МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ



**ОТЧЕТ**

**по лабораторной работе**

|  |  |
| --- | --- |
| *Дисциплина* | Теория языков программирования и вычислительных процессов (ТЯПиВП) |
| *Номер работы* | 1 |
| *Тема работы* | Исследование влияния псевдопараллелизма на производительность |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Принял:** | |  | **Выполнили:** | | | |
|  | В.В.Хашковский, доцент каф. МОП ЭВМ |  |  | *Фамилия И.О.* | *Группа* | *Оценка* |
|  | Брикалов В.А. | гр. КТбо3-1 |  |
|  | Зинченко А.С. | гр. КТбо3-1 |  |

Таганрог 2020

# **1 Цель работы**

* 1. **Дидактическая цель работы**

В ходе выполнения данной работы изучить механизм псевдопараллелизма и его влияние на производительность.

**1.2 Практическая цель работы**

Научиться использовать на практике методы управления потоками в языке C++.

**2. Вариант задания:**

Изучить влияние увеличения количества потоков на скорость перемножения матриц.

**3. Ход работы**

**3.1 Теоретическое описание проблемы**

Когда мы используем компьютеры, создается ощущение, что они могут делать сразу несколько дел одновременно. На самом деле процессор переключается между программами, предоставляя каждой от десятков до сотен миллисекунд – кванты времени. Это явление называется псевдопараллелизм.

Для распараллеливания разных подзадач одной программы могут использоваться процессы или потоки. Каждый процесс имеет свое адресное пространство, в то время как потоки внутри одного процесса разделяют между собой адресное пространство и, например, один поток может поменять значение переменной, используемой другим потоком. Следовательно полученные данные могут быть не корректны или же мы можем получить ошибку. К примеру, выйти за границу массива.

Чтобы этого не произошло используются различные механизмы синхронизации. В данной работе использованы мьютексы (от англ. **MUT**ual **EX**clusion). Мьютекс — переменная, которая может находиться в одном из двух состояний: блокированном или неблокированном.

В данной работе мьютекс блокируется в том месте кода, где происходит изменение общих данных.

Для того, чтобы обеспечить одновременный старт потоков используется блокирующий примитив *condition\_variable.*

**3.2 Описание алгоритма перемножения матриц**.

|  |  |
| --- | --- |
| 1 | N = размерность матрицы |
| 2 | i=0 |
| 3 | j=0 |
| 4 | k=0 |
| 5 | While(i>N) |
| 6 | mutex.lock() //блокируется доступ других потоков к общим переменным |
| 7 | If (i>=N) then end while |
| 8 | mat3[i][j]=mat1[i][k]\*mat2[k][j] |
| 9 | k=k+1 |
| 10 | If (k>=N) then |
| 11 | k=0 |
| 12 | j=j+1 |
| 13 | If (j>=N) then |
| 14 | j=0 |
| 15 | i=i+1 |
| 16 | mutex.unlock() |
| 17 | End While |

**3.3** Для написания программы была использована платформа Visual Studio и язык C++. Для разработки программы были использованы модули: <chrono>, <thread>, <mutex>, <condition\_variable>.

**3.4 Полученные результаты**:

Начнем выполнение с одного потока и постепенно увеличивая дойдем до 1000 потоков, чтобы каждый поток выполнил по одному вычислению (рис.1-11).

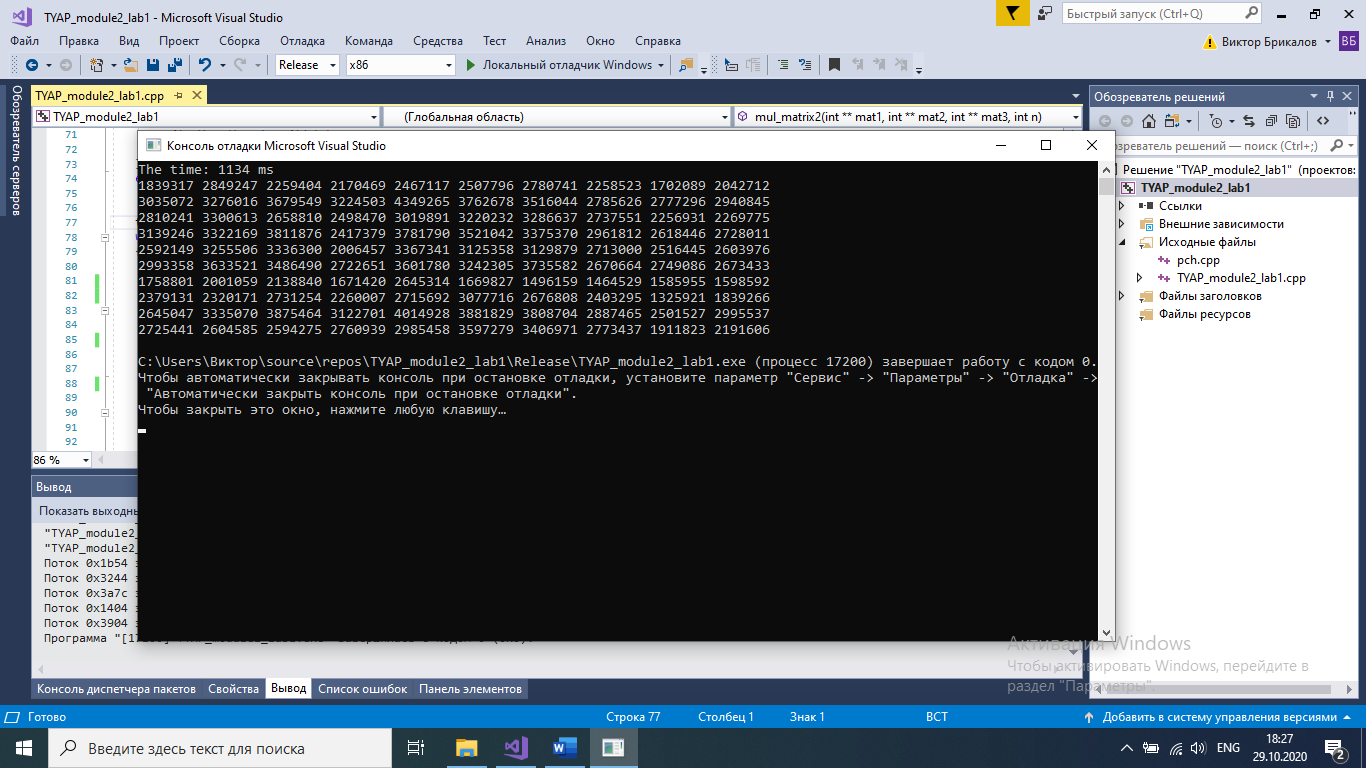


Рисунок 1 - 1 поток

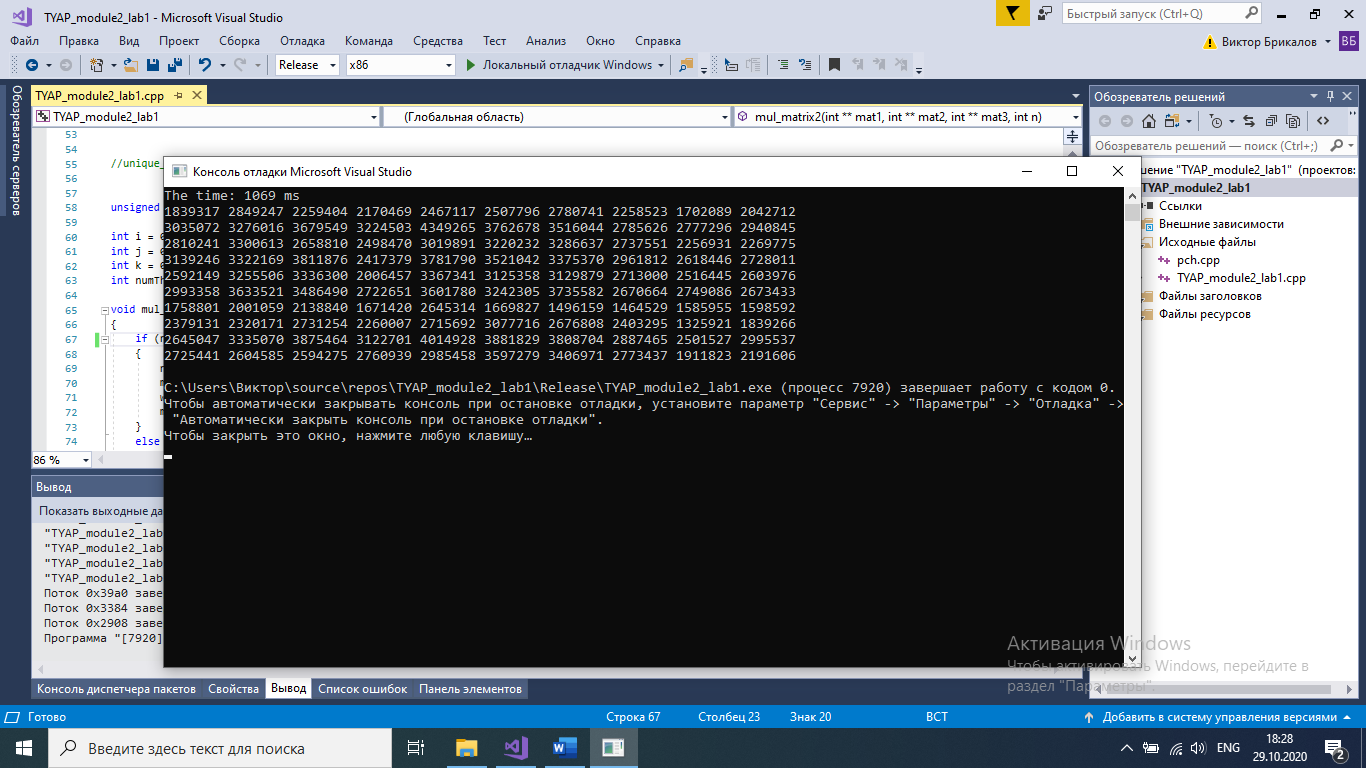


Рисунок 2 - 2 потока

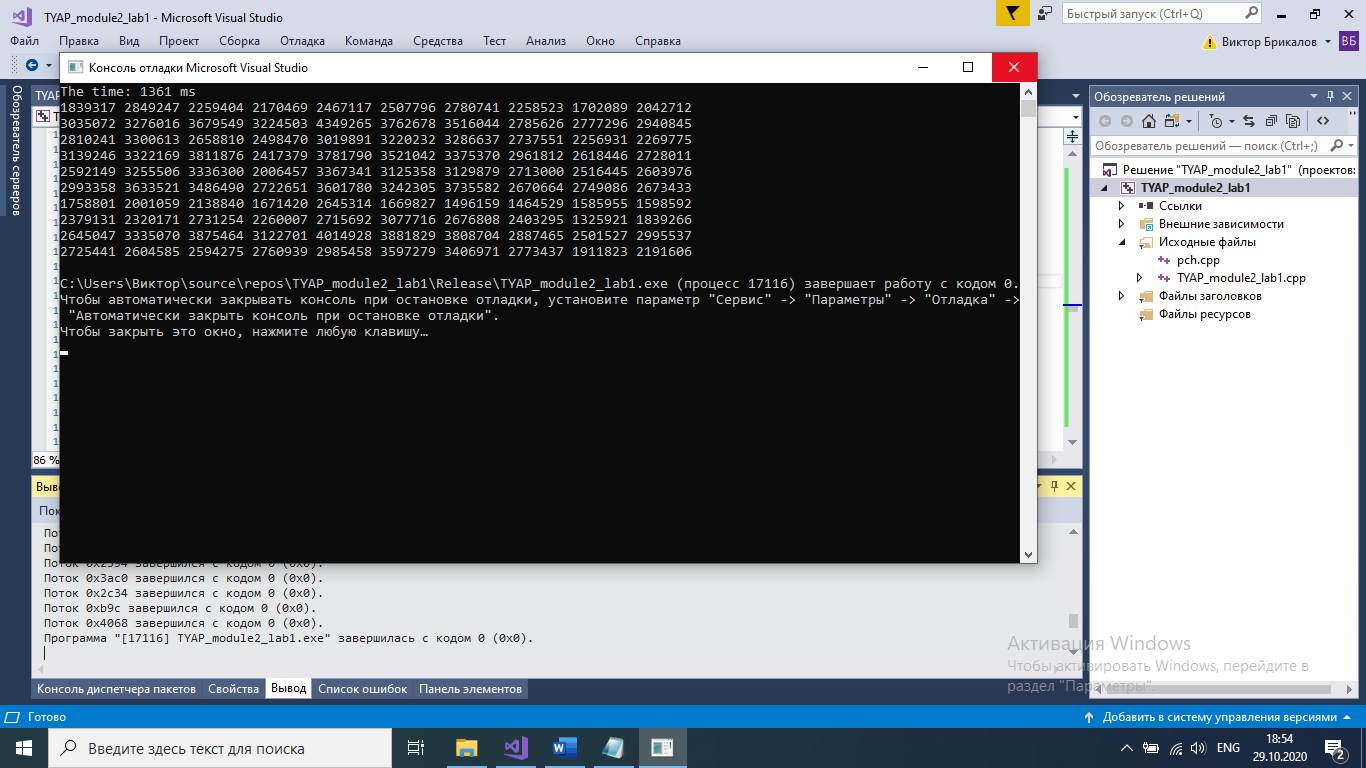


Рисунок 3 - 4 потока

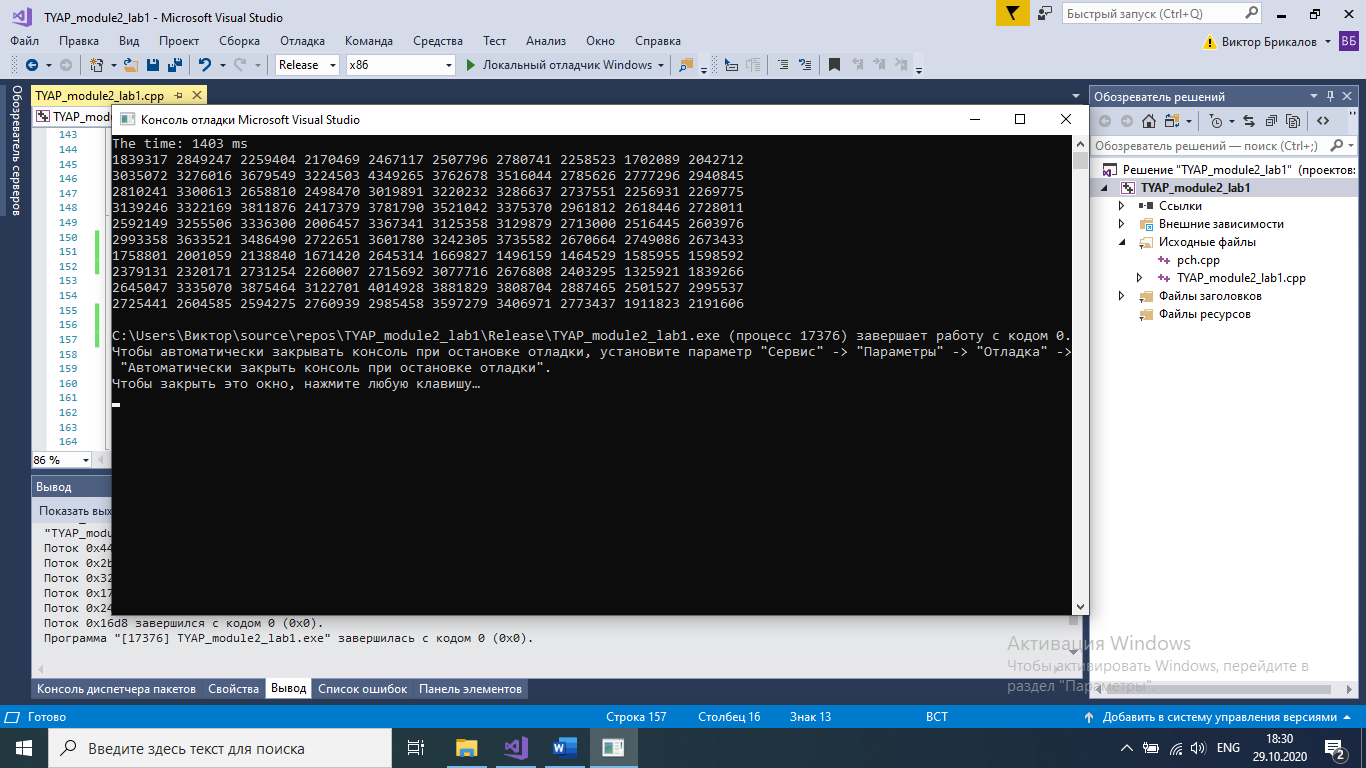


Рисунок 4 - 8 потоков

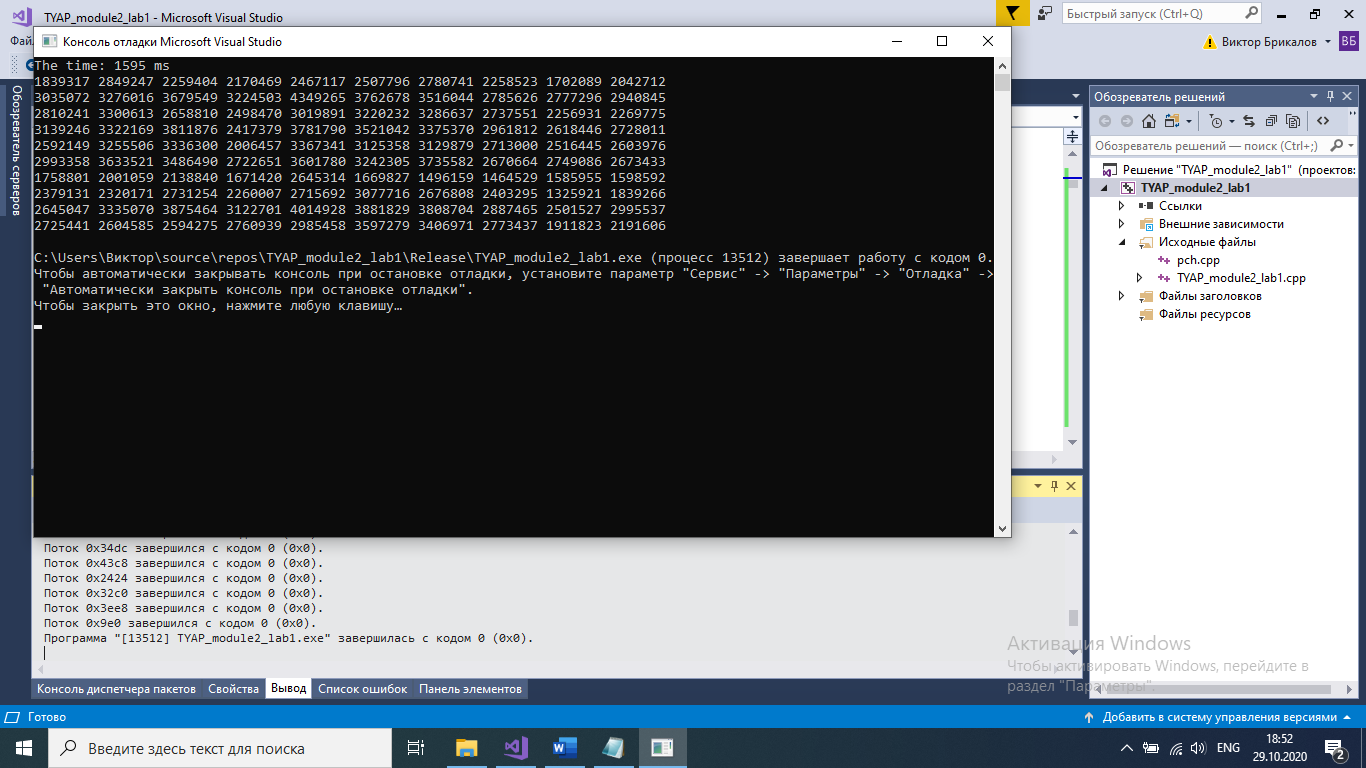


Рисунок 5 - 16 потоков

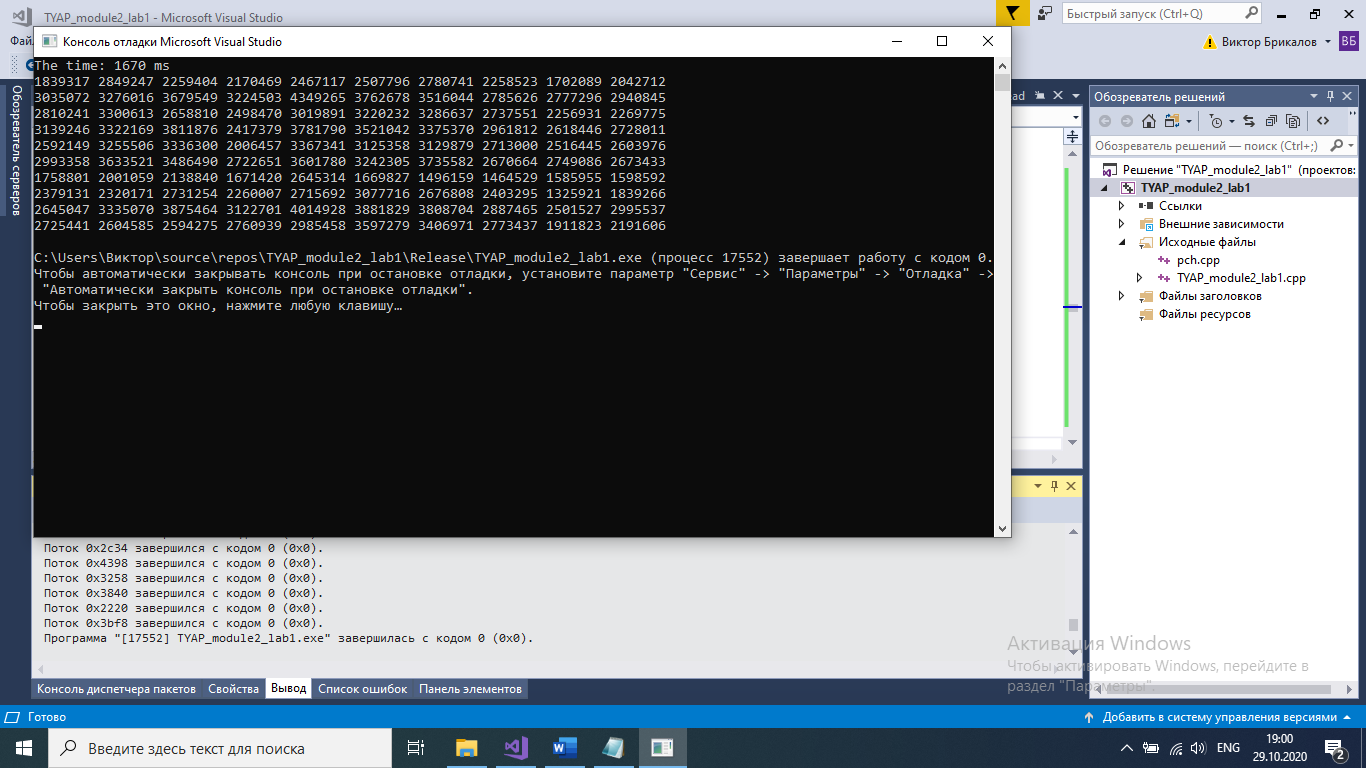


Рисунок 6 - 32 потока

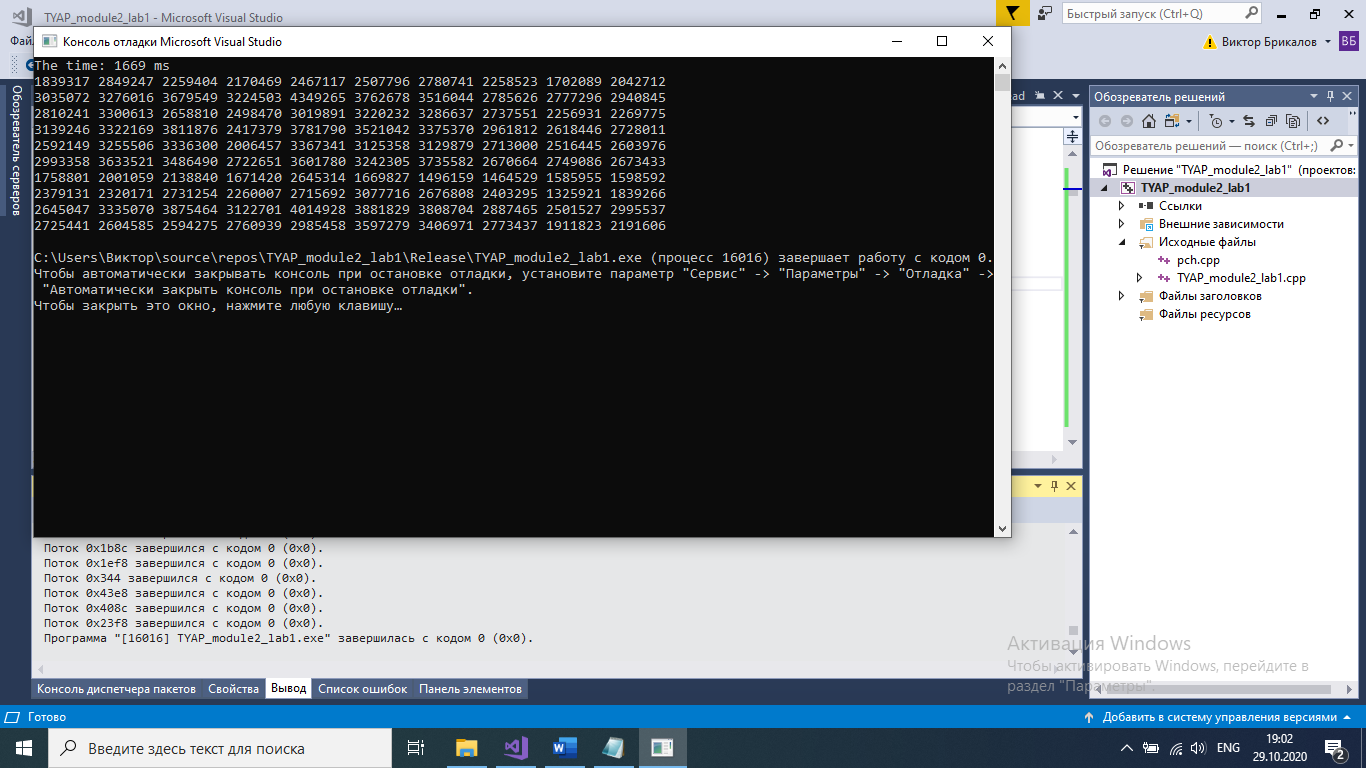


Рисунок 7 - 64 потока

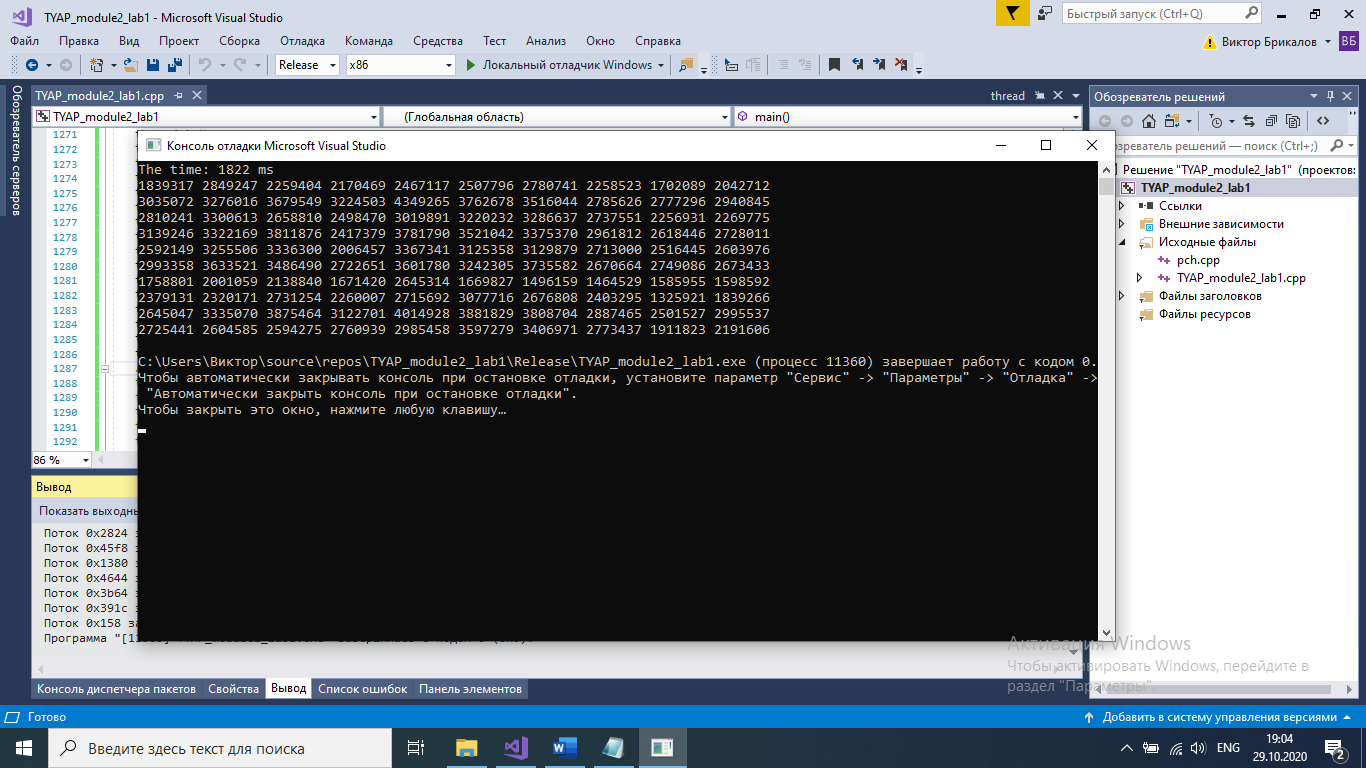


Рисунок 8 - 128 потоков

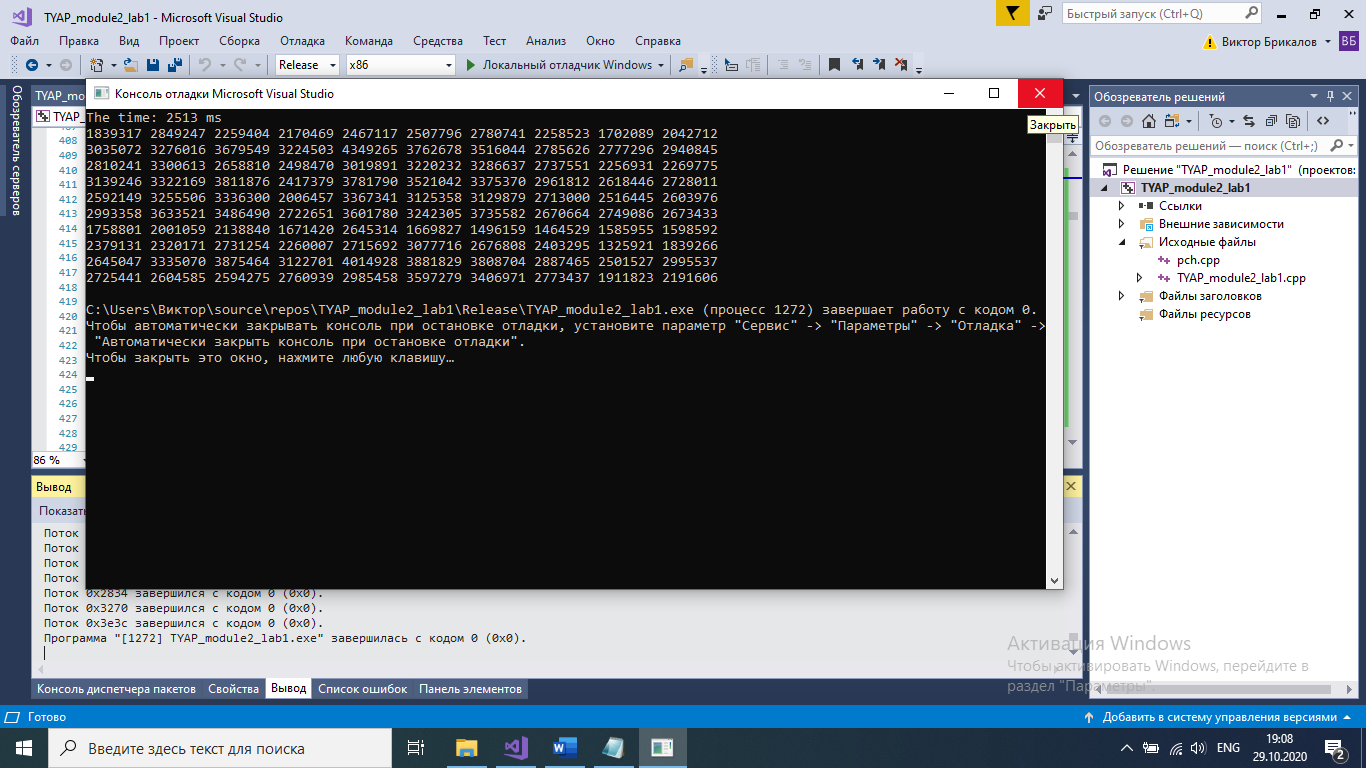


Рисунок 9 – 256 потоков

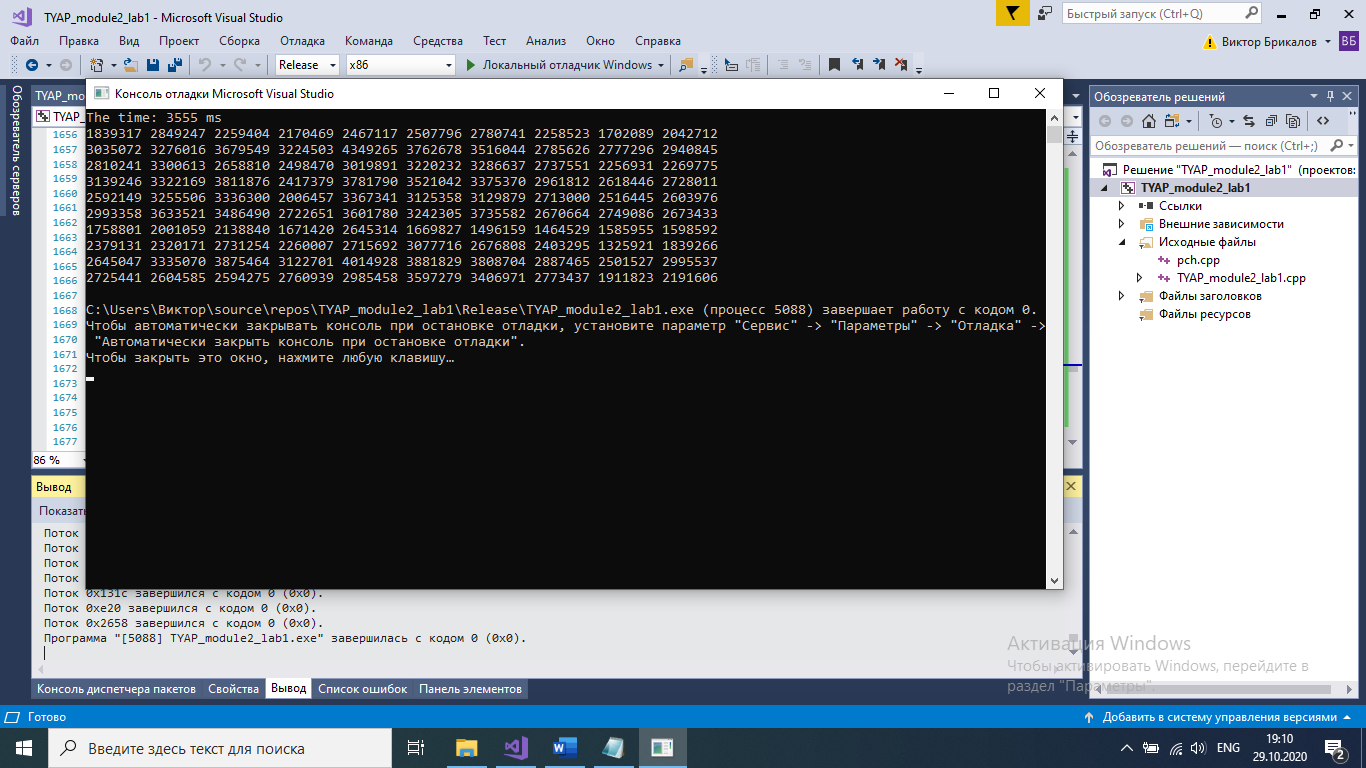


Рисунок 10 - 512 потоков

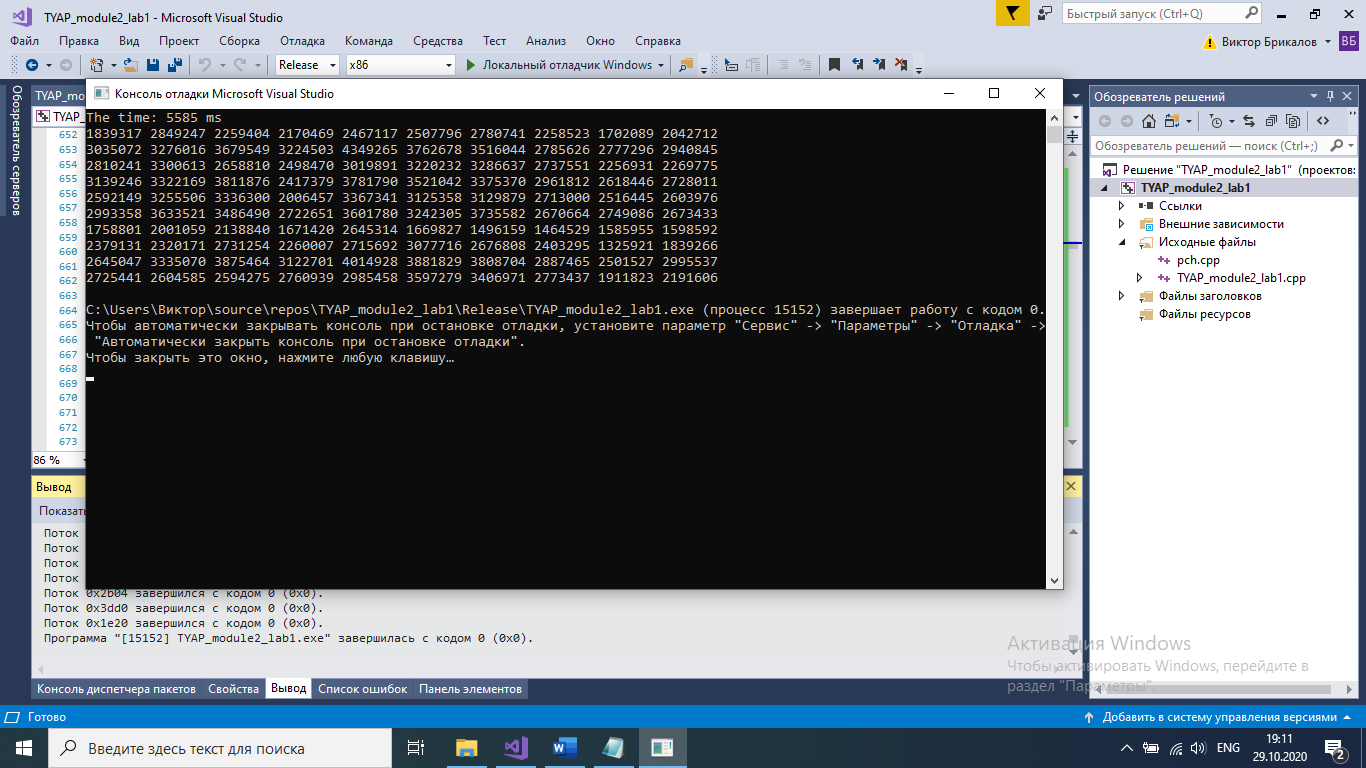


Рисунок 11 - 1000 потоков

Для наглядности построим график по полученным точкам (рис.12). По оси абсцисс количество потоков, по оси ординат время выполнения в милисекундах.

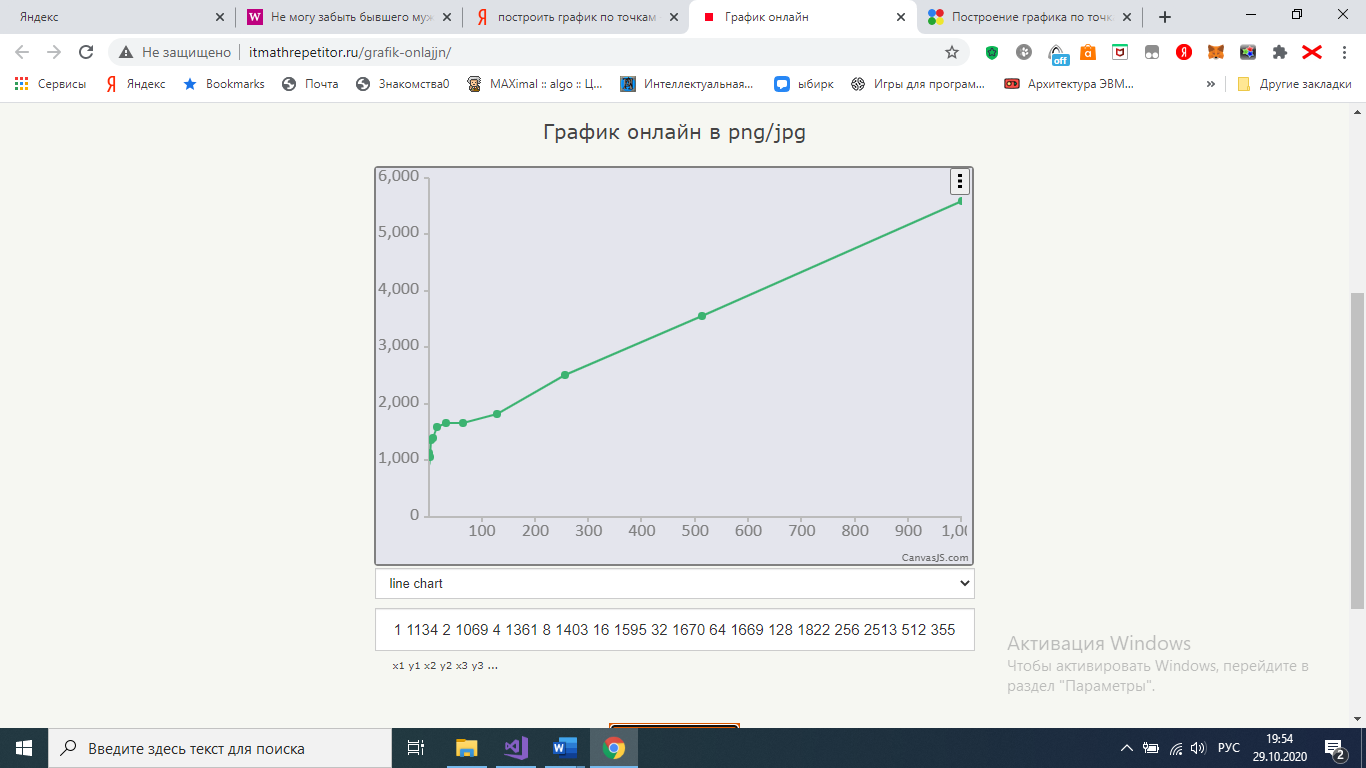


Рисунок 12

Увеличим участок до 128 потоков (рис. 13).

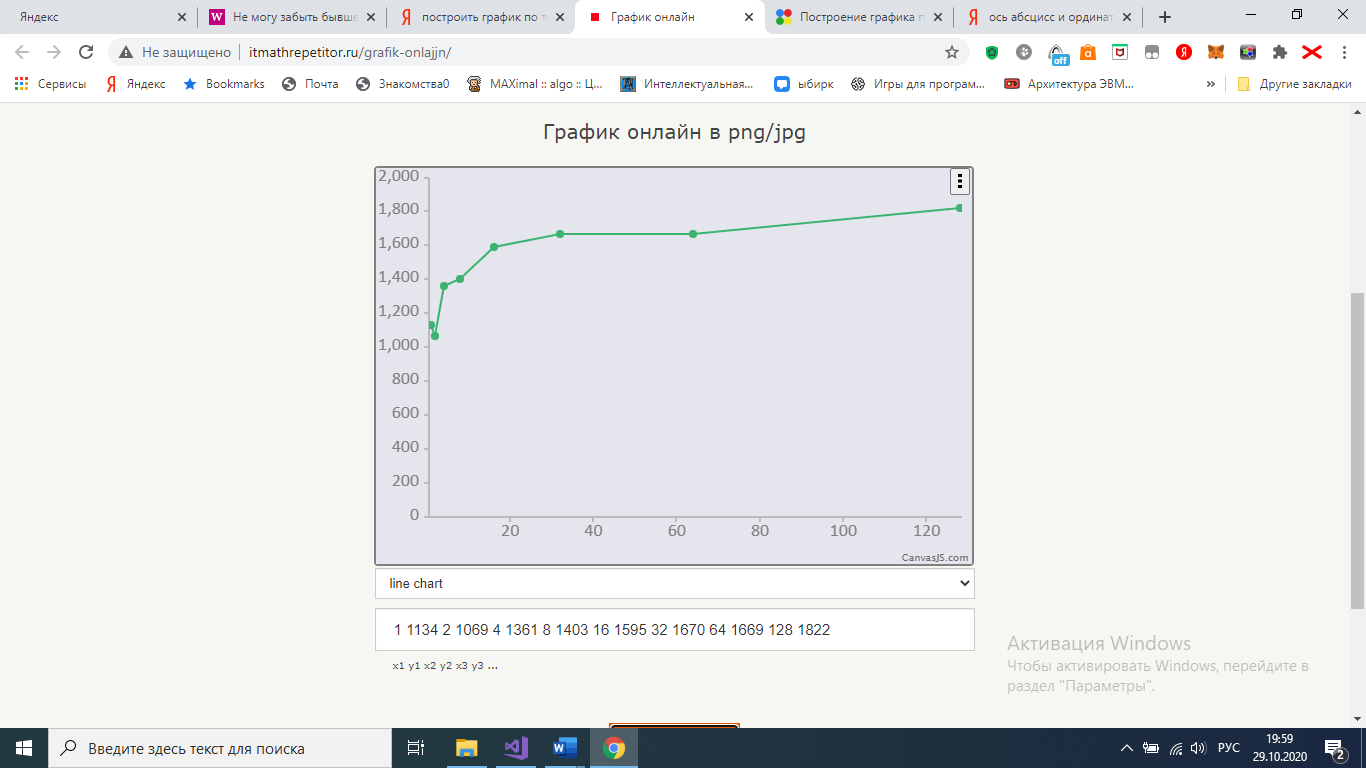


Рисунок 13

Из полученных результатов видно, что при 2 потоках программа отработала немного быстрей, чем при одном. При дальнейшем увеличении количества потоков время выполнения программы постепенно увеличивалось. После увеличения количества потоков до 100 скорость замедления программы значительно увеличилась.

Результаты можно объяснить тем, что для выполнения программы использовался двухъядерный процессор и выполнение двух потоков происходит действительно параллельно. Дальнейшее же увеличение потоков ведет только к росту накладных расходов на управление и синхронизацию потоков.

**4. Вывод**

В ходе выполнения лабораторной работыбыли изучены процесс псевдопараллелизма и синхронизация потоков. Была написана программа, позволяющая оценить влияние псевдопараллелизма на производительность.

**5. Приложения**

Листинг:

#include <iostream>

#include <fstream>

#include <chrono>

#include <vector>

#include <thread>

#include <mutex>

#include <fstream>

#include <string>

#include <condition\_variable>

using namespace std;

mutex mtx;

void new\_matrix\_null(int\*\* A, int n, int m)

{

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

{

A[i] = new int[n];

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

{

A[i][j] = 0;

}

}

}

void new\_matrix\_random(int\*\* A, int n, int m)

{

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

{

A[i] = new int[n];

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

{

A[i][j] = rand()%1000;

}

}

}

timed\_mutex t\_mtx;

condition\_variable wait\_other\_threads1;

condition\_variable\_any wait\_other\_threads;

//unique\_lock<timed\_mutex> firstGuard(t\_mtx, defer\_lock);

int num = 1000;

int i = 0;

int j = 0;

int k = 0;

int numThreads = 0;

void mul\_matrix2(int\*\* mat1, int\*\* mat2, int\*\* mat3, int n)

{

if (numThreads < (num-1))

{

numThreads++;

mtx.lock();

wait\_other\_threads.wait(mtx);

mtx.unlock();

}

else

wait\_other\_threads.notify\_all();

this\_thread::sleep\_for(chrono::milliseconds(1000));

while (i < n)

{

this\_thread::get\_id();

//lock\_guard<mutex> firstGuard(mtx);

mtx.lock();

if (i >= n)

{

mtx.unlock();

break;

}

int i\_local = i, j\_local = j, k\_local = k;

k++;

if (k >= n)

{

k = 0; j++;

}

if (j >= n)

{

j = 0; i++;

}

mtx.unlock();

mat3[i\_local][j\_local] += mat1[i\_local][k\_local] \* mat2[k\_local][j\_local];

}

}

void show\_matrix(int\*\* A, int n, int m)

{

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

{

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

{

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int n = 10;

int\*\* A = new int\*[n];

new\_matrix\_random(A, n,n);

int\*\* B = new int\*[n];

new\_matrix\_random(B, n, n);

int\*\* C = new int\*[n];

new\_matrix\_null(C, n, n);

int num\_of\_thread = \_Thrd\_hardware\_concurrency();

auto begin = std::chrono::steady\_clock::now();

vector <thread> th\_vec;

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

{

th\_vec.push\_back(thread(mul\_matrix2, A, B, C, n));

}

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

{

th\_vec[i].join();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

show\_matrix(C, n, n);

return 0;

}