 **МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО МГТУ «СТАНКИН»)**

Кафедра информационных систем

**Отчет**

по 3-5 лабораторным работам «Последовательная программа. Анализ скорости выполнения»

по дисциплине «Архитектура ЭВМ и ВС»

Вариант 17

**Выполнил: студент ИДБ-21-08**

**Николаенко Владислав Андреевич**

**Преподаватель:**

**к. т. н. Саркисова Ирина Олеговна**

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc118324564)

[ОСНОВНАЯ ЧАСТЬ 3](#_Toc118324565)

[ЗАКЛЮЧЕНИЕ 11](#_Toc118324566)

# ВВЕДЕНИЕ

**Цель работы:** изучить особенности последовательного, поточного и параллельного программирования с измерением времени, затраченного программой на исполнение кода обработки матрицы с заданными условиями, а так же загрузки центрального процессора во время исполнения программы.

# ОСНОВНАЯ ЧАСТЬ

Исходное задание:



Рис. 1. Задание варианта №17

Для своего варианта разработали последовательный алгоритм и представили его в виде блок-схемы.

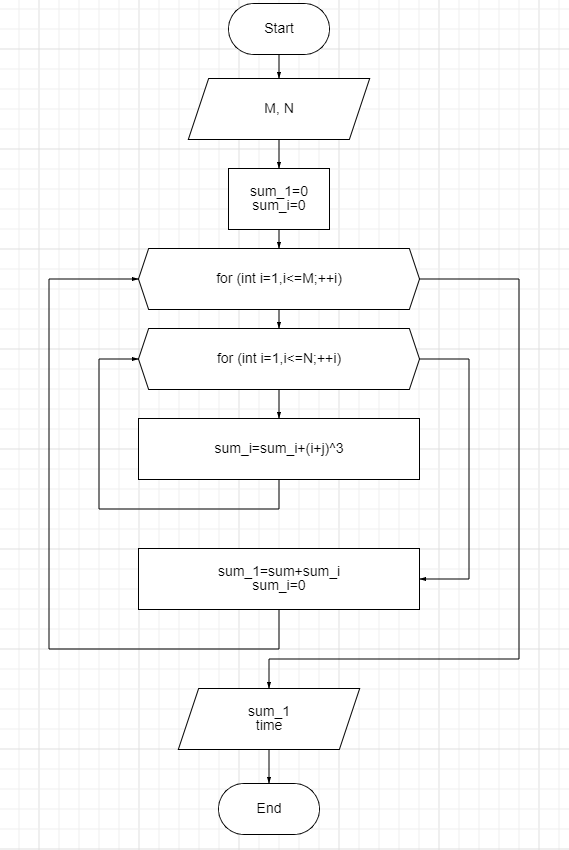


Рис. 2. Блок-схема

Листинг кода:

#include <iostream>

#include <ctime>

using namespace std;

void Schet(int m, int n)

{

setlocale(LC\_ALL, "ru");

float sum\_1 = 0;

float sum\_i = 0;

clock\_t start = clock();

for (int i = 1; i <= m; i++)

{

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

{

sum\_i += (i+j)\*(i+j)\*(i+j);

}

sum\_1 += sum\_i;

sum\_i = 0;

}

clock\_t end = clock();

cout << "Полученный pезультат: " << sum\_1 << endl;

cout << "Время (в милисекундах): " << float(end - start) << endl;

}

int main()

{

setlocale(LC\_ALL, "ru");

int M, N;

cout << "Введите размерность матрицы(сначала M, далее N):" << endl;

cin >> M >> N;

Schet(M, N);

return 0;

}

Многопоточная реализация:

**4 потока:**

#include<iostream>

#include <thread>

#include <ctime>

#include <mutex>

using namespace std;

float sum = 0;

mutex s;

void Matrix(float start\_i, int rows, int col)

{

setlocale(LC\_ALL, "ru");

float sum\_1 = 0;

float sum\_i = 0;

clock\_t start = clock();

for (float i = start\_i; i <= rows; i++)

{

for (float j = 1; j <= col; j++)

{

sum\_i += (i + j) \* (i + j) \* (i + j);

}

sum\_1 += sum\_i;

sum\_i = 0;

}

sum += sum\_1;

clock\_t end = clock();

s.lock();

cout << "Время в милисекундах: " << float(end - start) << endl;

s.unlock();

}

int main()

{

setlocale(LC\_ALL, "ru");

int M, N;

cout << "Введите размерность матрицы(сначала m, затем n):" << endl;

cin >> M >> N;

thread t1(Matrix, (M / 4) \* 0 + 1, (M / 4) \* 1, N);

thread t2(Matrix, (M / 4) \* 1 + 1, (M / 4) \* 2, N);

thread t3(Matrix, (M / 4) \* 2 + 1, (M / 4) \* 3, N);

thread t4(Matrix, (M / 4) \* 3 + 1, M, N);

t1.join();

t2.join();

t3.join();

t4.join();

cout << "Результат суммы: " << sum << endl;

return 0;

}

**8 потоков:**

#include<iostream>

#include <thread>

#include <ctime>

#include <mutex>

using namespace std;

float sum = 0;

mutex s;

void Matrix(float start\_i, int rows, int col)

{

setlocale(LC\_ALL, "ru");

float sum\_1 = 0;

float sum\_i = 0;

clock\_t start = clock();

for (float i = start\_i; i <= rows; i++)

{

for (float j = 1; j <= col; j++)

{

sum\_i += (i + j) \* (i + j) \* (i + j);

}

sum\_1 += sum\_i;

sum\_i = 0;

}

sum += sum\_1;

clock\_t end = clock();

s.lock();

cout << "Время в милисекундах: " << float(end - start) << endl;

s.unlock();

}

int main()

{

setlocale(LC\_ALL, "ru");

int M, N;

cout << "Введите размерность матрицы(сначала m, затем n):" << endl;

cin >> M >> N;

thread t1(Matrix, (M / 8) \* 0 + 1, (M / 8) \* 1, N);

thread t2(Matrix, (M / 8) \* 1 + 1, (M / 8) \* 2, N);

thread t3(Matrix, (M / 8) \* 2 + 1, (M / 8) \* 3, N);

thread t4(Matrix, (M / 8) \* 3 + 1, (M / 8) \* 4, N);

thread t5(Matrix, (M / 8) \* 4 + 1, (M / 8) \* 5, N);

thread t6(Matrix, (M / 8) \* 5 + 1, (M / 8) \* 6, N);

thread t7(Matrix, (M / 8) \* 6 + 1, (M / 8) \* 7, N);

thread t8(Matrix, (M / 8) \* 7 + 1, M, N);

t1.join();

t2.join();

t3.join();

t4.join();

t5.join();

t6.join();

t7.join();

t8.join();

cout << "Результат cуммы: " << sum << endl;

return 0;

}

**12 потоков:**

#include<iostream>

#include <thread>

#include <ctime>

#include <mutex>

using namespace std;

float sum = 0;

mutex s;

void Matrix(float start\_i, int rows, int col)

{

float sum\_1 = 0;

float sum\_i = 0;

clock\_t start = clock();

for (float i = start\_i; i <= rows; i++)

{

for (float j = 1; j <= col; j++)

{

sum\_i += (i + j) \* (i + j) \* (i + j);

}

sum\_1 += sum\_i;

sum\_i = 0;

}

sum += sum\_1;

clock\_t end = clock();

s.lock();

cout << "Время в милисекундах: " << float(end - start) << endl;

s.unlock();

}

int main()

{

setlocale(LC\_ALL, "ru");

int M, N;

cout << "Введите размерность матрицы(сначала m, затем n):" << endl;

cin >> M >> N;

thread t1(Matrix, (M / 12) \* 0 + 1, (M / 12) \* 1, N);

thread t2(Matrix, (M / 12) \* 1 + 1, (M / 12) \* 2, N);

thread t3(Matrix, (M / 12) \* 2 + 1, (M / 12) \* 3, N);

thread t4(Matrix, (M / 12) \* 3 + 1, (M / 12) \* 4, N);

thread t5(Matrix, (M / 12) \* 4 + 1, (M / 12) \* 5, N);

thread t6(Matrix, (M / 12) \* 5 + 1, (M / 12) \* 6, N);

thread t7(Matrix, (M / 12) \* 6 + 1, (M / 12) \* 7, N);

thread t8(Matrix, (M / 12) \* 7 + 1, (M / 12) \* 8, N);

thread t9(Matrix, (M / 12) \* 8 + 1, (M / 12) \* 9, N);

thread t10(Matrix, (M / 12) \* 9 + 1, (M / 12) \* 10, N);

thread t11(Matrix, (M / 12) \* 10 + 1, (M / 12) \* 11, N);

thread t12(Matrix, (M / 12) \* 11 + 1, M, N);

t1.join();

t2.join();

t3.join();

t4.join();

t5.join();

t6.join();

t7.join();

t8.join();

t9.join();

t10.join();

t11.join();

t12.join();

cout << "Результат суммы: " << sum << endl;

return 0;

}

Параллельная реализация:

#include <ctime>

#include <iostream>

#include <ppl.h>

using namespace concurrency;

using namespace std;

float sum = 0;

int M, N;

void Calculate(int i)

{

float sum\_i = 0;

for (float j = 1; j <= N; j++)

{

sum\_i += (i + j) \* (i + j) \* (i + j);

}

sum += sum\_i;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Введите размерность матрицы: " << endl;

cin >> M >> N;

clock\_t start = clock();

int i = 0;

parallel\_for(1, M + 1, Calculate);

clock\_t end = clock();

cout << "Время в милисекундах: " << (end - start) << endl;

cout << "Результат: " << sum << endl;

return 0;

}

Таблица времени выполнения программы:

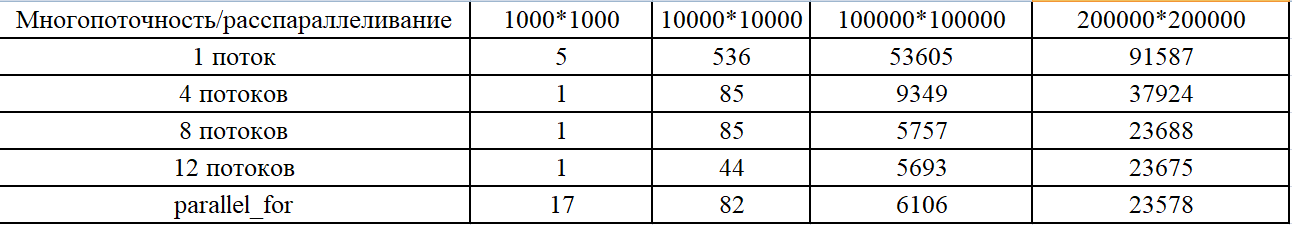


График времени выполнения программы (зависимость времени выполнения от размерности матрицы):

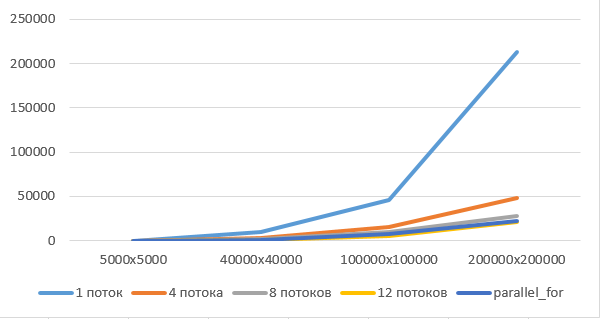


Рисунок3.График зависимости

Производительность ЦП:

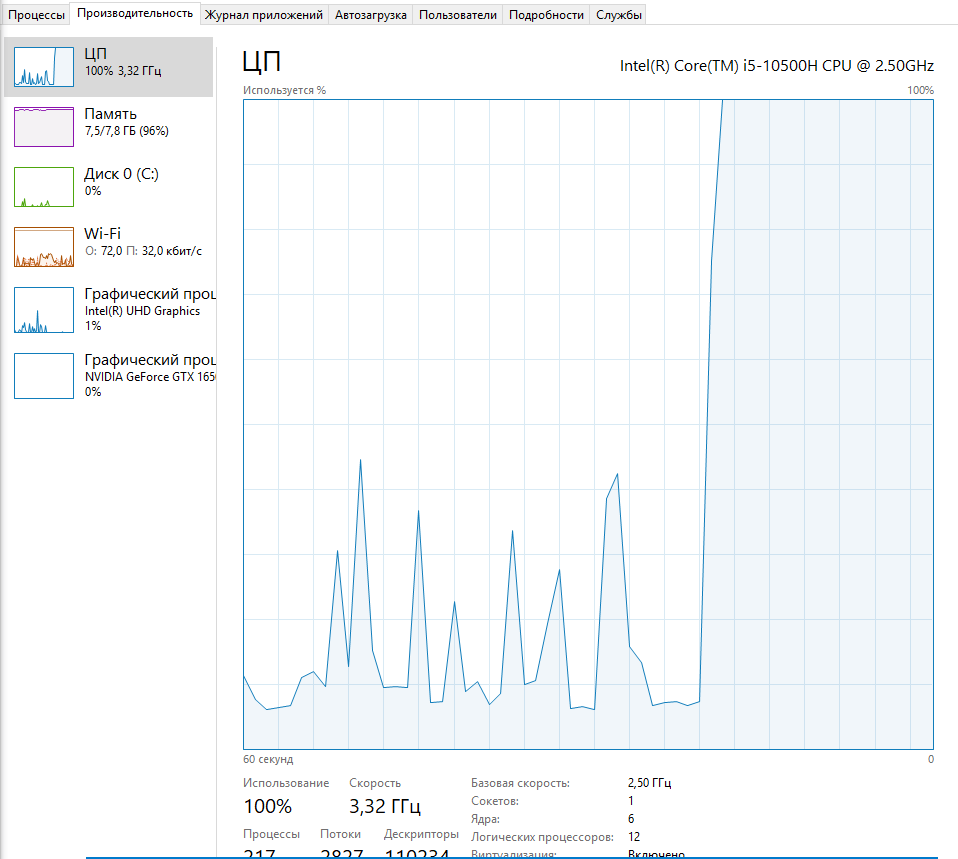


Рис.4. Производительность ЦП при 12 потоках

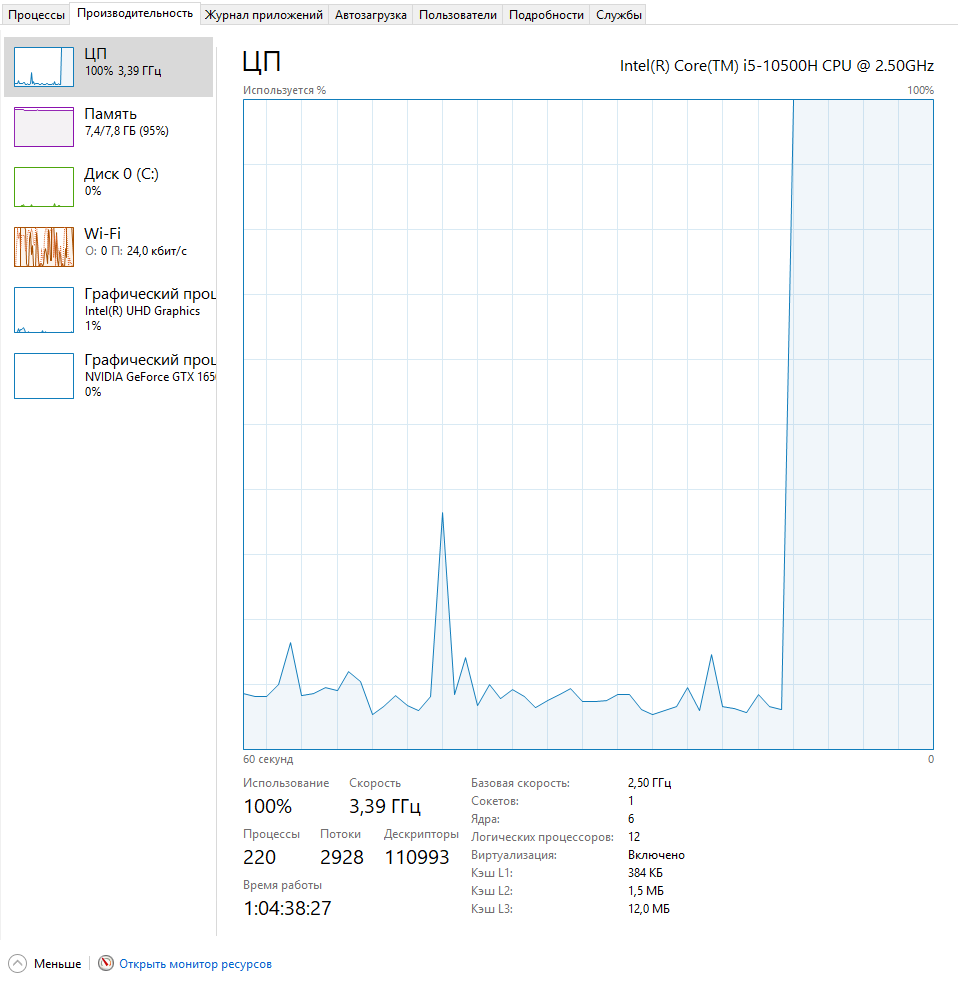


Рис.5. Производительность ЦП при parallel\_for

# ЗАКЛЮЧЕНИЕ

Таким образом, мы научились в лабораторной работе распараллеливанию простых арифметических выражений в рамках MIMD архитектуры ВС, анализу скорости выполнения однопоточного и многопоточного алгоритмов и распараллеливания.