Лабораторная работа №4

"Технология *OpenMP*. Особенности настройки"

Выполнил студент группы Б20-505
Сорочан Илья

Московский Инженерно-Физический Интститут
Москва 2023

1 Рабочая среда

Технические характеристики:
CPU: *6-core AMD Ryzen 5 4500U*
Kernel: *5.15.85-1-MANJARO x86_64*
Mem: *7303.9 MiB*

Используется:
Компилятор: *GCC 12.2.0*
OpenMP: *4.5*

2 Работа с *OpenMP*

2.1 Версия и дата принятия

Макрос *_OPENMP* является целочисленным числом и показывает дату принятия *OpenMP* в формате *yyyymm*, где *yyyy* - год принятия, а *mm* - месяц. Соответствие даты версии *OpenMP* можно найти в интернете.

2.2 *OMP_DYNAMIC*

Переменная окружения *OMP_DYNAMIC* отвечает за динамический выбор числа потоков. Например если она имеет значение *true*, то *OpenMP* автоматически выбирает число потоков для *parallel* участков. Если же *false*,

2.3 wtick

Функция *omp_get_wtick()* возвращает количество секунд, прошедшее между тиками таймера из *omp_get_wtime()*. *wtime* возвращает количество прошедших секунд.

Стоит отметить, что функция *clock()*, предоставленная в стандартной библиотеке *C* считает время использования *CPU*. Тогда как *omp_wtime()* считает просто прошедшее время.

2.4 Вложенность

Функция *omp_get_nested()* возвращает флаг, указывающий на то включен ли вложенный параллелизм. Если он истинен, то количество вложенных конструкций ограниченно числом, которое можно получить, вызвав *omp_get_max_active_levels()*.

2.5 schedule

Опция *schedule* директивы *for* задаёт тип распределения нагрузки и размер чанков. Тип определяет как циклы делятся на подмножества итераций – чанки:

- *static* – все подмножества распределяются между потоками один раз, в самом начале;
- *dynamic* – каждый из процессов получает чанк, по его выполнении он запрашивает новый. Так продолжается пока чанки не закончатся;
- *guided* – аналогично *dynamic*, однако по мере выполнения чанков их размер уменьшается;
- *auto* – компилятор выбирает на свое усмотрение;
- *runtime* – выбор производится непосредственно перед выполнением программы через переменную окружения *OMP_SCHEDULE*.

2.6 Пример использования *omp_lock*

Замки необходимы для обеспечения выполнении промежутка кода только одним потоком. Например чтение из файла.

```
omp_lock_t write_lock;

omp_init_lock(&write_lock);

#pragma omp parallel for
for ( i = 0; i < x; i++ )
```

```
{
  // do something important
  omp_set_lock(&write_lock);
  // do something important but only one thread access at a time
  omp_unset_lock(&write_lock);
  // do another important task
}

omp_destroy_lock(&write_lock);
```

2.7 Разработанный код

Для иллюстрации директив *OpenMP*, затронутых в данном разделе была разработана следующая программа:

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

#if _OPENMP == 200505
#define _OPENMP_VERSION "2.5"
#elif _OPENMP == 200805
#define _OPENMP_VERSION "3.0"
#elif _OPENMP == 201107
#define _OPENMP_VERSION "3.1"
#elif _OPENMP == 201307
#define _OPENMP_VERSION "4.0"
#elif _OPENMP == 201511
#define _OPENMP_VERSION "4.5"
#elif _OPENMP == 201811
#define _OPENMP_VERSION "5.0"
#elif _OPENMP == 202011
#define _OPENMP_VERSION "5.1"
#else
#define _OPENMP_VERSION "unknown"
#endif

int main(int argc, char** argv) {
    printf("OpenMP Version: %s\nRelease date: %d\n", _OPENMP_VERSION, _OPENMP);

    printf("\nAvaliable processors: %d\nAvaliable threads: %d\n", omp_get_num_procs(), omp_get_max_threads());

    if (omp_get_dynamic())
        puts("\nDynamic is on");
    else
        puts("\nDynamic is off");

    printf("\nOpenMP wtick: %fs\n", omp_get_wtick());

    if (omp_get_nested())
        printf("\nNested parallelism up to %d\n", omp_get_max_active_levels());
    else
        puts("Nested parallelism is off");


    omp_sched_t sched;
    int chunk_size;
    omp_get_schedule(&sched, &chunk_size);
    char *s;
    switch (sched) {
        case omp_sched_static: s = "static"; break;
        case omp_sched_dynamic: s = "dynamic"; break;
        case omp_sched_guided: s = "guided"; break;
        case omp_sched_auto: s = "auto"; break;
    }
    printf("\nOpenMP schedule: %s\nChunk size: %d\n", s, chunk_size);

    return 0;
}
```

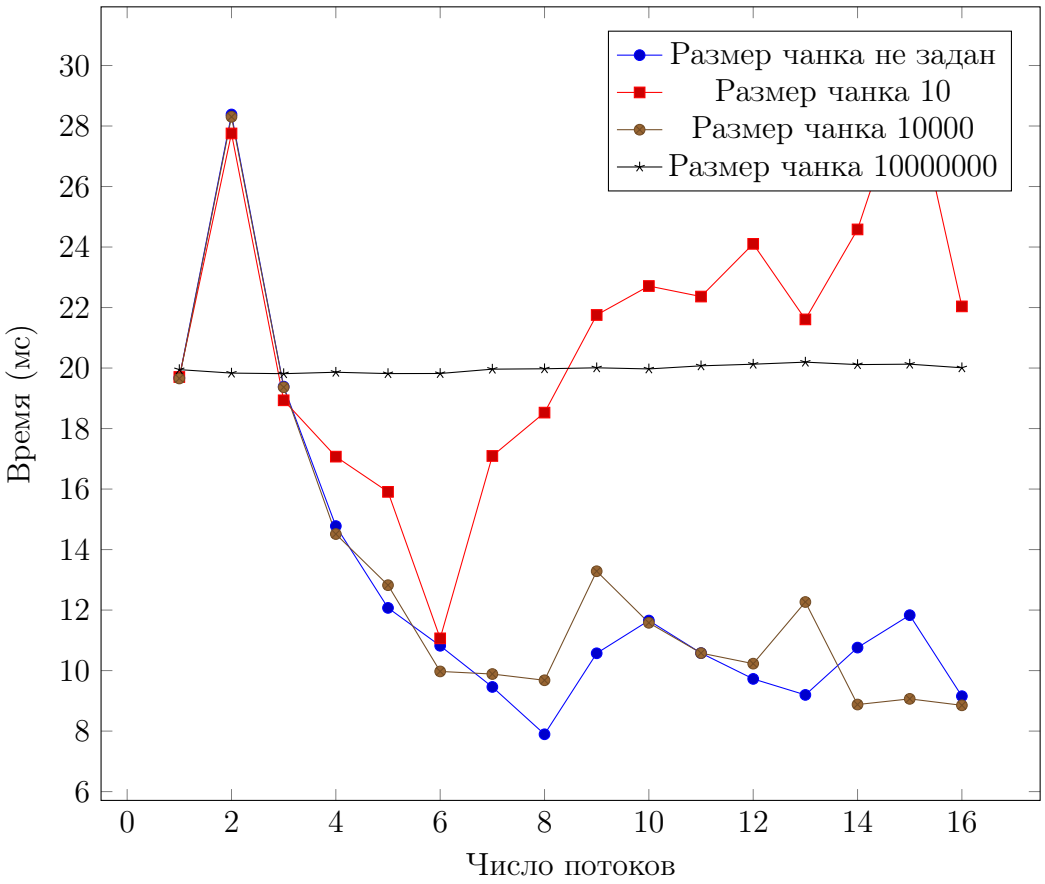
3 Применение *schedule*

За основу был взят код из второй лабораторной. Число запусков на поток было уменьшено до 10. Использовалось *schedule(runtime)*, которое затем менялось в различных тестах.

Использованные исходные коды можно найти в приложении.

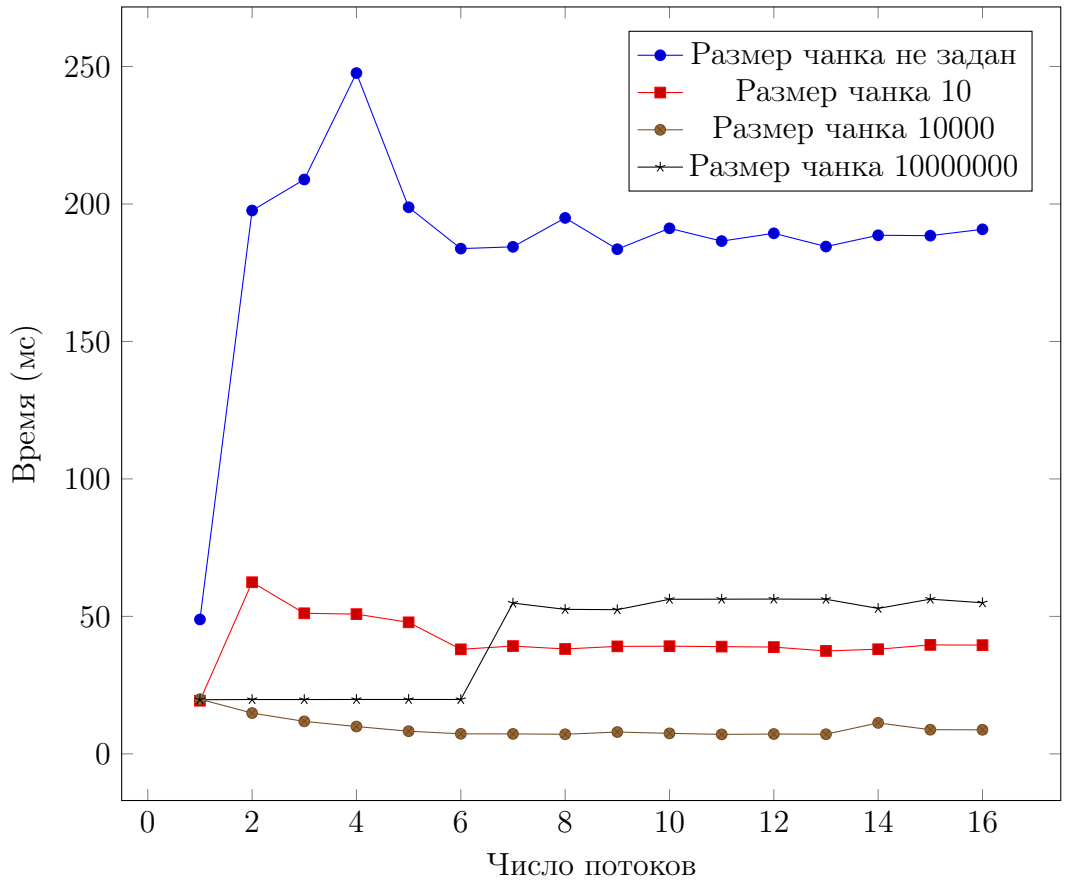
3.1 Графики

Для *schedule* = *static*:



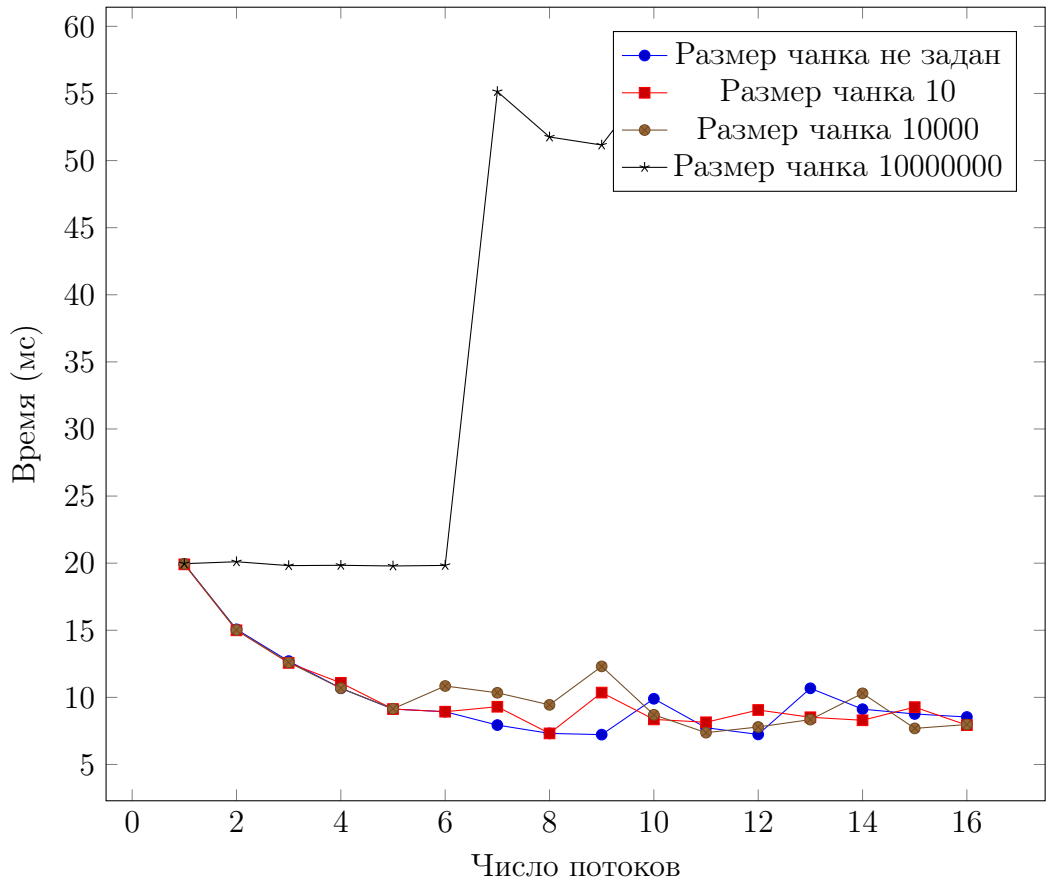
Замечу, что компилятор сам выбирает оптимальное значение чанка. Его производительность является одной из лучших.

Рассмотрим *schedule* = *dynamic*:



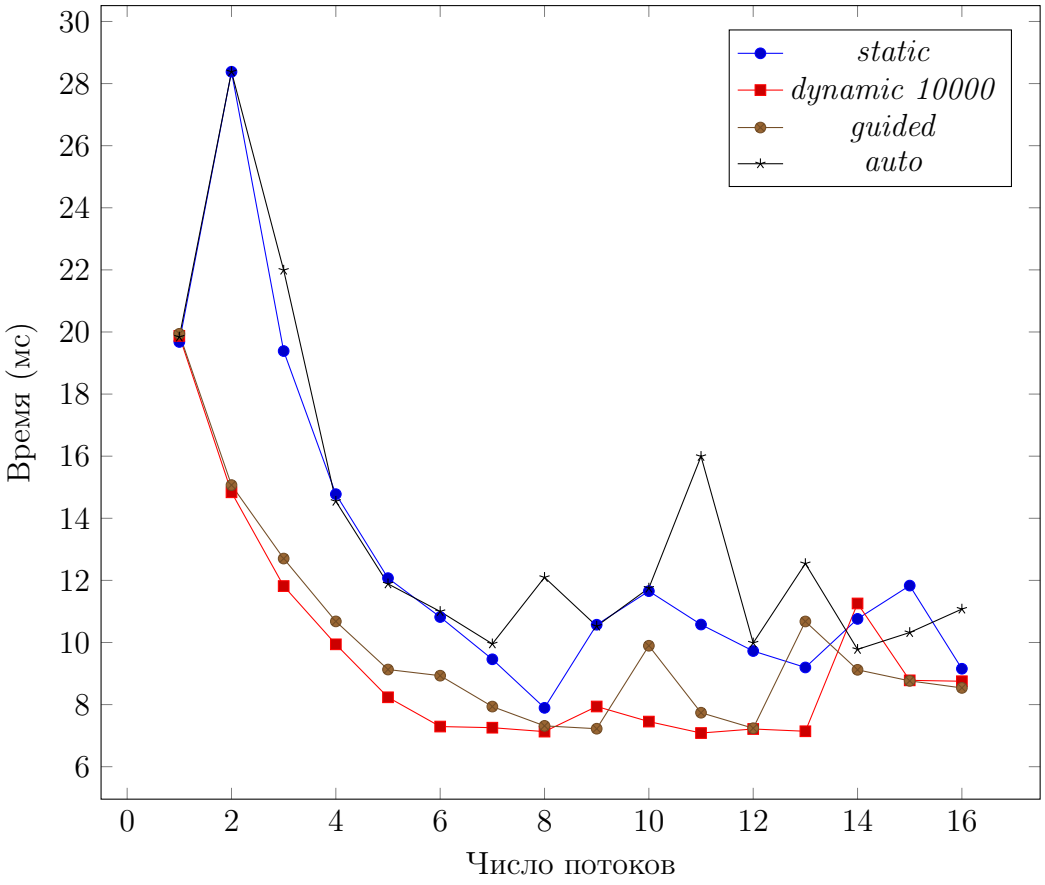
На удивление здесь стандартное значение показывает себя худшим образом. По умолчанию в режиме *dynamic* размер чанка равен 1.

Рассмотрим *schedule* = *guided*:



Здесь тенденция наблюдается положительная, за исключением чанков максимального размера. Разница между остальными размерами чанков не превышает нескольких миллисекунд.

Наконец, сравним лучшие из вышеприведенных результатов в автоматическим распределением *schedule = auto*. На графике указан режим распределения и размер чанка (если он был задан вручную):



4 Заключение

В данной работе было исследовано ускорение, получаемое при использовании различных режимов распределения нагрузки (*schedule*) в задании о поиске элемента. Была усовершенствована предоставленная программа и собранны данные. Так же был написан скрипт, осуществляющий сбор информации для разных параметров. Оформлен отчет.

В ходе работы было выяснено, что в данной задаче разница в производительности небольшая. Лучше всего себя показали режимы *dynamic 10000* и *guided*.

Приложение А

Использованные программные коды

Для измерения времени исполнения алгоритма использовался следующий код (выводит *csv* в стандартный вывод):

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

const int N = 10000000;
const int MAX_THREADS = 16;
const int RUNS_PER_THREAD = 10;

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

// run algo and return time elapsed
double run(const int threads, int *array, const int size, const int target) {
    double start = omp_get_wtime();;
    int index = -1;
    #pragma omp parallel num_threads(threads) shared(array, size, index, target) default(none)
    {
        #pragma omp for schedule(runtime)
        for(int i = 0; i < size; ++i) {
            if(array[i] == target) {
                #pragma omp critical
                index = array[i];
                #pragma omp cancel for
            };
        }
    }
    double end = omp_get_wtime();;
    return (end - start) * 1000;
}

int main(int argc, char **argv) {
    // set constant seeds
    int seed[MAX_THREADS];
    for (int i = 0; i < MAX_THREADS; ++i)
        seed[i] = rand();


    int *array = (int *)malloc(N * sizeof(int));

    puts("Threads,Worst (ms),Best (ms),Avg (ms)");

    for (int threads = 1; threads < MAX_THREADS + 1; ++threads) {
        double sum = 0, max_time = -1, min_time = 100000;
        for (int i = 0; i < RUNS_PER_THREAD; ++i) {
            // gen array with special seed
            srand(seed[i]);
            randArr(array, N);

            // calc value
            double time = run(threads, array, N, 16);
            if (time > max_time)
                max_time = time;
            if (time < min_time)
                min_time = time;
            sum += time;
        }

        printf("%d,%.3f,%.3f,%.3f\n", threads, max_time, min_time, sum / RUNS_PER_THREAD);
    }

    free(array);

    return 0;
}
```

Для вычисления сбора данных с разными параметрами был использован:

```
#!/bin/bash
# please run this script from lab4/code directory

# compile
gcc main.c -fopenmp -o main

# static
echo running static
OMP_SCHEDULE=static ./main > ../data/static.csv
echo 10
```

```
OMP_SCHEDULE=static,10 ./main > ../data/static_10.csv
echo 10000
OMP_SCHEDULE=static,10000 ./main > ../data/static_10000.csv
echo 10000000
OMP_SCHEDULE=static,10000000 ./main > ../data/static_10000000.csv

# dynamic
echo running dynamic
OMP_SCHEDULE=dynamic ./main > ../data/dynamic.csv
echo 10
OMP_SCHEDULE=dynamic,10 ./main > ../data/dynamic_10.csv
echo 10000
OMP_SCHEDULE=dynamic,10000 ./main > ../data/dynamic_10000.csv
echo 10000000
OMP_SCHEDULE=dynamic,10000000 ./main > ../data/dynamic_10000000.csv

# guided
echo running guided
OMP_SCHEDULE=guided ./main > ../data/guided.csv
echo 10
OMP_SCHEDULE=guided,10 ./main > ../data/guided_10.csv
echo 10000
OMP_SCHEDULE=guided,10000 ./main > ../data/guided_10000.csv
echo 10000000
OMP_SCHEDULE=guided,10000000 ./main > ../data/guided_10000000.csv

# auto
echo running auto
OMP_SCHEDULE=auto ./main > ../data/auto.csv
```