3η εργασία

Σταμούλος Αλέξανδρος 02954

Γαλάνης Αχιλλέας 02941

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)  (13) 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 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 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)  (13) 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 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 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, Device0 = Tesla K80, Device1 = Tesla K80  Result = PASS |

2)

Ο κώδικας για την gpu είναι παρομοίως με αυτόν της CPU απλά η θέση του πίνακα αντί για το διπλό for-loop υπολογίζεται ως εξής

x = threadIdx.x

y = threadIdx.y

3α)

Το μέγιστο μέγεθος εικόνας που υποστηρίζεται είναι 32x32 αυτό είναι λογικό αφού χρησιμοποιούμε ένα block και από το devicequery βλέπουμε ότι το Maximum number of threads per block: 1024 = 32x32

3β)

|  |  |
| --- | --- |
| Μέγεθος  φίλτρου | Ακρίβεια σε  δεκαδικά ψηφιά |
| 1 | 2 |
| 2 | 1 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 (<5) |
| 9 | 0 (<5) |
| 10 | 0 (<5) |
| 11 | 0 (<5) |
| 12 | 0 (<5) |
| 13 | 0 (<5) |
| 14 | 0 (<5) |
| 15 | 0 (<5) |

Έχουμε μέγιστη ακρίβεια δυο δεκαδικά ψηφία για μέγεθος φίλτρου 1

4)

Έχουμε μέγεθος του block 32x32 και υπολογίζουμε το μέγεθος

grid = ceil( width /32 ) x ceil(height/32) \*για να υποστηρίζει και εικόνες μικρότερες του 32x32 . Τώρα το x και y :

x = blockIdx.x \* blockDim.x + threadIdx.x

y = blockIdx.y \* blockDim.y + threadIdx.y

Τέλος προσθέτουμε ένα if που ελέγχει αν τα threads είναι έξω από την εικόνα οπότε επιστρέφουν ( χρησιμοποιείται για εικόνες μικρότερες του 32x32)

5α)Χρησιμοποίησέ μέγεθος εικόνας 1024x1024 και ακτίνα φίλτρου δυνάμεις του 2

|  |  |
| --- | --- |
| Μέγεθος  φίλτρου | Ακρίβεια σε  δεκαδικά ψηφιά |
| 1 | 1 |
| 2 | 1 |
| 4 | 0 |
| 8 | 0 (<5) |
| 16 | 0 (<5) |
| 32 | 0 (<50) |
| 64 | 0 (<50) |
| 128 | 0 (<500) |
| 256 | 0 (<5000) |

Όσο μεγαλώνει το μέγεθος φίλτρου , αυξάνονται ο αριθμός των πράξεων που γίνονται για να υπολογιστεί η κάθε θέση του πίνακα αφού η gpu κάνει διαφορετικά τις πράξεις

(floating point operations are not associative, FMAD, different architectures, different compilers) και άρα μειώνεται η ακρίβεια (Φαίνεται από το διάγραμμα εφόσον η ακρίβεια αυξάνεται πάρα πολύ προκειμένου να παίρνουμε ιδιά αποτελέσματα στη cup και την gpu).

5b)

Φαίνεται ότι οι επιδόσεις της gpu είναι πολύ καλύτερες από αυτές της cpu ειδικά όταν μεγαλώνει το μέγεθος της εικόνας εισόδου.

6)

|  |  |
| --- | --- |
| Μέγεθος  φίλτρου | Ακρίβεια σε  δεκαδικά ψηφιά |
| 1 | 10 |
| 2 | 9 |
| 4 | 8 |
| 8 | 8 |
| 16 | 7 |
| 32 | 7 |
| 64 | 6 |
| 128 | 6 |
| 256 | 5 |

Παρατηρώ ότι παρατήρησα και στην περίπτωση των floating point numbers περί ακρίβειας και παρόμοια με πριν η gpu έχει καλύτερη επίδοση από την cpu, όμως σε αυτήν την περίπτωση επειδή χρησιμοποιήσαμε αριθμούς κινητής υποδιαστολής διπλής ακρίβειας πετυχαίνετε πολύ καλή ακρίβεια (το μέγιστο σφάλμα είναι πάρα πολύ μικρό). Αυτό συμβαίνει γιατί η αναπαράσταση των doubles απαιτεί 64 bits ενώ των floats 32 bits. Όμως το μειονέκτημα είναι ότι έχουμε αύξηση του χρόνου εκτέλεσης ειδικά σε μεγάλα μεγέθη φίλτρου.

7a)

Το πόσες φορές διαβάζεται ένα στοιχείο εξαρτάται από το if .

Για τα rows :

* Αν x >= FilterR και x < imageW – FilterR τότε διαβάζεται όλο το for loop οπότε έχουμε

2 FilterR + 1 προσπελάσεις

* Αν x < FilterR τότε το for loop εκτελείται για

2 FilterR + 1 – (FilterR – x) = FilterR + x +1

* Αν x <= imageW – FilterR

2 FilterR + 1 – (ImageW – x)

Ομοίως για τα columns :

* Αν y >= FilterR και y < imageH – FilterR τότε όλο το for loop οπότε έχουμε

2 FilterR + 1 προσπελάσεις

* Αν y < FilterR τότε το for loop εκτελείται για

2 FilterR + 1 – (FilterR – y) = FilterR + y +1

* Αν y <= imageH – FilterR

2 FilterR + 1 – (ImageH – y)

Τα συνολικά είναι το άθροισμα τους.

Για το φίλτρο:

Για τα rows :

* Αν x>= FilterR και x <= imageW – FilterR διαβάζεται το κάθε στοιχείο του φίλτρου από κάθε thread, οπότε έχω ImageW\* ImageH προσπελάσεις.
* Αν x < filterR τότε (ImageW – x) \* ImageH
* Aν x > imageW – FilterR τότε (ImageW – (ImageW – x)) \* ImageH

Ομοίως για τα columns.

7b) Έχουμε δυο πράξεις μια πρόσθεση και έναν πολλαπλασιασμό και δυο προσπελάσεις μνήμης μια για τον πίνακα και μια για το φίλτρο αρά ο λόγος είναι 1

8) Βάζουμε ένα padding γύρω από τον πίνακα μεγέθους filter και μέσα από πειραματικό έλεγχο παρατηρώ ότι μειώνεται αρκετά ο χρόνος εκτέλεσης για την cpu ειδικά όσο αυξάνεται το μέγεθος της εικόνας. Αυτό συμβαίνει γιατί το if statement μέσα στο τριπλό for loop έχει μεγάλο κόστος . Για την gpu δεν παρατηρώ κάποια διαφορά καθώς το if τρέχει παράλληλα οπότε το κόστος είναι μικρό .