Национальный исследовательский ядерный университет “МИФИ”Лабораторная работа №1: “Введение впараллельные вычисления. Технология OpenMp

Шанкин Данила Б20-505

2022 год

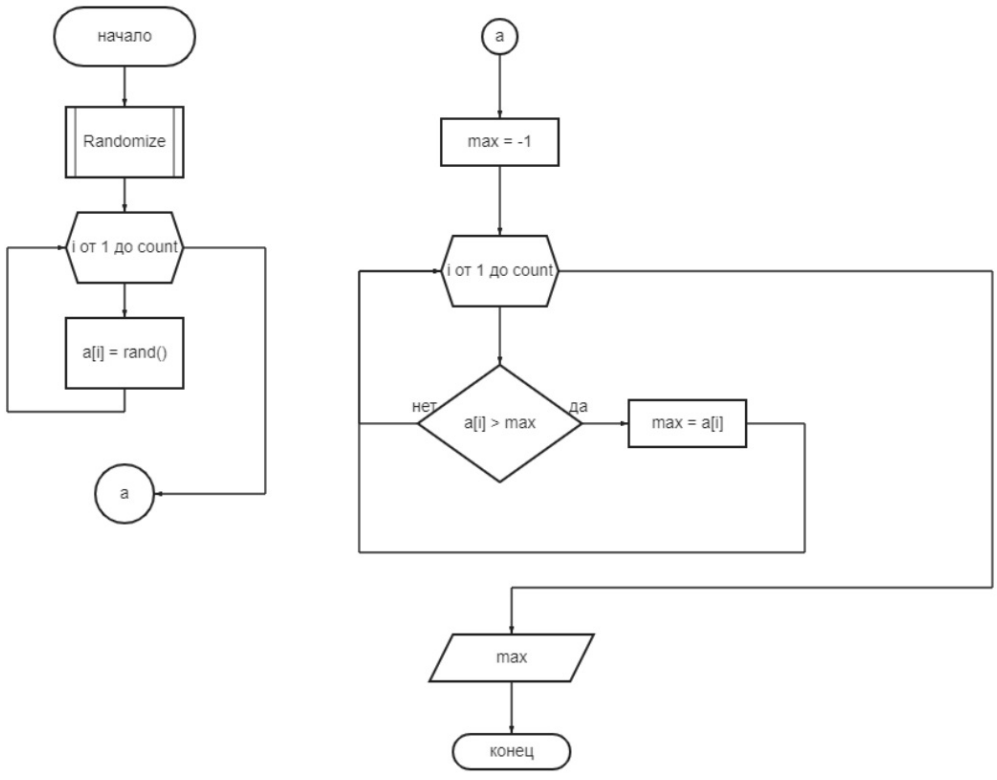
1. Описание рабочей среды

```
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-  `-MMMMMMMMMMM-`  -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:



Алгоритм работает с ассимптотикой $O(N)$

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;
- `reduction(operator:list)` -задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждом потоке; локальные копии инициализируются соответственно типу оператора; над локальными копиями переменных после выполнения всех операторов параллельной области выполняется заданный оператор; порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску.
- `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

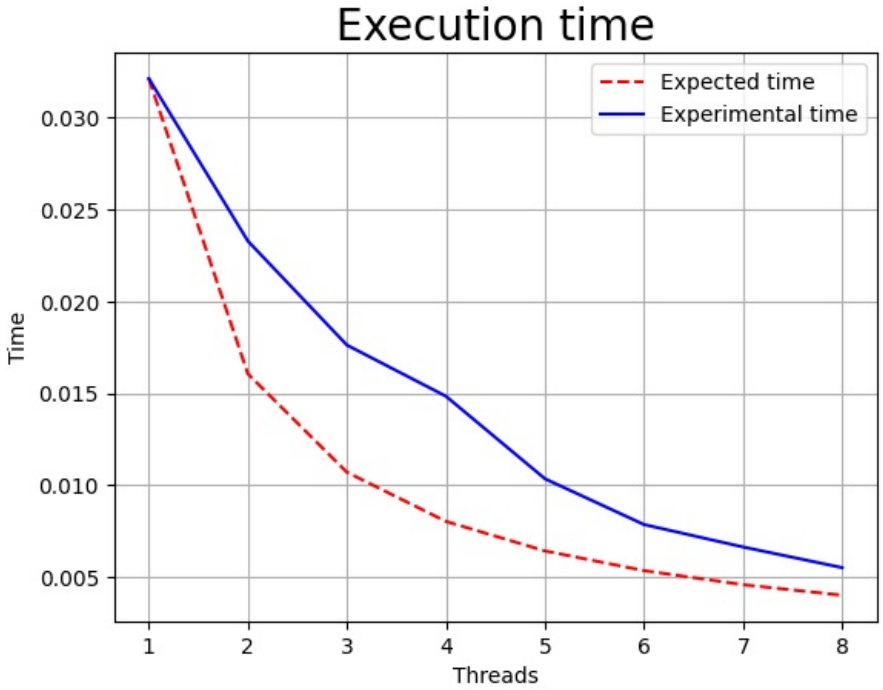
3. Код lab1.c

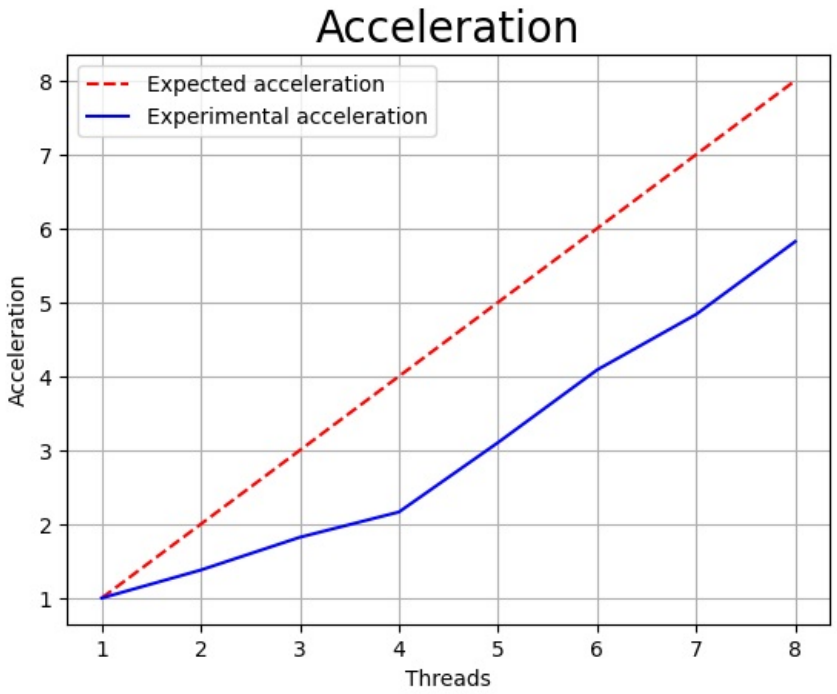
In [ ]:
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void fu(int rand_seed, float* times)
{
    const int count = 10000000;     ///< Number of array elements
    const int threads = 8;          ///< Number of parallel threads to use

    int* array = 0;                 ///< The array we need to find the max in

    /* Initialize the RNG */
    srand(rand_seed);

    /* Generate the random array */
    array = (int*)malloc(count*sizeof(int));
    for(int i=0; i<count; i++) { array[i] = rand(); }

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

                int max = -1;

                float t_start = 0, t_end = 0;
                t_start = omp_get_wtime();

                #pragma omp parallel num_threads(i + 1) shared(array, count) reduction(max: max) default(none)
                {
                        #pragma omp for
                        for(int j=0; j < count; j++)
                        {
                                if(array[j] > max) { max = array[j]; };
                        }
                }
                t_end = omp_get_wtime();
                times[i] = t_end - t_start;
        }

        free(array);
}


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

        int iter = 10;
        int random_seed = 920215; ///< RNG seed
        int threads = 8;
        FILE *file = fopen("data.txt", "w");

        fwrite(&threads, sizeof(int), 1, file);
        fwrite(&iter, sizeof(int), 1, file);
        float* times = 0;

        for (int i = 0; i < 10; i++) {
                times = (float*)calloc(threads, sizeof(float));
                fu(random_seed + i, times);
                fwrite(times, sizeof(float), threads, file);
                free(times);
        }

        fclose(file);
```

```
        return 0;
}
```

4. Код graph.py, графики и таблица

In [20]:
```
from pwn import *
from prettytable.colortable import ColorTable, Themes
import matplotlib.pyplot as plt

def inp():
        data = open('data.txt','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

def plots(times, threads):
        time_average = [sum(k)/len(k) for k in times]
        expected_time = [time_average[0]/(k + 1) for k in range(len(threads))]
        plt.title('Execution time', fontsize=20)
        plt.plot(threads, expected_time, 'r--')
        plt.plot(threads, time_average, 'b')
        plt.xlabel('Threads')
        plt.ylabel('Time')
        plt.grid(1)
        plt.legend(['Expected time', 'Experimental time'])
        plt.show()
        s = [(sum(times[0])/len(times[0]))/(sum(k)/len(k)) for k in times]
        expected_s = [time_average[0]/k for k in expected_time]
        plt.title('Acceleration', fontsize=20)
        plt.plot(threads, expected_s, 'r--')
        plt.plot(threads, s, 'b')
        plt.xlabel('Threads')
        plt.ylabel('Acceleration')
        plt.grid(1)
        plt.legend(['Expected acceleration', 'Experimental acceleration'])
        plt.show()
        e = [s[k]/(k + 1) for k in range(len(s))]
        expected_e = [expected_s[k]/(k + 1) for k in range(len(s))]
        plt.title('Efficiency', fontsize=20)
        plt.plot(threads, expected_e, 'r--')
        plt.plot(threads, e, 'b')
        plt.xlabel('Threads')
        plt.ylabel('Efficiency')
        plt.grid(1)
        plt.legend(['Expected efficiency', 'Experimental efficiency'])
        plt.show()

def table(times, threads):
        table = ColorTable(theme=Themes.DEFAULT)
        table.field_names = ['Thread'] + [i+1 for i in range(len(times[0]))]
        for i in range(len(threads)):
                times[i].insert(0, i+1)
        for i in range(len(times)):
                for j in range(len(times[i])):
                        times[i][j] = round(times[i][j], 5)
        table.add_rows(times)
        print(table)

if __name__ == '__main__':
        exp = inp()
        plots(exp[0], exp[1])
        table(exp[0], exp[1])
```


Execution time
Expected time
Experimental time
Time
0.030
0.025
0.020
0.015
0.010
0.005
1
2
3
4
5
6
7
8
Threads


Acceleration
Expected acceleration
Experimental acceleration
Acceleration
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
Threads


Efficiency
Efficiency
1.0
0.9
0.8
0.7
0.6
Expected efficiency
Experimental efficiency
1
2
3
4
5
6
7
8
Threads

```
+--------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Thread |    1    |    2    |    3    |    4    |    5    |    6    |    7    |    8    |    9    |    10   |
+--------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|   1    | 0.03223 | 0.03223 | 0.03174 | 0.03223 | 0.03223 | 0.03174 | 0.03223 | 0.03223 | 0.03223 | 0.03223 |
|   2    | 0.02344 | 0.02441 | 0.02393 | 0.02344 | 0.01758 | 0.02344 | 0.02637 | 0.02393 |  0.0249 | 0.02148 |
|   3    | 0.01709 | 0.01904 | 0.01855 | 0.01855 | 0.01758 | 0.01807 | 0.01465 | 0.01904 |  0.0166 | 0.01709 |
|   4    | 0.01514 | 0.01611 | 0.01367 |  0.0166 | 0.01416 | 0.01221 | 0.01465 | 0.01465 | 0.01562 | 0.01562 |
|   5    | 0.01367 | 0.01025 | 0.00977 | 0.01025 | 0.01074 | 0.00928 | 0.01025 | 0.00977 | 0.01172 | 0.00781 |
|   6    |  0.0083 | 0.00928 | 0.00732 | 0.00781 | 0.00684 | 0.00684 |  0.0083 | 0.00781 | 0.00732 | 0.00879 |
|   7    | 0.00635 | 0.00684 | 0.00537 | 0.00781 | 0.00684 | 0.00537 | 0.00684 | 0.00635 | 0.00684 | 0.00781 |
|   8    | 0.00488 | 0.00488 | 0.00488 | 0.00537 | 0.00635 | 0.00488 | 0.00635 | 0.00537 | 0.00488 | 0.00732 |
+--------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
```

In [6]:
```
!cp 'Рабочий стол'/parallel_lab1/data.txt ./
```

5. Теоретический анализ

- *Время от числа потоков:* $T_p = \alpha T_1 + (1 - \alpha) T_1 / p$, где α - доля последовательных операций в алгоритме, T_1 - время работы на одном потоке, а p - количество потоков. Однако в нашем случае $(\alpha = 0)$ эту формулу можно упростить до: $T_p = T_1 / p$. Экспериментальный результат был усреднен по 10 итерациям на рандомных входных данных
- *Ускорение от числа потоков* - Ускорением параллельного алгоритма называют отношение времени выполнения лучшего последовательного алгоритмам к времени выполнения параллельного алгоритма: $S = T_1 / T_p$, где T_1 - время работы на одном потоке, а T_p - время работы алгоритма на p потоках. Экспериментальный результат был усреднен по 10 итерациям на рандомных входных данных
- *Эффективность от числа потоков* - Параллельный алгоритм может давать большое ускорение, но использовать для этого множество процессов неэффективно. Для оценки масштабируемости параллельного алгоритма используется понятие эффективности: $E = S / p$, где S - Ускорение от числа потоков, p - количество потоков. Экспериментальный результат был усреднен по 10 итерациям на рандомных входных данных

6. Заключение В этой лабораторной работе я познакомился с основными принципами работы с *OpenMP* и приобрел базовые навыки теоретического и экспериментального анализа высокопроизводительных параллельных алгоритмов, построения параллельных программ.