Национальный исследовательский ядерный университет “МИФИ”Лабораторная работа №3: «Реализация алгоритма с использованием технологии OpenMP»

Шанкин Данила Б20-505

2022 год

1. Описание рабочей среды

```
danila@Danila-PK ~/Рабочий стол/parallel_lab1 neofetch                                  master
             ...-:::::-...                  danila@Danila-PK
          .-MMMMMMMMMMMMMMM-.               ----------------
      .-MMMM`..-:::::::-..`MMMM-.           OS: Linux Mint 20.2 x86_64
    .:MMMM.:MMMMMMMMMMMMMMM:.MMMM:.         Host: NBLK-WAX9X M1020
   -MMM-M---MMMMMMMMMMMMMMMMMMM.MMM-        Kernel: 5.4.0-74-generic
 `:MMM:MM`  :MMMM:....::-...-MMMM:MMM:`     Uptime: 1 hour, 21 mins
 :MMM:MMM`  :MM:`  ``    ``  `:MMM:MMM:     Packages: 2663 (dpkg), 9 (flatpak)
.MMM.MMMM`  :MM.  -MM.  .MM-  `MMMM.MMM.    Shell: zsh 5.8
:MMM:MMMM`  :MM.  -MM-  .MM:  `MMMM-MMM:    Resolution: 1920x1080
:MMM:MMMM`  :MM.  -MM-  .MM:  `MMMM:MMM:    DE: Cinnamon
:MMM:MMMM`  :MM.  -MM-  .MM:  `MMMM-MMM:    WM: Mutter (Muffin)
.MMM.MMMM`  :MM:--:MM:--:MM:  `MMMM.MMM.    WM Theme: Mint-Y-Purple (Mint-X)
 :MMM:MMM-  `-MMMMMMMMMMMM-`  -MMM-MMM:     Theme: Mint-Y [GTK2/3]
  :MMM:MMM:`                `:MMM:MMM:      Icons: Mint-Y-Teal [GTK2/3]
   .MMM.MMMM:--------------:MMMM.MMM.       Terminal: gnome-terminal
     '-MMMM.-MMMMMMMMMMMMMMM-.MMMM-'        CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (8) @ 2.100GHz
       '.-MMMM``--:::::--``MMMM-.'          GPU: AMD ATI 03:00.0 Picasso
            '-MMMMMMMMMMMMM-'               Memory: 2527MiB / 6951MiB
               ``-:::::-``
```

```
danila@Danila-PK ~/Рабочий стол/parallel_lab1 echo | cpp -fopenmp -dM | grep -i OPENMP   master
#define _OPENMP 201511
```

2. Анализ алгоритма

Схема алгоритма:

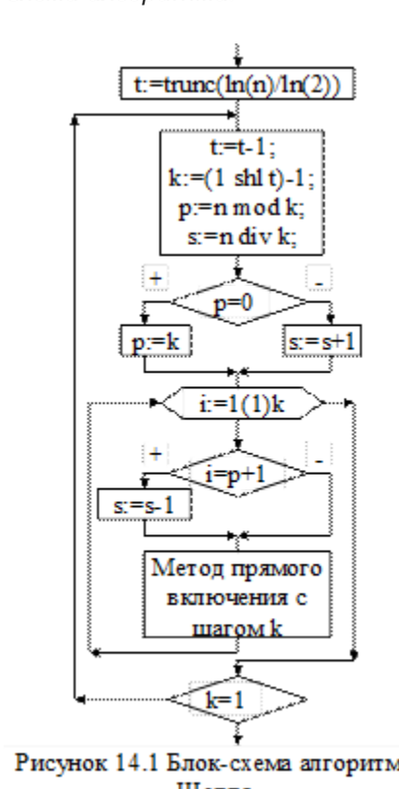


Рисунок 14.1 Блок-схема алгоритма Шелла

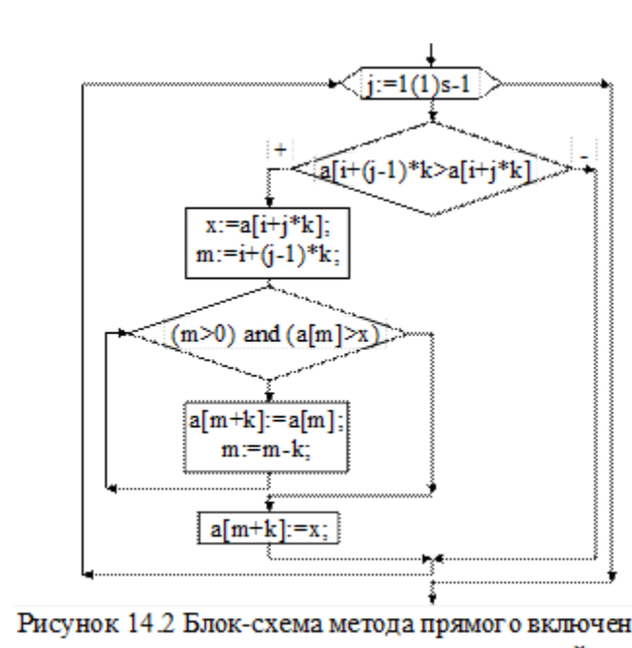


Рисунок 14.2 Блок-схема метода прямого включения для сортировки подпоследовательностей

3. Директива parallel

`#pragma omp parallel num_threads(threads) shared(array, count) reduction(max: max) default(none)`

- `pragma` - директива компилятора
- `omp` - принадлежность директивы к OpenMP
- Параллельная область задаётся при помощи директивы `parallel`
- `num_threads(int)` – явное задание количества потоков, которые будут выполнять параллельную область; по умолчанию выбирается последнее значение, установленное с помощью функции `omp_set_num_threads()`, или значение переменной `OMP_NUM_THREADS;`
- `shared(list)` – задаёт список переменных, общих для всех потоков;
- `default(private|firstprivate|shared|none)` – всем переменным в параллельной области, которым явно не назначен класс, будет назначен класс `private`, `firstprivate` или `shared` соответственно; none означает, что всем переменным в параллельной области класс должен быть назначен явно;

`#pragma omp for`

- `for` - Используется для распределения итераций цикла между различными потоками

4. Код алгоритма сортировки Шелла

In [ ]:
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define THREADS     8
#define SIZE        1000000

void InsertSort(int* arr, int i, int half, int* num_iteration){
    int temp = 0;
    int j = 0;

    for (int f = half + i; f < SIZE; f = f + half) {
        j = f;
        while(j > i && arr[j - half] > arr[j]) {
            temp = arr[j];
            arr[j] = arr[j - half];
            arr[j - half] = temp;
            *num_iteration += 1;
            j = j - half;
        }
    }
}

void ShellSort(int* array, float* times, int* iter) {
    int h;
    int i = 0;
    float t_start, t_end, t_total;
    for (int j = 0; j < THREADS; j++) {
        t_start = omp_get_wtime();
        for(h = SIZE/2; h > 0; h = h/2) {
            #pragma omp parallel num_threads(j + 1) shared(array, h, i, iter)  default(none)
            {
                #pragma omp for
                for(i = 0; i < h; i++) { InsertSort(array, i, h, iter); }
            }
        }
        t_end = omp_get_wtime();
        t_total = t_end - t_start;
        times[j] = t_total;
    }
}
```

5. Генерация различных случаев

Проверим алгоритм Шелла на чувствительность к разным исходным массивам. Выявим зависимость времени сортировки Шелла в нескольких краевых случаях:

1. Абсолютно случайно сгенерированный массив (5.1);
2. Массив, который практически полностью упорядочен (5.2);
3. Массив, упорядоченный в обратном порядке (5.3);
4. Назовем массив "странным", когда он состоит в большинстве своем из одного и того же элемента (5.4);

5.1 Абсолютно случайный массив

5.1.1 Код для генерации:

```
int* Random_array(int rand_seed) {
    int* rand_arr = calloc(SIZE, sizeof(int));
    srand(rand_seed);
    for (int i = 0; i < SIZE; i++) { rand_arr[i] = rand(); }
    return rand_arr;
}
```

5.1.2 Общее количество итераций в данном типе массива:

```
For absolutely random array number of iterations = 279493176
```

5.2 Почти упорядоченный массив:

5.2.1 Код для генерации:

```
int* Correct_array(int rand_seed) {
    int* correct_arr = calloc(SIZE, sizeof(int));
    for (int i = 0; i < SIZE; i++) { correct_arr[i] = i; }
    srand(rand_seed);
    int magic_num = 1000; //0.2% from all elements willn't be in right order
    for (int i = 0; i < magic_num; i++) {
        int target1, target2;
        target1 = rand() % SIZE;
        target2 = rand() % SIZE;
        correct_arr[target2] = correct_arr[target1];
        correct_arr[target1] = correct_arr[target2];
    }
    return correct_arr;
}
```

5.2.2 Общее количество итераций в данном типе массива:

```
For almost correct ordered array number of iterations = 54083340
```

5.3 Абсолютно неупорядоченный массив

5.3.1 Код для генерации:

```
int* Wrong_array() {
    int* wrong_arr = calloc(SIZE, sizeof(int));
    for (int i = 0; i < SIZE; i++) { wrong_arr[i] = (SIZE - i); }
    return wrong_arr;
}
```

5.3.2 Общее количество итераций в данном типе массива:

```
For wrong ordered array number of iterations = 51466272
```

5.4 "Странный" массив

5.4.1 Код для генерации:

```
int* Strange_array(int rand_seed) {
    int* strange_arr = calloc(SIZE, sizeof(int));
    srand(rand_seed);
    int repeating_elem = rand();
    for (int i = 0; i < SIZE; i++) {
        if (i < SIZE / 2) { strange_arr[i] = repeating_elem; } //in the half of array we put random element
        else { strange_arr[i] = rand(); }
    }
    return strange_arr;
}
```

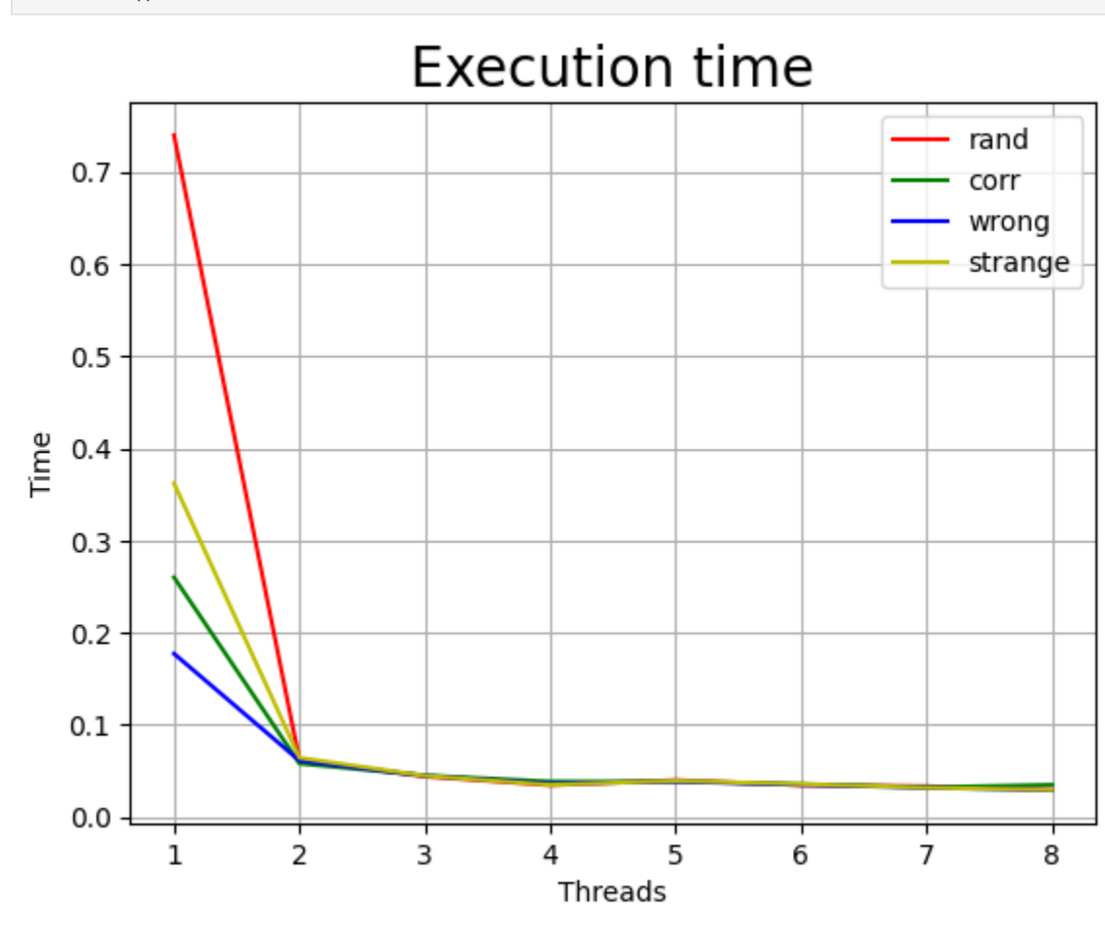
5.3.2 Общее количество итераций в данном типе массива:

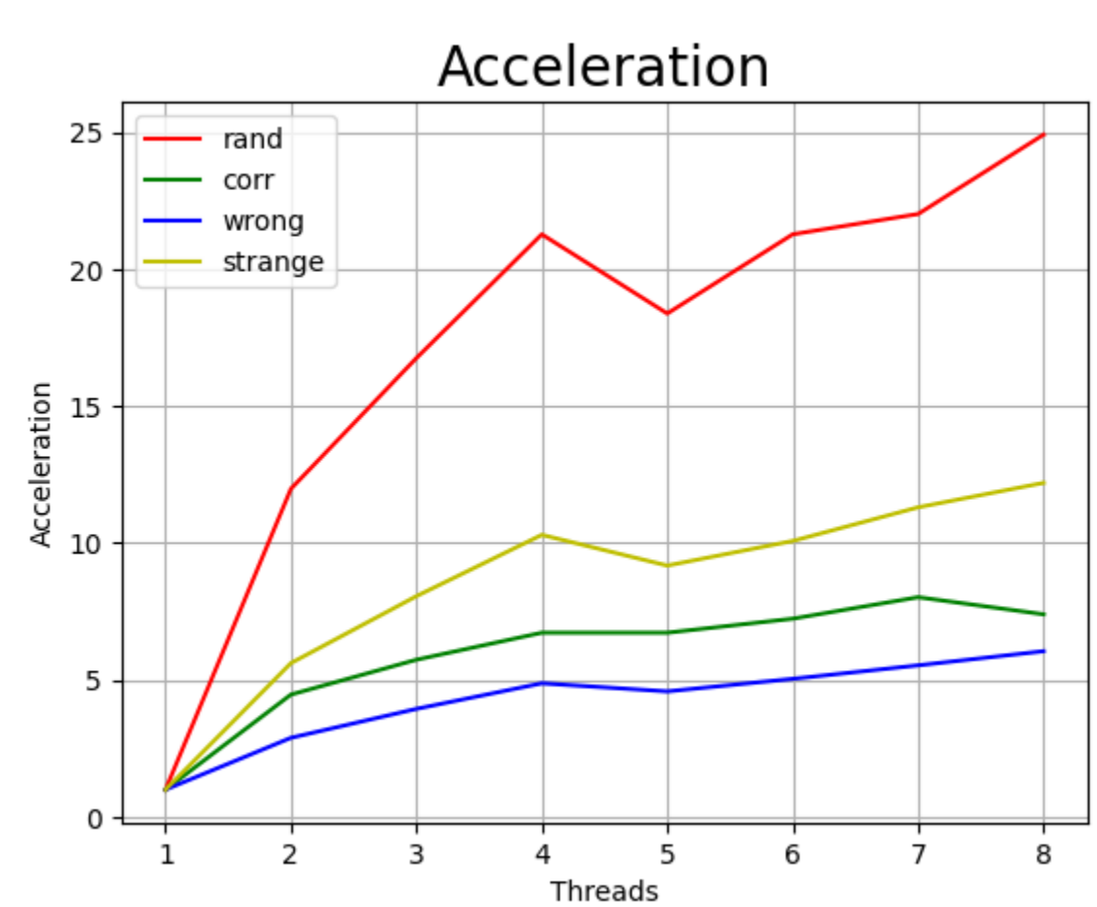
```
For 'strange' array number of iterations = 117084669
```

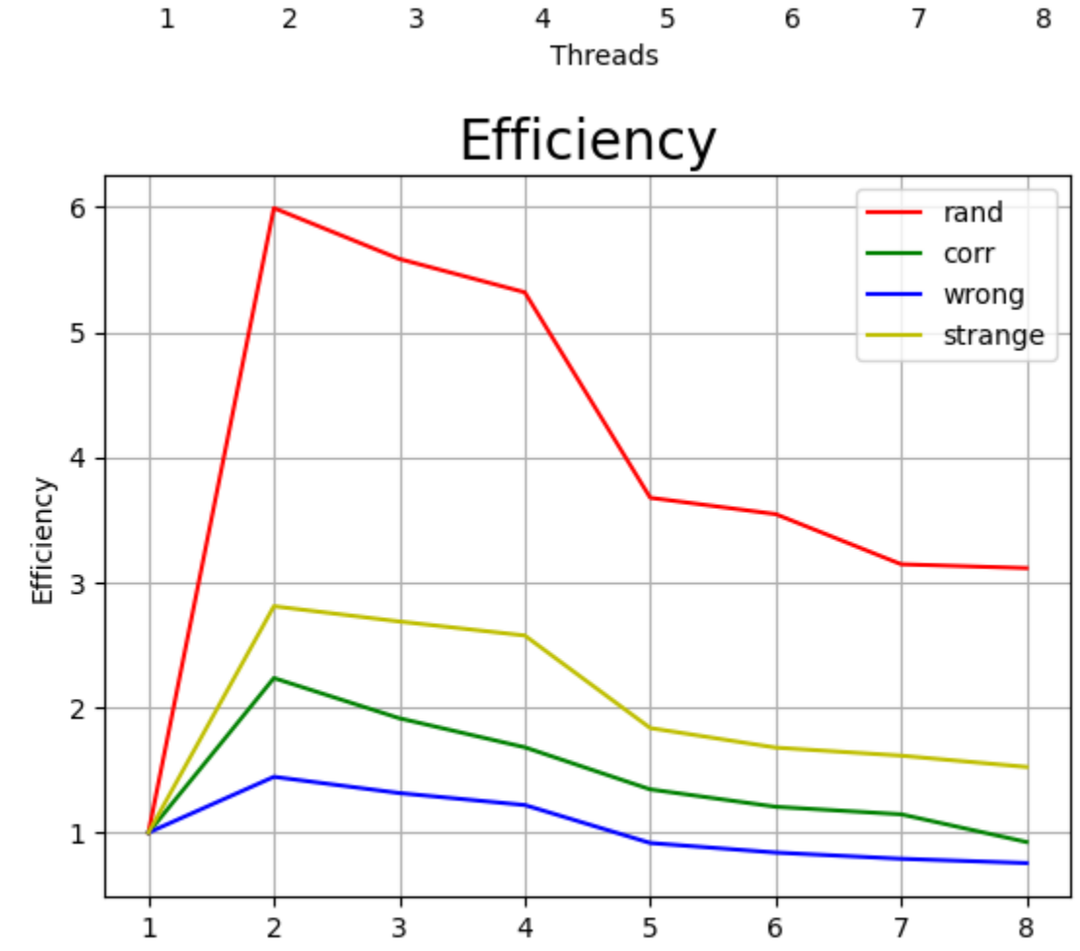
6. Код graph.py, графики и таблицы

In [4]:
```
!cp 'Рабочий стол'/parallel_lab3/d_rand.txt ./
!cp 'Рабочий стол'/parallel_lab3/d_corr.txt ./
!cp 'Рабочий стол'/parallel_lab3/d_wrong.txt ./
!cp 'Рабочий стол'/parallel_lab3/d_strange.txt ./
```

In [9]:
```
from pwn import *
from prettytable import PrettyTable
import matplotlib.pyplot as plt
import numpy as np


def inp(file_name):
    data = open(file_name, 'rb')
    try:
        num_of_threads = u32(data.read(4))
        threads = [i + 1 for i in range(num_of_threads)]
        iterations = u32(data.read(4))
        time = [[] for i in range(num_of_threads)]
        for i in range(iterations * num_of_threads):
            time[i % num_of_threads].append(float(struct.unpack('f', data.read(4))[0]))
    finally:
        data.close()
        return time, threads, num_of_threads


def plots(times, threads):

    list_t1 = []
    list_s = []
    list_e = []
    rgby = ['r','g','b','y']

    for i in range(len(times)):
        # 1
        t1 = [sum(k) / len(k) for k in times[i]]
        list_t1.append(t1)
        plt.plot(threads[i], t1, rgby[i])

    plt.title('Execution time', fontsize=20)


    plt.xlabel('Threads')
    plt.ylabel('Time')
    plt.grid(1)
    plt.legend(['rand','corr','wrong','strange'])
    plt.show()


    for i in range(len(times)):
        # 2
        s = [(sum(times[i][0]) / len(times[i][0])) / (sum(k) / len(k)) for k in times[i]]
        list_s.append(s)
        plt.plot(threads[i], s, rgby[i])


    plt.title('Acceleration', fontsize=20)

    plt.xlabel('Threads')
    plt.ylabel('Acceleration')
    plt.grid(1)
    plt.legend(['rand','corr','wrong','strange'])
    plt.show()


    for i in range(len(times)):
        # 3
        e = [list_s[i][k] / (k + 1) for k in range(len(list_s[i]))]
        list_e.append(e)
        plt.plot(threads[i], e, rgby[i])


    plt.title('Efficiency', fontsize=20)

    plt.xlabel('Threads')
    plt.ylabel('Efficiency')
    plt.grid(1)
    plt.legend(['rand','corr','wrong','strange'])
    plt.show()


# Table
def show_table(times, threads):
    title_text = 'Time table'

    fig_background_color = 'skyblue'
    fig_border = 'steelblue'

    column_headers = [i + 1 for i in range(len(times[0]))]
    row_headers = [i + 1 for i in range(threads)]

    cell_text = []
    for row in times:
        cell_text.append([f'{round(x, 5)}' for x in row])

    rcolors = plt.cm.BuPu(np.full(len(row_headers), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(column_headers), 0.1))
    plt.figure(linewidth=10,
               edgecolor=fig_border,
               facecolor=fig_background_color,
               tight_layout={'pad': 2},
               )
    the_table = plt.table(cellText=cell_text,
                          rowLabels=row_headers,
                          rowColours=rcolors,
                          rowLoc='right',
                          colColours=ccolors,
                          colLabels=column_headers,
                          loc='center')

    the_table.scale(1, 1.75)
    ax = plt.gca()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.box(on=None)
    plt.suptitle(title_text)
    plt.draw()
    plt.show()


def main():
    files = tuple(['d_rand.txt', 'd_corr.txt', 'd_wrong.txt', 'd_strange.txt'])
    times = []
    threads = []
    num_threads = []

    for file in files:
        exp = inp(file)
        times.append(exp[0])
        threads.append(exp[1])
        num_threads.append(exp[2])

    plots(times, threads)

    for i in range(len(num_threads)):
        show_table(times[i], num_threads[i])


if __name__ == '__main__':
    main()
```







Time table

	1	2	3	4	5	6	7	8	9	10
1	0.75391	0.78906	0.73047	0.72656	0.69141	0.69531	0.72266	0.79297	0.76562	0.72656
2	0.0625	0.05859	0.05859	0.05859	0.06641	0.0625	0.0625	0.0625	0.07422	0.05078
3	0.04688	0.04688	0.04688	0.04688	0.04297	0.04297	0.03906	0.04297	0.04297	0.04297
4	0.03516	0.03516	0.03516	0.03906	0.03516	0.03125	0.03516	0.03516	0.03516	0.03125
5	0.03906	0.03906	0.03906	0.03906	0.03906	0.04297	0.03906	0.04297	0.03906	0.04297
6	0.03516	0.03516	0.03516	0.03125	0.03516	0.03516	0.03516	0.03516	0.03516	0.03516
7	0.03125	0.03125	0.03125	0.03516	0.03516	0.03516	0.03516	0.03516	0.03516	0.03125
8	0.03125	0.03125	0.02734	0.03125	0.03125	0.03516	0.02734	0.02734	0.02734	0.02734

Time table

	1	2	3	4	5	6	7	8	9	10
1	0.25391	0.26562	0.26953	0.25391	0.26562	0.25391	0.25781	0.26562	0.25781	0.25781
2	0.05859	0.05859	0.06641	0.05469	0.05859	0.05859	0.05469	0.05078	0.05859	0.0625
3	0.05469	0.04688	0.04297	0.04297	0.05078	0.04297	0.04297	0.04297	0.04297	0.03906
4	0.03516	0.03516	0.03906	0.03516	0.03125	0.03906	0.04688	0.04688	0.03516	0.03906
5	0.03516	0.03906	0.03906	0.03906	0.04297	0.03906	0.03516	0.03516	0.03906	0.03906
6	0.03906	0.03125	0.03125	0.03906	0.03516	0.03906	0.03516	0.03516	0.03516	0.03516
7	0.03125	0.03516	0.03125	0.03125	0.02734	0.03516	0.03516	0.03125	0.03125	0.03516
8	0.02734	0.09766	0.03125	0.02734	0.02734	0.03125	0.02734	0.02734	0.02734	0.02734

Time table

	1	2	3	4	5	6	7	8	9	10
1	0.17969	0.17578	0.17578	0.17578	0.17969	0.18359	0.17969	0.17578	0.17188	0.17578
2	0.05859	0.06641	0.0625	0.0625	0.05859	0.05859	0.05859	0.05859	0.0625	0.06641
3	0.04688	0.04297	0.04688	0.03906	0.04297	0.04297	0.05469	0.04297	0.04688	0.04297
4	0.03516	0.03516	0.03906	0.03516	0.03125	0.04688	0.03125	0.03906	0.03516	0.03516
5	0.03906	0.03906	0.03906	0.03906	0.04297	0.03906	0.03516	0.03906	0.03516	0.03906
6	0.03125	0.03516	0.03516	0.03516	0.03516	0.03906	0.03906	0.03516	0.03516	0.03516
7	0.03125	0.03125	0.03125	0.03516	0.03125	0.03125	0.03125	0.03125	0.03125	0.03125
8	0.02734	0.03125	0.03125	0.02734	0.02734	0.03125	0.03125	0.02734	0.03125	0.03125

Time table

	1	2	3	4	5	6	7	8	9	10
1	0.38672	0.34375	0.36328	0.35156	0.36719	0.36719	0.35156	0.37891	0.35156	0.35938
2	0.05859	0.07812	0.05859	0.05859	0.06641	0.06641	0.0625	0.07031	0.0625	0.0625
3	0.04688	0.04297	0.04688	0.04688	0.03906	0.04688	0.03906	0.04688	0.04297	0.04297
4	0.03516	0.03125	0.03125	0.03125	0.03125	0.04297	0.03906	0.03516	0.03516	0.03516
5	0.03906	0.03906	0.04297	0.04297	0.03906	0.04297	0.03906	0.03516	0.03906	0.03516
6	0.03516	0.03906	0.03516	0.03516	0.03906	0.03516	0.03516	0.03516	0.03516	0.03516
7	0.03516	0.03125	0.03125	0.03125	0.03125	0.03125	0.03516	0.03125	0.03125	0.03125
8	0.02734	0.02734	0.03125	0.03125	0.02734	0.03125	0.03516	0.02734	0.02734	0.03125

5. Теоретический анализ

В данном случае ускорение и эффективность алгоритма вычисляется сложнее, чем для предыдущих примеров лабораторных работ. Это происходит из-за невозможности ввести параллельные вычисления для сортировки вставками, которую использует параллельный алгоритм Шелла. Теоретическая оценка для ускорения и эффективности:

$$A_p = (n*log_2(n))/(log_2(n/p)*(n/p)+2n)$$

$$E_p = (n*log_2(n))/(p*(log_2(n/p)*(n/p)+2n))$$

6. Заключение

В этой лабораторной работе я познакомился с новыми принципами работы с OpenMP и приобрел навыки разработки параллельных алгоритмов. Исследование различных входных данных показало, что алгоритм сортировки Шелла чувствителен к исходному массиву