ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ Η/Υ

ECE415 - ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΜΟΥ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ

Lab 4 - Βελτιστοποιήσεις προγράμματος CUDA

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

Ιανουάριος 2022

Περιεχόμενα

Εισαγωγή

Στην παρούσα εργασία μας ζητήθηκε να πραγματοποιήσουμε βελτιστοποίησεις στον κώδικα CUDA της προηγούμενης εργασίας ο οποίος εφαρμόζει, με χρήση συνέλιξης, ένα διαχωρίσιμο δισδιάστατο φίλτρο πάνω σε ένα δισδιάστατο πίνακα (ο οποίος υποθέσαμε ότι αντιστοιχεί σε μια εικόνα). Πιο συγκεκρίμενα, σε αυτή την άσκηση κληθήκαμε να βελτιστοποιήσουμε τον προηγούμενο κώδικα, να τον εκτελέσουμε για πίνακες πολύ μεγαλύτερου μεγέθους και να προσπαθήσουμε να επικαλύψουμε μεταξύ τους διαφορετικούς υπολογισμούς καθώς και υπολογισμό με μεταφορές δεδομένων. Χρησιμοποιείται η έκδοση του κώδικα με padding που χρησιμοποιεί doubles.

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

Βελτιστοποιήσεις του κώδικα σε GPU:

-Caching του πίνακα του φίλτρου d_Filter

Τοποθετήσαμε τον πίνακα του φίλτρου d_Filter σε κάποια cached memory ώστε να μειωθούν οι ακριβές προσπελάσεις στην global memory. Ειδικότερα, επιλέξαμε την constant memory η οποία υποστηρίζει μόνο αναγνώσεις καθώς ο συγκεκριμένος πίνακας παραμένει σταθερός και χρησιμοποιείται από όλα τα blocks για αναγνώσεις. Δεν επιλέξαμε την shared memory καθώς η συγκεκριμένη υποστηρίζει και εγγράφες οι οποίες δεν χρειάζονταν σε αυτό το σημείο και θα μπορούσαν να αξιοποιηθούν αργότερα.

-Tiling της εικόνας εισόδου στην shared memory

Η εικόνα εισόδου διασπάται σε μικρότερα τετράγωνα κομμάτια τα οποία αναθέτονται το καθένα σε ένα block για υπολογισμό. Τα threads του ίδιου block έχουν πρόσβαση στην ίδια shared memory και χρησιμοποιούν συνολικά για τους υπολογισμούς τους (blockDim.x + 2filterR) * blockDim.y στοιχεία του κομματιού που τους ανατέθηκε. Λαμβάνοντας υπόψιν τα παραπάνω αποφασίσαμε να χρησιμοποιήσουμε την shared memory ως cache. Έτσι κάθε φορά όταν τα threads αρχίζουν την εκτέλεση τους φέρνουν κάποια διαφορετικά στοιχεία του κομματιού τους της αρχικής εικόνας από την global memory στην shared. Στο τέλος όλα τα threads έχουν συνεργαστεί και έχουν φέρει μάζι όλα τα στοιχεία του κομματιού που έχει ανατέθει στο block τους στην shared memory, τα οποία θα χρειαστούν το καθένα αργότερα για τους υπολογισμούς τους και θα τα κάνουν caching (βελτιστοποίηση coalescing). Αυτή η βελτιστοποίηση ωστόσο μας περιορίζει στο μέγεθος της ακτίνας του φίλτρου (filterR) που μπορούμε να χρησιμοποίησουμε αφού το μέγεθος της shared memory κάθε block είναι αρκετά περιορισμένο, όπως βλέπουμε από το deviceQuery 49152 bytes. Πρέπει να ισχύει:

(blockDim.x + 2filterR) * blockDim.y * sizeof(double) $\leq$ Size of shared memory

Δηλαδή πρέπει filterR ≤ 80.

Επίσης με τον τρόπο με τον οποίο έχουμε υλοποίησει την μεταφορά των κομματιών στην shared memory το μικρότερο μέγεθος ακτίνας φίλτρου που μπορεί να χρησιμοποιηθεί είναι 16.

-Μείωση των ορισμάτων των συναρτήσεων του kernel

Αφαιρούμε από όρισμα μία απο τις δύο διαστάσεις της εικόνας αφού η εικόνα είναι τετράγωνη και είναι ίδιες. Αφαιρούμε επίσης την ακτίνα του φίλτρου από όρισμα και την ορίζουμε με #define κάθως την χρείαζομαστε και για τον ορισμό του πίνακα d_Filter στην constant memory ο οποίος επίσης αφαιρείται απο όρισμα της συνάρτησης.

GPU activities	Αρχική έκδοση	Βελτιστοποιημένη έκδοση
CUDA memcpy DtoH	407.20	408.77
CUDA memcpy HtoD	215.90	201.83
convolutionRowGPU	197.00	43.520
convolutionColumnGPU	167.29	32.210

Table 1: GPU Activities Execution Time

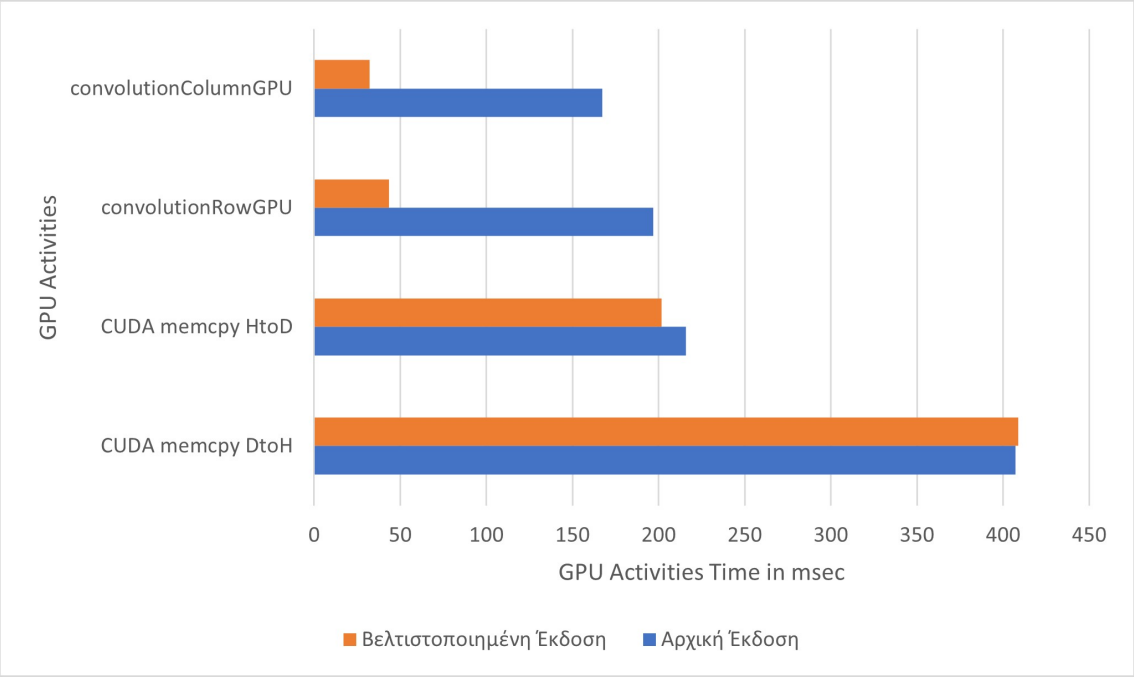

Figure 1: GPU Activities Time (msec)

GPU activities	Αρχική έκδοση	Βελτιστοποιημένη έκδοση
CUDA memcpy DtoH	41.00%	57.47%
CUDA memcpy HtoD	21.62%	30.17%
convolutionRowGPU	20.02%	6.84%
convolutionColumnGPU	17.01%	5.04%

Table 2: GPU Activities Execution Time (percentage)

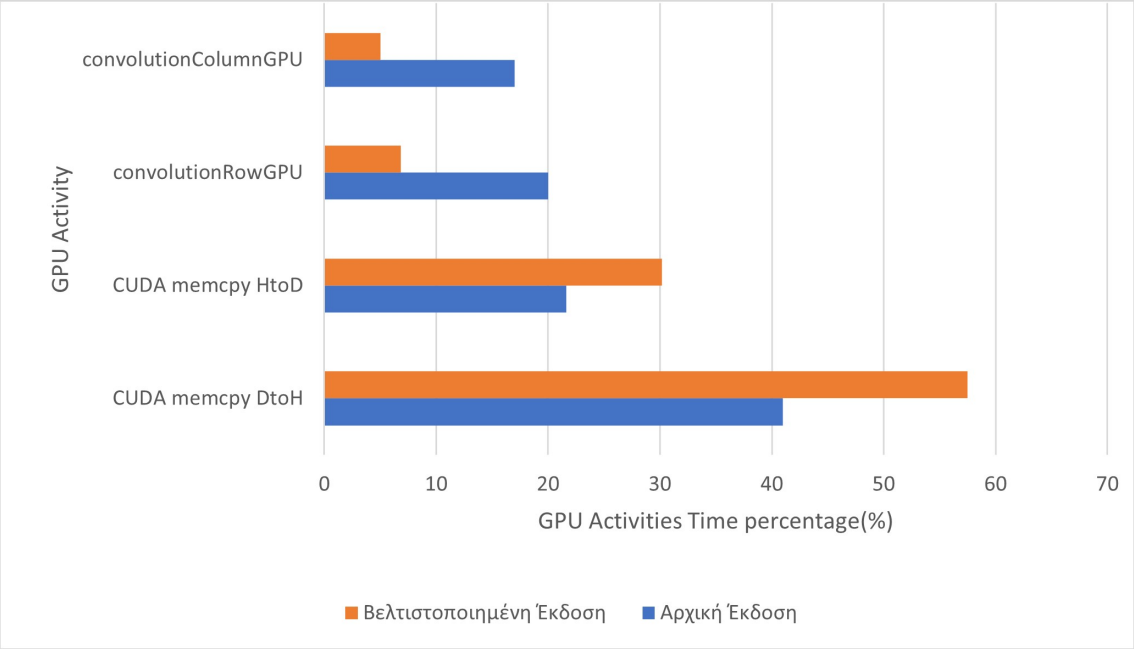

Figure 2: GPU Activities Time Percentage(%)

Βήμα 2

Κάνοντας profiling για διαδορετικές τιμές φίλτρου παρατηρούμε ότι όσο αυξάνεται το μέγεθος του φίλτρου τόσο η εκτέλεση των kernels ακριβαίνει σε χρόνο. Το ίδιο ωστόσο δεν ισχύει για τις μεταφορές δεδομένων, οι οποίες κυμαίνονται στις ίδιες τιμές ανεξαρτήτως ακτίνας φίλτρου και δε φαίνεται να επηρεάζονται απ' αυτό. Για το πείραμα χρησιμοποιήσαμε εικόνα μεγέθους 8192x8192.

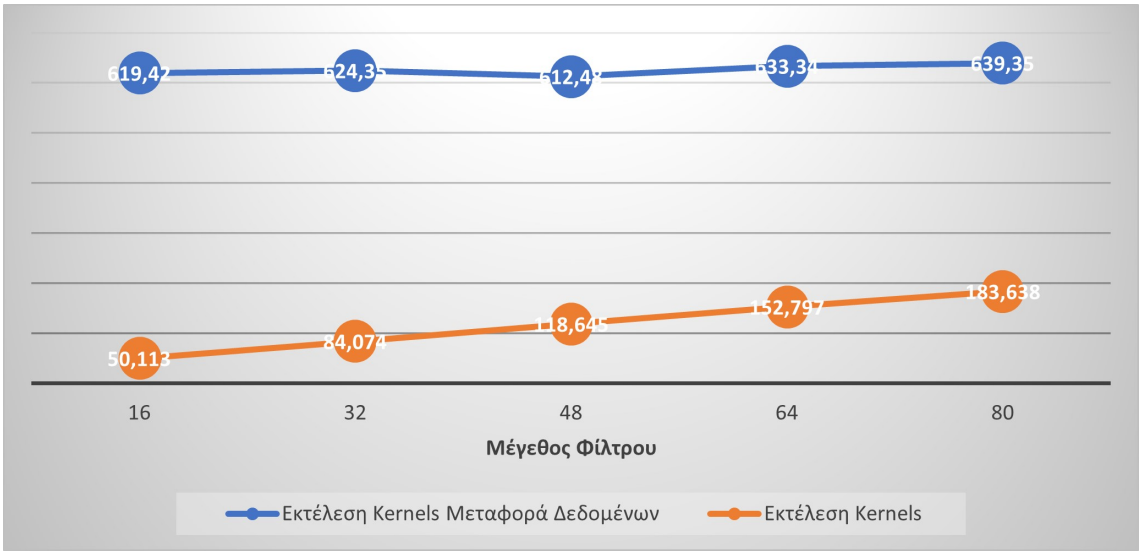

Figure 3: Kernel and Data Transfer diagram (msec)

Filter Radius	Εκτέλεση Kernel	Μεταφορά Δεδομένων
16	50,113	619,42
32	84,074	624,35
48	118,645	612,48
64	152,797	633,34
80	183,638	639,35

Table 3: Kernel and Data Transfer Table

Βήμα 3

Για να καταφέρουμε να υποστηρίξουμε πολύ μεγαλύτερες εικόνες θα πρέπει η εκτέλεση των kernels να είναι blocked, με τέτοιο τρόπο ώστε όλα τα δεδομένα που απαιτούνται για την εκτέλεση κάθε βήματος να χωράνε στη GPU. Για αυτό το λόγο η αρχική εικόνα σπάει σε μικρότερα κομμάτια και σε καθένα απο αυτά εφαρμόζεται ξεχωριστά η δισιάσταση συνέλιξη. Το μικρότερο κομμάτι της εικόνας που δίνεται για υπολογισμό στους kernels αποθηκεύεται σε ένα πίνακα σε μνήμη του host και μέτα με την cudaMemcpy μεταφέρεται σε μνήμη του device. Αφου ολοκληρωθεί ο υπολογισμός της δισδιάστατης συνέλιξης αυτού του κομματιού τότε περνάει στη θέση του στη μνήμη του device το επόμενο κομμάτι. Έτσι τώρα δεν μας περιορίζει πια η global memory του device άλλα η μνήμη του host καθώς έκει αποθηκέυεται η συνολική εικόνα.

Το μέγεθος του κομματιού (παράμετρος BLOCKING) μπορεί να μεταβληθεί, άλλα πρέπει να παίρνει τιμές που είναι δύναμη του 2 και να ναι μικρότερο από το μέγεθος της εικόνας. Στα πειράματα μας χρησιμοποιήσαμε μέγεθος BLOCKING 512.

Αυτό το βήμα είχε αρνητική επίδραση στον χρόνο εκτέλεσης, κάτι το οποίο είναι λογικό καθώς πλέον γίνονται πολύ περισσότερες κλήσεις των kernels και μεταφορές δεδομένων.

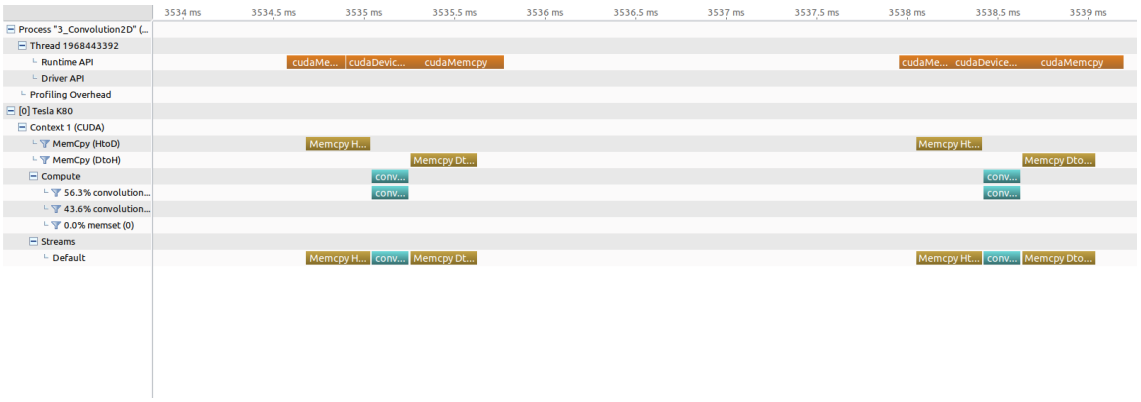

Figure 4: Profiling Row Convolution Step 3

Βήμα 4

Όπως παρατηρήσαμε απο το profiling στον κώδικα του προηγούμενου βήματος δεν υπάρχει καμία επικάλυψη ανάμεσα στον κώδικα που εκτελείται από διαφορετικούς kernels ή σε εκτέλεση κώδικα και μεταφορά δεδομένων. Σε αυτό το βήμα για να πετύχουμε επικάλυψη χρησιμοποιούμε 2 streams. Κάθε stream μετάφερει ασύγχρονα ένα κομμάτι της αρχικής εικόνας στο device με τη χρήσης της cudaMemcpyAsync πλέον, καλείται ο kernel για υπολογισμός της συνέλιξης και το αποτελέσμα έπειτα μεταφέται πάλι ασύγχρονα απο το device στον host.

Όπως είναι αναμενόμενο ο χρόνος εκτέλεσης μειώνεται απο το προηγούμενο βήμα αρκετά (για εικόνα 8192x8192 και ακτίνα φίλτρου 31 ο χρόνος πέφτει από 0.141 s σε 0.123 s) και όπως παρατηρούμε παρακάτω απο το στιγμιότυπο του profiling πλεόν υπάρχει επικάλυψη μεταξύ των εκτελέσεων των kernels και των μεταφορών δεδομένων.

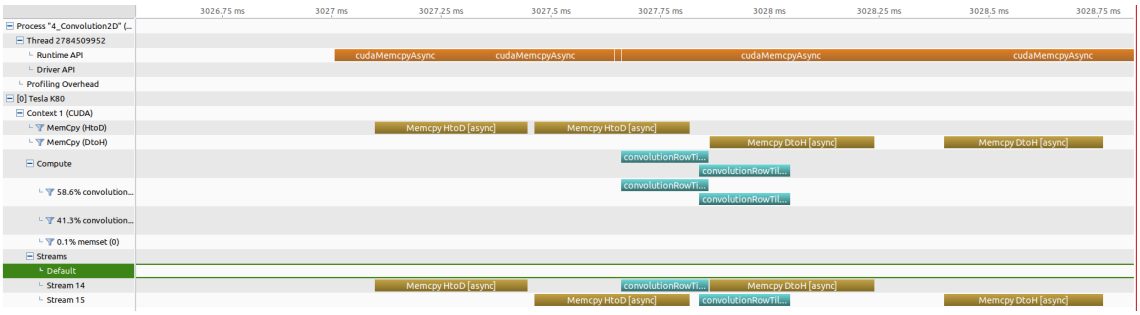

Figure 5: Profiling Row Convolution Step 4 with Streams

Βήμα 5

Το μέγιστο μέγεθος εικόνας που μπορεί να υποστηριχθεί πλέον είναι 65536x65536. Για εικόνα μεγέθους 131072x131072 εμφανίζεται το μήνυμα: Error while allocating host memory .Ο πόρος που μας περιορίζει πλέον δεν είναι η μνήμη του device αλλά η μνήμη του host η οποία με την εντολή cat/proc/meminfo βλέπουμε ότι είναι 128 GB.