Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №3:
«Реализация алгоритма с использованием технологии OpenMP»

Одношивкин Илья, Б21-515

Описание архитектуры

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               AuthenticAMD
    Model name:          AMD Ryzen 5 5500U with Radeon Graphics
    CPU family:          23
    Model:               104
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            1
    CPU(s) scaling MHz:  52%
    CPU max MHz:         4056.0000
    CPU min MHz:         400.0000
    BogoMIPS:            4192.31
```

```
Transient hostname: DESKTOP-J2NEN3H
Icon name: computer-laptop
Chassis: laptop
Machine ID: caf94732efe24b519ce9ca85095c24f4
Boot ID: 64bd1e3a27ad44128e6b00b59b2c533f
Operating System: Fedora Linux 38 (Workstation Edition)
CPE OS Name: cpe:/o:fedoraproject:fedora:38
OS Support End: Tue 2024-05-14
OS Support Remaining: 6month 1w 5d
Kernel: Linux 6.5.6-200.fc38.x86_64
Architecture: x86-64
Hardware Vendor: HUAWEI
Hardware Model: NBM-WXX9
Firmware Version: 2.09
Firmware Date: Wed 2022-03-23
```

	total	used	free	shared	buff/cache	available
Mem:	7428976	4246056	447036	117204	2735884	2759516
Swap:	7428092	858880	6569212			

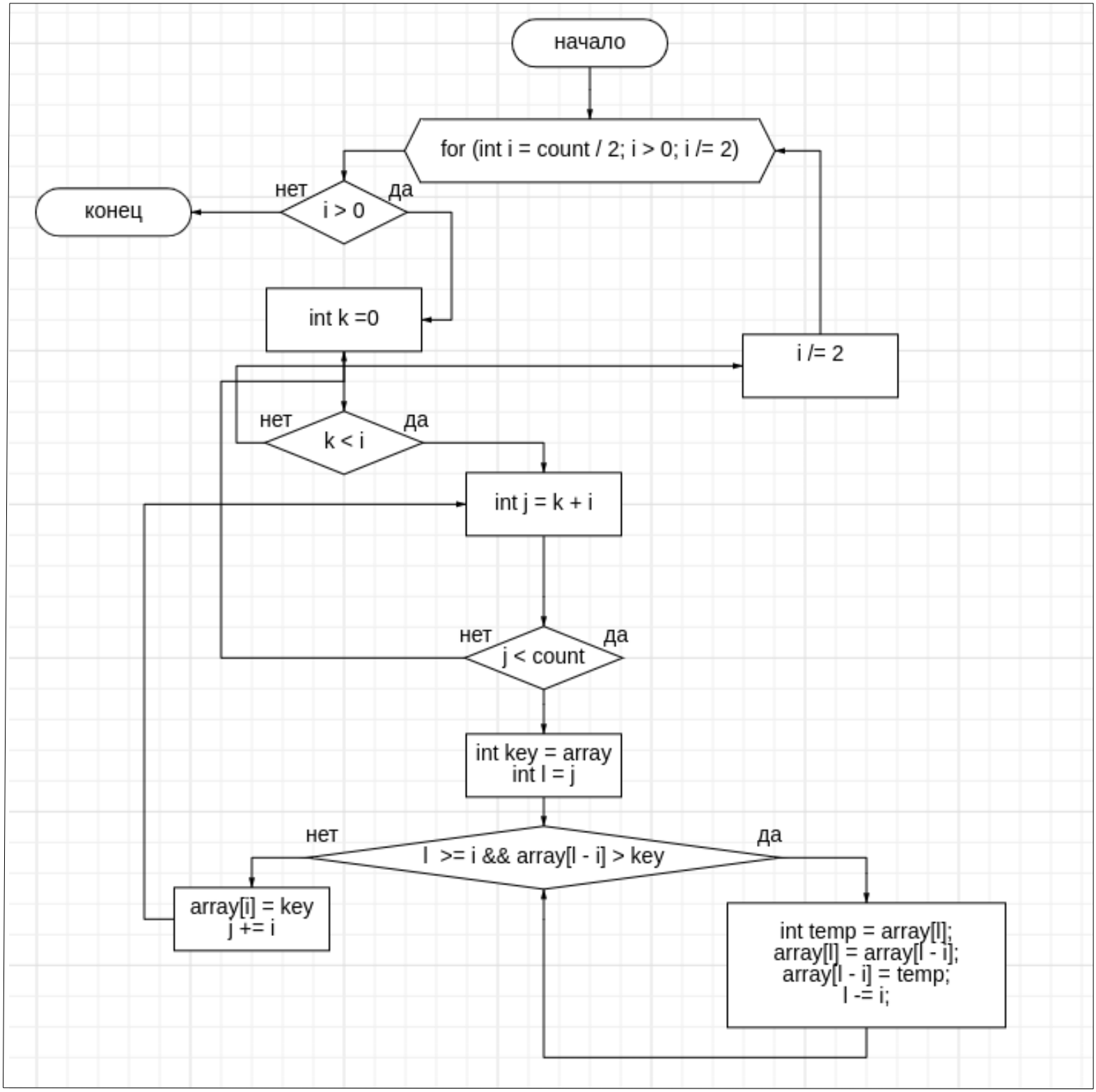
Среда разработки

Компилятор: gcc (GCC) 13.2.1 20230728 (Red Hat 13.2.1-1)
Средство сборки: Makefile
Текстовый редактор: Visual Studio Code
Версия OPENMP: 201511

Описание алгоритма

Сортировка Шелла является модификацией сортировки вставками, разбивающей массив на подмассивы и сортирующей элементы на определенных расстояниях. Основная идея заключается в предварительной сортировке элементов массива на больших интервалах, а затем уменьшении интервалов до значения 1, когда выполняется обычная сортировка вставками. Алгоритм позволяет уменьшить количество обменов элементов в сравнении с обычной сортировкой вставками и показывает лучшую производительность на некоторых видах данных.

Блок-схема алгоритма



Описание директив и функций OpenMP

parallel - директива OpenMP, которая указывает начало блока параллельной части программы. Этот блок будет выполняться параллельно в нескольких потоках.

omp_set_num_threads(threads)- директива OpenMP, которая указывает, что блок кода, следующий за директивой parallel, должен выполняться с использованием threads потоков.

shared(array, count, i) - директива, которая указывает, что указанные переменные (array, count, i) являются общими для всех потоков и будут доступны каждому потоку внутри блока parallel.

default(none) - директива OpenMP, которая устанавливает строгий режим контроля переменных.

График №1

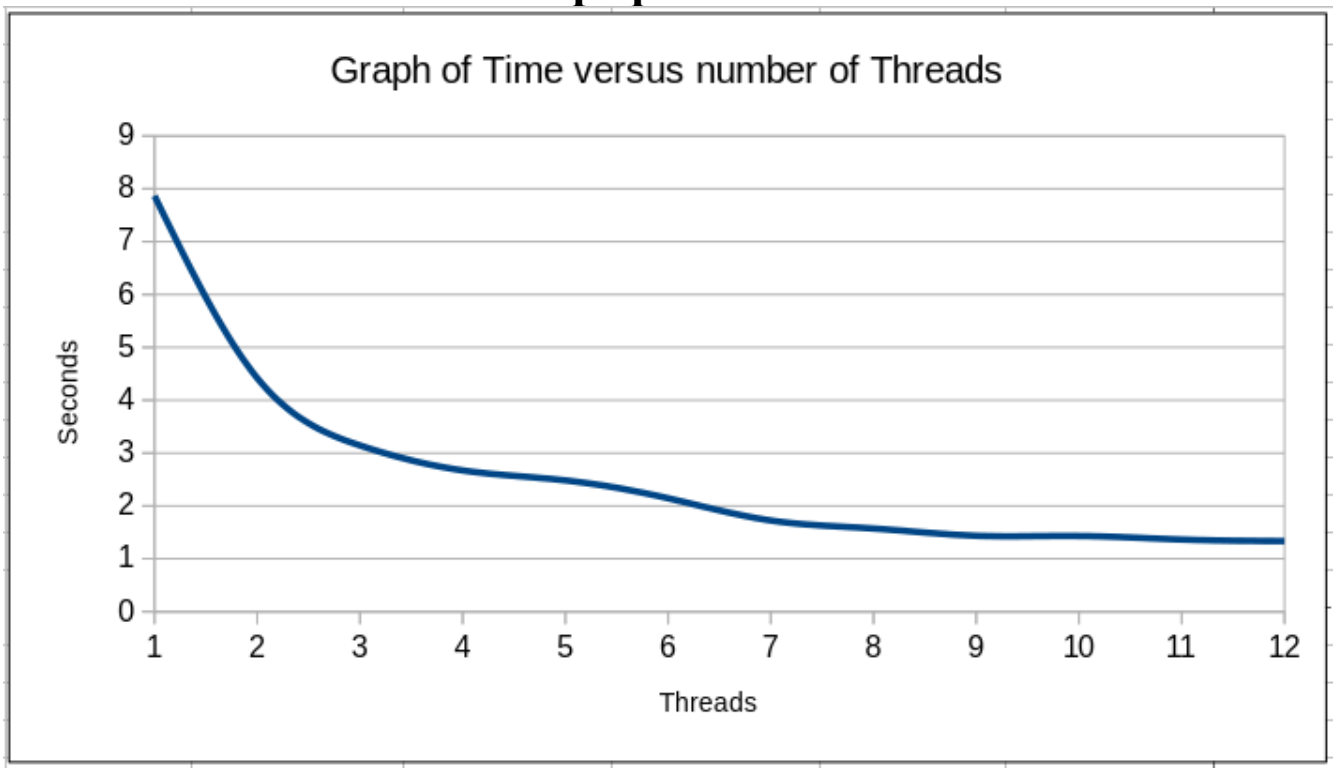


График №2

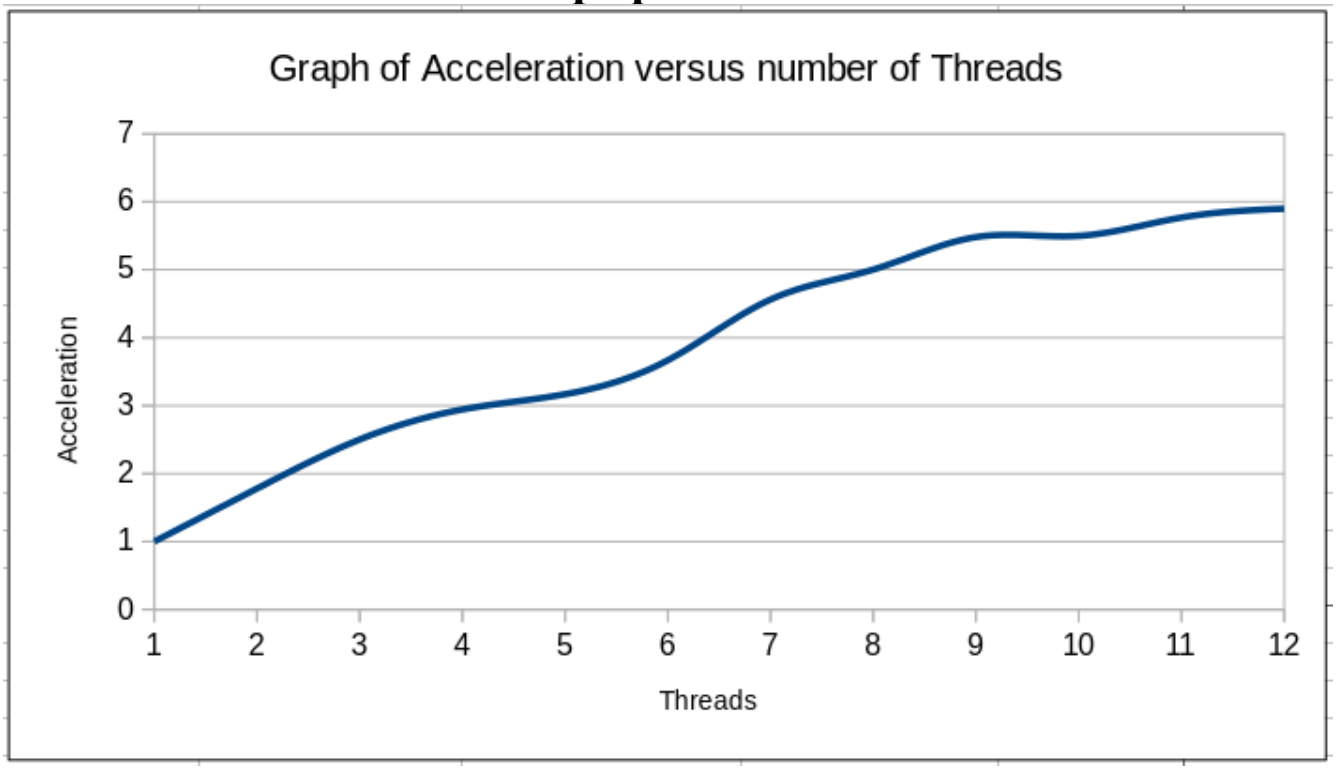


График №3

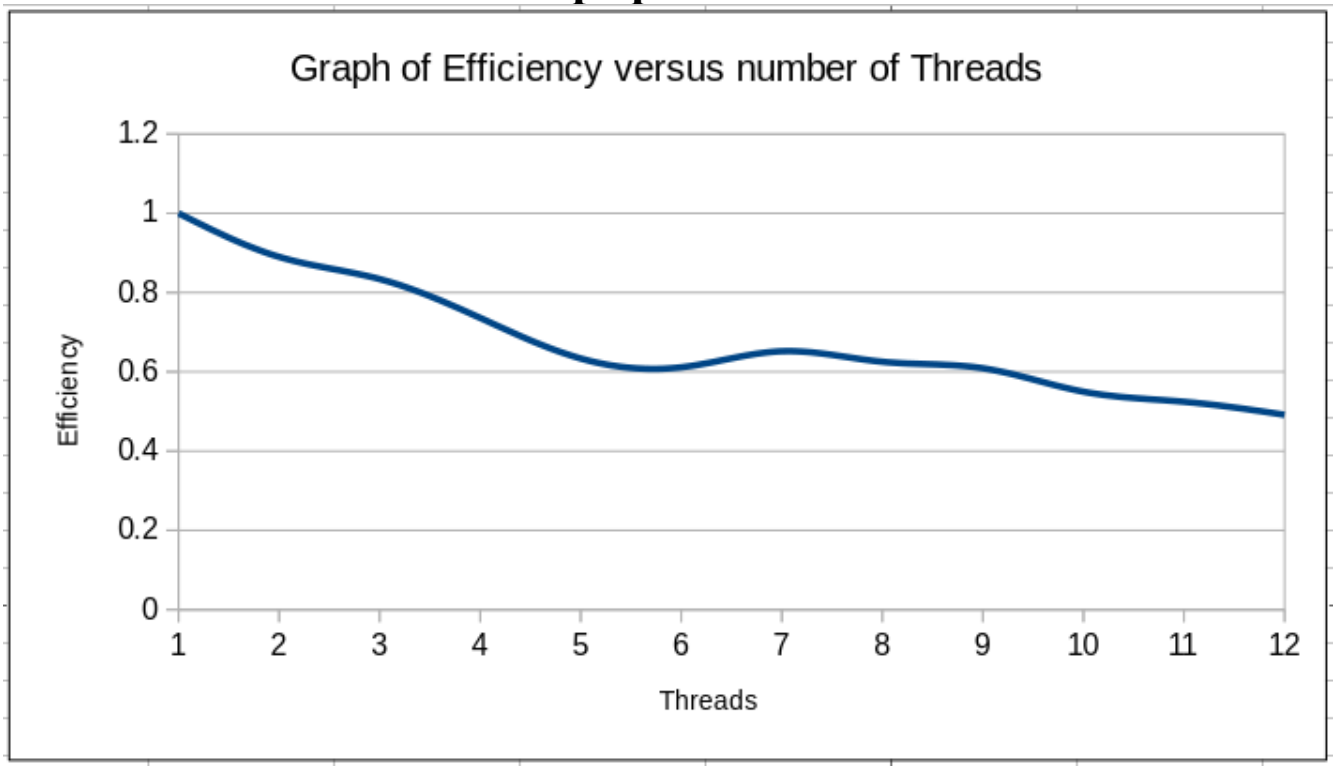


График №4

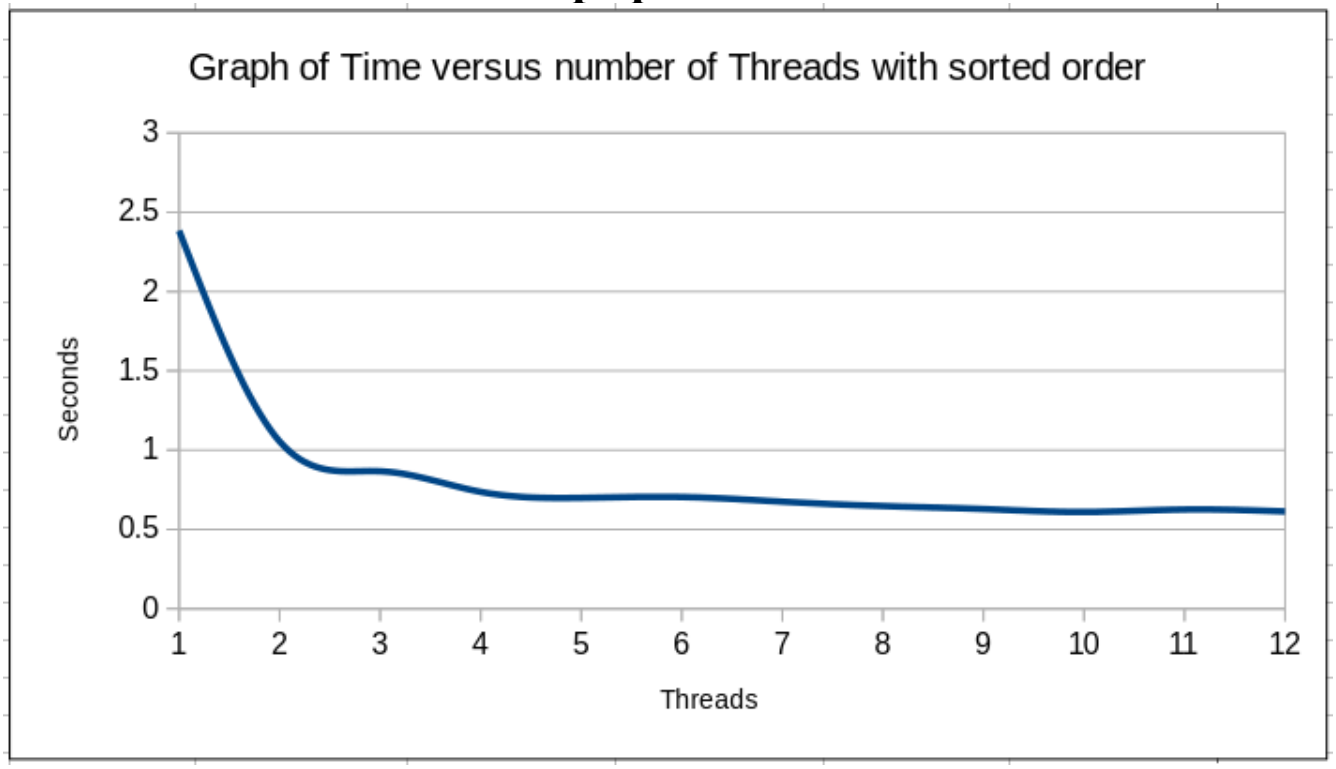


График №5

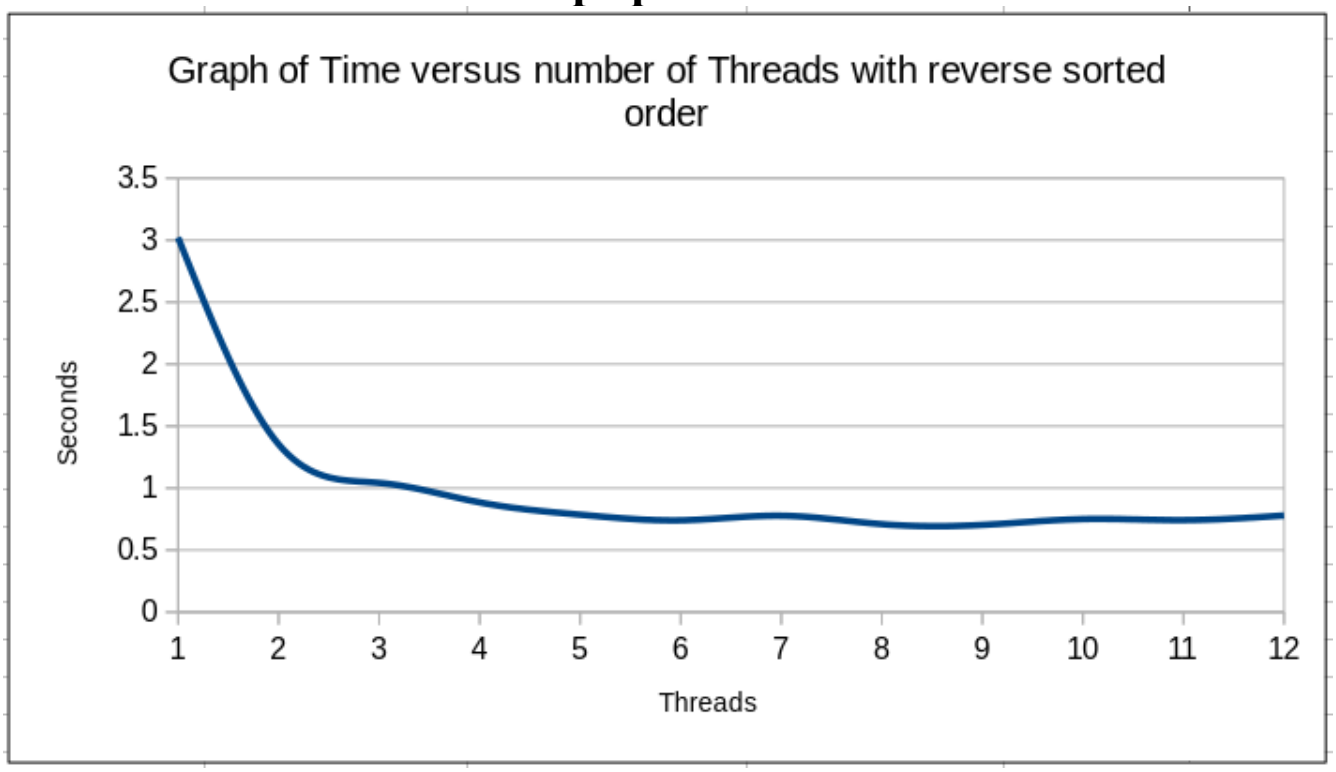


График №6

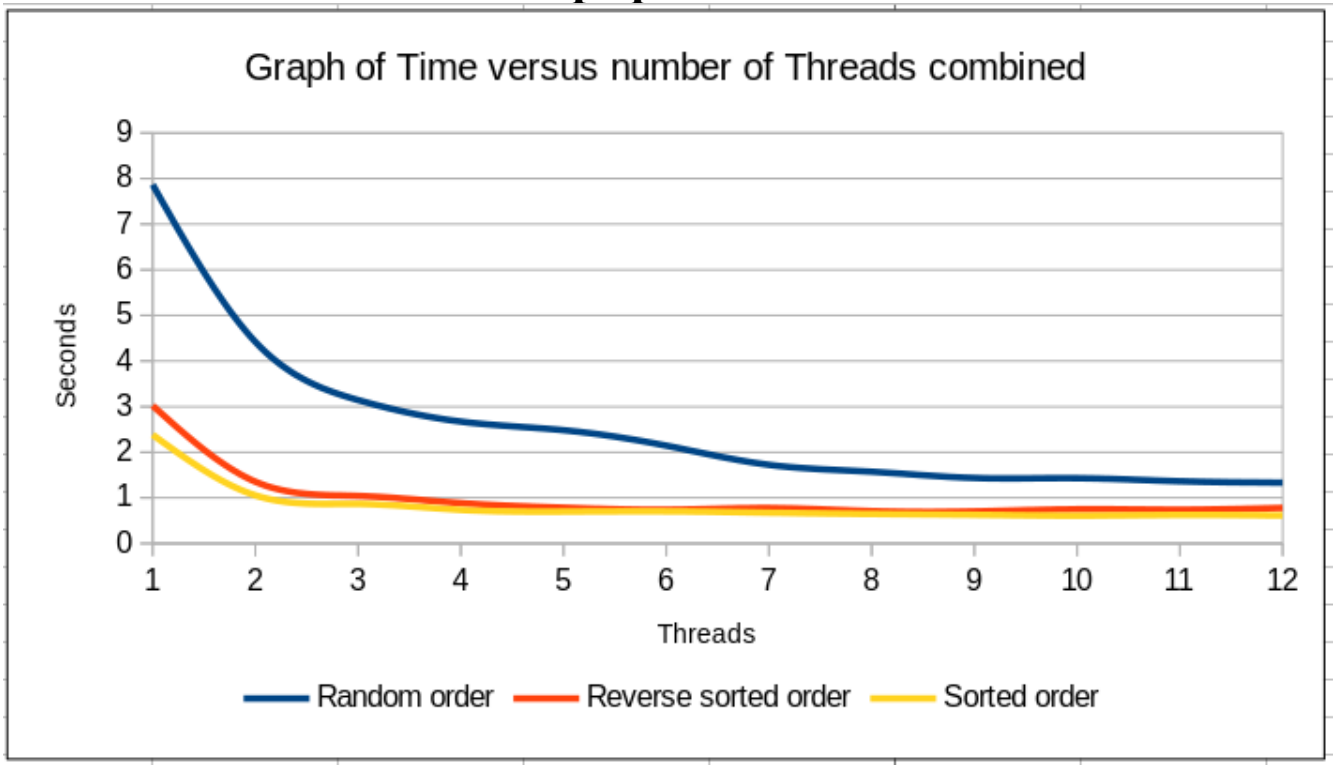


Таблица с данными

Threads		Attempt #1	Attempt #2	Attempt #3	Attempt #4	Attempt #5	Attempt #6	Attempt #7	Attempt #8	Attempt #9	Attempt #10		Random order	Acceleration	Efficiency
1		7.205017	8.417694	8.081944	7.535633	8.178634	8.136064	8.087032	7.426618	8.085773	7.508556		7.8662965	1	1
2		4.764895	4.930547	4.865805	4.843034	4.309028	4.816659	4.32849	3.69011	4.309028	3.3346		4.4192196	1.780019372651	0.890009686
3		2.922535	3.059272	3.027379	3.087743	3.034027	3.010313	3.982989	3.049286	3.130452	3.135368		3.1439364	2.502053317618	0.834017773
4		2.598083	2.675756	2.6378	2.723091	2.653646	2.60856	2.592744	2.715016	2.79889	2.715		2.6718586	2.944129041859	0.73603226
5		2.453155	2.655202	2.405435	2.460985	2.483572	2.409945	2.425284	2.441709	2.575377	2.517396		2.482806	3.168308961715	0.633661792
6		1.384605	1.917528	1.37378	2.454523	2.318871	2.433275	2.412697	2.433211	2.388709	2.327137		2.1444336	3.668239716072	0.611373286
7		2.400698	1.975998	2.48011	1.540916	1.467487	1.435052	1.409908	1.489956	1.519513	1.52154		1.7241178	4.562505241811	0.651786463
8		1.385701	1.642552	1.463793	1.511361	1.397167	1.393159	1.394032	1.442522	1.502348	2.591277		1.5723912	5.002760445365	0.625345056
9		1.359266	1.862197	1.292335	1.395366	1.307614	1.409173	1.424779	1.337248	1.540319	1.424485		1.4352782	5.480677195543	0.608964133
10		1.346808	1.759123	1.444006	1.365501	1.362441	1.388837	1.293337	1.402772	1.491865	1.451935		1.4306625	5.498359326536	0.549835933
11		1.272392	1.407646	1.358571	1.475825	1.312111	1.390339	1.257721	1.314543	1.51503	1.332584		1.3636762	5.768448917712	0.524404447
12		1.223018	1.388053	1.24133	1.576777	1.36637	1.337876	1.248358	1.319315	1.339218	1.29507		1.3335385	5.898814694889	0.491567891

		All sorted #1	All sorted #2	All sorted #3		Sorted order		Reversed sort #1	Reversed sort #2	Reversed sort #3	Reversed sort #4		Reverse sorted order		
		2.618613	2.697388	1.834684		2.3835616667		3.143588	3.124101	3.208402	2.610651		3.0216855		
		1.086546	1.002055	1.07474		1.054447		1.323876	1.29222	1.408493	1.388447		1.353259		
		0.855576	0.877026	0.866506		0.8663693333		1.021465	1.024585	1.063204	1.05926		1.0421285		
		0.729749	0.734584	0.745183		0.7365053333		0.850832	0.852985	0.93755	0.897357		0.884681		
		0.675699	0.634667	0.78569		0.6986853333		0.780701	0.767052	0.805954	0.788499		0.7855515		
		0.661483	0.697273	0.753008		0.7039213333		0.755789	0.667138	0.788868	0.748045		0.73996		
		0.705831	0.631753	0.686219		0.674601		0.817693	0.715194	0.758713	0.81657		0.7770425		
		0.672251	0.588851	0.679733		0.646945		0.706572	0.686971	0.693624	0.748737		0.708976		
		0.633311	0.613478	0.637741		0.6281766667		0.712967	0.670899	0.688109	0.739292		0.70281675		
		0.591711	0.607029	0.630239		0.6096596667		0.669438	0.658909	0.839228	0.832259		0.7499585		
		0.621085	0.602999	0.651739		0.6252743333		0.664677	0.695015	0.882655	0.723059		0.7413515		
		0.620113	0.543941	0.674494		0.6128493333		0.753344	0.730221	0.87142	0.760383		0.778842		

Код программы

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

void shell_sort(int *array, int count);

int main() {

    srand(time(NULL));

    const int count = 10000000;
    const int max_threads = 12;

    int *temp = malloc(count * sizeof(int));
    int *array = malloc(count * sizeof(int));

    for (int i = 0; i < count; i++) {
        temp[i] = rand();
        array[i] = temp[i];
    }

    for (int threads = 1; threads <= max_threads; threads++){
        double start_time = omp_get_wtime();

        omp_set_num_threads(threads);

        shell_sort(array, count);

        printf("Threads: %d. Time: %f\n", threads, omp_get_wtime() - start_time);

        for (int i = 0; i < count; i++) { array[i] = temp[i]; }
    }
    return 0;
}

void shell_sort(int *array, int count) {
    for (int i = count / 2; i > 0; i /= 2) {
        #pragma omp parallel for shared(array, count, i) default(none)
        for (int k = 0; k < i; k++) {
            for (int j = k + i; j < count; j += i) {
                int key = array[j];
                int l = j;

                while (l >= i && array[l - i] > key) {
                    int temp = array[l];
                    array[l] = array[l - i];
                    array[l - i] = temp;
                    l -= i;
                }
                array[l] = key;
            }
        }
    }
}
```

Заключение

В ходе данного исследования была разработана параллельная программа для сортировки элементов в массиве. Первоначально, на основе последовательной версии алгоритма, были построены параллельные версии, и было измерено время выполнения программы с разным числом запущенных потоков.

В результате работы программы была получена зависимость среднего времени выполнения T(n). Также была вычислена зависимость ускорения от числа потоков по формуле: A(n) = T(n)/ T(1). После этого была расчитана зависимость эффективности от числа потоков по формуле: E(n) = A(n)/n, Где T – время (в секундах) выполнения программы, n – количество потоков, A – ускорение, E – эффективность.

При добавлении дополнительных потоков уровень эффективности снижается с увеличением числа потоков. Эффективность начинается с 1 и уменьшается до 0.4915. При каждом добавлении нового потока снижение эффективности происходило в интервале [0; 0.1], что говорит о монотонной убывании. При добавлении дополнительных потоков ускорение увеличивается. Начальное значение ускорения составляет 1, а максимальное достигает 5.8988. График ускорения монотонно увеличивается в интервале [0; 1.24].

Исходя из полученных данных, мною были рассчитаны ускорение и эффективность алгоритма для разного числа потоков и построены соответствующие графики зависимости времени выполнения, ускорения и эффективности от числа запущенных потоков. Результаты показали, что увеличение числа потоков способствует ускорению программы, но каждый дополнительный поток делает это менее эффективным.

Дополнительно, я рассмотрел влияние изначальной сортировки заданного массива на время выполнения алгоритма. Путем сравнения разных сценариев, включая случаи, когда массив был сортрован по возрастанию, когда он был сортирован по убыванию, а также когда случайные элементы никак не сортрованы, я установил, что начальная прямая и обратная сортировка массива уменьшает время выполнения алгоритма. При 1 потоке время уменьшается в ~ 3 раза, при 12 потоках в ~ 2 раза.

Таким образом, результаты исследования позволяют сделать вывод, что эффективность параллельных алгоритмов зависит от количества потоков и особенностей данных, с которыми они работают.