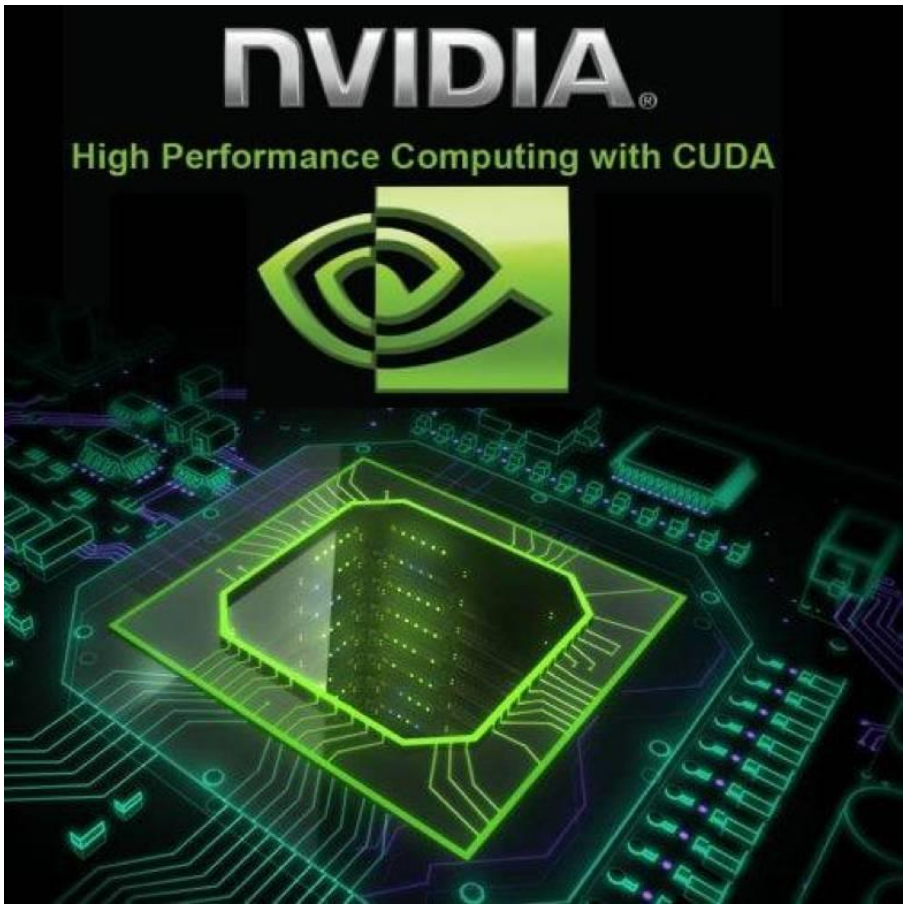
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

Λογισμικό και Προγραμματισμός Συστημάτων Υψηλής Επίδοσης

Χειμερινό Εξάμηνο 2017-18



Ομάδα 23:

Κανούτος Κωνσταντίνος 5775
Κυριακού Ανδρόνικος 5806
Ντενέζος Παναγιώτης 5853
Πλούμης Θωμάς 5880

Περιεχόμενα

Εισαγωγή 3
Μέρος 1ο 4
Μέρος 2ο 7
Μετρήσεις 11
Συμπεράσματα 18
Βιβλιογραφία 20

Εισαγωγή

Στα πλαίσια της εργασίας έγινε μια υλοποίησή του αλγορίθμου Principal Direction Divisive Partitioning (PDDP). Ο αλγόριθμος PDDP δέχεται για είσοδο ένα μητρώο, του οποίου κάθε στήλη αντιστοιχεί σε ένα στοιχείο του συνόλου δεδομένων, με σκοπό να καταλήξουμε σε ομαδοποίηση των δεδομένων αυτών.

Αρχικά, πρέπει να υπολογιστεί ο μέσος όρος όλων των στοιχείων δεδομένων. Ο υπολογισμός αυτός γίνεται ως εξής:

$$w = \frac{1}{m} * (d_1 + d_2 + \cdots + d_m)$$

Αν θεωρήσουμε διάνυσμα $e = (111 \ldots 1)^T$, τότε το w μπορεί να εκφραστεί ως:

$$w = \frac{1}{m} * (d_1 + d_2 + \cdots + d_m) = \frac{1}{m} * M * e$$

Στη συνέχεια, για να γίνει ο υπολογισμός του μητρώου C, πρέπει να αφαιρεθεί από κάθε στήλη του μητρώου M ο μέσος όρος των στοιχείων εισόδου, ώστε να σχηματιστεί το μητρώο A και το αποτέλεσμα να πολλαπλασιαστεί με τον ανάστροφο. Δηλαδή,

$C = (M - w * e^T)^T * (M - w * e^T) = A^T * A$, όπου $A = (M - w * e^T)$

Επειδή το μητρώο C έχει διαστάσεις $m \ x \ m$, με το m να μπορεί να είναι οποιοσδήποτε αριθμός, ακόμα και τεραστίας τάξης μεγέθους, καθίσταται αδύνατη η αποθήκευση του στην μνήμη και άρα δεν μπορεί να υπολογισθεί ρητά. Το ίδιο ισχύει και για το μητρώο A. Άρα δεν γίνεται να δημιουργηθεί μητρώο για την αποθήκευση της ενδιάμεσης πράξης. Έτσι προτιμάται να γίνει ομαδοποίηση δεδομένων και να υπολογίζεται κάθε φορά ένα κομμάτι των μητρώων.

Για την υλοποίηση του αλγορίθμου, χρειάζεται να υπολογιστεί το ιδιοδιάνυσμα που αντιστοιχεί στην μεγαλύτερη ιδιοτιμή του μητρώου C. Αυτό επιτυγχάνεται με την επαναληπτική μέθοδο “power iteration”. Ο τύπος που χρησιμοποιείται είναι:

$$x_{k+1} = \frac{C * x_k}{\|C * x_k\|} = \frac{(M - w * e^T)^T * (M - w * e^T) * x_k}{\|(M - w * e^T)^T * (M - w * e^T) * x_k\|}$$

όπου $x_0 = (111 \ldots 1)$

Για την επίλυση του παραπάνω τύπου επιλέξαμε οι πράξεις να γίνονται από τα δεξιά προς τα αριστερά, με σκοπό να αποφύγουμε τις πράξεις μητρώο με μητρώο και να υλοποιήσουμε τις λιγότερο απαιτητικές, αυτές μητρώο με διάνυσμα.

Ο αλγόριθμος τερματίζει (συγκλίνει) όταν το νέο διάνυσμα έχει μικρή διαφορά από το προηγούμενο διάνυσμα. Μετά την ολοκλήρωση του αλγορίθμου το τελευταίο διάνυσμα είναι το ζητούμενο ιδιοδιάνυσμα. Ο πολλαπλασιασμός του μητρώου *C* με ένα διάνυσμα από δεξιά λύνει το πρόβλημα του μη ρητού υπολογισμού των μητρώων *C* και *A*.

Μέρος 1ο

Για την επίλυση του πρώτου μέρους έγινε μια απλή υλοποίηση σε CUDA. Τα στοιχεία αποθηκευτήκαν στην καθολική μνήμη (global memory) και χρησιμοποιήθηκαν κάποιοι καταχωρητές τοπικά στα threads.

Αρχικά, φορτώνονται δεδομένα από τα datasets στο μητρώο *objects*. Ο αριθμός των blocks και των threads που χρησιμοποιούνται υπολογίζεται δυναμικά με βάση το μέγεθος εισόδου. Γίνεται αρχικοποίηση όλων των μεταβλητών που θα χρησιμοποιηθούν και δεσμεύεται ο κατάλληλος χώρος στην μνήμη του *host* και του *device* με χρήση της συνάρτησης *malloc* και *cudaMalloc* αντίστοιχα. Στη συνέχεια, γίνεται αντιγραφή μεταξύ των μεταβλητών από την μνήμη του host με τις μεταβλητές της μνήμης του device με χρήση της συνάρτησης *cudaMemcpy*. Όπου χρειάζεται να επιστρέψουν αποτελέσματα από το device στο host γίνεται πάλι χρήση της *cudaMemcpy*.

Για την συνέχεια της εκτέλεσης παρουσιάζονται μια μια οι συναρτήσεις πυρήνα που χρησιμοποιήθηκαν με τη σειρά εμφάνισης τους μέσα στον κώδικα.

- *ones*

  Η συνάρτηση αρχικοποιεί το διάνυσμα x με την τιμή 1. Κάθε thread θέτει την τιμή αυτή σε ένα στοιχείο του διανύσματος x.

- *w_calc*

  Υπολογίζει το *w*, μέσω της *w_calc*, όπου αναθέτει σε κάθε thread να υπολογίσει το άθροισμα κάθε γραμμής του μητρώου *M*.

- *pddp1*

  Υπολογίζει τον αριθμητή (από δεξιά προς αριστερά), δηλαδή του $(M - w * e^T) * x_k$. Η συνάρτηση αναλαμβάνει να επιστρέψει ένα διάνυσμα *tmp*, του οποίου η κάθε θέση υπολογίζεται από ένα thread.

- *pddp2*

  Ολοκληρώνει τον υπολογισμό του αριθμητή. Πιο συγκεκριμένα, η συνάρτηση εκτελεί τον πολλαπλασιασμό του $(M - w * e^T)^T$ με το διάνυσμα *tmp* που επιστρέφει η pddp1.

- *power*

  Η συνάρτηση υψώνει κάθε στοιχείο του μητρώου εισόδου στο τετράγωνο. Κάθε thread αναλαμβάνει την πράξη αυτή για κάθε ένα στοιχείο ξεχωριστά.

- *division*

  Η συνάρτηση διαιρεί κάθε στοιχείο του μητρώου εισόδου με μια σταθερά που δίνεται σαν όρισμα. Κάθε thread αναλαμβάνει την πράξη αυτή για κάθε ένα στοιχείο ξεχωριστά.

- *diff_pow*

  Κάνει μια αφαίρεσή δυο διανυσμάτων στοιχείο προς στοιχείο και υψώνει το κάθε στοιχείο του τελικού διανύσματος στο τετράγωνο. Κάθε thread αναλαμβάνει την πράξη αυτή για κάθε ένα στοιχείο ξεχωριστά.

- *swap*

  Αντιγράφει ένα διάνυσμα σε ένα άλλο διάνυσμα στοιχείο προς στοιχείο. Κάθε thread αναλαμβάνει την πράξη αυτή για κάθε ένα στοιχείο ξεχωριστά.

Πιο συγκεκριμένα, ξεκινάει η εκτέλεση του προγράμματος αρχικοποιώντας το διάνυσμα x με τη τιμή 1 (συνάρτηση *ones*). Στη συνέχεια, υπολογίζεται το *w* με χρήση της w_*calc* στο device και επιστρέφει στον κώδικα το διάνυσμα *w_d*, το οποίο αντιγράφεται στο *w*. Έπειτα ξεκινάει μια επαναληπτική μέθοδος για τον υπολογισμό του ιδιοδιανύσματος του μητρώου C, μέχρι η νόρμα, *fnorm*, του x_{k+1} να είναι μικρότερη από 10^{-6}. Μέσα στην επαναληπτική μέθοδο αρχικά, καλείται η *pddp1*, για τον υπολογισμό του πρώτου μέρους του αριθμητή επιστρέφοντας πίσω στο host ένα διάνυσμα *tmp*. Στη συνέχεια, καλείται η *pddp2*, η οποία ολοκληρώνει τον υπολογισμό του αριθμητή, επιστρέφοντας ένα διάνυσμα *final_d* που αντιγράφεται σε ένα διάνυσμα *final*. Το διάνυσμα αυτό (*final*) στέλνεται μέσω της *power* στο device για να υψωθούν όλα τα στοιχεία του στο τετράγωνο και να υπολογιστεί στη συνέχεια η νόρμα του παρονομαστή. Η *power* επιστρέφει ένα διάνυσμα *den_d*, με τα στοιχεία υψωμένα στο τετράγωνο και αντιγράφεται σε ένα διάνυσμα *den*.Μια επαναληπτικής δομής *for* αθροίζει όλα τα στοιχεία του δίνοντας μας το άθροισμα όλων των τετραγώνων των (*f_sum*) και στην πορεία υπολογίζεται η ρίζα του αθροίσματος αυτού. Η κλήση της *division* διαιρεί κάθε στοιχείο του διανύσματος *final* με τη νόρμα του αριθμητή. Έπειτα, η συνάρτηση *diff_pow* υπολογίζει τη νόρμα της διαφοράς των ιδιοδιανυσμάτων x_{k+1} και x_k, ώστε να ελεγχθεί αν η διαφορά είναι μικρότερη του 10^{-6}. Τέλος, γίνεται η ενημέρωση του *x* διανύσματος στο host από το *x_d* του device, ώστε να ενημερωθεί το διάνυσμα αυτό με τις καινούργιες τιμές, προκειμένου να χρησιμοποιηθεί στην επόμενη επανάληψη. Αφού ολοκληρωθεί ο επαναληπτικός βρόχος προτού τερματίσει ο αλγόριθμος γίνεται απελευθέρωση των πόρων τόσο στο device με χρήση του *cudaFree*, όσο και στο host με χρήση του *free*.

Μέρος 2ο

Για την υλοποίησης αυτού του ερωτήματος έγινε αξιοποίηση της ιεραρχίας μνήμης της CUDA και χρησιμοποιήθηκαν καταχωρητές και κοινή μνήμη (shared memory) ώστε να επιτευχθεί βελτιστοποίηση στην απόδοση του αλγορίθμου.

Λάβαμε υπόψιν μας ότι η προσπέλαση στοιχείων του ιδίου μητρώου έγινε ταυτόχρονα τόσο κατά στήλες όσο και κατά γραμμές. Αυτό συμβαίνει τόσο κατά την εκμετάλλευση της κοινής μνήμης όσο και για την αξιοποίηση της προσπέλασης συνεχόμενων στοιχείων στην καθολική μνήμη (global memory).
Οι αλλαγές έγιναν στις συναρτήσεις πυρήνα *pddp1* και *pddp2*. Επίσης, δημιουργήθηκαν οι συναρτήσεις πυρήνα *atomicAdd* και *zeros*. Πιο συγκεκριμένα οι συναρτήσεις αυτές υλοποιούν:

- *atomicAdd*
  Η ενσωματωμένη *atomicAdd* της CUDA είναι για integers, οπότε υλοποιήθηκε αντίστοιχη συνάρτηση για double.

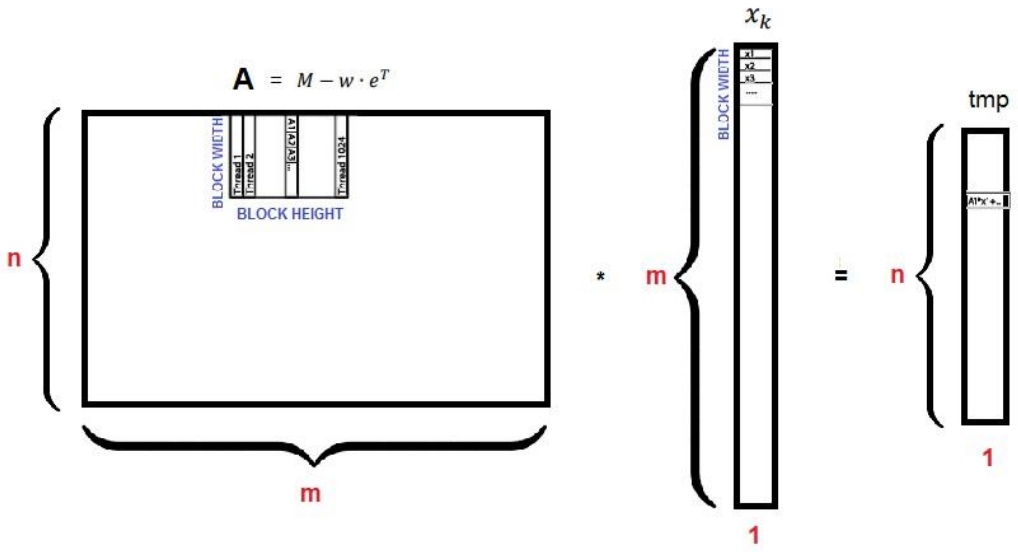
- *zeros*
  Αρχικοποιεί όλα τα διανύσματα με την τιμή 0, καθώς είναι απαραίτητη η αρχικοποίηση όλων των μεταβλητών στην κοινή μνήμη.

Όσον αφορά την *pddp1* και *pddp2* πέρα από τις αλλαγές μέσα στον κώδικα των συναρτήσεων άλλαξε και η προσέγγιση της λύσης του προβλήματος. Αρχικά, άλλαξε το πλήθος των threads και το μέγεθος του grid και των block. Για την επιλογή του μεγέθους των block έγιναν δοκιμές διαφόρων συνδυασμών των *BLOCK_HEIGHT* και *BLOCK_WIDTH* και καταλήξαμε στο βέλτιστο. Πιο συγκεκριμένα, δημιουργούμε *BLOCK_HEIGHT* threads, που το κάθε ένα υπολογίζει το εσωτερικό γινόμενο δυο διανυσμάτων μεγέθους *BLOCK_WIDTH*, το οποίο προστίθεται στην κατάλληλη θέση του τελικού διανύσματος. Αυξάνοντας το μέγεθος των *BLOCK_WIDTH*, εμφανίζεται μια αντιστάθμιση μεταξύ υπολογιστικής πολυπλοκότητας και προσπελάσεων μνήμης. Με την αύξηση του *BLOCK_WIDTH*, αυξάνεται το μέγεθος της κοινής μνήμης που χρησιμοποιείται σε κάθε block. Από την άλλη, όλα τα *BLOCK_HEIGHT* threads

μοιράζονται αυτό το κομμάτι του διανύσματος στον πολλαπλασιασμό με αποτέλεσμα να επιτυγχάνεται λιγότερη πρόσβαση στην μνήμη. Χρησιμοποιήθηκε, ακόμα, block 2-διαστάσεων με την x-διάσταση να είναι $^{m}/_{\text{BLOCK\_HEIGHT}}$ και την y-διάσταση $^{n}/_{\text{BLOCK\_WIDTH}}$ για την *pddp1* και αντίστροφα για την *pddp2*.

- *pddp1*

  Για την *pddp1*, γίνεται μια θεωριτική αναστροφή του μητρώου *A*, καθώς τα δεδομένα μας στην μνήμη είναι αποθηκευμένα κατά γραμμές (row major order). Συνεπώς, για να προσπελάσουν τα threads τα δεδομένα κατά στήλες θα πρέπει να γίνει η συγκεκριμένη προσέγγιση. Κατά τ' άλλα, η δομή του μητρώου παραμένει όπως φαίνεται και στην παρακάτω εικόνα.



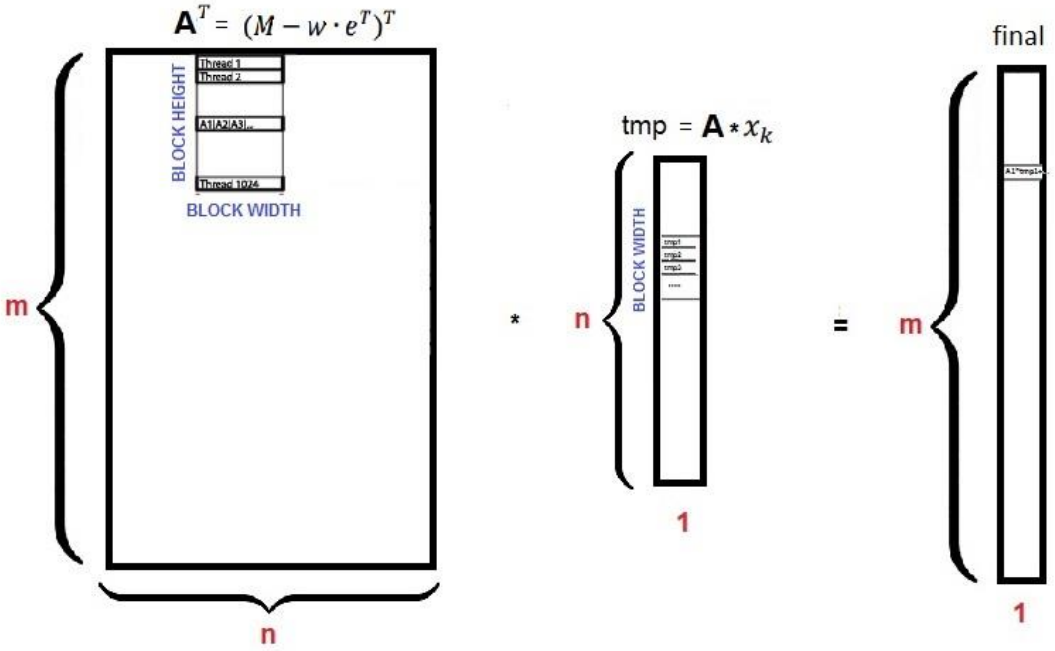
Thread 0	Thread 1	Thread 2	Thread 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Κάθε block υπολογίζει το workload του και ελέγχεται αν το τελευταίο block θα ξεπεράσει τα όρια του μητρώου. Ο έλεγχος αυτός γίνεται για να υποστηριχθούν μητρώα γενικού μεγέθους και όχι μόνο μητρώα μεγέθους δυνάμεων του 2. Στην shared memory αποθηκεύεται ένα μέρος του μητρώου x_kμεγέθους BLOCK_WIDTH για να είναι προσπελάσιμο ταυτόχρονα από όλα τα threads του ίδιου μπλοκ. Τέλος, στο κάθε block, τα threads αναλαμβάνουν τα δεδομένα κατά στήλες, ώστε να επιτευχθεί η πράξη του πολλαπλασιασμού.

Στον παραπάνω πίνακα εμφανίζεται ο τρόπος προσπέλασης της μνήμης από τα threads. Όπως φαίνεται υλοποιείται συγκερασμένη προσπέλαση καθώς οι διευθύνσεις που προσπελαύνει το κάθε thread απέχουν σταθερό αριθμό θέσεων μεταξύ τους. Έτσι το υλικό ανιχνεύει αυτή την διαδοχική προσπέλαση θέσεων και τις ενοποιεί σε μια συγκερασμένη.

- *pddp2*

  Για την *pddp2*, δεν γίνεται θεωρητική αναστροφή του μητρώου *A*, αλλά κανονική, καθώς πρέπει να υπολογίζεται η ανεστραμμένη μορφή του μητρώου A. Συνεπώς, για να προσπελάσουν τα threads τα δεδομένα κατά γραμμές θα ισχύει ότι φαίνεται και στην παρακάτω εικόνα.



Thread 0	0	4	8	12
Thread 1	1	5	9	13
Thread 2	2	6	10	14
Thread 3	3	7	11	15

Κάθε block υπολογίζει το workload του και ελέγχεται αν το τελευταίο block θα ξεπεράσει τα όρια του μητρώου. Ο έλεγχος αυτός γίνεται για να υποστηριχθούν μητρώα γενικού μεγέθους και όχι μόνο μητρώα μεγέθους δυνάμεων του 2. Τέλος, στο κάθε block, τα threads αναλαμβάνουν τα δεδομένα κατά γραμμές, ώστε να επιτευχθεί η πράξη του πολλαπλασιασμού.

Μετρήσεις

Παρακάτω παρατίθενται τα αποτελέσματα των μετρήσεων.

```
team23@tesla:~/cuda-workspace/Mytest$ gcc original.c file_io.c -lm -o original
team23@tesla:~/cuda-workspace/Mytest$ ./original /home/datasets/dataset1.csv
::Objects loaded::
Objects: 100000
Attributes: 20
y[99995] = 0.0042479
y[99996] = -0.0026528
y[99997] = 0.0010918
y[99998] = -0.0057215
y[99999] = -0.0005276
Rounds = 29
Time = 1259.897000 msec
team23@tesla:~/cuda-workspace/Mytest$ ./original /home/datasets/dataset2.csv
::Objects loaded::
Objects: 1000000
Attributes: 20
y[999995] = -0.0013410
y[999996] = 0.0008380
y[999997] = -0.0003451
y[999998] = 0.0018080
y[999999] = 0.0001665
Rounds = 46
Time = 18367.098000 msec
team23@tesla:~/cuda-workspace/Mytest$ ./original /home/datasets/dataset3.csv
::Objects loaded::
Objects: 1000000
Attributes: 60
y[999995] = 0.0013391
y[999996] = -0.0008382
y[999997] = 0.0003450
y[999998] = -0.0018090
y[999999] = -0.0001637
Rounds = 48
Time = 64726.811000 msec
team23@tesla:~/cuda-workspace/Mytest$
```

Εικόνα 1: Μετρήσεις χρόνων σειριακής εκτέλεσης.

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part1 /home/datasets/dataset1.csv
==7503== NVPROF is profiling process 7503, command: ./part1 /home/datasets/dataset1.csv
::Objects loaded::
Objects: 100000
Attributes: 20
Grid Size: (1,3125)
Block Size: (1,32)
-----------------
Elapsed Time=990.602234 ms
-----------------
y[99995] = 0.0042479
y[99996] = -0.0026528
y[99997] = 0.0010918
y[99998] = -0.0057215
y[99999] = -0.0005276
==7503== Profiling application: ./part1 /home/datasets/dataset1.csv
==7503== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 92.43%  931.22ms        29  32.111ms  32.110ms  32.113ms  pddp1(double*, int, int, double*, double*, double*)
  3.62%  36.491ms       176  207.33us  1.6950us  261.75us  [CUDA memcpy DtoH]
  1.82%  18.367ms         1  18.367ms  18.367ms  18.367ms  w_calc(double*, int, int, double*)
  0.76%  7.7046ms        32  240.77us  132.57us  3.5613ms  [CUDA memcpy HtoD]
  0.74%  7.4147ms        29  255.68us  254.55us  256.67us  pddp2(double*, double*, int, int, double*, double*)
  0.19%  1.9370ms        29  66.793us  66.558us  67.012us  diff_pow(double*, int, double*)
  0.19%  1.9263ms        29  66.422us  66.259us  66.561us  power(double*, int, double*)
  0.14%  1.4294ms        29  49.290us  48.893us  49.633us  division(double*, int, double, double*)
  0.10%  1.0100ms        29  34.825us  34.630us  35.133us  swap(double*, int, double*)
  0.00%  21.314us         1  21.314us  21.314us  21.314us  ones(double*, int)

==7503== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.23%  969.93ms       234  4.1450ms  5.4460us  32.279ms  cudaEventSynchronize
  7.21%  81.055ms       208  389.69us  20.200us  3.6619ms  cudaMemcpy
  5.84%  65.696ms         8  8.2120ms  11.417us  64.725ms  cudaMalloc
  0.21%  2.3235ms       176  13.201us  11.673us  38.214us  cudaLaunch
  0.18%  1.9907ms       468  4.2530us  3.0810us  16.027us  cudaEventRecord
  0.08%  863.03us       468  1.8440us     993ns  18.499us  cudaEventCreate
  0.07%  790.43us         8  98.803us  20.290us  255.46us  cudaFree
  0.05%  573.56us       468  1.2250us     843ns  10.969us  cudaEventDestroy
  0.05%  545.28us       166  3.2840us     222ns  130.11us  cuDeviceGetAttribute
  0.04%  461.76us       234  1.9730us  1.7080us  4.1110us  cudaEventElapsedTime
  0.02%  236.89us       731     324ns     211ns  5.4770us  cudaSetupArgument
  0.01%  93.600us       176     531ns     420ns  1.3780us  cudaConfigureCall
  0.01%  70.775us         2  35.387us  35.287us  35.488us  cuDeviceTotalMem
  0.01%  59.216us       118     501ns     370ns  5.0070us  cudaGetLastError
  0.00%  55.090us         2  27.545us  25.647us  29.443us  cuDeviceGetName
  0.00%  6.8900us         1  6.8900us  6.8900us  6.8900us  cudaSetDevice
  0.00%  1.7230us         2     861ns     430ns  1.2930us  cuDeviceGetCount
  0.00%  1.3590us         4     339ns     263ns     384ns  cuDeviceGet
```

Εικόνα 2: Μετρήσεις μέρους Α για dataset1

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part2 /home/datasets/dataset1.csv
==11929== NVPROF is profiling process 11929, command: ./part2 /home/datasets/dataset1.csv
::Objects loaded::
Objects: 100000
Attributes: 20
Grid Size: (1,3125)
Block Size: (1,32)
-----------------
Elapsed_Time=103.332397 ms
-----------------
y[99995] = 0.0042479
y[99996] = -0.0026528
y[99997] = 0.0010918
y[99998] = -0.0057215
y[99999] = -0.0005276
==11929== Profiling application: ./part2 /home/datasets/dataset1.csv
==11929== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 40.17%  58.042ms       164  353.92us  1.7920us  427.69us  [CUDA memcpy DtoH]
 32.53%  47.008ms        27  1.7411ms  1.6965ms  1.7732ms  pddp1(double*, int, int, double*, double*, double*)
 12.71%  18.367ms         1  18.367ms  18.367ms  18.367ms  w_calc(double*, int, int, double*)
  6.08%  8.7810ms        30  292.70us  171.06us  3.7924ms  [CUDA memcpy HtoD]
  4.17%  6.0294ms        27  223.31us  222.18us  225.46us  pddp2(double*, double*, int, int, double*, double*)
  1.25%  1.8074ms        27  66.940us  66.695us  67.185us  diff_pow(double*, int, double*)
  1.24%  1.7935ms        27  66.426us  66.228us  66.684us  power(double*, int, double*)
  0.92%  1.3306ms        27  49.281us  48.940us  49.572us  division(double*, int, double, double*)
  0.65%  938.93us        27  34.775us  34.505us  35.055us  swap(double*, int, double*)
  0.26%  379.13us        54  7.0200us  1.7950us  12.609us  zeros(double*, int)
  0.01%  21.044us         1  21.044us  21.044us  21.044us  ones(double*, int)

==11929== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 37.30%  100.79ms       194  519.55us  19.726us  4.7236ms  cudaMemcpy
 30.96%  83.642ms       218  383.68us  6.0510us  18.395ms  cudaEventSynchronize
 28.21%  76.212ms         8  9.5265ms  8.4050us  75.221ms  cudaMalloc
  0.99%  2.6870ms       436  6.1620us  3.4510us  28.305us  cudaEventRecord
  0.96%  2.5878ms       218  11.870us  8.2570us  43.647us  cudaLaunch
  0.30%  803.09us       436  1.8410us  1.0900us  11.257us  cudaEventCreate
  0.29%  785.69us         8  98.210us  10.146us  219.00us  cudaFree
  0.23%  613.04us       166  3.6920us     231ns  137.87us  cuDeviceGetAttribute
  0.19%  525.59us       436  1.2050us     830ns  2.7880us  cudaEventDestroy
  0.15%  400.89us         1  400.89us  400.89us  400.89us  cudaGetDeviceProperties
  0.15%  398.66us       218  1.8280us  1.6160us  3.2210us  cudaEventElapsedTime
  0.11%  293.55us       789     372ns     242ns  2.6760us  cudaSetupArgument
  0.05%  122.87us       218     563ns     359ns  6.2000us  cudaConfigureCall
  0.05%  121.79us        54  2.2550us  1.9260us  4.9950us  cudaThreadSynchronize
  0.03%  75.660us         2  37.830us  35.917us  39.743us  cuDeviceTotalMem
  0.02%  63.997us         2  31.998us  26.635us  37.362us  cuDeviceGetName
  0.02%  55.976us       110     508ns     366ns  6.2690us  cudaGetLastError
  0.00%  6.4230us         1  6.4230us  6.4230us  6.4230us  cudaSetDevice
  0.00%  2.0350us         2  1.0170us     364ns  1.6710us  cuDeviceGetCount
  0.00%  1.2980us         4     324ns     270ns     389ns  cuDeviceGet
```

Εικόνα 3: Μετρήσεις μέρους Β για dataset1

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part1 /home/datasets/dataset2.csv
==10176== NVPROF is profiling process 10176, command: ./part1 /home/datasets/dataset2.csv
::Objects loaded::
Objects: 1000000
Attributes: 20
Grid Size: (1,31250)
Block Size: (1,32)
-----------------
Elapsed_Time=15533.261719 ms
-----------------
y[999995] = -0.0013410
y[999996] = 0.0008380
y[999997] = -0.0003451
y[999998] = 0.0018080
y[999999] = 0.0001665
==10176== Profiling application: ./part1 /home/datasets/dataset2.csv
==10176== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.10%  14.7652s        46  320.98ms  320.98ms  320.99ms  pddp1(double*, int, int, double*, double*, double*)
  3.68%  583.89ms       278  2.1003ms  1.6950us  3.6120ms  [CUDA memcpy DtoH]
  1.16%  183.58ms         1  183.58ms  183.58ms  183.58ms  w_calc(double*, int, int, double*)
  0.74%  116.64ms        46  2.5356ms  2.5325ms  2.5386ms  pddp2(double*, double*, int, int, double*, double*)
  0.72%  113.98ms        49  2.3260ms  1.4709ms  35.244ms  [CUDA memcpy HtoD]
  0.19%  29.643ms        46  644.41us  643.78us  645.11us  power(double*, int, double*)
  0.19%  29.461ms        46  640.47us  638.28us  642.09us  diff_pow(double*, int, double*)
  0.14%  21.752ms        46  472.88us  472.30us  473.48us  division(double*, int, double, double*)
  0.10%  15.423ms        46  335.29us  334.70us  335.78us  swap(double*, int, double*)
  0.00%  198.73us         1  198.73us  198.73us  198.73us  ones(double*, int)

==10176== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 94.75%  15.1734s       370  41.009ms  4.9160us  321.12ms  cudaEventSynchronize
  4.71%  755.00ms       327  2.3089ms  24.759us  35.371ms  cudaMemcpy
  0.45%  71.967ms         8  8.9959ms  11.326us  70.884ms  cudaMalloc
  0.03%  5.0256ms       278  18.077us  12.887us  72.915us  cudaLaunch
  0.02%  3.7883ms       740  5.1190us  3.0570us  47.375us  cudaEventRecord
  0.01%  1.5017ms       740  2.0290us     925ns  15.064us  cudaEventCreate
  0.01%  915.50us         8  114.44us  21.384us  350.59us  cudaFree
  0.01%  899.82us       740  1.2150us     812ns  2.3590us  cudaEventDestroy
  0.01%  825.89us       370  2.2320us  1.6470us  6.2100us  cudaEventElapsedTime
  0.00%  565.75us       166  3.4080us     209ns  137.01us  cuDeviceGetAttribute
  0.00%  427.56us      1156     369ns     217ns  5.2440us  cudaSetupArgument
  0.00%  216.78us       278     779ns     414ns  11.525us  cudaConfigureCall
  0.00%  110.80us         2  55.399us  34.598us  76.200us  cuDeviceTotalMem
  0.00%  104.91us       186     564ns     387ns  4.4620us  cudaGetLastError
  0.00%  60.448us         2  30.224us  25.351us  35.097us  cuDeviceGetName
  0.00%  7.3480us         1  7.3480us  7.3480us  7.3480us  cudaSetDevice
  0.00%  1.7930us         2     896ns     396ns  1.3970us  cuDeviceGetCount
  0.00%  1.3350us         4     333ns     278ns     410ns  cuDeviceGet
```

Εικόνα 4: Μετρήσεις μέρους Α για dataset2

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part2 /home/datasets/dataset2.csv
==11874== NVPROF is profiling process 11874, command: ./part2 /home/datasets/dataset2.csv
::Objects loaded::
Objects: 1000000
Attributes: 20
Grid Size: (1,31250)
Block Size: (1,32)
------------------
Elapsed_Time=1544.008545 ms
------------------
y[999995] = -0.0013410
y[999996] = 0.0008380
y[999997] = -0.0003451
y[999998] = 0.0018080
y[999999] = 0.0001665
==11874== Profiling application: ./part2 /home/datasets/dataset2.csv
==11874== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 43.39%  1.00262s       278  3.6065ms  1.7910us  4.5625ms  [CUDA memcpy DtoH]
 33.92%  783.82ms        46  17.040ms  16.621ms  17.260ms  pddp1(double*, int, int, double*, double*, double*)
  7.95%  183.66ms         1  183.66ms  183.66ms  183.66ms  w_calc(double*, int, int, double*)
  6.14%  141.96ms        49  2.8972ms  1.7263ms  38.271ms  [CUDA memcpy HtoD]
  4.17%  96.235ms        46  2.0921ms  2.0883ms  2.0948ms  pddp2(double*, double*, int, int, double*, double*)
  1.29%  29.757ms        46  646.90us  644.57us  648.46us  power(double*, int, double*)
  1.28%  29.505ms        46  641.42us  639.48us  642.73us  diff_pow(double*, int, double*)
  0.94%  21.770ms        46  473.26us  472.54us  474.16us  division(double*, int, double, double*)
  0.67%  15.435ms        46  335.55us  335.04us  336.06us  swap(double*, int, double*)
  0.24%  5.5484ms        92  60.308us  1.7920us  119.54us  zeros(double*, int)
  0.01%  198.81us         1  198.81us  198.81us  198.81us  ones(double*, int)

==11874== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 48.78%  1.20973s       327  3.6995ms  24.404us  38.364ms  cudaMemcpy
 47.50%  1.17794s       370  3.1836ms  5.4820us  183.68ms  cudaEventSynchronize
  3.02%  74.955ms         8  9.3694ms  8.1050us  73.820ms  cudaMalloc
  0.25%  6.0784ms       370  16.428us  9.1730us  44.194us  cudaLaunch
  0.19%  4.6888ms       740  6.3360us  3.0110us  55.026us  cudaEventRecord
  0.06%  1.5288ms       740  2.0660us     947ns  10.183us  cudaEventCreate
  0.04%  964.02us         8  120.50us  13.459us  415.70us  cudaFree
  0.04%  955.35us       740  1.2910us     824ns  3.8900us  cudaEventDestroy
  0.03%  808.99us       370  2.1860us  1.6010us  8.8700us  cudaEventElapsedTime
  0.02%  517.96us       166  3.1200us     212ns  112.80us  cuDeviceGetAttribute
  0.02%  495.85us      1340     370ns     219ns  2.5320us  cudaSetupArgument
  0.02%  424.10us         1  424.10us  424.10us  424.10us  cudaGetDeviceProperties
  0.01%  295.23us        92  3.2080us  1.9290us  6.0830us  cudaThreadSynchronize
  0.01%  224.26us       370     606ns     328ns  1.5300us  cudaConfigureCall
  0.00%  98.442us       186     529ns     395ns  6.1200us  cudaGetLastError
  0.00%  69.599us         2  34.799us  34.754us  34.845us  cuDeviceTotalMem
  0.00%  54.059us         2  27.029us  25.607us  28.452us  cuDeviceGetName
  0.00%  6.2030us         1  6.2030us  6.2030us  6.2030us  cudaSetDevice
  0.00%  1.6880us         2     844ns     368ns  1.3200us  cuDeviceGetCount
  0.00%  1.3380us         4     334ns     268ns     424ns  cuDeviceGet
```

Εικόνα 5: Μετρήσεις μέρους Β για dataset2

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part1 /home/datasets/dataset3.csv
==11675== NVPROF is profiling process 11675, command: ./part1 /home/datasets/dataset3.csv
::Objects loaded::
Objects: 1000000
Attributes: 60
Grid Size: (1,31250)
Block Size: (1,32)
-----------------
Elapsed_Time=16505.087891 ms
-----------------
y[999995] = 0.0013391
y[999996] = -0.0008382
y[999997] = 0.0003450
y[999998] = -0.0018090
y[999999] = -0.0001637
==11675== Profiling application: ./part1 /home/datasets/dataset3.csv
==11675== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.18%  15.4778s        48  322.46ms  322.18ms  322.65ms  pddp1(double*, int, int, double*, double*, double*)
  3.87%  657.41ms       290  2.2669ms  1.7270us  3.4880ms  [CUDA memcpy DtoH]
  2.03%  344.45ms        48  7.1760ms  7.1659ms  7.1837ms  pddp2(double*, double*, int, int, double*, double*)
  1.24%  210.79ms        51  4.1331ms  1.4646ms  105.14ms  [CUDA memcpy HtoD]
  1.09%  184.52ms         1  184.52ms  184.52ms  184.52ms  w_calc(double*, int, int, double*)
  0.18%  30.948ms        48  644.75us  644.04us  645.60us  power(double*, int, double*)
  0.18%  30.751ms        48  640.64us  639.18us  641.83us  diff_pow(double*, int, double*)
  0.13%  22.713ms        48  473.18us  472.62us  473.77us  division(double*, int, double, double*)
  0.09%  16.102ms        48  335.46us  334.81us  336.00us  swap(double*, int, double*)
  0.00%  198.75us         1  198.75us  198.75us  198.75us  ones(double*, int)

==11675== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.80%  16.1188s       386  41.759ms  4.6850us  322.74ms  cudaEventSynchronize
  5.62%  965.60ms       341  2.8317ms  21.745us  105.27ms  cudaMemcpy
  0.47%  81.113ms         8  10.139ms  10.876us  79.749ms  cudaMalloc
  0.04%  6.3491ms       290  21.893us  12.935us  56.336us  cudaLaunch
  0.03%  5.5245ms       772  7.1560us  3.0720us  47.221us  cudaEventRecord
  0.01%  1.6349ms       772  2.1170us     954ns  12.401us  cudaEventCreate
  0.01%  1.1553ms         8  144.41us  20.503us  563.25us  cudaFree
  0.01%  1.0005ms       772  1.2960us     800ns  4.8240us  cudaEventDestroy
  0.01%  923.12us       386  2.3910us  1.7340us  7.7950us  cudaEventElapsedTime
  0.00%  543.54us       166  3.2740us     213ns  123.31us  cuDeviceGetAttribute
  0.00%  517.28us      1206     428ns     223ns  4.7770us  cudaSetupArgument
  0.00%  279.94us       290     965ns     406ns  49.222us  cudaConfigureCall
  0.00%  110.24us       194     568ns     384ns  4.5030us  cudaGetLastError
  0.00%  72.393us         2  36.196us  31.461us  40.932us  cuDeviceTotalMem
  0.00%  56.387us         2  28.193us  24.264us  32.123us  cuDeviceGetName
  0.00%  7.2720us         1  7.2720us  7.2720us  7.2720us  cudaSetDevice
  0.00%  2.2300us         2  1.1150us     395ns  1.8350us  cuDeviceGetCount
  0.00%  1.3530us         4     338ns     252ns     413ns  cuDeviceGet
```

Εικόνα 6: Μετρήσεις μέρους Α για dataset3

```
team23@tesla:~/cuda-workspace/Mytest$ nvprof ./part2 /home/datasets/dataset3.csv
==11765== NVPROF is profiling process 11765, command: ./part2 /home/datasets/dataset3.csv
::Objects loaded::
Objects: 1000000
Attributes: 60
Grid Size: (1,31250)
Block Size: (1,32)
-----------------
Elapsed_Time=2074.050293 ms
-----------------
y[999995] = 0.0013391
y[999996] = -0.0008382
y[999997] = 0.0003450
y[999998] = -0.0018090
y[999999] = -0.0001637
==11765== Profiling application: ./part2 /home/datasets/dataset3.csv
==11765== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 37.66%  1.13080s        48  23.558ms  22.979ms  23.711ms  pddp1(double*, int, int, double*, double*, double*)
 35.01%  1.05124s       290  3.6250ms  1.8560us  4.6524ms  [CUDA memcpy DtoH]
  8.89%  266.92ms        48  5.5608ms  5.5487ms  5.5692ms  pddp2(double*, double*, int, int, double*, double*)
  8.75%  262.88ms        51  5.1546ms  2.1688ms  148.63ms  [CUDA memcpy HtoD]
  6.14%  184.43ms         1  184.43ms  184.43ms  184.43ms  w_calc(double*, int, int, double*)
  1.03%  31.059ms        48  647.06us  644.48us  648.15us  power(double*, int, double*)
  1.03%  30.794ms        48  641.55us  639.52us  642.74us  diff_pow(double*, int, double*)
  0.76%  22.726ms        48  473.46us  472.85us  474.04us  division(double*, int, double, double*)
  0.54%  16.098ms        48  335.36us  334.87us  335.90us  swap(double*, int, double*)
  0.19%  5.7667ms        96  60.069us  1.3080us  119.26us  zeros(double*, int)
  0.01%  198.70us         1  198.70us  198.70us  198.70us  ones(double*, int)

==11765== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 53.62%  1.70115s       386  4.4071ms  4.9510us  184.46ms  cudaEventSynchronize
 43.25%  1.37214s       341  4.0239ms  22.965us  148.82ms  cudaMemcpy
  2.60%  82.577ms         8  10.322ms  10.620us  81.205ms  cudaMalloc
  0.19%  6.0407ms       386  15.649us  8.9480us  39.120us  cudaLaunch
  0.12%  3.8711ms       772  5.0140us  3.1800us  48.024us  cudaEventRecord
  0.05%  1.7059ms       772  2.2090us  1.0820us  13.872us  cudaEventCreate
  0.04%  1.3598ms         8  169.97us  52.195us  753.85us  cudaFree
  0.03%  988.69us       772  1.2800us     824ns  2.5810us  cudaEventDestroy
  0.03%  833.73us       386  2.1590us  1.6190us  3.3840us  cudaEventElapsedTime
  0.02%  580.60us       166  3.4970us     257ns  122.81us  cuDeviceGetAttribute
  0.02%  519.27us      1398     371ns     230ns  2.3140us  cudaSetupArgument
  0.01%  320.22us         1  320.22us  320.22us  320.22us  cudaGetDeviceProperties
  0.01%  305.49us        96  3.1820us  2.0940us  4.4490us  cudaThreadSynchronize
  0.01%  230.58us       386     597ns     332ns  1.2330us  cudaConfigureCall
  0.00%  109.32us       194     563ns     393ns  4.0380us  cudaGetLastError
  0.00%  75.947us         2  37.973us  36.134us  39.813us  cuDeviceTotalMem
  0.00%  58.279us         2  29.139us  26.847us  31.432us  cuDeviceGetName
  0.00%  6.4890us         1  6.4890us  6.4890us  6.4890us  cudaSetDevice
  0.00%  1.7990us         2     899ns     366ns  1.4330us  cuDeviceGetCount
  0.00%  1.3400us         4     335ns     317ns     378ns  cuDeviceGet
```

Εικόνα 7: Μετρήσεις μέρους Β για dataset3

Συμπεράσματα

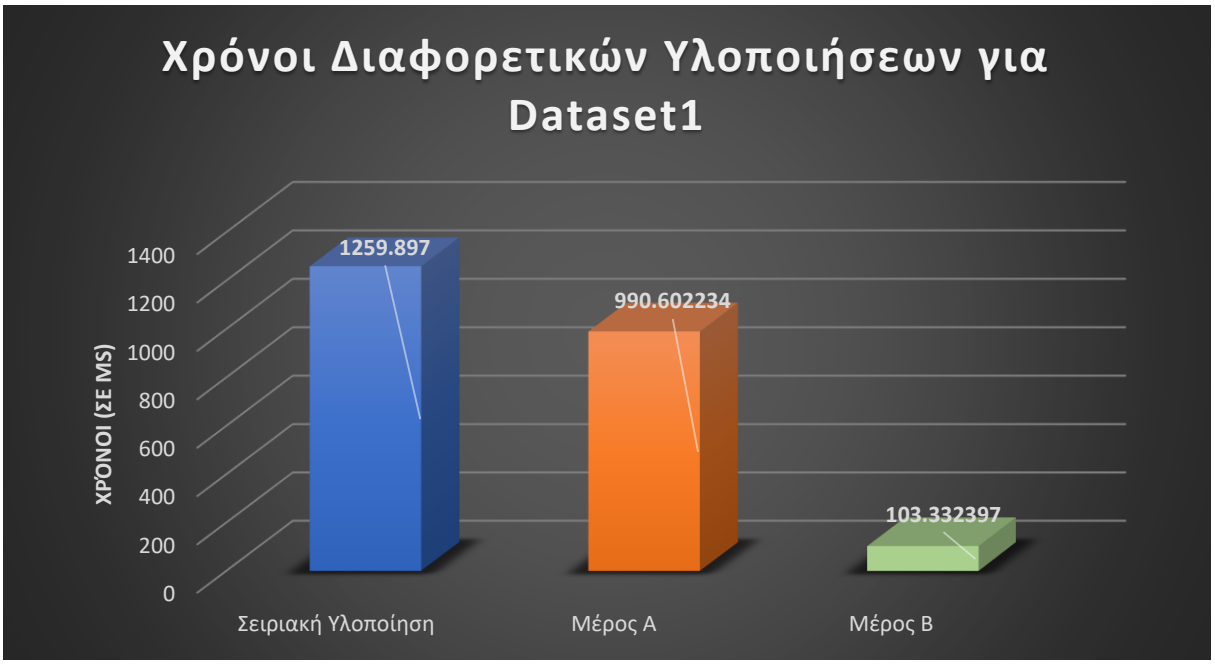
Τα συμπεράσματα των μετρήσεων οπτικοποιούνται σε πίνακα τιμών και έπειτα σε διαγράμματα.

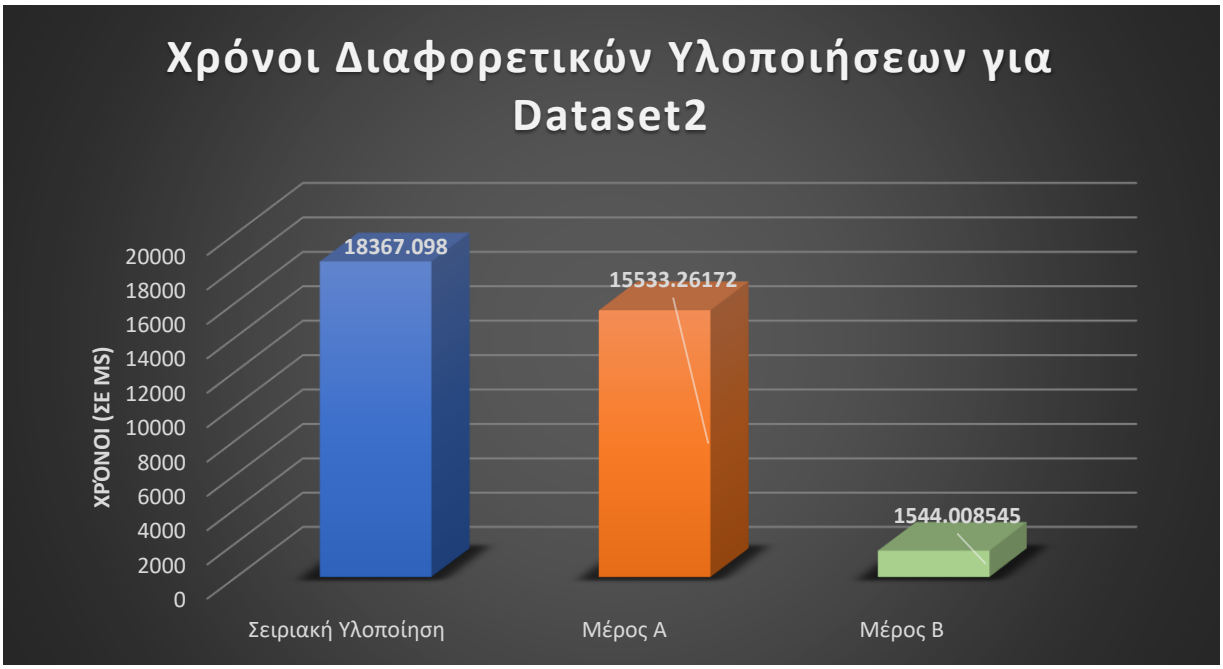
Υλοποιήση	Dataset1	Dataset2	Dataset3
Σειριακή	1259.897	18367.098	64726.811
Μέρος Α	990.602234	15533.26172	16505.08789
Μέρος Β	103.332397	1544.008545	2074.050293

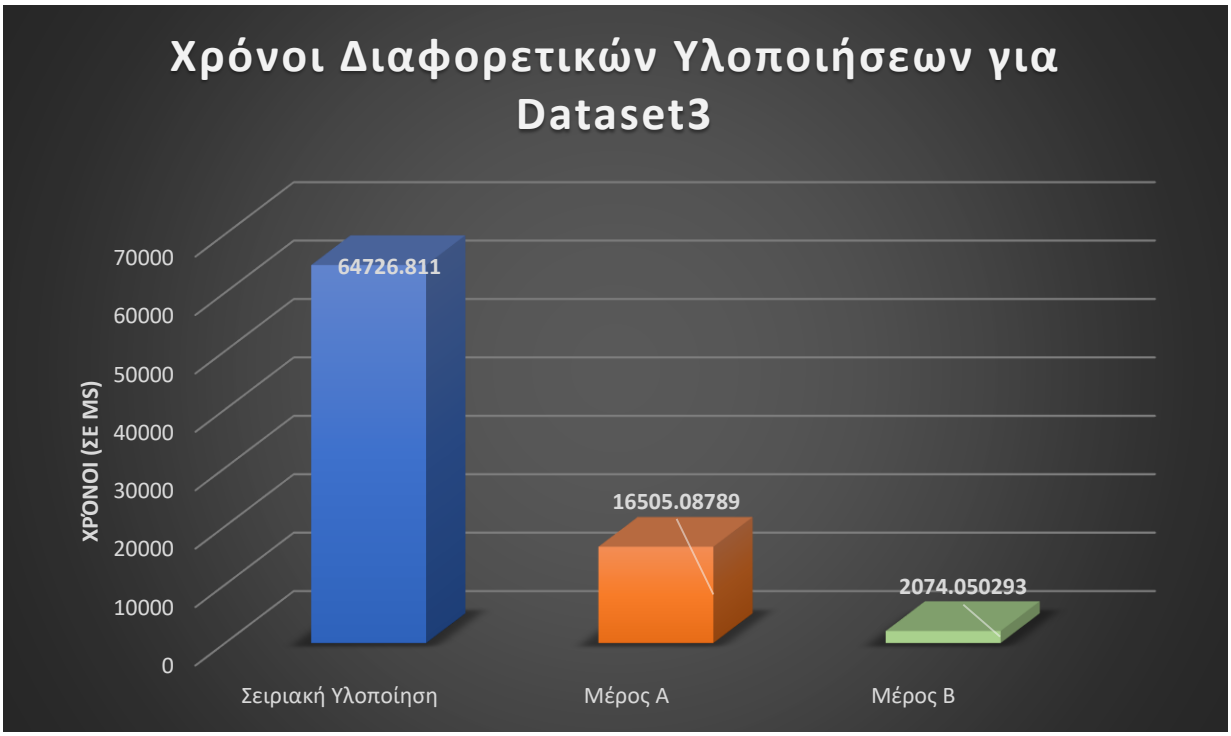
Οι τιμές των χρόνων στον πίνακα είναι σε ms.

Όλες οι παραπάνω μετρήσεις έγιναν με χρήση event της CUDA και λήφθηκε υπόψιν το σύνολο της υλοποίησης πέρα από τις αρχικοποιήσεις και τις μεταφορές.
Όποιες διαφορές μεταξύ του συνολικού χρόνου που προκύπτουν από τις χειροκίνητες μετρήσεις με του profiler (nvprof) οφείλονται σε εσωτερικές καθυστερήσεις του χρονομετρητή.

Ποσοστό βελτίωσης μέρους Β από σειριακή εκτέλεση	Ποσοστό βελτίωσης μέρους Β από μέρος Α
91.80%	89.57%
91.59%	90.06%
96.80%	87.43%


Χρόνοι Διαφορετικών Υλοποιήσεων για Dataset1
ΧΡΌΝΟΙ (ΣΕ MS)
1400
1200
1000
800
600
400
200
0
1259.897
990.602234
103.332397
Σειριακή Υλοποίηση
Μέρος Α
Μέρος Β


Χρόνοι Διαφορετικών Υλοποιήσεων για Dataset2
ΧΡΌΝΟΙ (ΣΕ MS)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
18367.098
15533.26172
1544.008545
Σειριακή Υλοποίηση
Μέρος Α
Μέρος Β



Συμπερασματικά, παρατηρείται δραστική διαφοροποίηση στους χρόνους των τριών διαφορετικών υλοποιήσεων, γεγονός που προκύπτει από την εκμετάλλευση της αρχιτεκτονικής της CUDA. Πιο συγκεκριμένα, η βελτίωση που εμφανίζεται με την υλοποίηση του μέρους Α στα πρώτα 2 datasets, όπου οι πίνακες είναι μικρού μεγέθους δεν είναι τόσο αισθητή όσο στο dataset 3, όπου το μητρώο είναι σημαντικά μεγαλύτερου μεγέθους. Σε αντίθεση, η υλοποίηση του μέρους Β, εμφανίζει μεγάλη βελτίωση στους χρόνους ακόμα και στα μικρά μητρώα των πρώτων 2 datasets, τόσο σε σύγκριση με την σειριακή εκτέλεση όσο και με αυτή του μέρους Α (περίπου 90%).

Βιβλιογραφία

NVIDIA. (2018, 2 4). *NVIDIA CUDA C Programming Guide.* Ανάκτηση από https://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf