Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

 «Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Технологии блокчейн и распределенные информационные системы»

Профиль: «Разработка программно-информационных систем»

ОТЧЕТ

по лабораторной работе №6

Тема: «Решение СЛАУ методом Гаусса с использованием oneTBB»

Выполнил: студент группы РИС-20-2б

Катаев М.М.   \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель кафедры ИТАС

Щапов В. А. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2024

**Цели и задачи**

1. Реализовать программу для решения СЛАУ с использованием OneTbb

2. Проанализировать результаты

3. Сделать выводы

**Выполнение работы**

OneAPI Threading Building Blocks (oneTBB) — это основанная на среде выполнения модель параллельного программирования для кода C++, в котором используются потоки. Это библиотека времени выполнения на основе шаблонов, предназначенная для использования скрытой производительности многоядерных процессоров. oneTBB упрощает параллельное программирование, разбивая вычисления на параллельно выполняющиеся задачи. Параллелизм осуществляется в рамках одного процесса с помощью потоков — механизма операционной системы, позволяющего одновременно выполнять одни и те же или разные наборы инструкций.

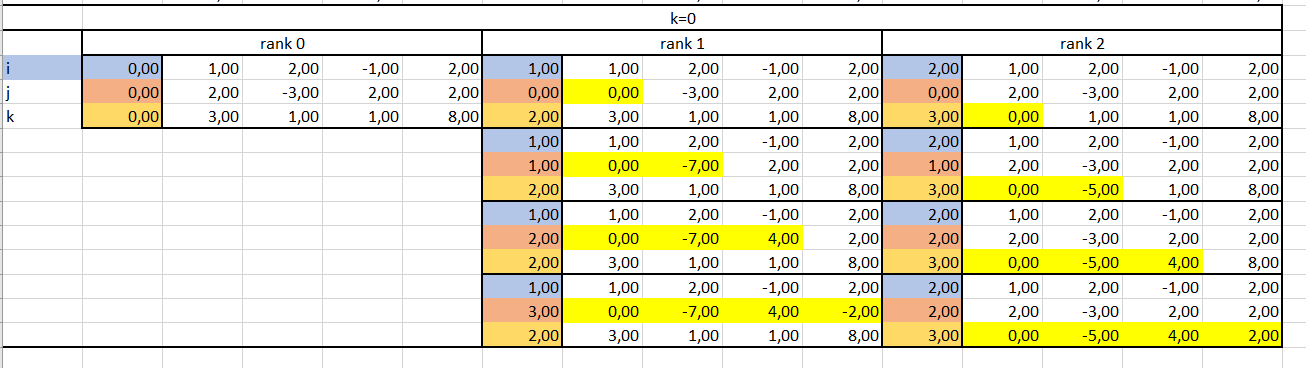
В отличии от MPI использование OneTbb позволяет с помощью встроенного функционала распределять прохождение по циклу с помощью одной функции - parallel\_for(). Данная функция принимает 3 агрумента: начало, конец и переменную цикла, при этом распределение задач по потокам происходит автоматически. Разделение по строкам, аналогично лабораторной работе 5 и продемонстрированно на рисунке ниже. 

Рисунок 1 – Параллельное выполнение прямого хода

Жёлтым цветом выделены данные, которые изменились в каждом процессе, в данном случае было 3 процесса, то есть каждый процесс обрабатывал по одной строке.

После прохождения итерации по k необходимо синхронизировать данные, чтобы каждый процесс получил обновленные данные, именно поэтому мы распараллеливаем только внутренний цикл.

Обратный ход реализован с такой же логикой: каждый процесс изменяет данные по своим строкам, собираются все строки в буфер, значения из которого заносятся в матрицу A корневого процесса, который после этого рассылает эту матрицу всем остальным процессам.

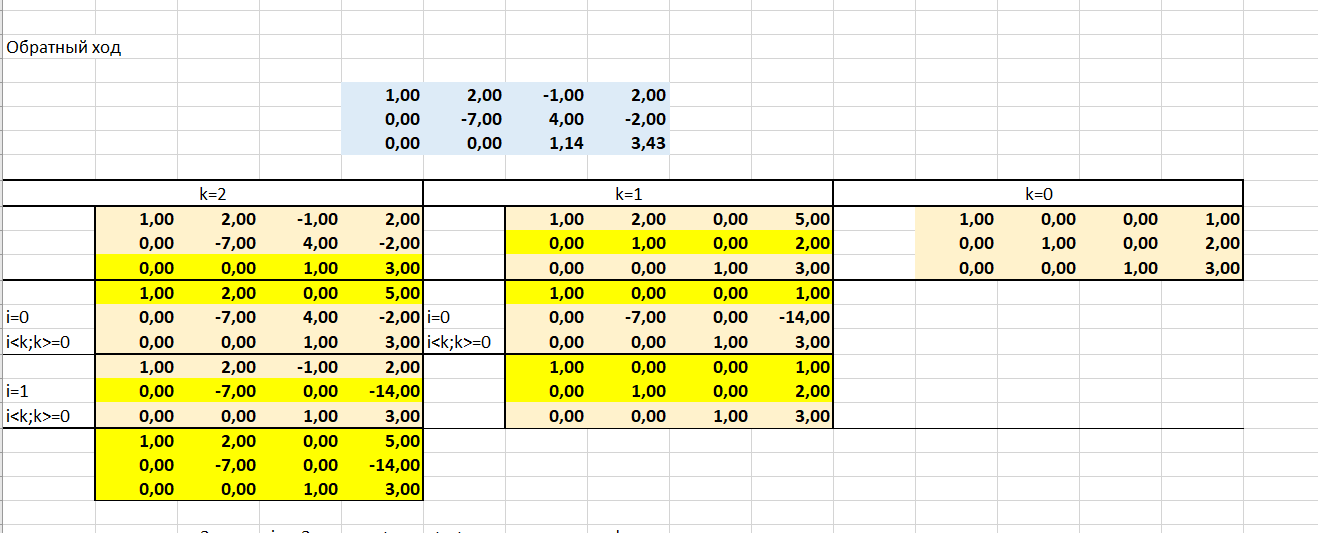


Рисунок 2 – Параллельное выполнение обратного хода

Ниже приведет полный код программы.

#include "iostream"

#include <tbb/tbb.h>

using namespace std;

const int N = 1500;

double B[N][N + 1];

double A[N][N + 1];

void gauss(double A[N][N + 1]) {

// Прямой ход

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

for (int i = k + 1; i < N; i++) {

double koef = A[i][k] / A[k][k];

for (int j = k; j < N + 1; j++) {

A[i][j] -= koef \* A[k][j];

}

}

}

// Обратный ход

for (int k = N - 1; k >= 0; k--) {

A[k][N] /= A[k][k];

A[k][k] = 1.0;

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

A[i][N] -= A[i][k] \* A[k][N];

A[i][k] = 0.0;

}

}

for (int i = 0; i < 10; ++i) {

cout << "x[" << i << "] = " << A[i][N] << endl;

}

}

void gaussTbb(double A[N][N + 1]) {

// Прямой ход

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

tbb::parallel\_for(k+ 1, N, [&](int i) {

double koef = A[i][k] / A[k][k];

for (int j = k; j < N + 1; j++) {

A[i][j] -= koef \* A[k][j];

}

//cout << "Прямой ход Поток " << tbb::this\_task\_arena::current\_thread\_index() << " обработал строку " << i << " k= "<<k<<endl;

});

}

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

// for (int j = 0; j < N+1; ++j) {

// std::cout << A[i][j] << "\t";

// }

// std::cout << std::endl;

// }

// Обратный ход

for (int k = N - 1; k >= 0; k--) {

A[k][N] /= A[k][k];

A[k][k] = 1.0;

tbb::parallel\_for(0, k, [&](int i) {

A[i][N] -= A[i][k] \* A[k][N];

A[i][k] = 0.0;

//cout << "Обратный ход Поток " << tbb::this\_task\_arena::current\_thread\_index() << " обработал строку " << i << " k= " << k<< endl;

});

}

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

for (int j = 0; j < N + 1; ++j) {

std::cout << A[i][j] << "\t";

}

std::cout << std::endl;

}\*/

for (int i = 0; i < 10; i++) {

std::cout << "x[" << i << "] = " << A[i][N] << std::endl;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

//double a[N][N + 1] = { {1, 1, 2, 3, 1}, { 1, 2, 3, -1,-4 }, { 3, -1, -1, -2,-4 }, { 2, 3, -1, -1,-6 } };

A[N][N + 1] = { {1, 2, -1, 2}, {2, -3, 2, 2}, {3, 1, 1, 8} };

B[N][N + 1] = { {1, 2, -1, 2}, {2, -3, 2, 2}, {3, 1, 1, 8} };

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

for (int j = 0; j < N + 1; ++j) {

A[i][j] = rand() % 10 + 1;

B[i][j] = A[i][j];

}

}

tbb::tick\_count startTbb = tbb::tick\_count::now(); // Засекаем начало времени выполнения параллельной функции

gaussTbb(A);

tbb::tick\_count endTbb = tbb::tick\_count::now(); // Засекаем конец времени выполнения параллельной функции

cout << "OneTbb Затрачено времени: " << (endTbb - startTbb ).seconds() << " сек." << endl;

tbb::tick\_count start = tbb::tick\_count::now(); // Засекаем начало времени

gauss(B);

tbb::tick\_count end= tbb::tick\_count::now(); // Засекаем конец времени

cout << "Затрачено времени: " << ( end- start).seconds() << " сек." << endl; // Выводим время выполнения

return 0;

}

Листинг 1 – Код программы

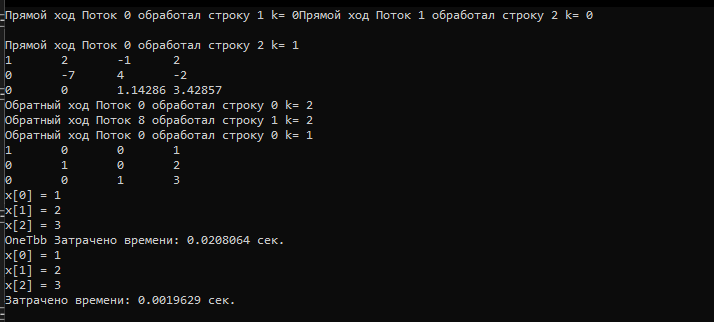
Ниже показаны результаты выполнения программы. 

Рисунок 3 – Результат выполнения программы(N=3)

На рисунке 3 проиллюстрировано, как делится прохождение цикла между потоками. Номер потока был получен при помощи встроенной функции OneTbb **tbb::this\_task\_arena::current\_thread\_index()**. Время выполнения было посчитано также с помощью **tbb::tick\_count::now()**. При этом время выполнения при параллельном выполнении превышает последовательное, вероятно, это происходит из-за маленького размера матрицы.

Проведем еще один тест и увеличим размер матрицы до N=1500. Для наглядного вывода уберем вывод потока и итерации из программы, а также изменим функцию вывода результатов – в цикле N заменим на 10, для того чтобы сравнить правильность вычислений с последовательным выполнением.

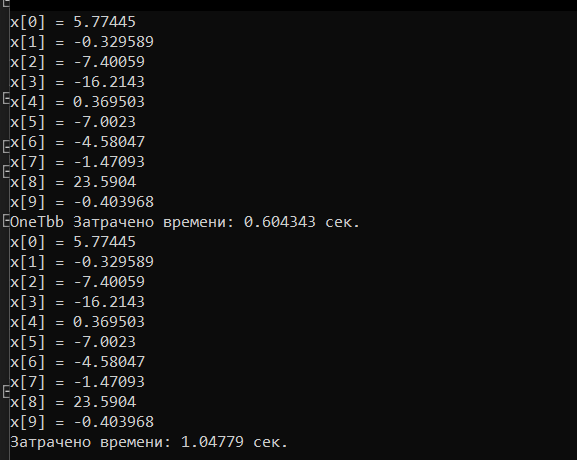


Рисунок 4 – Результат выполнения программы(N=1500)

На рисунке 4 видно, что благодаря использованию параллельных вычислений в методе Гаусса нам удалось сократить время выполнения программы почти в 2 раза при размере матрицы N=1500.

Таким образом, использование метода Гаусса в комбинации с библиотекой oneTbb демонстрирует повышение эффективности и ускорение выполнения решения систем линейных уравнений, особенно в случае больших размерностей матриц.