МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Направление подготовки: «Фундаментальная информатика и
информационные технологии»

Отчёт по лабораторной работе

Поразрядная сортировка для целых чисел с простым слиянием

Выполнил:
Студент ИИТММ гр. 381606-1

Каганов Д.А.

Проверила:
доцент каф. МОСТ, ИИТММ,
кандидат технических наук

Сысоев А.В.

Нижний Новгород
2019 г.

Оглавление

Введение 3

Постановка задачи 4

Описание алгоритма 5

Описание схемы распараллеливания 6

Описание OpenMP–версии 7

Описание TBB–версии 9

Результаты экспериментов 10

OpenMP 10

TBB 10

Выводы из результатов 11

Заключение 12

Литература 13

Введение

Сортировка является одной из фундаментальных алгоритмических задач программирования. Алгоритмом сортировки называется алгоритм для упорядочения некоторого множества элементов. Обычно под алгоритмом сортировки подразумевают алгоритм упорядочивания множества элементов по возрастанию или убыванию.

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.

Метод LSD-сортировки просматривает байты в направлении справа налево, также этот метод называют методом поразрядной сортировки. Метод упорядочивает целые числа за 4 прохода по массиву данных и связано это с тем, что в памяти ЭВМ целое число занимает 4 байта (int). Реализация метода распределяющего подсчета должна быть устойчивой иначе выходной результат некорректен. О требовании устойчивости можно прочитать в источнике [1].

Постановка задачи

Необходимо реализовать программу, которая делит произвольный массив целых чисел на заданное число кусков, сортирует их по возрастанию, используя алгоритм поразрядной сортировки и в конечном итоге соединяет их обратно используя простое слияние.

Цель данной работы:

1. Реализовать последовательную и параллельную версию программы;
2. Оценить эффективность и масштабируемость данной программы на кластере (или многопоточный запуск на персональном компьютере).
3. Проанализировать полученные результаты и исходя из анализа данных сделать заключение

Описание алгоритма

Алгоритм работы последовательной версии программы делится на следующие части:

1. Генерация данных.
2. Сортировка.
3. Вывод результатов.

На этапе 1 генерируются входные данные программы в виде положительных и отрицательных целых чисел.

Этап 2 является основным этапом программы. В ходе него происходит сортировка входного массива данных с помощью алгоритма поразрядной LSD сортировки.

Метод LSD-сортировки упорядочивает трехбуквенные слова за три прохода (слева направо).

Файл сначала сортируется по последней букве (методом распределяющего подсчета), потом по средней букве, а затем по первой.

Данный метод будет работать только в том случае, если сортировка будет устойчивой. После того, как определена необходимость устойчивости, уже нетрудно доказать, что LSD-сортировка работает: мы знаем, что после сортировки ключей по i последним байтам (при соблюдении устойчивости) любые два ключа в файле упорядочены (по просмотренным на текущий момент байтам) — либо потому, что первые из i последних байтов отличны друг от друга (и упорядочены по этому байту), либо потому, что первые из i последних байтов совпадают, (и они уже упорядочены в силу устойчивости).

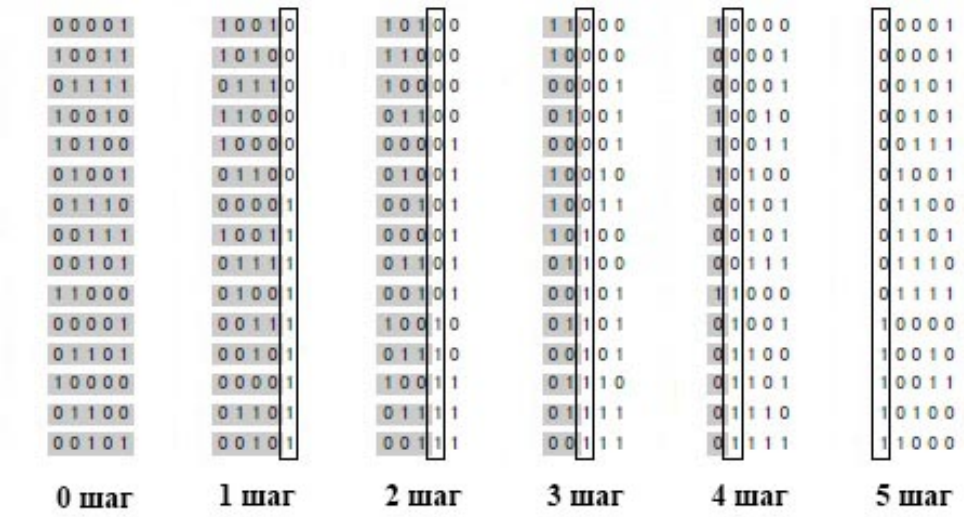
Алгоритм поразрядной LSD сортировки[1]:

```
void lsdSort(unsigned int* A, unsigned int count) {
    unsigned int* B = new unsigned int[count];
    unsigned int index[4][256] = { {0} };
    unsigned int x, y, z;
    for (unsigned int i = 0; i < count; i++) {
        x = A[i];
        for (int j = 0; j < 4; j++) {
            index[j][static_cast<int>(x & 0xff)]++;
            x >>= 8;
        }
    }
```

[1] Целые числа сортируются (методом распределяющего подсчета) по байтам справа налево.

```
    for (int i = 0; i < 4; i++) {
        y = 0;
        for (int j = 0; j < 256; j++) {
            z = index[i][j];
            index[i][j] = y;
            y += z;
        }
    }
    for (int i = 0; i < 4; i++) {
        for (unsigned int j = 0; j < count; j++) {
            x = A[j];
            y = static_cast<int>(x >> (i << 3)) & 0xff;
            B[index[i][y]++] = x;
        }
        std::swap(A, B);
    }
    delete[] B;
}
```

Пример работы алгоритма поразрядной LSD сортировки на некотором наборе данных:



Этап 3 работы программы является заключительным. В случае положительного ответа функции, проверяющей отсортирован ли массив, программа выводит данные пользователя.

Описание схемы распараллеливания

Алгоритм работы параллельной версии программы делится на следующие части:

1. Генерация данных.
2. Сортировка.
3. Каскадное слияние.
4. Вывод результатов.

Этапы 1, 4 аналогичны этапам 1, 3 последовательной версии.

Этап 2 параллельной версии отличается от аналогичного этапа 2 последовательной версии программы. Теперь, с помощью поразрядной сортировки каждым потоком сортируются независимые части массива.

Этап 3 необходим для того, чтобы из независимых отсортированных частей получить на выходе единый отсортированный массив. Каскадное слияние выполняется потоками по следующей схеме[2]:

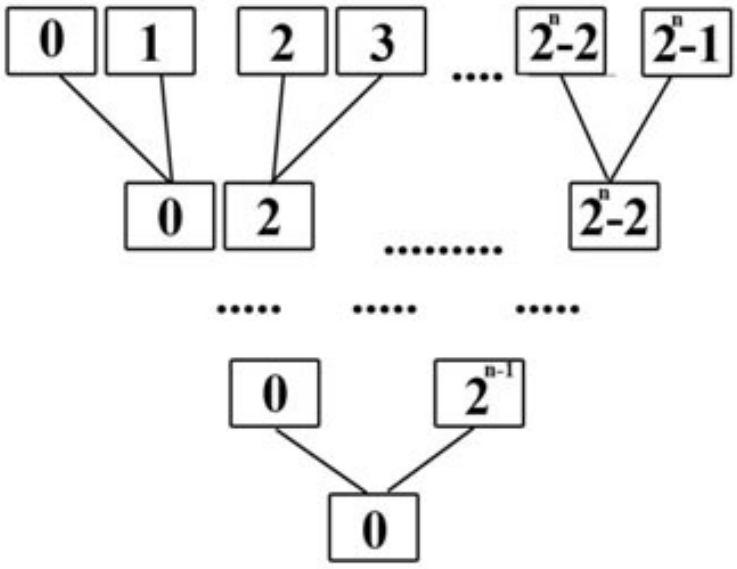


Рисунок 1. Схема каскадного слияния.

Корректность программы проверяется как полное совпадение выходных данных последовательной и параллельных версий на одном и том же наборе данных.

Описание OpenMP–версии

В OpenMP–версии, как говорилось ранее, все вычисления распределяются между потоками. Деление массива на куски происходит непосредственно внутри функции, с запоминанием размера каждого куска и размером остатка. Далее каждый кусок сортируется поразрядной LSD сортировкой, алгоритм которой описан в разделе “Последовательная версия”. После того как все куски массива будут отсортированы произойдёт слияние с помощью вспомогательных массивов, хранящих в себе адреса левых и правых концов каждых кусков. По завершению алгоритма функция возвращает

[2] Для наибольшей эффективности и простоты реализации каскадного слияния в программе используется количество потоков, являющихся степенью числа 2

отсортированный массив и освобождает память, выделенную ранее для вспомогательных массивов.

Для распараллеливания данного алгоритма использовались директивы `parallel for`, предназначенные для распараллеливания циклов, с параметром `schedule(static, 1)`, который делит итерации на блоки размером 1 и статически разделит их между потоками. Выбор сценария для данной сортировки не играет роли, так как поразрядная сортировка является быстрым алгоритмом с малым объемом вычислений на каждом шаге.

Алгоритм поразрядной LSD сортировки с использованием openMP:

```
unsigned int* radixSortParallel(unsigned int* A, unsigned int arrSize, int size, int mergeNum, int numThreads) {
    // pointless copy array A
    unsigned int* R = new unsigned int[arrSize];
    for (unsigned int i = 0; i < arrSize; i++)
        R[i] = A[i];
    // auxiliary array containing size of "size"
    int* sizeArr = new int[mergeNum];
    int remainder = arrSize % size;
    for (int i = 0; i < mergeNum; i++) {
        sizeArr[i] = size;
    }
    if (arrSize % mergeNum != 0) {
        sizeArr[mergeNum - 1] = remainder;
    }

    omp_set_num_threads(numThreads);
    #pragma omp parallel for schedule(static, 1)
    for (int i = 0; i < mergeNum; i++)
        lsdSort(R + i * size, sizeArr[i]);

    int counter = static_cast<int>(std::log(mergeNum) / std::log(2));

    int block = 0;
    for (int c = 0; c < counter; c++) {
        if (c != 0) {
            size = size * 2;
        }
        block = static_cast<int>(mergeNum / pow(2, c) / 2);
        int* r1 = new int[block];
        int* l1 = new int[block];
        int* r2 = new int[block];
        int* l2 = new int[block];
        for (int j = 0; j < block; j++) {
            l1[j] = j * size * 2;
            r1[j] = j * size * 2 + size - 1;
            l2[j] = j * size * 2 + size;
            r2[j] = j * size * 2 + size * 2 - 1;;
        }
        r2[block - 1] = arrSize - 1;
        #pragma omp parallel for schedule(static, 1)
        for (int i = 0; i < block; i++) {
            unsigned int* tmp = new unsigned int[r1[i] - l1[i] + 1 + r2[i] - l2[i] + 1];
            tmp = sortMerge(R + l1[i], r1[i] - l1[i] + 1, R + l2[i], r2[i] - l2[i] + 1);
            int j = l1[i], g = 0;
            while (j <= r2[i]) {
                R[j] = tmp[g++];
                j++;
            }
            delete[] tmp;
        }
    }
    return R;
    delete[] R;
    delete[] sizeArr;
}
```

Описание TBB–версии

Для распараллеливания алгоритма сортировки в TBB–версии используется шаблонная функция `tbb::parallel_for` с использованием одномерного итерационного пространства с диапазоном в виде полуинтервала `[begin, end)`. Алгоритм деления массива на куски и последующего слияния аналогичен алгоритму из openMP–версии.

Алгоритм поразрядной LSD сортировки с использованием TBB:

```
unsigned int* radixSortParallel(unsigned int* A, unsigned int arrSize, int size, int mergeNum, int numThreads) {
    // pointless copy array A
    tbb::task_scheduler_init init(numThreads);
    unsigned int* R = new unsigned int[arrSize];
    for (unsigned int i = 0; i < arrSize; i++)
        R[i] = A[i];
    // auxiliary array containing size of "size"
    int* sizeArr = new int[mergeNum];
    int remainder = arrSize % size;
    for (int i = 0; i < mergeNum; i++) {
        sizeArr[i] = size;
    }
    if (arrSize % mergeNum != 0) {
        sizeArr[mergeNum - 1] = remainder;
    }

    tbb::parallel_for(tbb::blocked_range<int>(0, mergeNum), [=, &R](const tbb::blocked_range<int> &thrds) {
        for (int i = thrds.begin(); i != thrds.end(); i++)
            lsdSort(R + i * size, sizeArr[i]);
    });
    init.terminate();

    int counter = static_cast<int>(std::log(mergeNum) / std::log(2));

    int block = 0;
    for (int c = 0; c < counter; c++) {
        if (c != 0) {
            size = size * 2;
        }
        block = static_cast<int>(mergeNum / pow(2, c) / 2);
        int* r1 = new int[block];
        int* l1 = new int[block];
        int* r2 = new int[block];
        int* l2 = new int[block];
        for (int j = 0; j < block; j++) {
            l1[j] = j * size * 2;
            r1[j] = j * size * 2 + size - 1;
            l2[j] = j * size * 2 + size;
            r2[j] = j * size * 2 + size * 2 - 1;;
        }
        r2[block - 1] = arrSize - 1;
        init.initialize(block);
        tbb::parallel_for(tbb::blocked_range<int>(0, block), [=, &R](const tbb::blocked_range<int> &thrds) {
            for (int i = thrds.begin(); i != thrds.end(); i++) {
                unsigned int* tmp = new unsigned int[r1[i] - l1[i] + 1 + r2[i] - l2[i] + 1];
                tmp = sortMerge(R + l1[i], r1[i] - l1[i] + 1, R + l2[i], r2[i] - l2[i] + 1);
                int j = l1[i], g = 0;
                while (j <= r2[i]) {
                    R[j] = tmp[g++];
                    j++;
                }
                delete[] tmp;
            }});
        init.terminate();
    }
    return R;
    delete[] R;
    delete[] sizeArr;
}
```

Результаты экспериментов

Эксперименты проводились на персональном компьютере с использованием многопоточного запуска.

Характеристики персонального компьютера:

1. Процессор Intel Core i5-5250U 1.6Ghz (2 ядра)
2. Оперативная память 4 Гб 1600 Мгц DDR3L

Входной массив 10^7 элементов целочисленного типа (int).

OpenMP

Количество потоков	2	4	8	16	32
Время последовательной части, с	0.238	0.278	0.364	0.527	0.913
Время параллельной части, с	0.196	0.238	0.302	0.359	0.484
Эффективность	0.607	0.2917	0.1502	0.091	0.058
Общее время, с	0,435	0.518	0.667	0.886	1.398

Таблица 1. Результаты работы программы с использованием openMP

TBB

Количество потоков	2	4	8	16	32
Время последовательной части, с	0.218	0.291	0.395	0.534	0.866
Время параллельной части, с	0.182	0.288	0.267	0.44	0.674
Эффективность	0.5993	0.2531	0.1853	0.0758	0.0401
Общее время, с	0,401	0.58	0.662	0.974	1.54

Таблица 2. Результаты работы программы с использованием TBB

Выводы из результатов

Данные результаты можно пояснить тем, что поразрядная сортировка является быстрым алгоритмом (сложность O(n)) с малым объемом вычислений на каждом шаге. Чтобы время алгоритма было достаточным для оценки ускорения нужно работать с огромным массивом данных. Из времени последовательной части видно, как много времени занимает работа с памятью (выделение, удаление). После того, как каждый поток сортирует свою независимую часть массива, происходит каскадное слияние на. В силу описанных выше причин данная операция может занимать по времени до 1 секунды.

Итог: малый объем вычислений на каждой итерации алгоритма и необходимость производить операции слияния на потоках (пусть даже каскадное) в сумме дают ускорение ниже, чем количество потоков (или их совокупности).

Заключение

В ходе работы был реализован последовательный и параллельный алгоритм поразрядной LSD сортировки. Была проведена серия экспериментов, которая позволяет оценить эффективность использования каждой из технологий и их совокупности.

Благодаря особенностям технологий openMP и TBB, при написании параллельных версий программ, не приходилось перерабатывать существенную часть программного кода. Также данные библиотеки позволяют осуществлять поэтапную разработку параллельных программ.

По результатам эксперимента эффективность распараллеливания алгоритма поразрядной LSD сортировки в openMP–версии оказалась схожа с эффективностью в TBB–версии, что может быть объяснено идентичностью распараллеливания алгоритма в обоих версиях программы и быстрой работой алгоритма сортировки на одном потоке.

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 Загл. с экрана.