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

Воробьев А.Д.

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.**

**Быстрая сортировка**

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

|  |  |  |
| --- | --- | --- |
| n\потоки | 1 | 2 |
| 50000 | 0.26 | 0.51 |
| 200000 | 2.35 | 1.125 |
| 500000 | 12.123 | 3.13 |
| 1000000 | 44.72 | 9.23 |

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).

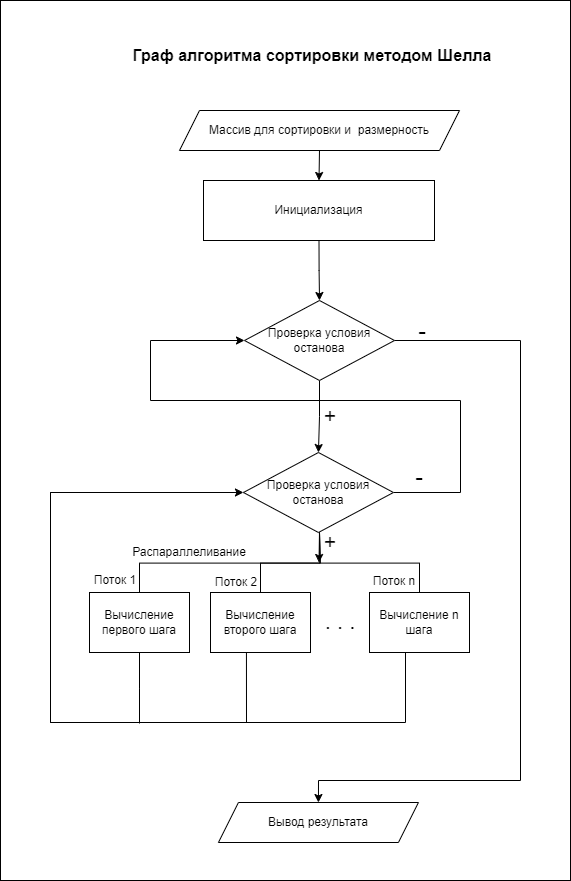
Сравнив результаты работы последовательной и параллельной программы, можно сделать вывод о том, что обе программы хорошо работают при малой размерности системы. С ростом этой величины последовательный алгоритм замедляет свою работу, в то время как параллельный все еще работает довольно быстро.

Сортировка Шеллом

Данные, полученные в результате экспериментов в секундах:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n\потоки | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1000000 | 0.36 | 0.19 | 0.14 | 0.13 | 0.12 | 0.12 | 0.12 | 0.13 |
| 10000000 | 5.06 | 3.15 | 2.43 | 2.13 | 1.88 | 1.71 | 1.92 | 1.90 |
| 30000000 | 20.1 | 12.08 | 9.35 | 7.6 | 6.65 | 6.5 | 6.97 | 6.78 |

**Граф алгоритма**



**Листинг программ**

Сортировка Шелла:

#include <stdlib.h>

#include <stdio.h>

#include <sys/time.h>

#include <string>

#include <omp.h>

void insertionsort(int a[], int n, int stride) {

for (int j=stride; j<n; j+=stride) {

int key = a[j];

int i = j - stride;

while (i >= 0 && a[i] > key) {

a[i+stride] = a[i];

i-=stride;

}

a[i+stride] = key;

}

}

void shellsort(int a[], int n)

{

int i, m;

for(m = n/2; m > 0; m /= 2)

{

#pragma omp parallel for shared(a,m,n) private (i) default(none)

for(i = 0; i < m; i++)

insertionsort(&(a[i]), n-i, m);

}

}

void printlist(std::string s, int a[], int n) {

printf("%s\n",s.c\_str());

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

printf("%d ", a[i]);

}

printf("\n");

}

int checklist(int a[], int n) {

int result = 0;

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

if (a[i] != i) {

result++;

}

}

return result;

}

void seedprng() {

struct timeval t;

/\* seed prng \*/

gettimeofday(&t, NULL);

srand((unsigned int)(1000000\*(t.tv\_sec)+t.tv\_usec));

}

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

const int n=10000000;

int nthreads;

for(nthreads=1; nthreads <= 10; nthreads++) {

double start\_time = omp\_get\_wtime();

int \*data;

int missorted;

//Define different number of threads

omp\_set\_num\_threads(nthreads);

data = (int \*)malloc(n\*sizeof(int));

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

data[i] = i;

seedprng();

/\* shuffle \*/

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

int i1 = rand() % n;

int i2 = rand() % n;

int tmp = data[i1];

data[i1] = data[i2];

data[i2] = tmp;

}

// printlist("Unsorted List:",data,n);

shellsort(data,n);

// printlist("Sorted List:",data,n);

missorted = checklist(data,n);

if (missorted != 0) printf("%d missorted nubmers\n",missorted);

double time = omp\_get\_wtime() - start\_time;

printf("Total time for thread %d (in sec):%.2f\n", nthreads, time);

}

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.