Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 5

по дисциплине Теория параллельных алгоритмов

Тема: Сортировка Шелла, быстрая сортировка

Выполнила: ст. гр. 36/1

Фролова А.А.

Проверила: Патыковская М.В.

Краснодар

2024

**Цель работы:**

Построить последовательную и параллельную программы для сортировки массива методом Шелла и быстрой сортировки. Проследить зависимость времени работы от количества потоков и ядер.

**Ход работы.**

В ходе замера времени работы последовательной и параллельной реализаций получили следующие данные:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n\потоки | 2 | 3 | 4 | 5 | 6 | 7 | 8 | послед |
| 1000 | 0,484 | 0,504 | 0,496 | 0,533 | 0,521 | 0,502 | 0,506 | 0,503 |
| 2000 | 1,228 | 1,106 | 1,158 | 1,186 | 1,313 | 1,355 | 1,155 | 1,118 |
| 4000 | 2,288 | 2,405 | 2,127 | 2,157 | 2,225 | 2,492 | 2,348 | 2,147 |
| 8000 | 4,31 | 3,804 | 3,844 | 4,224 | 4,167 | 4,276 | 4,043 | 4,002 |
| 10000 | 5,212 | 4,686 | 4,898 | 4,812 | 5,039 | 4,675 | 5,42 | 4,772 |

График зависимости времени работы от количества потоков:

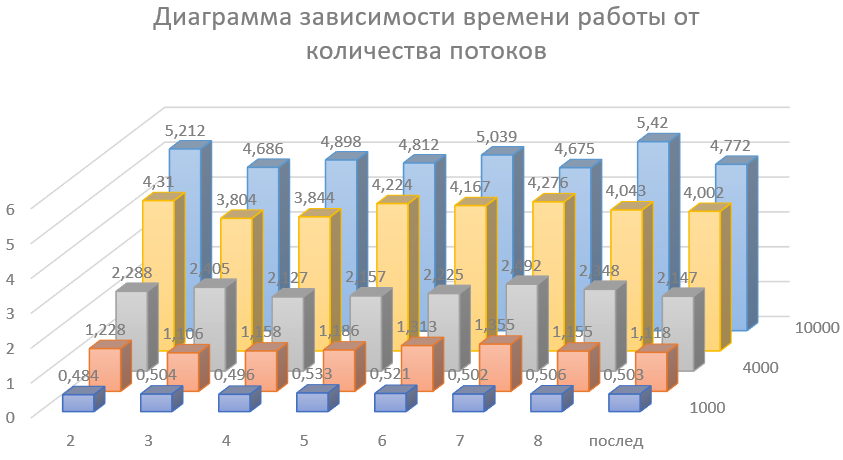


Рисунок 1 – Диаграмма зависимости времени работы от количества потоков

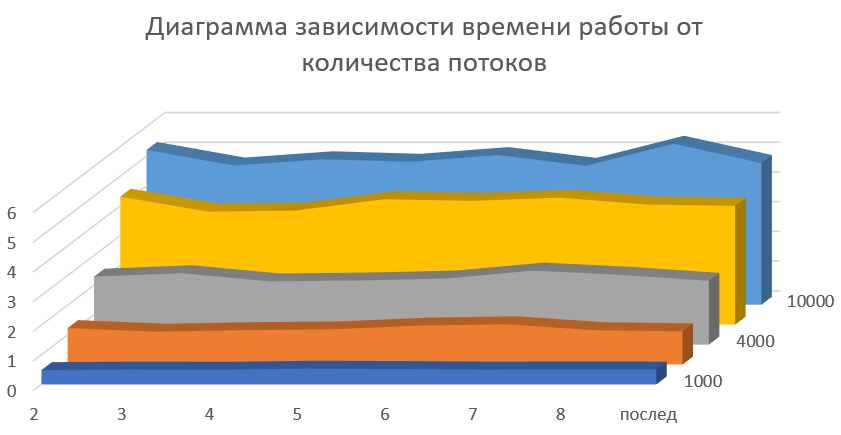


Рисунок 2 – Диаграмма зависимости времени работы программы от количества потоков и размерности данных

На диаграмме представлена зависимости времени работы программы от количества потоков. Можно заметить, что при увеличении размерности системы, увеличивается и время работы алгоритма. Оптимальное время достигается тогда, когда количество потоков равно 3-5. Это связано с количеством ядер компьютера, на котором производились вычисления (4 ядра).

Сравнив результаты работы последовательной и параллельной программы, можно сделать вывод о том, что обе программы хорошо работают при малой размерности системы. С ростом этой величины последовательный алгоритм замедляет свою работу, в то время как параллельный все еще работает довольно быстро.

Быстрая сортировка

Данные, полученные в результате экспериментов:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n\потоки | 2 | 3 | 4 | 5 | 6 | 7 | 8 | послед |
| 1000 | 0,384 | 0,504 | 0,446 | 0,531 | 0,524 | 0,512 | 0,503 | 0,597 |
| 2000 | 1,258 | 1,116 | 1,143 | 1,179 | 1,323 | 1,255 | 1,155 | 1,243 |
| 4000 | 2,288 | 2,405 | 2,107 | 2,157 | 2,225 | 2,492 | 2,348 | 2,52 |
| 8000 | 4,31 | 3,804 | 3,324 | 4,224 | 4,167 | 4,276 | 4,043 | 4,943 |
| 10000 | 5,312 | 4,886 | 4,497 | 4,913 | 5,14 | 5,475 | 5,42 | 5,706 |

Полученные диаграммы:

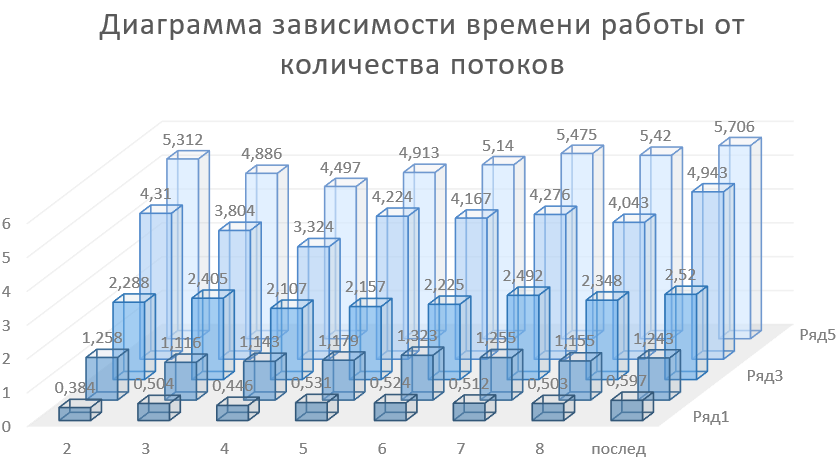


Рисунок 3 – Диаграмма зависимости времени работы от количества потоков

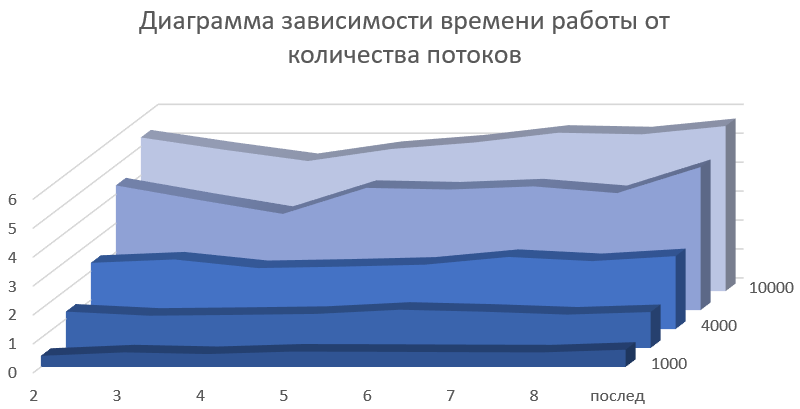
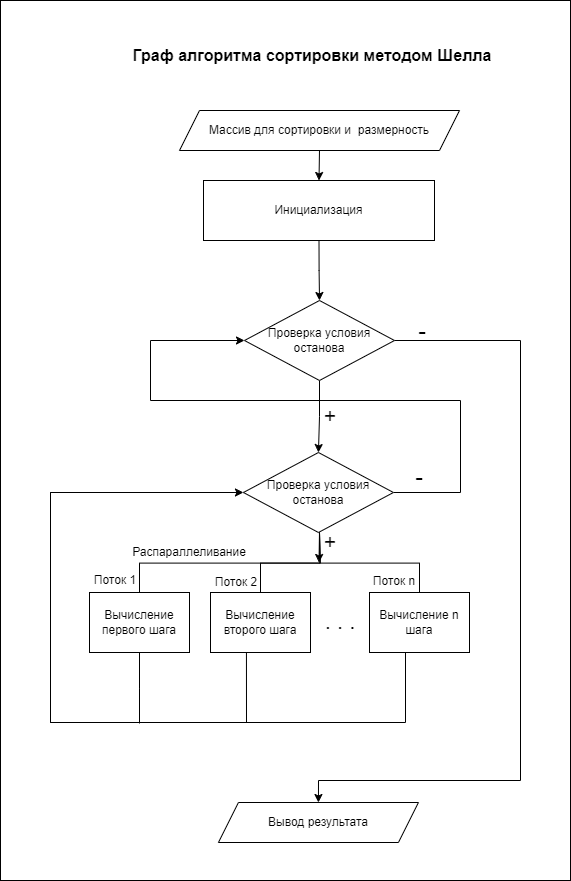
****

Рисунок 4 – Диаграмма зависимости времени работы от количества потоков

**Граф алгоритма**

****

**Листинг программ**

Сортировка Шелла:

- Последовательно:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <omp.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "rus");

clock\_t start = clock();

int n = 1000;

int digitals[1000]; // объявили массив на n ячеек

cout << "Массив для сортировки:\n";

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

digitals[i] = rand() % 10 + 2;

cout << digitals[i] << " ";

}

for (int step=n/2; step > 0; step /= 2) {

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

int tmp = digitals[i];

int j = i;

for (int j = i; j >= step; j -= step)

{

if (tmp < digitals[j - step])

digitals[j] = digitals[j - step];

else

break;

}

digitals[j] = tmp;

}

}

cout << "\nМассив в отсортированном виде:\n";

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

cout << digitals[i] << " "; // выводим элементы массива

}

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения: %f секунд\n", seconds);

system("pause");

return 0;

}

- Параллельно:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <omp.h>

using namespace std;

int main() {

setlocale(LC\_ALL, "rus");

clock\_t start = clock();

int n = 1000;

int digitals[1000]; // объявили массив на n ячеек

cout << "Массив для сортировки:\n";

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

digitals[i] = rand() % 10 + 2;

cout << digitals[i] << " ";

}

for (int step=n/2; step > 0; step /= 2) {

#pragma omp parallel for num\_threads(8) // Указываем количество потоков

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

int tmp = digitals[i];

int j = i;

for (int j = i; j >= step; j -= step)

{

if (tmp < digitals[j - step])

digitals[j] = digitals[j - step];

else

break;

}

digitals[j] = tmp;

}

}

cout << "\nМассив в отсортированном виде:\n";

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

cout << digitals[i] << " "; // выводим элементы массива

}

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения: %f секунд\n", seconds);

system("pause");

return 0;

}

Быстрая сортировка:

- Последовательно:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <omp.h>

void sort(int[], size\_t, size\_t);

void swap(int[], size\_t, size\_t);

int main()

{

setlocale(LC\_ALL, "rus");

clock\_t start = clock();

int n = 1000;

int digitals[1000]; // объявили массив на n ячеек

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

digitals[i] = rand() % 10 + 2;

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

}

//int nums[] {3, 0, 6, -2, -6, 11, 3};

sort(digitals, 0, std::size(digitals)-1);

for(auto digitals : digitals)

{

std::cout << digitals << "\t";

}

std::cout << std::endl;

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения: %f секунд\n", seconds);

}

void sort(int numbers[], size\_t start, size\_t end)

{

// начальный индекс должен быть меньше конечного индекса для массива из 2 и более элементов

if (start >= end)

return;

// проверяем все элементы относительно элемента с индексом start

size\_t current {start};

for (size\_t i {start + 1}; i <= end; i++)

{

// если i-ый элемент меньше начального

if (numbers[i] < numbers[start])

{

swap(numbers, ++current, i); // меняем его с левым

}

}

swap(numbers, start, current); // Меняем выбранный (start) и последний обмененный элементы

if (current > start)

{

sort(numbers, start, current - 1); // Сортируем элементы слева

}

if (end > current + 1)

{

sort(numbers, current + 1, end); // Сортируем элементы справа

}

}

void swap(int numbers[], size\_t first, size\_t second)

{

auto temp{numbers[first]};

numbers[first] = numbers[second];

numbers[second] = temp;

}

-Параллельно:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <omp.h>

void sort(int[], size\_t, size\_t);

void swap(int[], size\_t, size\_t);

int main()

{

setlocale(LC\_ALL, "rus");

clock\_t start = clock();

int n = 1000;

int digitals[1000]; // объявили массив на n ячеек

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

digitals[i] = rand() % 10 + 2;

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

}

//int nums[] {3, 0, 6, -2, -6, 11, 3};

sort(digitals, 0, std::size(digitals)-1);

for(auto digitals : digitals)

{

std::cout << digitals << "\t";

}

std::cout << std::endl;

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\nВремя выполнения: %f секунд\n", seconds);

}

void sort(int numbers[], size\_t start, size\_t end)

{

// начальный индекс должен быть меньше конечного индекса для массива из 2 и более элементов

if (start >= end)

return;

// проверяем все элементы относительно элемента с индексом start

size\_t current {start};

#pragma omp parallel for num\_threads(8) // Указываем количество потоков

for (size\_t i {start + 1}; i <= end; i++)

{

// если i-ый элемент меньше начального

if (numbers[i] < numbers[start])

{

swap(numbers, ++current, i); // меняем его с левым

}

}

swap(numbers, start, current); // Меняем выбранный (start) и последний обмененный элементы

if (current > start)

{

sort(numbers, start, current - 1); // Сортируем элементы слева

}

if (end > current + 1)

{

sort(numbers, current + 1, end); // Сортируем элементы справа

}

}

void swap(int numbers[], size\_t first, size\_t second)

{

auto temp{numbers[first]};

numbers[first] = numbers[second];

numbers[second] = temp;

}

Вывод:

В результате выполнения лабораторной работы №5 были созданы программы для сортировки массива методами Шелла и быстрой сортировки последовательно и параллельно. В ходе работы было установлено, что оптимальное количество потоков было примерно равно количеству ядер компьютера, на котором производились расчеты – 3-5 ядер. Также можно отметить, что обе реализации довольно быстро работают при малой размерности систему, но последовательный значительно замедляется с ростом этого показателя. В дальнейшем полученные знания будут использованы для написания более сложных программ.