Project Λογισμικού και Προγραμματισμού Συστημάτων Υψηλής Επίδοσης

Ονομ/νυμο	Α.Μ.	Έτος
Σκαμνέλος Νικόλαος	1041878 ή 236201	6ο
Γιάννης Ακαρέπης	1054368	4ο
Νικόλαος Βασιλακόπουλος	1044757	6ο

Περιεχόμενα

Αλγόριθμοι που υλοποιήθηκαν

1) Υλοποίηση με χρήση της βιβλιοθήκης Cublas.

   α) Αρχείο: HPC_task_1.cu

   β) Περιγραφή Αλγορίθμου:

   Αρχικά μέσω της συνάρτησης init() παίρνουμε τις διαστάσεις του μητρώου εισόδου(A) και μετά μέσω της συνάρτησης curand() γεμίζουμε τον A στην GPU με τιμές. Τέλος, καλούμε την συνάρτηση gpu_blas_mmul, η οποία αρχικοποιεί τις μεταβλητές που χρειαζόμαστε και καλεί την συνάρτηση cublasSgemm από την βιβλιοθήκη cuBLAS έτσι ώστε να γίνει ο υπολογισμός που ζητάμε(A^T*A).
   Ο απαιτούμενος χρόνος για να ολοκληρωθεί ο υπολογισμός μετριέται με την χρήση των Cuda Events. Επίσης, φροντίζουμε να αποδεσμεύσουμε όλες τις περιοχές μνήμης που χρησιμοποιήσαμε είτε αυτές είναι στην GPU, είτε στην CPU.

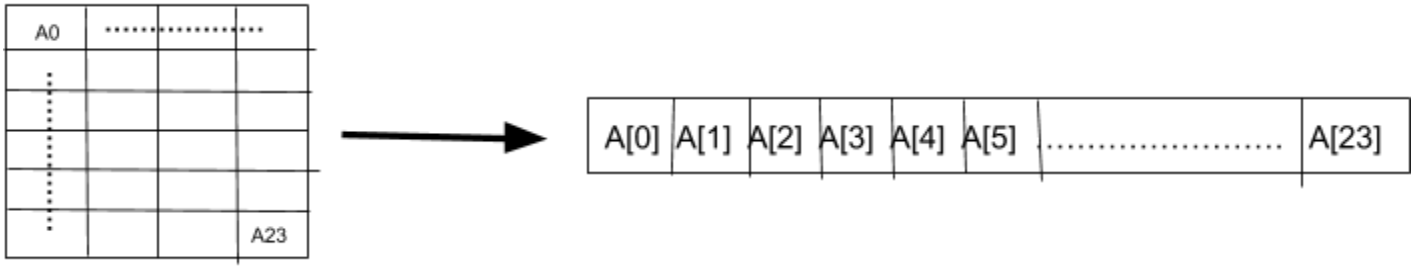
2) Υλοποίηση στην GPU χωρίς βελτιστοποιήσεις.

α) Αρχείο: HPC_task_2.cu

β) Περιγραφή Αλγορίθμου:

Όπως και παραπάνω χρησιμοποιήσαμε τη συνάρτηση init() για να πάρουμε τις διαστάσεις του μητρώου εισόδου (A). Έπειτα, καλούμε μία συνάρτηση με όνομα table_generator() η οποία γεμίζει τον πίνακα εισόδου με τυχαίες τιμές(οι οποίες εξαιτίας της rand(), δεν είναι και τόσο τυχαίες).

Δημιουργούμε το grid με τα threads που θα χρησιμοποιήσουμε και καλούμε το kernel που θα κάνει τους υπολογισμούς. Μέσα στο kernel αρχικά αποθηκεύουμε τα Thread IDs στις μεταβλητές x και y αντίστοιχα και αρχικοποιούμε τις αρχικές τιμές του μητρώου εξόδου(C). Ύστερα προχωράμε στο κύριο υπολογισμό κάθε στοιχείου, προσέχοντας τις συντεταγμένες αφού οι πίνακες μας βρίσκονται σε μονοδιάστατη αναπαράσταση. Αυτό φαίνεται και στην παρακάτω εικόνα:



Ο απαιτούμενος χρόνος για να ολοκληρωθεί ο υπολογισμός μετριέται με την χρήση των Cuda Events. Επίσης, φροντίζουμε να αποδεσμεύσουμε όλες τις περιοχές μνήμης που χρησιμοποιήσαμε είτε αυτές είναι στην GPU, είτε στην CPU.

3) Υλοποίηση στην GPU με βελτιστοποιήσεις.

α) Αρχείο: HPC_task_3.cu

β) Περιγραφή Αλγορίθμου:

Τηρώντας την ίδια λογική με το παραπάνω αλγόριθμο, προσθέτουμε κάποιες βελτιστοποιήσεις. Αρχικά, για να εκμεταλλευτούμε την ικανότητα της cuda για coalesced memory accesses εισάγουμε ένα padding στις στήλες του μητρώου εισόδου έτσι ώστε να είναι ακέραιο πολλαπλάσιο του 128. Έπειτα, χρησιμοποιούμε τη shared memory της cuda. Παρατηρούμε ότι στην περίπτωση που τα tiles ανήκουν στην κύρια διαγώνιο του μητρώου εξόδου, οι δύο shared memories(A_shared, A_shared_inv) περιέχουν τα ίδια στοιχεία στις ίδιες θέσεις, διευκολύνοντας τον υπολογισμό των παραπάνω στοιχείων. Για τα υπόλοιπα tiles, βλέπουμε ότι υπάρχει ένα είδος συμμετρίας ως προς τη διαγώνιο αλλά με ανάστροφο indexing των θέσεων τους, οπότε τα υπολογίζουμε accordingly(?).

Ο απαιτούμενος χρόνος για να ολοκληρωθεί ο υπολογισμός μετριέται με την χρήση των Cuda Events. Επίσης, φροντίζουμε να αποδεσμεύσουμε όλες τις περιοχές μνήμης που χρησιμοποιήσαμε είτε αυτές είναι στην GPU, είτε στην CPU.

Συστήματα που υλοποιήθηκαν

1) **init():**

Αρχικοποιεί τις διαστάσεις του πίνακα, λαμβάνοντας τιμές από το πληκτρολόγιο.

2) **GPU_fill_rand()/table_generator():**

Γεμίζει το πίνακα με τιμές είτε με την χρήση της curand()(GPU) είτε με την rand()(CPU)

3) **print_matrix():**

Τυπώνει στην οθόνη τα στοιχεία του πίνακα εισόδου(της συνάρτησης).

4) **gpu_blas_mmul()/__global__ void kernel():**

Πραγματοποιεί τους απαραίτητους υπολογισμούς είτε χρησιμοποιώντας την βιβλιοθήκη cuBLAS, είτε με ή χωρίς βελτιστοποιήσεις.

5) **main():**

Καλεί τις απαραίτητες εντολές-συναρτήσεις για να υλοποιηθεί ο αντίστοιχος αλγόριθμος.

Μετρήσεις

Μέγεθος Μητρώου (n x m)	cuBLAS (ms)	Χωρίς Βελτιστοποιήσεις (ms)	Με Βελτιστοποιήσεις (ms)
50 x 100	0.026272	0.210016	0.319296
100 x 100	0.028352	0.400928	0.643904
250 x 100	0.037408	0.978944	1.564192
250 x 500	0.217184	22.889952	22.371328
1000 x 1000	2.171648	717.970276	337.039612
1500 x 1500	7.192608	2389.531494	1113.064575
2000 x 2000	16.750912	5707.302734	2605.327148