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

Кафедра: Математического обеспечения и суперкомпьютерных технологий

Направление подготовки: «Прикладная математика и информатика»

ОТЧЕТ

по лабораторной работе

на тему:

«Поразрядная сортировка для вещественных чисел с простым слиянием»

Выполнил:студент группы 381703-3
_____Хватов А.Е
Подпись

Проверил:
Доцент кафедры МОСТ
_____Сысоев А.В
Подпись

Нижний Новгород
2019

Содержание

Введение

Сортировка – это одна из базовых операций обработки данных. Под сортировкой понимают процесс перестановки объектов множества в определенном порядке. Одна из целей сортировки облегчить последующий поиск элементов в отсортированном множестве. Например, для поиска элемента в произвольном множестве используется линейный поиск со сложностью O(n). Если ввести ограничение на множество, пусть множество отсортировано в определенном порядке, то для поиска элемента можно применить бинарный поиск со сложностью O(log(n)).

Алгоритмы сортировки можно классифицировать по тому, в какой сфере они применяются: внутренняя сортировка – сортировка элементов в оперативной памяти; внешняя сортировка – сортировка элементов, которые в оперативную память не помещаются.

В данной работе рассматривается внутренняя сортировка вещественных чисел. Процесс сортировки на больших данных занимает длительное время, но так как этот процесс хорошо распараллеливается, значит время, затраченное на сортировку, можно уменьшить, увеличив количество потребляемых ресурсов.

Целью лабораторной работы является реализовать параллельную версию поразрядной сортировки для вещественного типа double.

Постановка задачи

Необходимо реализовать последовательную и параллельную версии поразрядной сортировки вещественных чисел типа double, проверить корректность работы алгоритмов, провести численные эксперименты. По полученным результатам сделать выводы.

Для реализации параллельной версии использовать средства MPI.

Решение задачи

Алгоритм поразрядной сортировки состоит в следующем:

1. Взять младшую позицию байта.
2. Для каждого числа в множестве узнать значение на текущей позиции байта и запомнить количество таких значений.
3. Выписать в результирующее множество, упорядоченные по текущей позиции байта, значения.
4. Взять следующую позицию байта.

Таким образом, будет произведен обход всех байтов от младшего к старшему и по результату каждой итерации цикла мы получим, упорядоченное по текущей позиции байта, множество, причем это множество так же будет упорядоченно для всех позиций байта, которые меньше текущей. По результату прохода от младшей позиции байта к старшей, множество будет полностью отсортировано.

Недостатком подхода является то, что для сортировки множества размера n, необходимо выделить 2nэлементов в памяти. Но при достаточном количестве памяти проблем нет.

Схема распараллеливания

Чтобы распараллелить последовательный алгоритм, поделим исходное множество таким образом, чтобы каждому процессу на вход были отправлены одинаковые по мощности множества. Если мощность множества не кратна количеству процессов, то остаток добавим к нулевому процессу.

Дальше на каждом процессе запускается последовательная версия. По итогу мы имеем множества, количество которых равно количеству процессов. Каждый процесс отправляет на нулевой свое отсортированное множество.

На нулевом процессе запускается алгоритм слияния двух отсортированных множеств в одно. По итогу мы получаем отсортированное множество.

Программная реализация

Алгоритм последовательной сортировки представлен в функции seq_sort(sourse, size), где source–входной массив, в котором содержатся элементы и который будет отсортирован, size– размер входного массива. Функцияиспользуетметодыpass(source, dest, size, offset) иlast_pass(source, dest, size, offset), гдesource – входноймассив, dest – выходной массив, который отсортирован по байту offset, size, размер входного и выходного массивов. Функция seq_sortпоследовательно вызывает функцию passи передает ей порядок байта, по которому будет проводится сортировка. last_pass вызывается на старшем байте, она записывает в выходной массив значения с учетом знака.

Алгоритм параллельной сортировки представлен в функции par_sort(sourse, size), где source–входной массив, в котором содержатся элементы и который будет отсортирован, size– размер входного массива.Функция передает каждому процессу часть данных и ожидает возвращения отсортированных кусков, после чего вызывает функцию ordered_merge(source1, size1, source2, size2, dest), которая сливает два упорядоченных массива в массив dest. Соответственно она вызывается C_n^2раз.

Подтверждение коррекции

Для программы написаны тесты с использованием Google C++ Testing Framework. Проверяется, что последовательная и параллельные версии не падают с ошибкой, что последовательная и параллельная версии сортируют массив корректно. Все тесты были пройдены.

Результаты экспериментов

Тестовая архитектура:

2x Processors: Intel Xeon (2x cores 2,66Ггц)

Memory: 4GB DDR2

OperatingSystem: Linux version 3.10.0-1062.1.2.el7.x86_64

Compiler: GCC4.8.5

MPI: Intel MPI Library 2018 Update 4

Interconnect: Intel Corporation 80003ES2LAN Gigabit Ethernet Controller

Эксперимент заключался в запуске сортировки на массиве размером 10^7 элементов на разном количестве вычислительных узлов.

Результаты эксперементов представленны в таблице 1.

Количество узлов	**Общее время работы (с)**
1	18,634
2	17,346
4	17,086
8	16,466

Таблица 1.

Выводы

По результатам работы была реализована последовательная и параллельные версии поразрядной сортировки вещественных чисел типа double. Программа реализована при помощи стредства MPI.

Были проведены эксперементы, демонстрирующие ускорение при увеличении числа вычислительных узлов.

Литература

Дональд, Э. Кнут. Искусство программирования. Том 3. Сортировка и поиск. / перевод Красильников И. В – Издательтсво Вильямс, 2019

Приложение

Приложение 1

main.cpp

```
// Copyright 2019 Khvatov Alexander
#include <gtest-mpi-listener.hpp>
#include <gtest/gtest.h>
#include "./sort.h"
TEST(Seq_Sort, sort_vector_positive) {
   int rank = 0;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if (0 == rank) {
      int size = 5;
      double *vector = new double[size]{ 2.0, 8.66667, 1.69, 99.147, 57.3579 };
      bool result = false;
      seq_sort(vector, size);
      result = is_sorted(vector, size);
      ASSERT_EQ(true, result);
   }
}
TEST(Seq_Sort, sort_large_vector) {
   int rank = 0;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if (0 == rank) {
      int size = 1000;
      double *vector = new double[size];
      for (int i = 0; i < size; i++) vector[i] = (size - i) * 0.001 + (i + size / 2);
      bool result = false;
      seq_sort(vector, size);
      result = is_sorted(vector, size);
      ASSERT_EQ(true, result);
   }
}
TEST(Seq_Sort, sort_is_correct) {
   int rank = 0;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if (0 == rank) {
      int size = 5;
      double *result = new double[size]{ 1.69, 2.0, 8.66667, 57.3579, 99.147};
      double *vector = new double[size]{ 2.0, 8.66667, 1.69, 99.147, 57.3579 };
      seq_sort(vector, size);
      for (int i = 0; i < size; i++) {
         ASSERT_NEAR(result[i], vector[i], 0.01);
      }
   }
}
TEST(Par_Sort, sort_vector) {
   int rank = 0;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   int size = 5;
   double *vector = new double[size]{ 99.147, 8.66667, 1.69, 2.0, 57.3579 };
   par_sort(&vector, size);
   if (0 == rank) {
      bool result = false;
      result = is_sorted(vector, size);
      ASSERT_EQ(true, result);
   }
   delete[] vector;
}
TEST(Par_Sort, sort_large_vector) {
```

```
    int rank = 0;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size = 1000;
    double *vector = new double[size];
    for (int i = 0; i < size; i++) vector[i] = (size - i) * 0.001 + (i + size / 2);
    par_sort(&vector, size);
    if (0 == rank) {
        bool result = false;
        result = is_sorted(vector, size);
        ASSERT_EQ(true, result);
    }
    delete[] vector;
}
int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);
    MPI_Init(&argc, &argv);
    ::testing::AddGlobalTestEnvironment(new GTestMPIListener::MPIEnvironment);
    ::testing::TestEventListeners& listeners =
        ::testing::UnitTest::GetInstance()->listeners();
    listeners.Release(listeners.default_result_printer());
    listeners.Release(listeners.default_xml_generator());
    listeners.Append(new GTestMPIListener::MPIMinimalistPrinter);
    return RUN_ALL_TESTS();
}
```

Приложение 2

sort.h

```
// Copyright 2019 Khvatov Alexander
#ifndef MODULES_TASK_3_KHVATOV_A_DOUBLE_SORT_SORT_H_
#define MODULES_TASK_3_KHVATOV_A_DOUBLE_SORT_SORT_H_
bool is_sorted(double* source, int size);
void pass(double* source, double* dest, int size, int offset);
void last_pass(double* source, double* dest, int size, int offset);
void ordered_merge(double* source1, int size1, double* source2, int size2, double* dest);
void seq_sort(double* source, int size);
void par_sort(double** source, int size);
#endif  // MODULES_TASK_3_KHVATOV_A_DOUBLE_SORT_SORT_H_
```

Приложение 3

sort.cpp

```
// Copyright 2019 Khvatov Alexander
#include <mpi.h>
#include "../../../modules/task_3/khvatov_a_double_sort/sort.h"
bool is_sorted(double* source, int size) {
  for (int i = 0; i < size - 1; i++) {
    if (source[i + 1] < source[i]) {
      return false;
    }
  }
  return true;
}
void pass(double* source, double* dest, int size, int offset) {
  unsigned char* pmem = (unsigned char*)source;
  int counters[256];
  int sum = 0;
  for (int i = 0; i < 256; i++) counters[i] = 0;
  for (int i = 0; i < size; i++) counters[pmem[8 * i + offset]]++;
  for (int i = 0; i < 256; i++) {
    int tmp = counters[i];
    counters[i] = sum;
    sum += tmp;
  }
  for (int i = 0; i < size; i++) {
    int index = 8 * i + offset;
    dest[counters[pmem[index]]] = source[i];
    counters[pmem[index]]++;
  }
}
void last_pass(double* source, double* dest, int size, int offset) {
  unsigned char* pmem = (unsigned char*)source;
  int counters[256];
  int sum = 0;
  for (int i = 0; i < 256; i++) counters[i] = 0;
  for (int i = 255; i > 127; i--) {
    counters[i] += sum;
    sum = counters[i];
  }
  for (int i = 0; i < 128; i++) {
    int tmp = counters[i];
    counters[i] = sum;
    sum += tmp;
  }
  for (int i = 0; i < size; i++) {
    int index = 8 * i + offset;
    if (pmem[index] < 128) {
      dest[counters[pmem[index]]] = source[i];
      counters[pmem[index]]++;
    } else {
      counters[pmem[index]]--;
      dest[counters[pmem[index]]] = source[i];
    }
  }
}
void ordered_merge(double* source1, int size1, double* source2, int size2, double* dest) {
  int i = 0, j = 0, k = 0;
  while ((i < size1) && (j < size2)) {
    if (source1[i] < source2[j]) {
      dest[k] = source1[i];
```

```
            i++;
        } else {
            dest[k] = source2[j];
            j++;
        }
        k++;
    }
    while (i < size1) {
        dest[k] = source1[i];
        i++;
        k++;
    }
    while (j < size2) {
        dest[k] = source2[j];
        j++;
        k++;
    }
}
void seq_sort(double* source, int size) {
    double* temp = NULL;
    double* dest = new double[size];
    for (int i = 0; i < 8; i++) {
        pass(source, dest, size, i);
        temp = source;
        source = dest;
        dest = temp;
    }
    last_pass(source, dest, size, 7);
    delete[] dest;
}

void par_sort(double** source, int size) {
    int proc_rank = 0, proc_size = 0;
    MPI_Comm_size(MPI_COMM_WORLD, &proc_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &proc_rank);
    int lenght = size / proc_size, addition = size % proc_size;
    double* dest = NULL;
    double* temp = NULL;
    if (0 == proc_rank) {
        dest = new double[lenght + addition];
        temp = new double[size];
    } else {
        dest = new double[lenght];
    }
    int* displs = new int[proc_size];
    int* scounts = new int[proc_size];
    displs[0] = 0;
    scounts[0] = lenght + addition;
    for (int i = 1; i < proc_size; i++) {
        displs[i] = addition + lenght * i;
        scounts[i] = lenght;
    }
    MPI_Scatterv(*source, scounts, displs, MPI_DOUBLE, dest, scounts[proc_rank], MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    seq_sort(dest, scounts[proc_rank]);
    MPI_Gatherv(dest, scounts[proc_rank], MPI_DOUBLE, temp, scounts, displs, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    if (0 == proc_rank) {
        double* tmp = NULL;
        for (int i = 0; i < proc_size - 1; i++) {
            ordered_merge(temp, displs[i + 1], &temp[displs[i + 1]], scounts[i + 1], *source);
            tmp = *source;
            *source = temp;
```

```
            temp = tmp;
        }
        tmp = *source;
        *source = temp;
        temp = tmp;
    }
    delete[] dest;
    delete[] temp;
    delete[] displs;
    delete[] scounts;
}
```