МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационные системы

Бариев Эмин Юсуфович

Институт информационных технологий и управления в технических системах

курс 4 группа ИС/б-16-2

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №6

По дисциплине: «ТРСиПВ»

По теме: «Исследование алгоритмов сортировки данных методами пузырька и Шелла, используемых при проектировании параллельных вычислительных программных систем»

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

ст.преподаватель Дрозин А. Ю.

(должность) (подпись) (инициалы,фамилия)

Севастополь 2019

1 ЦЕЛЬ РАБОТЫ

Реализовать и исследовать эффективность алгоритмов параллельной сортировки с использованием функций библиотеки MPI в сравнении с последовательными версиями тех же алгоритмов.

2 ПОСТАНОВКА ЗАДАЧИ

Вариант – 1

Выполнить разработку и отладку программы параллельной сортировки данных с использованием вызовов требуемых функций библиотеки MPI в соответствии с вариантом, указанным преподавателем. Дополнительно реализовать последовательный вариант того же метода сортировки. Получить результаты работы программы в виде протоколов сообщений, комментирующих параллельное выполнение процессов и их взаимодействие в ходе выполнения. Оценить эффективность параллельного процесса сортировки в сравнении с последовательным на том же наборе исходных данных.

Вид сортировки: чет-нечетная.

3 ХОД РАБОТЫ

3.1 Код программы на языке С++

//

// main.cpp

// TRSPV\_6

//

// Created by Emin Bari on 12.12.2019.

// Copyright © 2019 Emin Bari. All rights reserved.

//

#include <iostream>

#include <ctime>

#include <sstream>

#include <mpi.h>

**using** **namespace** std;

**static** **int** proccessCount;

**static** **int** proccessRank;

**static** MPI\_Status status;

**static** MPI\_Comm comm;

**void** generateNumbers(**int**\* &data,**int** &countNumbers,**int** &elementCountForProc);

**void** sort(**int**\* &data\_1, **int**\* &data\_2, **int** length, **bool** smaller);

**void** predSort(**int**\* &mass,**int** length);

**int** main(**int** argc, **char** \*\*argv) {

**int** elementCountForProc;

**int**\* data;

**int**\* result;

**int**\* recv\_data;

**int**\* recv\_data\_2;

**bool** smaller;

MPI\_Init(&argc, &argv);

comm = MPI\_COMM\_WORLD;

**double** tn, tk, delfaf;

tn = MPI\_Wtime(); //measure runtime

MPI\_Comm\_size(comm, &proccessCount);

MPI\_Comm\_rank(comm, &proccessRank);

**if**(proccessRank == 0){

generateNumbers(data,proccessCount,elementCountForProc);

cout<<"Input : ";

**for**(**int** i = 0; i < proccessCount; i++) {

cout<<data[i]<<" ";

}

cout<<endl;

}

MPI\_Bcast(&elementCountForProc, 1, MPI\_INT, 0, comm);

recv\_data = **new** **int**[elementCountForProc];

recv\_data\_2 = **new** **int**[elementCountForProc];

result = **new** **int**[proccessCount];

MPI\_Scatter(data, elementCountForProc, MPI\_INT, recv\_data, elementCountForProc, MPI\_INT, 0, comm);

predSort(recv\_data, elementCountForProc);

**for**(**int** iter = 0; iter < proccessCount; iter++){

**if**(iter % 2 == 0){

**if**(proccessRank % 2 == 0){

MPI\_Sendrecv(recv\_data, elementCountForProc, MPI\_INT, proccessRank + 1, 99,recv\_data\_2, elementCountForProc, MPI\_INT, proccessRank + 1, 99, comm, &status);

smaller = **true**;

sort(recv\_data,recv\_data\_2, elementCountForProc, smaller);

}

**else** **if**(proccessRank % 2 == 1){

MPI\_Sendrecv(recv\_data, elementCountForProc, MPI\_INT, proccessRank - 1, 99,recv\_data\_2, elementCountForProc, MPI\_INT, proccessRank - 1, 99, comm, &status);

smaller = **false**;

sort(recv\_data,recv\_data\_2, elementCountForProc,smaller);

}

}

**else**{

**if**(proccessRank % 2 == 1 && proccessRank != (proccessCount-1)){

MPI\_Sendrecv(recv\_data, elementCountForProc, MPI\_INT, proccessRank+1, 99,recv\_data\_2, elementCountForProc, MPI\_INT, proccessRank+1, 99, comm, &status);

smaller = **true**;

sort(recv\_data,recv\_data\_2, elementCountForProc,smaller);

}

**else** **if**(proccessRank % 2 == 0 && proccessRank != 0){

MPI\_Sendrecv(recv\_data, elementCountForProc, MPI\_INT, proccessRank-1, 99,recv\_data\_2, elementCountForProc, MPI\_INT, proccessRank-1, 99, comm, &status);

smaller = **false**;

sort(recv\_data,recv\_data\_2, elementCountForProc,smaller);

}

}

}

MPI\_Barrier(comm);

MPI\_Gather(recv\_data, elementCountForProc, MPI\_INT, result, elementCountForProc, MPI\_INT, 0, comm);

**if**(proccessRank == 0){

stringstream stream;

stream<<"Sorted result: ";

**for**(**int** i = 0; i < proccessCount; i++){

stream<<" "<<result[i];

}

cout<<stream.str()<<endl;

}

MPI\_Barrier(comm);

tk = MPI\_Wtime();

delfaf = tk - tn;

**if**(proccessRank == 0){

cout << "Runtime = " << delfaf<< endl;

}

MPI\_Finalize();

**return** 0;

}

**void** generateNumbers(**int**\* &data,**int** &countNumbers,**int** &elementCountForProc){

srand(6);

elementCountForProc = countNumbers/proccessCount;

data = **new** **int**[proccessCount];

**for**(**int** i = 0; i < proccessCount; i++)

data[i] = rand()%201 - 100;

}

**void** sort(**int**\* &data\_1, **int**\* &data\_2, **int** length, **bool** smaller){

**int** length\_mass = length \* 2;

**int** \*mass = **new** **int**[length\_mass];

**for**(**int** i = 0, j = 0; i < length;i++){

mass[j++] = data\_1[i];

mass[j++] = data\_2[i];

}

**for**(**int** k = 0; k < length\_mass; k++){

**for**(**int** i = 0; i < length\_mass - 1; i++){

**if**(mass[i] > mass[i+1]){

**int** tmp = mass[i];

mass[i] = mass[i+1];

mass[i+1] = tmp;

}

}

}

**if**(smaller){

**for**(**int** i = 0; i < length; i++)

data\_1[i] = mass[i];

}

**else**{

**for**(**int** i = 0;i < length; i++)

data\_1[i] = mass[length + i];

}

}

**void** predSort(**int**\* &mass,**int** length){

**for**(**int** k = 0; k < length; k++){

**for**(**int** i = 0; i < length - 1; i++){

**if**(mass[i] > mass[i + 1]){

**int** tmp = mass[i];

mass[i] = mass[i + 1];

mass[i + 1] = tmp;

}

}

}

}

3.2 Результаты выполнения программы

**Input : 41 98 28 48 29 -74 17 37 34 -33 30 83 64 -58 -82 77 -86 -10 -30 -82**

**Sorted result: -86 -82 -82 -74 -58 -33 -30 -10 17 28 29 30 34 37 41 48 64 77 83 98**

**Runtime = 0.003524**

ВЫВОДЫ

В ходе выполнения лабораторной работы была реализована программа чет-нечетной сортировки, а так же была исследована эффективность алгоритмов параллельной быстрой сортировки с использованием функций библиотеки MPI в сравнении с последовательными версиями тех же алгоритмов.