ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΟΥ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ (ECE 415)

Ακαδημαϊκό έτος 2021-2022

3η Εργαστηριακή Άσκηση

Υλοποίηση απλού προγράμματος CUDA και πειραματισμός με γεωμετρίες block και grid, προβλήματα πλήθους νημάτων και ακρίβειας.

Φοιτητές:

Ηλιάδης Ηλίας, ΑΕΜ: 2523

Μακρής Δημήτριος-Κων/νος – ΑΕΜ: 2787

Σκοπός της 3ης εργαστηριακής άσκησης ήταν η υλοποίηση μιας εφαρμογής επεξεργασίας εικόνας που χρησιμοποιεί τη μέθοδο της συνέλιξης, και η μετατροπή του κώδικα σε εφαρμογή CUDA, ώστε να είναι δυνατή η μεταφορά και εκτέλεσή της στην GPU, σε συνδυασμό με επιπλέον πειραματικές μεθόδους όσον αφορά στην γεωμετρία των grids και των blocks, το μέγεθος των treads και την ακρίβεια των αποτελεσμάτων.

Απαντήσεις στα ερωτήματα που κληθήκαμε να παρουσιάσουμε:

0. Αποτελέσματα του deviceQuery:

```
Device 0: "Tesla K80"
  CUDA Driver Version / Runtime Version          11.4 / 11.5
  CUDA Capability Major/Minor version number:    3.7
  Total amount of global memory:                 11441 MBytes (11997020160 bytes)
  (013) Multiprocessors, (192) CUDA Cores/MP:    2496 CUDA Cores
  GPU Max Clock rate:                            824 MHz (0.82 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        114688 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:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 6 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

```
Device 1: "Tesla K80"
  CUDA Driver Version / Runtime Version          11.4 / 11.5
  CUDA Capability Major/Minor version number:    3.7
  Total amount of global memory:                 11441 MBytes (11997020160 bytes)
  (013) Multiprocessors, (192) CUDA Cores/MP:    2496 CUDA Cores
  GPU Max Clock rate:                            824 MHz (0.82 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        114688 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:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 7 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from Tesla K80 (GPU0) -> Tesla K80 (GPU1) : Yes
> Peer access from Tesla K80 (GPU1) -> Tesla K80 (GPU0) : Yes
```

2. Ο κώδικας βρίσκεται στο αρχείο task2.cu.

Μεταγλώττιση: nvcc -Xcompiler -O4 task2.cu -o task2

Εκτέλεση: cuda-memcheck ./task2

3.

α) Παρατηρώντας τα αποτελέσματα του deviceQuery, ο μέγιστος αριθμός threads ανά block είναι 1024, συνεπώς εφόσον χρησιμοποιούμε μόνο ένα block, το μέγιστο μέγεθος της εικόνας είναι 32x32.

β) Για να υπολογίσουμε την μέγιστη δυνατή ακρίβεια ούτως ώστε να μην συμβεί κάποιο σφάλμα σύγκρισης συντάχθηκε ο κωδικάς που απεικονίζεται παρακάτω στο αρχείο task2.cu:

```
for (i = 0; i < imageW * imageH; i++) {
  while(1){

    if(h_OutputGPU[i] > h_OutputCPU[i] + acc || h_OutputGPU[i] < h_OutputCPU[i] - acc){
      printf("Error: h_OutputGPU[%d] = %f, with difference: %f\n", i, h_OutputGPU[i], h_OutputGPU[i] - h_OutputCPU[i]);
      correct = false;
      acc += acc + 0.00001;
      continue;
    }

    else
      break;
  }
}

printf("accuracy: %f\n", acc);
```

Εσωτερικά του βρόγχου for αυξάνουμε συνεχώς την τιμή της ακρίβειας κατα 0.00001 (ξεκινώντας απο την τιμή 0.00005) μέχρι να εξαλειφθούν όλες οι αποκλήσεις που προκύπτουν απο την σύγκριση του τελικού πίνακα του

device με αυτον του host. Με την εκτέλεση του προγράμματος παρατηρήθηκε ότι η μέγιστη δυνατή ακρίβεια είναι ίση με 0.245750 .

4.

Ο κώδικας βρίσκεται στο αρχείο task4.cu

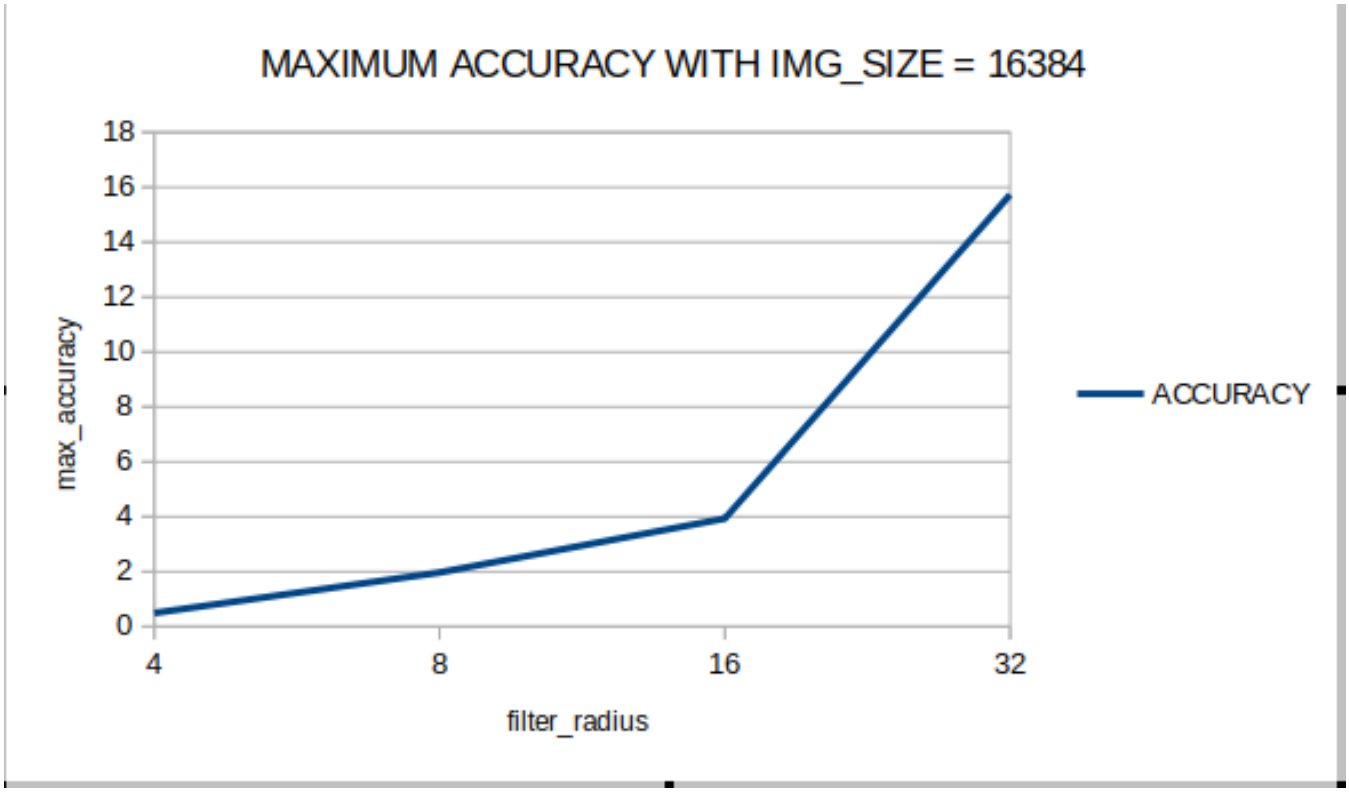
Μεταγλώττιση: nvcc -Xcompiler -O4 task4.cu -o task4

Εκτέλεση: cuda-memcheck ./task4

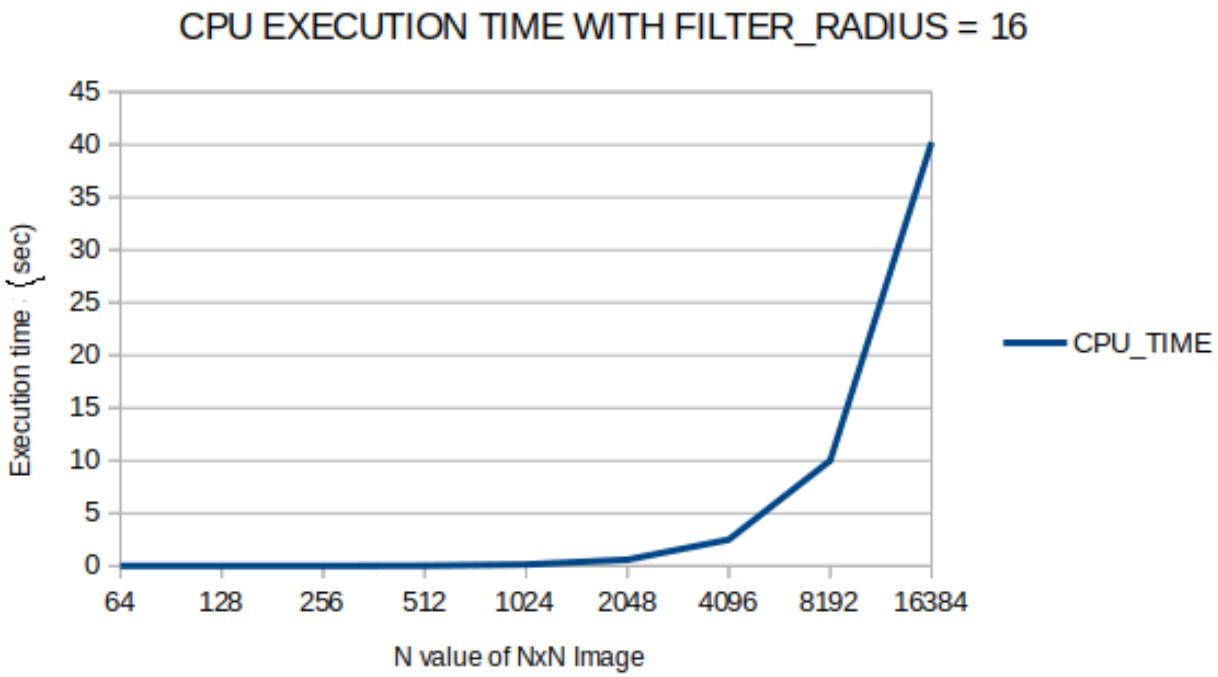
Έπειτα από τη δημιουργία πολλαπλών blocks μεγέθους 1024 thread σε (imageW/1024) x (imageH/1024) grid, το μέγιστο επιτρεπτό μέγεθος εικόνας αυξήθηκε σε 16384x16384, καθώς για την αμέσως επόμενη τιμή (32768x32768) το global memory του device μας περιορίζει, εμφανίζοντας μήνυμα “out of memory”.

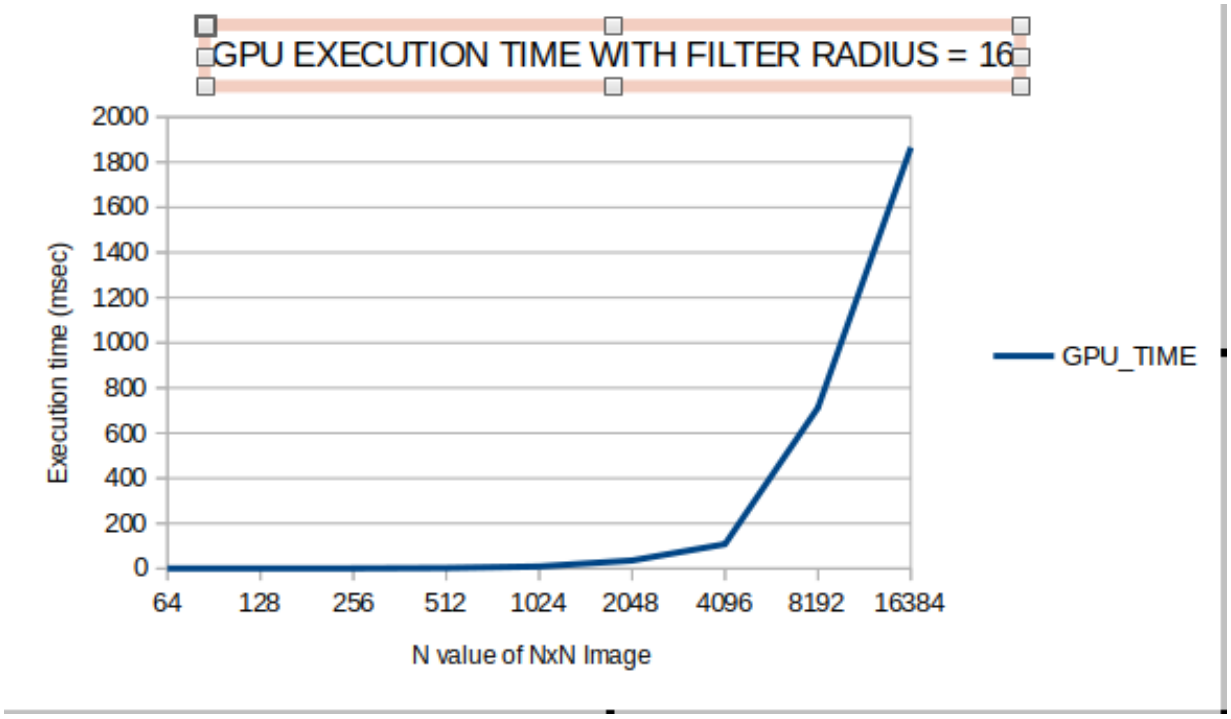
5.

α) Όσον αφορά στην ακρίβεια, από το διάγραμμα διαπιστώνουμε ότι για ακτίνα φίλτρου από 1, 2 έως και 16, η μέγιστη ακρίβεια αυξάνεται γραμμικά ενώ έπειτα πολλαπλασιάζεται σημαντικά, καθώς παρατηρούμε ότι για ακτίνα φίλτρου ίση με 32, η τιμή της έχει τετραπλασιαστεί στα 16 δεκαδικά ψηφία. Η αύξηση αυτή της ακρίβειας, οφείλεται πιθανότατα στο γεγονός ότι καθώς αυξάνεται το φίλτρο, η τελική τιμή είναι αποτέλεσμα πράξεων περισσότερων “γειτονικών” τιμών, κι έτσι υπάρχει λιγότερη απόκλιση μεταξύ τους στον σχηματισμό του τελικού αποτελέσματος.



β)Στα παρακάτω διαγράμματα απεικονίζονται οι χρόνοι εκτέλεσης στη CPU και την GPU αντίστοιχα. Όπως είναι αναμενόμενο, λόγω της ακολουθιακής εκτέλεσης στη CPU ο χρόνος είναι σημαντικά μεγαλύτερος, ενώ το τρέξιμο του αλγορίθμου με πολυνηματική μέθοδο είναι σαφέστατα ταχύτερο.


CPU EXECUTION TIME WITH FILTER_RADIUS = 16
Execution time (sec)
45
40
35
30
25
20
15
10
5
0
64
128
256
512
1024
2048
4096
8192
16384
N value of NxN Image
CPU_TIME


GPU EXECUTION TIME WITH FILTER RADIUS = 16
Execution time (msec)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
64
128
256
512
1024
2048
4096
8192
16384
N value of NxN Image
GPU_TIME

6.

Ο κώδικας βρίσκεται στο αρχείο task5.cu

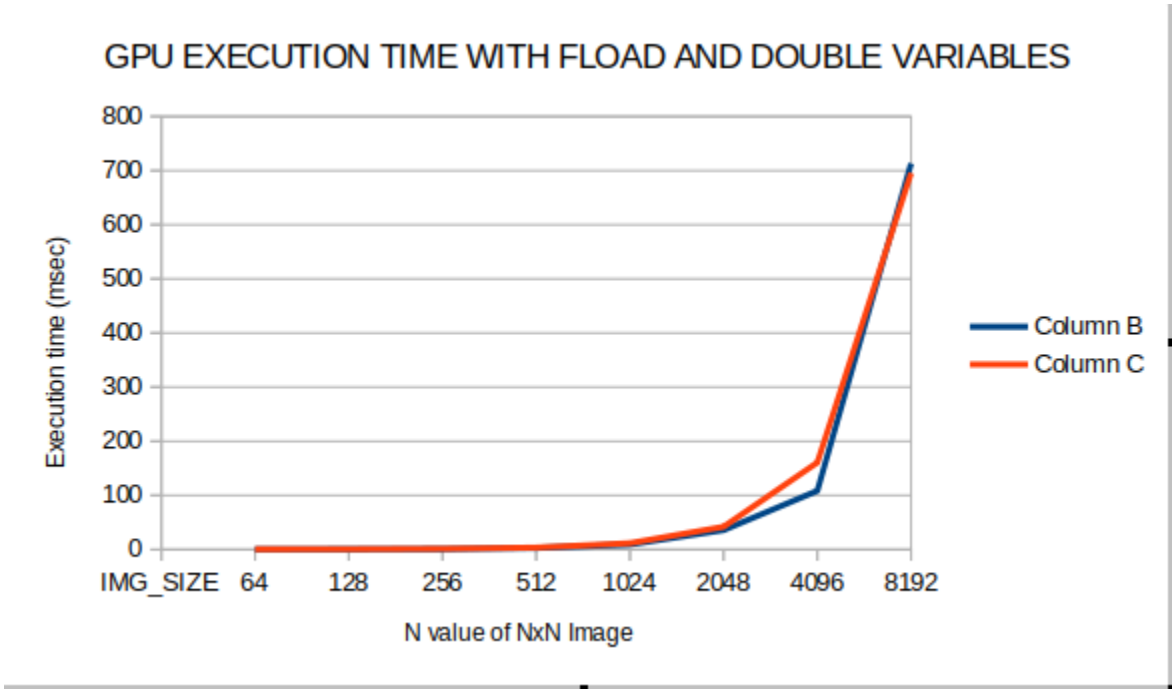
Μεταγλώττιση: nvcc -Xcompiler -O4 task5.cu -o task5

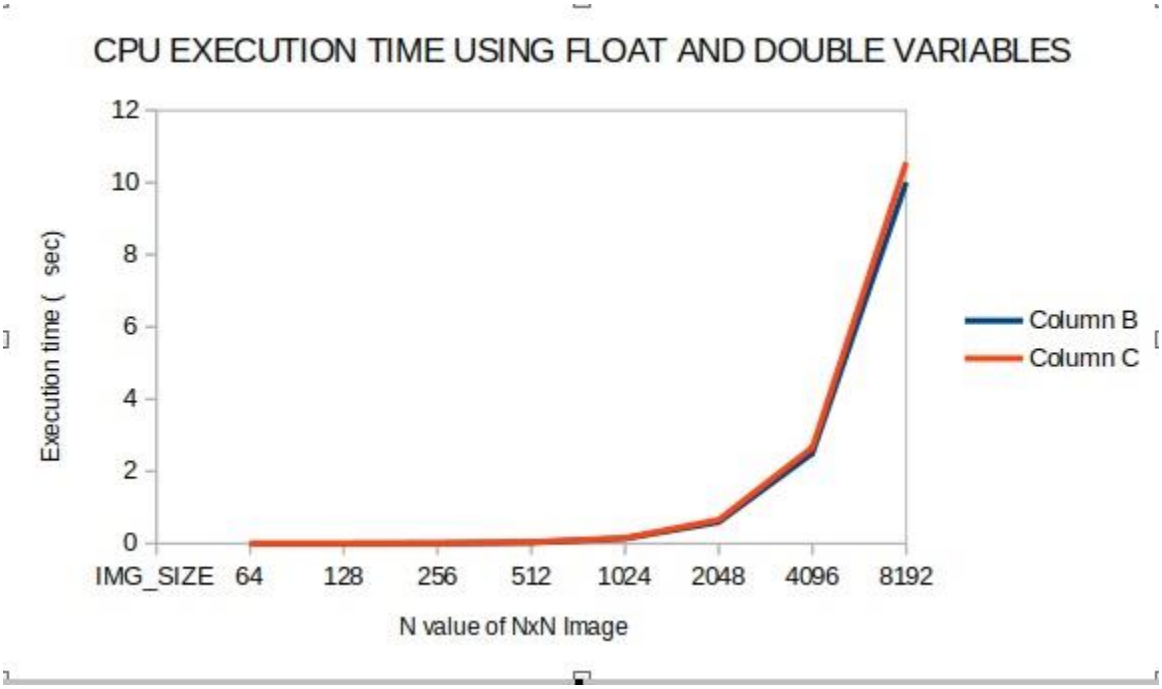
Εκτέλεση: cuda-memcheck ./task5

Ο compiler δεν εμφανίζει κάποιο warning με την αλλαγή του τύπου μεταβλητών σε double, ωστόσο στο makefile υπάρχουν τα αντίστοιχα arch flags σε περίπτωση που χρειαστούν σε κάποια αρχιτεκτονική.

α) Παρατηρείται ότι με την αλλαγή σε doubles, η μέγιστη δυνατή ακρίβεια για κάθε μέγεθος ακτίνας φίλτρου διατηρεί τιμή ίση με την τιμή αρχικοποίησης (0.00005) . Αυτό συμβαίνει καθώς οι doubles αποθηκεύονται σε διπλάσιο αριθμό bits απ'ότι οι floats με αποτέλεσμα να έχουμε πιο λεπτομερή αποθήκευση των ψηφίων του κάθεστοιχείου του πίνακα όπως και αναλυτικότερο υπολογισμό στις πράξεις.

β)Οι πράξεις μεταξύ των doubles απαιτούν περισσότερη ώρα για να εκτελεστούν. Επίσης, όπως είναι λογικό, η διαφορά μεταξύ των χρόνων εκτέλεσης από float σε doubles αυξάνεται όσο αυξάνεται και το μέγεθος της εικόνας.





7.

α) Κάθε στοιχείο της εικόνας διαβάζεται (imageW-2)*imageH*(2*filter+1), ενώ του φίλτρου (imageW-2)*imageH+imageW*(imageH-2).

β) Παρατηρούμε ότι στον κώδικα του kernel γίνονται 2 προσπελάσεις στην global memory της GPU και 2 πράξεις κινητής υποδιαστολής (1 πρόσθεση και 1 πολλαπλασιασμός) που βρίσκονται μέσα στην if. Συνεπώς, ο λόγος προσπελάσεων μνήμης προς πράξεις κινητής υποδιαστολής είναι 2/2, δηλαδή 1.

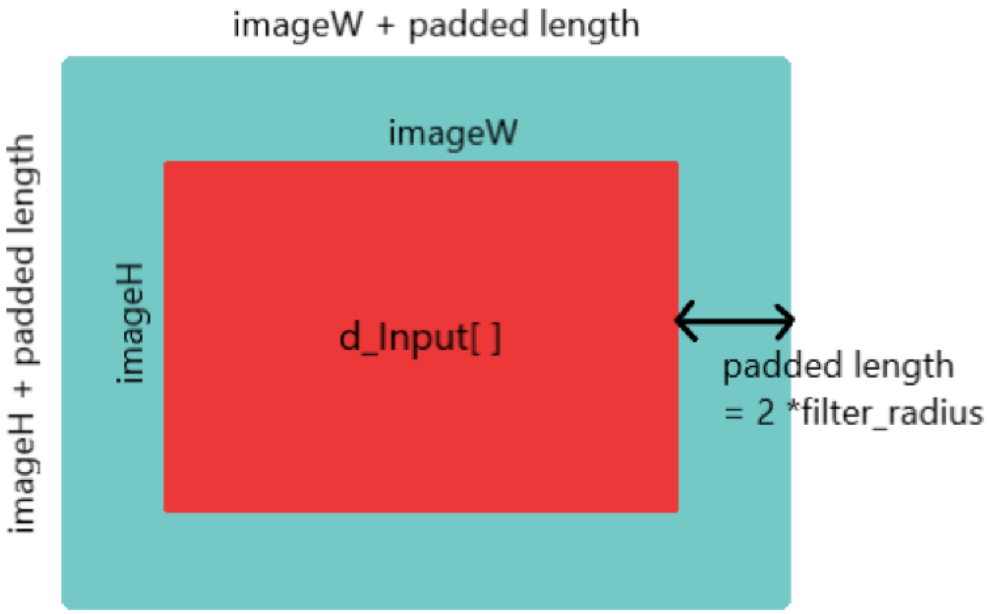
8.

Ο κώδικας βρίσκεται στο αρχείο task8.cu

Μεταγλώττιση: nvcc -Xcompiler -O4 task8.cu -o task8

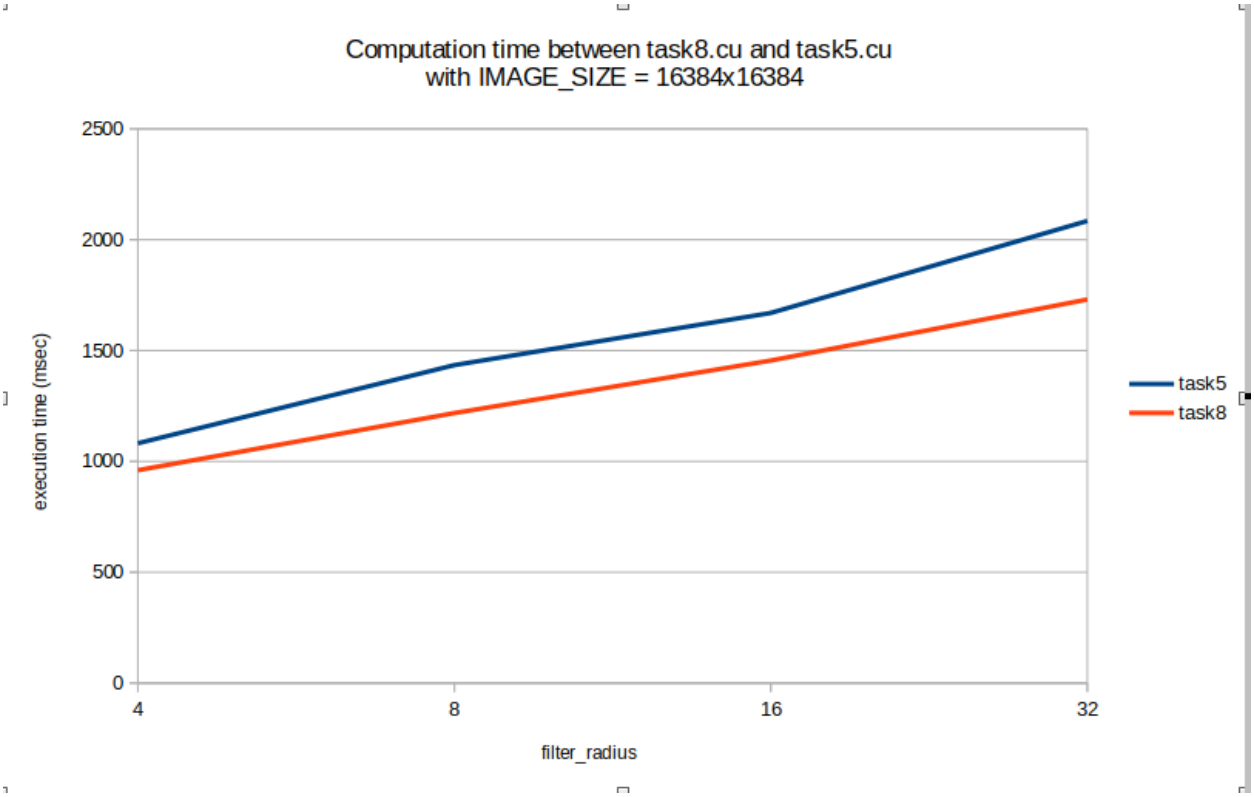
Εκτέλεση: cuda-memcheck ./task8

Για να εξαλείψουμε ολοκληρωτικά το φαινόμενο του control flow divergence, αυξήσαμε το μέγεθος του πίνακα h_Input και d_Input με μέγεθος padding 2*filter_radius ούτως ώστε να απαλαχθούμε απο τις δύο if εντός των for loop των wraps convolutionRowGPU και convolutionColumnGPU.



Αφου πρώτα ελέγχθηκε ο κώδικας ως προς την ορθότητα του συγκρίνοντας το output του device με το output του host, έγινε σύγκριση του χρόνου

εκτέλεσης του device με αυτόν του αρχείου task5.cu . Για μέγεθος εικόνας 16384x16384 πειραματιστήκαμε με διάφορες τιμές της ακτίνας φίλτρου και παρατηρήθηκε αισθητή βελτίωση του χρόνου εκτέλεσης:



Ωστόσο για μικρότερα μεγέθη εικόνας, παρατηρήθηκε μικρή αύξηση του χρόνου εκτέλεσης σε σχέση με τον κώδικα του task5.cu .