/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* Parallel and Distributed Computing \*

\* Task: A=sort(a\*B+C\*(MX\*MD)) \*

\* @author Kravchenko Ivan \*

\* @group IO-01 \*

\* @date 18.04.13 \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

#include "stdafx.h"

#include "mpi.h"

#include "Data.h"

#include <iostream>

using namespace std;

int N=200;

int P=4;

int H=N/P;

//ініціалізація допоміжних векторів та матриць

int \*MD = CreateMatrix(N, N);

int \*B = CreateVector(N);

int \*A = CreateVector(N);

int \*AH = CreateVector(H);

int \*DH1 = CreateVector(H);

int \*AH1 = CreateVector(2\*H);

int \*AH34 = CreateVector(2\*H);

int \*BH = CreateVector(H);

int \*C = CreateVector(N);

int \*ML = CreateMatrix(N, H);

int \*MDH = CreateMatrix(N, H);

int \*MKH = CreateMatrix(N, H);

int \*MX = CreateMatrix(N, 3\*H);

int \*MC = CreateMatrix(N, N);

int a;

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

{

cout<<"start"<<endl;

MPI\_Init(&argc, &argv);

int Tid;

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

if (Tid>5)

{

MPI\_Finalize();

return 0;

}

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

cout<<"Thread "<<Tid<<" started"<<endl;

double time = MPI\_Wtime();

MPI\_Status status;

if (Tid==4){

//введення даних

FillMatrix(MD, N, N, 1);

FillVector(B, N, 1);

FillVector(C, N, 1);

FillMatrix(MX, N, N, 1);

a=1;

//розсилання векторів та матриць іншим процесорам

Chenge(B,BH,0,H);

MPI\_Send(BH, H, MPI\_INT, 0, 3, MPI\_COMM\_WORLD);

Chenge(B,BH,H,H);

MPI\_Send(BH, H, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

Chenge(B,BH,2\*H,H);

MPI\_Send(BH, H, MPI\_INT, 2, 3, MPI\_COMM\_WORLD);

Chenge(B,BH,3\*H,H);

MPI\_Send(BH, H, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

Chenge1(MD,MDH,0,H);

MPI\_Send(MDH, H\*N, MPI\_INT, 0, 3, MPI\_COMM\_WORLD);

Chenge1(MD,MDH,H,H);

MPI\_Send(MDH, H\*N, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

Chenge1(MD,MDH,2\*H,H);

MPI\_Send(MDH, H\*N, MPI\_INT, 2, 3, MPI\_COMM\_WORLD);

Chenge1(MD,MDH,3\*H,H);

MPI\_Send(MDH, H\*N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send(C, N, MPI\_INT, 0, 3, MPI\_COMM\_WORLD);

MPI\_Send(C, N, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(C, N, MPI\_INT, 2, 3, MPI\_COMM\_WORLD);

MPI\_Send(C, N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send(&a, 1, MPI\_INT, 0, 3, MPI\_COMM\_WORLD);

MPI\_Send(&a, 1, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(&a, 1, MPI\_INT, 2, 3, MPI\_COMM\_WORLD);

MPI\_Send(&a, 1, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*N, MPI\_INT, 0, 3, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*N, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*N, MPI\_INT, 2, 3, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

}

if (Tid!= 4){

// прийняття процесорами даних від процесора №5

MPI\_Recv(BH, H, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MDH, H\*N, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(C, N, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&a, 1, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MX, N\*N, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

//обчислення AH=a\*BH+C\*(MX\*MDH), AH = Sort(АH)

for (int i=0;i<H;i++){

BH[i]=a\*BH[i];

}

for (int i=0;i<H;i++){

for (int j=0;j<N;j++){

int temp=0;

for (int q = 0; q < N; q++){

temp = temp + MX[j, q] \* MDH[q, i];

}

MKH[i, j] = temp;

}

}

for (int i=0; i<H ; i++){

int temp=0;

for(int j=0;j<N;j++){

temp += C[j] \* MKH[i, j];

}

AH[i]=temp;

}

for (int i=0;i<H;i++){

AH[i]=AH[i]+BH[i];

}

int k,r;

for (k = 0; k < H-1;k++ ){

for (int i = 0; i < H - 1; i++){

if (AH[i] < AH[i + 1]){

r = AH[i];

AH[i] = AH[i + 1];

AH[i + 1] = r;

}

}

}

cout<<"Tid"<<Tid;

} //виконання розсилки та прийому даних для злиття проміжних результатів сортування

if (Tid==0) MPI\_Send(AH, H, MPI\_INT, 4, 12, MPI\_COMM\_WORLD);

if (Tid==2) MPI\_Send(AH, H, MPI\_INT, 4, 2, MPI\_COMM\_WORLD);

if (Tid==4) {

MPI\_Recv(AH, H, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(AH, H, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Recv(AH, H, MPI\_INT, 2, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(AH, H, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

}

if (Tid==3){

DH1=AH;

MPI\_Recv(AH, H, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

mergeSort(AH,DH1,0,1,H,N,AH1);

}

if (Tid==1){

DH1=AH;

MPI\_Recv(AH, H, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

mergeSort(AH,DH1,0,1,H,N,AH1);

MPI\_Send(AH1, 2\*H, MPI\_INT, 4, 10, MPI\_COMM\_WORLD);

}

if (Tid==4){

MPI\_Recv(AH1, 2\*H, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(AH1, 2\*H, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

}

if(Tid==3){

AH34=AH1;

MPI\_Recv(AH1, 2\*H, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

mergeSort(AH1,AH34,0,2,H,N,A);

MPI\_Send(A, N, MPI\_INT, 4, 3, MPI\_COMM\_WORLD);

}

if (Tid==4){

MPI\_Recv(A, N, MPI\_INT, 3, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

//OutVector(A,N);

time-=MPI\_Wtime();

cout<<"Time: "<<-time<<endl;

}

cout<<"Thread "<<Tid<<" finished"<<endl;

MPI\_Finalize();

return 0;

}

#include "StdAfx.h"

#include <iostream>

#include "Data.h"

using namespace std;

int\* CreateVector(int n)

{

int\* Vector = new int[n];

return Vector;

}

int\* CreateMatrix(int n, int m)

{

int\* Matrix = new int[n\*m];

return Matrix;

}

void FillVector(int\* vector, int n, int c)

{

for (int i=0; i<n; i++)

vector[i]=c;

}

void FillMatrix(int\* matrix, int n, int m, int c)

{

for (int i=0; i<n; i++)

{

for (int j=0; j<m; j++)

{

matrix[i\*m+j]=c;

}

}

}

void OutVector(int\* vector, int n)

{

for (int i=0; i<n; i++)

cout<<vector[i]<<'\t';

}

void OutMatrix(int\* matrix, int n, int m)

{

for (int i=0; i<n; i++)

{

for (int j=0; j<m; j++)

{

cout<<matrix[i\*m+j]<<'\t';

}

cout<<endl;

}

}

void mergeSort(int\* Mas,int\* Mas1, int task,int q,int H,int N,int\* M3)

{

int i = (task\*H)\*q, j = (task\*H)\*q;

int k = (task \* H)\*q;

for (; k <((task+2) \* H)\*q; k++){

if (Mas[i] < Mas1[j])

{

M3[k] = Mas[i];

i++;

}

else {

M3[k] = Mas1[j];

j ++;

}

if (i > (task + 1) \* H\*q)

{

for (; k < (task + 2) \* H\*q; k++)

{

M3[k] = Mas1[j];

j++;

}

}

if (j > (task + 1) \* H\*q)

{

for (; k < (task + 2) \* H\*q; k++)

{

M3[k] = Mas[i];

i++;

}

}

}

}

void Chenge(int\* A,int\* B,int a,int H){

for(int i=0; i<H;i++) {

B[i]=A[i+a];

}

}

void Chenge1(int\* A,int\* B,int a,int H){

for(int i=0; i<H;i++) {

for (int j=0;j<4\*H;j++){

B[i,j]=A[i+a,j];

}

}

}

#pragma once

int\* CreateVector(int n);

int\* CreateMatrix(int n, int m);

void FillVector(int\* vector, int n, int c);

void FillMatrix(int\* matrix, int n, int m, int c);

void OutVector(int\* vector, int n);

void OutMatrix(int\* matrix, int n, int m);

void mergeSort(int\* Mas,int\* Mas1, int task,int q,int H,int N,int\* M3);

void Chenge(int\* A,int\* B,int a,int H);

void Chenge1(int\* A,int\* B,int a,int H);

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* Parallel and Distributed Computing \*

\* Task: A=sort(a\*B+C\*(MX\*MD)) \*

\* @author Kravchenko Ivan \*

\* @group IO-01 \*

\* @date 18.04.13 \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace kursova\_2

{

class Program

{

static int N = 1000;

static int P = 4;

static int H = N / P;

public static Semaphore sem1, sem2, sem3, sem4, sem5, sem6;

public static EventWaitHandle event2, event1;

public static Mutex mutex\_m = new Mutex(false);

public static object copy\_lock = new object();

public static object monitor\_x = new object();

public static int a;

public static int[] A = new int[N];

public static int[,] MX;

public static int[,] MD;

public static int[] B;

public static int[] C;

static void Main(string[] args)

{

Console.WriteLine("Start");

DateTime dold = DateTime.Now;

event2 = new ManualResetEvent(false);

event1 = new ManualResetEvent(false);

sem1 = new Semaphore(0, 3);

sem2 = new Semaphore(0, 3);

sem3 = new Semaphore(0, 3);

sem4 = new Semaphore(0, 3);

sem5 = new Semaphore(0, 3);

sem6 = new Semaphore(0, 3);

Thread t1 = new Thread(T1);

Thread t2 = new Thread(T2);

Thread t3 = new Thread(T3);

Thread t4 = new Thread(T4);

t1.Start();

t2.Start();

t3.Start();

t4.Start();

t1.Join();

t2.Join();

t3.Join();

t4.Join();

TimeSpan sp = DateTime.Now - dold;

Console.WriteLine(" finish");

Console.WriteLine(sp);

Console.ReadKey();

}

public static int[,] inputMatrix(int n, int value)

{

int[,] matrix = new int[n, n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

matrix[i, j] = value;

}

}

return matrix;

}

public static int[] inputVector(int n, int value)

{

int[] vector = new int[n];

for (int i = 0; i < n; i++)

{

vector[i] = value;

}

return vector;

}

public static void outputVector(int[] vector)

{

if (N < 13)

{

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

{

Console.WriteLine(vector[i]);

}

}

}

public static int[] add(int[] X, int[] Y, int l, int r, int[] Z)

{

for (int i = l; i < r; i++)

{

Z[i] = X[i] + Y[i];

}

return Z;

}

public static int[] multiply(int x, int[] V, int l, int r, int[] VZ)

{

for (int i = l; i < r; i++)

{

VZ[i] = x \* V[i];

}

return VZ;

}

public static int[,] multiply(int[,] MX, int[,] MY, int l, int r, int[,] MZ)

{

int temp;

for (int i = l; i < r; i++)

{

for (int j = 0; j < N; j++)

{

temp = 0;

for (int q = 0; q < N; q++)

{

temp = temp + MX[j, q] \* MY[q, i];

}

MZ[i, j] = temp;

}

}

return MZ;

}

public static void copy(int[,] MS, int[,] MD)

{

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

{

for (int j = 0; j < N; j++)

{

MD[i, j] = MS[i, j];

}

}

}

public static void copy(int[] S, int[] D)

{

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

{

D[i] = S[i];

}

}

public static int[] mul(int[] A, int[,] MB, int l, int r, int[] Z)

{

// int temp;

for (int i = l; i < r; i++)

{

int temp = 0;

for (int j = 0; j < N; j++)

{

temp += A[j] \* MB[i, j];

}

Z[i] = temp;

// Console.WriteLine("Z"+i+Z[i]);

}

return Z;

}

public static void Sort(int[] mas, int task)

{

int k;

int r;

for (k = 0; k < H \* (task + 1) - 1;k++ )

{

for (int i = task \* H; i < H \* (task + 1) - 1; i++)

{

if (mas[i] < mas[i + 1])

{

r = mas[i];

mas[i] = mas[i + 1];

mas[i + 1] = r;

}

}

}

}

public static void mergeSort(int[] Mas, int task,int q)

{

int i = (task\*H)\*q, j = ((task+1)\*H\*q-1);

int[] M3 = new int[N];

int k = (task \* H)\*q;

for (; k <((task+2) \* H)\*q; k++){

if (Mas[i] < Mas[j])

{

M3[k] = Mas[i];

i++;

}

else {

M3[k] = Mas[j];

j ++;

}

if (i >= (task + 1) \* H\*q)

{

for (; k < (task + 2) \* H\*q; k++)

{

M3[k] = Mas[j];

j++;

}

}

if (j >= (task + 2) \* H\*q)

{

for (; k < (task + 2) \* H\*q; k++)

{

M3[k] = Mas[i];

i++;

}

}

}

for ( k = task \* H\*q; k < (task + 1) \* H\*q; k++) {

Mas[k]=M3[k];

}

}

//TASK 1--------------------START-------------------------------------------------

public static void T1()

{

Console.WriteLine("TASK 1 started");

int task = 0;

bool LAST = task == P - 1;

int L\_I = H \* task;

int R\_I = (LAST ? N : H \* (task + 1));

int[,] MX1;

int[,] MA;

int[] B1 = new int[N];

int[] F = new int[N];

int[] C1 = new int[N];

MX1 = new int[N, N];

MA = new int[N, N];

int[] D = new int[N];

sem1.WaitOne();

sem2.WaitOne();

sem3.WaitOne();

Monitor.Enter(monitor\_x);

copy(C, C1);

Monitor.Exit(monitor\_x);

mutex\_m.WaitOne();

int a1 = a;

mutex\_m.ReleaseMutex();

lock (copy\_lock){

copy(MX, MX1);

}

multiply(a1, B, L\_I, R\_I, F);

multiply(MD, MX1, L\_I, R\_I, MA);

mul(C1, MA, L\_I, R\_I, D);

add(F, D, L\_I, R\_I, A);

Sort(A,task);

sem4.WaitOne();

mergeSort(A,task,1);

event1.WaitOne();

mergeSort(A, task, 2);

event2.Set();

Console.WriteLine("TASK 1 finished");

}

//TASK 1---------------------END--------------------------------------------------

//TASK 2--------------------START-------------------------------------------------

public static void T2()

{

Console.WriteLine("TASK 2 started");

int task = 1;

bool LAST = task == P - 1;

int L\_I = H \* task;

int R\_I = (LAST ? N : H \* (task + 1));

int[,] MX2;

int[,] MA;

int[] K = new int[N];

int[] B2 = new int[N];

int[] F = new int[N];

int[] C2 = new int[N];

MX2 = new int[N, N];

MA = new int[N, N];

int[] D = new int[N];

C=inputVector(N,1);

B=inputVector(N,1);

sem1.Release(3);

sem2.WaitOne();

sem3.WaitOne();

Monitor.Enter(monitor\_x);

copy(C, C2);

Monitor.Exit(monitor\_x);

mutex\_m.WaitOne();

int a2 = a;

mutex\_m.ReleaseMutex();

lock (copy\_lock)

{

copy(MX, MX2);

}

multiply(a2, B, L\_I, R\_I, F);

multiply(MD, MX2, L\_I, R\_I, MA);

mul(C2, MA, L\_I, R\_I, D);

add(F, D, L\_I, R\_I, A);

Sort(A, task);

sem4.Release(1);

event2.WaitOne();

outputVector(A);

Console.WriteLine("TASK 2 finished");

}

//TASK 2---------------------END--------------------------------------------------

//TASK 3--------------------START-------------------------------------------------

public static void T3()

{

Console.WriteLine("TASK 3 started");

int task = 2;

bool LAST = task == P - 1;

int L\_I = H \* task;

int R\_I = (LAST ? N : H \* (task + 1));

int[,] MX3;

int[,] MA;

// int[] K = new int[N];

int[] B3 = new int[N];

int[] F = new int[N];

int[] C3 = new int[N];

MX3 = new int[N, N];

MA = new int[N, N];

int[] D = new int[N];

MX=inputMatrix(N, 1);

a = 1;

sem2.Release(3);

sem1.WaitOne();

sem3.WaitOne();

Monitor.Enter(monitor\_x);

copy(C, C3);

Monitor.Exit(monitor\_x);

mutex\_m.WaitOne();

int a3 = a;

mutex\_m.ReleaseMutex();

lock (copy\_lock)

{

copy(MX, MX3);

}

multiply(a3, B, L\_I, R\_I, F);

multiply(MD, MX3, L\_I, R\_I, MA);

mul(C3, MA, L\_I, R\_I, D);

add(F, D, L\_I, R\_I, A);

Sort(A, task);

sem5.WaitOne();

mergeSort(A, task,1);

event1.Set();

Console.WriteLine("TASK 3 finished");

}

//TASK 3---------------------END--------------------------------------------------

//TASK 4--------------------START-------------------------------------------------

public static void T4()

{

Console.WriteLine("TASK 4 started");

int task = 3;

bool LAST = task == P - 1;

int L\_I = H \* task;

int R\_I = (LAST ? N : H \* (task + 1));

int[,] MX4;

int[,] MA;

int[] B4 = new int[N];

int[] F = new int[N];

int[] C4 = new int[N];

MX4 = new int[N, N];

MA = new int[N, N];

int[] D = new int[N];

MD = inputMatrix(N, 1);

sem3.Release(3);

sem1.WaitOne();

sem2.WaitOne();

Monitor.Enter(monitor\_x);

copy(C, C4);

Monitor.Exit(monitor\_x);

mutex\_m.WaitOne();

int a4 = a;

mutex\_m.ReleaseMutex();

lock (copy\_lock)

{

copy(MX, MX4);

}

multiply(a4, B, L\_I, R\_I, F);

multiply(MD, MX4, L\_I, R\_I, MA);

mul(C4, MA, L\_I, R\_I, D);

add(F, D, L\_I, R\_I, A);

Sort(A, task);

sem5.Release(1);

Console.WriteLine("TASK 4 finished");

}

//TASK 4---------------------END--------------------------------------------------

}

}