Лабораторная работа №6

по курсу:

«Паралельные и распределённые вычисления»

Тема: «OpenMP»

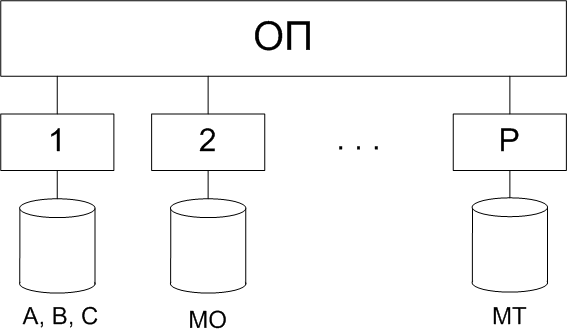
Выполнил: студент группы ИВ-83

НТУУ «КПИ» ФИВТ

Воробйов Виталий

**Техническое задание**

A=B + C \* (MO \* MT)



**Этап 1. Построение параллельного алгоритма**

Вычисление данного матричного уравнения можно разбить на шаги:

1. AH = BH + C \* (MO \* MTH)

**Общие ресурсы:** C, MO

**Этап 2. Разработка алгоритмов процессов (задач)**

**№ Алгоритм задачи №1 ТС/КУ**

1. Ввод B, C
2. Сигнал всем задачам о завершении ввода S\*.1
3. Ждать введения данных в задачах №2 и №P W2.1 WP.1
4. Копирование С1 := С и МО1 := МО КУ
5. Cчёт AH = BH + C1 \* (MO1 \* MTH)
6. Ждать окончания счёта во всех задачах. W\*.2
7. Вывод А

**№ Алгоритм задачи №2 ТС/КУ**

1. Ввод МО
2. Сигнал всем задачам о завершении ввода S\*.1
3. Ждать введения данных в задачах №1 и №P W1.1 WP.1
4. Копирование С2 := С и МО2 := МО КУ
5. Cчёт AH = BH + C2 \* (MO2 \* MTH)
6. Сигнал задаче №1 об окончании счёта S1.2

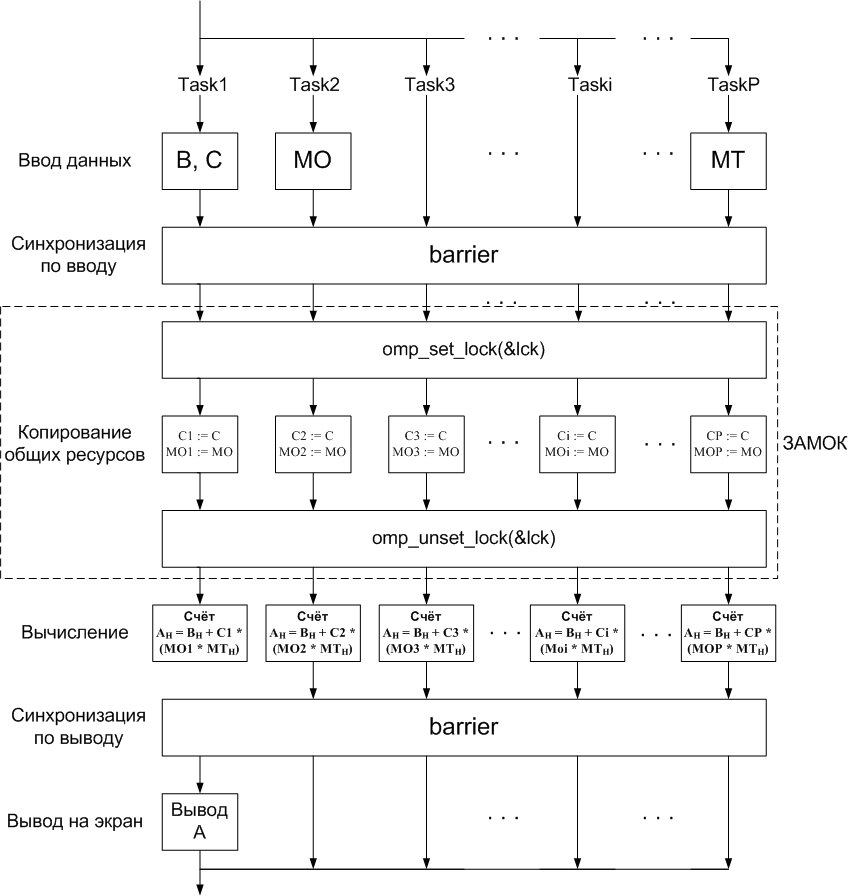
**№ Алгоритм задачи №P ТС/КУ**

1. Ввод МT
2. Сигнал всем задачам о завершении ввода S\*.1
3. Ждать введения данных в задачах №1 и №2 W1.1 W2.1
4. Копирование СP := С и МОP := МО КУ
5. Cчёт AH = BH + CP \* (MOP \* MTH)
6. Сигнал задаче №1 об окончании счёта S1.2

**№ Алгоритм задач №3 – (Р-1) ТС/КУ**

1. Ждать введения данных в задачах №1, №2 и №Р W1.1 W2.1 WP.1
2. Копирование Сi := С и МОi := МО КУ
3. Cчёт AH = BH + CP \* (MOP \* MTH)
4. Сигнал задаче №1 об окончании счёта S1.2

**Этап 3. Разработка схемы взаимодействия задач**



**Этап 4. Разработка программы.**

**Листинг программы**

/\*--------------------------------------------------------------

-- Paralel and distributed computing --

-- Laboratory work #6. OpenMP --

-- Func: A = B + C \* (MO \* MT) --

-- IO-83 Vorobyev Vitaliy --

-- 09.04.2011 --

--------------------------------------------------------------\*/

#include "stdafx.h"

#include <iostream>

#include "omp.h"

#include <windows.h>

using namespace std;

//Размер стека

#pragma comment(linker, "/stack:160000000")

const int N = 8;

const int P = 4;

const int H = N/P;

int \*A = new int[N];

int \*B = new int [N];

int \*C = new int [N];

int \*\*MO = new int\*[N];

int \*\*MT = new int\*[N];

void inMatrix(int\*\* matr) {

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

matr[i] = new int[N];

}

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

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

matr [i][j] = 1;

}

}

}

void inVector(int\* vec) {

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

vec [i] = 1;

}

}

void outVector(int\* vec) {

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

printf("%d ", vec [i]);

}

}

void cloneV(int\* V, int\* M) {

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

V [i] = M [i];

}

}

void cloneM(int \*\*MT, int \*\*MM) {

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

MT[i] = new int[N];

}

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

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

MT [i][j] = MM [i][j];

}

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

printf("Main thread: started\n");

omp\_lock\_t lck;

omp\_init\_lock(&lck);

omp\_set\_num\_threads(P);

//cin.get();

#pragma omp parallel

{

int num = omp\_get\_thread\_num();

printf("Thread %d: started\n", num);

//////////////////ВВОД ДАННЫХ////////////////////

switch (num) {

case 0:

printf("Thread %d: Input B, C\n", num);

inVector(B);

inVector(C);

break;

case 1:

printf("Thread %d: Input MO\n", num);

inMatrix(MO);

break;

case P-1:

printf("Thread %d: Input MT\n", num);

inMatrix(MT);

break;

}

//////////////////СИНХРОНИЗАЦИЯ ПО ВВОДУ////////////////////

#pragma omp barrier

int \*Cx = new int[N];

int \*\*MOx = new int\*[N];

//////////////////КРИТИЧЕСКИЙ УЧАСТОК////////////////////

omp\_set\_lock(&lck);

cloneV(Cx, C);

cloneM(MOx, MO);

omp\_unset\_lock(&lck);

//////////////////ВЫЧИСЛЕНИЕ ФУНКЦИИ////////////////////

printf("Thread %d: started calculating\n", num);

int sum;

int MF[N][N];

for (int z = num \* H; z < (num + 1) \* H; z++) {

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

sum = 0;

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

sum = sum + MOx[i][j] \* MT[j][z];

}

MF[i][z] = sum;

}

}

for (int z = num \* H; z < (num + 1) \* H; z++) {

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

sum = 0;

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

sum = sum + C[i] \* MF[j][z];

}

A[z] = sum;

}

}

for (int z = num \* H; z < (num + 1) \* H; z++) {

A[z] = A[z] + B[z];

}

printf("Thread %d: finished calculating\n", num);

//////////////////СИНХРОНИЗАЦИЯ ПО ВЫВОДУ////////////////////

#pragma omp barrier

//////////////////ВЫВОД РЕЗУЛЬТАТА////////////////////

if (num == 0) {

if (N <= 8) outVector(A);

}

printf("Thread %d: finished\n", num);

}

printf("Main thread: finished\n");

cin.get();

return 0;

}