

**МИНОБОРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

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

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **информационных систем** |

**Основная образовательная программа 09.03.02  
«Информационные системы и технологии»**

**Отчёт по дисциплине «Архитектура ЭВМ и ВС»**

**по лабораторной работе № 3,4,5**

**по теме: «Последовательная программа. Анализ скорости выполнения.»**

**Вариант №18**

|  |  |
| --- | --- |
| Студент  группы ИДБ-21-07 | Еремеев Д. В. |
|  |  |
| Преподаватель | Саркисова И.О. |

**Оглавление**

[**ВВЕДЕНИЕ** 3](#_Toc118657376)

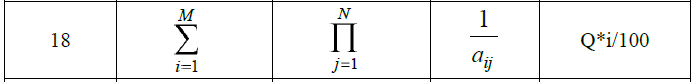
[**ХОД ВЫПОЛНЕНИЯ РАБОТЫ** 4](#_Toc118657377)

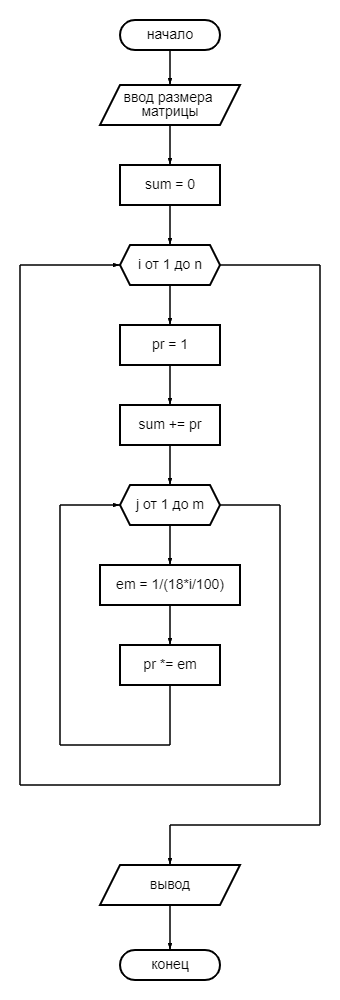
[**ВЫВОДЫ** 10](#_Toc118657378)

# **ВВЕДЕНИЕ**

**Цель работы:** анализ распараллеливания вычислений при помощи программных средств.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

**Вариант задания(рис.1):**

**** Рис. 1 Вариант задания

**Задание №1**

Для своего варианта разработать последовательный алгоритм решения задачи. Представить его в виде блок-схемы(рис.2). Выполнить контрольный расчёт на матрице небольшой размерности (таблица 1).

Таблица 1. Контрольный расчёт

|  |  |  |  |
| --- | --- | --- | --- |
| Матрица |  |  |  |
| 0,18 | 0,18 | 0,18 |  |
| 0,36 | 0,36 | 0,36 |  |
| 0,54 | 0,54 | 0,54 |  |
| Матрица 1/ на элемент | | | Произведение  строк |
| 5,555556 | 5,555556 | 5,555556 | 171,4678 |
| 2,777778 | 2,777778 | 2,777778 | 21,43347 |
| 1,851852 | 1,851852 | 1,851852 | 6,350658 |
| Сумма |  |  | 199,2519 |

Рис. 2 Блок-схема

**Задание №2**

Реализовать алгоритм программно, добавив функцию определения времени выполнения программы.

Код программы:

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

int n=3, m=3;

double em, sum = 0, pr;

double\*\* mas = new double\* [n];

double start = clock();

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

mas[i] = new double[m];

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

pr = 1;

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

mas[i][j] = (double)18 \* (i + 1) / 100;

em = (double)1 / mas[i][j];

pr \*= em;

}

sum += pr;

}

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

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

cout << mas[i][j] << '\t';;

}

cout << endl;

}

cout << endl;

cout << sum;

cout << endl;

cout << "runtime = " << (clock() - start)/CLOCKS\_PER\_SEC;

return 0;

}

Вывод программы совпадает с контрольным значением из таблицы 1(рис.3)

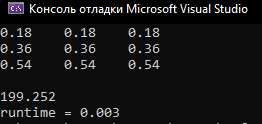


Рис. 3. Вывод программы

**Задание №3**

Протестируйте программу, определив время выполнения при изменении размерности матриц, результаты занесите в таблицу (Таблица 2).

Таблица 2. Время работы в зависимости от размерности матрицы и количества потоков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| размер | 3\*3 | 14750\*14750 | 29500\*29500 | 44250\*44250 | 59000\*59000 |
| Послед. | 0,001 | 3,576 | 14,1 | 31,605 | 57,714 |
| 2 пот | 0,003 | 1,43 | 5,583 | 12,928 | 23,003 |
| 6 пот | 0,005 | 0,522 | 1,911 | 4,394 | 7,838 |
| 12 пот | 0,01 | 0,335 | 1 | 2,446 | 4,885 |
| 24 пот | 0,014 | 0,296 | 0,953 | 2,315 | 4,228 |
| авто | 0,004 | 3,083 | 11,737 | 26,505 | 51,082 |

**Задание №4**

В случае, если время выполнения программы слишком мало для полноценного анализа, заменить секунды тактами (не актуально)

**Задание №5**

Если элемент массива генерируется случайным образом, то необходимо явно ограничить величину генерируемых чисел. Так же, в случае быстрого переполнения возможно модернизировать свои формулы (не актуально).

**Задание №6**

Проанализируйте созданный последовательный алгоритм и предложите его многопоточную реализацию.

Код программы:

#include <iostream>

#include <ctime>

#include <thread>

#include <future>

using namespace std;

void potok2(double sum, double pr, int n, int m, double\*\* mas, double em, promise<int> potok12) {

for (int i = 0; i < n/2; i++) {

pr = 1;

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

mas[i][j] = (double)18 \* (i + 1) / 100;

em = (double)1 / mas[i][j];

pr \*= em;

}

sum += pr;

}

for (int i = 0; i < n/2; i++) {

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

cout << mas[i][j] << '\t';;

}

cout << endl;

}

potok12.set\_value(sum);

}

int main()

{

int n = 3, m = 3;

double em, sum = 0, pr;

double\*\* mas = new double\* [n];

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

mas[i] = new double[m];

promise<int> potok12;// нужно для передачи данных из потока

future<int> potok22 = potok12.get\_future();

double start = clock();

thread a(potok2, sum, pr, n, m, mas, em, std::move(potok12));

for (int i = n/2; i < n; i++) {

pr = 1;

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

mas[i][j] = (double)18 \* (i + 1) / 100;

em = (double)1 / mas[i][j];

pr \*= em;

}

sum += pr;

}

a.join();

for (int i = n / 2; i < n; i++) {

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

cout << mas[i][j] << '\t';;

}

cout << endl;

}

sum += potok22.get();

cout << "runtime = " << (clock() - start) / CLOCKS\_PER\_SEC;

cout << endl;

cout << sum;

cout << endl;

return 0;

}

Вывод программы с 2-мя потоками приближено совпадает с контрольным значением из таблицы 1 из-за округления(рис.4)

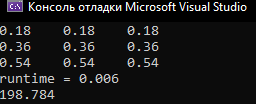


Рис.4. Вывод программы с 2-мя потоками

**Задание №7**

Реализуйте многопоточный алгоритм программно. (Возможен любой метод реализации, в том числе и не рассмотренный в данном пособии.)

Код программы:

#include <iostream>

#include <ctime>

#include <omp.h>

using namespace std;

int main()

{

int n = 3, m = 3;

double em, sum = 0, pr;

double\*\* mas = new double\* [n];

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

mas[i] = new double[m];

double start = clock();

#pragma omp parallel

{

#pragma omp for schedule(static)

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

pr = 1;

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

mas[i][j] = (double)18 \* (i + 1) / 100;

em = (double)1 / mas[i][j];

pr \*= em;

}

sum += pr;

}

}

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

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

cout << mas[i][j] << '\t';;

}

cout << endl;

}

cout << "runtime = " << (clock() - start) / CLOCKS\_PER\_SEC;

cout << endl;

cout << sum;

cout << endl;

return 0;

}

Вывод программы совпадает с контрольным значением из таблицы 1(рис.5)

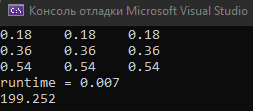


Рис.5 Вывод программы с автоматическим распараллеливанием

**Задание №8**

Протестировать программу, определив время выполнения при изменении размерности матриц, результаты занести в таблицу (таблица 2).

**Задание №9**

Построить график времени выполнения однопоточного и многопоточного приложений в зависимости от изменения размерности матриц (рис.6).

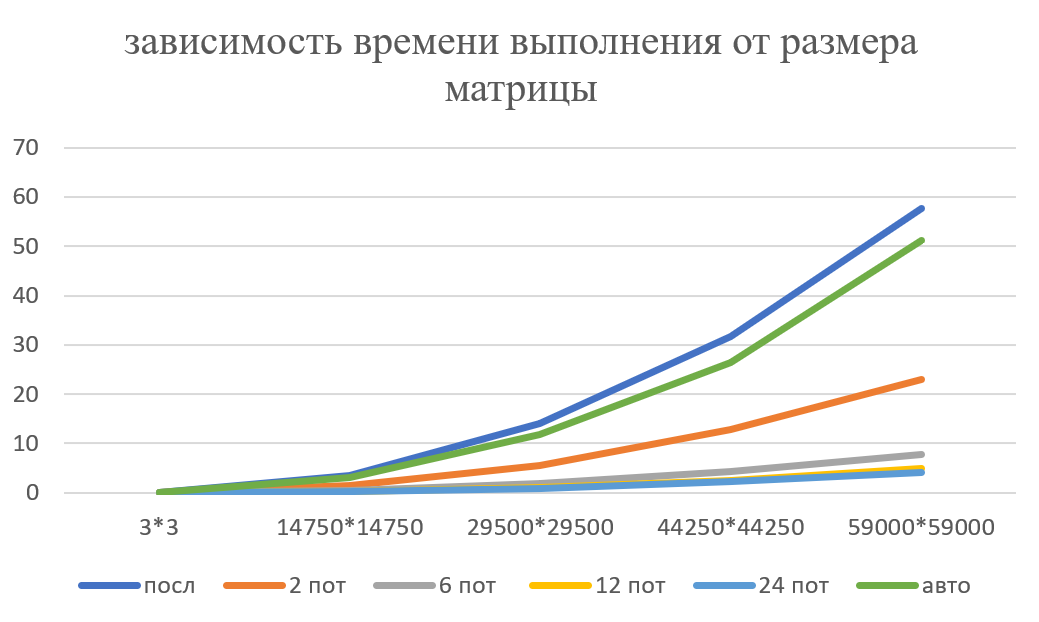
****

Рис. 6. График времени выполнения однопоточного и многопоточного приложений в зависимости от изменения размерности матриц

**Задание №10**

Аналогично п.7-9 реализуйте параллельное вычисление арифметического выражения используя Класс Parallel, добавив график времени выполнения этой версии программы на график, построенный в п.9 (рис. 6).

**Задание №11**

Добавьте в отчёт принтскрины загрузки ядер процессора в момент выполнения всех трех вариантов программы (рис.7 - 12)

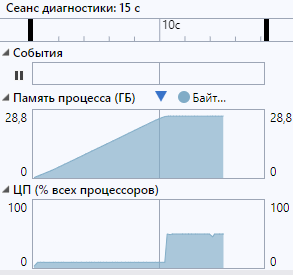
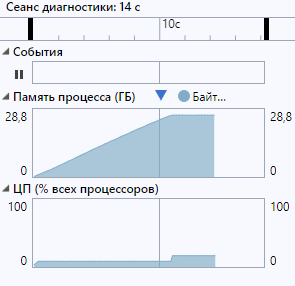
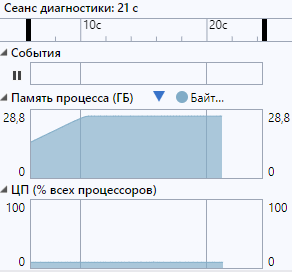
****

Рис.7. Последовательно Рис.8. 2 потока Рис.9. 6 потоков

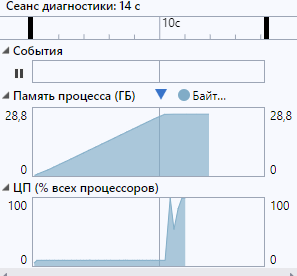
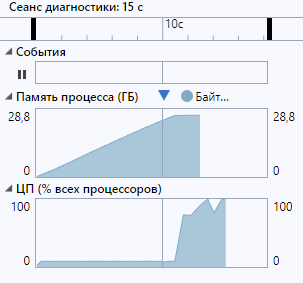
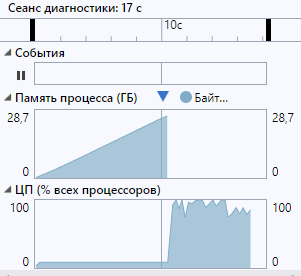


Рис.10. 12 потоков Рис.11. 24 потока Рис.12. Авто. распар.

**Задание №12**

Проанализируйте полученные результаты и сделайте выводы.

# **ВЫВОДЫ**

На основе выполненной работы можно сделать несколько выводов:

1. Распараллеливание программы ускоряет работу, кроме тех случаев, когда вычисления маленькие, в те моменты оно, наоборот, замедляет программу, потому что вызов потока и обработка в нём значений занимает больше времени чем сами вычисления.
2. Потоки не должны превышать количество ядер, так как они всё равно не станут быстрее. Самое лучшее, когда количество потоков и количество ядер совпадают.
3. Автоматическое распараллеливание не всегда эффективно. В моём случае оно не принесло мне ничего, время автоматического распараллеливания и последовательного практически совпадают.