# ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ “МИФИ”»**

Институт Интеллектуальных Кибернетических Систем Кафедра №42 "Криптология и кибербезопасность"

Дисциплина «Параллельное программирование»

Отчет к лабораторной работе № 1

«Введение в параллельные вычисления. Технология OpenMP»

Выполнил:

студент группы Б22-505 Титов Дмитрий Иванович

Принял:

Куприяшин Михаил Андреевич

Москва 2024 год

**Цель работы**

Приобрести базовые навыки теоретического и экспериментального анализа высокопроизводительных параллельных алгоритмов, построения параллельных программ.

# Рабочая среда

Процессор - amd Ryzen 7 7840HS (2023) 8 ядер и 16 потоков Оперативная память - 16GB DDR5

ОС - Ubuntu 24.04.1 LTS 64-бит

Среда разработки - VS Code, CMake 3.28.3, GCC 13.2.0 Версия OpenMP - 4.5 201511

# Про предложенный алгоритм

## Что делает

Представленная программа осуществляет параллельный поиск максимального элемента в массиве случайно сгенерированных чисел с помощью OpenMP.

## Блок-схема

Рисунок 1. ДРАКОН схема программы

## Оценка времени и сложности

Последовательный алгоритм имеет время работы соответствующей проходу по всему массиву, то есть O(n), где n - количество элементов в массиве.

Параллельный алгоритм в идеальном случае будет иметь ускорение S = Tp / T, где T - время последовательной программы, Tp - время параллельной программы которая работает с p - потоками.

# Экспериментальный анализ последовательной программы

## Время выполнения от количества элементов

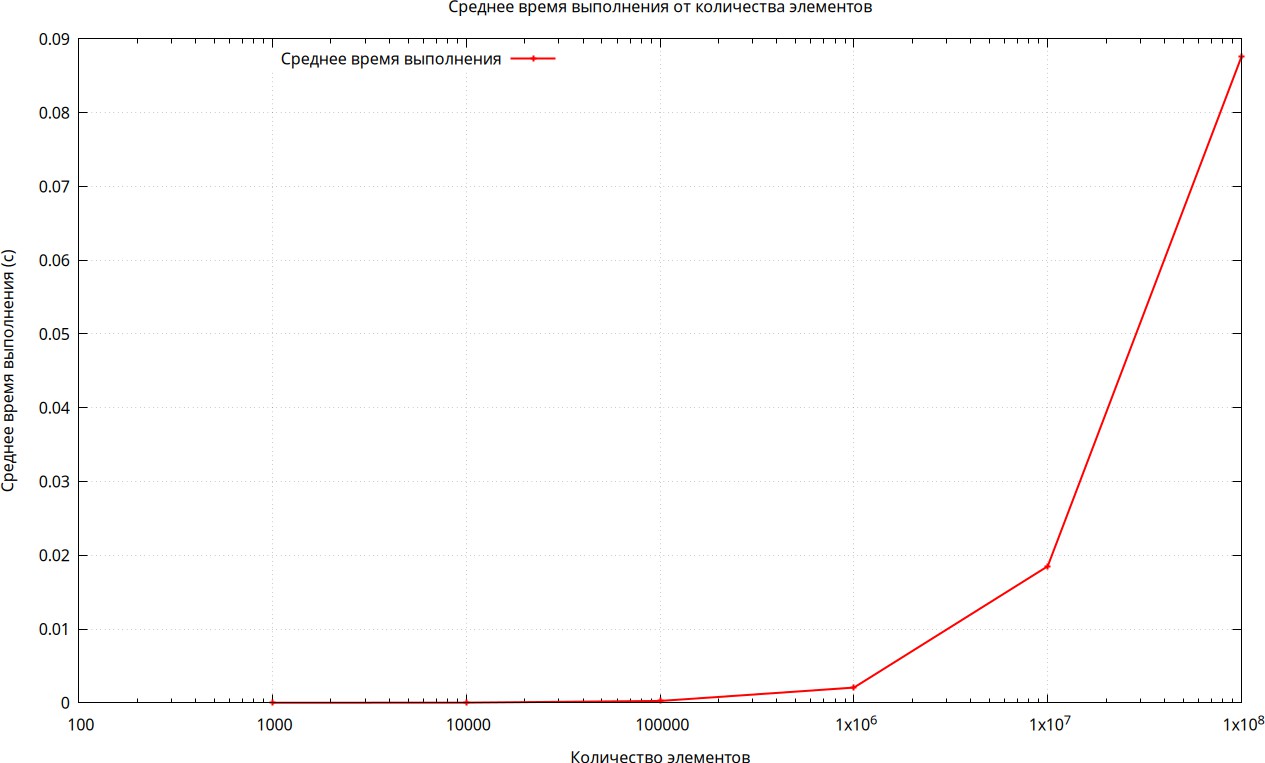
В теории время выполнения должно быть линейным и программа должна произвести n операций за одинаковое время.

График 1. Среднее время от количества элементов

Как видим на графике в рамках малых величин числа элементов можно наблюдать линейную зависимость

## Количество сравнений от количества элементов

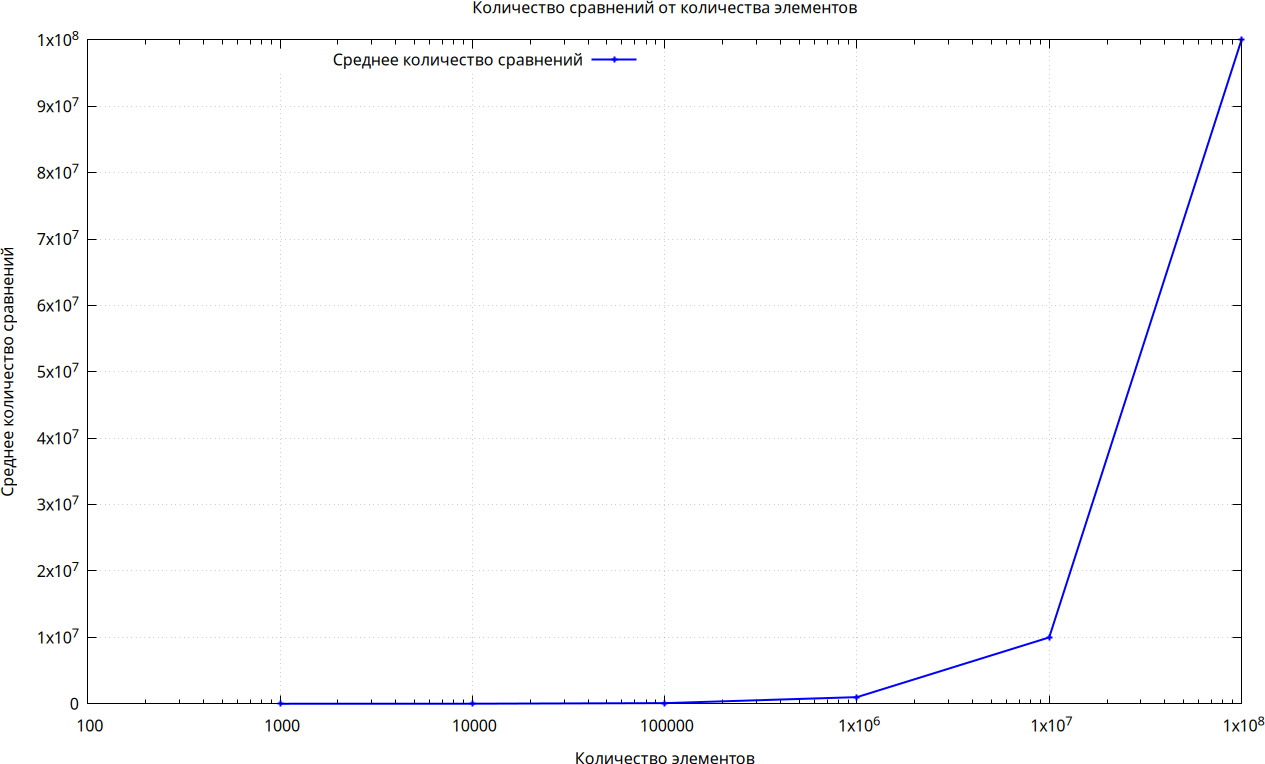
Так как в алгоритме цикл проверки идет от 0 индекса, до n - 1, то теоретически должно быть n сравнений.

График 2. Среднее число сравнений от числа элементов На графике мы видим подтверждении теории.

# Ускорение и эффективность параллельной программы.

## Время параллельной программы от количества процессов

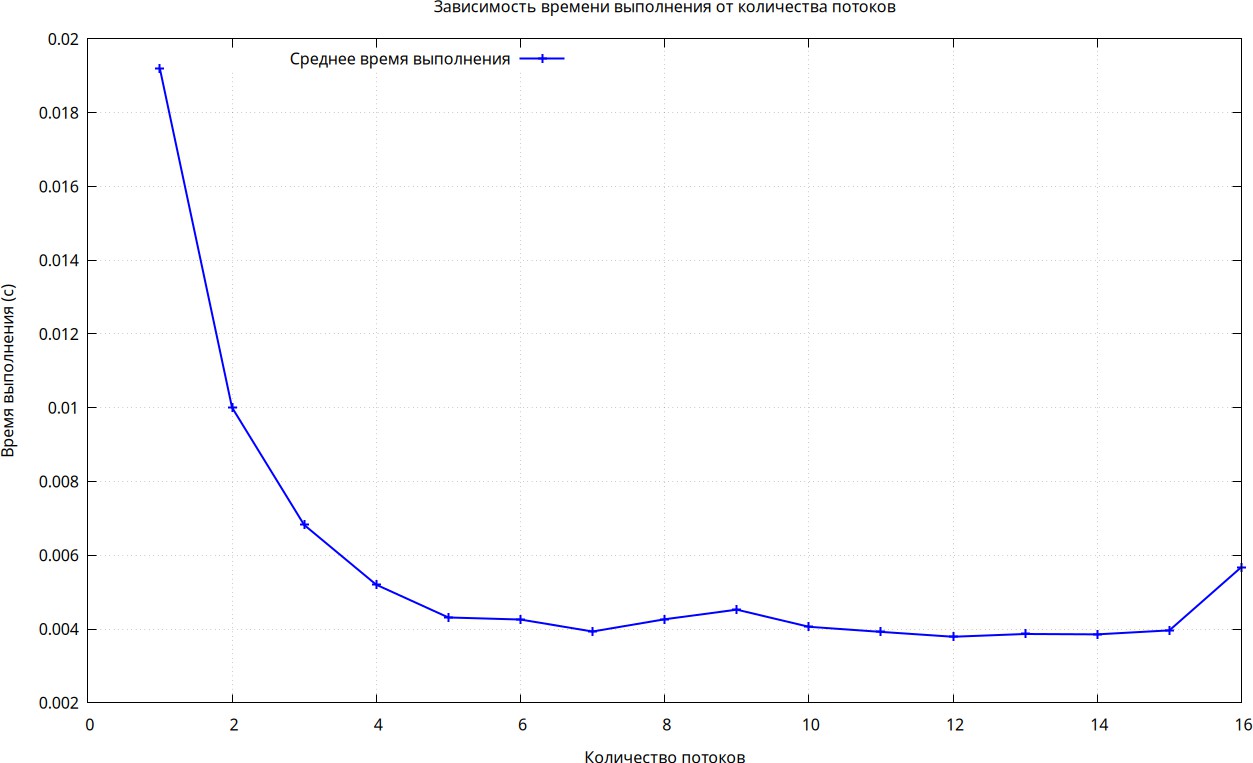
****

График 3. Время параллельной программы от числа процессов

## Ускорение от числа процессов

Как говорилось ранее ускорение можно вычислить по формуле S = Tp / T, где T - время последовательной программы, Tp - время параллельной программы которая работает с p - потоками, на графике мы видим, что идеальное соотношение между ускорением и числом процессов для моей системы находится в диапазоне от 1 до 5 что и соответствует числу процессов.

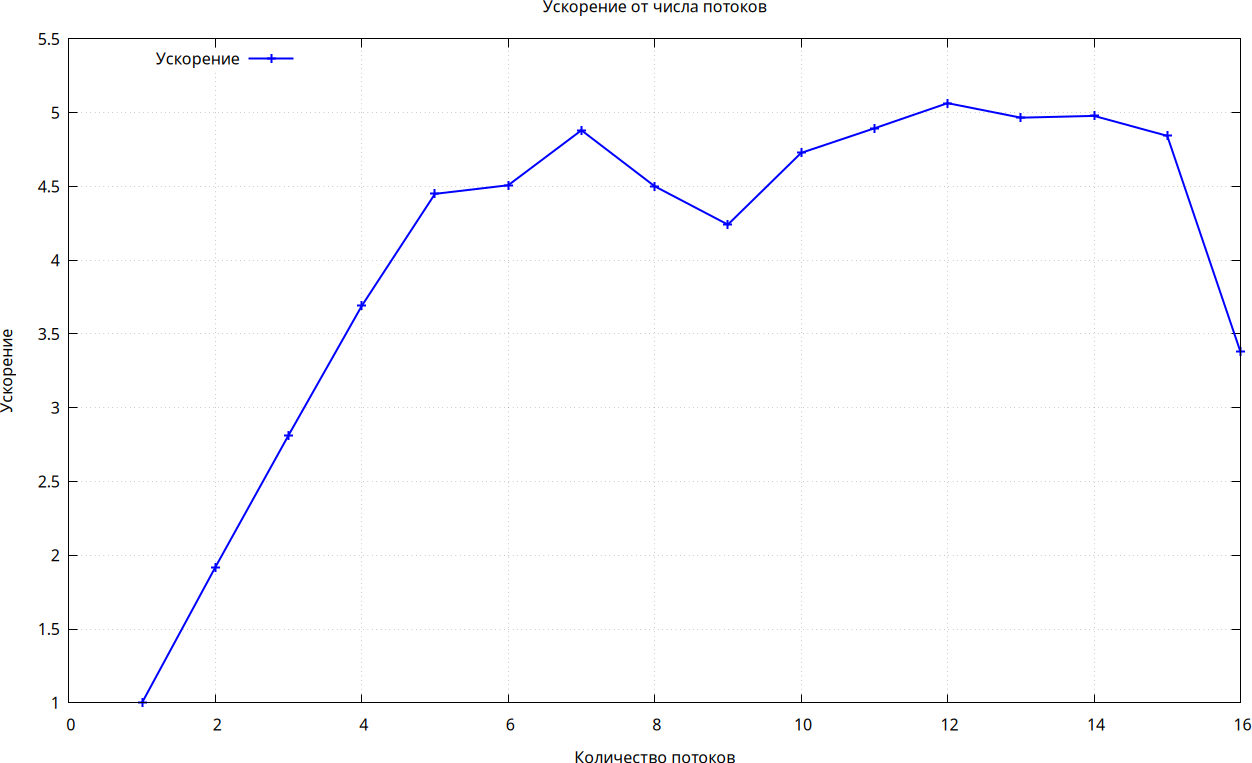


График 4. Ускорение программы от числа процессов

## Эффективность от числа процессов

Эффективность параллельной программы можно получить, если ускорение разделить на число процессов. Таким образом наилучшим показателем можно будет считать величины близкие к единице, так же как и в предыдущем пункте наиболее подходящее положение наблюдается в диапазоне от 1 до 5, так как для этого диапазона эффективность находится в промежутке от 1 до 0.9, что достаточно хорошо.

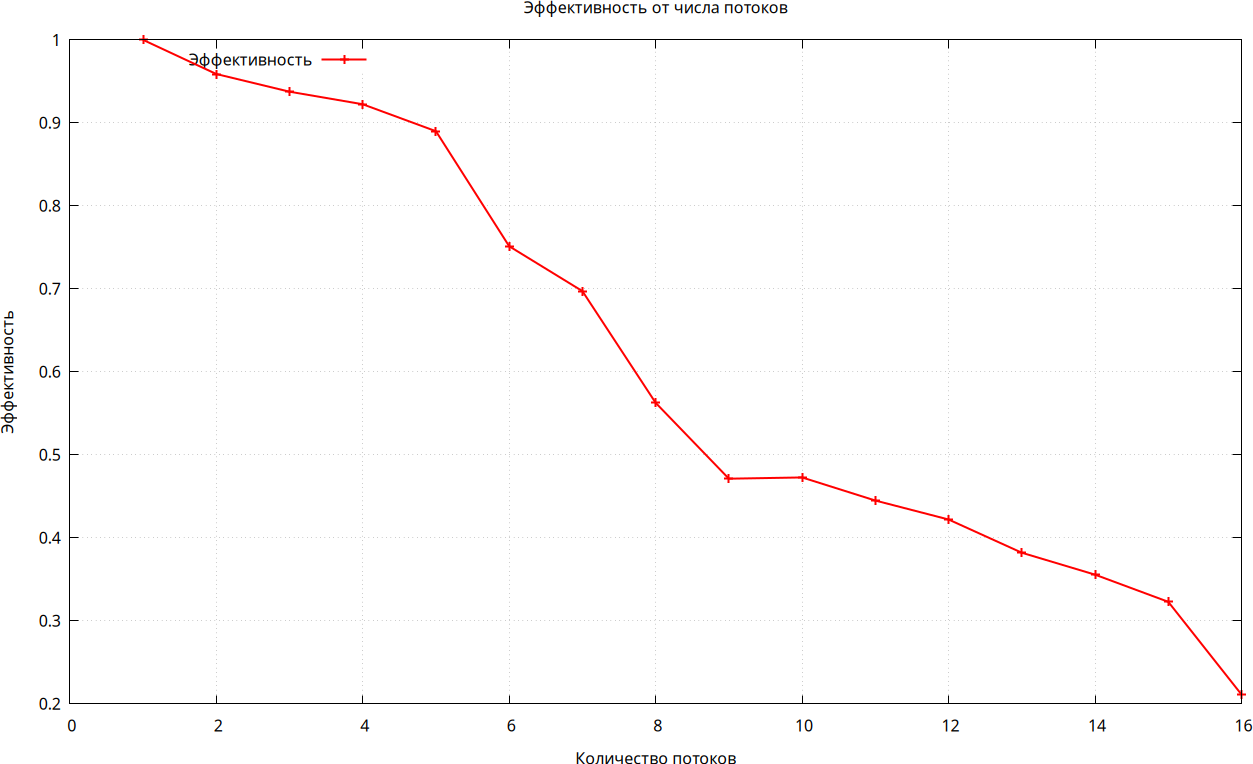


График 5. Эффективность от числа процессов

## Выводы

Как видим на практике ускорение не способно принять линейную зависимость, которая ожидается при теоретически идеальном случае S = p, эффективность также не стала 1 как ожидалось. Как видно после перехода на 7 - 9 процессах наблюдается резкий спад производительности из-за увеличивающихся накладных, таких как проблемы с работой кэш-памяти, потребности управления и синхронизации процессов, однако ускорение старается быть на неком достаточном диапазоне, эффективность же в свою очередь из-за недостатка ускорения стремительно падает. Для моей системы идеальными можно назвать вычисления одновременно на 4 - 5 процессах.

# Заключение

Целью данной работы было приобретение навыков теоретического и экспериментального анализа высокопроизводительных параллельных алгоритмов, а также построения параллельных программ с использованием стандарта OpenMP.

В ходе работы были выполнены следующие задачи:

* + 1. Настроена рабочая среда и проверена поддержка стандарта OpenMP компилятором.
    2. Изучен алгоритм поиска максимального элемента в массиве и оценена его временная сложность. Для последовательного алгоритма временная сложность оказалась O(N), что подтвердилось экспериментальными результатами.
    3. Разработана и протестирована параллельная программа для поиска максимального элемента с использованием OpenMP. Программа успешно выполнила задачу, подтверждая корректность параллелизации.
    4. Экспериментально определены временная сложность и количество операций сравнения для последовательного алгоритма. Результаты показали линейную зависимость времени выполнения от размера массива, что согласуется с теоретической сложностью.
    5. Измерено ускорение и эффективность параллельного алгоритма для различных количеств потоков. Было отмечено, что ускорение увеличивалось с ростом числа потоков до определённого предела, после чего начало снижаться. Это связано с оверхедами, связанными с управлением потоками и снижением эффективности использования кэш-памяти.
    6. Результаты экспериментов показали, что при использовании до 5 потоков наблюдается значительное ускорение, однако эффективность начинает снижаться при увеличении числа потоков. Это можно объяснить увеличением накладных расходов и несовершенством распределения задач между потоками. При использовании 16 потоков эффективность снизилась до 21%, что подтверждает теоретический предел ускорения, связанный с законом Амдала.
    7. Оформлены графики, показывающие зависимость ускорения и эффективности параллельных вычислений от числа потоков, а также зависимость времени выполнения и числа сравнений от размера массива для последовательного алгоритма.

Таким образом, цель работы достигнута — были изучены принципы построения параллельных программ, измерены ускорение и эффективность параллельных вычислений, и сопоставлены теоретические и экспериментальные данные.

# Приложение

## Последовательная программа

#include <stdio.h> #include <stdlib.h> #include <omp.h>

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

const long long sizes[] = {1000, 10000, 100000, 1000000, 10000000, 100000000}; // Array sizes

const int experiments = 10; // Number of experiments per size const int random\_seed = 920215; // RNG seed

long long comparisons; // Comparison counter

long long total\_comparisons; // Sum of comparisons for averaging double total\_time; // Sum of times for averaging

int\* array = 0; // Array for random numbers

int max = -1; // Maximal element double start\_time, end\_time; // Timing variables

/\* Initialize the RNG \*/ srand(random\_seed);

// Открываем файл для записи данных для gnuplot FILE \*fp = fopen("experiment\_results.csv", "w"); fprintf(fp, "Size,AvgComparisons,AvgTime\n");

printf("Size\tExperiment\tTime (s)\tComparisons\tMax Value\n");

/\* Loop over different array sizes \*/

for(int s = 0; s < sizeof(sizes)/sizeof(sizes[0]); s++) { long long size = sizes[s];

total\_comparisons = 0; // Initialize the total comparisons for averaging total\_time = 0.0; // Initialize the total time for averaging

for (int exp = 0; exp < experiments; exp++) {

/\* Allocate and fill the array with random values \*/ array = (int\*)malloc(size \* sizeof(int));

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

comparisons = 0; // Reset comparisons count for each experiment max = -1; // Reset max value before each experiment

/\* Start timing \*/

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

/\* Sequential computation \*/ for(int i = 0; i < size; i++) {

comparisons++;

if(array[i] > max) { max = array[i]; }

}

/\* End timing \*/

end\_time = omp\_get\_wtime();

/\* Add the current experiment's time and comparisons to totals \*/ total\_time += (end\_time - start\_time);

total\_comparisons += comparisons;

max);

/\* Output the results for the current experiment \*/

printf("%lld\t%d\t%f\t%lld\t%d\n", size, exp + 1, end\_time - start\_time, comparisons,

free(array); // Free the allocated array

}

/\* Compute and output the average time and comparisons for the current size \*/ printf("Average for size %lld: Avg Time = %f s, Avg Comparisons = %lld\n",

size, total\_time / experiments, total\_comparisons / experiments);

// Записываем данные в CSV для Gnuplot

fprintf(fp, "%lld %lld %f\n", size, total\_comparisons / experiments, total\_time / experiments);

}

fclose(fp); // Закрываем файл return 0;

}

Листинг 1. Последовательная программа lab1seq.с cmake\_minimum\_required(VERSION 3.10)

project(lab1seq C)

# Установка стандарта языка C set(CMAKE\_C\_STANDARD 99)

# Включение поддержки OpenMP find\_package(OpenMP REQUIRED)

# Добавление исполняемого файла add\_executable(lab1seq lab1seq.c)

# Линковка с библиотекой OpenMP if(OpenMP\_C\_FOUND)

target\_link\_libraries(lab1seq PUBLIC OpenMP::OpenMP\_C) endif()

Листинг 2. CMakeLists для последовательной программы

# Первый график - количество сравнений от количества элементов set terminal pngcairo size 1280, 800

set output 'comparisons\_of\_size.png'

set title "Количество сравнений от количества элементов" set xlabel "Количество элементов"

set ylabel "Среднее количество сравнений" set grid

set logscale x set key left top

plot 'experiment\_results.csv' using 1:2 with linespoints title "Среднее количество сравнений" lw 2 lc rgb "blue"

# Второй график - среднее время выполнения от количества элементов set terminal pngcairo size 1280, 800

set output 'time\_of\_size.png'

set title "Среднее время выполнения от количества элементов" set xlabel "Количество элементов"

set ylabel "Среднее время выполнения (с)" set grid

set logscale x set key left top

plot 'experiment\_results.csv' using 1:3 with linespoints title "Среднее время выполнения" lw 2 lc rgb "red"

Листинг 3. Gnuplot-script для последовательной программы

## Параллельная программа

#include <stdio.h> #include <stdlib.h> #include <omp.h>

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

{

const int count = 10000000; // Number of array elements

const int threads\_max = 16; // Maximum number of parallel threads to use const int experiments = 10; // Number of experiments per thread count

int\*\* arrays; // Array of pointers for random number arrays (now dynamic) int max = -1; // Maximal element

double start\_time, end\_time; // Timing variables double total\_time; // Sum of times for averaging

double time\_seq; // Time for sequential equal for 1 thread double avg\_time; // Average time for thread

double speedup, efficiency; // Speedup and efficiency variables

/\* Define 10 fixed random seeds \*/

const int seeds[10] = {123456, 789012, 345678, 901234, 567890,

112233, 445566, 778899, 990011, 223344};

/\* Open file to write results for Gnuplot \*/

FILE \*fp = fopen("experiment\_results.csv", "w"); if (!fp) {

perror("Unable to open file"); return 1;

}

/\* Allocate memory for the array of pointers dynamically \*/ arrays = (int\*\*)malloc(experiments \* sizeof(int\*));

/\* Generate 10 different arrays with fixed random seeds \*/ for (int exp = 0; exp < experiments; exp++) {

/\* Allocate memory for each array dynamically \*/ arrays[exp] = (int\*)malloc(count \* sizeof(int));

/\* Set a fixed seed for each array \*/ srand(seeds[exp]);

/\* Fill the array with random values \*/ for(int i = 0; i < count; i++) {

arrays[exp][i] = rand();

}

}

printf("Threads\tExperiment\tTime (s)\tMax Value\n");

/\* Perform experiments for different number of threads \*/ for(int threads = 1; threads <= threads\_max; threads++) {

total\_time = 0.0; // Reset total time for each thread count for (int exp = 0; exp < experiments; exp++) {

max = -1; // Reset max value before each experiment

/\* Start timing \*/

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

/\* Parallel computation using pre-generated array \*/

#pragma omp parallel num\_threads(threads) shared(arrays, count, exp) reduction(max: max) default(none)

{

#pragma omp for

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

if(arrays[exp][i] > max) { max = arrays[exp][i]; }

}

}

/\* End timing \*/

end\_time = omp\_get\_wtime();

/\* Add the current experiment's time \*/

total\_time += (end\_time - start\_time);

/\* Output the results \*/

printf("%d\t%d\t%f\t%d\n", threads, exp + 1, end\_time - start\_time, max);

}

/\* Compute and output the average time for the current threads \*/

printf("Average for threads %d: Avg Time = %f\n", threads, total\_time / experiments); avg\_time = total\_time / experiments;

if (threads == 1) { time\_seq = avg\_time;

}

speedup = time\_seq / avg\_time; efficiency = speedup /threads;

/\* Write results to CSV file \*/

fprintf(fp, "%d %f %f %f\n", threads, avg\_time, speedup, efficiency);

}

/\* Free all allocated arrays after all experiments \*/ for (int exp = 0; exp < experiments; exp++) {

free(arrays[exp]);

}

/\* Free the array of pointers \*/ free(arrays);

/\* Close the file \*/ fclose(fp);

return 0;

}

Листинг 4. Параллельная программа lab1parall.с

cmake\_minimum\_required(VERSION 3.10) project(lab1parall C)

# Установка стандарта языка C set(CMAKE\_C\_STANDARD 99)

# Включение поддержки OpenMP find\_package(OpenMP REQUIRED)

# Добавление исполняемого файла add\_executable(lab1parall lab1parall.c)

# Линковка с библиотекой OpenMP if(OpenMP\_C\_FOUND)

target\_link\_libraries(lab1parall PUBLIC OpenMP::OpenMP\_C) endif()

Листинг 5. CMakeLists для параллельной программы

# Настройки графика

set terminal pngcairo size 1280, 800 set output 'parallel\_performance.png'

set title "Зависимость времени выполнения от количества потоков" set xlabel "Количество потоков"

set ylabel "Время выполнения (с)" set grid

set datafile separator whitespace set key left top

# Настройка стилей точек и линий set style data linespoints

set pointsize 1.5

# Загрузка данных из файла и построение графика

plot 'experiment\_results.csv' using 1:2 with linespoints title "Среднее время выполнения" lw 2 lc rgb "blue"

# График: Ускорение от числа потоков

set output 'speedup\_vs\_threads.png' # Имя выходного файла set title "Ускорение от числа потоков"

set xlabel "Количество потоков" set ylabel "Ускорение"

plot 'experiment\_results.csv' using 1:3 with linespoints title "Ускорение" lw 2 lc rgb "blue"

# График: Эффективность от числа потоков

set output 'efficiency\_vs\_threads.png' # Имя выходного файла set title "Эффективность от числа потоков"

set xlabel "Количество потоков" set ylabel "Эффективность"

plot 'experiment\_results.csv' using 1:4 with linespoints title "Эффективность" lw 2 lc rgb "red"

Листинг 6. Gnuplot-script для параллельной программы

* 1. **Таблицы для графиков**

|  |  |  |
| --- | --- | --- |
| **Size** | **Average Comparisons** | **Average Time , sec** |
| **1000** | **1000** | **0.000002** |
| **10000** | **10000** | **0.000018** |
| **100000** | **100000** | **0.000248** |
| **1000000** | **1000000** | **0.002060** |
| **10000000** | **10000000** | **0.018508** |
| **100000000** | **100000000** | **0.087588** |

**Таблица 1. Среднее время и число сравнений от числа элементов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Threads** | **Average Time , sec** | **Speedup** | **Efficiency** |
| **1** | **0.019191** | **1.000000** | **1.000000** |
| **2** | **0.010007** | **1.917722** | **0.958861** |
| **3** | **0.006823** | **2.812635** | **0.937545** |
| **4** | **0.005201** | **3.689626** | **0.922406** |
| **5** | **0.004313** | **4.449529** | **0.889906** |
| **6** | **0.004257** | **4.507857** | **0.751309** |
| **7** | **0.003932** | **4.880560** | **0.697223** |
| **8** | **0.004264** | **4.501112** | **0.562639** |
| **9** | **0.004525** | **4.241050** | **0.471228** |
| **10** | **0.004060** | **4.727208** | **0.472721** |
| **11** | **0.003922** | **4.893655** | **0.444878** |
| **12** | **0.003790** | **5.064332** | **0.422028** |
| **13** | **0.003865** | **4.965916** | **0.381994** |
| **14** | **0.003855** | **4.977967** | **0.355569** |
| **15** | **0.003963** | **4.842520** | **0.322835** |
| **16** | **0.005681** | **3.378208** | **0.211138** |