МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ

Отчет по лабораторной работе №3,4,5

«Последовательная программа. Анализ скорости выполнения.»

Вариант №28

Выполнил студент гр. ИДБ-21-06 Музафаров К.Р.

Проверил Саркисова И.О.

Москва 2022г.

Задача:

Вычислить $\prod_{i=1}^{M}\sum_{j=1}^{N}\sqrt{c_{ij}}$, где c_{ij} =j*Q/100, (i=1,2,…,M, j=1,2,…,N), Q – номер вашего варианта.

28	$\prod_{i=1}^{K}$	$\prod_{j=1}^{N}$	$\lvert a_{ij} \rvert$	j*Q/100

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

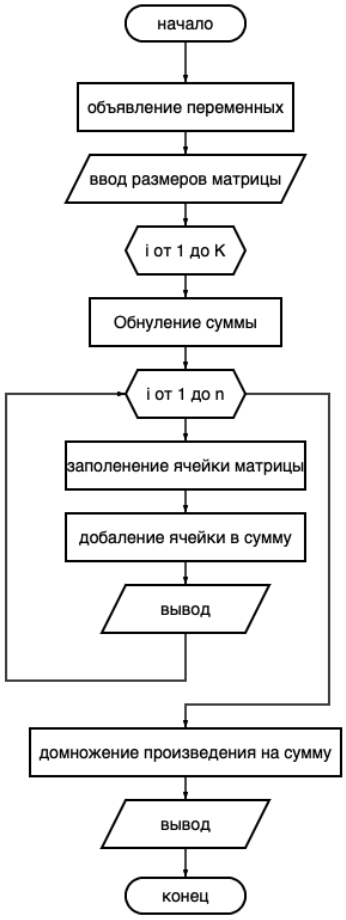


Рис.1. Блок-схема.

Контрольный расчёт на матрице небольшой размерности			
0.3	0.6	0.9	Сумма на строке = 1.68
0.3	0.6	0.9	Сумма на строке = 1.68
0.3	0.6	0.9	Сумма на строке = 1.68
Итоговое произведение выданное программой = 4.74163		Итоговое произведение посчитанное вручную = 4.74163	

2. Реализовать алгоритм программно, добавив функцию определения времени выполнения программы.

Код программы:

```
#include <iostream>
#include <cmath>
#include <ctime>
using namespace std;
int main()
{
   unsigned long start_time =  clock();
   setlocale(LC_ALL, "Russian");
   int N, K;
   double Q = 28, multiplication = 1, summ = 0;
   cout << "Введите количество трок матрицы: \n";
   cin >> K;
   cout << "Введите количество столбцов матрицы: \n";
   cin >> N;
   double** arr = new double* [K+1];
   for (int i = 1; i <= K; i++)
   {
      cout <<"| ";
      arr[i] = new double[N+1];
      summ = 0;
      for (int j = 1; j <= N; j++)
      {
         arr[i][j] = j*Q/100;
         summ += abs(arr[i][j]);
         cout << round(arr[i][j]*10)/10 <<" ";
      }
      multiplication *= summ;
      cout <<"| \n";
   }
   cout << "Итоговое прозведение:" << multiplication << "\n";
   cout << "Время работы програмы:" << clock() - start_time << "ms \n";
```

}

```
Введите количество трок матрицы:
3
Введите количество столбцов матрицы:
3
| 0.3 0.6 0.8 |
| 0.3 0.6 0.8 |
| 0.3 0.6 0.8 |
Итоговое прозведение:4.74163
Время работы програмы:308ms
Program ended with exit code: 0
```

Рис.2. Вывод программы с 1 потоком.

3. Протестируйте программу, определив время выполнения при изменении размерности матриц, результаты занесите в таблицу.

Размер	3x3	1000x1000	10000x10000	50000x50000
Последовательно	308ms	44763ms	1462832ms	30217542ms
2 потока	289ms	33593ms	1181609ms	28207112ms
8 потоков	1051ms	13042ms	1124735ms	22210815ms
parallel_for	234ms	38003ms	1319755ms	27042204ms

4. Проанализируйте созданный последовательный алгоритм и предложите его многопоточную реализацию.

Код программы:

```
#include <mutex>
#include <vector>
#include <thread>
#include <iostream>
#include <cmath>

using namespace std;
using namespace this_thread;

int main()
```

```
{

    setlocale(LC_ALL, "Russian");
    int N = 0, K = 0;
    float Q = 28, multiplication = 1, *multiplicationPointer = &multiplication;
    cout << "Введите количество трок матрицы: \n";
    cin >> K;
    cout << "Введите количество столбцов матрицы: \n";
    cin >> N;
    mutex threadMtx;
    float** arr = new float* [K+1];
    unsigned long start_time =  clock();
    thread tr([&threadMtx ,N ,K ,Q ,arr ,multiplicationPointer ]()
            {
            for (int i = 1; i <= K; ++i)
            {
               cout << "| ";
               arr[i] = new float[N+1];
               float summ = 0;
               for (int j = 1; j <= N; ++j)
               {
                  lock_guard<mutex> lockGuard(threadMtx);
                  arr[i][j] = j*Q/100;
                  summ += abs(arr[i][j]);
                  cout << round(arr[i][j]*10)/10 <<" ";
               }
               *multiplicationPointer *= summ;
            cout <<"| \n";
        }
    });

    tr.join();

    cout << "Итоговое прозведение:" << multiplication << "\n";
    cout << "Время работы програмы:" << clock() - start_time << "ms \n";

    return 0;
}
```

```
Введите количество трок матрицы:
3
Введите количество столбцов матриц
3
| 0.3 0.6 0.8 |
| 0.3 0.6 0.8 |
| 0.3 0.6 0.8 |
Итоговое прозведение:4.74163
Время работы програмы:289ms
Program ended with exit code: 0
```

Рис.3. Вывод программы с 2 потоками.

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

8 потоков

Код программы:

```
#include<iostream>
#include <thread>
#include <ctime>
#include <mutex>
#include <cmath>

using namespace std;

float multiplication = 1, summ = 0;;
mutex s;

void Matrix(float start_i, int rows, int col)
{
   setlocale(LC_ALL, "ru");
   float Q = 28, arr;
   for (float i = start_i; i <= rows; i++)
   {
      summ = 0;
      s.lock();
      cout << "|";
      s.unlock();
      for (float j = 1; j <= col; j++)
      {
         arr = j*Q/100;
         s.lock();
         cout << round(arr*10)/10 << " ";
         s.unlock();
         summ += abs(arr);
      }
      multiplication *= summ;
      s.lock();
      cout << "|\n";
      s.unlock();
   }
}
```

```
int main()
{
    setlocale(LC_ALL, "ru");
    int N, K;
    cout << "Введите количество трок матрицы: \n";
    cin >> K;
    cout << "Введите количество столбцов матрицы: \n";
    cin >> N;
    unsigned long start_time =  clock();
    thread t1(Matrix, (N / 8) * 0 + 1, (K / 8) * 1, N);
    thread t2(Matrix, (N / 8) * 1 + 1, (K / 8) * 2, N);
    thread t3(Matrix, (N / 8) * 2 + 1, (K / 8) * 3, N);
    thread t4(Matrix, (N / 8) * 3 + 1, (K / 8) * 4, N);
    thread t5(Matrix, (N / 8) * 4 + 1, (K / 8) * 5, N);
    thread t6(Matrix, (N / 8) * 5 + 1, (K / 8) * 6, N);
    thread t7(Matrix, (N / 8) * 6 + 1, (K / 8) * 7, N);
    thread t8(Matrix, (N / 8) * 7 + 1, N, N);

    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();
    t6.join();
    t7.join();
    t8.join();

    cout << "Итоговое прозведение:" << multiplication << "\n";
    cout << "Время работы програмы:" << clock() - start_time << "ms \n";

    return 0;
}
```

```
Введите количество трок матрицы:
3
Введите количество столбцов матрицы:
3
|0.3 0.6 0.8 |
|0.3 0.6 0.8 |
|0.3 0.6 0.8 |
Итоговое прозведение:4.74163
Время работы програмы:1135ms
Program ended with exit code: 0
```

Рис.4. Вывод программы с 8 потоками.

parallel_for

Код программы:

```
#include <iostream>
#include <ctime>
#include <omp.h>
using namespace std;
int main()
{
   setlocale(LC_ALL, "Russian");
   int N = 0, K = 0;
   float Q = 28, multiplication = 1, summ = 0;
   cout << "Введите количество трок матрицы: \n";
   cin >> K;
   cout << "Введите количество столбцов матрицы: \n";
   cin >> N;
   float** arr = new float* [K + 1];
   for (int i = 0; i <= K; i++)
      arr[i] = new float[N + 1];
   unsigned long start_time = clock();
#pragma omp parallel
   {
#pragma omp for schedule(static)
      for (int i = 1; i <= K; i++) {
         summ = 0;
         cout << "|";
         for (int j = 1; j <= N; j++) {
            arr[i][j] = j * Q / 100;
            cout << round(arr[i][j]*10)/10 <<" ";
            summ += abs(arr[i][j]);
         }
         cout << «|\n";
         multiplication *= summ;
      }
   }
   for (int i = 1; i < K; i++) {
      for (int j = 1; j < N; j++) {
         cout << round(arr[i][j] * 10) / 10 << " ";
      }
      cout << endl;
   }
   cout << "Итоговое прозведение:" << multiplication << "\n";
```

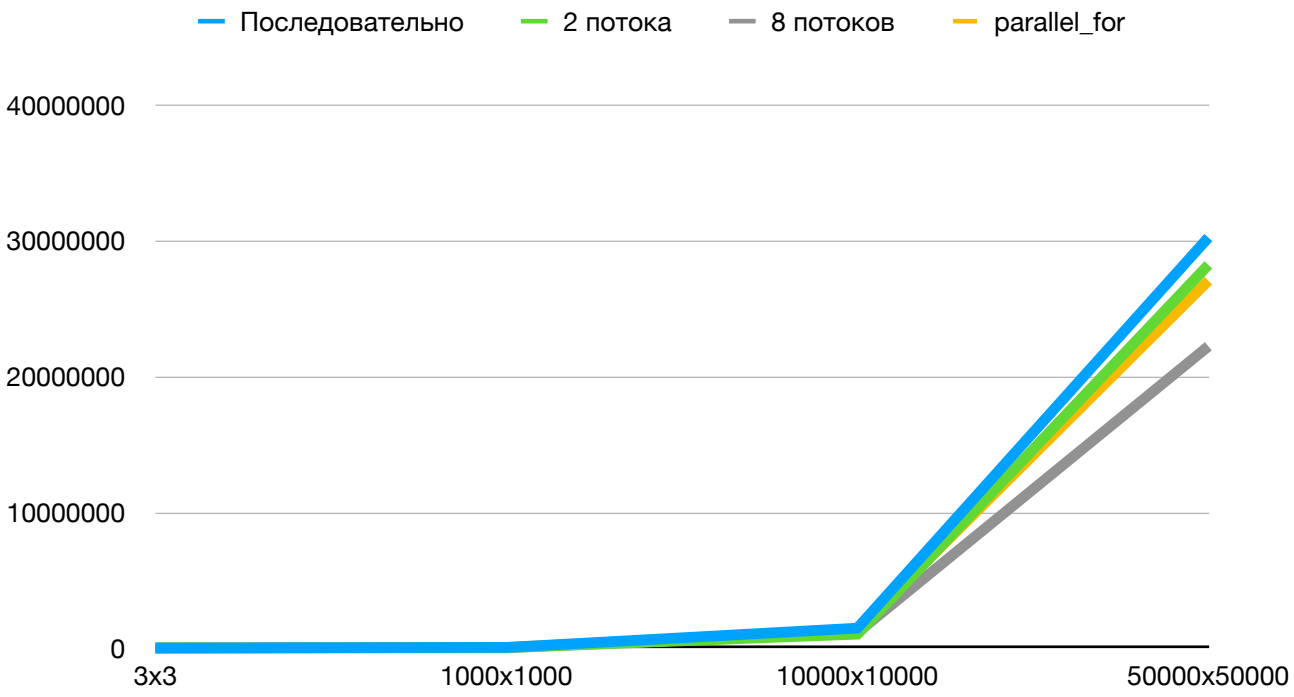
```
    cout << "Время работы програмы:" << clock() - start_time << "ms \n";

    return 0;
}
```

```
Введите количество трок матрицы:
3
Введите количество столбцов матрицы:
3
|0.3 0.6 0.8 |
|0.3 0.6 0.8 |
|0.3 0.6 0.8 |
Итоговое прозведение:4.74163
Время работы програмы:234ms
Program ended with exit code: 0
```

Рис.4. Вывод программы автоматическим распределением потоков.

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



7.Добавьте в отчёт принтскрины загрузки ядер процессора в момент выполнения всех трех вариантов программы.

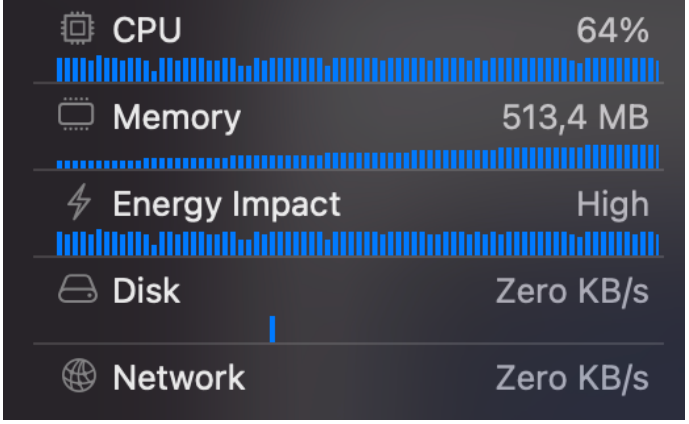


Рис.5. Загрузка ядер при 1 потоке

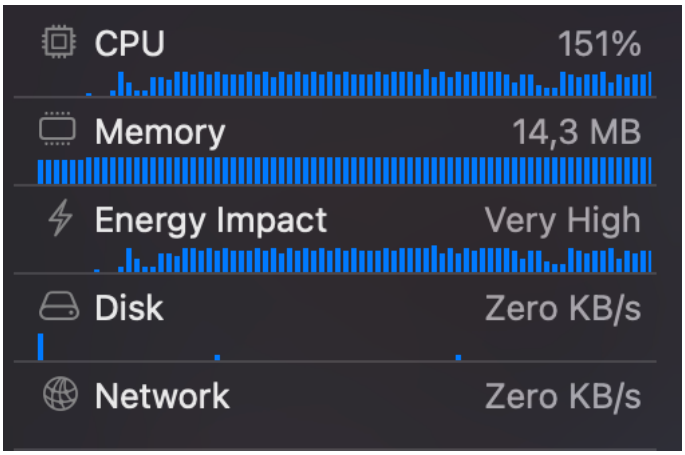


Рис.6. Загрузка ядер при 8 потоках

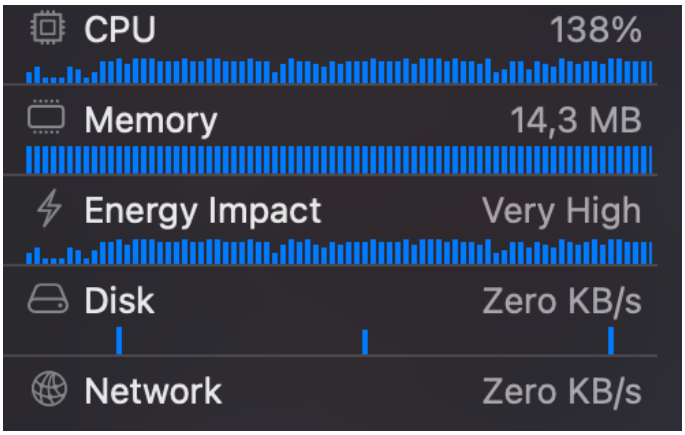


Рис.7. Загрузка ядер при авторасределении потоков

Вывод: Мы научились в лабораторной работе распараллеливанию простых арифметических выражений в рамках MIMD архитектуры ВС, анализу скорости выполнения однопоточного и многопоточного алгоритмов и распараллеливания. Выяснили что распараллеливание ускоряет работу программы при условии что в ней происходят большие вычисления, выяснили что автоматическое распараллеливание не приносит большой пользы, самый идеальный сценарий когда количество потоков совпадает с количеством ядер.