Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего профессионального образования «Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Кафедра: Программная инженерия

Специальность (направление): Программная инженерия

Отчет

по лабораторной работе

по дисциплине «Параллельное программирование»

тема:

«Сортировка Хоара со слиянием “Разделяй и властвуй”»

Выполнил: студент группы 381508

Дмитричев Н.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Научный руководитель:

Доцент, к.т.н. Сысоев А. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород 2018

Оглавление

[Содержание 2](#_Toc513930256)

[Введение 3](#_Toc513930257)

[Сортировка Хоара 3](#_Toc513930258)

[Разбиение «Разделяй и властвуй» 3](#_Toc513930259)

[Тестовая версия 5](#_Toc513930260)

[Solver 5](#_Toc513930261)

[Generator 7](#_Toc513930262)

[Checker 9](#_Toc513930263)

[Тесты 12](#_Toc513930264)

[Параллельная версия 13](#_Toc513930265)

[Общая структура 13](#_Toc513930266)

[OpenMP 13](#_Toc513930267)

[TBB 18](#_Toc513930268)

[Статистика 22](#_Toc513930269)

[Вывод 25](#_Toc513930270)

## Введение

### Сортировка Хоара

**Быстрая сортировка**, **сортировка Хоара** - один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем *О(n \* log n) {\displaystyle O(n\log n)}*обменов при упорядочении {\displaystyle n} *n* элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Общая идея алгоритма быстрой сортировки состоит в следующем:

* выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность;
* сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие»;
* для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

**Разбиение Хоара**

Данная схема использует два индекса (один в начале массива, другой в конце), которые приближаются друг к другу, пока не найдётся пара элементов, где один больше опорного и расположен перед ним, а второй меньше и расположен после. Эти элементы меняются местами. Обмен происходит до тех пор, пока индексы не пересекутся. Алгоритм возвращает последний индекс. Данная схема показывает эффективность в *O*(*n*2), когда входной массив уже отсортирован. Сортировка с использованием данной схемы нестабильна. Следует заметить, что конечная позиция опорного элемента необязательно совпадает с возвращённым индексом.

### Разбиение «Разделяй и властвуй»

Идея слияния по алгоритму «Разделяй и властвуй» заключается в разбиении массивов на участки, которые можно слить независимо [93]. В первом массиве выбирается центральный элемент *x* (он разбивает массив на две равные половины), а во втором массиве с помощью бинарного поиска находится позиция наибольшего элемента, меньшего *x* (позиция этого элемента разбивает второй массив на две части). После такого разбиения первые и вторые половины массивов могут сливать независимо, т.к. в первых половинах находятся элементы меньшие элемента *x*, а во второй – большие (рис. 8.3). Для слияния двух массивов несколькими потоками можно в первом массиве выбрать несколько ведущих элементов, разделив его на равные порции, а во втором массиве найти соответствующие подмассивы. Каждый поток получит свои порции на обработку.

Эффективность такого слияния во многом зависит от того, насколько равномерно произошло «разделение» второго массива.

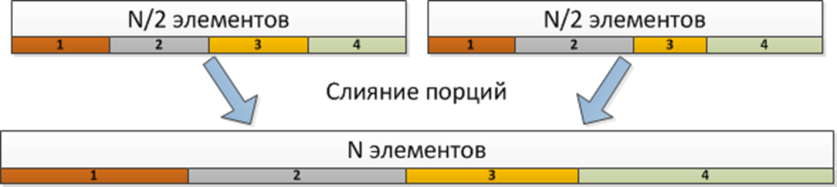


Рис. 1. Слияние «Разделяй и властвуй»

## Тестовая версия

### Solver

Запуск

solver <входной бинарный файл> <выходной бинарный файл>

<входной бинарный файл> - должен уществовать

В случае успешного выполнения в консоли будет `OK`.

В случае ошибки - её краткое описание.

Файл sol.cpp

#pragma once

void hoaraSort(double\*&, int, int);

void hoaraSort(double\* &arr, int first, int last){

int leftIndex = first,

rightIndex = last;

double tmp,

x = arr[(first + last) / 2];

do {

while (arr[leftIndex] < x)

leftIndex++;

while (arr[rightIndex] > x)

rightIndex--;

if (leftIndex <= rightIndex)

{

if (leftIndex < rightIndex)

{

tmp=arr[leftIndex];

arr[leftIndex]=arr[rightIndex];

arr[rightIndex]=tmp;

}

leftIndex++;

rightIndex--;

}

} while (leftIndex <= rightIndex);

if (leftIndex < last)

hoaraSort(arr, leftIndex, last);

if (first < rightIndex)

hoaraSort(arr, first,rightIndex);

}

void IHoaraSort(double\* &arr,const int sizeArr){

hoaraSort(arr,0,sizeArr-1);

}

Файл before\_code.cpp

#include <stdio.h>

#include <stdlib.h>

#include <fstream>

#include <iostream>

#include <omp.h>

#include <new>

#include "sol.cpp"

using namespace std;

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

{

if (argc != 3) {

cerr << argv[0] << " -> ERROR: not enough arguments\n";

exit(1);

}

FILE \*fIn = NULL,

\*fOut = NULL;

if ((fIn = fopen(argv[1], "rb")) == NULL) {

cerr << argv[0] << " -> ERROR: can not open '"<<argv[1]<<"'\n";

exit(1);

}

if ((fOut = fopen(argv[2], "wb")) == NULL) {

cerr << argv[0] << " -> ERROR: can not open '"<<argv[2]<<"'\n";

exit(1);

}

int N = 0;

double \*numbers = NULL;

double trash=0;

fread(&trash, sizeof(trash), 1, fIn);

fread(&N, sizeof(N), 1, fIn);

numbers = new double[N];

fread(numbers, sizeof(\*numbers), N, fIn);

double time = omp\_get\_wtime();

IHoaraSort(numbers, N); // the seq. hoara sort

time = omp\_get\_wtime() - time;

fwrite(&time, sizeof(time), 1, fOut);

fwrite(&N, sizeof(N), 1, fOut);

fwrite(numbers, sizeof(\*numbers), N, fOut);

cout<< argv[0]<<" -> OK\n";

fclose(fIn);

fclose(fOut);

delete[] numbers;

return 0;

}

### Generator

Запуск

generator <номер теста> <файл сохранения>

<номер теста> - принимает значения от `0` до `23`

<файл сохранения> - файл, в который будут выводиться тестовые данные

Размер выходного файла для теста `22` примерно 43Мб. Чем больше номер теста, тем больше выходной файл весит. Создавайте такие на своё усмотрение.

В случае успешного выполнения в консоли будет `OK`.

В случае ошибки - её краткое описание.

Файл generator.cpp

#include <iostream>

#include <random>

#include <chrono>

double\* genFunc(const int N){

double\* ar=new double[N];

std::default\_random\_engine generator(time(0));

for(int i = 0; i < N; ++i){

std::uniform\_real\_distribution<double> distribution(-10000.0,10000.0);

ar[i] = distribution(generator);

}

return ar;

}

using namespace std;

int n\_tests[] = {0,1,2,3,5,10,10,55,100,100,555,1000,

1000,5555,10000,10000,55555,100000,100000,555555,1000000,1000000, 5555555,10000000};

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

if (argc != 3) {

cerr << argv[0] << " -> ERROR: not enough arguments\n";

exit(1);

}

if (atoi(argv[1])>23 || atoi(argv[1])<0){

cerr << argv[0] << " -> ERROR: first parameter has to be >= 0 and <=23\n";

exit(1);

}

FILE\* outFile = NULL;

if ((outFile = fopen(argv[2],"wb"))==NULL){

cerr << argv[0] << " -> ERROR: can not open '"<<argv[2]<<"'\n";

exit(1);

}

int N = n\_tests[atoi(argv[1])];

double\* ar = genFunc(N);

double trash=0; // need to be canon

fwrite(&trash,sizeof(trash),1,outFile);

fwrite(&N,sizeof(N),1,outFile);

fwrite(ar,sizeof(\*ar),N,outFile);

cout << argv[0] << " -> OK\n";

delete[] ar;

return 0;

}

### Checker

Запуск

checker <проверяемый файл>

<проверяемый файл> - бинарный

На выходе будет бинарный файл `result.txt` с результатом выполнения.

В случае успешного выполнения в консоли будет `OK`.

В случае ошибки - её краткое описание.

Файл classResult.h

#pragma once

#include <cstdio>

#include <cmath>

#include <string>

using namespace std;

// Используется для взаимодействия с тестирующей системой

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

/\*

// Checker может устанавливать вот эти три вердикта:

AC = Accepted = Решение выдаёт корректный результат на данном тесте

WA = Wrong Answer = Решение выдаёт некорректный результат на данном тесте

PE = Presentation Error = Ошибка формата выходных данных

// Остальные вердикты checker не может устанавливать

NO = No verdict = Вердикт отсутствует

CE = Compilation Error = Ошибка компиляции

ML = Memory Limit Exceeded = Превышено ограничение по памяти

TL = Time Limit Exceeded = Превышено ограничение по времени работы

RE = Runtime Error = Ошибка времени исполнения программы

IL = Idle Limit Exceeded = Превышено время простоя (бездействия) программы

DE = Deadly Error = Ошибка тестирующей системы

\*/

enum verdict { NO = 1, AC, WA, CE, ML, TL, RE, IL, PE, DE };

class result

{

private:

FILE \* bur;

public:

enum ext\_cls {

NO = 1,

VERDICT,

MESSAGE,

TIME,

MEMORY

};

result (bool read = false) {

if (read)

bur = fopen("result.txt", "r");

else bur = fopen("result.txt", "w");

}

~result() {

fclose (bur);

}

void write\_type(ext\_cls t) {

fwrite(&t, sizeof (t), 1, bur);

}

// Сообщить тестирующей системе, что решение получило один из вердиктов verdict

void write\_verdict(verdict v) {

write\_type(ext\_cls::VERDICT);

fwrite(&v, sizeof (v), 1, bur);

}

// Написать сообщение от checker'a пользователю.

// Например, что решение верное, или неверное.

// Использовать только латинские буквы и знаки препинания

void write\_message(string str) {

write\_type(ext\_cls::MESSAGE);

int l = str.size ();

fwrite(&l, sizeof (l), 1, bur);

fwrite (&str[0], sizeof (str[0]), l, bur);

}

// Сообщить тестирующей системе время работы программы участника,

// вычисленное с помощью before\_code

// x имеет размерность 100 нс = 10 ^ (-7) сек

void write\_time(long long x) {

write\_type(ext\_cls::TIME);

fwrite(&x, sizeof (x), 1, bur);

}

// Сообщить тестирующей системе, память затребованную программой участника

void write\_memory(unsigned long long x) {

write\_type(ext\_cls::MEMORY);

fwrite(&x, sizeof (x), 1, bur);

}

}

checker\_result;

Файл checker.cpp

#include <iostream>

#include "classResult.h"

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

{

if (argc != 2) {

cerr << argv[0] << " -> ERROR: not enough arguments\n";

exit(1);

}

FILE \*buo = NULL;

if ((buo=fopen(argv[1], "rb"))==NULL){

cerr << argv[0] << " -> ERROR: can not open '"<<argv[1]<<"'\n";

exit(1);

}

int No;

double res\_time;

fread(&res\_time, sizeof (res\_time), 1, buo);

fread(&No, sizeof (No), 1, buo);

if (No==0){

checker\_result.write\_message ("AC. Zero size array.");

checker\_result.write\_verdict (verdict::AC);

}

else {

double curNum, prevNum;

fread(&curNum, sizeof (curNum), 1, buo);

prevNum=curNum;

bool result = true;

for (int i = 1; i < No; ++i) {

fread(&curNum, sizeof (curNum), 1, buo);

if (curNum<prevNum)

{

result=false;

break;

}

prevNum=curNum;

}

if (result)

{

checker\_result.write\_message ("AC. Array is sorted.");

checker\_result.write\_verdict (verdict::AC);

}

else

{

checker\_result.write\_message ("WA. Array isn't sorted.");

checker\_result.write\_verdict (verdict::WA);

}

}

checker\_result.write\_time (res\_time \* 1e7);

cout << argv[0] << " -> OK\n";

fclose(buo);

return 0;

}

### Тесты

23 теста в папке *tests.* Имена представлены в виде чисел от 0 до 22. Сгенерированы за проход скрипта *genTests.*

Имена соответствуют номеру теста в генераторе.

## Параллельная версия

### Общая структура

Реализованные OpenMP и TBB версии имеют одинаковую структуру, поэтому их можно представить в виде одной схемы:

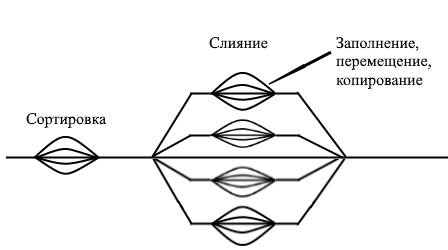


Рис. 2. Схема параллелизации.

На участке «сортировка» исходный массив разбивается на столько частей, сколько потоков было задано. По прохождении этого участка получается массив, состоящий из отсортированных подмассивов.

На участке «слияние» происходит слияние подмассивов парами. Пусть есть 8 подмассивов, тогда слияние будет выполнено так:

1. слияние первого и второго в первый, слияние третьего и четвертого в третий, слияние пятого и шестого в пятый, слияние седьмого и восьмого в седьмой («в первый» означает, что два подмассива буду записаны в первый подмассив);
2. слияние первого и третьего в первый, слияние пятого и седьмого в пятый;
3. слияние первого и пятого в первый;
4. первый будет записан в исходный массив.

### OpenMP

Запуск

ompsolver <входной файл> <выходной файл> <потоки>

<входной файл> и <выходной файл> - бинарные

Файл sol.cpp

#include <omp.h>

#include <vector>

#include <algorithm>

using namespace std;

void hoaraSortOMP(double\* &arr, int first, int last){

int leftIndex = first,

rightIndex = last;

double tmp,

pivot = arr[(first + last) / 2];

do {

while (arr[leftIndex] < pivot)

leftIndex++;

while (arr[rightIndex] > pivot)

rightIndex--;

if (leftIndex <= rightIndex)

{

if (leftIndex < rightIndex)

{

tmp = arr[leftIndex];

arr[leftIndex] = arr[rightIndex];

arr[rightIndex] = tmp;

}

leftIndex++;

rightIndex--;

}

} while (leftIndex <= rightIndex);

if (leftIndex < last)

hoaraSortOMP(arr, leftIndex, last);

if (first < rightIndex)

hoaraSortOMP(arr, first, rightIndex);

}

int binSearch(double x, double\* &vec, int N){

int size = N / 2;

int median = N / 2;

while (true) {

size = (size == 0) ? 1 : (size /= 2);

if (vec[median] < x) {

if (vec[median + 1] >= x)

return median;

else

median += size;

}

else if (vec[median] > x) {

if (vec[median + 1] <= x)

return median;

else

median -= size;

}

else {

while (median > 0 && vec[median] == x)

median--;

return median;

}

if (median == 0)

return -1;

else if (median == N)

return N;

}

}

double\* vectorToDouble(vector<double>& vec){

double\* result = new double[vec.size()];

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

result[i] = vec[i];

return result;

}

vector<double> mergeDivideConquer(vector<double>& left,vector<double>& right, int threads) {

vector<double> result;

if ((left.size() + right.size()) <= 100) {

result.resize(left.size() + right.size());

merge(left.begin(), left.end(), right.begin(), right.end(), result.begin());

return result;

}

int elementsThread = left.size() / threads;

int otherElements = left.size() - threads \* elementsThread;

vector<vector<double>> lefts(threads);

vector<vector<double>> rights(threads);

for(int i =0; i < threads; ++i) {

int k = i \* elementsThread + ((i < otherElements) ? i : otherElements);

lefts.at(i).resize(elementsThread + ((i < otherElements) ? 1 : 0));

for(int j = 0; j < lefts.at(i).size(); ++j){

lefts.at(i).at(j) = left.at(k);

k++;

}

}

vector<double> tmp(right.begin(), right.end());

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

double\* doubleAr = vectorToDouble(tmp);

int bso = binSearch(lefts.at(i).at(lefts.at(i).size()-1), doubleAr, tmp.size());

rights.at(i) = \*(new vector<double>(tmp.begin(), tmp.begin() + bso + 1));

tmp = \*(new vector<double>(tmp.begin() + bso + 1, tmp.end()));

delete[] doubleAr;

}

rights.at(threads - 1) = \*(new vector<double>(tmp.begin(), tmp.end()));

#pragma parallel for shared (lefts, rights)

for(int i=0; i < threads; ++i) {

vector<double> tmp(lefts.at(i).size() + rights.at(i).size());

merge(lefts.at(i).begin(), lefts.at(i).end(), rights.at(i).begin(), rights.at(i).end(), tmp.begin());

lefts.at(i) = tmp;

}

result.resize(left.size() + right.size());

for(int i = 0; i < threads; ++i){

int k = 0;

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

k += lefts.at(z).size();

for (int j = 0; j < lefts.at(i).size(); ++j, ++k)

result.at(k) = lefts.at(i).at(j);

}

return result;

}

vector<double>& mainMerge(vector<vector<double>>& vecs, int threads) {

double iterations = log2(threads);

int i = 0;

int dis = 1;

while(i < iterations) {

#pragma omp parallel for shared(vecs,i,dis)

for(int j = 0; j < threads; j += (dis \* 2)) {

if((j + dis) < threads) {

vecs.at(j) = mergeDivideConquer(vecs.at(j), vecs.at(j + dis),threads);

}

}

dis \*= 2;

i++;

}

return vecs.at(0);

}

void IHoaraSortOMP(double\* &numbs, int N, int threads) {

omp\_set\_num\_threads(threads);

int elementsThread=N/threads;

int otherElements=N-elementsThread\*threads;

vector<vector<double>> vecs(threads);

#pragma omp parallel for shared(numbs)

for (int i = 0; i < threads; ++i) {

int leftIndex = elementsThread\*i+((i < otherElements) ? i : otherElements);

int rightIndex = leftIndex + ((i < otherElements) ? elementsThread : (elementsThread - 1));

vector<double> ar(rightIndex - leftIndex + 1);

hoaraSortOMP(numbs, leftIndex, rightIndex);

for(int j = (leftIndex); j < (rightIndex + 1); ++j)

ar[j - leftIndex] = numbs[j];

vecs.at(i) = ar;

}

vector<double> mergeResult;

if (threads > 1)

{

#pragma omp single

mergeResult = mainMerge(vecs, threads);

}

else

mergeResult = vecs.at(0);

delete[] numbs;

numbs = vectorToDouble(mergeResult);

}

### TBB

Запуск

tbbsolver <входной файл> <выходной файл> <потоки>

<входной файл> и <выходной файл> - бинарные

Файл sol.h

#pragma once

#include <vector>

#include <algorithm>

#include "tbb/task\_scheduler\_init.h"

#include "tbb/parallel\_for.h"

#include "tbb/tbb.h"

void hoaraSortTBB(double\* &arr, int first, int last){

int leftIndex = first,

rightIndex = last;

double tmp,

pivot = arr[(first + last) / 2];

do {

while (arr[leftIndex] < pivot)

leftIndex++;

while (arr[rightIndex] > pivot)

rightIndex--;

if (leftIndex <= rightIndex)

{

if (leftIndex < rightIndex)

{

tmp = arr[leftIndex];

arr[leftIndex] = arr[rightIndex];

arr[rightIndex] = tmp;

}

leftIndex++;

rightIndex--;

}

} while (leftIndex <= rightIndex);

if (leftIndex < last)

hoaraSortTBB(arr, leftIndex, last);

if (first < rightIndex)

hoaraSortTBB(arr, first, rightIndex);

}

int binSearch(double x, double\* &vec, int N){

int size = N / 2;

int median = N / 2;

while (true) {

size = (size == 0) ? 1 : (size /= 2);

if (vec[median] < x) {

if (vec[median + 1] >= x)

return median;

else

median += size;

}

else if (vec[median] > x) {

if (vec[median + 1] <= x)

return median;

else

median -= size;

}

else {

while (median > 0 && vec[median] == x)

median--;

return median;

}

if (median == 0)

return -1;

else if (median == N)

return N;

}

}

std::vector<double> mergeSorted(std::vector<double>& left, std::vector<double>& right) {

std::vector<double> result(left.size() + right.size());

merge(left.begin(), left.end(), right.begin(), right.end(), result.begin());

return result;

}

double\* vectorToDouble(std::vector<double>& vec){

double\* result = new double[vec.size()];

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

result[i] = vec[i];

return result;

}

std::vector<double> mergeDivideConquer(std::vector<double>& left, std::vector<double>& right, int threads) {

if ((left.size() + right.size()) <= 1000) {

return mergeSorted(left, right);

}

int tmpThreads = (int)log2((int)(log10(left.size() + right.size())));

int localThreads = (tmpThreads !=0)?((tmpThreads < threads)? tmpThreads :threads):1;

int elementsThread = left.size() / localThreads;

int otherElements = left.size() - localThreads \* elementsThread;

std::vector<std::vector<double>> lefts(localThreads);

std::vector<std::vector<double>> rights(localThreads);

tbb::parallel\_for(size\_t(0), size\_t(localThreads), [&](size\_t i) {

int k = i \* elementsThread + ((i < otherElements) ? i : otherElements);

lefts.at(i).resize(elementsThread + ((i < otherElements) ? 1 : 0));

for(int j = 0; j < lefts.at(i).size(); ++j){

lefts.at(i).at(j) = left.at(k);

k++;

}

});

std::vector<double> tmp(right.begin(), right.end());

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

double\* doubleAr = vectorToDouble(tmp);

int bso = binSearch(lefts.at(i).at(lefts.at(i).size()-1), doubleAr, tmp.size());

rights.at(i) = \*(new std::vector<double>(tmp.begin(), tmp.begin() + bso + 1));

tmp = \*(new std::vector<double>(tmp.begin() + bso + 1, tmp.end()));

delete[] doubleAr;

}

rights.at(localThreads - 1) = \*(new std::vector<double>(tmp.begin(), tmp.end()));

tbb::task\_scheduler\_init init(tbb::task\_scheduler\_init::deferred);

init.initialize(localThreads);

tbb::parallel\_for(size\_t(0), size\_t(localThreads), [&lefts, &rights](size\_t i) {

lefts.at(i) = mergeSorted(lefts.at(i), rights.at(i));

});

init.terminate();

std::vector<double> result(left.size() + right.size());

init.initialize(localThreads);

tbb::parallel\_for(size\_t(0),size\_t(localThreads),[&](size\_t i){

int k = 0;

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

k += lefts.at(z).size();

for (int j = 0; j < lefts.at(i).size(); ++j, ++k)

result.at(k) = lefts.at(i).at(j);

});

init.terminate();

return result;

}

std::vector<double>& mainMerge(tbb::task\_scheduler\_init& tbbTSI, std::vector<std::vector<double>>& vecs, int threads) {

double iterations = log2(threads);

int i = 0;

int dis = 1;

while(i < iterations) {

int curIterations = (int)(threads / (dis \* 2));

tbbTSI.initialize(threads);

tbb::parallel\_for(size\_t(0), size\_t(curIterations), [&](size\_t j) {

int leftVectorIndex = j \* dis \* 2;

if ((leftVectorIndex + dis) < threads) {

vecs[leftVectorIndex] = mergeDivideConquer(vecs[leftVectorIndex], vecs[leftVectorIndex + dis], threads);

}

});

tbbTSI.terminate();

dis \*= 2;

i++;

}

return vecs.at(0);

}

std::vector<std::vector<double>> sortThread(tbb::task\_scheduler\_init& tbbTSI, double\* &numbs, int N, int threads) {

int elementsThread = N / threads;

int otherElements = N - elementsThread \* threads;

std::vector<std::vector<double>> resultVectors(threads);

tbbTSI.initialize(threads);

tbb::parallel\_for(size\_t(0), size\_t(threads),[&](size\_t i) {

int leftIndex = elementsThread \* i + ((i < otherElements) ? i : otherElements);

int rightIndex = leftIndex + ((i < otherElements) ? elementsThread : (elementsThread - 1));

resultVectors.at(i) = std::vector<double>(rightIndex - leftIndex + 1);

hoaraSortTBB(numbs, leftIndex, rightIndex);

for (int j = leftIndex; j < (rightIndex + 1); ++j)

resultVectors[i].at(j-leftIndex) = numbs[j];

});

tbbTSI.terminate();

return resultVectors;

}

void IHoaraSortTBB(double\* &numbs, int N, int threads) {

tbb::task\_scheduler\_init init(tbb::task\_scheduler\_init::deferred);

std::vector<std::vector<double>> vecs = sortThread(init, numbs, N, threads);

std::vector<double> mergeResult;

if (threads > 1)

mergeResult = mainMerge(init, vecs, threads);

else

mergeResult = vecs.at(0);

delete[] numbs;

numbs = vectorToDouble(mergeResult);

}

## Статистика

Сортировка массива из 10 млн. элементов с использованием двух, четырех и восьми потоков.

Было сгенерировано и запущено 100 тестов.

Рис. 3. OpenMP

Из графика видно, что чем больше потоков, тем больше и ускорение.

Рис. 4. TBB

Из графика можно сделать такой же вывод, как выше.

Рис.5. OpenMP и TBB на 2 потоках

Рис.6. OpenMP и TBB на 4 потоках

Рис.7. OpenMP и TBB на 8 потоках

Из трех графиков выше видно, что в принципе с помощью TBB можно достичь большего ускорения.

## Вывод

Были изучены средства параллелизации OpenMP и TBB.

Были проведены опыты, в ходе которых, средство TBB оказалось в среднем эффективнее OpenMP.