Παράλληλα και Διανεμημένα Συστήματα

Όνομα : Μελεζιάδης Γιάννης
ΑΕΜ: 8760

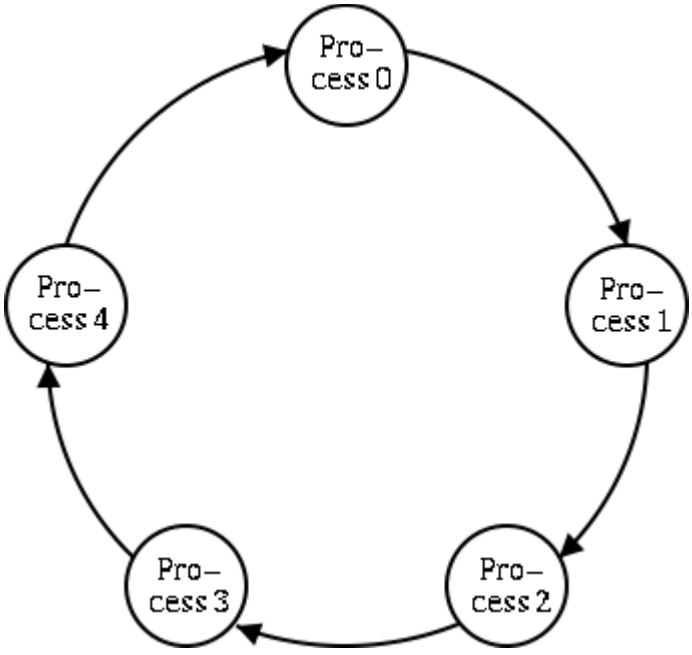
ΆΣΚΗΣΗ 2

Στον σύνδεσμο υπάρχουν 6 αρχεία : 1) σειριακός κώδικας seiriakos.c + εκτελέσιμο
2)παράλληλος κώδικας(blocking) mpi2.c + εκτελέσιμο
3)παράλληλος κώδικας(non blocking) mpinb.c +εκτελέσιμο
4)myFile.txt (πίνακας)
5)DIST10000x30.txt (αποστάσεις)
6)solutions.txt (kNN)

Όλα τα προγράμματα χρησιμοποιούν τον πίνακα 10000x784 από το matlab που υπάρχει στο αρχείο myFile.txt. Επίσης έγιναν έλεγχοι και με το αρχείο DIST10000x30.txt που έχει τις αποστάσεις αλλά και με το solutions.txt το οποίο έχει τη λύση με τα kNN σημεία για έλεγχο ορθότητας.

Εντολή για compile: mpicc mpi2.c -o mpi2 -O3 -lm -fopenmp
Ενολή για εκτέλεση: mpirun -np 4 mpi2

link: https://drive.google.com/open?id=11NHt5z-fwTopbuKMZm0raeUJrCCEjBBD



Mpi2.c ΜΕΘΟΔΟΣ ΠΑΡΑΛΛΗΛΙΣΜΟΥ blocking

Σημείωση: για χρήση openMP απλά βγαίνουν απο τα σχόλια οι γραμμές που βρίσκονται πάνω από τις διπλές for σε όλο το πρόγραμμα.

//#pragma omp parallel for private(j,dist,tempPoint,m)

έτσι ο χρόνος εκτέλεσης βελτιώνεται σημαντικά όπως φαίνεται και παρακάτω.

Συναρτήσεις

float getDistance(float *point1,float *point2) : συνάρτηση που υπολογίζει την απόσταση μεταξύ δυο σημείων με 784 συντεταγμένες.

void swap(float *var1,float *var2) : συνάρτηση για εναλλαγή float

void swap2(int *var1,int *var2) : συνάρτηση για εναλλαγή int

float **alloc_2d_float(int rows, int cols): συνάρτηση που δημιουργεί δυναμικά πίνακα με floats σε συνεχόμενες θέσεις μνήμης για να μπορεί να σταλθεί μέσω mpi

int **alloc_2d_int(int rows, int cols): δυναμικός πίνακας σε συνεχόμενες θέσεις μνήμης με ints

Κυρίως πρόγραμμα

Αφού δημιουργηθούν δυναμικά οι πίνακες block, kNNperBlock, pointPerkNN ακολουθεί το διάβασμα του αρχείου myFile.txt από κάθε διεργασία ώστε κάθε διεργασία να πάρει στον πίνακα block το κομμάτι του αρχείου που της αναλογεί.

Ακολουθεί ένα διπλό for του οποίου η δουλειά είναι να υπολογίζει αποστάσεις σημείο με σημείο και ταυτόχρονα να το τοποθετεί σε κατάλληλη θέση στον πίνακα κοντινότερων αποστάσεων kNNperBlock και ταυτόχρονα να βάζει και το σωστό σημείο στον πίνακα κοντινότερων σημείων PointPerkNN. Οι υπολογισμοί όλοι του αλγορίθμου kNN γίνονται σε παραλλαγές αυτού του διπλού for. Βλέπουμε ότι στην αρχή ο υπολογισμός της απόστασης γίνεται με block με block ενώ αργότερα θα γίνεται με το blockReceived που θα λαμβάνεται από άλλη διεργασία.

```
for(i=0;i<size;i++){
        for(j=0;j<size;j++){
                dist=getDistance(block[i],block[j]); //BLOCK ME BLOCK
                tempPoint=j+world_rank*size+1; //to world_rank mpainei gia na einai to swsto
                                                //Point
                if(dist!=0){
                        for(m=0;m<k;m++){
                                if(dist<kNNperBlock[i][m]){
                                        swap(&dist,&kNNperBlock[i][m]);
                                        swap2(&pointPerkNN[i][m],&tempPoint);
                                }
                        }
                }
        }
}
```

Ακολουθεί ένα ring topology μπλοκ κώδικα για την πρώτη ανταλλαγή μηνυμάτων. Κάθε διεργασία στέλνει το block της και δέχεται το block άλλων διεργασιών στο blockReceived της. Τα MPI_Send και MPI_Receive έχουν τοποθετηθεί με τέτοιο τρόπο ώστε να αποφεύγεται πάγωμα του προγράμματος όταν όλες οι διεργασίες περιμένουν και καμιά δεν στέλνει.

Στη συνέχεια είναι πάλι ένα διπλό for το οποίο έχει διαφορά με το προηγούμενο ότι ο υπολογισμος αποστάσεων έιναι με block και blockReceived γιατι δεν είναι η πρώτη φορά.

Ακολουθεί ένα for για τις υπόλοιπες world_size-2 ανταλλαγές μηνυμάτων καθώς τα 2 kNN έγιναν έξω απο το loop(θα μπορούσα να βάλω το δεύτερο μέσα). Μέσα σε αυτό πρώτα αντιγράφω το blockReceived στο blockToSend ώστε να σταλθεί το σωστό block. Μετά κάνω την ανταλλαγη μηνυμάτων με τον ίδιο τροπο. Ακολουθεί ένας αλγόριθμος ο οποίος βρίσκει από ποιά διεργασία ξεκίνησε αρχικά το block που πήρε η κάθε διεργασία και χρησιμοποιείται ώστε να τοποθετούνται τα σωστά σημεία στους πίνακες pointPerkNN . Τέλος ακολουθεί το γνωστό διπλό for το οποίο υλοποιεί όλη τη δουλεία .

Αφού τελείωσουν οι επικοινωνίες και κάθε διεργασία έχει παραλάβει και έχει κάνει kNN με κάθε αρχικό block που είχε παραλάβει κάθε άλλη διεργασία(δηλαδή με όλα τα σημεία του αρχικού πινακα). Ακολουθεί η εξακρίβωση αποτελεσμάτων(**έλεγχος ορθότητας**) με το αρχείο solutions.txt με δίαβασμα με τον ίδιο τρόπο απο το αρχέιο. Κάθε διεργασία δηλαδη εξακριβώνει τα δικά της αποτελέσματα για τα δικά της σημεία και μετράει τα σφάλματα τα οποία υπάρχουν.

Τέλος κάθε διεργασία στέλνει τον αριθμό των σφαλμάτων της στην διεργασία 0 όπου και αθροίζονται.

<u>Mpinb.c ΜΕΘΟΔΟΣ ΠΑΡΑΛΛΗΛΙΣΜΟΥ non-blocking</u>

Σημείωση: για χρήση openMP απλά βγαίνουν απο τα σχόλια οι γραμμές που βρίσκονται πάνω από τις διπλές for σε όλο το πρόγραμμα.

//#pragma omp parallel for private(j,dist,tempPoint,m)

έτσι ο χρόνος εκτέλεσης βελτιώνεται σημαντικά όπως φαίνεται και παρακάτω.

Λογική: η λογική για την μη φραγμένη επικοινωνία είναι ότι πρώτα στέλνω το block, μετά κάνω τους υπολογισμούς(διπλο for) και τέλος δέχομαι το επόμενο block(και wait), αντίθετα με την φραγμένη επικοινωνία όπου στέλνω-λαμβάνω και μετά κάνω υπολογισμούς.

Κυρίως πρόγραμμα

Οι συναρτήσεις παραμένουν οι ίδιες το μόνο που έχει αλλάξει είναι η θέση τους και ένας παραπάνω έλεγχος ο οποίος χρησιμοποιείται για να μην σταλθέι περισσότερες φορές το block από όσες επιτρέπεται καθώς στο τελικό loop η σειρά είναι receive-wait(το προηγουμενο send)-send

```
if(TIMES<world_size-3)
    MPI_Issend(&(blockToSend[0][0]),size*dimensions, MPI_FLOAT, 1, 23,
    MPI_COMM_WORLD,&req);
```

<u>seiriakos.c ΣΕΙΡΙΑΚΟΣ ΚΩΔΙΚΑΣ</u>

Η λογική είναι η ίδια με το non-blocking απλά σε μια διεργασία που παίρνει ολόκληρο το αρχείο.

<u>ΕΥΡΟΣ ΠΑΡΑΜΕΤΡΩΝ</u>

Το blocking αρχείο(mpi2.c) δουλεύει για ζυγό αριθμό διεργασιών που διαιρεί τέλεια τον αριθμό totalPoints = 10000 .

Το ίδιο και το nonblocking αρχείο(mpinb.c) εκτός του αριθμου 2.

ΧΡΌΝΟΙ ΣΤΟ hellasgrid
Χρησιμοποιώντας το παρακάτω script:

```
#!/bin/bash
#PBS -q pdlab
#PBS -N mpi-mm
#PBS -j oe
#PBS -l nodes=1:ppn=16

module load gcc/7.2.0
module load openmpi
module load mpi

export OMP_NUM_THREADS=$PBS_NP

cd $PBS_O_WORKDIR

mpirun -np $PBS_NP ./mpi2
```

παρατήρησα τους εξής χρόνους:

Σειριακός χρόνος **300,3216 sec**
1. blocking,without openMP----> **24.5916 sec**
2. blocking,with openMP –------> **10.2734 sec**
3. non-blocking,without openMP---------->**22.4311 sec**
4.non-blocking,with openMP---------->**9.4451 sec**

Συμπεράσματα: η χρήση παραλληλισμού με MPI επιτάχυνε πολύ τον αλγόριθμο kNN σε σχέση με το σειριακό και αν γινόταν και συνδυασμένη χρήση μαζί με openMP τότε ο χρόνος γινόταν ακόμα πιο μικρός (συγκεκριμένα τουλάχιστον υποδιπλάσιος σε σχέση με μόνο MPI).