

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №4**

“OpenMP. Барьеры, Критические секции”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

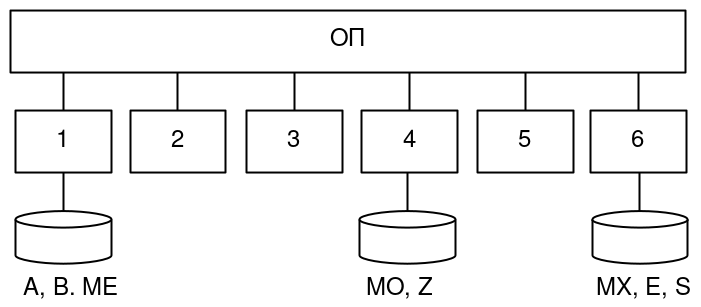
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

A = (B\*MO) \* (MX\*ME) + (Z\*E) \* S

Язык программирования:

C++

Библиотека параллельного программирования:

OpenMP

Средства взаимодействия задач:

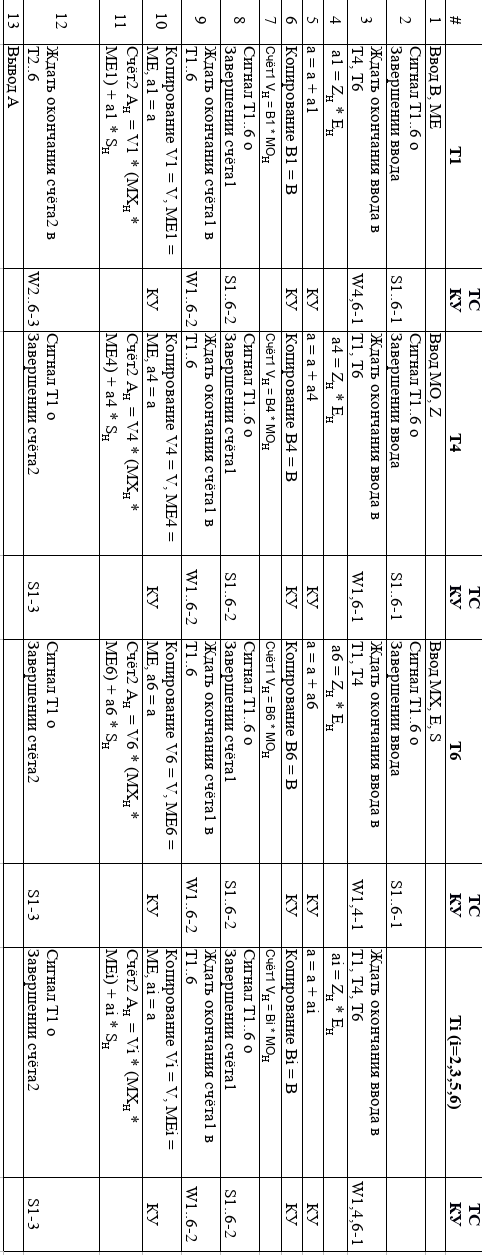
Барьеры, Критические Секции

**Выполнение работы**

**Этап 1:** Разработка параллельного математического алгоритма

1. ai = ZH \* EH , i=(1..P)
2. a = a + ai , i=(1..P)
   * OP: a
3. VH = B \* MOH
   * OP: B
4. AH  = V \* (MXH \* ME) + a \* SH
   * OP: V, ME, a

**Этап 2:** Разработка алгоритмов потоков



**Этап 3:** Разработка схемы взаимодействия потоков

**Этап 4:** Разработка программы

//-----------------------------------------------------------------------------

// Lab4: OpenMP. Barriers, Critical Sections, Parallel Loops

// Task: A = (B\*MO) \* (MX\*ME) + (Z\*E) \* S

// Author: Igor Boyarshin

// Date: 01.04.2018

//-----------------------------------------------------------------------------

#include <omp.h>

#include <iostream>

#include <sys/resource.h>

// Constants

static const unsigned int N = 300;

static const unsigned int P = 6;

static const unsigned int H = N / P;

static const unsigned int OUTPUT\_THRESHOLD = 8;

// Types

struct Vector {

private:

int elements[N];

public:

Vector() {}

Vector(int value) {

std::fill(elements, elements + N, value);

}

int& operator[](unsigned int row) {

return elements[row];

}

const int& operator[](unsigned int row) const {

return elements[row];

}

Vector copy() const {

Vector v;

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

v[j] = elements[j];

}

return v;

}

};

struct Matrix {

private:

Vector elements[N];

public:

Matrix() {}

Matrix(int value) {

for(unsigned int i = 0; i < N; i++) elements[i] = Vector(value);

}

Vector& operator[](unsigned int row) {

return elements[row];

}

const Vector& operator[](unsigned int row) const {

return elements[row];

}

Matrix copy() const {

Matrix m;

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

const Vector& v = elements[i];

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

m[i][j] = v[j];

}

}

return m;

}

};

// Variables

int a;

Vector A, B, Z, E, S, V;

Matrix ME, MO, MX;

// Functions

void fillVector(Vector& vector, int value);

void fillMatrix(Matrix& matrix, int value);

void outputVector(const Vector& vector);

void outputMatrix(const Matrix& matrix);

//-----------------------------------------------------------------------------

void threadFunction (unsigned int tid) {

#pragma omp critical(info\_output)

std::cout << ":> Thread " << tid << " started..." << std::endl;

const unsigned int low = (tid - 1) \* H;

const unsigned int high = tid \* H;

// Input

switch (tid) {

case 1:

fillVector(B, 1);

fillMatrix(ME, 1);

break;

case 4:

fillVector(Z, 1);

fillMatrix(MO, 1);

break;

case 6:

fillVector(E, 1);

fillVector(S, 1);

fillMatrix(MX, 1);

break;

default:

break;

}

// Input Barrier

#pragma omp barrier

int ai = 0;

#pragma omp parallel for

for (unsigned int i = low; i < high; i++) {

ai += Z[i] \* E[i];

}

#pragma omp critical(update\_a)

{

a = a + ai;

}

Vector Bi;

#pragma omp critical(copy\_B)

{

Bi = B.copy();

}

#pragma omp parallel for

for (unsigned int i = low; i < high; i++) {

const Vector& vector = MO[i];

int elem = 0;

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

elem += Bi[j] \* vector[j];

}

V[i] = elem;

}

// V(and a) barrier

#pragma omp barrier

// Copies

Vector Vi;

#pragma omp critical(copy\_V)

{

Vi = V.copy();

}

Matrix MEi;

#pragma omp critical(copy\_ME)

{

MEi = ME.copy();

}

#pragma omp critical(copy\_a)

{

ai = a;

}

#pragma omp parallel for

for (unsigned int h = low; h < high; h++) {

Vector temp;

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

int sum = 0;

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

sum += MX[h][j] \* MEi[j][i];

}

temp[i] = sum;

}

int elem = 0;

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

elem += Vi[i] \* temp[i];

}

A[h] = elem + ai \* S[h];

}

// Output barrier

#pragma omp barrier

// Output

if (N < OUTPUT\_THRESHOLD) {

switch (tid) {

case 1:

outputVector(A);

break;

default:

break;

}

}

#pragma omp critical(info\_output)

std::cout << ":> Thread " << tid << " finished..." << std::endl;

}

//-----------------------------------------------------------------------------

int main() {

// Preparations

omp\_set\_num\_threads(P);

a = 0;

#pragma omp parallel

{

const unsigned int tid = omp\_get\_thread\_num() + 1;

threadFunction(tid);

}

return 0;

}

//-----------------------------------------------------------------------------

// Function definitions

void fillVector(Vector& vector, int value) {

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

vector[i] = value;

}

void fillMatrix(Matrix& matrix, int value) {

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

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

matrix[i][j] = value;

}

void outputVector(const Vector& vector) {

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

#pragma omp critical(info\_output)

std::cout << vector[j] << " ";

}

#pragma omp critical(info\_output)

std::cout << std::endl;

}

void outputMatrix(const Matrix& matrix) {

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

const Vector& v = matrix[i];

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

std::cout << v[i] << " ";

}

std::cout << std::endl;

}

}