МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

Отчёт по лабораторной работе

по курсу "Параллельное программирование"

**"Сортировка Шелла с простым слиянием"**

**Выполнил:**

студент гр. 381506-3

Львова А.Д.

**Проверил:**

Доцент кафедры МОСТ

Сысоев. А.В.

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc515071921)

[О роли сортировок 3](#_Toc515071922)

[Постановка задачи 4](#_Toc515071923)

[Метод решения 5](#_Toc515071924)

[Подход к решению 5](#_Toc515071925)

[Схема распараллеливания 6](#_Toc515071926)

[Описание программной реализации 8](#_Toc515071927)

[Руководство пользователя 9](#_Toc515071928)

[Подтверждение корректности 10](#_Toc515071929)

[Результаты экспериментов по оценке масштабируемости 11](#_Toc515071930)

[Заключение 13](#_Toc515071931)

# Введение

## О роли сортировок

**"Но мы не успеем просмотреть все номера автомобилей" — возразил Дрейк.  
"А нам и не нужно этого делать. Пол. Мы просто расположим их по порядку и поищем одинаковые."**

**- Перри Мейсон "The Case of the Angry Mourner" (1951)**

Сортировки занимают особую нишу в программировании. Алгоритмы сортировок необходимы для решения таких задач, как группирование (когда нужно собрать вместе все элементы с одинаковыми значениями некоторого признака), поиск общих элементов в двух или более массивах, поиск информации по значениям ключей, и др.

По оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины времени. Исходя из этих статистических данных можно заключить, что либо сортировка имеет много важных применений, либо ее часто пользуются без нужды, либо применяются в основном неэффективные алгоритмы сортировки. По-видимому, каждое из приведенных предположений содержит долю истины.

Во всяком случае ясно, что сортировка заслуживает серьезного изучения с точки зрения ее практического использования.

На протяжении последних 60 лет было придумано большое количество алгоритмов сортировок, которые справляются в разы быстрее прежних. Но вместе с тем, объёмы данных, как и характеристики вычислительных машин, выросли в разы. А значит, нужно изобретать все более эффективные алгоритмы, сортирующие данные больших размеров. Для подобных задач используются параллельные вычисления.

### 

# Постановка задачи

Разработать и реализовать программу для сортировки массива данных методом Шелла в последовательной и параллельных версиях. Параллельные алгоритмы реализовать с помощью стандарта OpenMP и библиотеки TBB.

Выполнение задачи включает:

1. Освоение темы (постановка задачи)
2. Изучение метода решения
3. Разработку схемы параллельных вычислений
4. Реализацию программы
5. Проведение численных экспериментов с анализом масштабируемости
6. Подготовку отчета с анализом результатов экспериментов

Данная программа должна поддерживать следующие возможности:

* Генерация (чтение из файла) данных для задач с различными параметрами. Ввод размерности массива данных и количество использованных узлов в вычислениях вводятся пользователем;
* Подсчет времени исполнения последовательной и параллельной части, а также ускорения при выполнении параллельного алгоритма;
* Подтверждение корректности выполнения;
* Демонстрация результатов работы алгоритмов на малых размерностях данных.

# Метод решения

## Подход к решению

**Сортировка Шелла**  (*Shell sort*) — 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 {\displaystyle d}d. После этого процедура повторяется для некоторых меньших значений {\displaystyle d}d, а завершается сортировка Шелла упорядочиванием элементов при {\displaystyle d=1}d (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(n²), что хуже, чем худшее гарантированное время для сортировки Шелла

пример

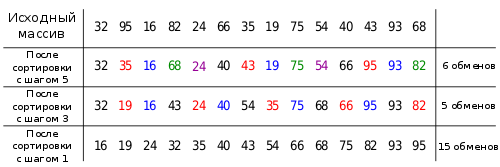


Рис 1. Сортировка Шелла

# Схема распараллеливания

1. Рассчитывается количество элементов, которое будет обработано каждым потоком, кроме последнего. Назовем это количество, условно, длиной группы = длина массива / количество потоков. Последний поток обрабатывает количество элементов равное длине группы + остаток от деления длины массива на количество потоков.

2. Параллельно сортируем массив методом Шелла.

Метод на вход принимает следующие параметры: массив, смещение от начала массива, длину сортируемой части.

N/6

N/6

N/6

N/6

N/6

mod

N/6

N/6 + mod

N/6

N/6

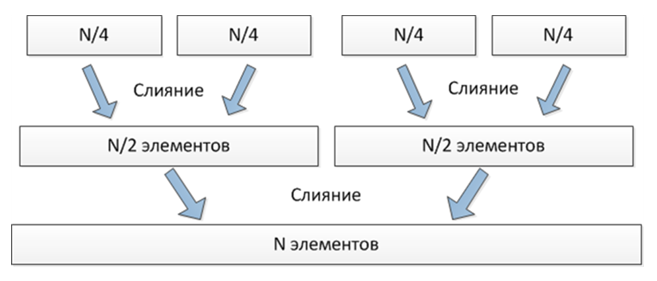
N/6

N/6

N/6

*Рис 2. Параллельная сортировка Шелла на 6 потоках*

3. Параллельно сливаем отсортированные части массива, изменяя количество потоков на каждом уровне сливания.



*Рис 3. Простое сливание*

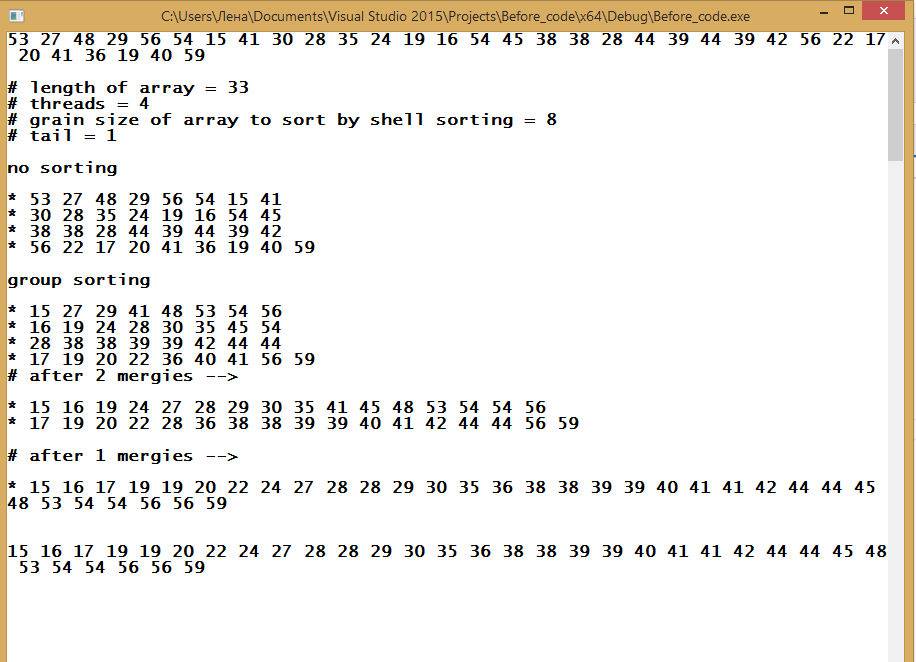


Рис 4. пример работы программы

На Рис 4. представлен пример работы программы сортировки массива данных, длина которого равна 33 элементам. Количество потоков 4. Первые три потока сортируют по 8 элементов, последний - 8+остаток = 9. После того, как элементы в группах отсортированы, начинается параллельное сливание и меняется количество потоков. Первые 2 части массива сливаются первым потоком, последние две - вторым. На экран выводится результат сливания. Уменьшаем количество потоков. Сливание происходит до тех пор, пока количество потоков не станет равное нулю.

# Описание программной реализации

sol.cpp включает 3 метода:

* void insert\_sort(double\* arr, int i, int length, int half) - метод сортировки вставками
* void shell\_sort\_with\_a\_shift(double\* array, int shift, int length) - метод сортировки Шелла, при котором сортируется только часть массива array, т.е. элементы массива array[shift, shift + length]
* void shell\_sort(double\* array, int length) - метод сортировки Шелла, при котором сортируется весь массив array длинной length

tbb.cpp и openmp.cpp включают по 2 метода:

* void merge\_in\_the\_same\_array(double\* array, int m, int n, int it1, int it2) - метод слияния двух частей в одном массиве, где сливаются части array[it1, it1+m] и

array[it2, i2 + m]

* void merge\_2\_arrays(double\* A, double\*B, int size\_A, int size\_B) - метод слияния двух массивов размерами size\_A и size\_B

# Руководство пользователя

1. Генерация данных
2. Ввод данных вручную
   1. Записать данные в файл "massiv.txt"
   2. Запустить "typer.exe"
   3. Перейти к шагу II
3. Генератор данных
   1. Запустить "generator.exe" с параметром [номер теста]
   2. Перейти к шагу II
4. Запустить "before\_code.exe" с параметром [количество потоков]
5. Запустить "checker.exe" для проверки на корректность расчетов
6. Запустить "viewer.exe" для вывода на экран времени выполнения сортировки и размера массива

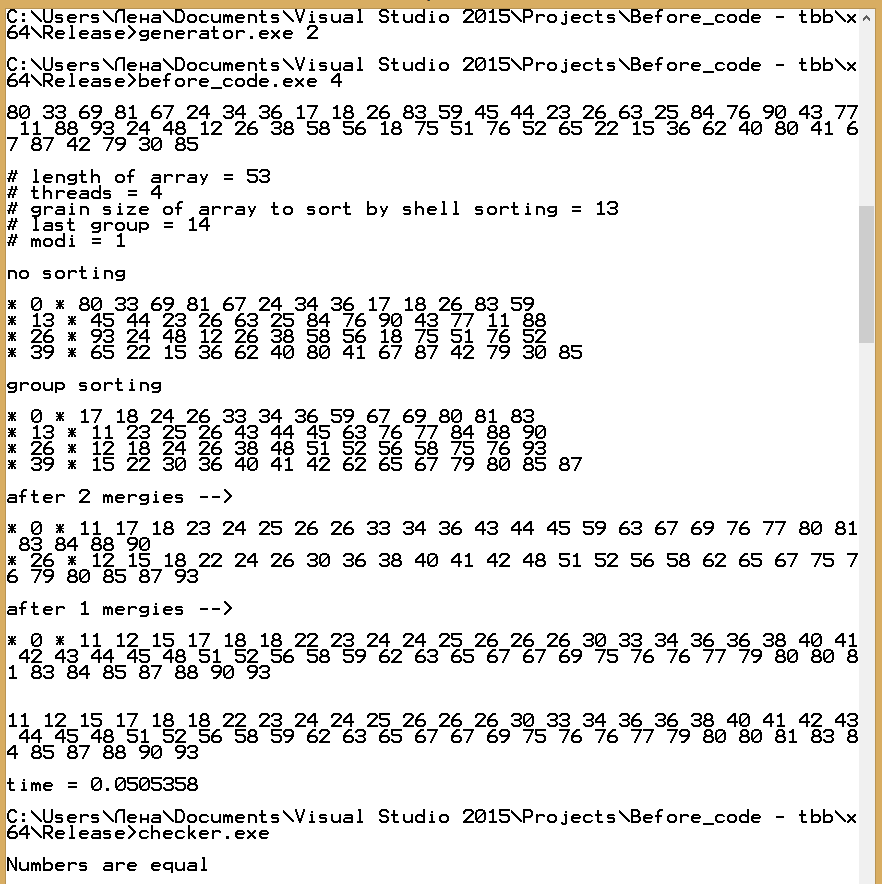


Рис 5. Пример работы программы

Программs выводят:

* данные о размерности, количестве потоков, длину массива в группе, остаток
* проверку на совпадение результатов двух версий
* начальные, промежуточные и итоговые результаты работы программы "before\_code.exe" при размерностях массива меньших 150 (для наглядности)
* время работы

# Подтверждение корректности

За проверку на совпадение результатов сортировки отвечает "cheker.exe"

Алгоритм работы:

* Открываем файл входных данных, файл выходных данных и ответ участника
* Считываем время работы программы участника
* Считываем размерность массива
* Выделяем память для массива ответа жюри и ответа участника
* Считываем массив участника
* Считываем исходные данные для проверки стандартной функцией
* сортируем исходные данные сортировкой std
* Сравниваем ответы
* Записываем время в правильной размерности ( интервалы по 100 нс = 10 ^ (-7) сек)

# 

# Результаты экспериментов по оценке масштабируемости

Ускорение, получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений, определяется величиной:

где – количество входных данных задачи.

– время работы алгоритма на процессах (измеряется в секундах).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Количество процессов/ размерность | Последовательный алгоритм (время вып.) | OpenMP | | | | TBB | | | |
| 1 | 2 | 4 | 8 | 16 | 2 | 4 | 8 | 16 |
| 100 | 0,00000462 | 0.0000918 | 0.00018 | 0.00006 | 0.000043 | 0.000085 | 0.0000828 | 0.0000758 | 0.000147 |
| 1000 | 0,0000652 | 0.0189 | 0.02 | 0.0149 | 0.011 | 0.016 | 0.016 | 0.013 | 0.017 |
| 10000 | 0.0000812 | 0.19802 | 0.185 | 0.19 | 0.185 | 0.14 | 0.182 | 0.18 | 0.19 |
| 100000 | 0.00928 | 1.053 | 1.44 | 1.25 | 1.28 | 0.914 | 1.119 | 1.10 | 1.20 |
| 500000 | 0.0606 | 1.83 | 2.18 | 2.28 | 2.41 | 1.797 | 2.32 | 2.44 | 2.28 |
| 1000000 | 0.1388 | 2.11 | 3.20 | 2.63 | 3.03 | 1.87 | 2.99 | 2.94 | 2.567 |
| 2500000 | 0.396 | 1.96 | 2.80 | 3.27 | 3.19 | 1.9 | 2.86 | 2.83 | 2.944 |
| 5000000 | 0.8759 | 1.97 | 2.66 | 3.15 | 3.32 | 1.84 | 2.56 | 2.869 | 3.21 |

Таблица 1: Результаты экспериментов: зависимость ускорения от количества потоков, версии исполнения и размерности данных

Характеристики ПК:

* *процессор i5-4460S 2.90GHz*
* *оперативная память: 6 GB*
* *OS: windows 8.1*

График 1: Зависимость ускорения от количества параллельных процессов

и размерности при выполнении версии OpenMP

График 2: Зависимость ускорения от количества параллельных процессов

и размерности при выполнении версии TBB

# Заключение

В результате выполнения лабораторной работы была реализована сортировка Шелла с использованием технологий распараллеливания OpenMP и TBB.

За основу взят последовательный алгоритм сортировки Шелла и реализована его параллельная версия.

Проведены тесты алгоритма на данных разных размерностей, а так же на разном количестве процессов. Была доказана корректность полученного решения.

По графикам 1 и 2 видно, что для лучшего ускорения количество процессов нужно подбирать в зависимости от размерности массива. Дальнейшее увеличение процессов не влияет критически на ускорение времени выполнения сортировки.