Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Πολλαπλασιασμός Ανάστροφου Πίνακα

Δεύτερη Άσκηση Παράλληλων Συστημάτων
Μέρος Β, Ερώτημα Β

Σωτήριος Αίας Καριώρης
Α.Μ.: 19390079
Πρόγραμμα ΠΑΔΑ
Ημερομηνία Παράδοσης: 23-01-2023

Περιεχόμενα

Εισαγωγή

Στην άσκηση αυτή παρουσιάζεται ένα πρόγραμμα γραμμένο σε CUDA C++ που πολλαπλασιάζει ένα διάνυσμα με έναν πίνακα και έπειτα πολλαπλασιάζει το αποτέλεσμα με το ανάστροφο του αρχικού πίνακα. Το πρόγραμμα χρησιμοποιεί πολλαπλούς πυρήνες CUDA για την διαμέριση των πράξεων και τελικά επιτυγχάνει πολύ μικρούς χρόνους εκτέλεσης.

Όλες οι δοκιμές και χρονομετρήσεις έγιναν στον διακομιστή του πανεπιστημίου με κάρτα γραφικών NVIDIA Titan RTX.

Σύντομη περιγραφή προγράμματος

Το πρόγραμμα είναι γραμμένο έτσι ώστε να είναι όσο πιο παραμετρικό γίνεται και να χρησιμοποιεί όσο λιγότερη μνήμη γίνεται. Οι πυρήνες χρησιμοποιούν την τεχνική πλακιδίων (tiles) για γρηγορότερη εκτέλεση.

Συνοπτικά, το πρόγραμμα φορτώνει έναν πίνακα και ένα διάνυσμα και υπολογίζει το γινόμενό τους, που είναι ένα διάνυσμα ίσο σε μέγεθος με τον αριθμό γραμμών του πίνακα. Έπειτα, το διάνυσμα αυτό πολλαπλασιάζεται με τον ανάστροφο πίνακα του αρχικού πίνακα. Στην πράξη ο ανάστροφος πίνακας δεν υπολογίζεται ποτέ. Ο δεύτερος πολλαπλασιασμός χειρίζεται τον αρχικό πίνακα διαφορετικά, πολλαπλασιάζοντας κάθε στήλη του με το διάνυσμα (αντί για κάθε γραμμή). Οι λεπτομέρειες αυτών των διαδικασιών περιγράφονται στην συνέχεια.

Περιγραφή αρχείων πηγαίου κώδικα

Ο κώδικας του προγράμματος είναι μοιρασμένος σε δύο αρχεία, το `ex3_2.cu` και το `ex3_2.cuh`. Συνοπτικά, το αρχείο `.cu` περιέχει την συνάρτηση main του προγράμματος μαζί με τους βασικούς πυρήνες CUDA που υλοποιούν τις πράξεις. Το αρχείο `.cuh` είναι η επικεφαλίδα (header file) του `ex3_2.cu` και περιλαμβάνει διάφορες βασικές συναρτήσεις. Το πρόγραμμα είναι χωρισμένο έτσι χάριν ευκολίας καθώς δημιουργεί πιο ευανάγνωστο κώδικα. Οι συναρτήσεις που περιλαμβάνονται στην επικεφαλίδα μπορούν να χωριστούν σε τρεις ομάδες, τις **συναρτήσεις χρόνου εκτέλεσης CUDA**, τις **συναρτήσεις υπολογισμού παραμέτρου πυρήνων** και τις **συναρτήσεις I/O αρχείων**.

Οι συναρτήσεις χρόνου εκτέλεσης CUDA είναι δύο και χειρίζονται διαδικασίες σχετικά με την CUDA που χρησιμοποιούνται πολύ συχνά στο πρόγραμμα. Η `checkErrors` (που στην πραγματικότητα είναι μακροεντολή) ελέγχει αν υπήρξαν σφάλματα κατά την εκτέλεση μιας συνάρτησης της CUDA και τερματίζει το πρόγραμμα αν εντοπίσει κάποιο πρόβλημα. Αν και ο ακαριαίος τερματισμός ενός προγράμματος δεν είναι καλή προγραμματιστική τεχνική, για λόγους απλότητας έχει χρησιμοποιηθεί. Η συνάρτηση `allocateAndLoad` καλεί απλώς τις συναρτήσεις `cudaMalloc` και `cudaMemcpy` έτσι ώστε να δεσμευτεί μνήμη στην συσκευή CUDA και προαιρετικά να φορτώσει δεδομένα.

Οι συναρτήσεις υπολογισμού παραμέτρων πυρήνα είναι μια συλλογή συναρτήσεων host και device που υπολογίζουν τον βέλτιστο αριθμό blocks που πρέπει να χρησιμοποιηθεί για την πραγματοποίηση της πράξης. Περισσότερες λεπτομέρειες υπάρχουν στην ενότητα ‘Επιλογή παραμέτρων πυρήνα’.

Τέλος, οι συναρτήσεις I/O αρχείων χειρίζονται τα αρχεία που χρησιμοποιεί το πρόγραμμα. Συγκεκριμένα, η `loadFile` φορτώνει τον πίνακα και το διάνυσμα εισόδου, και η `exportResults` αποθηκεύει το διάνυσμα που προέκυψε από την πράξη ώστε να ελεγχθεί η εγκυρότητά του αργότερα (ενότητα ‘Εργαλείο ελέγχου’).

Αρχεία εισόδου

Όπως και στις προηγούμενες ασκήσεις του μαθήματος, τα δεδομένα εισόδου βρίσκονται αποθηκευμένα σε δυαδικά αρχεία μέσα στον φάκελο test. Δημιουργούνται με το εργαλείο `gen` που δημιουργήθηκε αποκλειστικά για αυτόν τον σκοπό. Η δομή των αρχείων είναι πολύ απλή: Τα πρώτα 16 bytes είναι η επικεφαλίδα με τις διαστάσεις του πίνακα και τα υπόλοιπα bytes είναι ο ίδιος ο πίνακας. Τέλος, τα τελευταία bytes είναι τα περιεχόμενα του διανύσματος. Αν ο πίνακας έχει διαστάσεις X×Y το διάνυσμα θα έχει X αριθμούς.

04 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00	Header
00 00 00 00 00 00 80 3F 0E 2D F2 3E C1 CA 41 3F	Matrix
F8 53 A3 BE 58 39 34 3E 3F 35 3E BF 00 00 80 BF	
6F 12 83 3D 1B 2F 5D 3F 81 95 03 3F 8D 97 6E BF	
D9 CE F7 3D 00 00 80 BF BA 49 6C 3F 00 00 00 80	
E3 A5 5B 3F F0 A7 C6 3D 93 18 24 3F 87 16 79 3F	Vector

Αρχείο με έναν 4x4 πίνακα και 1x4 διάνυσμα στο δεκαεξαδικό σύστημα

Η επικεφαλίδα περιέχει δύο 4-bytes ακεραίους και τόσα μηδενικά ώστε να έχει μέγεθος 16 bytes. Οι υπόλοιποι αριθμοί είναι 4-bytes floats.

Για την δημιουργία ενός αρχείου εισόδου καλείται το πρόγραμμα gen με τις διαστάσεις ως ορίσματα, πχ `./gen 4 8`. Το αρχείο δημιουργείται και ονομάζεται αυτόματα.

Ενδεικτική εκτέλεση (1)

```
cuda11@rncp-ubuntu:~/ex3_2$ ./ex3_2 332
CUDA Exc. II
Loaded test 332:
        Matrix: 393216 numbers (512x768)
        Vector: 512 numbers (1x512)
* 384 blocks (out of 576), 2 lines per block.
* 512 blocks (out of 576), 1 lines per block.

* Kernel #1 time:       0.0208 ms
* Kernel #2 time:       0.0287 ms
Done. Writing to 332-out.bin.
```

Εκτέλεση προγράμματος για πίνακα 512x768. Ο πρώτος πολλαπλασιασμός γίνεται με 384 blocks με 2 γραμμές το καθένα ενώ ο δεύτερος πολλαπλασιασμός γίνεται με 512 blocks, όσα και οι γραμμές. Ο δεύτερος πυρήνας χρειάζεται συνήθως περισσότερη ώρα για να εκτελεστεί.

Περιγραφή προγράμματος

Στην ενότητα αυτή περιγράφεται αναλυτικά η δομή του προγράμματος χωρίς, όμως, λεπτομέρειες για τους πυρήνες CUDA.

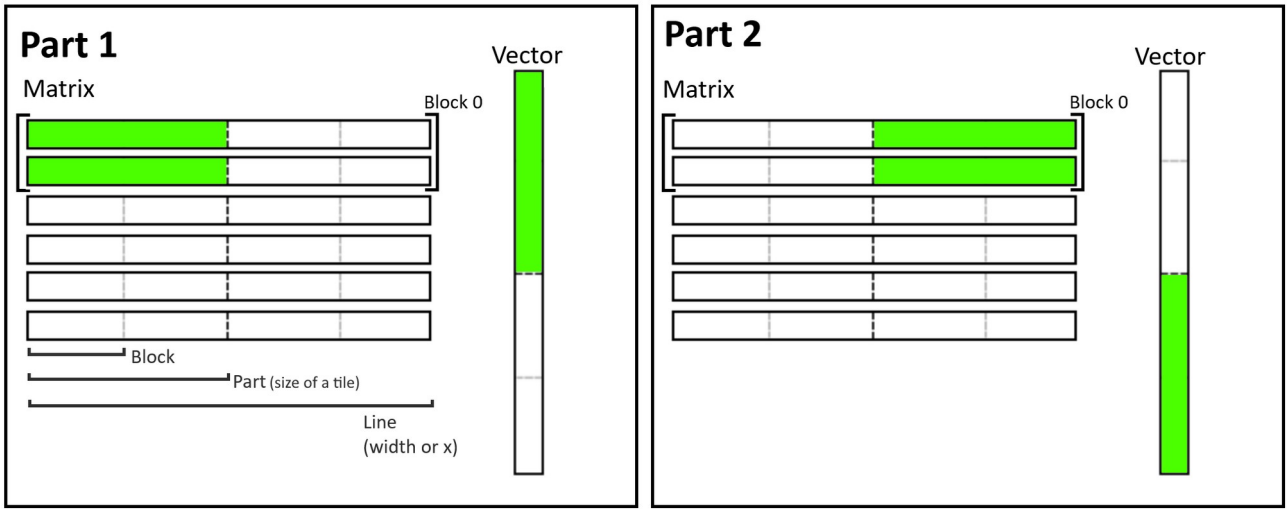
Το πρόγραμμα δέχεται τον αριθμό αρχείου που θα χρησιμοποιηθεί από την γραμμή εντολών. Αφού το φορτώσει χρειάζεται να εξετάσει αν οι διαστάσεις εισόδου μπορούν να χρησιμοποιηθούν. Για λόγους που περιγράφονται στις ενότητες 'Περιγραφή πυρήνων' και 'Επιλογή παραμέτρων πυρήνα', οι διαστάσεις πρέπει να είναι ακέραια πολλαπλάσια του αριθμού νημάτων των CUDA blocks. Πρώτα πραγματοποιείται ο πολλαπλασιασμός του πίνακα με το διάνυσμα. Από εδώ και πέρα το διάνυσμα δεν θα χρειαστεί ξανά. Ωστόσο, καθώς το αρχικό διάνυσμα έχει ίδιο μέγεθος με το τελικό διάνυσμα που θα προκύψει από τον δεύτερο πολλαπλασιασμό, δεν χρειάζεται να αποδεσμευτεί κάποια μνήμη. Στην συνέχει ακολουθεί ο δεύτερος πολλαπλασιασμός. Καθώς τα δεδομένα που φορτώνονται στην συσκευή CUDA μέσω της `cudaMemcpy` δεν διαγράφονται αυτόματα δεν υπάρχει λόγος να γίνει κάποια αντιγραφή δεδομένων από ή προς την συσκευή ανάμεσα στις κλήσεις των δύο πυρήνων.

Περιγραφή πυρήνων

Οι δύο πυρήνες που χρησιμοποιούνται είναι σχεδόν ίδιοι, με μόνη διαφορά την επιλογή στοιχείων από τον πίνακα: Ο πυρήνας `mult_MatByVec` πολλαπλασιάζει τα στοιχεία του διανύσματος με τα στοιχεία της κάθε γραμμής του πίνακα, όπως στον κανονικό πολλαπλασιασμό, ενώ ο πυρήνας `mult_TransByVec` πολλαπλασιάζει τα στοιχεία του διανύσματος με αυτά της κάθε στήλης του πίνακα. Στην ενότητα αυτή θα αναλυθεί ο πρώτος πυρήνας αλλά η ακριβώς ίδια λογικά χρησιμοποιείται και στον δεύτερο.

Ο πυρήνας λειτουργεί χρησιμοποιώντας δύο μεγέθη, τα parts (μέρη) και τα blocks. Κάθε γραμμή του πίνακα (όπως και το διάνυσμα) χωρίζονται σε πολλαπλά μέρη ενώ κάθε μέρος χωρίζεται σε πολλαπλά blocks. Ο πυρήνας χρησιμοποιεί κοινή μνήμη για την αποθήκευση του διανύσματος στον πίνακα `Vds`. Το μέγεθος του `Vds` ορίζεται από την σταθερά `IN_VEC_BLOCK_COUNT` που ορίζει την σταθερά `IN_VEC_TILE_SIZE`. Ένα block μνήμης είναι 128 νούμερα, ακριβώς όσα και τα νήματα σε ένα block πυρήνα. Ένα μέρος αποτελείται από `IN_VEC_BLOCK_COUNT` blocks και αυτό το μέγεθος ορίζει το πόσο μεγάλο είναι ένα μέρος. Έτσι, αν ο αριθμός block οριστεί να είναι 64 ένα μέρος θα αποτελείται από 8192 νούμερα ή 32KiB. Αυτά τα μεγέθη είναι σημαντικά καθώς ορίζουν άμεσα τον αριθμό βημάτων που θα χρειαστεί η συσκευή CUDA για να διεκπεραιώσει τον υπολογισμό.

Κάθε block φορτώνει στην κοινή μνήμη ένα μέρος του διανύσματος. Καθώς το τελικό διάνυσμα αποτελείται από αθροίσματα, από το μέρος του διανύσματος που φορτώθηκε ένα block μπορεί να υπολογίσει το μερικό άθροισμα της κάθε γραμμής που του αντιστοιχεί. Για να ολοκληρώσει τα αθροίσματα των γραμμών, το block θα πρέπει να περάσει από όλα τα μέρη του διανύσματος. Αυτή η λογική έχει ως προτεραιότητα την ελαχιστοποίηση των φορτώσεων στην κοινή μνήμη: Κάθε μέρος του διανύσματος φορτώνεται μόνο μια φορά ενώ κάθε στοιχείο μιας γραμμής επίσης διαβάζεται μόνο μια φορά. Καθώς τα στοιχεία μιας γραμμής χρησιμοποιούνται μόνο μια φορά δεν υπάρχει λόγος να αποθηκευτούν στην κοινή μνήμη. Το κάθε block έχει επίσης έναν πίνακα στην κοινή μνήμη ονομασμένο `Rds` και λειτουργεί σαν αθροιστής γραμμών. Αν για παράδειγμα το κάθε block πρέπει να υπολογίσει 3 γραμμές, μόνο τα πρώτα 3 στοιχεία του `Rds` θα χρησιμοποιηθούν, ένα για κάθε γραμμή. Στο τέλος του υπολογισμού τα αθροίσματα από τις έγκυρες θέσεις των `Rds` θα αποθηκευτούν στην καθολική μνήμη, στην κατάλληλη θέση του τελικού διανύσματος.



Διαμέριση γραμμών και διανύσματος σε δύο μέρη

Οι ίδιες διαδικασίες επαναλαμβάνονται για κάθε μέρος. Κάθε μέρος αποτελείται από συγκεκριμένο αριθμό block. Σε κάθε διαδικασία που αφορά ένα block, δηλαδή στην φόρτωση του διανύσματος στην κοινή μνήμη και στον υπολογισμό του μερικού αθροίσματος, τα νήματα δουλεύουν όλα ταυτόχρονα και εκτελούν ακριβώς την ίδια διαδικασία. Οι διαδικασίες αυτές αριθμούνται στον κώδικα με την μεταβλητή `j`.

Η αντιγραφή του διανύσματος είναι αρκετά απλή. Από την άλλη, ο υπολογισμός των μερικών αθροισμάτων μιας γραμμής γίνεται ως εξής: Σε κάθε `j`, το κάθε νήμα διαβάζει ένα στοιχείο από το τρέχον block μνήμης του πίνακα και το αντίστοιχο στοιχείο από το διάνυσμα. Εκτελεί τον πολλαπλασιασμό και γράφει το αποτέλεσμα στην θέση που του αντιστοιχεί στον πίνακα `a`. Ο πίνακας αυτός περιέχει μια θέση για κάθε νήμα και είναι στην πραγματικότητα ένας πίνακας αθροιστών στην κοινή μνήμη. Όταν όλα τα νήματα έχουν γράψει στον πίνακα `a` ξεκινά η άθροιση του πίνακα με αλγόριθμο δυαδικού δέντρου. Το αποτέλεσμα είναι αποθηκευμένο στην πρώτη θέση του `a` (θέση 0) και το νήμα 0 προσθέτει την τιμή στον αθροιστή της γραμμής στον πίνακα `Rds`. Όταν όλα τα blocks του τρέχοντος μέρους έχουν υπολογισθεί ο αλγόριθμος συνεχίζει στην επόμενη γραμμή. Όταν όλες οι γραμμές του block νημάτων έχουν υπολογισθεί ο αλγόριθμος προχωρά στην αντιγραφή του επόμενου μέρους.

Όταν, τελικά, όλα τα στοιχεία του τελικού διανύσματος είναι έτοιμα, μερικά νήματα από κάθε block θα αντιγράψουν τα κατάλληλα στοιχεία του `Rds` στην καθολική μνήμη.

Ο δεύτερος πυρήνας λειτουργεί ίδια. Η μόνη αξιοσημείωτη παρατήρηση είναι ότι για την κλήση του δεύτερου πυρήνα δεν χρειάζεται κάποια περαιτέρω διαδικασία από τον host. Όλα τα δεδομένα που θα χρειαστούν για τον επόμενο υπολογισμό βρίσκονται ήδη στην συσκευή CUDA. Επίσης επισημαίνεται πως αν το μέγεθος των γραμμών ή των στηλών είναι μικρότερο από το μέγεθος του πλακιδίου τα νήματα θα ανταποκριθούν αντίστοιχα. Ένας περιορισμός στις διαστάσεις είναι να είναι ακέραια πολλαπλάσια του αριθμού νημάτων ανά block, αλλιώς η λογική των blocks και της μεταβλητής j δεν λειτουργεί. Ένας πιθανός τρόπος λύσης αυτού του περιορισμού περιγράφεται στην ενότητα ‘Πιθανές βελτιώσεις’.

Επιλογή παραμέτρων πυρήνα

Μέχρι τώρα έχουν αναφερθεί διάφορες σταθερές, όπως ο αριθμός νημάτων ανά block. Σε αυτήν την ενότητα θα επεξηγηθούν περαιτέρω αυτές οι τιμές όπως και η λογική με την οποία επιλέγονται.

Το πρόγραμμα επιλέγει δυναμικά τον αριθμό στοιχείων ανά blocks νημάτων ανάλογα με τις διαστάσεις εισόδου. Ωστόσο, ο αριθμός νημάτων ανά blocks είναι σταθερά 128, τουλάχιστον κατά την εκτέλεση του προγράμματος. Γενικά, οι πιθανές τιμές αυτής της παραμέτρου δεν είναι πολλές. Ο μηχανισμός των wraps λειτουργεί βέλτιστα με μεγέθη πολλαπλάσια του 32, η άθροιση των τιμών ανά block γραμμών απαιτεί πλήθος νημάτων δύναμη του δύο ενώ οι δομή των multiprocessors επιβάλει κι άλλους περιορισμούς με βάση τον μέγιστο αριθμό blocks, νημάτων και καταχωρητών ανά multiprocessor.

```
cuda11@rncp-ubuntu:~/ex3_2$ ../cudastats
 -- 1 CUDA Compatible devices found--
0: NVIDIA TITAN RTX (CUDA Ver. 7.5)
===Thread Partitioning===
SM: 72 [Max Blocks: 16 | Max Threads: 1024]
Maximum Block Dimensions <1024, 1024, 64>
Maximum Threads per Block: 1024
Maximum threads per SM: 65536 | Maximum registers per block: 65536 (512 reg./thread for 128 thr.)
===Block Sizes===
Maximum blocks for      64 thr.: 1152   128 thr.: 576   256 thr.: 288   512 thr.: 144
===Memory Details===
Total Global Memory: 24217MB
SM Shared memory: 64KB
Block Shared Memory: 48KB
```

Το πρόγραμμα cudastats καλεί την συνάρτηση cudaGetDeviceProperties και εμφανίζει κάποια βασικά δεδομένα για τις δυνατότητες της κάρτας γραφικών

Σε κάρτες όπως αυτή του μηχανήματος, οι multiprocessors (ή streaming multiprocessors) μπορούν να χειριστούν blocks με το πολύ 1024 νήματα ενώ οι ίδιοι οι mutliprocessors μπορούν αν χειριστούν 1024 νήματα γενικά. Έτσι, blocks μεγέθους 512 νημάτων θα ανάγκαζαν κάθε SM να χειρίζεται μόνο δύο blocks, δημιουργώντας τελικά μόνο 72×2 =144 blocks. Επίσης, η κοινή μνήμη ανά SM είναι περιορισμένη. Αυτό έχει ως αποτέλεσμα η κοινή μνήμη ανά block να είναι πολύ μικρή σε έναν SM με πολλά blocks. Για παράδειγμα θα χωρούσαν 16 blocks των 64 νημάτων σε έναν SM αλλά θα έπρεπε να μοιραστούν τα 64KiB που ο SM μπορεί να προσφέρει, δίνοντας μονάχα 4KiB κοινής μνήμης στο κάθε block. Γενικά, το μέγεθος των 128 νημάτων είναι μια καλή μέση λύση που οδηγεί σε το πολύ 576 blocks με το πολύ 8KiB κοινή μνήμη το καθένα. Στην ενότητα 'Σύγκριση αποτελεσμάτων' φαίνονται οι διαφορές στις επιδόσεις του προγράμματος με βάση διαφορετικά μεγέθη blocks.

Η επιλογή του αριθμού blocks γίνεται με την συνάρτηση `findKernelParameters1D` που καλεί την συνάρτηση πυρήνα `findBlockCount`. Όπως έχει αναφερθεί, κάθε block λαμβάνει έναν αριθμό γραμμών και αργότερα έναν αριθμό στηλών. Ιδανικά, υπάρχουν αρκετοί πυρήνες ώστε κάθε ένας να λάβει μόνο μια γραμμή. Στην περίπτωση που αυτό δεν γίνεται ο πυρήνας `findBlockCount` θα ενεργοποιηθεί και 1024 νήματα θα ψάξουν παράλληλα για τον μέγιστο τέλειο ακέραιο διαιρέτη του αριθμού γραμμών. Ο τρόπος που αυτός ο αλγόριθμος υλοποιείται φαίνεται στον πηγαίο κώδικα. Η διαδικασία γίνεται μια φορά για να βρεθεί διαιρέτης των γραμμών και έπειτα διαιρέτης των στηλών.

Επιλογή μεγέθους πλακιδίου

Η επιλογή μεγέθους κοινής μνήμης προκύπτει άμεσα από τους περιορισμούς της κάρτας. Για 128 νήματα ο πίνακας a χρειάζεται 128×4=512 bytes. Καθώς ένας SM στο μηχάνημα μας μπορεί να χειριστεί το πολύ 1024 νήματα, 1024 μέγιστο/128 νήματα = 8 blocks. Τα 8 blocks αυτά θα πρέπει να μοιραστούν 64KiB. Άρα το καθένα θα έχει 64KiB/8=8KiB κοινής μνήμης. Τελικά, αν θεωρήσουμε οτι ο μέγιστος αριθμός γραμμών που μπορεί να αναλάβει ένα block είναι 128, ο *Rds* θα έχει μέγεθος 512 bytes. Μένουν 7KiB για το *Vds*, πράγμα που σημαίνει ότι θα χωράει 1792 αριθμούς ή 14 blocks 128 αριθμών. Κάθε block μπορεί να χειριστεί 128 γραμμές και αργότερα στήλες άρα ο μεγαλύτερος πίνακας που το πρόγραμμα θα μπορούσε, θεωρητικά, να χειριστεί στο μηχάνημα του πανεπιστημίου θα είχε μέγιστη διάσταση 512×128 = 65536. Ένας πίνακας 65536×65536 περιέχει περίπου 4 δις. αριθμούς και χρειάζεται 16GiB αποθηκευτικού χώρου, μέγεθος εντός των 24GiB global μνήμης της κάρτας του μηχανήματος.

Ενδεικτική εκτέλεση (2)

```
cuda11@rncp-ubuntu:~/ex3_2$ ./ex3_2 398
CUDA Exc. II
Loaded test 398:
        Matrix: 2981888 numbers (1664x1792)
        Vector: 1664 numbers (1x1664)
* 448 blocks (out of 576), 4 lines per block.
* 416 blocks (out of 576), 4 lines per block.

* Kernel #1 time:       0.1025 ms
* Kernel #2 time:       0.1732 ms
Done. Writing to 398-out.bin.
```

Στην εκτέλεση αυτήν δεν χρησιμοποιούνται ποτέ πάνω από 450 blocks ενώ κάθε block πρέπει να δουλέψει για 4 γραμμές. Ωστόσο οι πόροι που απομένουν δεν μπορούν να χρησιμοποιηθούν εύκολα αφού κάτι τέτοιο θα χαλούσε την συμμετρία των πυρήνων, δίνοντας λιγότερες γραμμές/στήλες σε κάποια block.

Εργαλείο ελέγχου

Για την διαπίστωση της εγκυρότητας των αποτελεσμάτων του προγράμματος δημιουργήθηκε το εργαλείο `check`. Όταν το κύριο πρόγραμμα ολοκληρώνεται αποθηκεύει το τελικό διάνυσμα σε ένα δυαδικό αρχείο το οποίο το πρόγραμμα `check` μπορεί να ελέγξει. Το εργαλείο αυτό υπολογίζει τα δεδομένα με σειριακό τρόπο και τα συγκρίνει με αυτά του παράλληλου. Ο σειριακός κώδικας είναι πολύ πιο απλός και για αυτό είναι σίγουρα σωστός. Η ύπαρξή του αναφέρεται καθώς χωρίς αυτό η αποσφαλμάτωση του προγράμματος θα ήταν αδύνατη. Επίσης μπορούν να συγκριθούν οι χρόνοι του σειριακού κώδικα με του παράλληλου.

```
cuda11@rncp-ubuntu:~/ex3_2$ ./check 332
Input matrix:           OK [512x768 (393216)]
Input vector:           OK [512x1 (512)]
Output from cuda:       OK [512x1 (512)]
(Input: 1538KB  Output: 2KB)

Calculating Matrix by vector results...OK
Calculating final vector...OK
Checking...OK

Time for step 1:                2.002 ms
Time for step 2:                2.041 ms
Time for both steps:    4.043 ms
Total time ellapsed:    5.132 ms

Writing to checkout.bin: 2KB
```

Έλεγχος εγκυρότητας της ενδεικτικής εκτέλεσης από παραπάνω. Όλα φαίνονται καλά.

Όπως είναι αναμενόμενο, ο σειριακός κώδικας είναι πολύ πιο αργός. Όταν οι πυρήνες ήταν ακόμα σε ανάπτυξη και δεν έβγαζαν πάντα σωστά αποτελέσματα το εργαλείο ελέγχου εντόπιζε πόσα λάθη υπήρχαν.

Σύγκριση αποτελεσμάτων

Σε αυτήν την ενότητα θα συγκριθούν οι χρόνοι του προγράμματος για διάφορες αλλαγές στις παραμέτρους του.

Για αρχή θα συγκριθούν οι επιδόσεις για διαφορετικά μεγέθη πινάκων εισόδου. Για μικρούς τετραγωνικούς πίνακες, με διαστάσεις μέχρι και 1024x1024, οι πυρήνες χρειάζονται λιγότερο από 100ns για να υπολογίσουν τα ζητούμενα ενώ, άσχετα από τα μεγέθη εισόδου, οι χρόνοι είναι πάνω κάτω ίδιοι.

```
cuda11@rncp-ubuntu:~/ex3_2$ ./ex3_2 589
CUDA Exc. II
Loaded test 589:
        Matrix: 1048576 numbers (1024x1024)
        Vector: 1024 numbers (1x1024)
* 512 blocks (out of 576), 2 lines per block.
* 512 blocks (out of 576), 2 lines per block.

* Kernel #1 time:       0.0414 ms
* Kernel #2 time:       0.0635 ms
Done. Writing to 589-out.bin.
```

Όταν οι διαστάσεις ξεπερνούν τα χίλια νούμερα οι χρόνοι αρχίζουν να εμφανίζουν πιο σταθερή συμπεριφορά.

Διαστάσεις εισόδου	**Χ.Ε. πυρήνα #1 (ns)**	**Χ.Ε. πυρήνα #2 (ns)**
1792×1792	110	190
3584×3584	400	2000
7168×7168	1470	1300
14336×14336	5500	5750

Όπως προαναφέρθηκε, ο χρόνος εκτέλεσης του δεύτερου πυρήνα είναι λίγο μεγαλύτερος από αυτόν του πρώτου, ακόμα και όταν οι στήλες είναι όσες και οι γραμμές. Η πιο πιθανή εξήγηση σε αυτό το φαινόμενο είναι πως ο υπολογισμός του δείκτη `globalIndex` είναι πιο ακριβός όταν δεικτοδοτούνται τα στοιχεία των στηλών, έναντι των γραμμών (4 πολλαπλασιασμοί έναντι των 3 αντίστοιχα).

Στην συνέχεια θα εξεταστούν οι επιδόσεις του προγράμματος για block νημάτων μεγέθους 128, 256 και 512 νημάτων. Αν και τα 64 νήματα δεν έχουν κάποιο απαγορευτικό χαρακτηριστικό, ο αλγόριθμος του πυρήνα `findBlockCount` παύει να λειτουργεί χωρίς τροποποιήσεις όταν καλείται να βρει βέλτιστο πλήθος blocks 64 νημάτων, καθώς το μέγεθος 64 νημάτων οδηγεί την κάρτα στο να θεωρήσει ότι μπορεί να υποστηρίξει 1152 blocks. Αν και πράγματι μπορεί, ο πυρήνας ενεργοποιείται με ένα block μεγέθους όσο και το μέγιστο πιθανό πλήθος blocks. Καθώς η κάρτα υποστηρίζει το πολύ 1024 νήματα ανά block, τα 1152 νήματα οδηγούν σε σφάλμα.

Για την χρονομέτρηση, για κάθε διαφορετικό μέγεθος block, οι πυρήνες χρησιμοποιούν και διαφορετικά μεγέθη πλακιδίων. Τα μεγέθη πλακιδίων για τα blocks 256 και 512 νημάτων προκύπτουν όπως περιγράφεται στην ενότητα 'Επιλογή μεγέθους πλακιδίου'.

Μέγεθος block	Μέγιστο πλήθος block	Μέγεθος πλακιδίου	IN_VEC_BLOCK_COUNT
128	576	1920 αριθμοί, 7.5KiB	15
256	288	3840 αριθμοί, 15KiB	15
512	144	7680 αριθμοί, 30KiB	15

Η διαμοίραση δεν είναι τέλεια αλλά είναι αναγκαίο να γίνει έτσι ώστε τα πλακίδια να έχουν κοινά πολλαπλάσια. Έτσι, το πλακίδιο των 128 νημάτων θα χρειαστεί να "φάει" λίγη από την μνήμη του πίνακα αθροισμάτων `Rds` ενώ τα πλακίδια των άλλων δύο μεγεθών αφήνουν 1KiB και 2KiB αντίστοιχα αχρησιμοποίητα.

Ακολουθούν οι μετρήσεις για πίνακες 1920x1024, 3840x1024, 7680x1024 και 12288x1024.

1920×1024

Μέγεθος Block	Χ.Ε. Πυρήνα #1 (ns)	Χ.Ε. Πυρήνα #2 (ns)	Άθροισμα (ns)
128	76.2	12.3	88.5
256	*N/A*	*N/A*	*N/A*
512	*N/A*	*N/A*	*N/A*

3840×1024

Μέγεθος Block	Χ.Ε. Πυρήνα #1 (ns)	Χ.Ε. Πυρήνα #2 (ns)	Άθροισμα (ns)
128	148	375	523
256	155	641	796
512	*N/A*	*N/A*	*N/A*

7680×1024

Μέγεθος Block	Χ.Ε. Πυρήνα #1 (ms)	Χ.Ε. Πυρήνα #2 (ms)	Άθροισμα (ms)
128	0.288	1.75	2.04
256	0.302	2.02	2.30
512	2.42	1.61	4.03

12288×1024

Μέγεθος Block	Χ.Ε. Πυρήνα #1 (ms)	Χ.Ε. Πυρήνα #2 (ms)	Άθροισμα (ms)
128	0.42	3.13	3.55
256	0.44	3.13	3.57
512	0.35	2.87	3.22

Τελικά γίνεται φανερό πως τα μικρά blocks είναι καλύτερα από τα μεγαλύτερα. Όχι μόνο μπορούν να χειριστούν περισσότερες διαστάσεις αλλά φαίνεται να λειτουργούν και πιο γρήγορα. Γενικά, τα μεγάλα blocks είναι πιο αργά από τα μικρά και η διαφορά στον χρόνο συγκλίνει καθώς οι διαστάσεις μεγαλώνουν. Τουλάχιστον για τους πίνακες που δοκιμάστηκαν, τα blocks 512 νημάτων έχουν καλύτερες επιδόσεις από αυτά των 128 νημάτων μόνο αφού η μια διάσταση ξεπεράσει τα 10.000 νούμερα που, ακόμα και τότε, η διαφορά δεν είναι πολύ μεγάλη.

```
CUDA Exc. II
Loaded test 413:
        Matrix: 12582912 numbers (12288x1024)
        Vector: 12288 numbers (1x12288)
* 128 blocks (out of 144), 8 lines per block.
* 128 blocks (out of 144), 96 lines per block.

* Kernel #1 time:       0.3474 ms
* Kernel #2 time:       2.8668 ms
Done. Writing to 413-out.bin.
```

Αποτελέσματα προγράμματος με blocks 512 νημάτων που υπολογισε έναν μεγάλο πολλαπλασιασμό. Τα blocks είναι λίγα και πρέπει να δουλέψουν πάνω σε πολλές γραμμές.

Τέλος, θα συγκριθούν οι επιδόσεις του προγράμματος με τις διάφορες παραμέτρους ρυθμισμένες όπως στην αρχή, με τα blocks μεγέθους 128 νημάτων, αλλά με διαφορετικά μεγέθη πλακιδίων. Θα μετρηθούν οι χρόνοι για πλακίδια 4 blocks αριθμών και 8 blocks αριθμών ή 512 και 1024 αριθμών. Σημειώνεται πως καθώς και οι δύο τιμές είναι μικρότερες από το βέλτιστο μέγεθος των 14 blocks το πρόγραμμα δεν θα λειτουργεί βέλτιστα.

```
cuda11@rncp-ubuntu:~/ex3_2$ ./ex3_2 333
CUDA Exc. II
Loaded test 333:
        Matrix: 160563200 numbers (8960x17920)
        Vector: 8960 numbers (1x8960)
* 560 blocks (out of 576), 32 lines per block.
* 560 blocks (out of 576), 16 lines per block.

* Kernel #1 time:       4.1877 ms
* Kernel #2 time:       43.3096 ms
Done. Writing to 333-out.bin.
```

7 blocks

```
cuda11@rncp-ubuntu:~/ex3_2$ ./ex3_2 333
CUDA Exc. II
Loaded test 333:
        Matrix: 160563200 numbers (8960x17920)
        Vector: 8960 numbers (1x8960)
* 560 blocks (out of 576), 32 lines per block.
* 560 blocks (out of 576), 16 lines per block.

* Kernel #1 time:       4.1881 ms
* Kernel #2 time:       44.0543 ms
Done. Writing to 333-out.bin.
```

14 blocks

Εκτελέσεις για τον ίδιο πίνακα με τα μεγέθη των πλακιδιών ρυθμισμένα διαφορετικά

Οι αλλαγές στα μεγέθη των πλακιδίων δεν φαίνεται να προκαλούν σημαντικές διαφορές στις επιδόσεις. Μάλιστα, το πρόγραμμα έδωσε ίδιους μέσους χρόνους και για τις δύο ρυθμίσεις στον παραπάνω πίνακα.

Πιθανές βελτιώσεις

Ο κώδικας, αν και αποδοτικός, έχει μερικούς περιορισμούς που θα μπορούσαν να λυθούν με προσθέσεις κώδικα στους πυρήνες CUDA. Οι περιορισμοί στις τιμές των διαστάσεων πηγάζουν από τον τρόπο με τον οποίο οι πυρήνες διαμοιράζουν τις γραμμές/στήλες σε blocks δεδομένων και μέρη. Αν μια διάσταση είναι μεγαλύτερη της κοινής μνήμης (`IN_VEC_TILE_SIZE`) αλλά δεν είναι ακέραιο πολλαπλάσιό της ο κώδικας λειτουργεί λάθος καθώς η διαίρεση για τον υπολογισμό του `partCount` θα στρογγυλοποιήσει την τιμή προς τα κάτω, οδηγώντας τον πυρήνα να παραλείψει τον υπολογισμό των παραπανίσιων block δεδομένων.

```
// Lines the block must compute
unsigned linesPerBlock = _lines / gridDim.x;
// First line the block gets
unsigned startLine = blockIdx.x * linesPerBlock;

// Number of parts a line must be broken to
unsigned partCount;
if(_width > IN_VEC_TILE_SIZE)
    partCount = _width / IN_VEC_TILE_SIZE;
else
    partCount = 1;
```

Η εντολή `part = _width / IN_VEC_TILE_SIZE` είναι η διαίρεση που προκαλεί προβλήματα στο παραπάνω σενάριο.

Για παράδειγμα, για πλακίδια 256 αριθμών, η γραμμή 384 ενώ είναι πολλαπλάσιο του 128 θα οδηγήσει σε `partCount` 1.5 που τελικά θα γίνει 1. Έτσι, ενώ η γραμμή αποτελείται από 3 blocks των 128 αριθμών μόνο τα δύο πρώτα blocks θα υπολογιστούν, τα δύο blocks που ανήκουν στο μέρος 0. Το πρόβλημα αυτό θα μπορούσε να λυθεί αν γινόταν έλεγχος μετά την διαίρεση του αποσπάσματος ώστε το `partCount` να στρογγυλοποιείται προς τα πάνω. Τότε όμως, μέσα στις επαναλήψεις του πυρήνα θα έπρεπε να γίνονται έλεγχοι ώστε οι υπολογισμοί να μην συνεχίζονται σε blocks που δεν αντιστοιχούν σε νούμερα των δεδομένων εισόδων. Για παράδειγμα στο παράδειγμα παραπάνω, τα νήματα θα έπρεπε να σταματήσουν πριν το block 1 του μέρους 1, ειδάλλως για γραμμή μεγέθους 384, θα συνέχιζαν να υπολογίζουν τιμές για τις θέσεις 384 με 511 που δεν θα ήταν έγκυρες.

Ο περιορισμός στις διαστάσεις για διαιρετότητα με τον αριθμό νημάτων ανά block θα μπορούσε, θεωρητικά, να λυθεί αν το κάθε νήμα είχε μεγαλύτερη αυτονομία στο πότε σταματάει να συμμετέχει σε υπολογισμούς. Ωστόσο, κάτι τέτοιο θα απαιτούσε ιδιαιτέρα σύνθετους ελέγχους μέσα στον πυρήνα και, γενικά, οι πολλές διακλαδώσεις στους κώδικες που εκτελούν τα νήματα μπορεί να οδηγήσουν σε χειρότερες επιδόσεις.

Μια ακόμα λειτουργία που έχει παραληφθεί από το πρόγραμμα για λόγους απλότητας είναι η δυναμική επιλογή πυρήνα. Στις ενότητες ‘Επιλογή παραμέτρων πυρήνα’ και ‘Επιλογή μεγέθους πλακιδίου’ οι τιμές υπολογίσθηκαν με βάση τις δυνατότητες του μηχανήματος πάνω στο οποίο αναπτύχθηκε το πρόγραμμα. Ωστόσο, άλλες συμβατές με CUDA κάρτες γραφικών μπορεί να μην έχουν 48KiB/block, 64KiB/SM κοινή μνήμη αλλά λιγότερο, όπως είναι συχνό σε παλιότερες κάρτες. Σε μια εφαρμογή CUDA που προορίζεται για εμπορική χρήση σε πολλά διαφορετικά μηχανήματα θα συνέφερε να οριστούν διαφορετικοί πυρήνες, ο καθένας με άλλα μεγέθη πλακιδίων

και μέγιστων γραμμών ανά block (*IN_VEC_BLOCK_COUNT*, *MAX_LINE_COUNT*). Αντίστοιχα, υπάρχουν κάρτες που υποστηρίζουν περισσότερα ή λιγότερα νήματα, όπως κάποιες κάρτες υποστηρίζουν 2048 νήματα ανά block αντί για 1024. Σε τέτοιες περιπτώσεις μπορεί να συνέφερε το μέγεθος block να μην είναι σταθερό αλλά να αλλάζει από 128 σε 64, ανάλογα τις δυνατότητες της κάρτας.

Τέλος, αν και η επιλογή διαστάσεων πλέγματος γίνεται αποκλειστικά σε μονοδιάστατο επίπεδο ίσως να υπήρχαν βελτιώσεις αν κάθε block νημάτων είχε δύο διαστάσεις. Για παράδειγμα, η διάσταση X των blocks θα μπορούσε να ορίσει με ποιες γραμμές ασχολούνται τα blocks και η διάσταση Y με ποια μέρη των γραμμών. Κάτι τέτοιο, όμως, θα προσέφερε βελτιώσεις μόνο αν υπήρχαν πολλά περισσότερα νήματα.