**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

Разработка многопоточных приложений с использованием OpenMP. Вариант 2

**Исполнитель**

Студентка группы БПИ195

Баранова Е. В.

Задание

Найти определитель матрицы А. Входные данные: целое положительное число n, произвольная матрица А размерности n х n. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

Составление​​ программы

Программа считаем определитель матрицы через перестановки с помощью формулы [1].

Исходный код программы документирован.

В программе выделено две функции:

int inversion\_rate(const int \*p)

Метод для подсчета количества инверсий в перестановке.

p – перестановка.

void thread\_task(const int \*p)

Метод для передачи потокам. В методе считается определитель матрицы по слагаемым.

p – перестановка

int factorial(int i)

Метод для подсчета количества перестановок.

i – длина перестановки.

Источники

[1] - http://www.cleverstudents.ru/matrix/computation\_of\_determinant.html

Текст программы:

// Вариант 2, Баранова Екатерина

#include <iostream>

#include <vector>

#include <algorithm>

#include <omp.h>

using namespace std;

vector<vector<int> > matrix; // Матрица

vector<int> permutation; // Перестановка

int n, threads\_count; // Размер матрицы и количество потоков

long long determinant; // Определитель матрицы - результат

// Метод для подсчета количества инверсий в перестановке

int inversion\_rate(const int\* p) {

int ans = 0;

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

for (int j = i + 1; j < n; ++j)

if (p[i] > p[j])

++ans;

return ans;

}

// Метод для потока

void thread\_task(const int\* p) {

// Считаем очередное слагаемое в определителе

int inversions = inversion\_rate(p);

int ans = 1;

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

ans \*= matrix[i][p[i]];

determinant += inversions % 2 ? -ans : ans;

}

int factorial(int i) {

int ans = 1;

for (int j = 2; j <= i; ++j)

ans \*= j;

return ans;

}

int main() {

setlocale(0, "ru\_RU");

// Считывем все необходимое

while (n <= 0 || n > 10) {

cout << "Введите размер матрицы (от 1 до 10): ";

cin >> n;

}

while (threads\_count <= 0 || threads\_count > 100) {

cout << "Введите количество потоков (от 1 до 100): ";

cin >> threads\_count;

}

cout << "Значения элементов матрицы должны лежать в пределе от -9 до 9!" << endl;

matrix.resize(n);

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

matrix[i].resize(n);

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

do {

cout << "A[" << i + 1 << "][" << j + 1 << "] = ";

cin >> matrix[i][j];

} while (matrix[i][j] < -9 || matrix[i][j] > 9);

}

}

// Выводим матрицу

cout << "Ваша матрица А:" << endl << endl;

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

cout << "( ";

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

cout << (matrix[i][j] < 0 ? "" : " ") << matrix[i][j] << " ";

cout << ")" << endl;

}

// Инициализируем глобальную перестановку для потоков

permutation.resize(n);

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

permutation[i] = i;

int i, x = factorial(n);

int\* nums = new int[n];

#pragma omp parallel for private(i, nums) shared(permutation) schedule(dynamic) num\_threads(threads\_count)

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

#pragma omp critical

{

nums = new int[n];

copy(permutation.begin(), permutation.end(), nums);

next\_permutation(permutation.begin(), permutation.end());

}

thread\_task(nums);

}

delete[] nums;

// Выводим определитель

cout << endl << "Опредеделитель матрицы |A| = " <<

determinant;

return 0;

}