Παράλληλα και Διανεμημένα Συστήματα

Εργασία 3

k nearest neighbours σε CUDA

Ζαμπόκας Γεώργιος

ΑΕΜ: 7173

Ημερομηνία: 29/1/2014

Λογική Υλοποίησης

Η πολυπλοκότητα του υπολογισμού των k nearest neighbours (knns) έγκειται σε δύο σημεία. Πρώτον, στον υπολογισμό όλων των αποστάσεων μεταξύ κάθε ερωτήματος (query) και κάθε σημείου του dataset, και δεύτερον στην επιλογή των k ελαχίστων αυτών αποστάσεων. Γι αυτό τον λόγο χρησιμοποιήσαμε την κάρτα γραφικών για να επιταχύνουμε τους υπολογισμούς εκτελώντας τους παράλληλα.

Αρχικά, να σημειωθεί ότι εκτελείται παράλληλη υλοποίηση για κάθε ερώτημα ξεχωριστά, δηλαδή για κάθε ερώτημα εκτελείται σειριακά ο παράλληλος κώδικας. Αυτό συμβαίνει γιατί δεν υπήρχε αρκετός χώρος στην μνήμη της κάρτας γραφικών για την αποθήκευση του πίνακα με τις αποστάσεις κάθε ερωτήματος από κάθε σημείο (numObjects * numQueries * numAttributes * sizeof(float))= 4 GB.
Δυστυχώς η κάρτα μου έχει 512 GB και του Διάδη 1.6 GB.

Έτσι λοιπόν για κάθε ερώτημα υπολογίζουμε την απόστασή του από κάθε σημείο του dataset. Για να γίνει αυτό παράλληλα, δημιουργείται ένα grid με μέγεθος όσα και τα σημεία (numObjects) με BlockSize threads στο καθένα (512). Οι αποστάσεις αποθηκεύονται σε έναν πίνακα distances μεγέθους numObjects φυσικά.

Τώρα που έχουμε στη διάθεσή μας όλες τις αποστάσεις χρησιμοποιούμε ένα ίδιου μεγέθους grid για να τις κάνουμε reduce σε έναν πίνακα ίσο με τα blocks τα οποία εκτελούν reduce επί τον αριθμό των γειτόνων που έχουμε επιλέξει. Το μέγεθος αυτού του πίνακα για τα δοσμένα όρια μπορεί να προκύψει από 1024 έως 16384 στοιχεία.

Τα στοιχεία αυτά πάλι θέλουν reduce για να προκύψει ένας μικρότερος πίνακας ελαχίστων αποστάσεων μεγέθους από 2 έως 256 στοιχεία. Τέλος, εφαρμόζεται ένα τελικό reduce σε ένα grid με 1 block και 2 έως 256 threads ώστε να προκύψουν και οι τελικοί γείτονες για το συγκεκριμένο ερώτημα. Αυτοί αποθηκεύονται στους πίνακες NN_dist (οι αποστάσεις τους) και NN_idx (τα indexes τους).

Είναι προφανές ότι τα reduces που περιγράφηκαν στην παραπάνω παράγραφο θα μπορούσαν να παραλειφθούν και να εκτελεστούν σειριακά ή με 1 thread. Υποθέτω πως δεν θα υπήρχε διαφορά στην ταχύτητα εκτέλεσης αφού τα στοιχεία είναι πολύ λίγα. Παρ'όλα αυτά υλοποιήθηκε έτσι ώστε όλη η δουλειά να γίνεται απο την GPU με την λογική του reduce που ζητήθηκε.

Η παραπάνω παράλληλη διαδικασία επαναλαμβάνεται για κάθε ερώτημα και προκύπτουν οι τελικοί NN_dist, NN_idx.

Ανάλυση του κώδικα

Δημιουργία dataset: Το dataset των σημείων που χρησιμποποίθηκε δημιουργήθηκε από το δοσμένο αρχείο με το matlab με το script που περιλαμβάνεται στην εργασία. Δημιουργήθηκαν 3 datasets σημείων (1m, 750k, 500k περίπου) και 11 datasets ερωτημάτων (1, 100, 200, 300,...1000).

theknn.cu: Το κυρίως πρόγραμμα. Ορίζεται η δομή knn_struct όπως και στο σεριακό. Ακόμη ορίζονται οι παρακάτω συναρτήσεις:

void error_message_fewer(),void error_message_more() :
Τυπώνουν μύνημα όταν εισάγεται λανθασμένος αριθμός ορισμάτων στην κονσόλα.

char* choose_data_file(int n), char* choose_query_file(int q):
Επιστρέφουν το κατάλληλο όνομα αρχείου για να διαβαστούν τα δεδομένα ανάλογα με τον αριθμό στοιχείων και τον αριθμό ερωτημάτων.

void save_distances(float* tmp_dataset, char *filename1,int n,int k),
void save_indexes(int* tmp_dataset, char *filename1,int n,int k):
Αποθηκεύουν τα δεδομένα που υπολογίστηκαν σε αρχεία όπως και στο σεριακό πρόγραμμα.

void cleanDevice(knn_struct *data):
Σβήνει την δεσμευμένη μνήμη.

__global__ void calculate_distances_seperately(float* dataset ,float* queries ,float* distances,int numObjects,int numAttributes,int numQueries,int k,int q):

Υπολογίζει παράλληλα όλες τις αποστάσεις ανάμεσα στο ερώτημα και το dataset των σημείων. Χρησιμοποιεί shared memory για να αποθηκεύσει τις συντεταγμένες του κάθε σημείου και έπειτα για κάθε σημείο που αντιπροσωπεύεται από ένα thread καλεί την __device__ float my_euclidean_distance_gpu(float *v1, float* v2, int attributes,int

numObjects) η οποία υπολογίζει την ευκλείδια απόσταση των δοσμένων διανυσμάτων ανάλογα με τις διαστάσεις τους. Επιστρέφοντας στην calculate_distaces_seperately η απόσταση που υπολογίστηκε αποθηκεύεται στον πίνακα με τις αποστάσεις distances.

__global__ void select_kernel(float* distances,float* next_distances,int* next_indexes,int numObjects,int k):
Η συνάρτηση εκτελεί reduce των δεδομένων ανάλογα με το δοσμένο k. Χρησιμοποιεί την shared memory για την φόρτωση και διαχείρηση αποστάσεων αλλά και indexes τα οποία πρέπει να αντιστοιχίζονται με τις αποστάσεις.

Εφαρμόζεται η τεχνική Sequential Addressing όπως περιγράφεται στο pdf της εργασίας. Μετά την εφαρμογή του reduction το ελάχιστο εμφανίζεται στην 1η θέση και αποθηκεύεται. Η διαδικασία επαναλαμβάνεται k φορές.

__global__ void select_kernel_2(float* next_distances,int* next_indexes,float* next_distances_2,int* next_indexes_2,int numQueries,int k):
Η 2η συνάρτηση που εκτελεί το reduce στα εναπομείναντα δεδομένα με τον ίδιο ακριβώς τρόπο αποθηκεύοντας τα αποτελέσματα στους next_distances_2, next_indexes_2 πάνω στους οποίους θα εκτελεστεί το τελικό reduce.

__global__ void select_kernel_last(float* next_distances_2, int* next_indexes_2,float* NNdist,int* NNidx,int k, int q,int bs,float* last_distances,int* last_indexes):

Η τελική συνάρτηση reduce. Εκτελείται για ένα block μόνο και αποθηκεύει τα αποτελέσματα του reduce της , στους τελικούς πίνακες NN_dist, NN_idx.

void knns(knn_struct* d_training_set, knn_struct* d_query_set, float* d_NNdist, int* d_NNidx, int tk,float* distances):

Η εφαρμογή του αλγορίθμου knns στο πρόγραμμά μας. Οι πίνακες dataset και queries αντιπροσωπέυουν τα σημεία και τα ερωτήματα αντίστοιχα. Είναι υπεύθυνη για την χρονομέτρηση της εκτέλεσης στην κάρτα γραφικών. Αρχικά, δεσμεύει την απαραίτητη μνήμη για τους πίνακες των διαδοχικών reduce και για κάθε ερώτημα εκτελεί τον αλγόριθμο knns παράλληλα στην GPU.

Πρώτα καλείται η calculate_distances_seperately<<<grid , threads , numAttributes*sizeof(float)>>> για τον υπολογισμό του πίνακα όλων των αποστάσεων, distances. Εκτελείται σε ένα grid όπου κάθε στοιχείο του dataset αναπαριστάται απο ένα thread.

Στη συνέχεια καλείται η select_kernel<<< grid, threads, 2*BlockSize*sizeof(float) >>> για ίδιο grid, για το πρώτο reduction των αποστάσεων. Έπειτα, η select_kernel_2<<< newgrid, newthreads, 2*BlockSize*sizeof(float) >>> πραγματοποιεί το 2ο reduction των αποστάσεων σε ένα grid μεγέθους numObjects*k/512. Τέλος, η select_kernel_last<<< lastgrid, lastthreads, 2*BlockSize*sizeof(float) >>> εκτελεί το τελικό reduce των λίγων στοιχείων και τα αποθηκεύει στους πίνακες NN_dist, NN_idx.

Όπως, ξαναείπαμε η διαδικασία επαναλαμβάνεται για κάθε ερώτημα.

int main(int argc, char **argv):
Η κύρια συνάρτηση του προγράμματος μας. Αφού ελένξει ότι τα ορίσματα είναι σωστά δοσμένα από τον χρήστη, δεσμεύει την απαραίτητη μνήμη για την φόρτωση των στοιχείων. Αυτή πραγματοποιείται από το κατάλληλο αρχείο που έχει δημιουργηθεί με το script της MATLAB που προαναφέρθηκε. Στη συνέχει δεσμεύει την απαραίτητη μνήμη και στην GPU και αντιγράφει το dataset των στοιχείων και των ερωτημάτων ώστε να είναι στην διάθεση της για την εκτέλεση των υπολογισμών. Έπειτα, καλεί την knns και αφού ολοκληρωθεί ο αλγόριθμος αποθηκεύει τα αποτελέσματα στα τελικά αρχεία, και εμφανίζει και τον χρόνο εκτέλεσης του προγράμματος.

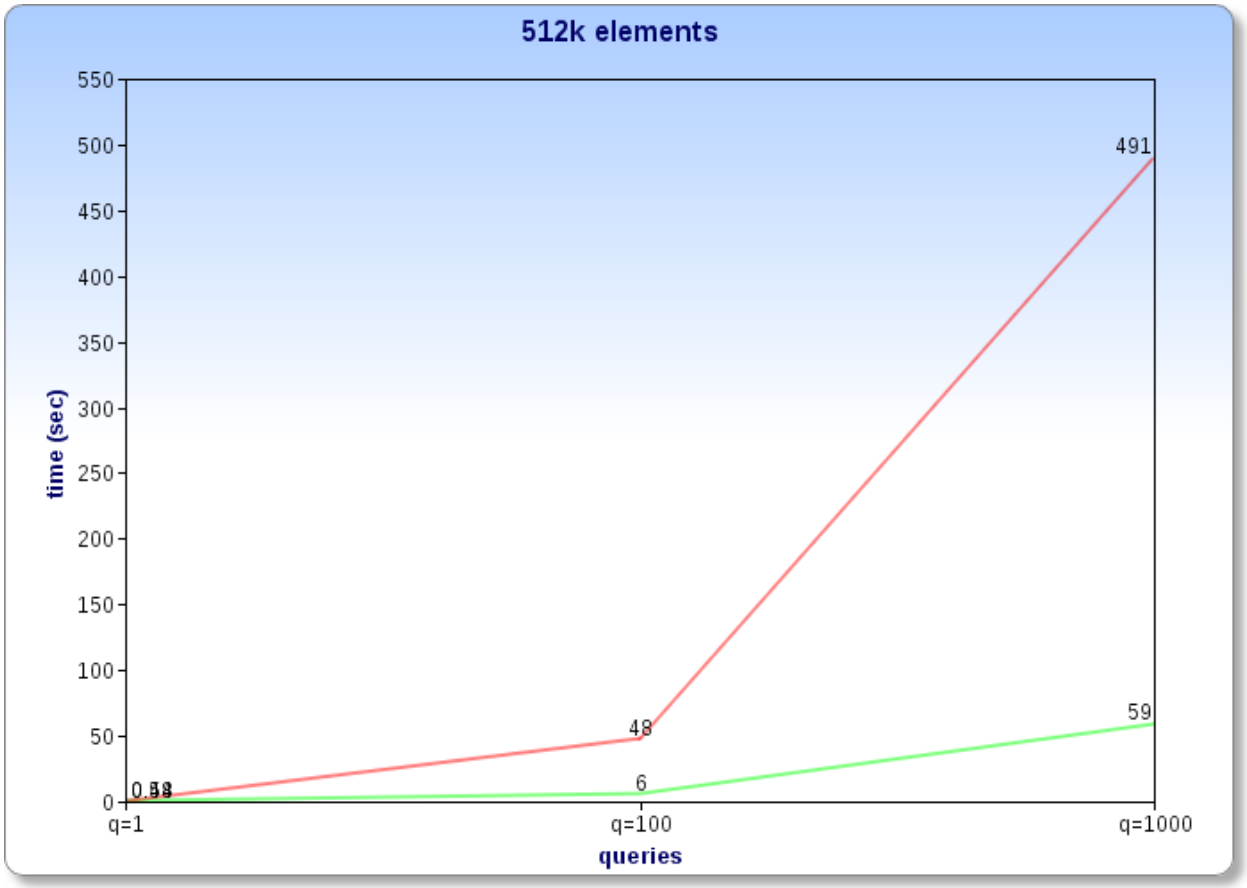
Έλεγχος ορθότητας

Δυστυχώς, τα τελικά αποτελέσματα δεν συμπίπτουν εξ’ ολοκλήρου με αυτά που υπολογίζει το σειριακό πρόγραμμα. Ο έλεγχος πραγματοποιήθηκε με ένα μικρό πρόγραμμα C που ανοίγει τα αρχεία και εκτυπώνει αποστάσεις και θέσεις, όπου φαίνεται ότι πολλές από αυτές δεν συμπίποτουν. Ακόμη, υπολογίστηκε το άθροισμα των αποστάσεων και των indexes και υπήρξε διαφορά ανάμεσα στο σειριακό και το παράλληλο. Γενικά, δεν χρησιμοποιήθηκε το MATLAB και το δοσμένο script για έλεχγο των αποτελεσμάτων αφού κάποιες μόνο αποστάσεις και θέσεις είναι ίδιες.

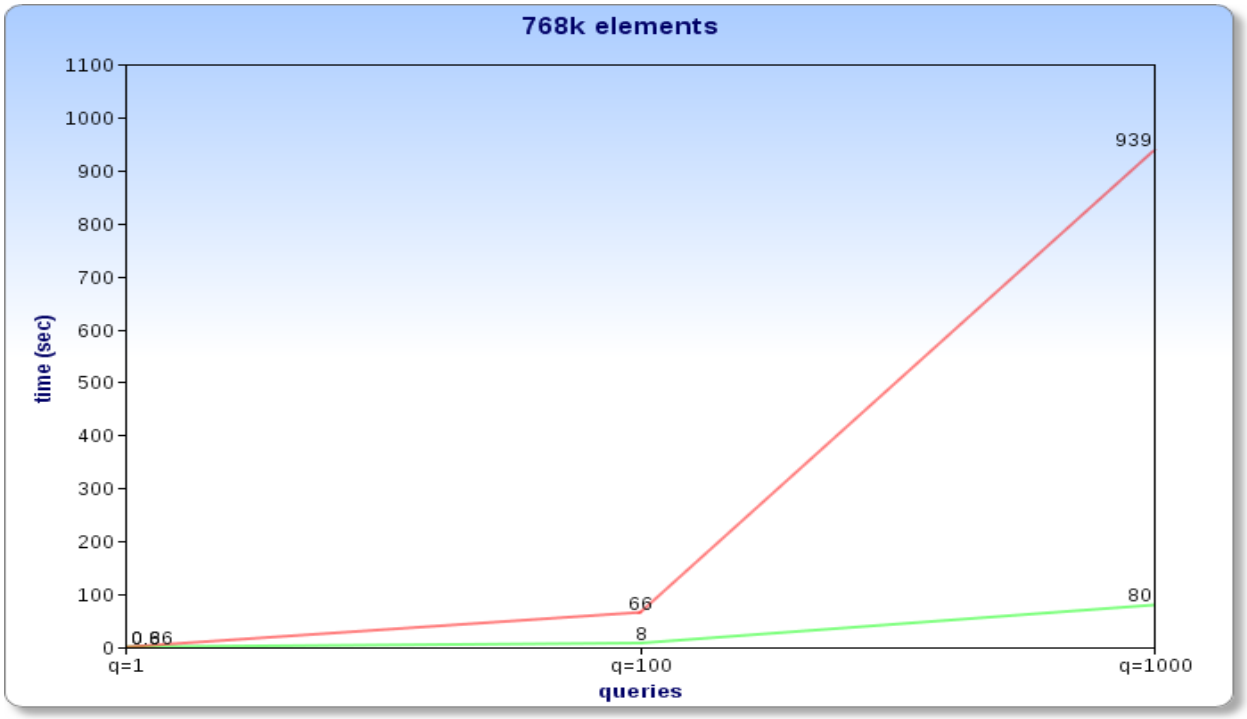
Χρόνος εκτέλεσης

Εδώ φαίνονται τα διαγράμματα με τους χρόνους εκτέλεσης σε σειριακό και k=1. Δεν εμφανίζονται για μεγαλύτερο k καθώς δεν αυξάνεται σημαντικά ο χρόνος εκτέλεσης και δεν θα είχε νόημα. Οι κόκκινες γραμμές αναπαριστούν τον χρόνο του σειριακού προγράμματος ενώ οι πράσινες του παράλληλου.

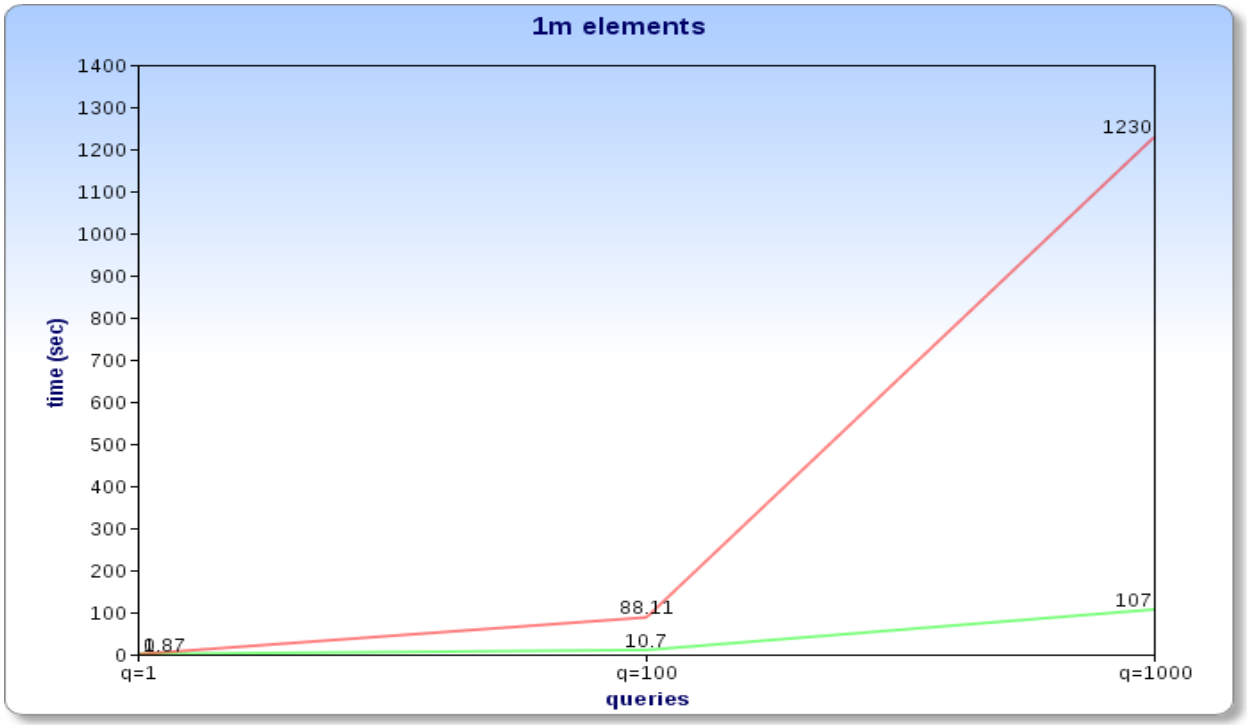
Για 512k στοιχεία:



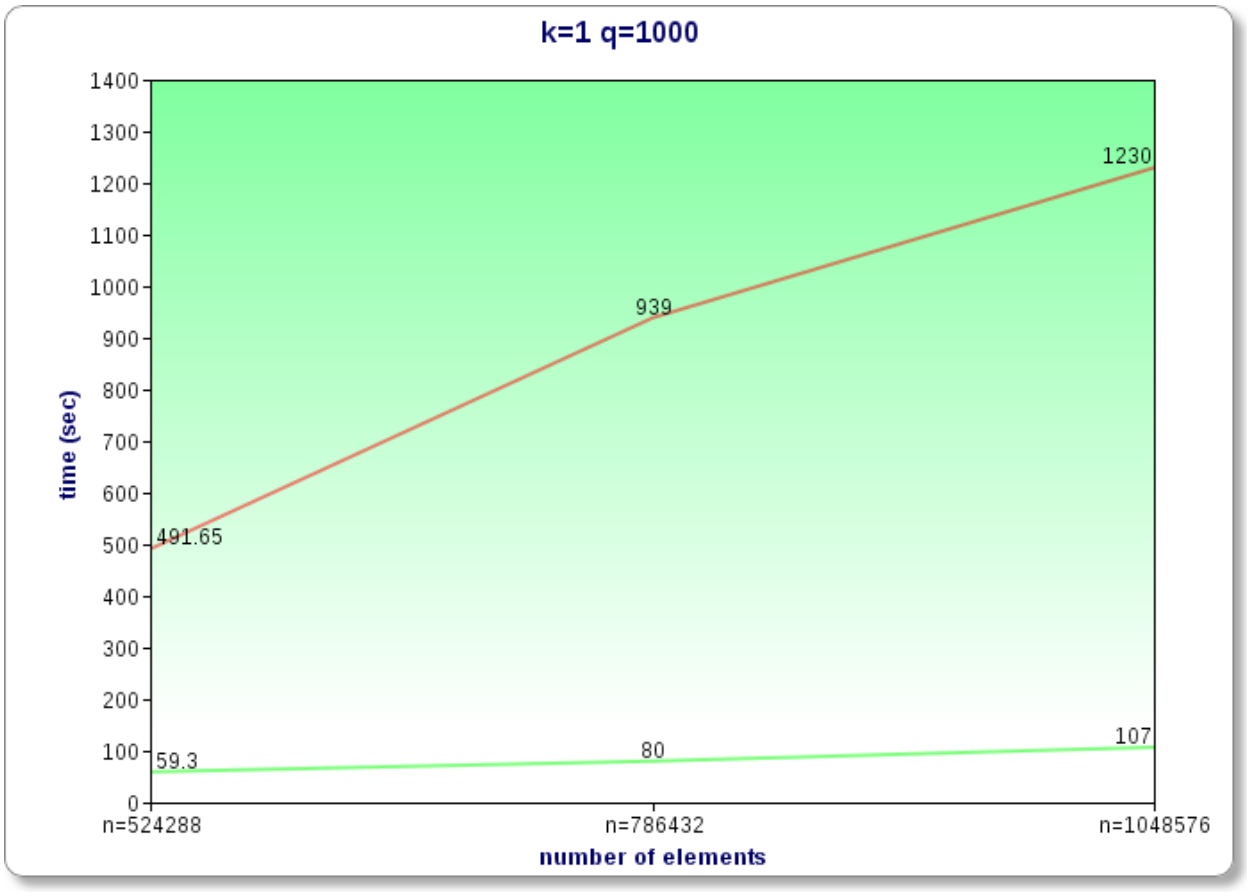
Για 768k στοιχεία:



Για 1m στοιχεία:



Και ένα συνολικό για k=1 και queries=1000:



Από τα παραπάνω διαγράμματα συμπεραίνουμε ότι η υλοποίηση σε CUDA επιταχύνει σημαντικά την εκτέλεση του αλγορίθμου knns. Όσο περισσότερα ερωτήματα ή όσο περισσότερα στοιχεία έχουμε να ικανοποιήσουμε τόσο περισσότερο αυξάνεται η απόδοση έναντι της σειριακής υλοποίησης, ενώ το πλήθος των γειτόνων k δεν φαίνεται να επηρεάζει σε σημαντικό βαθμό την ταχύτητα των υπολογισμών. Συγκεκριμένα, για 512k στοιχεία και q=1000 έχουμε τον κώδικα σε CUDA να τρέχει σχεδόν 8 φορές πιο γρήγορα από τον σειριακό. Ακόμα περισσότερο για 1 εκατ στοιχεία έχουμε σχεδόν 11 φορές πιο γρήγορο υπολογισμό!

Όλα αυτά θα είχαν μεγαλύτερη απόσταση εάν τα αποτελέσματα ήταν ακριβώς ίδια με το σειριακό αλλά δυστυχώς δεν μπορώ να εντοπίσω το λάθος που έχω κάνει.