Προγραμματισμός Συστημάτων Υψηλής Επίδοσης

Lab4 Report

Χουλιαράς Ανδρέας ΑΕΜ:2143

Το σύστημα στο οποίο έγινε η εργασία έχει τα εξής χαρακτηριστικά:

CPU: Intel Core i7 6700 RAM: 16GB (2x8) DDR4-2133Mhz GPU: NVidia GTX 1070

OS: Ubuntu 16.4.5 LTS (σε Virtual Machine)

Kernel: 4.15.0-36-generic

Τα ερωτήματα απαντήθηκαν με βάση τις εκτελέσεις στο σύστημα Artemis.

Ερώτημα 0: το device Query στο μηχάνημα μου, εμφάνισε τα εξής :

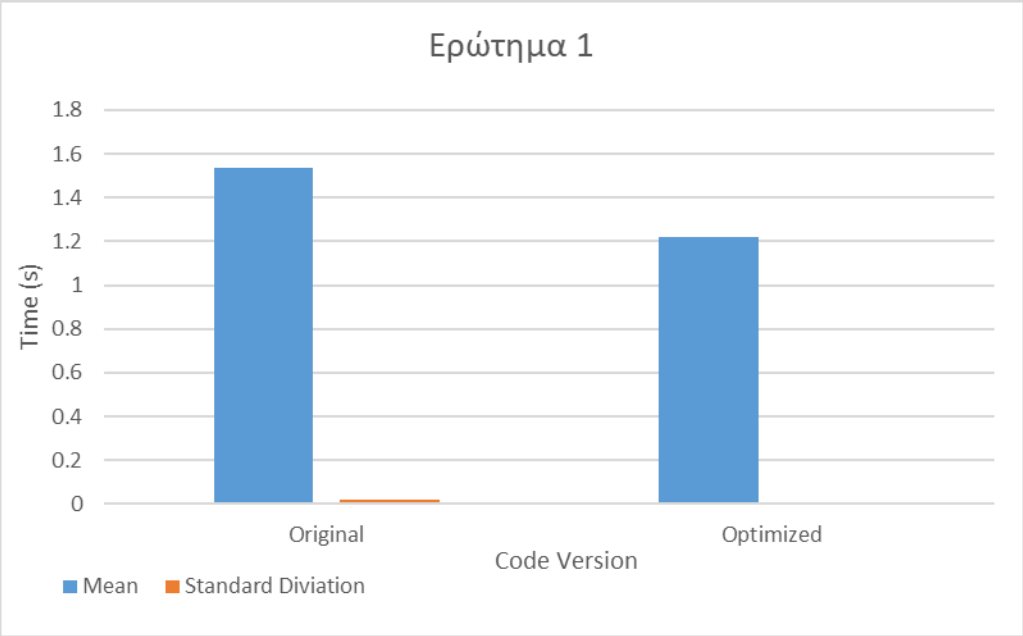
```
Device 0: "GeForce GTX 1070"
  CUDA Driver Version / Runtime Version          10.0 / 10.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 8118 MBytes (8512602112 bytes)
  (15) Multiprocessors, (128) CUDA Cores/MP:     1920 CUDA Cores
  GPU Max Clock rate:                            1835 MHz (1.84 GHz)
  Memory Clock rate:                             4004 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.0, CUDA Runtime Version = 10.0, NumDevs = 1
Result = PASS
user@admin:~/NVIDIA_CUDA-10.0_Samples/1_Utilities/deviceQuery$
```

Ερώτημα 1:

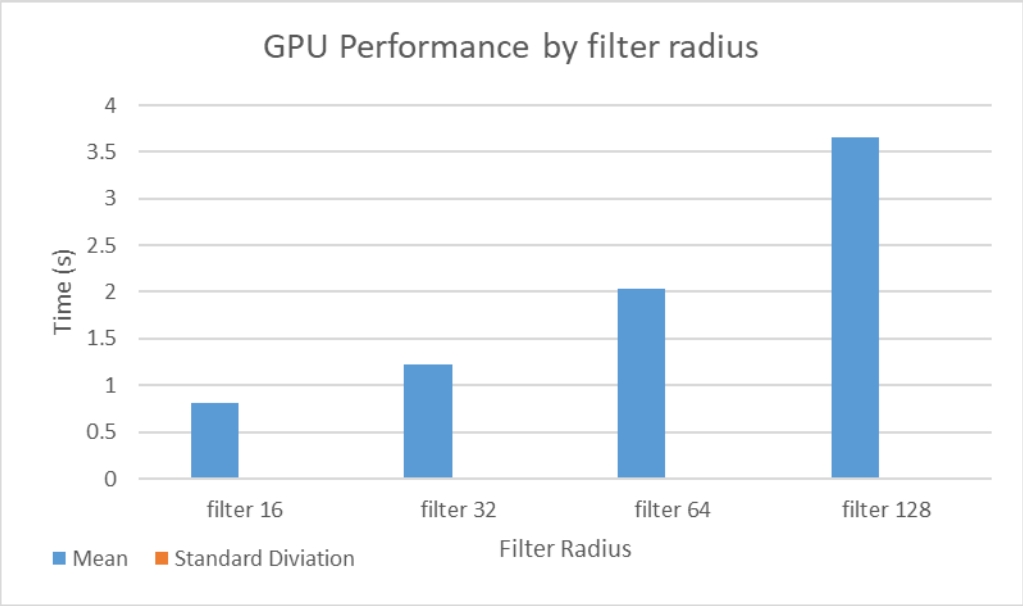
Η πιο πρόσφορη κίνηση είναι να βάλουμε το φίλτρο στην constant memory γιατί είναι πιο γρήγορη από τις προσπελάσεις στην κύρια μνήμη, τα δεδομένα του φίλτρου χρησιμοποιούνται πάρα πολύ, και το μοτίβο προσπελάσεων του φίλτρου κάνει την constant cache ιδανική.
Επειδή όμως πρέπει να δηλώσουμε ρητά απο την αρχή το μέγεθος των δεδομένων που θα δεσμεύσουμε στην constant memory το έκανα αρκετά μεγάλο για να χωράει φίλτρο ακτίνας μέχρι 256.

Οι βελτίωση φαίνετε στο ακόλουθο γράφημα η οποία ανέρχεται στα 20.78%:



<u>Ερώτημα 2</u>:

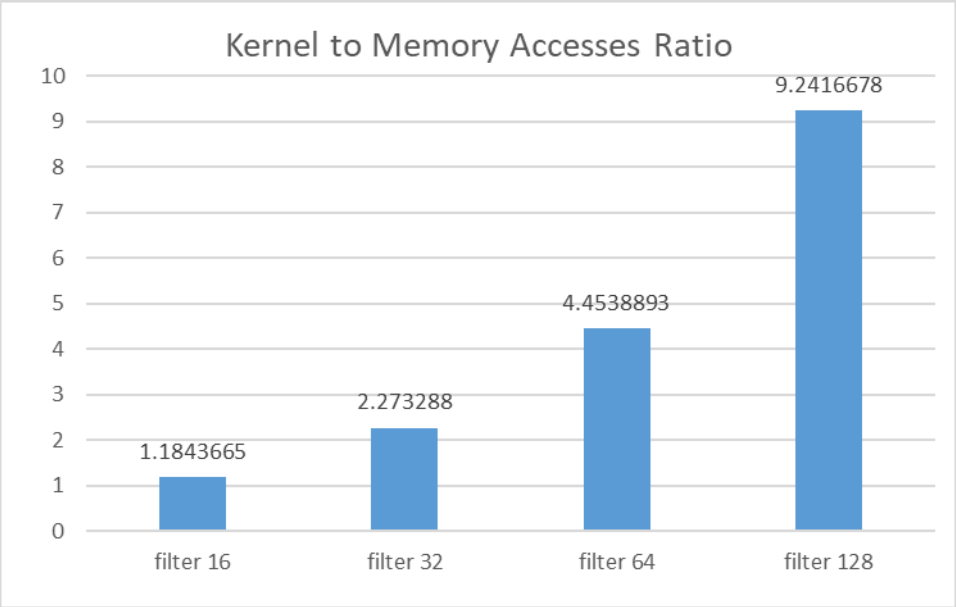
Όπως φαίνεται στο παρακάτω γράφημα υπάρχει αύξηση στον χρόνο εκτέλεσης του προγράμματος καθώς αυξάνετε το μέγεθος της ακτίνας του φίλτρου και μάλιστα ο ρυθμός μοιάζει να είναι εκθετικός:



Υπολογίζοντας τον λόγο μεταξύ χρόνου εκτέλεσης kernels και χρόνου μεταφορών μνήμης προκύπτει ότι οι χρόνοι των μεταφορών στην μνήμη δέν αλλάζουν και τόσο παρά την αύξηση στους χρόνους εκτέλεσης του προγράμματος. Οι χρόνοι εκτέλεσης των kernels αυξάνονται μόνο όσο μεγαλώνει το φίλτρο. Αυτό σημαίνει ότι το πρόβλημα γίνετε όλο και λιγότερο memory bound όσο προχωράμε σε μεγαλύτερα μεγέθη φίλτρου.

Αυτό αποδεικνύει πως η κίνηση να βάλουμε το φίλτρο στην cache ήταν αποδοτική.

Ακολουθεί το γράφημα σχετικό γράφημα



Ερώτημα 3:

Εφαρμόστηκε η τεχνική tiling ώστε όλα τα δεδομένα που απαιτούνται για την εκτέλεση κάθε βήματος να χωράνε στη GPU. Έγινε χρήση shared memory ανάμεσα σε νήματα του ίδιου block για μεγαλύτερη επίδοση.

Επειδή τμηματοποίησα πολύ τους υπολογισμούς σε πολλά kernel invocations το overhead είναι μεγαλύτερο.

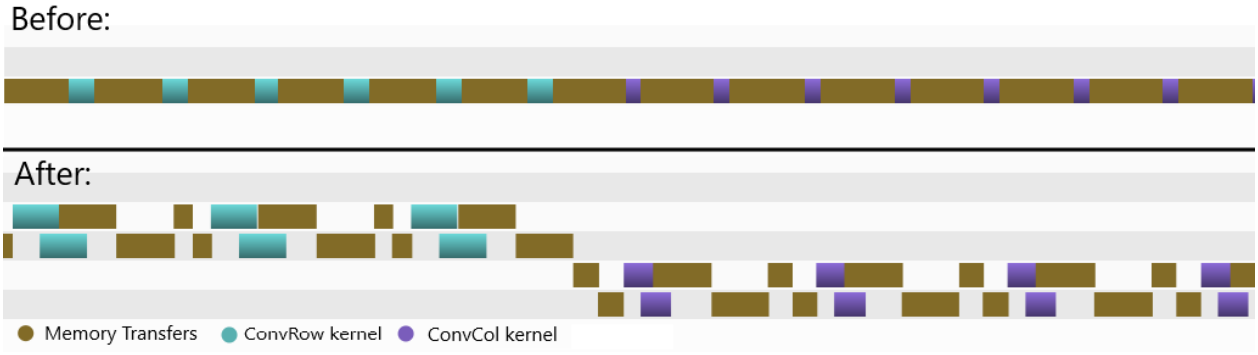
Ο χρόνος εκτέλεσης είναι μεγαλύτερος, αν και το πόσο εξαρτάται απο το μέγεθος των tiles.

Αλλά μπορούμε να υποστηρίξουμε μεγαλύτερες εικόνες, πράγμα που δεν ίσχυε πριν λόγω του σχετικά μικρού μεγέθους της μνήμης της GPU.

η υλοποίησή δουλεύει σωστά για εικόνες μεγέθους 16384x16384

Ερώτημα 4:

Παρατηρώντας τα αποτελέσματα του profiler δεν υπάρχει καμία επικάλυψη ανάμεσα σε εκτελέσεις kernels και μεταφορές δεδομένων. Αυτό φαίνετε να αλλάζει με την χρήση streams.



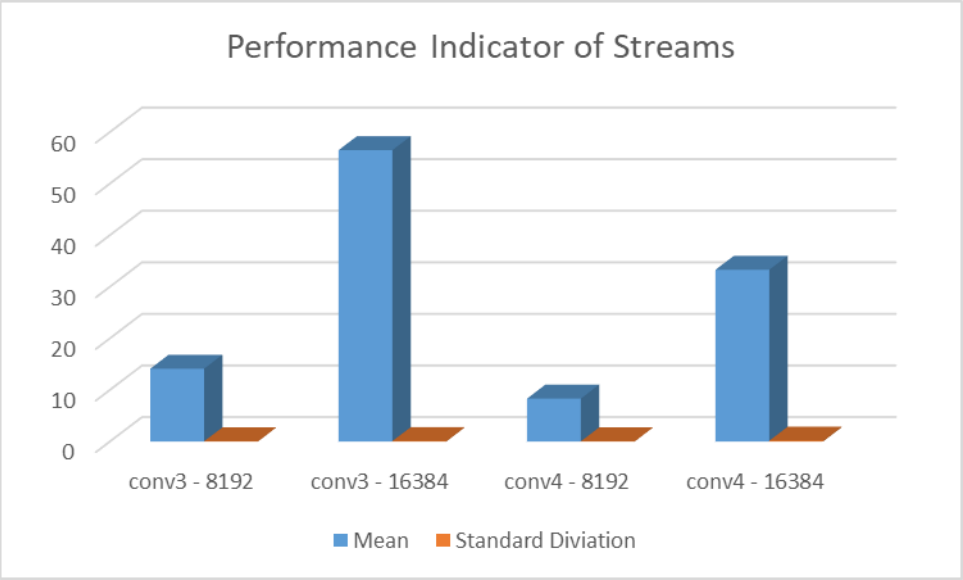
Η παραπάνω εικόνα απεικονίζει τα αποτελέσματα απο τον profiler. Χρησιμοποίησα 2 streams για την συνέλιξη κατα γραμμές και 2 για την συνέλιξη κατά στήλες.

Τα πλάτη των σχημάτων που απεικονίζονται δεν είναι συγκρίσημα μεταξύ των δυο υλοποιήσεων γιατί έκανα την εικόνα με το χέρι στη ζωγραφική και το ζουμ πού έκανα στις δύο περιπτώσεις διαφέρει λίγο, αλλά το παρέθεσα κυρίως για να δείξω ότι όντως υπάρχει επικάλυψη.

Στο παρακάτω γράφημα φάινετε η βέλτίωση και στους χρόνους εκτέλεσης χάρις την χρήση των streams.

Conv3 εννοεί τον κώδικα με tiling χωρίς streams

Conv4 εννοεί τον κώδικα με tiling και με streams



Ερώτημα 5:

Χάρις την τεχνική του tiling κατάφερα να φτάσω μέχρι εικόνες μεγέθους 32.768x32.768 χωρίς κανένα πρόβλημα, αλλά με αρκετά μεγάλο χρόνο εκτέλεσης που φαίνετε να δικαιολογείτε γιατί μιλάμε για μεγάλο όγκο δεδομένων. Για εικόνες μεγέθους 65.536x65.536 δεν κατάφερε να τρέξει. Ο μόνος πόρος που μπορεί να μας περιορίζει είναι πλέον η μνήμη RAM γιατί εκεί αποθηκεύονται οι πίνακες ολόκληροι, αντίθετα με την μνήμη της GPU που έχει μόνο ένα tile της είκονας που υπολογίζει.