Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Ускорение быстрой сортировки с помощью распараллеливания»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

В данном реферате показан метод ускорения быстрой сортировки, используя распараллеливание, проведено сравнение и результат представлен на графике. Результаты времени записываются в файл.

#include <iostream>

#include<Windows.h>

#include <conio.h>

#include <vector>

#include <random>

#include <chrono>

#include <future>

#include <thread>

#include <fstream>

using namespace std;

void quick\_sort(vector<int>& e, int first, int last) { // быстрая сортировка

int middle;

int l = first, r = last;

middle = e[(l + r) / 2];

while (l < r) {

while (e[l] < middle) l++;

while (e[r] > middle) r--;

if (l <= r) {

swap(e[l], e[r]);

l++;

r--;

}

}

if (first < r) quick\_sort(e, first, r);

if (l < last) quick\_sort(e, l, last);

}

void parallel\_quick\_sort(vector<int>& b, int first, int last, int n) {

int middle = (first + last) / 2;

auto future1 = async(launch::async, quick\_sort, ref(b), first, middle - 1);

auto future2 = async(launch::async, quick\_sort, ref(b), middle, n - 1);

future1.wait();

future2.wait();

int i = 0; int j = middle;

int k = 0;

vector<int> c(n);

while (i != middle && j != n) {

if (b[i] < b[j]) {

c[k] = b[i];

k++;

i++;

}

else {

c[k] = b[j];

k++;

j++;

}

}

if (i == middle) {

while (j != n) {

c[k] = b[j];

k++;

j++;

}

}

else if (j == n) {

while (i != middle) {

c[k] = b[i];

k++;

i++;

}

}

b = c;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int n = 250;

unsigned long long int time1 = 0;

unsigned long long int time2 = 0;

ofstream fout("quick\_sort.txt");

ofstream fou("parallel\_quick\_sort.txt");

while (n != 25000) {

cout << "Количество элементов = " << n << '\n';

vector<int> a(n);

vector<int> b(n);

for (int j = 0; j < 100; j++) {

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

a[i] = b[i] = rand() % 20000 - 10000;

}

auto start = chrono::high\_resolution\_clock::now();

quick\_sort(a, 0, n - 1);

auto end = chrono::high\_resolution\_clock::now();

auto time = chrono::duration\_cast<std::chrono::microseconds>(end - start);

time1 += time.count();

start = chrono::high\_resolution\_clock::now();

parallel\_quick\_sort(b, 0, n - 1, n);

end = chrono::high\_resolution\_clock::now();

time = chrono::duration\_cast<std::chrono::microseconds>(end - start);

time2 += time.count();

}

fout << time1 << '\n';

fou << time2 << '\n';

if (n < 2500) {

n += 250;

}

else if (n < 10000) {

n += 500;

}

else if (n < 15000) {

n += 1000;

}

else {

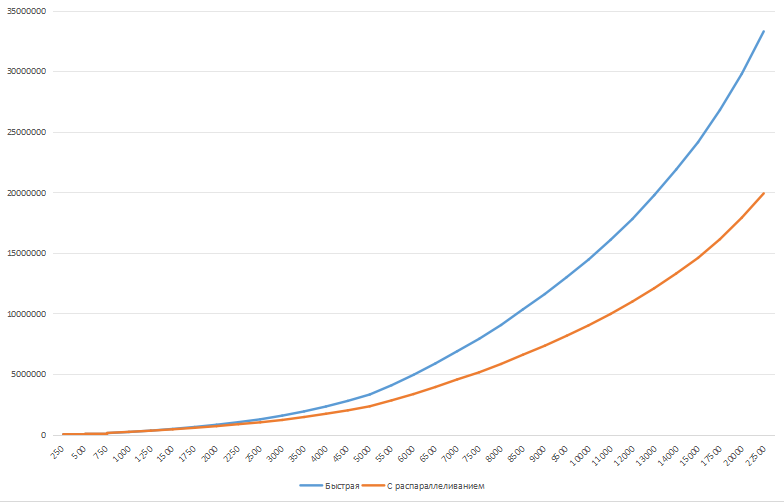
n += 2500;

}

}

}

По результат выполнения был построен график зависимости времени от количества элементов в массиве.



По результатам сравнения были сделаны следующие выводы:

1) данный алгоритм при сортировке массива, в котором количество элементов меньше 1000, не дает ускорения базовой версии быстрой сортировки

2) однако, данный алгоритм дает значительный выигрыш в скорости, если количество элементов в массиве превышает 1000 элементов.

Подводя итоги, можно сказать, что распараллеливание алгоритмов сортировки может быть эффективным при работе с большими объемами данных и может привести к значительному улучшению производительности. Однако, для небольших массивов, где накладные расходы на распараллеливание могут превышать выигрыш, базовые версии алгоритмов сортировки могут оставаться более оптимальным выбором.