МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ІНСТИТУТ КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ, АВТОМАТИКИ ТА МЕТРОЛОГІЇ



**Розрахунково-графічна робота**

з дисципліни: “Паралельні та розподілені обчислення”

Виконав:

ст.гр. КІ-34

Мороз О.Ю.

Прийняв:

Козак Н.Б.

**Львів 2020**

Завдання: відповідно до схеми з першої лабораторної роботи(замість матриць та векторів використовуються числа):



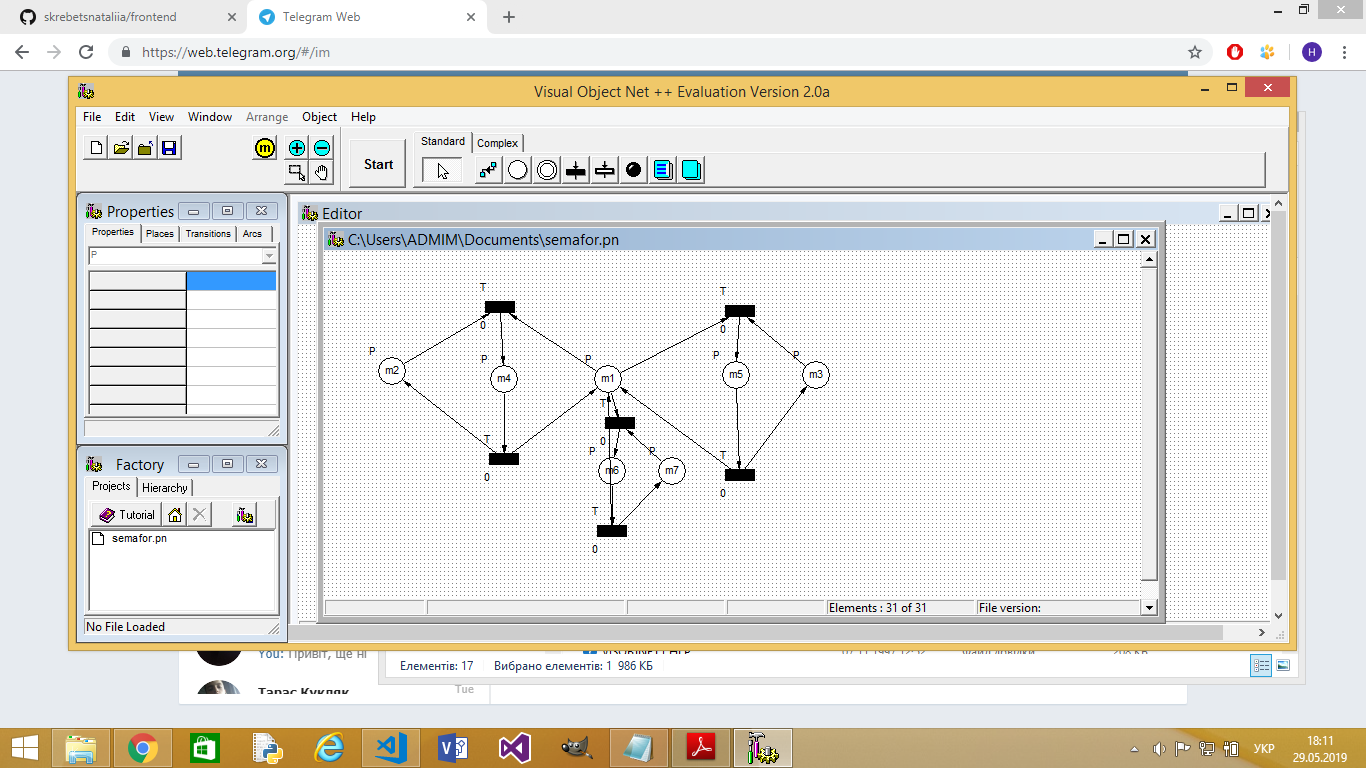
При чому елементи  визначаються згідно правил:

**y1=A×b**, де для парних **bi=1/(i2+2+i)** для непарних **bi=1/i**, **і=1,2,...n**

**y2=A1(b1-2c1)**;

**Y3 = A2­(B2-C2)**, де **Cij=1/(i+j)**

1) Реалізувати модель семафора для трьох процесів, що виконуються послідовно

Рис.2 Модель семафора

2)Реалізувати на MPI обчислення виразу

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <stdio.h>

#include "mpi.h"

// for run use command: {mpiexec\_\_path}/mpiexec -n 2 {mpi\_df\_\_path}/mpi\_df.exe

// A\*X + B

#define C 1/((1+2)\*(1+2)\*(1+2) )

#define X 24

#define B 7

#define RESULT (C \* X + B)

// (f0:A\*X) (f1:B)

// || ||

// \/ \/

// (f2:+)

void f0(int \* argArr, int \* resArr) {

MPI\_Request request;

//printf("f0\r\n");

argArr[0] = C;

argArr[1] = X;

resArr[0] = argArr[0] \* argArr[1];

//\_sleep(1000);

MPI\_Isend(resArr, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request);

MPI\_Request\_free(&request);

}

void f1(int \* argArr, int \* resArr) {

MPI\_Request request;

//printf("f1\r\n");

resArr[0] = B;

//\_sleep(1000);

MPI\_Isend(resArr, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request);

MPI\_Request\_free(&request);

}

void f2(int \* argArr, int \* resArr) {

MPI\_Status status;

MPI\_Request request;

//printf("f2\r\n");

MPI\_Recv(argArr, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

//printf("resv from f0 (value = %d)\r\n", argArr[0]);

MPI\_Recv(argArr + 1, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &status);

//printf("resv from f1 (value = %d)\r\n", argArr[1]);

resArr[0] = argArr[0] + argArr[1];

//\_sleep(1000);

MPI\_Isend(resArr, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request);

MPI\_Request\_free(&request);

}

#define MAX\_STAGE\_COUNT 2

#define MAX\_PE\_COUNT 2

void(\*fArr[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT])(int \* argArr, int \* resArr) = {

{ f0, f1 },

{ f2, NULL }

};

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

MPI\_Status status;

int procNum, procRank;//, recvRank;

int argArr[MAX\_PE\_COUNT \* 2], resArr[MAX\_PE\_COUNT \* 2];

MPI\_Init(&argc, &argv);

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

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

for (int iIndex = 0; iIndex < MAX\_STAGE\_COUNT; ++iIndex) {

switch (procRank) {

case 0:

if (fArr[iIndex][0]) fArr[iIndex][0](argArr, resArr);

break;

case 1:

if (fArr[iIndex][1]) fArr[iIndex][1](argArr, resArr);

break;

default:

break;

}

}

if (procRank == 0) {

MPI\_Recv(resArr, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

printf("result of execution = %d \r\n", \*resArr);

printf("expected result = %d \r\n", RESULT);

printf("-------------------------------\r\n");

if (\*resArr == RESULT) {

printf("verify status: succes\r\n");

}

else {

printf("verify status: not success\r\n");

}

}

MPI\_Finalize();

}

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

{

compute(argc, argv);

return 0;

}



Рис.2 Результат обчислень

3) Реалівати вираз на рис.1 за допомогою OpenMP(виконати паралельне виконання всіх операндів).

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <dos.h>

void f0(int \* argArr, int \* resArr) {

//resArr[0] = argArr[0] + argArr[1];

resArr[0] = 1 / (argArr[0] + argArr[1] );//resArr[1] = argArr[1]; // transport data to next stage

Sleep(1000);

}

void f1(int \* argArr, int \* resArr) {

resArr[1] = argArr[1]; // transport data to next stage

Sleep(1000);

}

void f2(int \* argArr, int \* resArr) {

resArr[2] = argArr[2]; // transport data to next stage

Sleep(1000);

}

void f3(int \* argArr, int \* resArr) {

resArr[3] = argArr[3]; // transport data to next stage

Sleep(1000);

}

void f4(int \* argArr, int \* resArr) {

resArr[4] = argArr[4]; // transport data to next stage

Sleep(1000);

}

void f5(int \* argArr, int \* resArr) {

resArr[5] = argArr[5]; // transport data to next stage

Sleep(1000);

}

void f6(int \* argArr, int \* resArr) {

if (argArr[6] % 2 == 0) {

resArr[6] = 1 / (argArr[6] \* argArr[6] + argArr[6] + 2); // transport data to next stage

}

else {

resArr[6] = 1 / argArr[6]; // transport data to next stage

}

Sleep(1000);

}

void f7(int \* argArr, int \* resArr) {

resArr[7] = argArr[7]; // transport data to next stage

Sleep(1000);

}

void sum1(int \* argArr, int \* resArr) {

argArr[0] = resArr[0]+resArr[1]; // transport data to next stage

Sleep(1000);

}

void sum2(int \* argArr, int \* resArr) {

argArr[1] = resArr[2] + resArr[3]; // transport da

Sleep(1000);

}

void sum3(int \* argArr, int \* resArr) {

argArr[2] = resArr[4] + resArr[5]; // transport da

Sleep(1000);

}

void sum3\_1(int \* argArr, int \* resArr) {

argArr[3] = resArr[6] + resArr[7]; // transport da

Sleep(1000);

}

void sum4(int \* argArr, int \* resArr) {

resArr[0] = argArr[0]+ argArr[1]; // transport data to next stage

Sleep(1000);

}

void sum5(int \* argArr, int \* resArr) {

resArr[1] = argArr[2]+argArr[3]; // transport data to next stage

Sleep(1000);

}

void sum6(int \* argArr, int \* resArr) {

argArr[0] = resArr[0]+resArr[1]; // transport data to next stage

Sleep(1000);

}

#define MAX\_STAGE\_COUNT 10

#define MAX\_PE\_COUNT 10

void(\*fArr[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT])(int \* argArr, int \* resArr) = {

{ f0, f1, f2, f3, f4, f5, f6, f7 },

{ sum1, NULL, sum2, NULL, sum3, NULL,sum3\_1, NULL },

{ sum4, NULL, NULL, NULL, sum5, NULL, NULL, NULL },

{ sum6, NULL, NULL, NULL, NULL, NULL, NULL, NULL },

};

void compute() {

int argArr[MAX\_PE\_COUNT \* 2], resArr[MAX\_PE\_COUNT \* 2];

for (int i = 0; i <MAX\_PE\_COUNT \*2; i++)

{

argArr[i] = 3;

}

int result;

if (argArr[6] % 2 == 0){

result = 1 / (argArr[0] + argArr[1] ) + argArr[1] + argArr[2] + argArr[3] + argArr[4] + argArr[5] + 1 / (argArr[6] \* argArr[6] +argArr[6] + 2) + argArr[7];

}

else {

result = 1 / (argArr[0] + argArr[1] ) + argArr[1] + argArr[2] + argArr[3] + argArr[4] + argArr[5] +( 1 / argArr[6] ) + argArr[7];

}

for (int iIndex = 0; iIndex < MAX\_STAGE\_COUNT; ++iIndex) {

#pragma omp parallel for shared(argArr, resArr)

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

else {

fArr[iIndex][jIndex](argArr, resArr);

}

}

}

printf("open mpi code - %d\n", argArr[0]);

if (argArr[6] % 2 == 0) {

printf("1 / (argArr[0] + argArr[1] ) + argArr[1] + argArr[2] + argArr[3] + argArr[4] + argArr[5] + 1 / (argArr[6] \* argArr[6] + argArr[6] + 2) +argArr[7]=%d", result);

}

else {

printf("1 / (argArr[0] + argArr[1] ) + argArr[1] + argArr[2] + argArr[3] + argArr[4] + argArr[5] + (1 / argArr[6]) +argArr[7]=%d", result);

}

}

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

compute();

return 0;

}

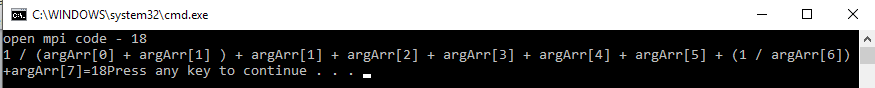


Рис.3 Результат обчислень

4. Реалізувати вираз на рис.1 за допомогою Іntel tbb (виконати паралельне виконання всіх операндів).

// N.Kozak // Lviv'2019

//#include "stdafx.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <stdio.h>

#include <stdlib.h>

//#include <windows.h>

#include <errno.h>

#include <pthread.h>

// open Ubuntu "Administration"

// for open disk C: cd /mnt/c/ && ls

// for compile: gcc -pthread -o pthread\_pro\_15\_df pthread\_pro\_15.cpp

// for run: ./pthread\_pro\_15\_df

// A\*X\*X\*X\*X\*X\*X\*X + B\*X\*X\*X\*X\*X\*X + C\*X\*X\*X\*X\*X + D\*X\*X\*X\*X + E\*X\*X\*X + F\*X\*X + G\*X + H

#define A 1

#define i 1

#define X 1

#define j 1

#define B 1

#define C 1

#define D 1

#define E 1

#define F 1

#define G 1

#define H 1

#define RESULT\_P ( i + B + C + D + E + F + (15 / (i\*i + j)) + H )

#define RESULT\_NP ( (15/i) + B + C + D + E + F + (15 / (i\*i + j)) + H )

// (f0:A\*X\*\*7) (f1:B\*X\*\*6) (f2:C\*X\*\*5) (f3:D\*X\*\*4) (f4:E\*X\*\*3) (f5:F\*X\*\*2) (f6:G\*X) (f7:H)

// || || || ||

// \/ \/ \/ \/

// (f8:+) (f9:+) (f10:+) (f11:+)

// || ||

// \/ \/

// (f12:+) (f13:+)

// ||

// \/

// (f14:+)

#define MAX\_STAGE\_COUNT 10

#define MAX\_PE\_COUNT 10

int argArr[MAX\_PE\_COUNT \* 2], resArr[MAX\_PE\_COUNT \* 2];

int \* argArr\_ = argArr, \* resArr\_ = resArr;

#define GET\_ARR(PARAM) \

int \* argArr, \* resArr;\

if (!(PARAM)) {\

argArr = argArr\_;\

resArr = resArr\_;\

}\

else{\

argArr = resArr\_;\

resArr = argArr\_;\

}

// stage with index 0

void \* f0(void \*param) {

GET\_ARR(param);

if (i % 2 == 0)

resArr[0] = i; // transport data to next stage

else

resArr[0] = 15 / i; // transport data to next stage

//sleep(1);

pthread\_exit(NULL);

}

void \* f1(void \*param) {

GET\_ARR(param);

resArr[1] = B;

//sleep(1);

pthread\_exit(NULL);

}

void \* f2(void \*param) {

GET\_ARR(param);

resArr[2] = C;

//sleep(1);

pthread\_exit(NULL);

}

void \* f3(void \*param) {

GET\_ARR(param);

resArr[3] = D;

//sleep(1);

pthread\_exit(NULL);

}

void \* f4(void \*param) {

GET\_ARR(param);

resArr[4] = E;

//sleep(1);

pthread\_exit(NULL);

}

void \* f5(void \*param) {

GET\_ARR(param);

resArr[5] = F;

//sleep(1);

pthread\_exit(NULL);

}

void \* f6(void \*param) {

GET\_ARR(param);

resArr[6] = 15 / (i\*i + j);

//sleep(1);

pthread\_exit(NULL);

}

void \* f7(void \*param) {

GET\_ARR(param);

resArr[7] = H;

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 1

void \* f8(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[1];

//sleep(1);

pthread\_exit(NULL);

}

void \* f9(void \*param) {

GET\_ARR(param);

resArr[2] = argArr[2] + argArr[3];

//sleep(1);

pthread\_exit(NULL);

}

void \* f10(void \*param) {

GET\_ARR(param);

resArr[4] = argArr[4] + argArr[5];

//sleep(1);

pthread\_exit(NULL);

}

void \* f11(void \*param) {

GET\_ARR(param);

resArr[6] = argArr[6] + argArr[7];

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 2

void \* f12(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[2];

//sleep(1);

pthread\_exit(NULL);

}

void \* f13(void \*param) {

GET\_ARR(param);

resArr[4] = argArr[4] + argArr[6];

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 3

void \* f14(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[4];

//sleep(1);

pthread\_exit(NULL);

}

void\*(\*fArr[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT])(void \*param) = {

{ f0, f1, f2, f3, f4, f5, f6, f7 },

{ f8, NULL, f9, NULL, f10, NULL, f11, NULL },

{ f12, NULL, NULL, NULL, f13, NULL, NULL, NULL },

{ f14, NULL, NULL, NULL, NULL, NULL, NULL, NULL },

};

#define LAST\_STAGE\_INDEX 3

void compute(){

pthread\_t threads[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT];

for (int iIndex = 0; iIndex < MAX\_STAGE\_COUNT; ++iIndex) {

// create threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

int retunValue;

while(EAGAIN == (retunValue = pthread\_create(&threads[iIndex][jIndex], NULL, fArr[iIndex][jIndex], (void\*)(long long int)(iIndex % 2))));

if(retunValue){

printf("Thread create error. Error code %X\r\n", retunValue);

return;

}

}

// join threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

pthread\_join(threads[iIndex][jIndex], NULL);

}

}

int \* res = (LAST\_STAGE\_INDEX + 1) % 2 ? resArr : argArr;

if (i % 2 == 0) {

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_P);

printf("-------------------------------\r\n");

if (\*res == RESULT\_P){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}

else{

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_NP);

printf("-------------------------------\r\n");

if (\*res == RESULT\_NP){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}}

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

compute();

return 0;

}



Рис.4 Результат обчислень

5) Реалізувати вираз на рис.1 за допомогою Boost (виконати паралельне виконання всіх операндів).

// N.Kozak // Lviv'2019

//#include "stdafx.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <stdio.h>

#include <stdlib.h>

//#include <windows.h>

#include <errno.h>

#include <boost/thread/thread.hpp>

// for compile and run use web resource https://wandbox.org/

// A\*X\*X\*X\*X\*X\*X\*X + B\*X\*X\*X\*X\*X\*X + C\*X\*X\*X\*X\*X + D\*X\*X\*X\*X + E\*X\*X\*X + F\*X\*X + G\*X + H

#define A 1

#define i 1

#define X 1

#define j 1

#define B 1

#define C 1

#define D 1

#define E 1

#define F 1

#define G 1

#define H 1

#define RESULT\_P ( 1/(i\*i+i+2) + B + C + D + E + F + (1 / (i + j)) + H )

#define RESULT\_NP ( (1/i) + B + C + D + E + F + (1 / (i + j)) + H )

// (f0:A\*X\*\*7) (f1:B\*X\*\*6) (f2:C\*X\*\*5) (f3:D\*X\*\*4) (f4:E\*X\*\*3) (f5:F\*X\*\*2) (f6:G\*X) (f7:H)

// || || || ||

// \/ \/ \/ \/

// (f8:+) (f9:+) (f10:+) (f11:+)

// || ||

// \/ \/

// (f12:+) (f13:+)

// ||

// \/

// (f14:+)

// stage with index 0

void f0(int \* argArr, int \* resArr) {

if (i % 2 == 0)

resArr[0] = 1/(i\*i+i+2); // transport data to next stage

else

resArr[0] = 1 / i; // transport data to next stage

//sleep(1);

}

void f1(int \* argArr, int \* resArr) {

resArr[1] = B;

//sleep(1);

}

void f2(int \* argArr, int \* resArr) {

resArr[2] = C;

//sleep(1);

}

void f3(int \* argArr, int \* resArr) {

resArr[3] = D;

//sleep(1);

}

void f4(int \* argArr, int \* resArr) {

resArr[4] = E;

//sleep(1);

}

void f5(int \* argArr, int \* resArr) {

resArr[5] = F;

//sleep(1);

}

void f6(int \* argArr, int \* resArr) {

resArr[6] = 1 / (i + j); // transport data to next stage

}

void f7(int \* argArr, int \* resArr) {

resArr[7] = H;

//sleep(1);

}

// stage with index 1

void f8(int \* argArr, int \* resArr) {

resArr[0] = argArr[0] + argArr[1];

//sleep(1);

}

void f9(int \* argArr, int \* resArr) {

resArr[2] = argArr[2] + argArr[3];

//sleep(1);

}

void f10(int \* argArr, int \* resArr) {

resArr[4] = argArr[4] + argArr[5];

//sleep(1);

}

void f11(int \* argArr, int \* resArr) {

resArr[6] = argArr[6] + argArr[7];

//sleep(1);

}

// stage with index 2

void f12(int \* argArr, int \* resArr) {

resArr[0] = argArr[0] + argArr[2];

//sleep(1);

}

void f13(int \* argArr, int \* resArr) {

resArr[4] = argArr[4] + argArr[6];

//sleep(1);

}

// stage with index 3

void f14(int \* argArr, int \* resArr) {

resArr[0] = argArr[0] + argArr[4];

//sleep(1);

}

#define MAX\_STAGE\_COUNT 10

#define MAX\_PE\_COUNT 10

void(\*fArr[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT])(int \* argArr, int \* resArr) = {

{ f0, f1, f2, f3, f4, f5, f6, f7 },

{ f8, NULL, f9, NULL, f10, NULL, f11, NULL },

{ f12, NULL, NULL, NULL, f13, NULL, NULL, NULL },

{ f14, NULL, NULL, NULL, NULL, NULL, NULL, NULL },

};

#define LAST\_STAGE\_INDEX 3

void compute(){

int argArr[MAX\_PE\_COUNT \* 2], resArr[MAX\_PE\_COUNT \* 2];

boost::thread \* boostThreadPtrs[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT];

for (int iIndex = 0; iIndex < MAX\_STAGE\_COUNT; ++iIndex) {

// create threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

if (!(iIndex % 2)) boostThreadPtrs[iIndex][jIndex] = new boost::thread(fArr[iIndex][jIndex], argArr, resArr);

else boostThreadPtrs[iIndex][jIndex] = new boost::thread(fArr[iIndex][jIndex], resArr, argArr);

}

// join threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

boostThreadPtrs[iIndex][jIndex]->join();

delete boostThreadPtrs[iIndex][jIndex];

}

}

int \* res = (LAST\_STAGE\_INDEX + 1) % 2 ? resArr : argArr;

if (i % 2 == 0) {

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_P);

printf("-------------------------------\r\n");

if (\*res == RESULT\_P){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}

else{

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_NP);

printf("-------------------------------\r\n");

if (\*res == RESULT\_NP){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}

}

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

compute();

return 0;

}

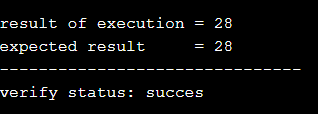


Рис.5 Результат обчислень

6. Реалізувати вираз на рис.1 за допомогою Pthread (виконати паралельне виконання всіх операндів).

// N.Kozak // Lviv'2019

//#include "stdafx.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <stdio.h>

#include <stdlib.h>

//#include <windows.h>

#include <errno.h>

#include <pthread.h>

// open Ubuntu "Administration"

// for open disk C: cd /mnt/c/ && ls

// for compile: gcc -pthread -o pthread\_pro\_15\_df pthread\_pro\_15.cpp

// for run: ./pthread\_pro\_15\_df

// A\*X\*X\*X\*X\*X\*X\*X + B\*X\*X\*X\*X\*X\*X + C\*X\*X\*X\*X\*X + D\*X\*X\*X\*X + E\*X\*X\*X + F\*X\*X + G\*X + H

#define A 1

#define i 1

#define X 1

#define j 1

#define B 1

#define C 1

#define D 1

#define E 1

#define F 1

#define G 1

#define H 1

#define RESULT\_P ( 1/(i\*i+i+2) + B + C + D + E + F + (1 / (i + j)) + H )

#define RESULT\_NP ( (1/i) + B + C + D + E + F + (1 / (i + j)) + H )

// (f0:A\*X\*\*7) (f1:B\*X\*\*6) (f2:C\*X\*\*5) (f3:D\*X\*\*4) (f4:E\*X\*\*3) (f5:F\*X\*\*2) (f6:G\*X) (f7:H)

// || || || ||

// \/ \/ \/ \/

// (f8:+) (f9:+) (f10:+) (f11:+)

// || ||

// \/ \/

// (f12:+) (f13:+)

// ||

// \/

// (f14:+)

#define MAX\_STAGE\_COUNT 10

#define MAX\_PE\_COUNT 10

int argArr[MAX\_PE\_COUNT \* 2], resArr[MAX\_PE\_COUNT \* 2];

int \* argArr\_ = argArr, \* resArr\_ = resArr;

#define GET\_ARR(PARAM) \

int \* argArr, \* resArr;\

if (!(PARAM)) {\

argArr = argArr\_;\

resArr = resArr\_;\

}\

else{\

argArr = resArr\_;\

resArr = argArr\_;\

}

// stage with index 0

void \* f0(void \*param) {

GET\_ARR(param);

if (i % 2 == 0)

resArr[0] = 1/(i\*i+i+2); // transport data to next stage

else

resArr[0] = 1 / i; // transport data to next stage

//sleep(1);

pthread\_exit(NULL);

}

void \* f1(void \*param) {

GET\_ARR(param);

resArr[1] = B;

//sleep(1);

pthread\_exit(NULL);

}

void \* f2(void \*param) {

GET\_ARR(param);

resArr[2] = C;

//sleep(1);

pthread\_exit(NULL);

}

void \* f3(void \*param) {

GET\_ARR(param);

resArr[3] = D;

//sleep(1);

pthread\_exit(NULL);

}

void \* f4(void \*param) {

GET\_ARR(param);

resArr[4] = E;

//sleep(1);

pthread\_exit(NULL);

}

void \* f5(void \*param) {

GET\_ARR(param);

resArr[5] = F;

//sleep(1);

pthread\_exit(NULL);

}

void \* f6(void \*param) {

GET\_ARR(param);

resArr[6] = 1 / (i + j);

//sleep(1);

pthread\_exit(NULL);

}

void \* f7(void \*param) {

GET\_ARR(param);

resArr[7] = H;

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 1

void \* f8(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[1];

//sleep(1);

pthread\_exit(NULL);

}

void \* f9(void \*param) {

GET\_ARR(param);

resArr[2] = argArr[2] + argArr[3];

//sleep(1);

pthread\_exit(NULL);

}

void \* f10(void \*param) {

GET\_ARR(param);

resArr[4] = argArr[4] + argArr[5];

//sleep(1);

pthread\_exit(NULL);

}

void \* f11(void \*param) {

GET\_ARR(param);

resArr[6] = argArr[6] + argArr[7];

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 2

void \* f12(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[2];

//sleep(1);

pthread\_exit(NULL);

}

void \* f13(void \*param) {

GET\_ARR(param);

resArr[4] = argArr[4] + argArr[6];

//sleep(1);

pthread\_exit(NULL);

}

// stage with index 3

void \* f14(void \*param) {

GET\_ARR(param);

resArr[0] = argArr[0] + argArr[4];

//sleep(1);

pthread\_exit(NULL);

}

void\*(\*fArr[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT])(void \*param) = {

{ f0, f1, f2, f3, f4, f5, f6, f7 },

{ f8, NULL, f9, NULL, f10, NULL, f11, NULL },

{ f12, NULL, NULL, NULL, f13, NULL, NULL, NULL },

{ f14, NULL, NULL, NULL, NULL, NULL, NULL, NULL },

};

#define LAST\_STAGE\_INDEX 3

void compute(){

pthread\_t threads[MAX\_STAGE\_COUNT][MAX\_PE\_COUNT];

for (int iIndex = 0; iIndex < MAX\_STAGE\_COUNT; ++iIndex) {

// create threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

int retunValue;

while(EAGAIN == (retunValue = pthread\_create(&threads[iIndex][jIndex], NULL, fArr[iIndex][jIndex], (void\*)(long long int)(iIndex % 2))));

if(retunValue){

printf("Thread create error. Error code %X\r\n", retunValue);

return;

}

}

// join threads

for (int jIndex = 0; jIndex < MAX\_PE\_COUNT; ++jIndex) {

if (!fArr[iIndex][jIndex]) continue;

pthread\_join(threads[iIndex][jIndex], NULL);

}

}

int \* res = (LAST\_STAGE\_INDEX + 1) % 2 ? resArr : argArr;

if (i % 2 == 0) {

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_P);

printf("-------------------------------\r\n");

if (\*res == RESULT\_P){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}

else{

printf("result of execution = %d \r\n", \*res);

printf("expected result = %d \r\n", RESULT\_NP);

printf("-------------------------------\r\n");

if (\*res == RESULT\_NP){

printf("verify status: succes\r\n");

}

else{

printf("verify status: not success\r\n");

}

}}

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

compute();

return 0;

}

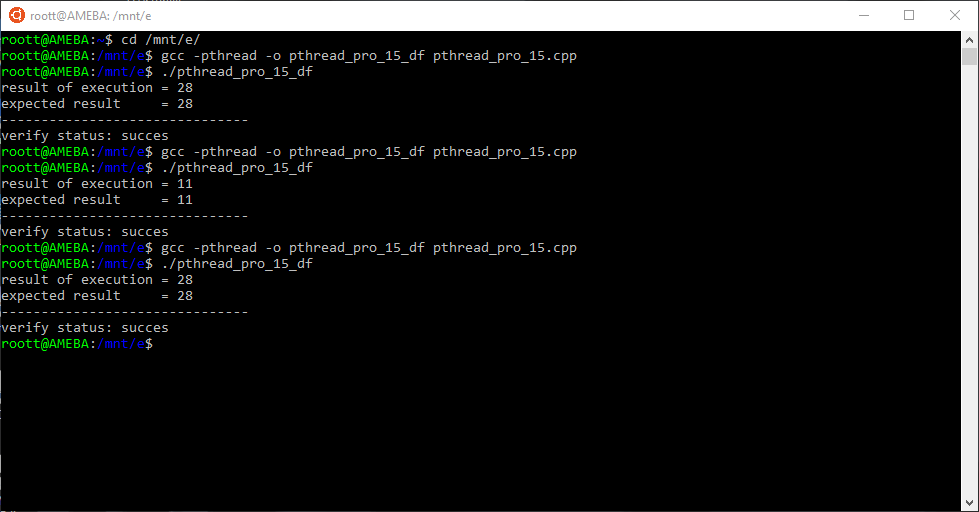


Рис.6 Результат обчислень

8.Оволодіти навичками програмування та налаштування BOINC- серверу

Результат роботи:

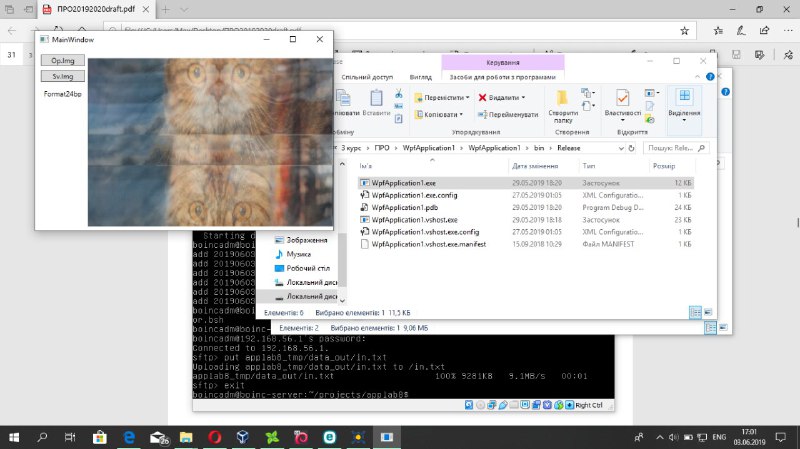
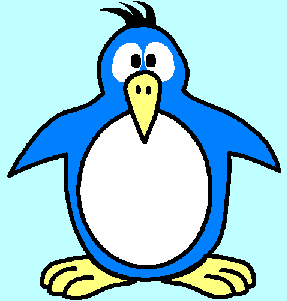


Рис.7 Результат роботи

9. Навчитися моделювати розподілені обчислення шляхом використання засобів програмування для веб-застосунків.

Результат роботи:

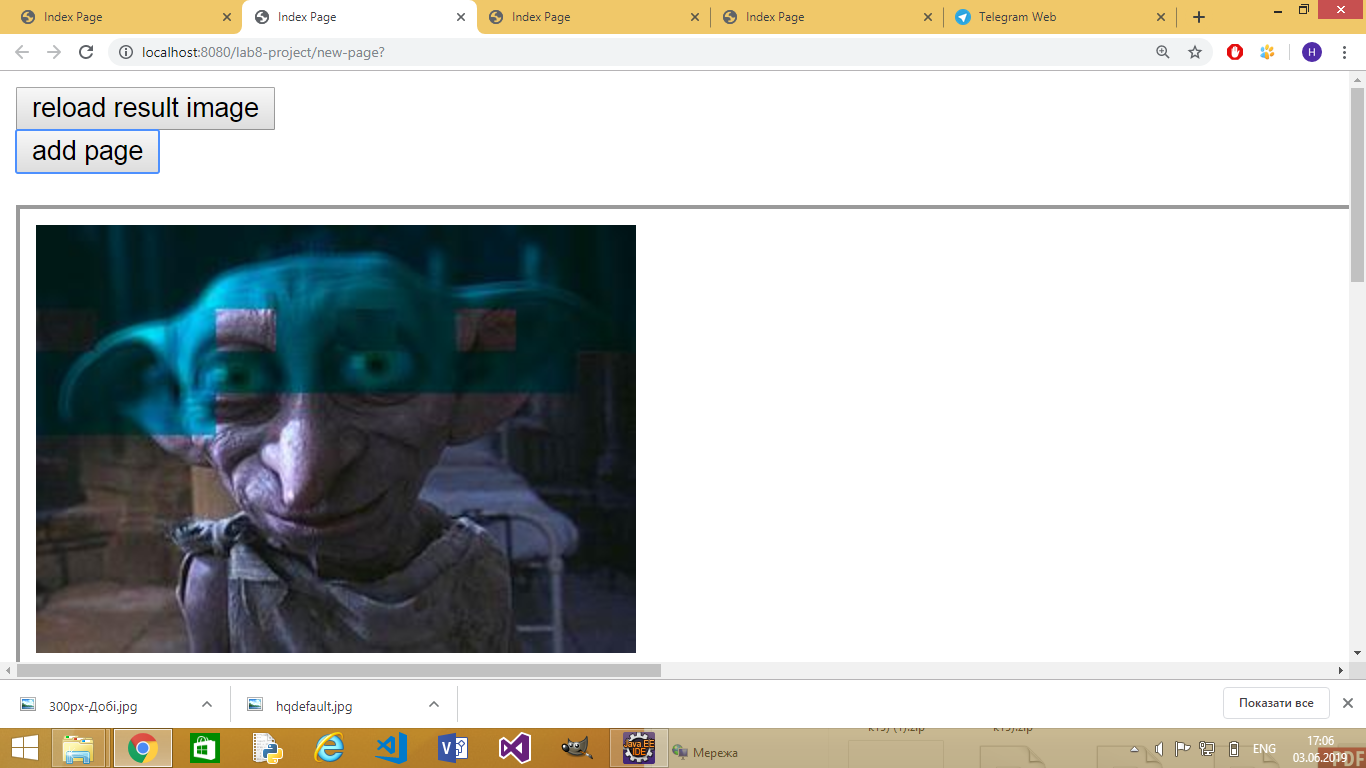
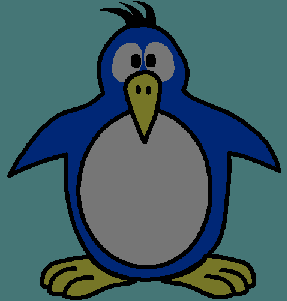


Рис.8 Результат роботи