ECE445: ΠΑΡΑΛΛΗΛΟΣ ΚΑΙ ΔΙΚΤΥΑΚΟΣ ΥΠΟΛΟΓΙΣΜΟΣ

Χειμερινό Εξάμηνο 2021-2022

Εργασία 3

Ομάδα φοιτητών #10

Ψαρού Αναστασία - 2860

Λυμπερόπουλος Νικόλαος 2812

**Εισαγωγή**

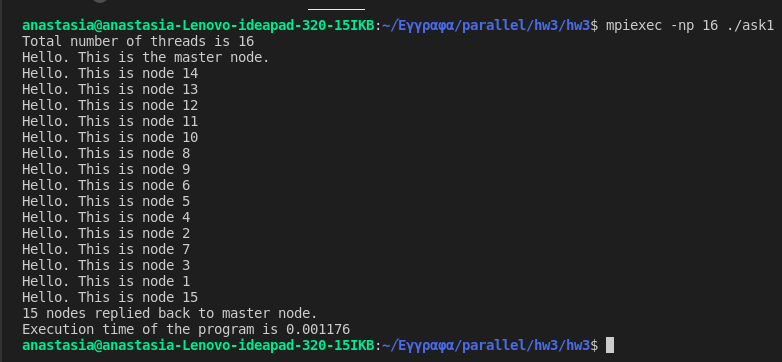
To software και το hardware που χρησιμοποιήσαμε για την υλοποίηση και των τριών ασκήσεων

παρουσιάζεται παρακάτω:

1. Operating System: Ubuntu 20.04.3 LTS
2. Compiler: gcc 4:9.3.0-1ubuntu2
3. Processor: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
4. Ram: 7737MB

## Ερώτηση 1

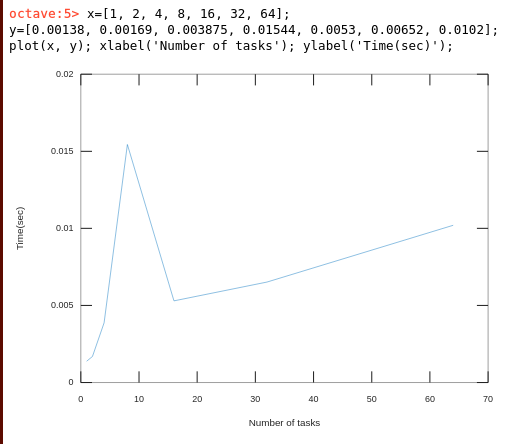
Σκοπός αυτής της άσκησης είναι η επικοινωνία μεταξύ του master thread και των υπολοίπων νημάτων που υπάρχουν στο πρόγραμμα. Αρχικά, δημιουργούμε n νήματα, κάθε ένα από τα οποία έχει ένα id. Για το master thread το id είναι ίσο με 0. Το master thread, λοιπόν, στέλνει ένα μήνυμα σε όλα τα νήματα και αυτό επιτυγχάνεται μέσω των συναρτήσεων MPI\_Send και MPI\_Recv. Στη συνέχεια, το κάθε ένα από τα υπόλοιπα νήματα στέλνει και αυτό με τη σειρά του ένα μήνυμα στο master thread. Τέλος, αφού επιβεβαιώσουμε ότι όλα τα νήματα απάντησαν στο master thread υπολογίζουμε το χρόνο εκτέλεσης του προγράμματος. Αυτό γίνεται με τη χρήση της συνάρτησης MPI\_Wtime. Για τον υπολογισμό του χρόνο υπολογίζεται ο μέσος χρόνος 10 εκτελέσεων του κώδικα με τον αντίστοιχο αριθμό νημάτων. Ακολουθεί ένα παραδειγμα εκτέλεσης του προγράμματος για 16 νήματα.



Στη συνέχεια υπολογίζουμε τους χρόνους εκτέλεσης του προγράμματος για διαφορετικό αριθμό νημάτων:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of tasks | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| Time (sec) | 0.00138 | 0.00169 | 0.003875 | 0.01544 | 0.0053 | 0.00652 | 0.0102 |

Τα αποτελέσματα μας σε γραφική παράσταση:

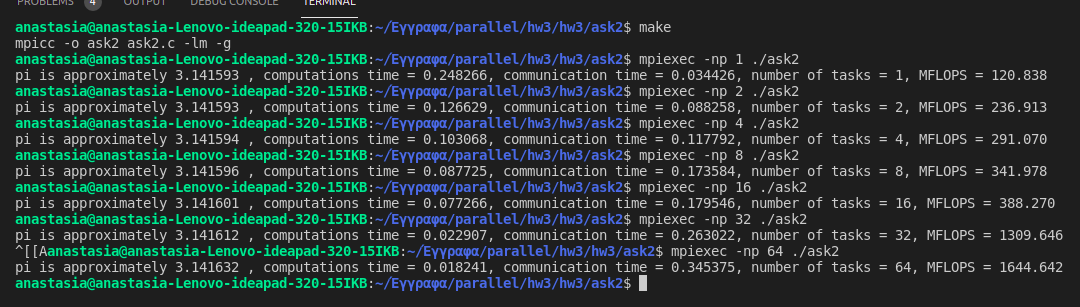


## Ερώτηση 2

Σκοπός αυτής της άσκησης είναι ο υπολογισμός μιας προσέγγισης του π, χρησιμοποιώντας τον κανόνα του Simpson.

Αρχικά, αρχικοποιούμε το περιβάλλον που θα δουλέψουμε και δεσμεύουμε την κατάλληλη μνήμη για τους πίνακες που θα χρησιμοποιήσουμε. Το νήμα 0 αρχικοποιεί τον πίνακα x, ο οποίος έχει μέγεθος Ν+1 και κάθε θέση του περιέχει ένα xi. Αφού γίνει αυτό, ανάλογα με τον αριθμό των νημάτων και το μέγεθος του Ν μοιράζονται οι υπολογισμοί στα εκάστοτε νήματα. Αυτό γίνεται με τη χρήση της συνάρτησης MPI\_Scatter. Στη συνέχεια, το κάθε νήμα κάνει τους υπολογισμούς του και αποθηκεύει τα αποτελέσματα του στην εκάστοτε θέση του πίνακα y\_procs, του οποίου κάθε θέση περιέχει το αντίστοιχο yi. Το κάθε νήμα, λοιπόν, βρίσκει το ολικό άθροισμα των yi του και αφού ολοκληρωθούν οι υπολογισμοί όλων των νημάτων υπολογίζεται το άθροισμα όλων των yi με χρήση της συνάρτησης MPI\_Reduce και flag MPI\_SUM. Η αντίστοιχη διαδικασία γίνεται για τον υπολογισμό του αριθμού των FLOPS. Όσον αφορά τους χρόνος communication\_time, computation\_time με τη χρήση της συνάρτησης MPI\_Reduce και flag MPI\_MAX παίρνουμε το μέγιστο χρόνο από όλα τα νήματα.

Ακολουθεί ένα παράδειγμα εκτέλεσης του προγράμματος μας:

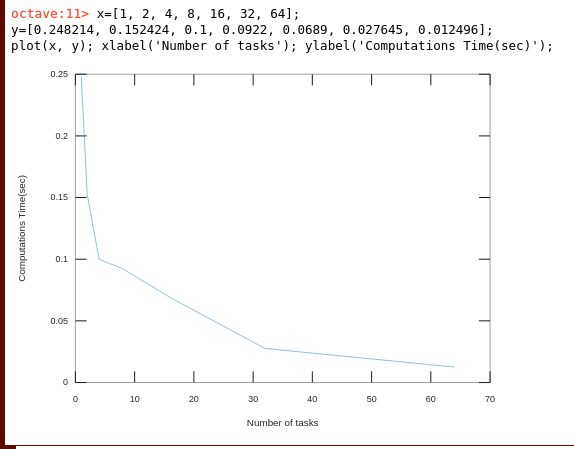


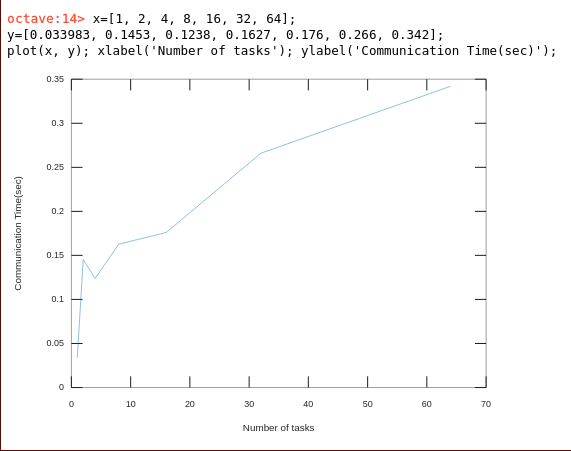
Στη συνέχεια ακολουθεί ένας πίνακας που αναπαριστά τα αποτελέσματα μας για Ν+1 = 108:

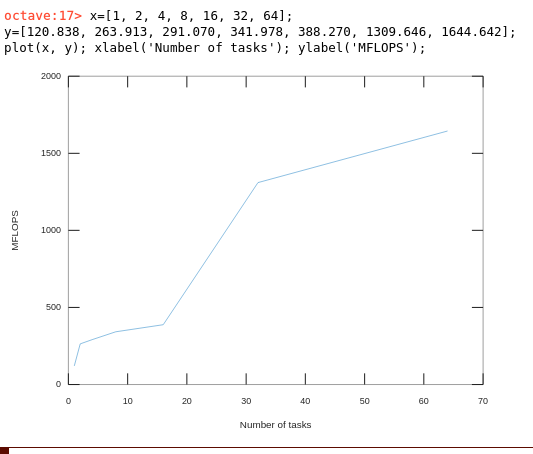
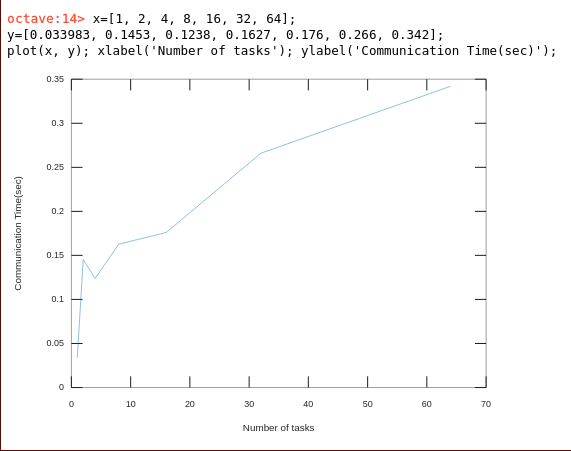
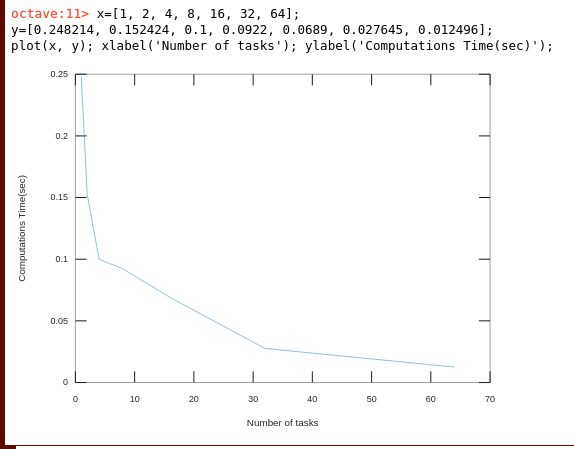
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of tasks | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| Computations Time (sec) | 0.248214 | 0.152424 | 0.1 | 0.0922 | 0.0689 | 0.027645 | 0.012496 |
| Communication Time (sec) | 0.033983 | 0.1453 | 0.1238 | 0.1627 | 0.176 | 0.266 | 0.342 |
| MFLOPS | 120.838 | 263.913 | 291.070 | 341.978 | 388.270 | 1309.646 | 1644.642 |

Παρατηρούμε ότι όσο αυξάνεται ο αριθμός των νημάτων αυξάνεται ο χρόνος επικοινωνίας. Αυτό συμβαίνει, διότι όσο περισσότερα νήματα τόσος περισσότερος χρόνος απαιτείται για να επικοινωνήσουν μεταξύ τους. Από την άλλη, παρατηρούμε ότι μειώνεται ο χρόνος των υπολογισμών. Αυτό συμβαίνει, επειδή όσο περισσότερα τα νήματα τόσο λιγότερη δουλειά θα έχει το κάθε νήμα και άρα θα χρειάζεται λιγότερο χρόνο για να την εκτελέση. Όσον αφορά το MFLOPS για τον υπολογισμό του έγινε χρήση του χρονου υπολογισμών. Είναι λογικό που αυξάνεται όσο αυξάνεται ο αριθμός των νημάτων, καθώς ο αριθμός των πράξεων κινητής υποδιαστολής είναι ανεξάρτητος από το χρόνο και παράλληλα παρατηρούμε ότι μειώνεται ο χρόνος υπολογισμών.

Τα αποτελέσματα μας σε γραφική παράσταση:







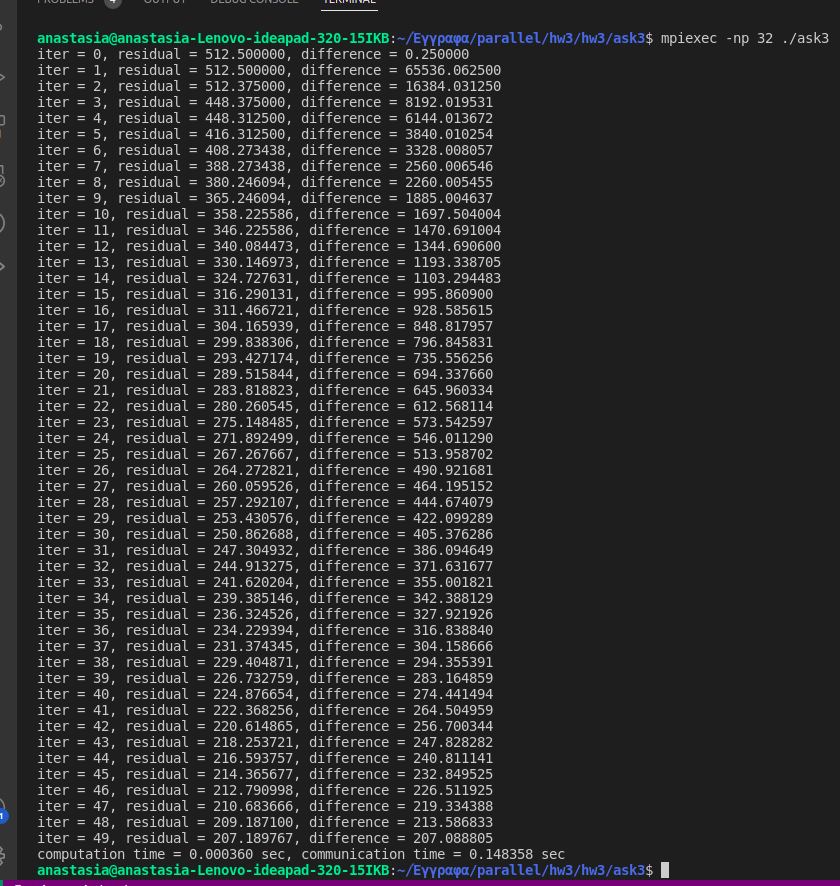
## Ερώτηση 3

Στόχος αυτής της εργασίας είναι να λύσουμε το σύστημα Ax = b με τη χρήση της μεθόδου του Jacobi.

Αρχικά, δημιουργούμε το περιβάλλον που θα δουλέψουμε και δεσμεύουμε μνήμη για τους πίνακες που θα χρησιμοποιήσουμε. Αυτό που κάνουμε, στη συνέχεια, είναι ότι χωρίζουμε σε κομμάτια τα διανύσματα b, x\_new, x\_old και κάθε ένα από αυτά τα κομμάτια μοιράζεται στα επιμέρους νήματα. Αυτό επιτυγχάνεται με τη βοήθεια των συναρτήσεων MPI\_Scatter, MPI\_Scatterv. Παράλληλα, υπολογίζονται και τα residual, difference για το κάθε νήμα και με τη χρήση της συνάρτησης MPI\_Reduce και όρισμα MPI\_Sum βρίσκόνται τα αθροίσματα των residual, difference για όλα τα νήματα. Τέλος, με χρήση της συνάρτησης MPI\_Gather συλλέγουμε το x\_new και θέτουμε το x\_new στο x\_old.

Για τον υπολογισμό των χρόνων βρίσκουμε το χρόνο κάθε νήματος ανά τμήμα και μετά από αυτούς τους χρόνους κρατάμε τον μεγαλύτερο με τη χρήση της συνάρτησης MPI\_Reduce και όρισμα MPI\_Max. Ανάλογα με το αν βρισκόμαστε σε φάση υπολογισμών ή επικοινωνίας μεταξύ των νημάτων το συγκεκριμένο αποτέλεσμα προστίθεται στην αντίστοιχη κατηγορία.

Ακολουθεί ένα παράδειγμα εκτέλεσης του προγράμματος μας για 32 νήματα:

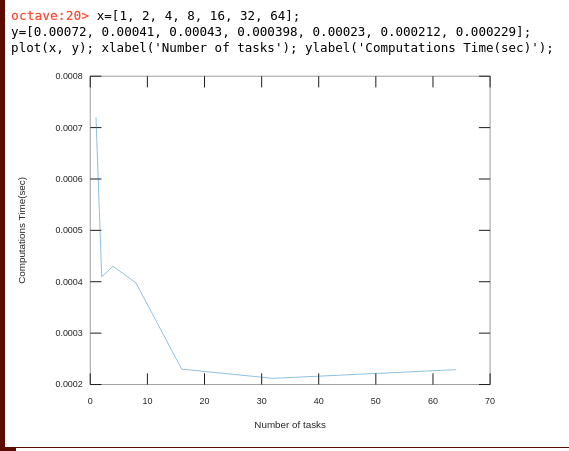
Παρατηρούμε ότι όσο αυξάνονται οι επαναλήψεις τα residual, difference μειώνονται. Αυτό συμβαίνει, διότι με την αύξηση των επαναλήψεων προσεγγίζουμε όλο και πιο πολύ το τελικό αποτέλεσμα και άρα οι διαφορές των τιμών x\_new, x\_old ανάμεσα στις επαναλήψεις μειώνονται.

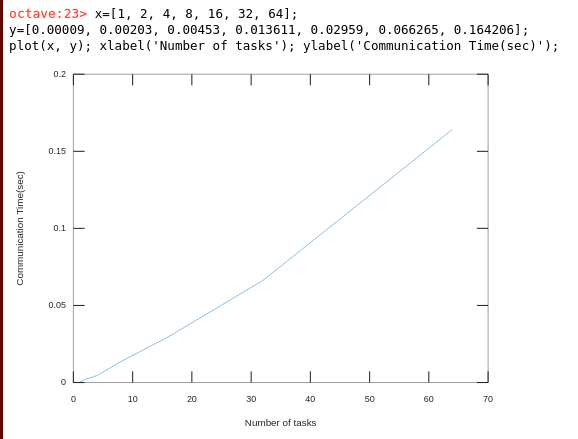
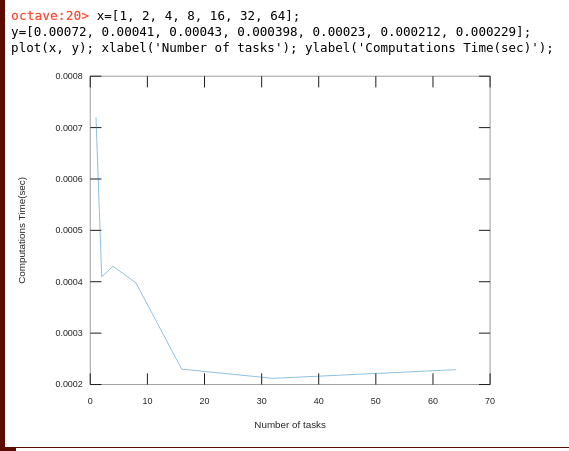
Στη συνέχεια, αναπαριστούμε τα αποτελέσματα μας σε πίνακα. Οι συγκεκριμένες μετρήσεις έχουν γίνει για Μ = 50 και Ν = 1024(2^10).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of tasks | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| Computations Time(sec) | 0.00072 | 0.00041 | 0.00043 | 0.000398 | 0.00023 | 0.000212 | 0.000229 |
| Communication Time(sec) | 0.00009 | 0.00203 | 0.00453 | 0.013611 | 0.02959 | 0.066265 | 0.164206 |

Παρατηρούμε ότι ο χρόνος επικοινωνίας μεταξύ των νημάτων αυξάνεται όσο περισσότερα νήματα έχουμε που είναι λογικό. Επίσης, ο χρόνος υπολογισμών μειώνεται όσο μεγαλύτερος είναι ο αριθμός των νημάτων. Αυτό εξηγείται από το ότι όσο μεγαλύτερος ο αριθμός των νημάτων τόσο λιγότερη δουλειά κάνει το κάθε νήμα άρα και λιγότερος ο χρόνος των υπολογισμών.

Ακολουθούν τα γραφήματα μας:





# ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Γραμματή Πάντζιου, Βασίλειος Μάμαλης, Αλέξανδρος Τομαράς, «Εισαγωγή στον Παράλληλο Υπολογισμό».
2. wikipedia.org

# ΠΑΡΑΡΤΗΜΑ

## Κώδικας Ερώτησης 1

Ακολουθεί ο κώδικας για το ερώτημα 1. Όνομα Αρχείου «ask1.c»

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ergasia 3 – Askhsh 1 – February 5, 2022

\* Psarou Anastasia - 2860

\* Lymperopoulos Nikos - 2812

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Anastasia Psarou

//Nikos Lymperopoulos

#include <mpi.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define SIZE 256

int main( int argc, char \*argv[] ) {

int numprocs, myid, times = 0;

int source, rc, count, c;

double end\_time, start\_time;

int i, dest, tag = 1;

char\* num;

MPI\_Status Stat;

char buf[SIZE];

char inmsg[SIZE];

char send\_proc\_buffer[SIZE];

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

if(myid == 0){

start\_time = MPI\_Wtime();

printf("Total number of threads is %d\n", numprocs);

printf("Hello. This is the master node.\n");

strcpy(buf, "Hello. This is the master node.");

for(i = 1; i < numprocs; i++){

rc = MPI\_Send(buf, SIZE, MPI\_BYTE, i, 0, MPI\_COMM\_WORLD);

}

while(1){

if(times == (numprocs - 1)){

break;

}

rc = MPI\_Recv(&inmsg, SIZE, MPI\_BYTE, MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD, &Stat);

times++;

}

printf("%d nodes replied back to master node.\n", times);

}

else{

rc = MPI\_Recv(buf, SIZE, MPI\_BYTE, 0, 0, MPI\_COMM\_WORLD, &Stat);

printf("Hello. This is node %d\n", myid);

sprintf(send\_proc\_buffer, "Hello. This is node %d.", myid);

rc = MPI\_Send(&send\_proc\_buffer, SIZE, MPI\_BYTE, 0, 1, MPI\_COMM\_WORLD);

}

if(myid == 0){

end\_time = MPI\_Wtime();

printf("Execution time of the program is %lf\n", end\_time - start\_time);

}

c = MPI\_Get\_count(&Stat, MPI\_CHAR, &count);

MPI\_Finalize();

return(0);

}

## Κώδικας Ερώτησης 2

Ακολουθεί ο κώδικας για το ερώτημα 2. Όνομα Αρχείου «ask2.c»

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ergasia 3 – Askhsh 2 – February 5, 2022

\* Psarou Anastasia - 2860

\* Lymperopoulos Nikos - 2812

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

//Anastasia Psarou

//Nikos Lymperopoulos

#include <mpi.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define N 9999999

int main( int argc, char \*argv[] ) {

int numprocs, myid, i;

double dx = 1 / (double)N;

double num\_flops = 0;

double sum = 0;

double final\_sum;

double \*x;

double \*x\_procs;

double \*y\_procs;

double communication\_time = 0;

double computation\_time = 0;

double start\_time\_com;

double start\_time\_comp;

double final\_time, fin, final, final\_communication;

double pi, total\_time;

double number\_flops;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

int thread\_work = (N + 1)/numprocs;

x\_procs = (double \*)malloc((thread\_work) \* sizeof(double));

y\_procs = (double \*)malloc((thread\_work) \* sizeof(double));

x = (double \*)malloc((N+1) \* sizeof(double));

if(myid == 0){

for(i = 0; i <= N; i++){

x[i] = i \* dx;

}

}

//start counting communication time

start\_time\_com = MPI\_Wtime();

MPI\_Scatter(x, thread\_work, MPI\_DOUBLE, x\_procs, thread\_work, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

final = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&final, &communication\_time, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

//stop counting communication time and start counting computation time

start\_time\_comp = MPI\_Wtime();

if(myid == 0){

y\_procs[0] = 4/(1+ pow(x\_procs[0], 2));

sum += y\_procs[0];

for(i = 1; i < thread\_work; i++){

if(i % 2 == 0){

y\_procs[i] = 4 \* 4 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

else{

y\_procs[i] = 4 \* 2 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

}

}

else if (myid < numprocs - 1){

y\_procs[thread\_work - 1] = 4/(1 + pow(x\_procs[thread\_work - 1], 2));

sum += y\_procs[thread\_work - 1];

for(i = 0; i <= thread\_work; i++){

if((i + myid \* thread\_work) % 2 == 0){

y\_procs[i] = 4 \* 4 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

else{

y\_procs[i] = 4 \* 2 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

}

}

else{

for(i = 0; i < thread\_work; i++){

if((i + myid \* thread\_work) % 2 == 0){

y\_procs[i] = 4 \* 4 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

else{

y\_procs[i] = 4 \* 2 / (1 + pow(x\_procs[i], 2));

num\_flops += 3;

sum += y\_procs[i];

continue;

}

}

}

//stop counting computation time

final\_time = MPI\_Wtime() - start\_time\_comp;

MPI\_Reduce(&final\_time, &computation\_time, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

//start counting communication time

start\_time\_com = MPI\_Wtime();

MPI\_Reduce(&sum, &final\_sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

MPI\_Reduce(&num\_flops, &number\_flops, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

fin = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&fin, &final\_communication, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

communication\_time += final\_communication;

if(myid == 0){

pi = final\_sum \* dx / 3;

printf("pi is approximately %lf , computations time = %lf, communication time = %lf, number of tasks = %d, MFLOPS = %4.3lf\n", pi, computation\_time, communication\_time, numprocs, number\_flops/computation\_time/1000000);

}

MPI\_Finalize();

return(0);

}

## Κώδικας Ερώτησης 3

Ακολουθεί ο κώδικας για το ερώτημα 3. Όνομα Αρχείου «ask3.c»

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ergasia 3 – Askhsh 3 – February 5, 2022

\* Psarou Anastasia - 2860

\* Lymperopoulos Nikos - 2812

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Anastasia Psarou

//Nikos Lymperopoulos

#include <mpi.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define N 1024

#define M 50

int main(int argc, char \*argv[]){

int numprocs, myid;

double \*\*A;

double \*b;

double \*x\_old;

double \*x\_new;

double \*b\_procs, \*x\_old\_procs, \*x\_new\_procs;

int i, m;

double residual, difference, total\_residual, total\_difference;

int \*sendcounts, \*displs;

double start\_time\_com;

double start\_time\_comp;

double final\_time, fin, final, final\_communication, final\_computation;

double communication\_time = 0;

double computation\_time = 0;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

int thread\_work = N / numprocs;

x\_new\_procs = (double \*)malloc((thread\_work+2) \* sizeof(double));

b\_procs = (double \*)malloc(thread\_work \* sizeof(double));

x\_old\_procs = (double \*)malloc((thread\_work+2) \* sizeof(double));

if(myid == 0){

x\_old = (double \*)malloc(N \* sizeof(double));

x\_new = (double \*)malloc(N \* sizeof(double));

b = (double \*)malloc(N \* sizeof(double));

sendcounts = (int \*)malloc(numprocs \* sizeof(int));

displs = (int \*)malloc(numprocs \* sizeof(int));

//arrays initialization

for (i = 0; i < N; i++){

if(i < (thread\_work + 2)){

x\_old\_procs[i] = 0;

x\_new\_procs[i] = 0;

}

if(i < thread\_work){

b\_procs[i] = 0;

}

b[i] = 0;

x\_old[i] = 1;

x\_new[i] = 0;

}

b[N - 1] =(double)N + 1;

displs[0]=0;

if(numprocs == 1){

sendcounts[0]=N;

}

else if(numprocs == 2){

sendcounts[0]=N/2+1;

sendcounts[1]=N/2+1;

displs[1]=N/2-1;

}

else{

sendcounts[0]=thread\_work+1;

sendcounts[numprocs - 1] = thread\_work + 1;

displs[numprocs - 1] = N - thread\_work - 1;

for(i = 1; i <= numprocs - 2; i++){

sendcounts[i] = thread\_work + 2;

displs[i] = i \* thread\_work - 1;

}

}

}

//start counting communication time

start\_time\_com = MPI\_Wtime();

MPI\_Scatter(b, thread\_work, MPI\_DOUBLE, b\_procs, thread\_work, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

final = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&final, &communication\_time, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

for(m = 0; m < M; m++){

//start counting communication time

start\_time\_com = MPI\_Wtime();

MPI\_Scatterv(x\_old, sendcounts, displs, MPI\_DOUBLE, x\_old\_procs, thread\_work + 2, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

fin = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&fin, &final\_communication, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

communication\_time += final\_communication;

//stop counting communication time and start counting computation time

start\_time\_comp = MPI\_Wtime();

residual = 0;

difference = 0;

//find x\_new

if(myid == 0){

x\_new\_procs[0] = 0.5 \* (x\_old\_procs[1] + b\_procs[0]);

for(i = 1; i <= thread\_work - 1; i++){

x\_new\_procs[i] = 0.5 \* (x\_old\_procs[i - 1] + x\_old\_procs[i + 1] + b\_procs[i]);

}

}

else if(myid < numprocs - 1){

for(i = 0; i < thread\_work; i++){

x\_new\_procs[i] = (0.5) \* (x\_old\_procs[i] + x\_old\_procs[i+2] + b\_procs[i]);

}

}

else{

x\_new\_procs[thread\_work - 1] = (0.5)\*(x\_old\_procs[thread\_work-2] + b\_procs[thread\_work-1]);

for(i = 0; i <= thread\_work - 2; i++){

x\_new\_procs[i] = (0.5) \* (x\_old\_procs[i] + x\_old\_procs[i+2] + b\_procs[i]);

}

}

//start counting communication time and stop counting computation

final\_time = MPI\_Wtime() - start\_time\_comp;

MPI\_Reduce(&final\_time, &final\_computation, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

computation\_time += final\_computation;

start\_time\_com = MPI\_Wtime();

//collect x\_new

MPI\_Gather(x\_new\_procs, thread\_work, MPI\_DOUBLE, x\_new, thread\_work, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Scatterv(x\_new, sendcounts, displs, MPI\_DOUBLE, x\_new\_procs, thread\_work+2, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

fin = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&fin, &final\_communication, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

communication\_time += final\_communication;

//stop counting communication time and start counting computation time

start\_time\_comp = MPI\_Wtime();

//find difference

for(int k = 0; k < thread\_work; k++){

difference += pow(x\_old\_procs[k] - x\_new\_procs[k], 2);

}

//find residual

if(myid == 0){

residual += fabs(2 \* x\_new\_procs[0] - x\_new\_procs[1] - b\_procs[0]);

for(i = 1; i <= thread\_work - 1; i++){

residual += fabs(2 \* x\_new\_procs[i] - x\_new\_procs[i + 1] - x\_new\_procs[i +-1] - b\_procs[0]);

}

}

else if(myid < numprocs-1){

for(i = 0; i < thread\_work; i++)

residual += fabs(-x\_new\_procs[i] + 2 \*x\_new\_procs[i + 1] - x\_new\_procs[i + 2] - b\_procs[i]);

}

else{

residual += fabs(-x\_new\_procs[thread\_work - 1] + 2 \* x\_new\_procs[thread\_work] - b\_procs[thread\_work - 1]);

for(i = 0; i < thread\_work - 1; i++)

residual += fabs(-x\_new\_procs[i] + 2 \* x\_new\_procs[i + 1] - x\_new\_procs[i + 2] -b\_procs[i]);

}

final\_time = MPI\_Wtime() - start\_time\_comp;

MPI\_Reduce(&final\_time, &final\_computation, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

computation\_time += final\_computation;

start\_time\_com = MPI\_Wtime();

MPI\_Scatter(x\_old, thread\_work, MPI\_DOUBLE, x\_old\_procs, thread\_work, MPI\_DOUBLE, 0,MPI\_COMM\_WORLD);

MPI\_Scatter(x\_new, thread\_work, MPI\_DOUBLE, x\_new\_procs, thread\_work, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

//collect x\_old from x

MPI\_Gather(x\_new\_procs, thread\_work, MPI\_DOUBLE, x\_old, thread\_work, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

//find total difference and total residual

MPI\_Reduce(&residual, &total\_residual, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

MPI\_Reduce(&difference, &total\_difference, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

//stop counting time for communication

fin = MPI\_Wtime() - start\_time\_com;

MPI\_Reduce(&fin, &final\_communication, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

communication\_time += final\_communication;

if(myid == 0){

printf("iter = %d, residual = %lf, difference = %lf\n", m, total\_residual, total\_difference);

}

}

if(myid == 0){

printf("computation time = %lf sec, communication time = %lf sec\n", computation\_time, communication\_time);

free(x\_old);

free(x\_new);

free(b);

free(sendcounts);

free(x\_new\_procs);

free(x\_old\_procs);

free(displs);

free(b\_procs);

}

MPI\_Finalize();

return(0);

}