МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное бюджетное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационных систем

Голикова Александра Антоновна

Институт информационных технологий и управления в технических системах

курс 4 группа ИС/б-16-2-о

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Теория распределенных систем и параллельных вычислений»

на тему «Исследование алгоритмов параллельной быстрой сортировки данных, используемых при проектировании параллельных вычислительных программных систем»

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

(дата)

Руководитель практикума

ст. преподаватель   А. Ю. Дрозин

(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. ПОРЯДОК ВЫПОЛНЕНИЯ

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. ТЕКСТ ПРОГРАММЫ

#include <iostream>

#include <mpi.h>

#include <vector>

using namespace std;

int rankNum, size;

void printVector(vector<int> a){

for (int i=0; i<a.size(); i++)

cout << a[i] << " ";

cout << "\n";

}

vector<int> generate(int n){

srand(time(0));

vector<int> v(n);

for (int i=0; i<n; i++)

v[i] = rand() % 100;

return v;

}

int partition(vector<int> &v, int left, int right){

int pivot = v[right];

int i = left;

for (int j=left; j<right; j++) {

if (v[j] <= pivot){

if (i != j){

int aux = v[i];

v[i] = v[j];

v[j] = aux;

}

i++;

}

}

if (v[right] < v[i]){

v[right] = v[i];

v[i] = pivot;

}

return i;

}

void quick\_sort(vector<int> &v, int left, int right, int procsLeft){

if (left == right)

return;

int piv = partition(v, left, right);

cout << "Процесс " << rankNum << " отсортировал (парсер = " << v[piv] << "): ";

printVector(v);

if (piv == left){

quick\_sort(v, left + 1, right, procsLeft);

return;

}

if (piv == right){

quick\_sort(v, left, right - 1, procsLeft);

return;

}

if (procsLeft > 1){

int child = rankNum + procsLeft/2 + procsLeft%2;

int size = right - piv;

MPI\_Status status;

MPI\_Send(&size, 1, MPI\_INT, child, 1, MPI\_COMM\_WORLD);

MPI\_Send(v.data() + piv + 1, size, MPI\_INT, child, 2, MPI\_COMM\_WORLD);

quick\_sort(v, left, piv - 1, procsLeft/2 + procsLeft%2);

// cout << rankNum << " waiting for data...\n";

MPI\_Recv(v.data() + piv + 1, size, MPI\_INT, child, 3, MPI\_COMM\_WORLD, &status);

// cout << rankNum << " received data...\n";

} else {

quick\_sort(v, left, piv - 1, 1);

quick\_sort(v, piv + 1, right, 1);

}

}

void quick\_sort\_worker(){

int child = 0, procsLeft = size, parent = 0;

while (child < rankNum){

int step = procsLeft/2 + procsLeft%2;

if (child + step <= rankNum){

parent = child;

procsLeft /= 2;

child += step;

} else{

procsLeft = step;

}

}

MPI\_Status status;

int size;

MPI\_Recv(&size, 1, MPI\_INT, parent, 1, MPI\_COMM\_WORLD, &status);

vector<int> v(size);

MPI\_Recv(v.data(), size, MPI\_INT, parent, 2, MPI\_COMM\_WORLD, &status);

quick\_sort(v, 0, size - 1, procsLeft);

MPI\_Send(v.data(), size, MPI\_INT, parent, 3, MPI\_COMM\_WORLD);

}

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

MPI\_Init(NULL, NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rankNum);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (!rankNum){

if (argc != 2){

cout<<"Используйте команду: mpirun -n <КОЛ-ВО\_ПРОЦЕССОВ> <исполняемый файл> <размер массива>\n";

return 1;

}

int n = atoi(argv[1]);

vector<int> v = generate(n);

cout << "Создан "; printVector(v);

quick\_sort(v, 0, v.size() - 1, size);

printVector(v);

} else {

quick\_sort\_worker();

}

MPI\_Finalize();

return 0;

}

4. РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

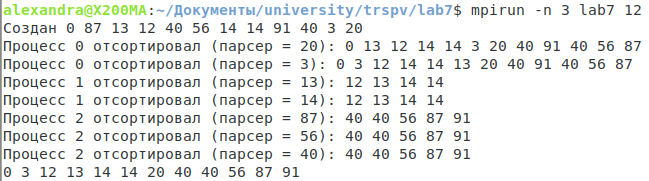


Рисунок 1 – Выполнение программы

ВЫВОДЫ

В рамках лабораторной работы была разработана программа, организующая механизм взаимодействия параллельно выполняющихся процессов — «Клиент - Сервер». Сервер реализован в соответствии со схемой управления использования рандеву. Программа реализует случай, когда сервер отделён от ресурса и к серверу идёт обращение на разрешение использования или освобождение ресурса, а клиент может непосредственно обратиться к ресурсу.