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 №7

По дисциплине: «ТРСиПВ»

По теме: «Исследование алгоритмов параллельной быстрой сортировки данных, 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 быстрой сортировки с использованием функций библиотеки MPI в сравнении с последовательными версиями тех же алгоритмов.

2 ПОСТАНОВКА ЗАДАЧИ

Вариант – 1

Выполнить разработку и отладку программы быстрой сортировки данных с использованием вызовов требуемых функций библиотеки MPI для реализации варианта сортировки в соответствии с вариантом, указанным преподавателем. В качестве базового варианта реализовать также сортировку последовательным методом. 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: быстрая сортировка.

3 ХОД РАБОТЫ

3.1 Код программы на языке С++

//

// main.cpp

// TRSPV\_7

//

// Created by Emin Bari on 30.11.2019.

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

//

#include <iostream>

#include <cmath>

#include <mpi.h>

#include <vector>

#include <stdlib.h>

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

**static** **int** processRank;

**static** **int** processCount;

**static** MPI\_Comm comm;

**static** MPI\_Status status;

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

**enum** {A00 = 0, A01 = 1, A10 = 2, A11 = 3};

**void** swap(**int**\* a, **int**\* b);

**int** partition (**int** arr[], **int** low, **int** high);

**void** quickSort(**int** arr[], **int** low, **int** high, **int** pivot);

**void** printArray(**int** arr[], **int** size);

**void** printTotalArray(**int** arr[], **int** size);

**void** printSubArray();

**int** countPivot(**int** quatroArr[]);

vector<**int**> rightVec;

vector<**int**> leftVec;

**static** **int** arr[] = {2, 7, 9, 8, 0, 10, 11, 20, 3, 5, 7, -3, 6, 8, 1, 16};

**static** **int** n = **sizeof**(arr) / **sizeof**(arr[0]);

**static** **int** subArr[4];

**int** pivot1 = 0;

**int** pivot2\_1 = 0;

**int** pivot2\_2 = 0;

**int** size2\_left;

**int** size2\_right;

**static** **int** pivotElementsCount = 0;

**int** \*leftArr;

**int** \*rightArr;

**int** \*totalsubArr;

**int** \*recvPieceArr;

**int** \*newSubArr\_1;

**int** \*sortedArr = **new** **int** [n];

**int** bufCount;

**void** firstPhase();

**void** secondPhase();

**void** thirdPhase();

**void** thirdPhaseQuickSort(**int** arr[], **int** low, **int** high);

**int** thirdPhasePartition (**int** arr[], **int** low, **int** high);

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

MPI\_Init(&argc, &argv);

comm = MPI\_COMM\_WORLD;

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

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

firstPhase();

MPI\_Barrier(comm);

secondPhase();

MPI\_Barrier(comm);

thirdPhase();

MPI\_Barrier(comm);

MPI\_Finalize();

**return** 0;

}

**void** thirdPhase(){

**if** (processRank == A00) {

totalsubArr = **new** **int**[leftVec.size()];

copy(leftVec.begin(), leftVec.end(), totalsubArr);

thirdPhaseQuickSort(totalsubArr, 0, leftVec.size() - 1);

printTotalArray(totalsubArr, leftVec.size());

}

MPI\_Barrier(comm);

**if** (processRank == A01) {

totalsubArr = **new** **int**[rightVec.size()];

copy(rightVec.begin(), rightVec.end(), totalsubArr);

thirdPhaseQuickSort(totalsubArr, 0, rightVec.size() - 1);

printTotalArray(totalsubArr, rightVec.size());

}

MPI\_Barrier(comm);

**if** (processRank == A10) {

totalsubArr = **new** **int**[leftVec.size()];

copy(leftVec.begin(), leftVec.end(), totalsubArr);

thirdPhaseQuickSort(totalsubArr, 0, leftVec.size() - 1);

printTotalArray(totalsubArr, leftVec.size());

}

MPI\_Barrier(comm);

**if** (processRank == A11) {

totalsubArr = **new** **int**[rightVec.size()];

copy(rightVec.begin(), rightVec.end(), totalsubArr);

thirdPhaseQuickSort(totalsubArr, 0, rightVec.size() - 1);

printTotalArray(totalsubArr, rightVec.size());

}

//

// int gsize;

// int \*buffer;

//

// int \*displs;

// int stride = 16;

//

// MPI\_Comm\_size(comm, &gsize);

// buffer = (int \*)malloc(gsize\*stride\*sizeof(int));

// displs = (int \*)malloc(gsize\*sizeof(int));

// //recvcounts = (int \*)malloc(gsize\*sizeof(int));

//

// for (int i = 0; i < gsize; ++i) {

// displs[i] = i\*stride;

// }

// int recvcounts[4] = {2,1,4,9};

// MPI\_Gatherv(totalsubArr, n, MPI\_INT, buffer, recvcounts, displs, MPI\_INT, A00, comm);

}

**void** secondPhase(){

//first part of PHASE2: SENDING PIVOT ELEMENT

**if** (processRank == A00) {

leftArr = **new** **int**[leftVec.size()];

//vector to array

copy(leftVec.begin(), leftVec.end(), leftArr);

pivot2\_1 = countPivot(leftArr);

//cout<<"PIVOT IS = "<<pivot2\_1<<" of count of numbers = "<<pivotElementsCount<<endl;

//sending pivot to process A01

MPI\_Send(&pivot2\_1, pivotElementsCount, MPI\_INT, A01, 99, comm);

}

**if** (processRank == A01) {

MPI\_Probe(A00, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

MPI\_Recv(&pivot2\_1, bufCount, MPI\_INT, A00, 99, comm, &status);

leftArr = **new** **int**[leftVec.size()];

copy(leftVec.begin(), leftVec.end(), leftArr);

}

MPI\_Barrier(comm);

**if** (processRank == A10) {

rightArr = **new** **int**[rightVec.size()];

//vector to array

copy(rightVec.begin(), rightVec.end(), rightArr);

pivot2\_2 = countPivot(rightArr);

//cout<<"PIVOT IS = "<<pivot2\_2<<" of count of numbers = "<<pivotElementsCount<<endl;

//sending pivot to process A11

MPI\_Send(&pivot2\_2, pivotElementsCount, MPI\_INT, A11, 99, comm);

}

**if** (processRank == A11) {

MPI\_Probe(A10, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

MPI\_Recv(&pivot2\_2, bufCount, MPI\_INT, A10, 99, comm, &status);

rightArr = **new** **int**[rightVec.size()];

copy(rightVec.begin(), rightVec.end(), rightArr);

}

///////////////////////////////////////////////////////////////

//Second part of PHASE2: sorting

size2\_left = leftVec.size();

size2\_right = rightVec.size();

leftVec.clear();

rightVec.clear();

**if** (processRank == A00) {

cout<<"---------SECOND PHASE---------"<<endl;

quickSort(leftArr, 0, size2\_left - 1, pivot2\_1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A01) {

quickSort(leftArr, 0, size2\_left - 1, pivot2\_1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A10) {

quickSort(rightArr, 0, size2\_right - 1, pivot2\_2);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A11) {

quickSort(rightArr, 0, size2\_right - 1, pivot2\_2);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

//left(00 10) or right(01 11) array parts exchanging

**switch** (processRank) {

**case** A00:

MPI\_Send(rightArr, rightVec.size(), MPI\_INT, A01, 99, comm);

MPI\_Probe(A01, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A01, 99, comm, &status);

// cout<<"process "<<processRank<<" recives array "<<endl;

// for (int i = 0; i < bufCount; i++) {

// cout<<recvPieceArr[i]<<" ";

// }

//

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

leftVec.push\_back(recvPieceArr[i]);

}

// cout<<"Concatinated left vector"<<endl;

// //it is also a new array to the next iteration of algorithm

// for(int i : leftVec)

// cout<<i<<" ";

// cout<<endl;

**break**;

**case** A01:

MPI\_Probe(A00, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A00, 99, comm, &status);

// cout<<"process "<<processRank<<" recives array "<<endl;

// for (int i = 0; i < bufCount; i++)

// cout<<recvPieceArr[i]<<" ";

//

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

rightVec.push\_back(recvPieceArr[i]);

// cout<<"Concatinated right vector"<<endl;

// //it is also a new array to the next iteration of algorithm

// for(int i : rightVec)

// cout<<i<<" ";

// cout<<endl;

MPI\_Send(leftArr, leftVec.size(), MPI\_INT, A00, 99, comm);

**break**;

**case** A10:

MPI\_Send(rightArr, rightVec.size(), MPI\_INT, A11, 99, comm);

MPI\_Probe(A11, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A11, 99, comm, &status);

// cout<<"process "<<processRank<<" recives array "<<endl;

//

// for (int i = 0; i < bufCount; i++)

// cout<<recvPieceArr[i]<<" ";

//

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

leftVec.push\_back(recvPieceArr[i]);

// cout<<"Concatinated left vector"<<endl;

// //it is also a new array to the next iteration of algorithm

// for(int i : leftVec)

// cout<<i<<" ";

// cout<<endl;

**break**;

**case** A11:

MPI\_Probe(A10, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A10, 99, comm, &status);

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

rightVec.push\_back(recvPieceArr[i]);

MPI\_Send(leftArr, leftVec.size(), MPI\_INT, A10, 99, comm);

**break**;

**default**:

**break**;

}

**if** (processRank == A00) {

cout<<processRank<<"process"<<endl;

**for** (**int** i: leftVec)

cout<<i<<" ";

cout<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A10) {

cout<<processRank<<"process"<<endl;

**for** (**int** i: leftVec)

cout<<i<<" ";

cout<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A01) {

cout<<processRank<<"process"<<endl;

**for** (**int** i: rightVec)

cout<<i<<" ";

cout<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A11) {

cout<<processRank<<"process"<<endl;

**for** (**int** i: rightVec)

cout<<i<<" ";

cout<<endl;

}

**delete** [] leftArr;

**delete** [] rightArr;

**delete** [] recvPieceArr;

}

**void** firstPhase(){

MPI\_Scatter(arr, processCount, MPI\_INT, subArr, processCount, MPI\_INT, A00, comm);

MPI\_Barrier(comm);

**if** (processRank == A00) {

printSubArray();

pivot1 = countPivot(subArr);//calculating pivot

cout<<"PIVOT==="<<pivot1<<endl;

}

//sending pivot to everyone

MPI\_Bcast(&pivot1, 1, MPI\_INT, A00, comm);

MPI\_Barrier(comm);

**if** (processRank == A00) {

quickSort(subArr, 0, 3, pivot1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A01) {

printSubArray();

quickSort(subArr, 0, 3, pivot1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A10) {

printSubArray();

quickSort(subArr, 0, 3, pivot1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

**if** (processRank == A11) {

printSubArray();

quickSort(subArr, 0, 3, pivot1);

cout<<"---------ended "<<processRank<<"---------"<<endl;

}

MPI\_Barrier(comm);

//left(00 01) or right(10 11) array parts exchanging

**switch** (processRank) {

**case** A00:

MPI\_Send(rightArr, rightVec.size(), MPI\_INT, A10, 99, comm);

MPI\_Probe(A10, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A10, 99, comm, &status);

cout<<"process "<<processRank<<" recives array "<<endl;

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

cout<<recvPieceArr[i]<<" ";

}

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

leftVec.push\_back(recvPieceArr[i]);

}

cout<<"Concatinated left vector"<<endl;

//it is also a new array to the next iteration of algorithm

**for**(**int** i : leftVec)

cout<<i<<" ";

cout<<endl;

**break**;

**case** A10:

MPI\_Probe(A00, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A00, 99, comm, &status);

cout<<"process "<<processRank<<" recives array "<<endl;

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

cout<<recvPieceArr[i]<<" ";

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

rightVec.push\_back(recvPieceArr[i]);

cout<<"Concatinated right vector"<<endl;

//it is also a new array to the next iteration of algorithm

**for**(**int** i : rightVec)

cout<<i<<" ";

cout<<endl;

MPI\_Send(leftArr, leftVec.size(), MPI\_INT, A00, 99, comm);

**break**;

**case** A01:

MPI\_Send(rightArr, rightVec.size(), MPI\_INT, A11, 99, comm);

MPI\_Probe(A11, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A11, 99, comm, &status);

cout<<"process "<<processRank<<" recives array "<<endl;

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

cout<<recvPieceArr[i]<<" ";

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

leftVec.push\_back(recvPieceArr[i]);

cout<<"Concatinated left vector"<<endl;

//it is also a new array to the next iteration of algorithm

**for**(**int** i : leftVec)

cout<<i<<" ";

cout<<endl;

**break**;

**case** A11:

MPI\_Probe(A01, MPI\_ANY\_TAG, comm, &status);

MPI\_Get\_count(&status, MPI\_INT, &bufCount );

recvPieceArr = **new** **int**[bufCount];

MPI\_Recv(recvPieceArr, bufCount, MPI\_INT, A01, 99, comm, &status);

cout<<"process "<<processRank<<" recives array "<<endl;

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

cout<<recvPieceArr[i]<<" ";

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

rightVec.push\_back(recvPieceArr[i]);

cout<<"Concatinated right vector"<<endl;

//it is also a new array to the next iteration of algorithm

**for**(**int** i : rightVec)

cout<<i<<" ";

cout<<endl;

MPI\_Send(leftArr, leftVec.size(), MPI\_INT, A01, 99, comm);

**break**;

**default**:

**break**;

}

**delete** [] leftArr;

**delete** [] rightArr;

**delete** [] recvPieceArr;

}

**int** countPivot(**int** quatroArr[]){

**int** pivot = 0;

**for** (**int** i = 0; i < 4; i++) {

pivot += quatroArr[i];

}

pivot = ceil(pivot/4);

pivotElementsCount = pivot == 0 ? 1 : **int** (log10 (pivot) + 1);

**return**(pivot);

}

**void** printSubArray(){

cout<<"Process "<<processRank<<endl;

**for** (**int** i = 0; i < 4; i++) {

cout<<subArr[i]<<" ";

}

cout<<endl;

}

**void** printArray(**int** arr[], **int** size){

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

cout << arr[i] << " ";

cout << endl;

}

**void** printTotalArray(**int** arr[], **int** size){

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

cout << arr[i] << " ";

}

**void** swap(**int**\* a, **int**\* b){

**int** t = \*a;

\*a = \*b;

\*b = t;

}

**int** partition (**int** arr[], **int** low, **int** high, **int** pivot){

**int** i = low - 1;

**int** j = high;

**for**(;;)

{

**while**(arr[++i] < pivot)

;

**while**(pivot < arr[--j])

**if**(j == low)

**break**;

**if**( i >= j)

**break**;

swap(&arr[i], &arr[j]);

}

swap(&arr[i], &arr[high]);

**return** i;

}

**void** quickSort(**int** arr[], **int** low, **int** high, **int** pivot)

{

**if** (low < high)

{

partition(arr, low, high, pivot);

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

**if** (arr[i] < pivot) {

leftVec.push\_back(arr[i]);

}

**else**{

rightVec.push\_back(arr[i]);

}

}

cout<<endl;

cout<<"left SubArray:"<<endl;

**for**(**int** i:leftVec)

cout<<i<<" ";

cout<<endl;

cout<<"right SubArray:"<<endl;

**for**(**int** i:rightVec)

cout<<i<<" ";

cout<<endl;

leftArr = **new** **int**[leftVec.size()];

rightArr = **new** **int**[rightVec.size()];

//vector to array

copy(leftVec.begin(), leftVec.end(), leftArr);

copy(rightVec.begin(), rightVec.end(), rightArr);

}

}

**int** thirdPhasePartition (**int** arr[], **int** low, **int** high)

{

**int** pivot = arr[high]; // pivot

**int** i = (low - 1); // Index of smaller element

**for** (**int** j = low; j <= high - 1; j++)

{

// If current element is smaller than the pivot

**if** (arr[j] < pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

**return** (i + 1);

}

**void** thirdPhaseQuickSort(**int** arr[], **int** low, **int** high)

{

**if** (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

**int** pi = thirdPhasePartition(arr, low, high);

// Separately sort elements before

// partition and after partition

thirdPhaseQuickSort(arr, low, pi - 1);

thirdPhaseQuickSort(arr, pi + 1, high);

}

}

3.2 Результаты выполнения программы

**Process 0**

**2 7 9 8**

**PIVOT===6**

**left SubArray:**

**2**

**right SubArray:**

**8 9 7**

**---------ended 0---------**

**Process 1**

**0 10 11 20**

**left SubArray:**

**0**

**right SubArray:**

**20 11 10**

**---------ended 1---------**

**Process 2**

**3 5 7 -3**

**left SubArray:**

**3 5 -3**

**right SubArray:**

**7**

**---------ended 2---------**

**Process 3**

**6 8 1 16**

**left SubArray:**

**1**

**right SubArray:**

**16 6 8**

**---------ended 3---------**

**process 2 recives array**

**8 9 7process 3 recives array**

**20 11 10 Concatinated right vector**

**16 Concatinated right vector**

**7 8 9 7**

**6 8 20 11 10**

**process 0 recives array**

**3 5 -3 process 1 recives array**

**1 Concatinated left vectorConcatinated left vector**

**2 3 5 -3**

**0 1**

**---------SECOND PHASE---------**

**left SubArray:**

**-3**

**right SubArray:**

**3 5 2**

**---------ended 0---------**

**left SubArray:**

**0**

**right SubArray:**

**1**

**---------ended 1---------**

**left SubArray:**

**right SubArray:**

**7 8 9 7**

**---------ended 2---------**

**left SubArray:**

**6**

**right SubArray:**

**10 8 20 11 16**

**---------ended 3---------**

**0process**

**-3 0**

**2process**

**6**

**1process**

**1 3 5 2**

**3process**

**10 8 20 11 16 7 8 9 7**

**-3 0 1 2 3 5 6 7 7 8 8 9 10 11 16 20 Program ended with exit code: 0**

ВЫВОДЫ

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