Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Λογισμικό & Προγραμματισμός Συστημάτων Υψηλής Επίδοσης
Εργασία Ακ. Έτους 2018-2019

Ονοματεπώνυμο	**ΑΜ**
Εμμανουήλ Κατεφίδης	6077
Παναγιώτης Σταυρινάκης	6217

Χαρακτηριστικά Κάρτας Γραφικών

Οι υλοποιήσεις των ερωτημάτων εκτελέστηκαν στην κάρτα γραφικών του εργαστηρίου(Tesla C2075), η οποία διαθέτει τα εξής χαρακτηριστικά:

```
Device 0: "Tesla C2075"
  CUDA Driver Version / Runtime Version          4.2 / 4.2
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (14) Multiprocessors x ( 32) CUDA Cores/MP:    448 CUDA Cores
  GPU Clock rate:                                1147 MHz (1.15 GHz)
  Memory Clock rate:                             1566 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and execution:                 Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Concurrent kernel execution:                   Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support enabled:                Yes
  Device is using TCC driver mode:               No
  Device supports Unified Addressing (UVA):      Yes
```

Compilation

Το compilation των αρχείων κώδικα του πυρήνα για όλα τα ερωτήματα έγινε με την εξής εντολή:

```
team09@tesla: ~
team09@tesla:~$ nvcc -arch=sm_20 -ccbin /usr/bin/gcc-4.8 arxeio.cu
```

Ερωτήματα

Για την υλοποίηση των ερωτημάτων σε γλώσσα προγραμματισμού CUDA εξετάστηκαν και υλοποιήθηκαν διάφοροι μέθοδοι αλγορίθμων με στόχο τον όσο αποδοτικότερο χρόνο είναι εφικτό να παράγουμε. Οι μέθοδοι που υλοποιήθηκαν ήταν οι εξής:

- Ένας naïve τρόπος εκτέλεσης ο οποίος καλεί κάθε φορά τα δεδομένα από την καθολική μνήμη
- Εκτέλεση με χρήση της εντολής atomicAdd()
- Εκτέλεση με χρήση της κοινής μνήμης για αποθήκευση ενδιάμεσων αποτελεσμάτων που επαναχρησιμοποιούνται
- Εκτέλεση με χρήση της κοινής μνήμης και των TILES για αποθήκευση και των vectors/matrixes

Τους χειρότερους χρόνους παρουσιάζουν ο naïve τρόπος και η χρήση της εντολής atomicAdd() η οποία παρουσίαζε μεγάλη καθυστέρηση, ενώ οι άλλοι δύο τρόποι εκτέλεσης παρουσιάζουν σημαντικά καλύτερη απόδοση χρόνου λόγω της shared memory. Συγκεκριμένα τον αποδοτικότερο χρόνο είχε η χρήση των TILES στην κοινή μνήμη, όμως λόγω του ότι έβγαζε σωστά αποτελέσματα μόνο για τετραγωνικά μητρώα δεν χρησιμοποιήθηκε. Ο κώδικας που χρησιμοποιήθηκε, λοιπόν, είναι αυτός που αποθηκεύει τα ενδιάμεσα αποτελέσματα τα οποία, μάλιστα, επαναχρησιμοποιούνται επανειλημμένα στις for().

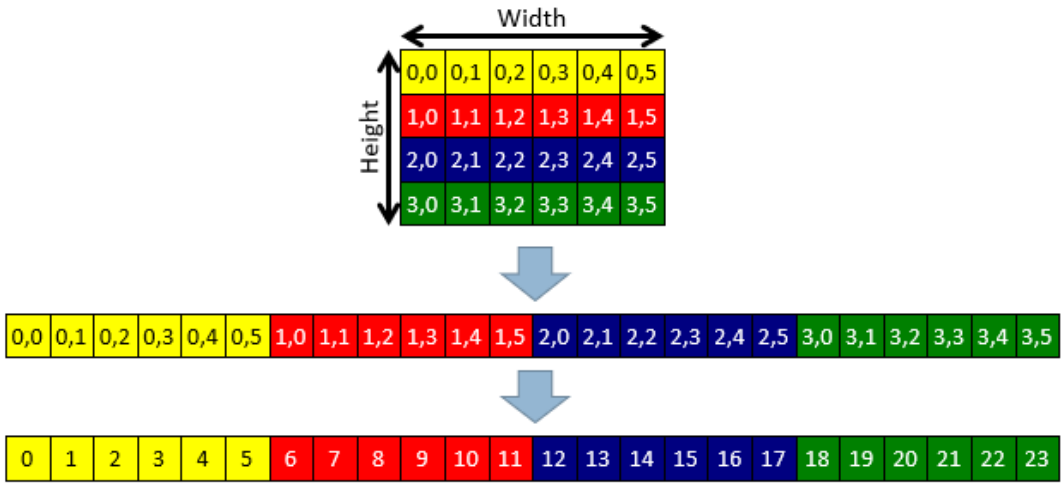
Σε κάθε ερώτημα θα παραθέτουμε και διαγράμματα χρόνου-μεγέθους.

Ερώτημα 1

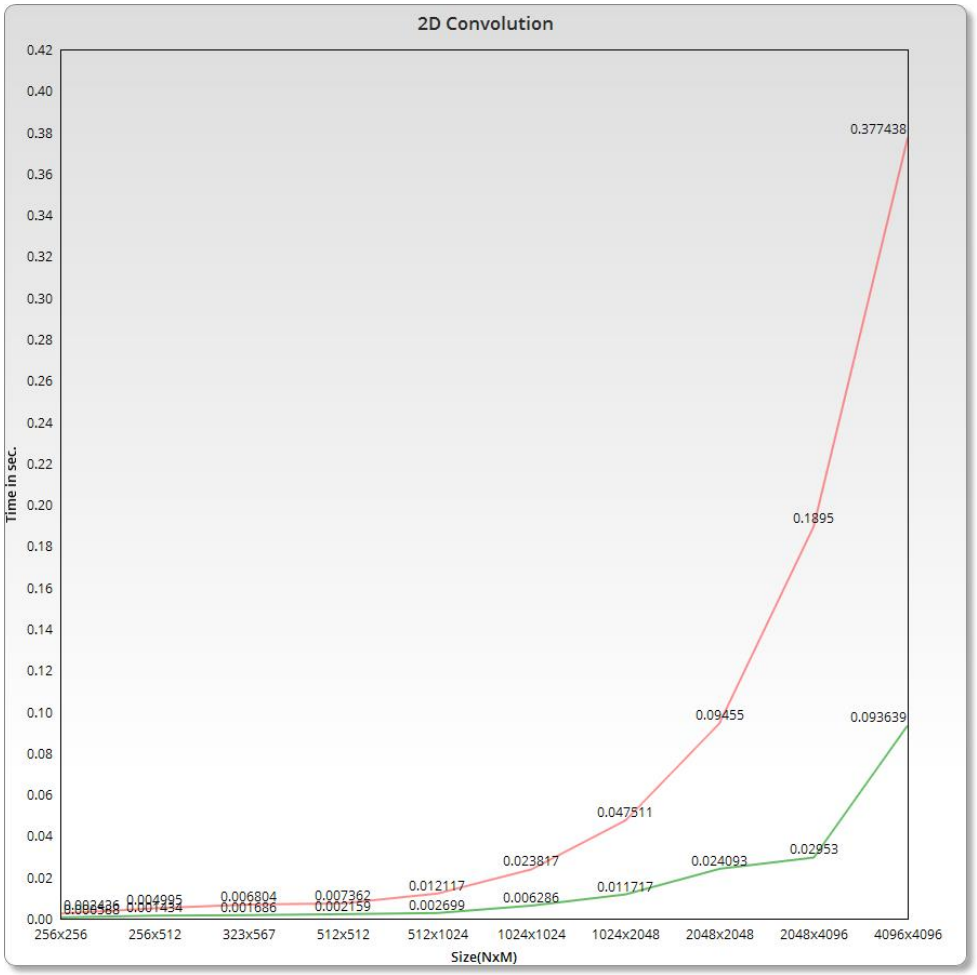
Μεταφέρουμε τα δεδομένα από τον host στον device, τα blocks στο δισδιάστατο πλέγμα που δημιουργήσαμε έχουν διάσταση 32x32, στρογγυλοποιούμε προς τα πάνω για το πλήθος των blocks σε κάθε διάσταση και τα δεδομένα αποθηκεύονται στην καθολική μνήμη. Κάθε thread εκτελεί πράξεις για τον υπολογισμό μιας θέσης του μητρώου. Για να μπορέσει ο υπολογιστικός πυρήνας να φτάσει το μέγιστο εύρος ζώνης της καθολικής μνήμης, θα πρέπει τα threads με διαδοχικό threadIdx.x να προσπελαύνουν διαδοχικά στοιχεία στη μνήμη. Με τον τρόπο αυτό, γίνεται συγκερασμός του υλικού και με μία

αίτηση για προσπέλαση της καθολικής μνήμης επιστρέφονται πολλά διαδοχικά στοιχεία (coalesced memory). Προκειμένου να εφαρμοστεί η τεχνική του συγκερασμού μνήμης στο συγκεκριμένο υπολογισμό της συνάρτησης, έγινε η θεώρηση της προσπέλασης του μητρώου κατά στήλες, ενώ η αποθήκευση του στη μνήμη γίνεται κατά γραμμές. Συνεπώς, κάθε στοιχείο του τελικού αποτελέσματος προκύπτει ως εσωτερικό γινόμενο στήλης του μητρώου επί το διάνυσμα.

Στο παρακάτω σχήμα φαίνεται πως απεικονίζεται το μητρώο στη μνήμη:



Παρακάτω βλέπουμε την βελτίωση του χρόνου εκτέλεσης προγράμματος χρησιμοποιώντας την γλώσσα προγραμματισμού CUDA(πράσινο χρώμα) σε σχέση με την ακολουθιακή(κόκκινο χρώμα) εκτέλεση:

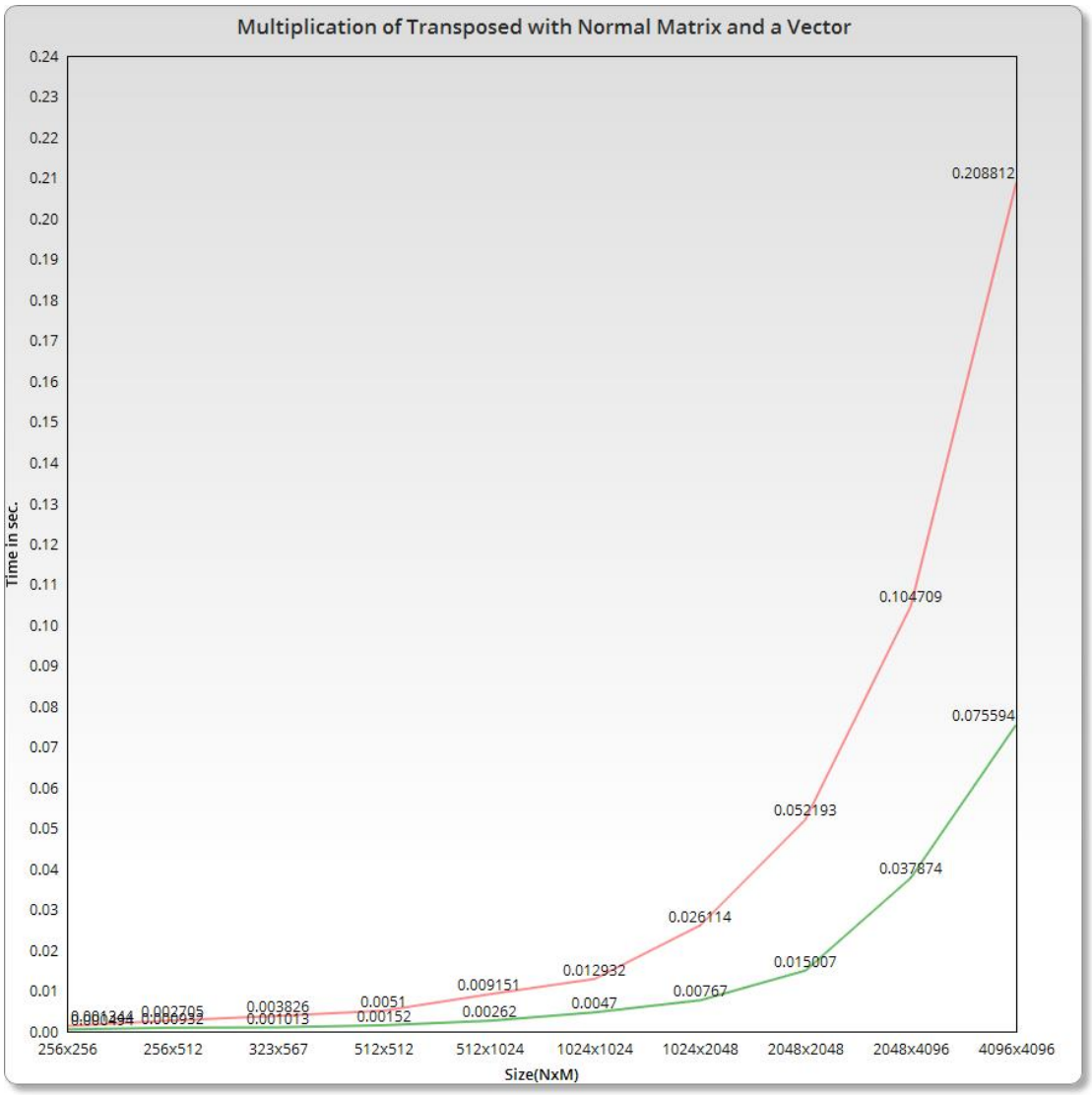


Ερώτημα 2

Χωρίς την χρήση της atomicAdd() που προαναφέρθηκε γιατί παρουσίαζε μεγάλη καθυστέρηση στον χρόνο εκτέλεσης, χωρίσαμε την συνάρτηση trans_norm_vector() σε δύο επιμέρους συναρτήσεις. Με αυτόν το τρόπο αποφύγαμε την χρήση της καθυστέρησης που πρόσθετε η atomicAdd() χρησιμοποιώντας το default stream διότι ο δεύτερος υπολογισμός απαιτούσε τα αποτελέσματα της πρώτης. Η πρώτη υπολογίζει τον tmp vector και η δεύτερη τον y vector. Μεταφέρουμε τα δεδομένα από τον host στον device, τα blocks στα δυο μονοδιάστατα πλέγματα που δημιουργήσαμε έχουν διάσταση

16x16, στρογγυλοποιήσαμε προς τα πάνω για το πλήθος των blocks σε κάθε διάσταση και τα δεδομένα αποθηκεύονται στην καθολική μνήμη. Κάθε thread εκτελεί πράξεις για τον υπολογισμό μιας θέσης του κάθε διανύσματος και τα αποτελέσματα αποθηκεύονται στην shared memory, μιας και επαναχρησιμοποιούνται στον υπολογισμό του τελικού αποτελέσματος. Έτσι καταφέραμε να μειώσουμε κατά πολύ τον χρόνο εκτέλεσης του προγράμματος σε αντίθεση με την προσπέλαση τους στην καθολικής μνήμης. Έπειτα από πολλά πειράματα φτάσαμε στο συμπέρασμα ότι η διάσταση 16x16 για την συγκεκριμένη κάρτα γραφικών προσφέρει την καλύτερη απόδοση.

Παρακάτω βλέπουμε την βελτίωση του χρόνου εκτέλεσης προγράμματος χρησιμοποιώντας την γλώσσα προγραμματισμού CUDA(πράσινο χρώμα) σε σχέση με την ακολουθιακή(κόκκινο χρώμα) εκτέλεση:



Ερώτημα 3

Με παρόμοιο τρόπο όπως και πριν χωρίσαμε την συνάρτηση covariance() σε τρεις επιμέρους συναρτήσεις. Η πρώτη υπολογίζει τον mean vector, η δεύτερη τον data matrix και η τρίτη τον symmat vector που είναι το αποτέλεσμα του υπολογισμού του covariance. Μεταφέρουμε τα δεδομένα από τον host στον device, τα blocks στα πλέγματα που δημιουργήσαμε έχουν διάσταση 16x16, στρογγυλοποιήσαμε προς τα πάνω για το πλήθος των blocks σε κάθε διάσταση και τα δεδομένα αποθηκεύονται στην καθολική μνήμη. Κάθε thread εκτελεί πράξεις για τον υπολογισμό μιας θέσης του κάθε διανύσματος/μητρώου και τα αποτελέσματα αποθηκεύονται στην shared memory, μιας και επαναχρησιμοποιούνται στον υπολογισμό του τελικού αποτελέσματος.

Παρακάτω βλέπουμε την βελτίωση του χρόνου εκτέλεσης προγράμματος χρησιμοποιώντας την γλώσσα προγραμματισμού CUDA(πράσινο χρώμα) σε σχέση με την ακολουθιακή(κόκκινο χρώμα) εκτέλεση: