МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Отчет по лабораторной работе**

**«Сортировка Шелла с простым слиянием»**

**Выполнил:** студент группы 381506-1

Цепляева Анастасия Андреевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Кустикова Валентина Дмитриевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

1. Постановка задачи 2

2. Последовательный алгоритм 2

3. Схема распараллеливания 2

4. Описание программной реализации 3

5. Подтверждение корректности 4

6. Результаты экспериментов по оценке масштабируемости 4

7. Выводы 8

Постановка задачи

Требуется разработать программу, которая производят сортировку массива посредством заданного алгоритма. Для этого разработать последовательный и параллельный алгоритмы сортировки Шелла, затем сравнить время выполнения на различном числе процессов (потоков) и определить ускорение.

Для распараллеливания использовать следующие инструменты:

1. стандарт OpenMP
2. библиотеку Threading Building Blocks

Последовательный алгоритм

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). В качестве последовательности d = {,..} я выбрала последовательность Пратта: все значения \*≤N/2, i,j∈ ℕ. В таком случае сложность алгоритма составляет O(N(logN)²).

Схема распараллеливания

Количеством потоков для распараллеливания должно быть число, которое является степенью двойки. Это необходимо для реализации простого слияния.

Сначала выполняется параллельная сортировка массива. Для каждого потока вычисляются параметры left и right, которые определяют *блок* массива, который сортирует данный поток обычной последовательной сортировкой Шелла, выделенной в отдельную функцию void shellsort (<указатель массив>, left, right).

Далее выполняется слияние отсортированных *блоков*. Идея простого слияния заключается в том, что один поток может выполнять слияние двух отсортированных подмассивов по классическому алгоритму (void simple\_merge (<…>). В моей реализации это сначала отсортированные отдельные *блоки* (в этом случае слияние *n* массивов могут выполнять *n*/2 параллельных потоков), затем блоки вдвое большего размера (слияние *n*/2 полученных массивов будут выполнять *n*/4 потоков) и т.д. (Рисунок 1.1)

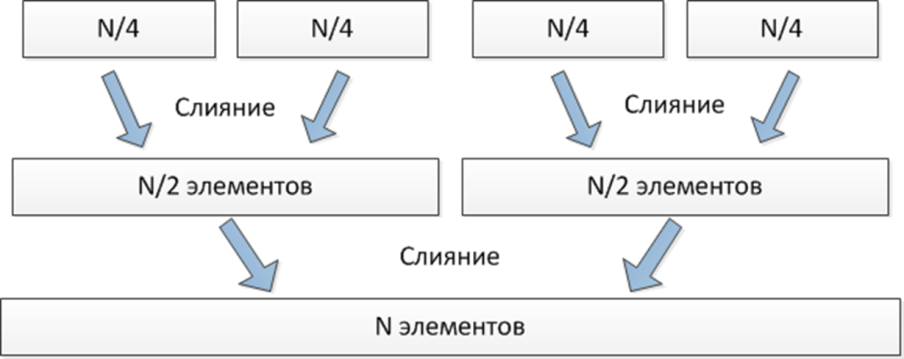


Рис.1 Схема слияния

В итоге получим отсортированный массив. Нужно отметить, что схема распараллеливания одинакова для OpenMP и TBB версий.

Описание программной реализации

Решение представляет собой набор следующих проектов:

1. Solver - решение задачи последовательным алгоритмом.
2. Generator - программа для генерации набора тестовых данных и их сохранение в бинарные файлы.

Через командную строку пользователь вводит номер теста, который соответствует индексу в массиве. Значения массива – числа, равные количеству элементов массива для сортировки. Далее этот массив заполняется случайными вещественными числами и записывается в бинарный файл.

1. Checker - программа для проверки корректности параллельных версий.
2. Omp - решение задачи параллельным алгоритмом, используя технологию OpenMP.
3. Tbb - решение задачи параллельным алгоритмом, используя технологию TBB.
4. Deserializer - считывание данных из бинарного файла в текстовый.
5. Serializer - считывание данных из текстового файла в бинарный.
6. Набор тестов (папка tests) для проверки корректности параллельных версий.

Подтверждение корректности

Для подтверждения корректности нужно запустить программу Checker, в которой с помощью стандартного метода сортировки qsort(<…>) получаем эталонный результат. Этот результат сравнивается с результатом, полученным с помощью программы Solver, omp или tbb. Результат сравнения записывается в файл result.txt.

Результаты экспериментов по оценке масштабируемости

Тесты по оценке масштабируемости выполнялись на следующей инфраструктуре:

* Процессор Pentium(R) Dual-Core CPU T4200 @ 2.00GHz 2.00 GHz
* ОЗУ 3GB
* 32-разрядная операционная система
* OS Windows 7

Ниже приведены значения времени исполнения (Таблица 1) и ускорения (Таблица 2) с использованием стандарта OpenMP, и времени исполнения (Таблица 3) и ускорения (Таблица 4) с использованием библиотеки TBB.

Таблица 1. Время выполнения OpenMP (с)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество**  **процессов**  **Размер массива** | **2** | **4** | **8** | **16** |
| **100** | 0,00054 | 0,00185 | 0,00180 | 0,00303 |
| **1000** | 0,00305 | 0,00529 | 0,00335 | 0,0021 |
| **5000** | 0,01518 | 0,0285 | 0,0186 | 0,05216 |
| **10000** | 0,02980 | 0,05258 | 0,0782 | 0,0396 |

Рис.2 График зависимости времени выполнения от количества потоков

Таблица 2. Ускорение параллельной реализации по сравнению с однопотоковой

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество**  **процессов**  **Размер массива** | **2** | **4** | **8** | **16** |
| **100** | 0,2203 | 0,0643 | 0,0661 | 0,0396 |
| **1000** | 0,2622 | 0,1525 | 0,2408 | 0,3842 |
| **5000** | 0,1982 | 0,1056 | 0,1618 | 0,0577 |
| **10000** | 0,2057 | 0,116 | 0,0782 | 0,154 |

Рис.3 График ускорения по сравнению с однопотоковой реализацией

Таблица 3. Время выполнения TBB (с)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество**  **процессов**  **Размер массива** | **2** | **4** | **8** | **16** |
| **100** | 0,00638 | 0,00698 | 0,06362 | 0,03601 |
| **1000** | 0,02811 | 0,0050 | 0,0226 | 0,0148 |
| **5000** | 0,12266 | 0,2486 | 0,5632 | 0,4794 |
| **10000** | 0,14133 | 0,0727 | 0,0583 | 0,0465 |

Рис.4. График зависимости времени выполнения от количества потоков

Таблица 4. Ускорение параллельной реализации по сравнению с однопотоковой

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество**  **процессов**  **Размер массива** | **2** | **4** | **8** | **16** |
| **100** | 0,1865 | 0,1704 | 0,187 | 0,33 |
| **1000** | 0,035 | 0,2 | 0,044 | 0,067 |
| **5000** | 0,1226 | 0,5229 | 0,235 | 0,027 |
| **10000** | 0,255 | 0,496 | 0,619 | 0,776 |

Рис.4 График ускорения по сравнению с однопотоковой реализацией

Выводы

В результате выполнения лабораторной работы я узнала, как работать с технологиями OpenMP и TBB.

При проведении экспериментов по оценке масштабируемости можно увидеть, что при малых значениях размера массива последовательная версия работает примерно так же, как параллельная. При увеличении размера массива появляется ускорение. При увеличении количества процессов время выполнения также уменьшается, но не всегда, это может зависеть от разных факторов, как от реализации алгоритма, так же и от аппаратного обеспечения. С моими алгоритмами и данными получилось, что OpenMP программа работает либо примерно так же, как и программа, написанная при помощи TBB, либо чуть быстрее.