ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ Η/Υ

ECE415 - ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΟΥ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ

Lab 3 - Εισαγωγή στην CUDA

Φοιτητές:
Καραγεώργος Νικόλαος 02528, nkarageorgos@uth.gr
Λαμπρινός Ισίδωρος 02551, ilamprinos@uth.gr

Δεκέμβριος 2021

Περιεχόμενα

Εισαγωγή

Η συνέλιξη (convolution) βρίσκει εφαρμογή σε πλήθος εφαρμογών της μηχανικής και των μαθηματικών. Πολλά φίλτρα μετασχηματισμού εικόνας στην ουσία υλοποιούνται ως συνελίξεις, όπως για παράδειγμα το φίλτρο Gaussian blur. Στο πεδίο της επεξεργασίας εικόνας, ένα φίλτρο συνέλιξης είναι απλά το σημείο-προς-σημείο γινόμενο των βαρών του φίλτρου με τα pixels της εικόνας εισόδου, εντός ενός παραθύρου που περικλείει κάθε pixel εξόδου.

Στην παρούσα εργασία θα ασχοληθούμε με ένα διαχωρίσιμο δισδιάστατο φίλτρο συνέλιξης που εφαρμόζεται σε μία NxN εικόνα. Πιο συγκεκριμένα, θα υλοποιήσουμε τον κώδικα του παραπάνω φίλτρου σε CUDA και θα πραγματοποίησουμε διάφορα πειράματα πάνω στην υλοποιήση του σε GPU ώστε να απαντήσουμε τα ερωτήματα της εργασιάς όσον αφόρα την απόδοση του και άλλες παραμέτρους.

Όλη η ανάπτυξη έγινε στο σύστημα inf-mars1 (10.64.82.31) το οποιο διαθέτει μία κάρτα GTX690, με 2 chips, ενώ τελικές μετρήσεις εγίναν στο πιο ισχυρό csl-artemis (10.64.82.65) το οποίο διαθέτει μία κάρτα Tesla K80 η οποία έχει 2 GK210 GPU chips

Βήμα 0

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 2 CUDA Capable device(s)

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

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.4, CUDA Runtime Version
    = 11.5, NumDevs = 2
Result = PASS
```

Βήμα 1

Μελετάμε τις παραμέτρους εκτέλεσης του compiler nvcc, γράφοντας nvcc –help σε ένα τερματικό. Για τη μεταγλώτιση του κώδικα .cu χρησι-

μοποιούμε τον nvcc compiler με το μέγιστο βαθμό βελτιστοποιήσεων (-O4).

Βήμα 2

Σε αυτό το βήμα ζητείται να γράψουμε κώδικα που εκτελεί την συνέλιξη δύο διαστάσεων σε μια εικόνα διαστάσεων NxN στη GPU με βάση τον κώδικα που μας δίνεται για τη CPU.

Σε αυτόν τον κώδικα παρατηρούμε πως μπόρει να πραγματοποιηθεί μια σημαντική βελτιστοποίηση loop invariant code motion.

Η έκφραση:

$$h\_Dst[y * imageW + x] = sum$$

στην συνάρτηση για την συνέλιξη κατα γραμμές αλλά και στην συνάρτηση για την συνέλιξη κατά γραμμές βρίσκεται μέσα σε μία for με μετρητή k. Ωστόσο όπως παρατηρούμε δεν εξαρτάται κανένας όρος της άπο το k οπότε την τοποθετούμε εκτός της for αυτής.

Όσον αφορά την υλοποίηση για την GPU σε αυτό το βήμα ζητείται να χρησιμοποιηθούν τα threads ενός μόνο block ώστε κάθε thread να υπολογίζει ένα στοιχείο της εικόνας-αποτελέσματος. Κάθε thread της GPU δουλεύει για μία διαφορετική θέση στον πίνακα των αποτελεσμάτων οπότε τα threads δεν χρειάζεται να συγχρονιστούν μεταξύ τους. Για να εντοπίσουν την θέση που δουλεύουν στο block υπάρχουν οι δείκτες threadIdx.x και threadIdx.y. Ο πρώτος δείχνει την οριζόντια συντεταγμένη του thread στο block ενώ ο δεύτερος την κατακόρυφη. Έτσι δεν χρειάζεται να χρειάζεται να χρησιμοιηθεί οι δύο εξωτερικές for με τους μετρητές x και y και αυτοί αντικαθιστούνται απο τους threadIdx.x και threadIdx.y αντίστοιχα.

Βήμα 3

3α. Μέχρι ποιο μέγεθος εικόνας (πειραματιστείτε πάντα με μεγέθη εικόνας που είναι δυνάμεις του 2 και μεγαλύτερες από το μήκος του φίλτρου) μπορείτε να υποστηρίξετε χωρίς να συμβεί κάποιο σφάλμα χρόνου εκτέλεσης; Γιατί συμβαίνει αυτό το σφάλμα;

Το μέγιστο μέγεθος εικόνας που μπορεί να υποστηριχθεί χωρίς να συμβεί κάποιο σφάλμα χρόνου εκτέλεσης είναι 32x32. Όταν δοκιμάζουμε σαν είσοδο μέγεθος εικόνας 64x64 εμφανίζεται το παρακάτω:

```
Enter filter radius : 16
Enter image size. Should be a power of two and greater than 33 : 64
Image Width x Height = 64 x 64

Allocating and initializing host arrays...
Allocating and initializing device arrays...
CPU computation...
GPU computation...
Cuda error Convolution2D.cu: 247: 'invalid configuration argument'.
```

Figure 1: Σφάλμα χρόνου εκτέλεσης για είσοδο εικόνα μεγέθους 64x64

Αυτό μας φαίνεται απόλυτα λογικό καθώς όπως παρατηρήσαμε απο το deviceQuery νωρίτερα κάθε block διαθέτει 1024 threads. Στο συγκεκριμενο ερώτημα χρησιμοποιούμε μόνο ένα block άρα εχούμε διαθέσιμα 1024 threads για μία εικόνα με imageW*imageH pixels οπου imageW=imageH οπότε και μαθηματικά μπορουμέ να φτάσουμε μέχρι 32x32=1024.

3β) Για το μέγιστο μέγεθος εικόνας που μπορείτε να υποστηρίξετε, ποια είναι η μέγιστη ακρίβεια, σε σχέση με το μέγεθος του φίλτρου, που μπορεί να υποστηριχθεί χωρίς να παρουσιάζονται σφάλματα σύγκρισης;

Ως ακρίβεια ορίζουμε την μέγιστη απόλυτη διαφόρα μεταξύ των αποτελεσμάτων της συνέλιξης στη CPU και τη GPU (σφάλμα).

Για το μέγιστο μέγεθος εικόνας που μπορεί να υποστηριχθεί δηλαδη 32x32 το μέγιστο μέγεθος της ακτινάς του φίλτρου που μπορέι να χρησιμοποιηθεί είναι 15.

Ακτίνα φίλτρου	Μέγιστο απόλυτο σφάλμα
1	0.007812
2	0.015625
3	0.0625
4	0.1875
5	0.25
6	0.5
7	0.375
8	0.75
9	1
10	1
11	1.5
12	1
13	1.5
14	1
15	1

Παρατηρούμε οτί η μέγιστη ακρίβεια δηλαδή το μικρότερο απόλυτο σφάλμα(0.007812) παρατηρείται με την μικρότερη ακτίνα φίλτρου που είναι 1 και αρα για συνολικό μήκος φίλτρου 3.

Βήμα 4

Σε αυτό το ερώτημα χρησιμοποιούμε πολλαπλά τετράγωνα blocks που είναι οργανωμένο σε ένα τετράγωνο grid. Τα block αυτά όπως είδαμε και

προηγουμένως διαθέτουν 1024 threads άρα εχουν διαστάσεις 32x32. Τα threads πλέον για να βρούνε το στοιχείο της είκονας το οποίο τους έχει ανατεθεί για υπολοσγισμό χρησιμοποιούν και τα blockDim.x, blockIdx.x, blockDim.y και blockIdx.y.

Το μέγιστο μέγεθος εικόνας που μπορεί υποστηριχθεί πλέον στο csl-artemis (10.64.82.65) ειναι 16384x16384. Μετά απο αυτό το μέγεθος εικόνας εμφανίζεται:

```
Enter filter radius : 4
Enter image size. Should be a power of two and greater than 9 : 32768
Image Width x Height = 32768 x 32768

Allocating and initializing host arrays...
Allocating and initializing device arrays...
Error while allocating device memory.
```

Figure 2: Σφάλμα χρόνου εκτέλεσης για είσοδο εικόνα μεγέθους 32768x32678

Αυτό συμβαίνει γιατί πλέον δεν τα υπάρχει αρκετή μνήμη στη GPU για τόσο μεγάλα μεγέθη.

Βήμα 5

5a)

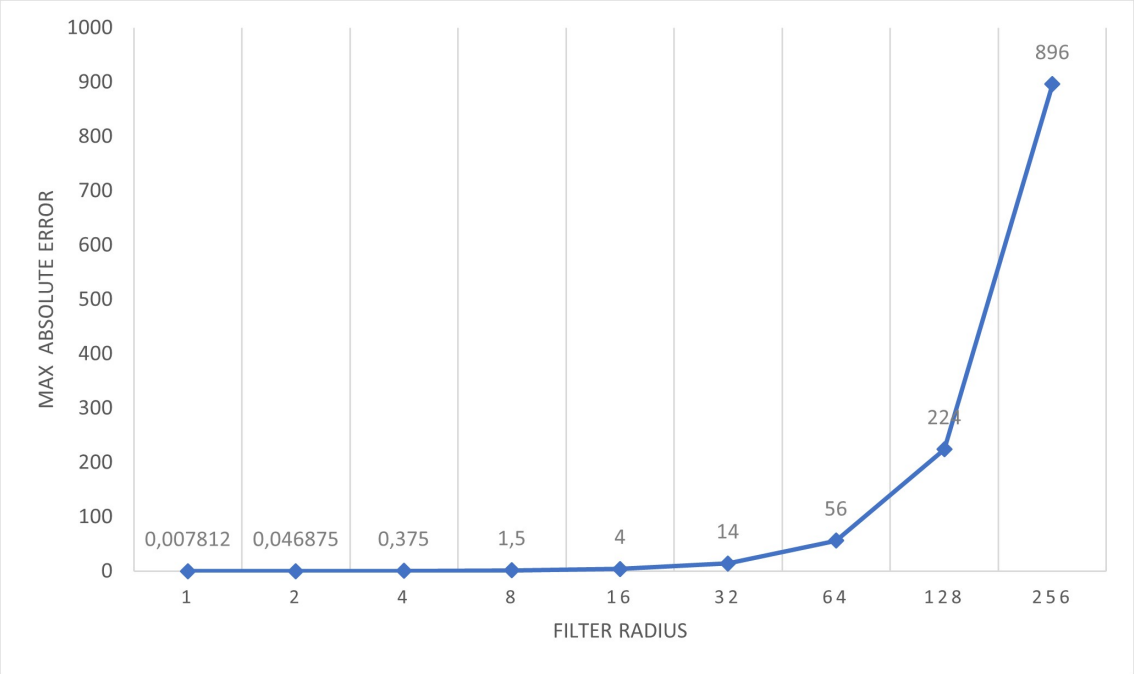

Figure 3: Ακρίβεια σε σχέση με την ακτίνα του φίλτρου για εικόνα μεγέθους 16384x16384

Όπως παρατηρούμε απο τα παραπάνω υπάρχει μία εκθετική αύξηση της μέγιστης ακρίβειας των αποτελεσμάτων της GPU σε σχέση με τη CPU όσο ανεβαίνει το μέγεθος της ακτίνας του φίλτρου. Αυτό συμβαίνει καθώς οι GPU πετυχαίνουν πολύ καλύτερες επιδόσεις στον χρόνο εκτέλεσης καθώς κάνουν fused multiply-add (FMAD) υπολογισμούς, με συνέπεια όμως να θυσιάζουν κάποια ακρίβεια. Ειδικά στην συγκεκριμένη περίπτωση χρησιμοποιουνται αριθμοί κινητής υποδιαστολής απλής ακρίβειας (float). Αυτοί οι αριθμοί κάνουν σημαντικά προσεγγιστικά λάθη με αποτέλεσμα να μην πετυχαίνεται η ακρίβεια που θα είχαμε αν χρησιμοποιούνταν αριθμοί κινητής υποδιαστολής διπλής ακρίβειας (doubles).

5b)

Διαστάσεις Εικόνας	**Μέσος Χρόνος Εκτέλεσης CPU για floats**	**Μέσος Χρόνος Εκτέλεσης GPU για floats**
64x64	0,000364805	0,000121654
128x128	0,001507745	0,000237222
256x256	0,006107532	0,000673197
512x512	0,024851352	0,002491741
1024x1024	0,101809057	0,008731715
2048x2048	0,485297299	0,033883184
4096x4096	2,326409672	0,149977916
8192x8192	9,183927318	0,542511845
16384x16384	37,31040972	2,116709781

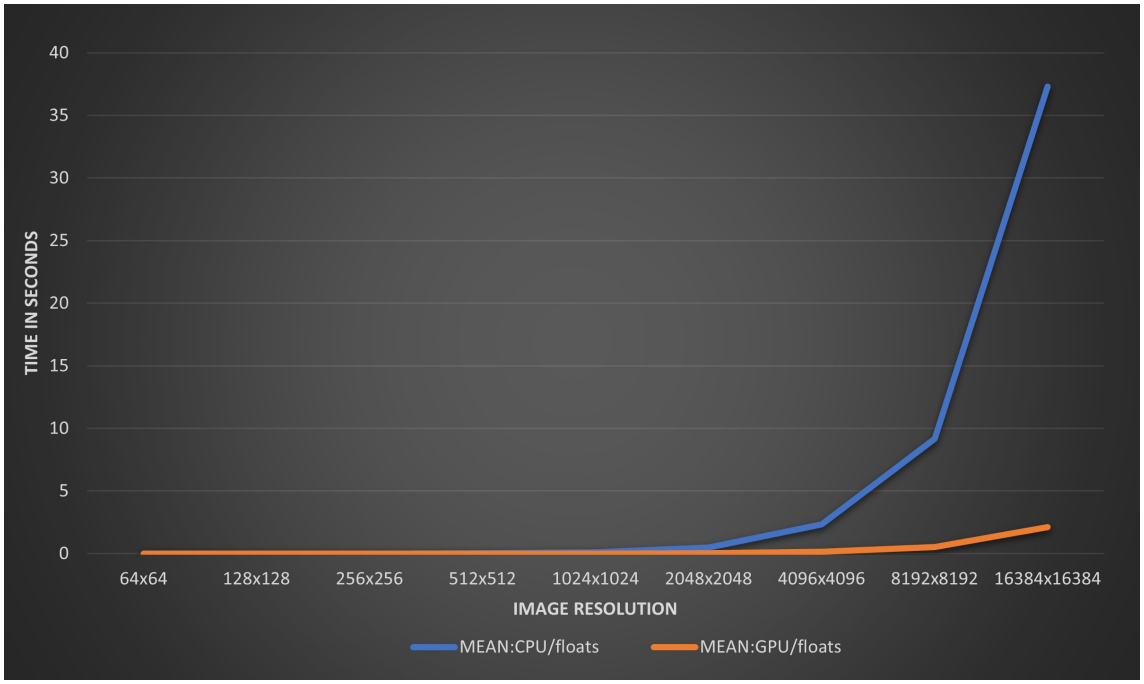

Figure 4: Χρόνος εκτέλεσης για τύπο floats σε CPU και GPU

Παρατηρούμε πως οι επιδόσεις του χρόνου εκτέλεσης στην GPU ειναι πολύ καλύτερες σε σχεση με αυτές στην CPU. Αυτό γίνεται ακόμη πιο ξεκάθαρο όσο μεγαλώνει το μέγεθος της εικόνας-εισόδου.

Βήμα 6

Διαστάσεις Εικόνας	Μέσος Χρόνος Εκτέλεσης CPU doubles	Μέσος Χρόνος Εκτέλεσης GPU doubles
64x64	0,000408155	0,000149894
128x128	0,001703403	0,00032137
256x256	0,00669445	0,000950256
512x512	0,027434598	0,00323968
1024x1024	0,111931962	0,011528599
2048x2048	0,567745006	0,044662288
4096x4096	2,467726286	0,200996171
8192x8192	9,756948556	0,790095544
16384x16384	39,51873891	3,072551799

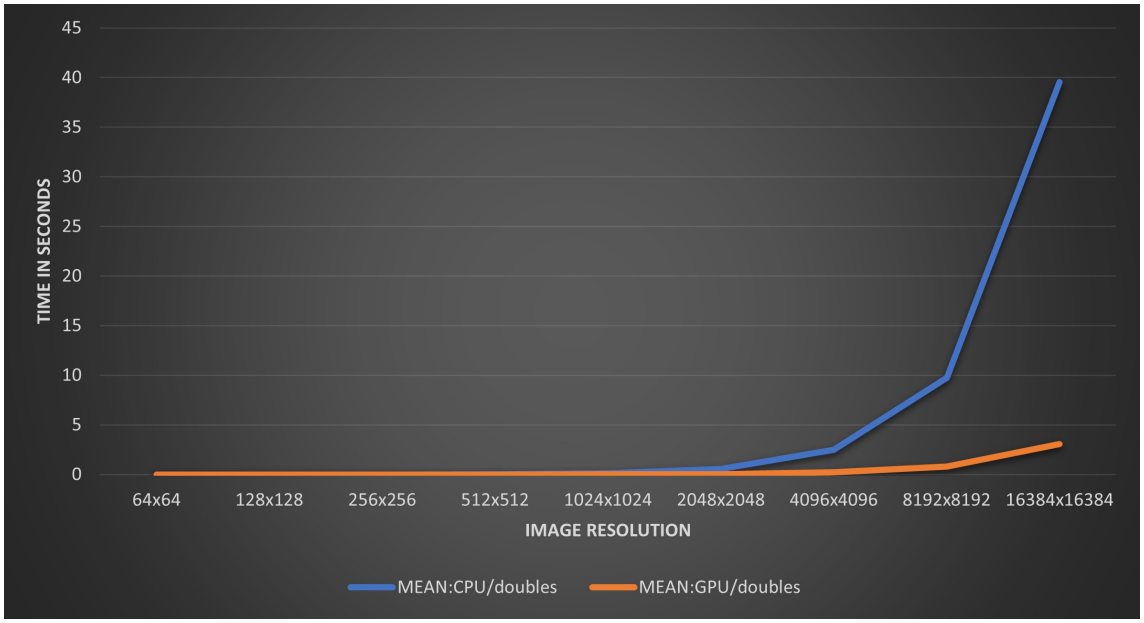

Figure 5: Χρόνος εκτέλεσης για τύπο doubles σε CPU και GPU

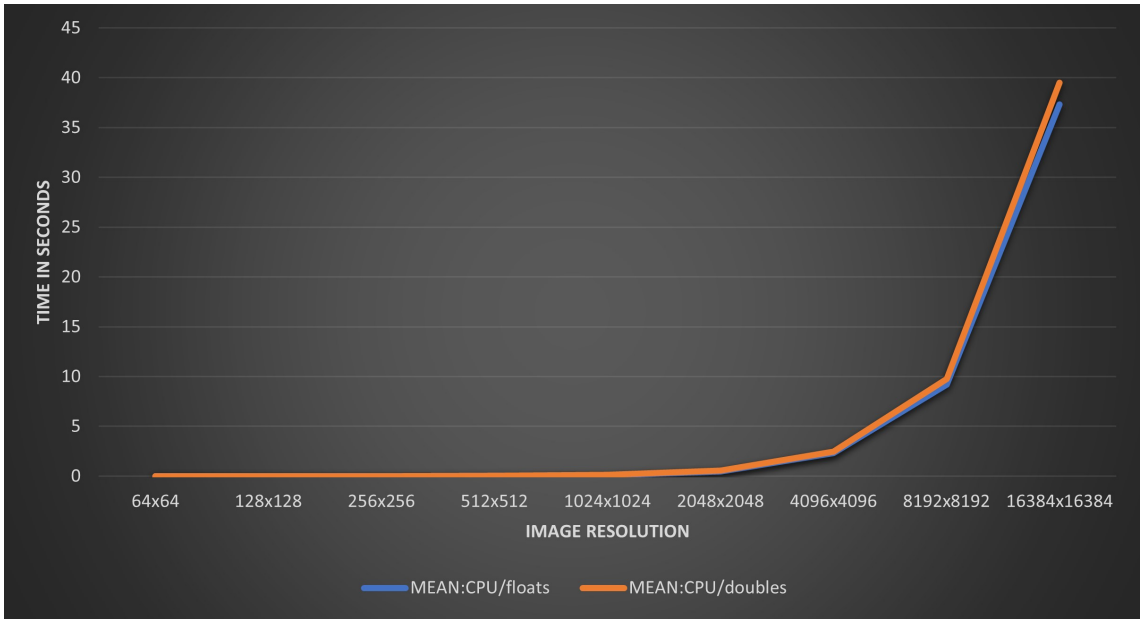

Figure 6: Χρόνος εκτέλεσης για τύπους floats και doubles σε CPU

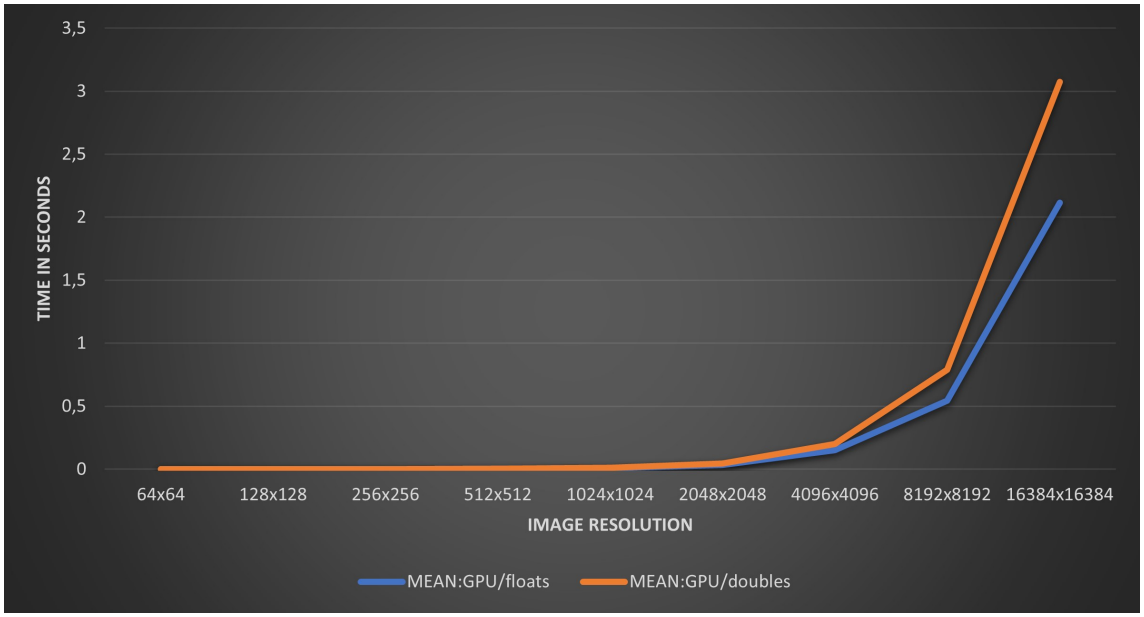

Figure 7: Χρόνος εκτέλεσης για τύπους floats και doubles σε GPU

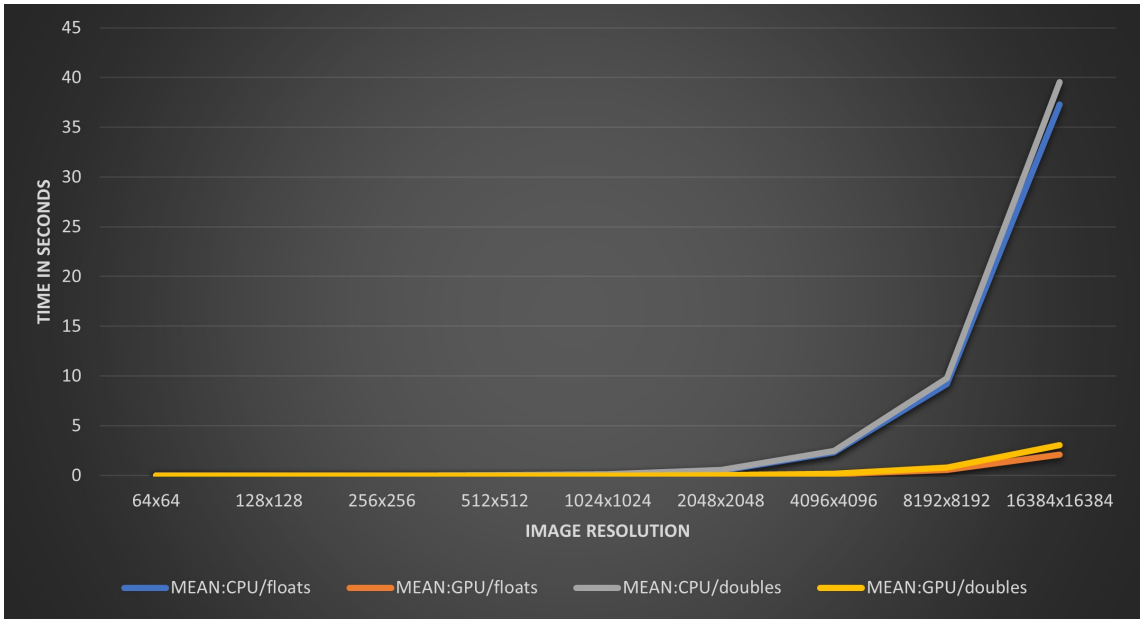

Figure 8: Χρόνος εκτέλεσης για τύπους floats και doubles σε CPU και GPU

Χρησιμοποιώντας στοιχεία κινητής υποδιαστολής διπλής ακρίβειας (double) παρατηρούμε ότι πετυχαίνεται πάρα πολύ καλή ακρίβεια αφού δοκιμάζοντας μέχρι και μέγεθος ακτίνας φίτρου ίσο με 128 το μέγιστο απόλυτο σφάλμα παραμένει 0. Αυτό συμβαίνει καθώς πλέον χρησιμοποιούνται 64 bits για την αναπαράσταση των αριθμών κινητής υποδιαστολής ενώ προηγουμένως με floats χρησιμοποιούνταν 32 bits οπότε γίνονται πολύ λιγότερα προσεγγιστικά λάθη. Ωστόσο, παρατήρούμε μια μικρή αύξηση στο χρόνο εκτέλεσης , το οποίο είναι λογικό αφού πλέον έχουμε να κάνουμε υπολογισμούς με αριθμούς μεγαλύτερου μεγέθους.

Βήμα 7

α) Πόσες φορές διαβάζεται κάθε στοιχείο της εικόνας εισόδου και του φίλτρου κατά την εκτέλεση του kernel;

Όσον αφορά την εικόνα εισόδου είναι ξεκάθαρο ότι τα στοιχεία-pixels που βρισκόνται εσωτερικά δηλαδή πιο κοντά στο κέντρο θα διαβάστουν περισσότερες φορές στην πράξη της συνέλιξης από αυτα που βρίσκονται στις άκρες της εικόνας. Πιο συγκεκριμένα, το κάθε thread αναλαμβάνει να υπολογίσει το αποτέλεσμα-pixel διαβάζοντας 2filterR+1 στοιχεία γύρω απο το αρχικό pixel. Αυτό συνεπάγεται ότι ένα στοιχείο μπορεί να δια-

βαστεί το πολύ 2filterR+1 κατά την πράξη μίας συνέλιξης.

Μετά απο πειραματισμούς με μεγέθη φίτρου και εικόνας καταλήγουμε στους γενικούς τύπους οπου καθε pixel(x,y) με x ∈ [0,imageW-1] και y ∈ [0,imageH-1] διαβάζεται:

Στη συνέλιξη κατά γραμμές:

$$Row\_Reads_{(x,y)} = \begin{cases} filterR + x + 1, & x \leq filterR \\ 2filterR + 1 & filterR, < x < imageW - filterR \\ filterR + imageW - x & x, \geq imageW - filterR \end{cases}$$

Στη συνέλιξη κατά στήλες:

$$Col\_Reads_{(x,y)} = \begin{cases} filterR + y + 1, & y \leq filterR \\ 2filterR + 1 & filterR, < y < imageW - filterR \\ filterR + imageW - y, & y \geq imageW - filterR \end{cases}$$

Συνολικά το κάθε στοιχείο διαβάζεται:

$$Reads_{(x,y)} = Col\_Reads_{(x,y)} + Row\_Reads_{(x,y)}$$

Όσον αφορά το πόσες φορές διαβάζονται τα στοιχεία του φίλτρου είναι πάλι ευδιάκριτο πως τα κεντρικό στοιχείο του φίλτρου, που βρίσκεται στη θέση filterR από τα 2*filterR+1 στοιχεία, θα διαβαστεί όσες φορές όσες είναι και τα στοιχεία της εικόνας εισόδου δηλαδή imageH*imageW. Ενώ όσο πιο εξωτερικά είναι τα στοιχεία στο φίλτρο θα διαβάζονται λιγότερες φόρες. Πιο συγκεκριμένα, ένα στοιχείο που βρίσκεται στη θέση filterR-n οπου n ∈ [0,R] του φίλτρου θα διαβαστεί (imageW-n)*imageH φορές στη συνέλιξη κατα γραμμές και (imageH-n)*image φορές στη συνέλιξη κατά στήλες.

Συνολικά:

$$Reads_{(n)} = (imageW - |filterR - n|) * imageH + (imageW - |filterR - n|) * imageH$$

Γνωρίζοντας ότι imageW=imageH:

$$Reads_{(n)} = 2(imageW - |filterR - n|) * imageW$$

b)Ποιός είναι ο λόγος προσπελάσεων μνήμης προς πράξεις κινητής υποδιαστολής; Θεωρήστε τους πολλαπλασιασμούς και τις προσθέσεις ως ξεχωριστές πράξεις και αγνοείστε την αποθήκευση του αποτελέσματος. Μετρήστε μόνο τις αναγνώσεις από την global memory της GPU ως προσπελάσεις μνήμης.

Παρατηρούμε οτι για κάθε πράξη πολλαπλασιασμού και πρόσθεσης στη συνέλιξη γίνονται δύο προσπελάσεις στην μνήμη , μία για την ανάγνωση του στοιχείου του φίλτρου και μία για την ανάγνωση του στοιχείου της εικόνας εισόδου που είναι για υπολογισμό. Επομένως, συμπεραίνουμε πως ο λόγος προσπελάσεων μνήμης προς πράξεις κινητής υποδιαστολής είναι 1.

Βήμα 8

Διαστάσεις Εικόνας	**Μέσος Χρόνος Εκτέλεσης CPU χωρίς padding**	**Μέσος Χρόνος Εκτέλεσης CPU με padding**
64x64	0,000364805	0,000225059
128x128	0,001507745	0,000861735
256x256	0,006107532	0,003696144
512x512	0,024851352	0,014604245
1024x1024	0,101809057	0,058377305
2048x2048	0,485297299	0,240614454
4096x4096	2,326409672	1,054468082
8192x8192	9,183927318	4,283031374
16384x16384	37,31040972	17,3529493

Διαστάσεις Εικόνας	**Μέσος Χρόνος Εκτέλεσης GPU χωρίς padding**	**Μέσος Χρόνος Εκτέλεσης GPU με padding**
64x64	0,000121654	0,000161725
128x128	0,000237222	0,000290666
256x256 2	0,000673197	0,000769075
512x512	0,002491741	0,002616608
1024x1024	0,008731715	0,009078586
2048x2048	0,033883184	0,034716791
4096x4096	0,149977916	0,14242692
8192x8192	0,542511845	0,535792637
16384x16384	2,116709781	2,117604208

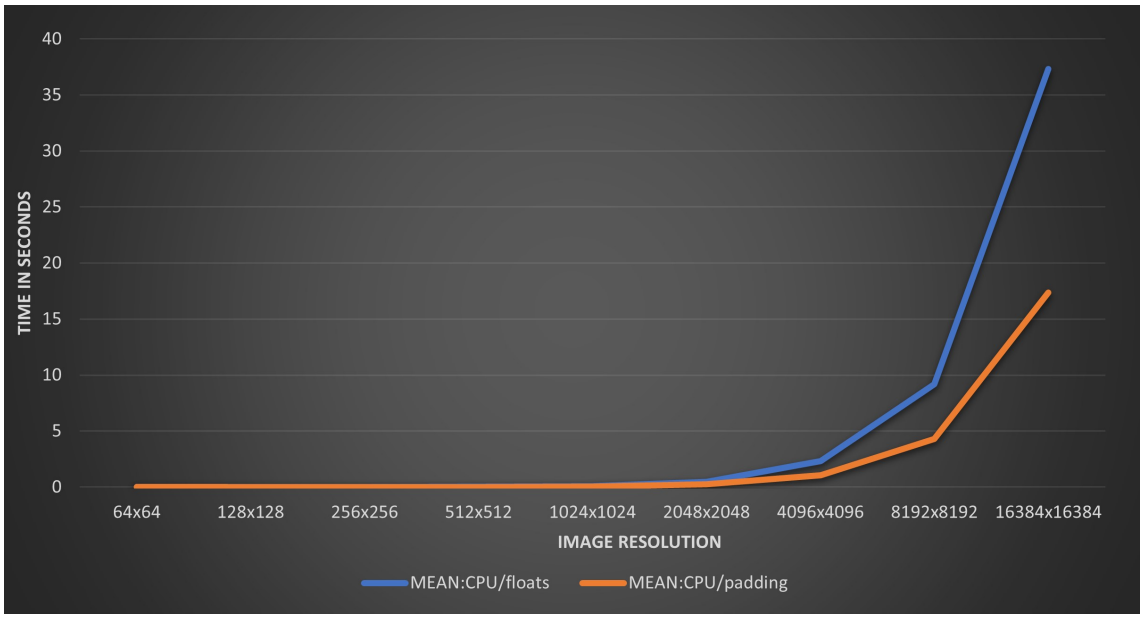

Figure 9: Χρόνος εκτέλεσης σε CPU με και χωρίς padding

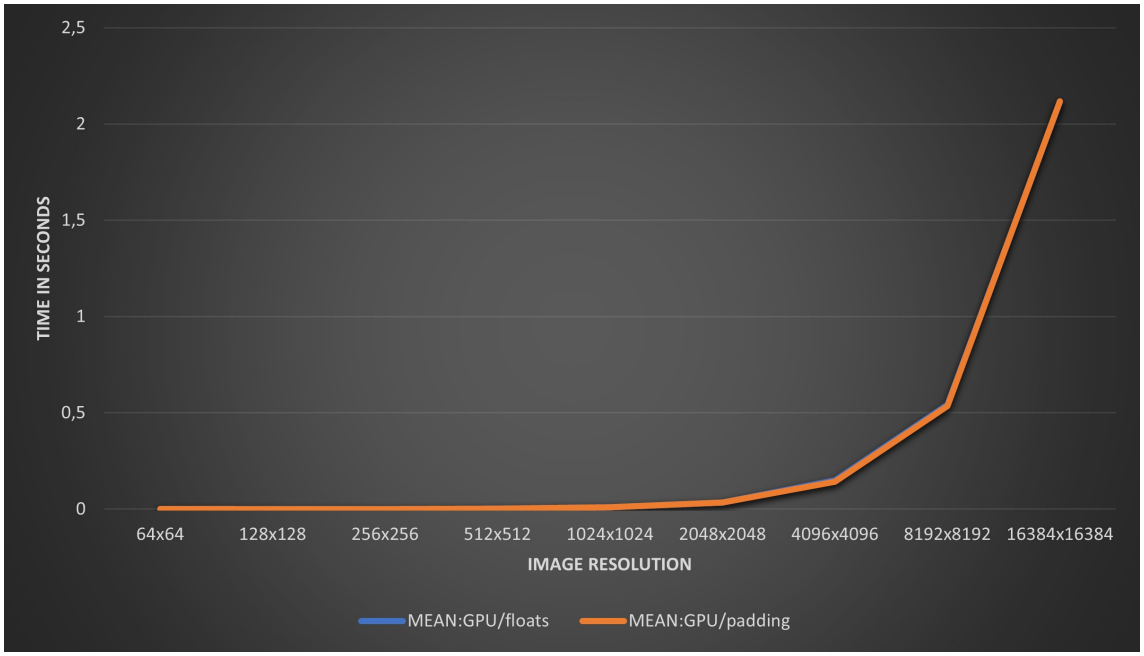

Figure 10: Χρόνος εκτέλεσης σε GPU με και χωρίς padding

Χρησιμοποιώντας padding καταφέρνουμε να απαλλαχθούμε εντελώς από τις if που ελέγχουν αν το φίλτρο του πίνακα βρίσκεται εκτός πίνακα. Αυτό όπως παρατηρείται παραπάνω είχε ως αποτέλεσμα την σημαντική πτώση του χρόνου εκτέλεσης για τη CPU ειδικά οσό ανεβάινει το μέγεθος της εικόνας. Για τη GPU ωστόσο δεν παρατηρείται καποιά σημαντική διαφόρα στους χρόνους εκτέλεσης.