Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 4

Дисциплина Анализ алгоритмов

Тема Параллельная сортировка слиянием

Студент Искакова К.М.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Сортировки используются в самом широком спектре задач, включая обработку коммерческих, сейсмических, космических и прочих данных [1]. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными и т. п.

Сортировка применяется во всех без исключения областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным.

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике.

В данной лабораторной работе рассматривается алгоритм сортировки слиянием и его параллельная версия

Задачи работы:

1) рассмотрение алгоритма сортировки слиянием;
2) проведение сравнительного анализа алгоритма сортировки слиянием и его параллельной версией;
3) определение зависимости времени работы алгоритма от числа потоков исполнения и размера массивов.

1 | Аналитическая часть

Сортировка массива — одна из самых частых операций над массивом. Алгоритмы реализуют упорядочивание 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.

1.1. Описание алгоритма сортировки слиянием

Сортировка слиянием (англ. Merge sort) — алгоритм сортировки, использующий $O(n)$ дополнительной памяти и работающий за $O(n * log(n))$ времени. Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

1) если в рассматриваемом массиве один элемент, то он уже отсортирован – алгоритм завершает работу, иначе – массив разбивается на две части, которые сортируются рекурсивно;

2) после сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.

На Рис. 1.1 изображен пример работы алгоритма сортировки слиянием.

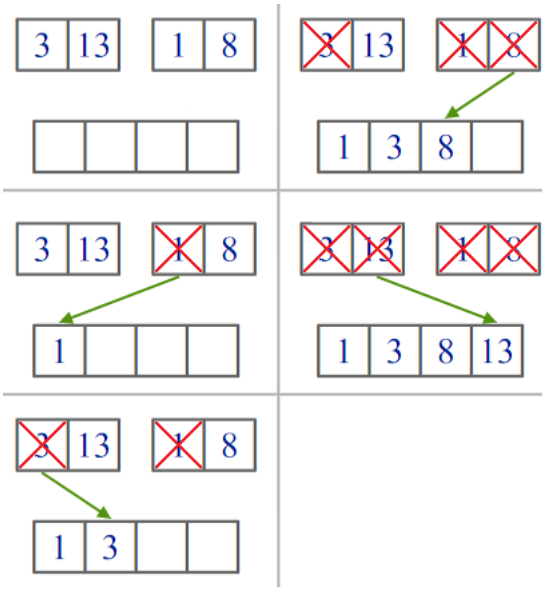

Рис. 1.1. Пример работы алгоритма сортировки слиянием

1.2. Многопоточность

Поток выполнения — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

На одном процессоре многопоточность обычно происходит путём временного мультиплексирования (как и в случае многозадачности): процессор переключается между разными потоками выполнения. Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное. В многопроцессорных и многоядерных системах потоки или задачи могут реально выполняться одновременно, при этом каждый процессор или ядро обрабатывает отдельный поток или задачу.

Потоки возникли в операционных системах как средство распараллеливания вычислений.

Параллельное выполнение нескольких работ в рамках одного интерактивного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещать набор нового текста с такими продолжительными по времени операциями, как переформатирование значительной части текста, печать документа или его сохранение на локальном или удаленном диске. Еще одним примером необходимости распараллеливания является сетевой сервер баз данных. В этом случае параллелизм желателен как для обслуживания различных запросов к базе данных, так и для более быстрого выполнения отдельного запроса за счет одновременного просмотра различных записей базы. Именно для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading). Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем процессов. В отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, нежели процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для

разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [2].

1.3. Вывод

К программе будут применены следующие требования к программному обеспечению.

Требования к вводу: на вход подаются массив, заполненный случайными числами, индексы первого и последнего элементов.

Требования к программе:

1) на выходе необходимо получить массив, отсортированный по возрастанию;

2) требуется замерить время работы каждого из алгоритмов.

2 | Конструкторская часть

В этом разделе содержатся схемы алгоритма сортировки массива слиянием и описание его паралелльной версии.

На вход алгоритм принимает массив, индексы первого и последнего элементов.

2.1. Схемы алгоритмов

На Рис. 2.1 представлена схема алгоритма сортировки массива слиянием.

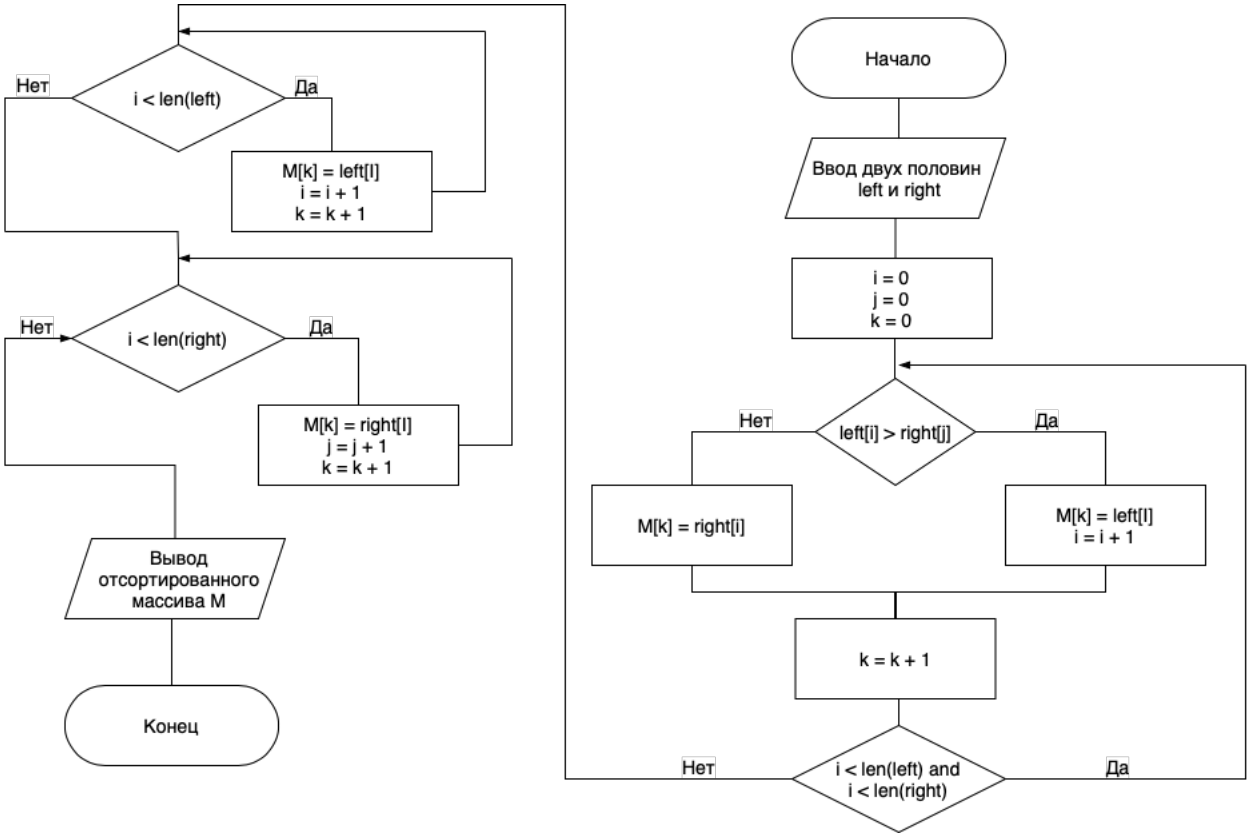

Рис. 2.1. Схема алгоритма сортировки массива слиянием.

2.2. Распараллеливание алгоритма сортировки слиянием

Распараллеливание программы должно ускорять время работы. Это достигается за счет реализации в узких участках (например, в циклах с большим количеством независимых вычислений).

В предложенном алгоритме данными участком является разделения массива на два подмассива. Данный участок программы как раз предлагается распараллелить.

3 | Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации, представлен листинг кода и описание тестирования.

3.1. Средства реализации

В качестве языка программирования был выбран C++ т.к. я знакома с данным языком, у него есть уникальный баланс между возможностями объектно-ориентированного программирования и производительностью. Он одновременно позволяет писать высокоуровневый абстрактный код, который при этом работает со скоростью близкой к машинному коду.

Среда разработки — CLion, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим рефакторингом кода, и богатыми возможностями в навигации.

Время работы алгоритмов было замерено с помощью функции steady_clock() из библиотеки chrono [4].

Для тестирования использовался компьютер на базе процессора 2,2 GHz Intel Core i7, 6 ядер, 4 логических процессора

Многопоточное программирование было реализовано с помощью библиотеки pthread [5].

3.2. Листинг кода

В Листинге 3.1 показана реализация вспомогательной функции для алгоритма сортировки слиянием. Подпрограмма образовывает упорядоченный результирующего массив путем слияния двух также отсортированных массивов меньших размеров.

Листинг 3.1. Вспомогательной функция для алгоритма сортировки слиянием

```
void merge(int first, int last)
{
  int *mas = new int[M];

  int middle = (first + last) / 2;
  int start = first;
  int final = middle + 1;

  int j;
  for(j = first; j <= last; j++)
  {
    if ((start <= middle) && ((final > last) || (a[start] < a[final])))
    {
      mas[j] = a[start];
      start++;
    }
    else
    {
      mas[j] = a[final];
      final++;
    }
  }
  for (j = first; j <= last; j++) a[j] = mas[j];
  delete []mas;
}
```

В Листинге 3.2 показана реализация алгоритма сортировки массива слиянием.

Листинг 3.2. Алгоритм сортировки массива слиянмием

```
int mergesort(void *a, int first, int last)
{
  if (first < last)
  {
    mergesort(a, first, (first+last) / 2);
    mergesort(a, (first+last) / 2 + 1, last);

    merge(first, last);
  }
}
```

В Листинге 3.3 описана структура для индекса массива .

Листинг 3.3. Структура для индекса массива

```
typedef struct Arr {
  int low;
  int high;
} ArrayIndex;
```

В Листинге 3.4 показана реализация распараллеленного алгоритма сортировки массива слиянием.

Листинг 3.4. Многопоточный алгоритм сортировки массива слиянмием

```
void * mergesort_threads(void *a)
{
  ArrayIndex *pa = (ArrayIndex *)a;
  int mid = (pa->low + pa->high)/2;

  ArrayIndex aIndex[N];
  pthread_t thread[N];

  aIndex[0].low = pa->low;
  aIndex[0].high = mid;

  aIndex[1].low = mid+1;
  aIndex[1].high = pa->high;

  if (pa->low >= pa->high) return 0;

  int i;
  for(i = 0; i < N; i++) pthread_create(&thread[i], NULL, mergesort_threads,
      &aIndex[i]);
  for(i = 0; i < N; i++) pthread_join(thread[i], NULL);

  merge(pa->low, pa->high);

  return 0;
}
```

3.3. Тестирование

Реализовано функциональное тестирование отдельным файлом test.cpp. Полученные результаты функций сравниваются с контрольными значениями.

Тестирование происходит по следующим данным:

1) проверка работы сортировки на массиве размером 1;
2) проверка работы сортировки на массиве размеров 100, числа расположены случайно;
3) проверка работы сортировки на массиве размером 100, числа расположеню в порядке убывания;
4) проверка работы сортировки на массиве размером 100, числа расположеню в порядке возрастания;

Также тестирование проходило сначала стандартым алгоритмом сортировки слиянием, затем его многопоточной версией.

Программа успешно прошла все тестовые случаи, см. Рис. 3.1.

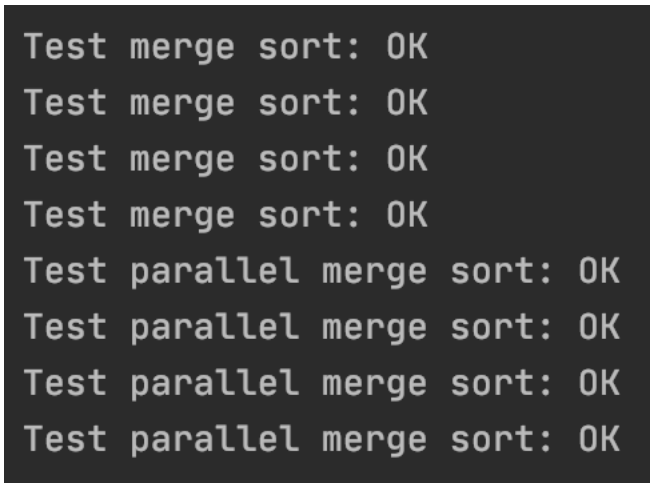

Рис. 3.1. Тестирование программы

4 | Экспериментальная часть

В данном разделе приведены примеры работы программы и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1. Примеры работы

На Рис. 4.1 приведен пример работы программы.

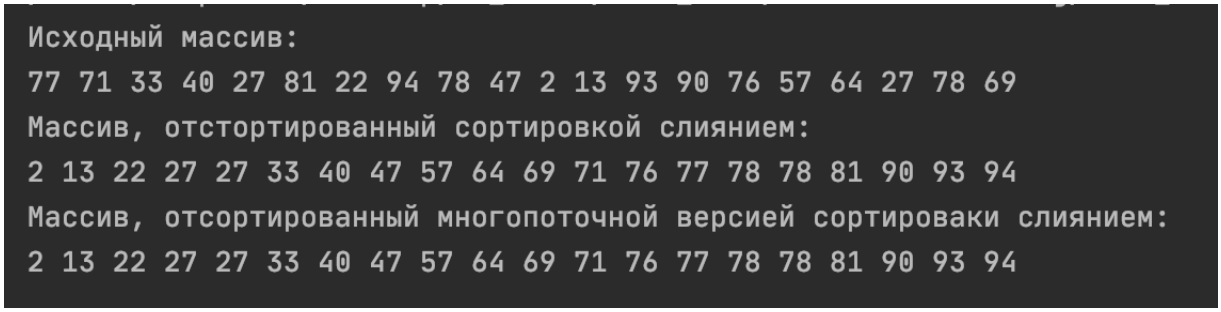

Рис. 4.1. Пример работы программы

4.2. Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализации алгоритмов будет использована формула:

$$t = \frac{T}{N} \tag{4.1}$$

где t — среднее время выполнения алгоритма, N — количество замеров, T — время выполнения N замеров. Неоднократное измерение времени необходимо для получения более точного результа.

Все эксперименты производятся на массивах размеров от 100 до 1000 с шагом 100. Время измеряется в микросекундах.

На рисунке 4.1 изображены графики зависимости времени выполнения программы от размера входных квадратных матриц и количество потоков:

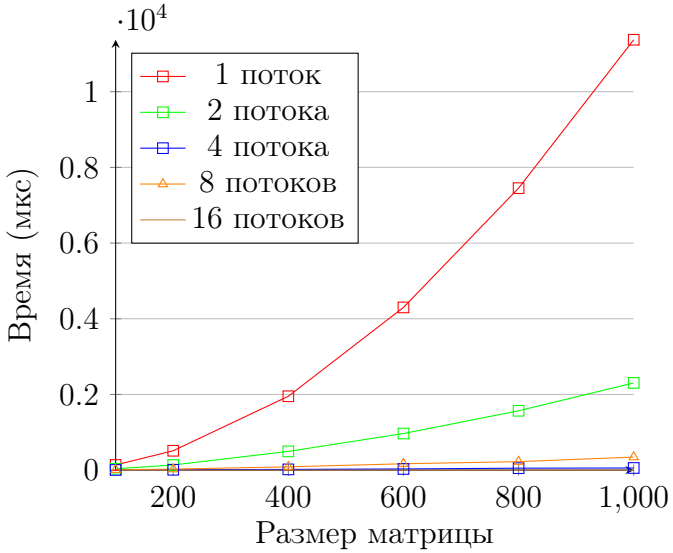

Рис. 4.1: Сравнение времени работы алгоритма работы сортировки слиянием при различном количестве потоков.

Как видно из Рис. 4.3, начиная с 4 потоков увеличение числа потоков не дало сильного прироста в скорости, это связано с тем, что число потоков, которые работают параллельно равно числу логических процессоров, которых на экспериментальном процессоре, как было сказано в Технологической части, 4.

Вывод

По результатам исследования получилось, что, учитывая количество логических процессоров, программа, работающая на 4-ех потоках, выигрывает по скорости работы по сравнению с программой, которая работает на 1-ом или 2-ух потоках. Исползовать больше потоков, чем логических процессоров на компьютере, неэффективно.

Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений и использованы на практике. Был реализован алгоритм сортировки массива слиянием с помощью параллельных вычислений. Было произведено сравнение работы многотопоточного алгоритма на разном количестве потоков. Эксперементально было установлено, что увеличение потоков имеет смысл, пока не будет достигнуто число логических процессоров в системе, причем максимальная скорость работы достигается именно при нем.

Цель работы достигнута. Получены практические навыки использования параллельных вычислений, а также проведена исследовательская работа по временной эффективности такого подхода.

Список литературы

1. Основные виды сортировок и примеры их реализации. [электронный ресурс]. Режим доступа: https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii (Дата обращения: 16.10.2020)

2. Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/4447/983/lecture/14925, свободный (Дата обращения: 15.10.20)

3. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с.

4. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017, свободный (Дата обращения: 15.10.20)

5. Официальный сайт Microsoft, документация [электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-class?view=vs-2019, свободный (Дата обращения: 15.10.20)