Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

**(ПНИПУ)**

Факультет: Электротехнический

Кафедра: Информационные технологии и автоматизированные системы.

# Направление подготовки: Разработка программно-информационных систем.

**Лабораторная работа № 3**

Технология Intel Threading Building Blocks. Решение СЛАУ методом Гаусса.

Выполнил студент гр. РИС-18-1б

Гилязов Р.А.

(Фамилия, имя, отчество)

###### Проверил:

Старший преподаватель кафедры ИТАС, Щапов В.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О. руководителя от кафедры)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**Пермь, 2020**

**Постановка задачи:**

1. Написать программу, которая будет решать СЛАУ методом Гаусса.
2. Используя технологию TBB, распараллелить решение.

**Решение**

#include <iostream>

#include "tbb/task\_scheduler\_init.h"

#include "tbb/parallel\_for.h"

#include "tbb/blocked\_range2d.h"

#include "tbb/tick\_count.h"

#include <chrono>

using namespace tbb;

using namespace std;

class gauss\_method

{

public:

double\*\* arr;

int i;

const int rows = 1000, cols = rows + 1;

gauss\_method() // инициализирует матрицу размером 500:501

{

arr = new double\* [rows];

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

{

arr[i] = new double[cols];

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

arr[i][j] = double(rand() % 1000) / 100;

}

}

void operator() (const blocked\_range<int>& r) const // параллельная часть алгоритма

{

for (int j = r.begin(); j < r.end(); j++)

{

double t = arr[j][i] / arr[i][i];

for (int k = i; k < cols; k++)

arr[j][k] -= t \* arr[i][k];

}

}

int copy(gauss\_method d) // копирует матрицу

{

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

{

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

arr[i][j] = d.arr[i][j];

}

return 0;

}

double\* solve() // решение приведенной к треугольному виду

{

double\* sol = new double[rows];

for (int i = rows - 1; i >= 0; i--)

{

double sum = arr[i][cols - 1];

for (int j = 1; j < rows - i; j++)

sum -= arr[i][cols - 1 - j] \* sol[rows - j];

sol[i] = sum / arr[i][i];

}

return sol;

}

void print\_r(double\* res) {

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

std::cout << "x[" << i << "] = " << res[i];

}

}

void solve\_without\_parallel() // гаусс без паралельности

{

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

{

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

{

double t = arr[j][i] / arr[i][i];

for (int k = i; k < cols; k++)

arr[j][k] -= t \* arr[i][k];

}

}

double \* res = solve();

//print\_r(res);

delete res;

}

};

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

{

const int arr[10] = { 1, 3, 6, 9, 12, 100, 150, 300, 500, 1000 }; // массив масштабирования

const int k = 1000;

tbb::task\_scheduler\_init init;

gauss\_method s[11];

// создание одинаковых матриц

for (int i = 1; i < 11; ++i) {

s[i].copy(s[0]);

}

int row = s[0].rows;

for (int p = 0; p < 10; p++)

{

auto beg = chrono::high\_resolution\_clock::now();

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

{

s[p].i = j;

parallel\_for(blocked\_range<int>(j + 1, row, arr[p]), s[p]);

s[p].solve();

}

auto end = chrono::high\_resolution\_clock::now();

auto time = chrono::duration\_cast<chrono::microseconds>(end - beg).count();

cout << "Runtime parallel on " << arr[p] << " grain: " << time << "mcs\n";

}

auto beg = chrono::high\_resolution\_clock::now();

s[10].solve\_without\_parallel();

auto end = chrono::high\_resolution\_clock::now();

auto time = chrono::duration\_cast<chrono::microseconds>(end - beg).count();

cout << "Runtime parallel without parallel: " << time << "mcs\n";

return 0;

}

**Тесты на результат**

Конфигурация используемого аппаратного обеспечения:

- процессор: AMD Ryzen 2600, 6 core, L3 cache 16 MB

Конфигурация программы: Release x64

Таблица 1: Измерение времени при различных размерах матрицы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер матрицы | Без параллельности, мкс | 3 ядра, мкс | 6 ядер, мкс | 9 ядер, мкс |
| 100х101 | 122 | 1218 | 1294 | 1172 |
| 500х501 | 13315 | 82212 | 85990 | 87155 |
| 1000х1001 | 116775 | 654185 | 652350 | 562871 |

Исходя из данных замеров по времени причины таких плохих показателей может быть объяснено различиями между архитектурой процессоров Intel и AMD. Другим же неэффективный способ решения СЛАУ, так как при решении распараллеливается всего одна часть и затраты на разбиение на блоки и имеют больше затрат, чем пользы. Тем не менее, результат получился следующим: алгоритм решения работает эффективнее без распараллеливания.