include<iostream>
#include<cassert>


#defineCOPYSIZE 134217728
#defineLOSITTERATIONS 10
#defineMEMCOPYITTERATIONS 10
#defineMATMULITTERATIONS 10

//data structures creation

typedeffloat2Complex;

typedefstruct {
float d2h;
float d2d;
float h2d;
  } BWidth;

BWidthmemcopy(){

float *data_h;
float *data_d;
float *datacopy_h;
float *datacopy_d;
float bandwidth1;
float bandwidth2;
float bandwidth3;
StopWatchInterface *timer1 = NULL;
StopWatchInterface *timer2 = NULL;
StopWatchInterface *timer3 = NULL;
float elapsedTimeInMs = 0.0f;
longint reiterations,i;
int tablesize;
BWidth bwidth;
cudaEvent_t start, stop;
```

```
//512MB= 536870912bytes
//64MB =  COPYSIZE bytes
//copy x data in t time
//result= x/t MB/s

reiterations = COPYSIZE/sizeof(float);

//populating data to be copied...
//calculating data size
	tablesize=reiterations*sizeof(float);
//allocating data on HOST
	data_h=(float *)malloc(tablesize);
//initializing data on HOST
	for (i=0;i<reiterations;i++)
		{
			data_h[i]=1.1f ;
		}

//alocating data on device
	cudaMalloc((void **) &data_d, tablesize);
//creating and starting timer
//	cutilCheckError(cutCreateTimer(&timer1));
//	cutilCheckError(cutStartTimer(timer1));
//is replaced with (for v5.5 compliance)


	//creating timer and event
sdkCreateTimer(&timer1);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
	//starting timer
	sdkStartTimer(&timer1);
	checkCudaErrors(cudaEventRecord(start, 0));
//copy data from host to device
	cudaMemcpy(data_d, data_h, tablesize, cudaMemcpyHostToDevice);
	checkCudaErrors(cudaEventRecord(stop, 0));
	checkCudaErrors(cudaDeviceSynchronize());

//stoping timer,getting data,  printing and deleting timer
	sdkStopTimer(&timer1);
	checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));
	sdkDeleteTimer(&timer1);

//calculating throughput
	bandwidth1=(COPYSIZE/1024/1024)/ (elapsedTimeInMs/1000); //(gbytes/s)

//alocating copy data on device
	cudaMalloc((void **) &datacopy_d, tablesize);
//creating and starting timer
	//creating timer and event
sdkCreateTimer(&timer2);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
	//starting timer
	sdkStartTimer(&timer2);
```

```
	checkCudaErrors(cudaEventRecord(start, 0));
//copy data from host to host
	cudaMemcpy(datacopy_d, data_d, tablesize, cudaMemcpyDeviceToDevice);
	checkCudaErrors(cudaEventRecord(stop, 0));
	//cutilSafeCall( cudaThreadSynchronize() );
checkCudaErrors(cudaDeviceSynchronize());

//stoping timer,getting data,  printing and deleting timer
//stoping timer,getting data,  printing and deleting timer
	sdkStopTimer(&timer2);
	checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));
	sdkDeleteTimer(&timer2);

//calculating throughput
	bandwidth2=(COPYSIZE/1024/1024)/ (elapsedTimeInMs/1000); //(mbytes/s)
//alocating copy data on host
	datacopy_h=(float *)malloc(tablesize);
//creating and starting timer
                    //creating timer and event
sdkCreateTimer(&timer3);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));

	//starting timer
	sdkStartTimer(&timer3);
	checkCudaErrors(cudaEventRecord(start, 0));
//copy data from host to host
	cudaMemcpy(datacopy_h, datacopy_d, tablesize, cudaMemcpyDeviceToHost);
	checkCudaErrors(cudaEventRecord(stop, 0));
checkCudaErrors(cudaDeviceSynchronize());

//stoping timer,getting data,  printing and deleting timer
	sdkStopTimer(&timer3);
	checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));
	sdkDeleteTimer(&timer3);


//calculating throughput
	bandwidth3=(COPYSIZE/1024/1024)/ (elapsedTimeInMs/1000); //(gbytes/s)

	bwidth.h2d=bandwidth1;
	bwidth.d2d=bandwidth2;
	bwidth.d2h=bandwidth3;


free(data_h);
cudaFree(data_d);
free(datacopy_h);
cudaFree(datacopy_d);

return(bwidth);
}

__global__void mat_mul2(float *a, float *b, float *ab){
int idx = blockIdx.x * blockDim.x + threadIdx.x;
```

```
ab[idx]=a[idx]*b[idx];
}

float matrixmul(){

float *matrixhost_a;
float *matrixhost_b;
float *matrixhost_ab;
float *matrixdevice_a;
float *matrixdevice_b;
float *matrixdevice_ab;
unsignedint matrixlen=200000;
unsignedint matrixsize;
unsignedint i;
float GPU_time = 0.0f;
StopWatchInterface *timer = NULL;
cudaEvent_t start, stop;

//alocate host matrices
//calculate size of matrix
matrixsize=matrixlen*sizeof(float);
matrixhost_a=(float *)malloc(matrixsize);
matrixhost_b=(float *)malloc(matrixsize);
matrixhost_ab=(float *)malloc(matrixsize);

//alocate device matrices
cudaMalloc((void **) &matrixdevice_a, matrixsize);
cudaMalloc((void **) &matrixdevice_b, matrixsize);
cudaMalloc((void **) &matrixdevice_ab, matrixsize);

//initiate host matrices
for(i=0;i<matrixlen;i++){
       matrixhost_a[i]=2*i;
       matrixhost_b[i]=i/10;
       }
// copy matrices from host to device
cudaMemcpy(matrixdevice_a, matrixhost_a,matrixsize,cudaMemcpyHostToDevice);
cudaMemcpy(matrixdevice_b, matrixhost_b,matrixsize,cudaMemcpyHostToDevice);
//start timer
sdkCreateTimer(&timer);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
//call mat_mul2
       int block_size = 4;
int n_blocks = matrixlen/block_size + (matrixlen%block_size == 0 ? 0:1);
    mat_mul2<<< n_blocks, block_size >>> (matrixdevice_a,matrixdevice_b,matrixdevice_ab );
//stop timer retrieve time and delete timer
       checkCudaErrors(cudaEventRecord(stop, 0));
       checkCudaErrors(cudaDeviceSynchronize());
       sdkStopTimer(&timer);
checkCudaErrors(cudaEventElapsedTime(&GPU_time, start, stop));
sdkDeleteTimer(&timer);
//  retrieve multiplied matrix from device");
cudaMemcpy(matrixhost_ab,matrixdevice_ab,matrixsize,cudaMemcpyDeviceToHost);
```

```
//free memories and return value
cudaFree(matrixdevice_a);
cudaFree(matrixdevice_b);
cudaFree(matrixdevice_ab);

free(matrixhost_a);
free(matrixhost_b);
free(matrixhosPtt_ab);
return(GPU_time);
}


__global__void create_hmap(float *a,intdistance){
int idx= blockIdx.x * blockDim.x + threadIdx.x;
a[idx]=(sinf(idx)+1)*40;
}

__global__void calc_los(float *a,int *dvmap, intTotaldistance, floattheta,floatobjh)
{
float ah,b;
int idx = blockIdx.x * blockDim.x + threadIdx.x;
ah=((Totaldistance-idx)*theta)+objh;
   b=a[idx];
if (ah>b){dvmap[idx]=1;}
elsedvmap[idx]=0;
 }

float los(){
float *hmap_h;
float *hmap_d;
int  *vlmap;
int *dvmap;
int mapsize=200000; //ari8mos deigmatwn
float theta;
int tablesize;
int resultsize;
float GPU_time;
StopWatchInterface *timer = NULL;
cudaEvent_t start, stop;


//creating and starting timer
       sdkCreateTimer(&timer);
checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&stop));
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));

tablesize=mapsize*sizeof(float);
resultsize=mapsize*sizeof(int);
//create table
//run 200.000 sin table using cuda createhmap
hmap_h=(float *)malloc(tablesize);
//alocate heightmap space on device
cudaMalloc((void **) &hmap_d,tablesize);
//call height map creation routing on device
```

```
int block_size = 4;
int n_blocks = mapsize/block_size + (mapsize%block_size == 0 ? 0:1);
create_hmap <<< n_blocks, block_size >>> (hmap_d,mapsize);
//copy result height map to host for refference
cudaMemcpy(hmap_h,hmap_d,tablesize,cudaMemcpyDeviceToHost); //needed?

checkCudaErrors(cudaDeviceSynchronize());
//allocate result map on device
cudaMalloc((void **) &dvmap,resultsize);
//allocate result map on host
vlmap=(int *)malloc(resultsize);
//calculate theta
theta=(hmap_h[0]-hmap_h[mapsize-1])/mapsize;
//call los detection routine
 block_size = 4;
 n_blocks = mapsize/block_size + (mapsize%block_size == 0 ? 0:1);
 calc_los <<< n_blocks, block_size >>> (hmap_d,dvmap,mapsize,theta,hmap_h[mapsize-1]);
//recover result map from device
cudaMemcpy(vlmap,dvmap,resultsize,cudaMemcpyDeviceToHost);


        checkCudaErrors(cudaEventRecord(stop, 0));
        checkCudaErrors(cudaDeviceSynchronize());
        sdkStopTimer(&timer);
        checkCudaErrors(cudaEventElapsedTime(&GPU_time, start, stop));


free(hmap_h);
cudaFree(hmap_d);
free(vlmap);
cudaFree(dvmap);

return(GPU_time);
}


int main(intargc, char **argv){
BWidth mmcpresult;
BWidth mmcptemp;
float matrixresult=0.0;
float losresult=0.0;
int i;
cudaError_t  status;
int dev=0;
int devicecount;
char deviceName[256];
CUresult error_id = cuInit(0);
FILE *myFile;

if(argc>1){
        myFile = fopen(argv[1], "wb+");
printf(argv[1]);
}
else {
myFile= fopen("result.xml","wb+");
printf("no arguments where given, using result.xml instead \n");
```

```
}
printf("Cuda Device Benchmark  \n");
fprintf(myFile,"<?xml version=\"1.0\"?>\n");
fprintf(myFile,"<benchmark>\n");

//get cuda device count
printf("Detecting Cuda Devices  \n");
cudaGetDeviceCount(&devicecount);
// Device Selection
printf(" %d devices detected  \n",devicecount);

//starting benchmark loop for every detected device
for (dev=0;dev<devicecount;dev++){
       printf("running benchmark on device %d \n",dev);
error_id = cuDeviceGetName(deviceName, 256, dev);
if (error_id != CUDA_SUCCESS)
        {
printf("cuDeviceGetName returned %d\n-> %s\n", (int)error_id, getCudaDrvErrorString(error_id));
printf("Result = FAIL\n");
exit(EXIT_FAILURE);
        }
       printf("\nDevice %d: \"%s\"\n", dev, deviceName);

//setting device
status = cudaSetDevice(dev);
if (status != cudaSuccess) {
printf ("!!!! Set Device error\n");
returnEXIT_FAILURE;
}

//starting result file
fprintf(myFile,"<gpu>\n");
fprintf(myFile,"<gpuid>%d</gpuid>\n",dev);
fprintf(myFile,"<gpuName>%s</gpuName>\n",deviceName);

//starting memory copy benchmark
printf("Initiating Memory Bandwidth Test\n");
//Initializing values
mmcpresult.d2h=0.0;
mmcpresult.d2d=0.0;
mmcpresult.h2d=0.0;

//Running memory copy MEMCOPYITERATIONS times
for(i=0;i<MEMCOPYITTERATIONS;i++){
mmcptemp=memcopy();
mmcpresult.d2h=mmcpresult.d2h+mmcptemp.d2h;
mmcpresult.d2d=mmcpresult.d2d+mmcptemp.d2d;
mmcpresult.h2d=mmcpresult.h2d+mmcptemp.h2d;
printf(".");
}

//extrapolating result
mmcpresult.d2h=mmcpresult.d2h/MEMCOPYITTERATIONS;
mmcpresult.d2d=mmcpresult.d2d/MEMCOPYITTERATIONS;
mmcpresult.h2d=mmcpresult.h2d/MEMCOPYITTERATIONS;
```

```
//Printing results to console and file
printf("\n Memory Bandwidth   \n\n Host to Device  Device to Device      Device to Host\n %f Gb/s
%f Gb/s         %fGb/s \n\n",mmcpresult.h2d/1024,mmcpresult.d2d/1024,mmcpresult.d2h/1024);
fprintf(myFile,"<h2d>%f</h2d>\n<d2d>%f</d2d>\n<d2h>%f</d2h>\n",mmcpresult.h2d/1024,mmcpresult.d2d/1
024,mmcpresult.d2h/1024);


//starting Matrix Multiplication Benchmark

printf(" Commencing Matrix Multiplication" );
//running the benchmark MATMULITTERATIONS times
for(i=0;i<MATMULITTERATIONS;i++){
printf(".");
matrixresult=matrixresult+matrixmul();
}
//printing results on file and console
printf("\n Matrix Multiplication time :   %f (ms) \n\n",matrixresult/MATMULITTERATIONS);
fprintf(myFile,"<mmul>%f</mmul>\n",matrixresult/MATMULITTERATIONS);

//starting Line of Sight Benchmark
printf("Initiating Line of Sight Benchmark \n");

//running benchmark LOSITTERATIONS times
for(i=0;i<LOSITTERATIONS;i++){
printf(".");
losresult=losresult+los();
}
//printing results on file and console
printf("\n\n Time to complete Line of Sight Detection  %f (ms)\n",losresult/LOSITTERATIONS);
fprintf(myFile,"<los>%f</los>\n",losresult/LOSITTERATIONS);

//close tags for GPU on XML
fprintf(myFile,"</gpu>\n");
}
//close tags for BENCHMARK on XML and close file
fprintf(myFile,"</benchmark>");
fclose(myFile);
//kill all stray cuda threads
cudaThreadExit();
       }
```