Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Отчёт к лабораторной работе №13

по дисциплине

«Языки программирования»

Работу выполнила

Студент группы СКБ222 _____ М. Х. Халимов

подпись, дата

Работу проверил _____ С. А. Булгаков

подпись, дата

Содержание

Постановка задачи

Реализовать алгоритм обработки данных (на свое усмотрение), а также его параллельную версию с использованием возможностей *std::thread*.

Получить эмпирическую зависимость изменения быстродействия от объема данных при фиксированном числе параллельных потоков используя возможности *std::chrono*.

Построить теоретическую оценку увеличения быстродействия при фиксированном объеме данных и различном числе параллельных потоков. Получить эмпирическое подтверждение построенной теоретической оценки используя возможности *std::chrono*.

1. Описание вспомогательных функций программы Lab-13

1.1. Функция *Partition*

Функция получает на вход вектор и два целочисленных значения. Функция разделяет массив на подмассивы. Вызывается в *ParallelQuickSort*.

1.2. Функция *ParallelQuickSort*

Функция получает на вход вектор и три целочисленных значения. Выполняет быструю сортировку в многопоточном режиме. Вызывается в *main.*

2. Описание функции main программы Lab-13

Функция производит быструю сортировку массивов размером от 10 до 990010. Время каждой сортировки замеряется и выводится в консоль.

3. Описание вспомогательных функций программы *Lab-13_single*

3.1. Функция Partition

Функция получает на вход вектор и два целочисленных значения. Функция разделяет массив на подмассивы. Вызывается в *QuickSort.*

3.2. Функция QuickSort

Функция получает на вход вектор и два целочисленных значения. Выполняет быструю сортировку в однопоточном режиме. Вызывается в *main.*

4. Описание функции *main* программы *Lab-13_single*

Функция производит быструю сортировку массивов размером от 10 до 990010. Время каждой сортировки замеряется и выводится в консоль.

5. Оценка быстродействия

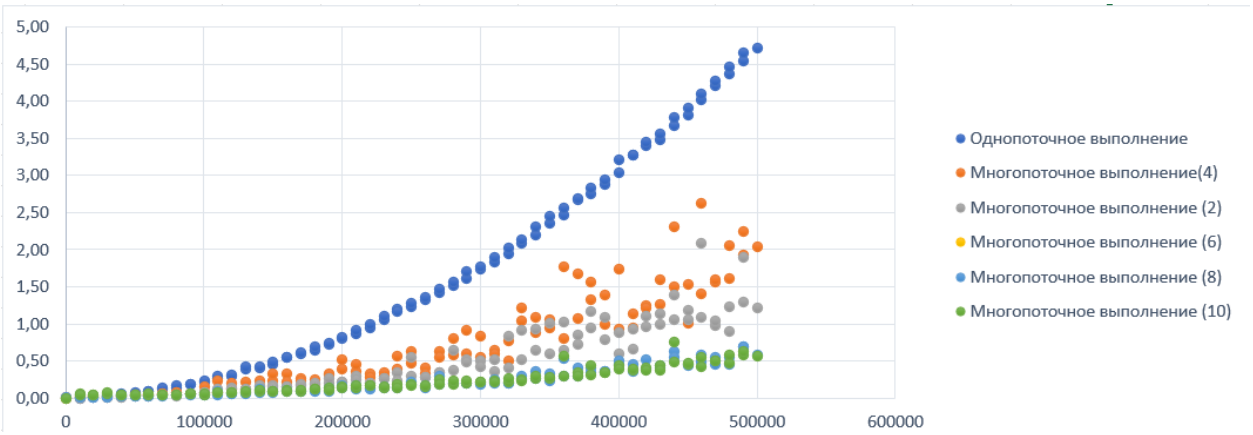
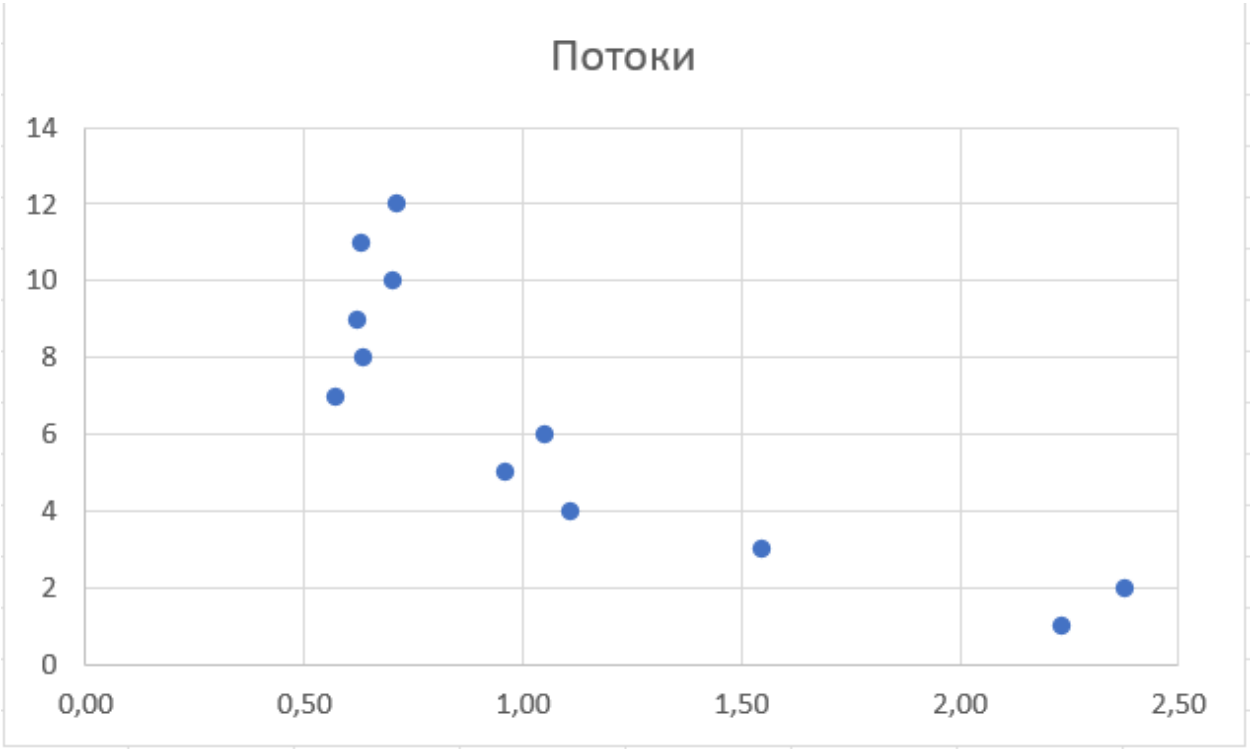


Рисунок 1. ***Графики зависимости времени выполнения алгоритма от объема данных.***

На графике выше продемонстрирована зависимость быстродействия от объема при разном количестве потоков. Можно заметить, что код, запущенный в многопоточном режиме (в нашем случае в двухпоточном) в большинстве случае быстрее аналогичного, запущенного последовательно.

В случае же фиксированного объема данных наблюдается следующая зависимость.



По закону Амдала в многопоточном режиме алгоритм работает в 3,128 раза быстрее.

6. Листинг

Листинг Lab-13

```
 0,0000011440
 0,0023661800
 0,0056732800
 0,0097392200
 0,0177279000
 0,0310957000
 0,0388315000
 0,0443983000
 0,0536334000
 0,0649274000
 0,0730635000
 0,0879740000
 0,0966457000
 0,1182690000
 0,1318830000
 0,1393050000
 0,1753830000
 0,1778250000
 0,1925140000
 0,2147060000
 0,2358310000
 0,2582020000
 0,2911460000
 0,2993950000
 0,3172540000
 0,3959860000
 0,4249790000
 0,4029190000
 0,4246650000
 0,4592110000
 0,4790720000
 0,5549980000
 0,5544000000
 0,5992610000
 0,6076230000
 0,6450380000
 0,6859570000
 0,7177640000
 0,7453110000
 0,7997780000
 0,8250640000
```

0,8660140000
0,9162560000
0,9430870000
0,9958820000
1,0575800000
1,1058200000
1,1922300000
1,1655300000
1,2335200000
1,2776400000
1,3279700000
1,3633600000
1,4244300000
1,4686800000
1,5131800000
1,5694000000
1,6164500000
1,7067000000
1,7378800000
1,7750800000
1,8268000000
1,8995500000
1,9490200000
2,0186400000
2,0838900000
2,1362900000
2,1999800000
2,3146900000
2,3493100000
2,4563500000
2,4651100000
2,5692200000
2,6659500000
2,6905200000
2,7485800000
2,8339600000
2,8748900000
2,9460200000
3,0373800000
3,2199700000
3,2732900000
3,2697000000
3,4039900000
3,4578900000
3,4836500000
3,5578000000
3,6785000000
3,7877300000

3,8098600000
3,9059000000
4,0228000000
4,1019300000
4,2033900000
4,2718700000
4,3701100000
4,4630300000
4,5495700000
4,6527200000
4,7166400000

Листинг Lab-13_single

0,0023530000
0,0648404000
0,0377048000
0,0450264000
0,0396891000
0,0619855000
0,0805472000
0,0383150000
0,0368245000
0,0407577000
0,0433169000
0,0455237000
0,0575242000
0,0408142000
0,0487353000
0,0465464000
0,0459435000
0,0540908000
0,0564186000
0,0474876000
0,0631279000
0,0745418000
0,0684402000
0,0705847000
0,0768651000
0,0761322000
0,0904344000
0,0946429000
0,1112590000
0,0938243000
0,0923538000
0,1069310000
0,0910436000
0,0974400000
0,1086140000

0,1219420000
0,1217870000
0,1344700000
0,1122650000
0,1560480000
0,1323010000
0,1382370000
0,1753280000
0,1762650000
0,1620060000
0,1453790000
0,1596990000
0,1794420000
0,1445050000
0,1897190000
0,1657510000
0,1585260000
0,1744740000
0,2468140000
0,1792730000
0,1894930000
0,2275950000
0,1998900000
0,2248210000
0,2234970000
0,2235980000
0,2253220000
0,2157130000
0,2123350000
0,2686260000
0,2489520000
0,2396680000
0,2646340000
0,2883150000
0,2657330000
0,2771260000
0,3009010000
0,5594570000
0,3477210000
0,2891460000
0,4328490000
0,3088360000
0,3411450000
0,3390280000
0,4415890000
0,3852760000
0,3709120000
0,3861340000

0,3992630000
0,3824250000
0,4349950000
0,3825850000
0,7533510000
0,4917160000
0,4506840000
0,4703240000
0,4213900000
0,5500270000
0,5035210000
0,4999090000
0,4684870000
0,5870910000
0,5789130000
0,6305670000
0,5715460000

Приложение А

Исходный код Lab-13

```
#include <iostream>
#include <vector>
#include <chrono>
#include <thread>

// Функция разделения массива на подмассивы
int Partition(std::vector<int>& arr, int start, int end)
{
    int pivot = arr[end];

    int pIndex = start;

    for (int i = start; i < end; i++)
    {
        if (arr[i] <= pivot)
        {
            std::swap(arr[i], arr[pIndex]);
            pIndex++;
        }
    }

    std::swap(arr[pIndex], arr[end]);
    return pIndex;
}

void ParallelQuickSort(std::vector<int>& arr, int start, int end,
int depth){

    if(start >= end){
        return;
    }

    int pivot = Partition (arr, start, end);

    if(depth-- > 0){
        // Создание новых потоков для сортировки двух половин
массива
        std::thread t1(ParallelQuickSort, std::ref(arr), start,
pivot - 1, depth);
        std::thread t2(ParallelQuickSort, std::ref(arr), pivot +
1, end, depth);

        // Ожидание завершения потоков
        t1.join();
        t2.join();
    } else {
        // Сортировка последовательно в глубину рекурсии
        ParallelQuickSort(arr, start, pivot - 1, depth);
        ParallelQuickSort(arr, pivot + 1, end, depth);
```

```
    }
}

int main() {
    for (int k = 10; k <= 1000000;  k = k + 10000)
    {
        std::vector<int> arr (k);
        for(int i = 0; i < arr.size(); i++){
            arr[i] = rand() % 1000;
        }
        int n = arr.size();

   /* std::cout << "Исходный массив: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    */

        auto start = std::chrono::steady_clock::now();
    // Запуск параллельной быстрой сортировки
        ParallelQuickSort(arr, 0, n - 1, 10);
        auto end = std::chrono::steady_clock::now();

    /*std::cout << "Отсортированный массив: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    */

        std::chrono::duration<double> es = end - start;

        std::cout << es.count() << std::endl;
    }
    return 0;
}
```

Приложение Б

Исходный код Lab-13_single

```
#include <iostream>
#include <vector>
#include <chrono>


// Функция разделения массива на подмассивы
int Partition(std::vector<int>& arr, int start, int end)
{
    int pivot = arr[end];
```

```
    int pIndex = start;

    for (int i = start; i < end; i++)
    {
        if (arr[i] <= pivot)
        {
            std::swap(arr[i], arr[pIndex]);
            pIndex++;
        }
    }

    std::swap(arr[pIndex], arr[end]);
    return pIndex;
}

void QuickSort(std::vector<int>& arr, int start, int end){

    if(start >= end){
        return;
    }

    int pivot = Partition (arr, start, end);

    QuickSort(arr, start, pivot - 1);
    QuickSort(arr, pivot + 1, end);
}

int main() {
    for(int k = 10; k <= 1000000; k += 10000)
    {
        std::vector<int> arr (k);
        for(int i = 0; i < arr.size(); i++){
            arr[i] = rand() % 1000;
        }
        int n = arr.size();

    /* std::cout << "Исходный массив: ";
        for (int num : arr) {
            std::cout << num << " ";
        }
        std::cout << std::endl;
        */

        auto start = std::chrono::steady_clock::now();
        // Запуск быстрой сортировки
        QuickSort(arr, 0, n - 1);
        auto end = std::chrono::steady_clock::now();

        /*std::cout << "Отсортированный массив: ";
        for (int num : arr) {
            std::cout << num << " ";
        }
```

```
            std::cout << std::endl;
            */

            std::chrono::duration<double> es = end - start;

            std::cout << es.count() << std::endl;
        }
        return 0;
    }
```