Παράλληλα και Διανεμημένα Συστήματα

Όνομα : Μελεζιάδης Γιάννης
ΑΕΜ : 8760

ΆΣΚΗΣΗ 3

Στον σύνδεσμο υπάρχουν 5 αρχεία : 1) σειριακός κώδικας seir2.c + εκτελέσιμο
2)παράλληλος κώδικας(nonshared) nonshared.cu
3)παράλληλος κώδικας(shared) shared.cu
4)eisodos.txt (είσοδος από matlab)
5)eksodos.txt (έξοδος από matlab s=1, e=1)

link: https://drive.google.com/open?id=1iRlxmf-_Pz4liSHFn24zyeEfBjMPEnYQ

seir2.c ΣΕΙΡΙΑΚΟΣ ΚΩΔΙΚΑΣ

Συναρτήσεις

float distanceFunction(float *point1,float *point2) : Συνάρτηση που επιστρέφει την απόσταση δυο σημείων.

float kernelFunction(float *pointYk,float *arrayXi) :Συνάρτηση που επιστρέφει το βάρος σύμφωνα με τη συνάρτηση πυκνότητας Gaussian k(x).

float movedDistance(float *moved) : Συνάρτηση που επιστρέφει την απόσταση που έχει κινηθεί το σημείο που πραγματοποιεί shift.

void shiftFunction(float *Ykplus1,float* Yk,float *X,float e,int index) : Συνάρτηση που υλοποιεί τον αλγόριθμο meanshift . Με εισόδους τον πίνακα Yk(σημεία πριν), Ykplus1(σημεία μετα), X(σημεία στην αρχή), e(σταθερά σύγκλισης), index(ποιο σημείο θα κάνει το meanshift)

Κυρίως πρόγραμμα

Σημείωση: Χρησιμοποιήθηκαν μονοδιάστατοι πίνακες αντί για δυσδιάστατους καθώς η υλοποίηση με μονοδιάστατους πίνακες οδηγεί σε πιο εύκολο και γρήγορο αλγόριθμο με χρήση cuda. Έτσι οι πίνακες καθώς είναι μονοδιάστατοι που αντιπροσωπεύουν δυσδιάστατους έχουν μέγεθος arrays*columns και αντίστοιχα τα σημεία πάνε ανά δυο.

Αφού δημιουργηθούν δυναμικά οι μονοδιάστατοι πίνακες arrayStatic(αρχικά σημεία), arrayYk, arrayYkplus1 διαβάζονται τα δεδομένα από το αρχείο eisodos.txt και αποθηκεύονται στον arrayStatic αλλά και τον arrayYk καθώς στην αρχή του αλγορίθμου meanshift Yk=αρχικά σημεία. Στην συνέχεια μετατρέπω την παράμετρο εισόδου σε float με την atof( ), μετρώ τον χρόνο αρχής και ξεκινώ τον αλγόριθμο meanshift για κάθε σημείο του συνόλου :

```
for(i=0;i<points;i++){
        shiftFunction(arrayYkplus1,arrayYk,arrayStatic,e,i);
 }
```

Σημείωση: Για την κατανόηση της συνάρτησης shiftFunction( ) υπάρχουν σχόλια στον κώδικα. Τα σχόλια υπάρχουν μόνο στο αρχείο seir2.c και όχι στα shared.cu & nonshared.cu και αυτό γιατί ο κώδικας και η λογική είναι ίδια.

Αφού τελειώσει η συνάρτηση shiftFunction( ) για κάθε σημείο του συνόλου υπάρχει πλέον αποθηκευμένο στον arrayYk το τελικό αποτέλεσμα(μετά από πολλά shifts) για κάθε σημείο του αρχικού συνόλου. Υπολογίζω το πόσο κράτησε η όλη διαδικασία σε msecs και το τυπώνω στο τερματικό. Στη συνέχεια δημιουργώ αρχείο output.txt στο οποίο και τυπώνω τα αποτελέσματα(δηλαδή τον πίνακα Yk)(δεν χρειαζόταν αλλά ήταν χρήσιμο για έλεγχο ορθότητας).

Στο τέλος κάνω τον έλεγχο ορθότητας χρησιμοποιώντας το αρχειο eksodos.txt που πήρα από το matlab :

```
FILE* f2=fopen("eksodos.txt", "r");
        float myvar2;
        int errors=0;

                for(i=0;i<points;i++){
                for(j=0;j<dimensions;j++){

                        //den diavazei ton xaraktira meta to float eite ine , eite ine \n
                        scanfReturn=fscanf(f2,"%f%*c",&myvar2);
                        //elegxos an i scanf litourgise swsta
                        if(scanfReturn==1){/*ola kala*/}
                        else{ printf("error in %d,%d. \n",i,j); return 1;}

                        if(fabs(myvar2-arrayYk[i*dimensions+j])>0.5) errors++;
                }
        }

        printf("The number of errors is %d \n",errors);
```

Αν βρεθεί απόκλιση μεγαλύτερη απο 0.5 κατα απόλυτη τιμή τότε το μετράω σαν λάθος και στο τέλος τυπώνω το συνολικό αριθμό λαθών.

nonshared.cu ΠΑΡΑΛΛΗΛΟΣ ΚΩΔΙΚΑΣ ΧΩΡΙΣ ΧΡΗΣΗ shared memory

Συναρτήσεις

__global__ void shiftingFunction(float *Ykplus1, float *Yk, float *X, float e) : Συνάρτηση που έχει τον ίδιο ρόλο με την shiftFunction() του σειριακού με τη διαφορά ότι δεν υπάρχει index πλέον καθώς αυτό ορίζεται μέσω του cuda με την γραμμή
*int index = blockIdx.x *blockDim.x + threadIdx.x;*

Σημείωση: *΄Ολες οι υπόλοιπες συναρτήσεις αποκτούν ένα __device__ στον ορισμό τους καθώς ενεργούν πλέον σε δεδομένα της gpu.*

Κυρίως πρόγραμμα
Είναι ακριβώς ο ίδιος κώδικας με πρόσθετα τα στοιχεία που σχετίζονται με το cuda. Έτσι ορίζονται οι νέοι πίνακες deviceArrayStatic, deviceArrayYk, deviceArrayYkplus1 οι οποίοι είναι πίνακες της gpu και δεσμεύονται δυναμικά με την εντολή *cudaMalloc((void **)&deviceArrayStatic,nBytes);* .

Αφού διαβαστούν τα δεδομένα από το αρχείο eisodos.txt τότε χρησημοποιείται η cudaMemcpy για να περαστούν τα δεδομένα από τους πίνακες του host στους αντίστοιχους πίνακες του device. Σε αυτήν την περίπτωση ο αλγόριθμος meanshift ξεκινάει για κάθε σημείο του αρχικού συνόλου με την εντολή :

shiftingFunction<<< 6, 100 >>>(deviceArrayYkplus1,deviceArrayYk,deviceArrayStatic,e);
cudaDeviceSynchronize();

Η εντολή αυτή ξεκινάει 6 blocks με 100 threads το κάθε ένα για ευκολία καθώς υπάρχουν 600 σημεία. Απαραίτητη είναι η εντολή *cudaDeviceSynchronize()* κάθως μερικά νήματα μπορεί να έχουν τελειώσει νωρίτερα και για να είναι σωστή η υλοποίηση πρέπει να αρχίσει ο έλεγχος ορθότητας αφού έχουν τελειώσει όλα τα νήματα για την αποφυγή λαθών.Προτού γίνει ο έλεγχος ορθότητας είναι σημαντικό να περαστούν τα αποτελέσματα από τη gpu στο host με την εντολή *cudaMemcpy(arrayYk,deviceArrayYk,nBytes,cudaMemcpyDeviceToHost);*

Ακολουθεί πάλι η μέτρηση του χρόνου σε msecs, η δημιουργία του αρχείου output.txt με τα αποτελέσματα και ο έλεγχος ορθότητας με την καταμέτρηση των λαθών.

shared.cu ΠΑΡΑΛΛΗΛΟΣ ΚΩΔΙΚΑΣ ΜΕ ΧΡΗΣΗ shared memory

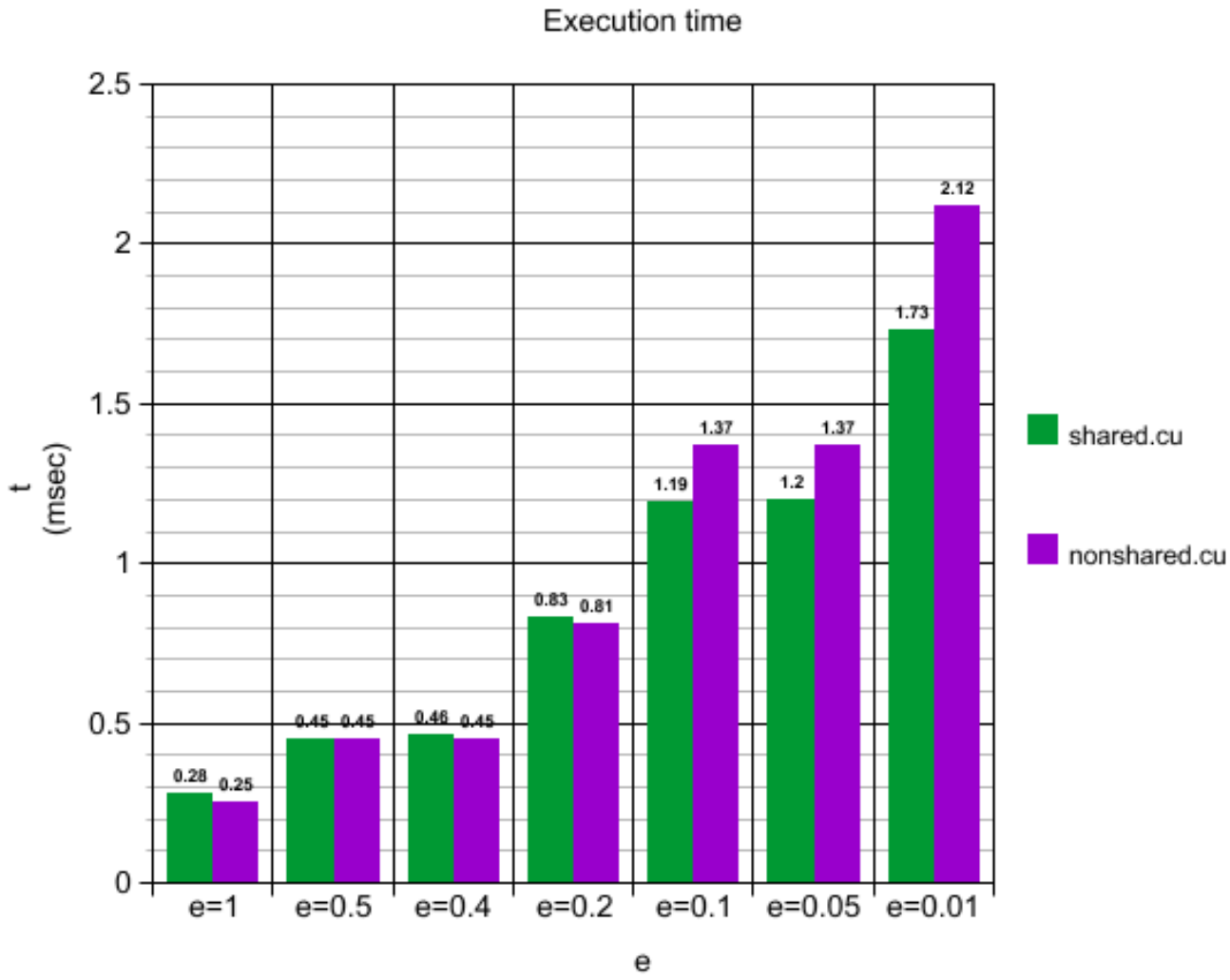
Είναι ο ίδιος ακριβώς κώδικας με το nonshared.cu με τη μόνη διαφορά στη shiftingFunction( ) όπου προστίθεται ο εξής κώδικας :

```
__shared__ float X[1200] ; //1200 giati points*dimensions=1200

if (index == 0) {
        for(i=0;i<600;i++){
                for(j=0;j<2;j++){
                        X[i*2+j]=arrayX[i*2+j];
                }
        }

}
__syncthreads();
```

Δηλώνω ως shared memory τον πίνακα X(αρχικό σύνολο σημείων) καθώς μόνο αυτός χρησιμοποιείται συνεχώς στον αλγόριθμο meanshift για κάθε σημείο. Επίσης περνάει σε αυτόν τις τιμές του ορίσματος arrayX **μόνο** το index==0 και μετά γίνεται και ένα __syncthreads() ώστε από εκεί και μετά να έχουν πρόσβαση στον πίνακα X όλα τα νήματα .

ΧΡΟΝΟΙ ΣΤΟ ΣΥΣΤΗΜΑ diades

Έγιναν μετρήσεις και με τα τρία προγράμματα στο σύστημα diades και παρατηρήθηκε ότι χρόνοι του σειριακού κώδικα ήταν πάρα πολύ μεγαλύτεροι από τους χρόνους των προγραμμάτων που έγινε χρήση cuda. Ακολουθεί πίνακας που δέιχνει τους χρόνους των shared.cu και nonshared.cu με αρχική παράμετρο e=1, 0.5, 0.4, 0.2, 0.1, 0.05, 0.01 .



Συμπεράσματα: Παρατηρείται ότι όταν το e είναι μεγάλο(δηλαδή συμβαίνουν λίγα shifts για κάθε σημείο) η υλοποίηση με χρήση shared memory δεν αξίζει διότι δεν υπάρχουν πολλές χρήσεις του πίνακα X. Έτσι ο χρόνος που σπαταλήθηκε για τη χρήση του shared memory είναι μεγαλύτερος από τα οφέλη. Όσο πιο μικρό γίνεται το κριτήριο σύγκλισης e τόσο πιο πολλά οφέλη σε χρόνο έχει ο αλγόριθμος όπως και φαίνεται από τα e=0.1, 0.05, 0.01 όπου ο χρόνος shared είναι κατά πολύ μικρότερος του χρόνου nonshared.

Οι χρόνοι του σειριακού κώδικα ήταν για e=1 → 14 msecs και για e=0.01→46 msecs οπότε φαίνεται ξεκάθαρα η επιτάχυνση που προσφέρει ο παραλληλισμός με cuda.