Συστήματα Υπολογισμού Υψηλών Επιδόσεων (ECE 415)
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

3η Εργαστηριακή Άσκηση

Στοιχεία φοιτητών:

Μπαλτάς Νικόλαος, 2757
Ξωχέλλη Ελένη, 2761

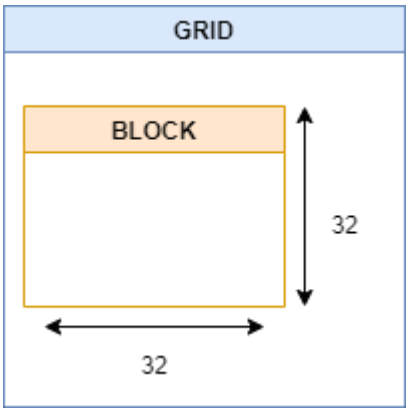
0.Παράθεση Device Query

```
CUDA Device Query (Runtime API) version (CUDART static linking)
  Detected 1 CUDA Capable device(s)
  Device 0: "NVIDIA GeForce GTX 1650 SUPER"
  CUDA Driver Version / Runtime Version          11.5 / 11.5
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 3912 MBytes (4101505024 bytes)
  (020) Multiprocessors, (064) CUDA Cores/MP:    1280 CUDA Cores
  GPU Max Clock rate:                            1755 MHz (1.75 GHz)
  Memory Clock rate:                             6001 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 1048576 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 shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  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 3 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 Managed Memory:                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 / 38 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.5, CUDA Runtime Version = 11.5, NumDevs = 1
Result = PASS
```

3.Πειραματική μελέτη: μέγιστο μέγεθος εικόνας

I. Το μέγιστο μέγεθος εικόνας που μπορούμε να υποστηρίξουμε χωρίς να συμβεί κάποιο σφάλμα εκτέλεσης είναι το **32 X 32**.Αυτό το runtime error το προκύπτει με τον έλεγχο για error μετά την κλήση κάποιου kernel. Στο ερώτημα αυτό χρησιμοποιήσαμε ένα μοναδικό δισδιάστατο block απο threads. Το σφάλμα αυτό οφείλεται στο γεγονός οτι ο μέγιστος αριθμός threads που μπορεί να υπάρχουν σε ένα block, βλέποντας το **Device Query** είναι **1024 threads (Maximum number of threads per block: 1024)**. Επομένως αφού το block είναι δισδιάστατο θα έχουμε **32 X 32 = 1024 threads**.'Ετσι προκύπτει το μέγιστο μέγεθος εικόνας που μπορούμε να υποστηρίξουμε.

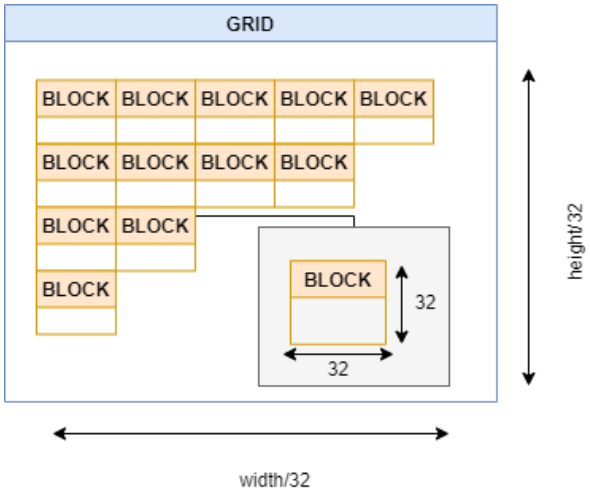


II. Για μέγιστο μέγεθος εικόνας 32 X 32 σε παρακάτω γράφημα φαίνεται η ακρίβεια που μπορεί να υποστηριχθεί ανάλογα με το μέγεθος του φίλτρου σε σύγκριση με την ακρίβεια που υποστηρίζεται με χρήση grid μεγαλύτερης διάστασης.

4.Εξήγηση αλλαγών στον κώδικα

Με την νέα δομή τετράγωνου δυσδιάστατου grid ο μέγιστος αριθμός block είναι 65536 δηλαδή η κάθε διάσταση μπορεί να έχει μέγεθος 256. Όπως αναφέρθηκε προηγουμένως το κάθε block μπορεί να έχει διάσταση 32 και μέχρι 1024 threads. Τελικά η μέγιστη διάσταση εικόνας που μπορεί να υποστηρίξει το πρόγραμμά μας είναι 256*32=8192, αφού κάθε thread είναι υπεύθυνο για ένα pixel.

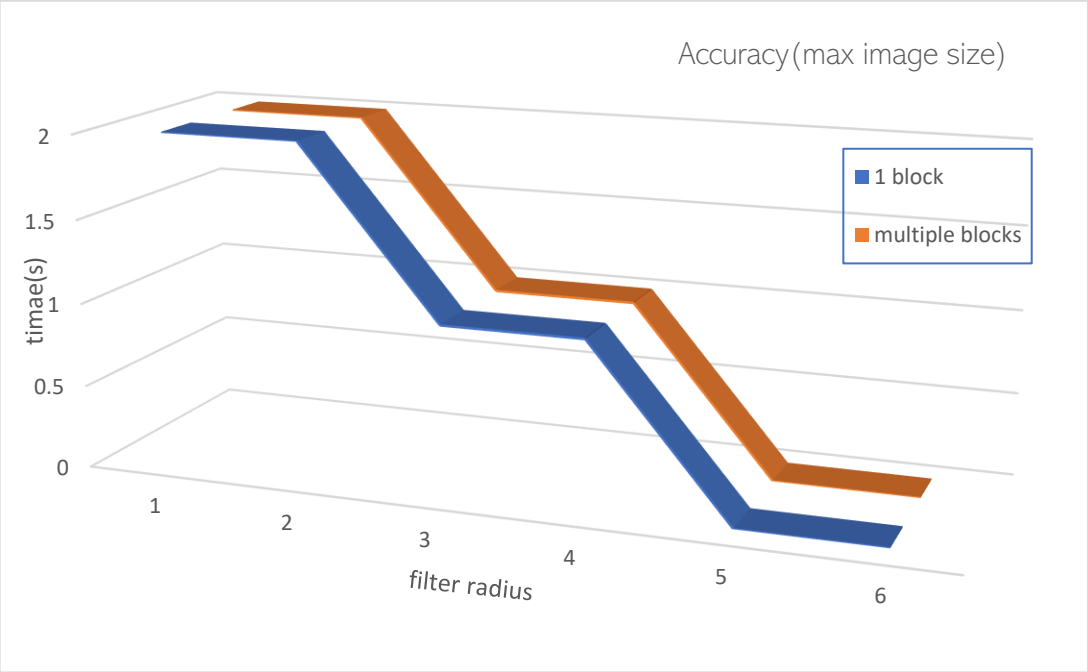
Στον κώδικα το οι διαστάσεις του grid ορίζονται ως height/32 * width/32 όπου width= height και οι διαστάσεις του κάθε block 32*32.



5.Πειραματική μελέτη: πολλαπλά blocks

I. Μέγιστη ακρίβεια συναρτήσει του μεγέθους φίλτρου

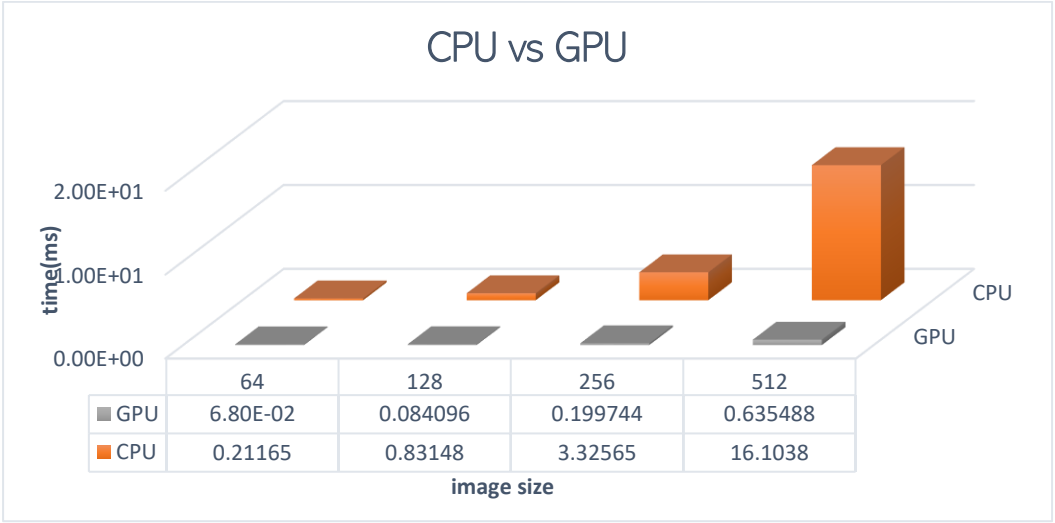
Ξεκινώντας με ακρίβεια 0.5 και αυξάνοντας την σε κάθε βήμα καταλήγουμε στα συμπεράσματα που φαίνονται στο διάγραμμα. Βλέπουμε τι ακρίβεια επιτυγχάνεται για το μέγιστο μέγεθος εικόνας σε κάθε περίπτωση μεγέθους φίλτρου.



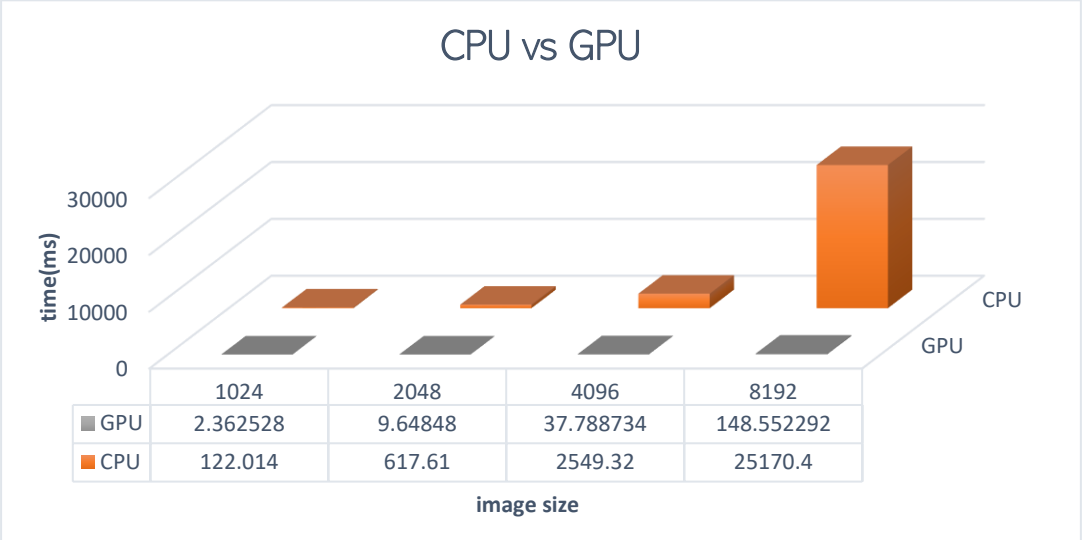
Να σημειώσουμε ότι η συμπεριφορά του κώδικα είναι αντίστοιχη ανεξάρτητα από τη δομή του grid. Βέβαια, μέσω της δεύτερης δομής(κώδικας 4) το μέγιστο μέγεθος εικόνας είναι μεγαλύτερο οπότε οι μετρήσεις γίναν με διαφορετικό μέγιστο ανάλογα με τις δυνατότητες της κάθε έκδοσης κώδικα.

II. Σύγκριση CPU/GPU

Όπως εξηγήσαμε προηγουμένως το μέγιστο μέγεθος εικόνας με το οποίο μπορούμε να έχουμε ακριβή αποτελέσματα είναι 8192*8192. Για τις συγκεκριμένες διαστάσεις και για ακτίνα φίλτρου ίση με 16 πραγματοποιήθηκαν οι μετρήσεις σύγκρισης CPU/GPU. Με την αύξηση του μεγέθους εικόνας οι χρόνοι αυξάνονται δραματικά οπότε τα αποτελέσματα παρουσιάζονται σε 2 μέρη αναλυτικά στα παρακάτω 2 διαγράμματα.



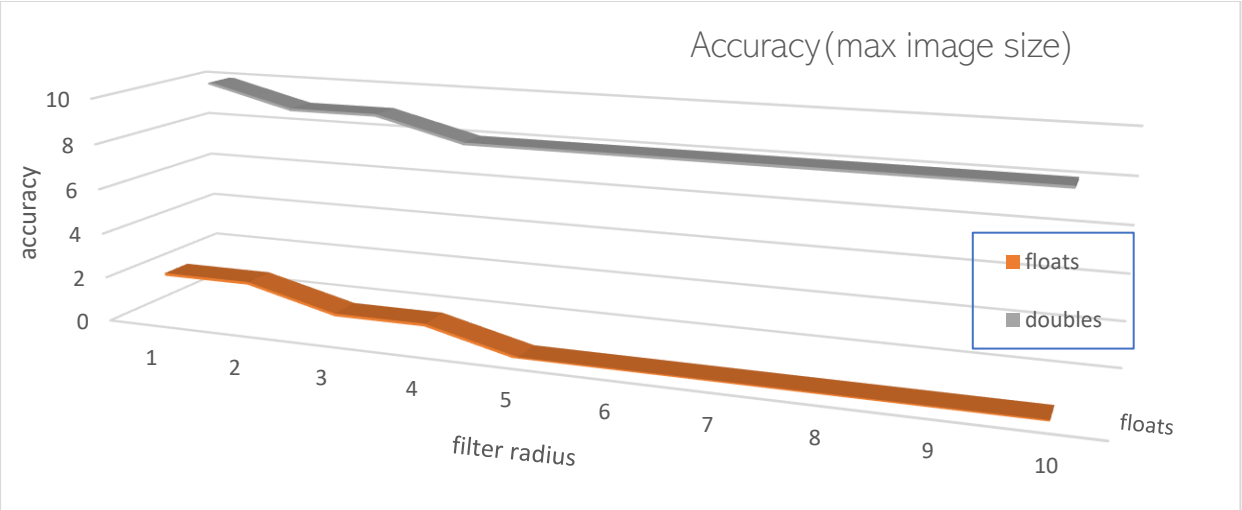
Είναι εμφανές ότι ενώ και οι δύο ποσότητες αυξάνονται με την αύξηση του μεγέθους της εικόνας η διαφορά ανάμεσα στους χρόνους τους είναι τεράστια και η επίδοση της GPU φαίνεται πολύ καλύτερη.



Οι μετρήσεις για την CPU πραγματοποιούνται με την χρήση **struct timespec** και αντίστοιχων συναρτήσεων ενώ για την GPU ακολουθήθηκε διαφορετική μέθοδος. Δημιουργήθηκαν event για start και stop και μέσω κλήσεων των κατάλληλων συναρτήσεων για τον χειρισμό τους προέκυψε το τελικό συμπέρασμα.

6.Πειραματική μελέτη: doubles

Σε συνδυασμό με τις μετρήσεις του ερωτήματος 5.α προκύπτει το συγκεκριμένο γράφημα για σύγκριση της ακρίβειας του αποτελέσματος συναρτήσει του τύπου δεδομένων. Οι μετρήσεις γίναν με το μέγιστο μέγεθος εικόνας. Όπως περιμέναμε με χρήση double επιτυγχάνεται πολύ καλύτερη ακρίβεια για όλα τα μεγέθη φίλτρου.



7.Προσπελάσεις μνήμης

I. Το κάθε thread σε κάθε κλήση του kernel εκτελεί **2*filterR+1** επαναλήψεις λόγω της δομής επανάληψης. Σε κάθε επανάληψη (εφόσον είναι αληθής η συνθήκη και μόνο στην πρώτη κλήση kernel που έχει ως είσοδο την αρχική εικόνα) διαβάζει ένα στοιχείο της εικόνας και ένα στοιχείο του φίλτρου.
Άρα έχουμε **n^2*(2*filterR+1)** αναγνώσεις της αρχικής εικόνας.
Το φίλτρο χρησιμοποιείται και στην δεύτερη κλήση του kernel οπότε έχουμε **2* n^2*(2*filterR+1)** αναγνώσεις.
Λαμβάνοντας υπόψη μας τις περιπτώσεις μέσα στη δομή επανάληψης που βγαίνουμε εκτός του πίνακα και δεν είναι αληθής η συνθήκη για την ανάγνωση θα έχουμε τελικά:

Αναγνώσεις αρχ. εικόνας:	$n^2*(2*filterR+1)-2*filter*n$
Αναγνώσεις αρχ. εικόνας/buffer:	$2*(n^2*(2*filterR+1)-2*filter*n)$
Αναγνώσεις φίλτρου:	$2*(n^2*(2*filterR+1)-2*filter*n)$

II. Σε κάθε επανάληψη εφόσον ισχύει η συνθήκη πραγματοποιείται μία προσπέλαση στην μνήμη για ανάγνωση της εικόνας και μία για ανάγνωση του φίλτρου. Υπολογίζεται το γινόμενο των δύο και αθροίζεται με μία μεταβλητή. Έχουμε λοιπόν 2 αναγνώσεις, 2 πράξεις κινητής υποδιαστολής και λόγο ίσο με 1.

8.Divergence και padding

Στους κώδικες των ερωτημάτων (3α) και (4) ο κώδικας πάσχει απο το φαινόμενο του divergence. Εφαρμόζουμε το padding στον πίνακα μεγαλώνοντας την κάθε διάστασή του κατά 2 * filter_radius και στην συνέχεια αρχικοποιούμε τα εξωτερικά έξτρα κελιά του με μηδενικά(zero padding).
Χρησιμοποιόντας την τεχνική του padding λύνεται το πρόβλημα του divergence καθώς κατά την εκτέλεση των kernel κάθε thread θα εκτελεί ακριβώς το ίδιο κομμάτι κώδικα χωρίς να ελέγχει αν βρίσκεται εντός ή εκτός του πίνακα-εικόνας.

Η διαφορά που παρατηρούμε στους χρόνους είναι αμελητέα. Αυτό οφείλεται στο γεγονός οτι είναι λίγα τα στοιχεία σε σχέση με το μέγεθος της εικόνας που ικανοποιούν την συνθήκη του **if** και κατά συνέπεια θα εκτελέσουν διαφορετικό κομμάτι κώδικα.