

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Нижегородский государственный университет им. Н.И. Лобачевского»  
**Институт информационных технологий, математики и механики**  
Направление подготовки: Фундаментальная информатика и  
информационные технологии

**Отчет**  
по лабораторной работе  
**«Сортировка Шелла со слиянием «Разделяй и  
властвуй»»**

**Выполнила:**

студент группы 381606-1

Яковлев Д. Л.

**Проверил:**

доцент каф. МОСТ, ИИТММ,

кандидат технических наук

Сысоев А.В.

Нижний Новгород

2019

# Содержание

|                                 |    |
|---------------------------------|----|
| Введение.....                   | 3  |
| Постановка задачи.....          | 4  |
| Метод решения .....             | 5  |
| Схема распараллеливания .....   | 7  |
| OpenMP .....                    | 7  |
| TBV .....                       | 8  |
| Руководство пользователя.....   | 10 |
| Подтверждение корректности..... | 12 |
| Результаты экспериментов.....   | 13 |
| Выводы из результатов.....      | 15 |
| Заключение .....                | 16 |
| Литература .....                | 17 |

## Введение

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить там, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить. Ни одна другая проблема не породила такого количества разнообразнейших решений, как задача сортировки. Универсального, наилучшего алгоритма сортировки на данный момент не существует.

## Постановка задачи

Была поставлена задача разработать программу, сортирующую массив элементов с помощью сортировки Шелла с использованием методов параллельного программирования. Для этого разработать последовательный и параллельный алгоритм данной сортировки. При параллельном алгоритме сортировки, массив чисел разбивается на части, потом каждая часть сортируется независимо, затем выполняется слияние методом «Разделяй и властвуй».

Программа должна иметь последовательную и 2 параллельные реализации, выполненные при помощи средств библиотек параллельного программирования OpenMP и TBB.

В программе необходимо:

1. Выполнить проверку совпадения результатов последовательной и параллельной реализаций.
2. Обеспечить генерацию данных для задач произвольной размерности. Параметры задачи должен задавать пользователь.
3. Вывести время работы последовательного и параллельного алгоритмов.

## Метод решения

**Сортировка Шелла** — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии  $d$ . После этого процедура повторяется для некоторых меньших значений  $d$ , а завершается сортировка Шелла упорядочиванием элементов при  $d=1$  (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до  $O(n^2)$ , что хуже, чем худшее гарантированное время для сортировки Шелла.

Рассмотрим пример:

Пусть дан массив  $A = (32, 95, 16, 82, 24, 66, 35, 19, 75, 54, 40, 43, 93, 68)$  и выполняется его сортировка методом Шелла, а в качестве значений  $d$  выбраны 5, 3, 1.

На первом шаге сортируются подмассивы  $A$ , составленные из всех элементов  $A$ , различающихся на 5 позиций, то есть подмассивы  $A_{\{5,1\}} = (32, 66, 40)$ ,  $A_{\{5,2\}} = (95, 35, 43)$ ,  $A_{\{5,3\}} = (16, 19, 93)$ ,  $A_{\{5,4\}} = (82, 75, 68)$ ,  $A_{\{5,5\}} = (24, 54)$ .

В полученном массиве на втором шаге вновь сортируются подмассивы из отстоящих на 3 позиции элементов.

Процесс завершается обычной сортировкой вставками получившегося списка. На рисунке представлена схема для рассмотренного примера.

|                            |   |            |
|----------------------------|---|------------|
| Исходный массив            | 32 95 16 82 24 66 35 19 75 54 40 43 93 68 |            |
| После сортировки с шагом 5 | 32 35 16 68 24 40 43 19 75 54 66 95 93 82 | 6 обменов  |
| После сортировки с шагом 3 | 32 19 16 43 24 40 54 35 75 68 66 95 93 82 | 5 обменов  |
| После сортировки с шагом 1 | 16 19 24 32 35 40 43 54 66 68 75 82 93 95 | 15 обменов |

Рисунок 1. Сортировка Шелла

## Схема распараллеливания

Для распараллеливания сортировки используется слияние «Разделяй и властвуй». Идея слияния по алгоритму «Разделяй и властвуй» заключается в разбиении массивов на участки, которые можно слить независимо. В первом массиве выбирается центральный элемент  $x$  (он разбивает массив на две равные половины), а во втором массиве с помощью бинарного поиска находится позиция наибольшего элемента меньшего  $x$  (позиция этого элемента разбивает второй массив на две части). После такого разбиения первые и вторые половины массивов могут сливаться независимо, т.к. в первых половинах находятся элементы меньше элемента  $x$ , а во второй – большие (Рисунок 2). Для слияния двух массивов несколькими потоками можно в первом массиве выбрать несколько ведущих элементов, разделив его на равные порции, а во втором массиве найти соответствующие подмассивы. Каждый поток получит свои порции на обработку. Эффективность такого слияния во многом зависит от того, насколько равномерно произошло «разделение» второго массива.



Рисунок 2. Слияние "Разделяй и властвуй"

Массив данных разбивается на равные части. Если это невозможно, остаток распределяется равномерно между потоками. Каждый поток сортирует свою часть независимо. Затем каждый второй поток, бинарным поиском находит позицию наибольшего элемента меньшего первого элемента в полученной части массива из 1-го потока и выполняет слияние. Далее действия повторяются для всех остальных потоков, кратных степеням двойки пока массив не сольётся полностью

### OpenMP

В функции, реализующей слияние "Разделяй и властвуй", для распараллеливания необходимо внести минимальные изменения. Достаточно добавить директиву `#pragma omp parallel sections`, которая распределяет вычисления для отдельных фрагментов кода, которые, в свою очередь, выделяются при помощи директивы `#pragma omp section`. Стоит отметить, что каждая секция выполняется в отдельном потоке и завершение директивы по умолчанию синхронизируется.

```
void ShellMerge(int* arr, int* res, uint32_t size, uint32_t amountOfThr)
```

```

{

    if (amountOfThr == 1) {
        ShellSort(arr, size);
    }
    else if (amountOfThr > 1) {
#pragma omp parallel sections
    {
#pragma omp section
        ShellMerge(arr, res, size / 2, amountOfThr / 2);
#pragma omp section
        ShellMerge(arr + size / 2, res + size / 2, size - size / 2,
amountOfThr - amountOfThr / 2);

    }
    MergeArrays(arr, arr + size / 2, res, size / 2, size - size / 2);
    std::copy(res, res + size, arr);
    }
}

```

Происходящие при этом действия подробно изложены в подразделе "Схема распараллеливания".

## ТВВ

Аналогично реализации нашей задачи с помощью технологии параллельного программирования OpenMP, необходимо произвести распараллеливание функции, реализующей слияние "Разделяй и властвуй".

Для этого создается объект класса `tbb::task_group`, с помощью которого мы вызываем метод `run`, запускающий первую задачу. Таким же образом мы запускаем вторую задачу, а затем с помощью метода `wait` дожидаемся завершения выполнения задач.

```

void ShellMerge(int* arr, int* res, uint32_t size, int32_t numOfThr)
{
    if (numOfThr == 1) {
        ShellSort(arr, size);
    }
    else if (numOfThr > 1) {
        tbb::task_group tg;
        tg.run([&arr, &res, size, numOfThr] {
            ShellMerge(arr, res, size / 2, numOfThr / 2);

```



```

    });
    tg.run([&arr, &res, size, numOfThr] {
        ShellMerge(arr + size / 2, res + size / 2, size - size / 2, numOfThr
- numOfThr / 2);
    });
    tg.wait();
}
MergeArrays(arr, arr + size / 2, res, size / 2, size - size / 2);
std::copy(res, res + size, arr);
}

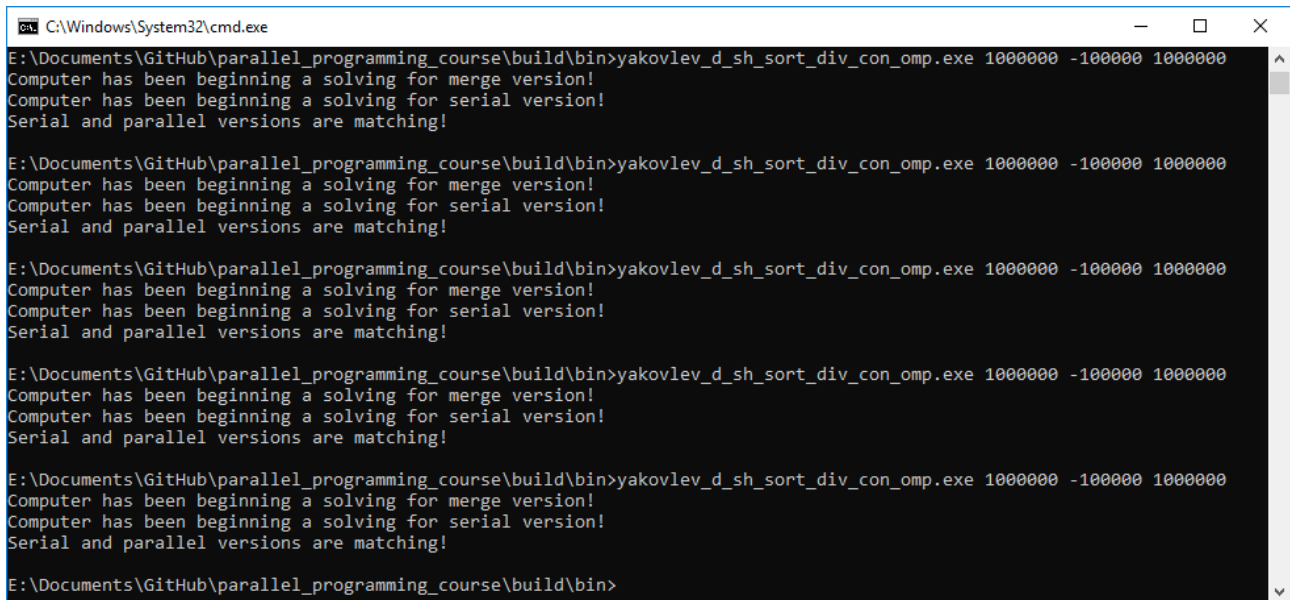
```

Происходящие при этом действия подробно изложены в подразделе "Схема распараллеливания".

## Руководство пользователя

Для запуска последовательной версии программы в консоли необходимо ввести следующую команду:

<путь\_до\_исполняемого\_файла>            <размер\_массива>            <начало\_диапазона>  
<конец\_диапазона>



```
C:\Windows\System32\cmd.exe
E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_omp.exe 1000000 -100000 1000000
Computer has been beginning a solving for merge version!
Computer has been beginning a solving for serial version!
Serial and parallel versions are matching!

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_omp.exe 1000000 -100000 1000000
Computer has been beginning a solving for merge version!
Computer has been beginning a solving for serial version!
Serial and parallel versions are matching!

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_omp.exe 1000000 -100000 1000000
Computer has been beginning a solving for merge version!
Computer has been beginning a solving for serial version!
Serial and parallel versions are matching!

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_omp.exe 1000000 -100000 1000000
Computer has been beginning a solving for merge version!
Computer has been beginning a solving for serial version!
Serial and parallel versions are matching!

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_omp.exe 1000000 -100000 1000000
Computer has been beginning a solving for merge version!
Computer has been beginning a solving for serial version!
Serial and parallel versions are matching!

E:\Documents\GitHub\parallel_programming_course\build\bin>
```

Рисунок 3. Интерфейс последовательной версии.

Для запуска параллельной версии программы в консоли необходимо ввести следующую команду:

<путь\_до\_исполняемого\_файла>            <количество\_потоков>            <размер\_массива>  
<начало\_диапазона> <конец\_диапазона>

```
C:\Windows\System32\cmd.exe
E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_task2_omp.exe 4 1000000 -1000000 1000000
Computer has been beginning a solving for merge version!
Time of work parallel version: 0.211145 sec.
Computer has been beginning a solving for serial version!
Time of Work serial version: 0.668068 sec.
Serial and parallel versions are matching!
Speed up: 3.16403

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_task2_omp.exe 4 1000000 -1000000 1000000
Computer has been beginning a solving for merge version!
Time of work parallel version: 0.225772 sec.
Computer has been beginning a solving for serial version!
Time of Work serial version: 0.675749 sec.
Serial and parallel versions are matching!
Speed up: 2.99305

E:\Documents\GitHub\parallel_programming_course\build\bin>yakovlev_d_sh_sort_div_con_task2_omp.exe 4 1000000 -1000000 1000000
Computer has been beginning a solving for merge version!
Time of work parallel version: 0.226509 sec.
Computer has been beginning a solving for serial version!
Time of Work serial version: 0.745933 sec.
Serial and parallel versions are matching!
Speed up: 3.29317

E:\Documents\GitHub\parallel_programming_course\build\bin>
```

Рисунок 4. Интерфейс параллельной версии.

## **Подтверждение корректности**

Для подтверждения корректности в программе реализована функция сравнения буферов, полученных в результате последовательной и параллельной версий алгоритма. В случае, если мы не получаем одинаковые результаты, выводится сообщение: “Serial and parallel versions are matching!”. В противном случае, на экране будет распечатано: “Serial and parallel versions are not matching!”

## Результаты экспериментов

Одной из задач данной лабораторной работы было сравнение времени работы параллельных и последовательной версий сортировки одного и того же массива. В случае последовательной версии для сортировки массива используется алгоритм сортировки Шелла, сложность которого составляет  $O(n^2)$  на случайных данных.

Были проведены следующие эксперименты, на основании которых нужно сделать вывод об эффективности параллельной программы. Все результаты занесены в таблицы, также построены графики на основании результатов. Время вычисляется в секундах.

Характеристики персонального компьютера:

1. Процессор Intel Core i7-860 2.7Ghz (4 ядра)
2. Оперативная память 4 Гб 1600 МГц DDR3

Входной массив  $2 \cdot 10^7$  элементов типа double в диапазоне от  $-10^8$  до  $10^8$ .

| Количество потоков            | 2     | 4     | 8     | 16    |
|-------------------------------|-------|-------|-------|-------|
| Последовательная версия, с    | 8.364 | 8.826 | 8.924 | 8.558 |
| Параллельная версия openMP, с | 5.297 | 4.167 | 3.433 | 2.938 |
| Эффективность                 | 1.579 | 2.118 | 2.599 | 2.912 |

Таблица 1. Результаты работы OpenMP

| Количество потоков         | 2     | 4     | 8     | 16    |
|----------------------------|-------|-------|-------|-------|
| Последовательная версия, с | 8,827 | 8,771 | 8,725 | 8,932 |
| Параллельная версия ТВВ, с | 5,202 | 3,395 | 3,150 | 3.094 |
| Эффективность              | 1,696 | 2,217 | 2,769 | 2.886 |

Таблица 2. Результаты работы ТВВ

Входной массив 1000 элементов типа double в диапазоне от  $-10^8$  до  $10^8$ .

|                             |        |        |        |        |
|-----------------------------|--------|--------|--------|--------|
| Количество потоков          | 2      | 4      | 8      | 16     |
| Последовательная версия, мс | 0,0734 | 0.0816 | 0.0093 | 0,0788 |
| Параллельная версия ТВВ, мс | 0.144  | 0,344  | 0.636  | 301    |
| Эффективность               | 0.241  | 0.0536 | 0.0146 | 0,0021 |

Таблица 3. Результаты работы OpenMP

|                             |       |        |        |        |
|-----------------------------|-------|--------|--------|--------|
| Количество потоков          | 2     | 4      | 8      | 16     |
| Последовательная версия, мс | 0,112 | 0,0097 | 0,0097 | 0,0830 |
| Параллельная версия ТВВ, мс | 0,24  | 0,0340 | 0,402  | 0,301  |
| Эффективность               | 0.462 | 0.287  | 0,235  | 0,275  |

Таблица 4. Результаты работы ТВВ

## Выводы из результатов

На основе проведенных экспериментов можно сделать следующий вывод:

1. Использованное распараллеливание более эффективно при большем числе потоков;
2. Целесообразно использовать параллельные версии на массивах больших размеров, так как в обратном случае это неэффективно, поскольку накладные расходы на работу потоков сравнимы по времени с вычислительной нагрузкой на них.
3. Сравнив версию TBV и OpenMP, нетрудно заметить, что результаты практически одинаковы. Это может быть объяснено схожестью алгоритмов распараллеливания.

## Заключение

Поставленная задача полностью выполнена. Были реализованы и протестированы последовательный и параллельные версии алгоритмов сортировки Шелла. В параллельном алгоритме слияние выполнялось по принципу «Разделяй и властвуй». По данным экспериментов видно, что время работы сортировки в OpenMP-версии оказалась схожа с версией в ТВВ. Это может быть объяснено схожестью алгоритмов распараллеливания. Однако под разные задачи, стоит задуматься о тех или иных преимуществах каждой из библиотек.

Но в случаях малых объемов данных возникает ситуация, при которой параллельная реализация дает результат по времени хуже, чем последовательная версия. Поэтому для таких задач, в которых требуется небольшой объем данных, лучше отказаться от параллельной реализации программы.



## Литература

1. Гергель, В.П. Параллельное программирование с использованием OpenMP // Нижегородский государственный университет им. Н.И. Лобачевского / Национальный исследовательский университет. – 2014. №1. – с. 44.
2. Сысоев, А.В. Параллельное программирование // Параллельное программирование с использованием OpenMP // Нижегородский государственный университет им. Н.И. Лобачевского / Национальный исследовательский университет. – 2016. №2. – с. 27.
3. Мееров И. Б. Инструменты параллельного программирования для систем с общей памятью. Библиотека Intel Threading Building Blocks – краткое описание / Сысоев А.В., Сиднев А.А. // Нижегородский государственный университет им. Н.И. Лобачевского. – 2009. – с. 172.
4. Седжвик, Р. Алгоритмы на C++ // «Национальный Открытый Университет «ИНТУИТ». – Режим доступа: <http://www.intuit.ru/studies/courses/12181/1174/lecture/25257?page=5>