МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
КАФЕДРА №42 «КРИПТОЛОГИЯ И КИБЕРБЕЗОПАСНОСТЬ»

ОТЧЁТ

по дисциплине «Параллельное программирование»
Лабораторная работа №4
«Технология OpenMP. Особенности настройки»

Группа Б21-525

Студент Г.О. Шулындин

Преподаватель М.А. Куприяшин

Москва 2023

Оглавление

1. Описание рабочей среды

- Модель процессора: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
- Число ядер: 8
- Архитектура: x86-64
- ОС: Linux, дистрибутив Ubuntu v22.04
- RAM объем: 2x4096 MB
- RAM тип: DDR4
- Используемая среда разработки: Visual Studio Code
- Компилятор: gcc v11.4.0
- Поддерживаемая версия OpenMP: 201511

2. Проверка настроек OpenMP

Используемая версия

`_OPENMP` - переменная предпроцессора содержит дату принятия используемого стандарта OpenMP в формате yyyymm, где yyyy - год, mm - месяц. По дате принятия стандарта можно определить версию OpenMP. В используемой конфигурации `_OPENMP` = 201511 и, соответственно, поддерживаемая версия - OpenMP 4.5.

Число процессоров и потоков

`omp_get_num_procs()` - возвращает число доступных процессоров;
`omp_get_max_threads()` - возвращает верхнюю границу количества потоков, которые могли бы быть использованы для формирования новой команды, если бы после выполнения этой процедуры была обнаружена параллельная конструкция без условия `num_threads`.

В используемой конфигурации данные параметры следующие:

- `omp_get_num_procs()` = 8
- `omp_get_max_threads()` = 8

Динамическое выделение числа потоков

`omp_get_dynamic()` - возвращает значение dyn-var, которое определяет, включена или отключена динамическая регулировка количества потоков.
В текущей конфигурации `omp_get_dynamic()` = 0.

Параметры таймера

`omp_get_wtick()` - функция возвращает значение двойной точности, равное количеству секунд между последовательными tick'ами часов, используемых функцией `omp_get_wtime()`.
В текущей конфигурации `omp_get_wtick()` = 0.000000001.

Вложенный параллелизм

Вложенный параллелизм в OpenMP позволяет создавать параллельные регионы внутри других параллельных регионов. Это позволяет эффективно использовать ресурсы многопроцессорных систем, разделяя задачи между несколькими уровнями параллелизма.
Для проверки, разрешено ли вложение параллельных регионов на текущей машине, можно использовать следующие функции:
`omp_get_nested()` - возвращает, включен вложенный параллелизм или отключен, в соответствии со значением параметра max-active-levels-var ICV;
`omp_get_max_active_levels()` - возвращает значение max-active-levels-var ICV, которое соответствует максимальному уровню вложенности параллелизма.
В текущей конфигурации значения параметров следующие:

- `omp_get_nested()` = 0.
- `omp_get_max_active_levels()` = 1.

Расписания

Опция `schedule(type, [chunk])` задаёт, каким образом распределяются итерации цикла между потоками.
Существуют следующие типы:

- `static` - нулевой поток выполняет первые chunk итераций, первый поток - вторые chunk операций и так далее, пока не закончатся свободные потоки. Потом схема повторяется. В этом случае возможен простой, когда поток выполнил все отведенные ему задачи, а новые взять не может, т.к. они предназначены другим.
- `dynamic` - тот поток, который заканчивает свою порцию из chunk итераций, получает первую свободную порцию.
- `guided` - динамическое распределение итераций, при котором размер порции уменьшается с некоторого начального значения до величины chunk пропорционально количеству ещё не распределённых итераций, делённому на число потоков, выполняющих цикл.
- `auto` - способ распределения выбирается компилятором.
- `runtime` - способ распределения итераций выбирается во время работы программы по значению переменной `OMP_SCHEDULE`.

`omp_get_schedule()` - возвращает параметры расписания, которые применяются при типе распределения `runtime`.
В текущей конфигурации:

- `chunk_size` = 1.
- `schedule_kind` = 2 (т.е. dynamic).

3. OpenMP Locks

В рамках данной секции была реализована "наивная"программа, демонстрирующая необходимость использования механизма явных блокировок.

Явные блокировки в OpenMP используются для предотвращения гонок данных (race conditions) в параллельных программах, когда несколько потоков пытаются одновременно получить доступ к общим ресурсам или изменить общие переменные. Гонки данных могут привести к непредсказуемому поведению программы, такому как неправильные результаты вычислений или повреждение данных.

Явные блокировки в OpenMP обеспечивают безопасный доступ к общим ресурсам, гарантируя, что только один поток может выполнять критические участки кода в любой момент времени. Это позволяет избежать гонок данных и обеспечить корректное выполнение параллельных программ.

В приведенной программе используется общий счётчик, который в результате выполнения должен быть умножен на число предоставленных потоков. Сначала программа запускается с использованием механизма явных блокировок, а затем - без. Как видно из приведенного скриншота, результат выполнения программы во втором случае отличается от ожидаемого:

```
--------------------Started with_locks section--------------------
Started program without locks...
Thread 2 starting...
Thread 1 starting...
Thread 3 starting...
Thread 0 starting...
Thread 0 finished.
Thread 2 finished.
Thread 1 finished.
Thread 3 finished.
Final value of the shared counter without lock: 40000
Expected value: 40000


--------------------Started without_locks section--------------------
Started program without locks...
Thread 2 starting...
Thread 0 starting...
Thread 2 finished.
Thread 3 starting...
Thread 1 starting...
Thread 0 finished.
Thread 1 finished.
Thread 3 finished.
Final value of the shared counter without lock: 31546
Expected value: 40000
```

4. Расписания OpenMP при реализации параллельных алгоритмов

Целью данной секции является сравнение производительности конкретного алгоритма при параллельной реализации с использованием различных видов расписаний. В качестве используемого алгоритма был выбран алгоритм поиска максимального элемента в массиве из 1 лабораторной работы.

Описание эксперимента

- Измеряется время работы алгоритма на одном и том же массиве, но на разном числе потоков: от 1 до 20 и при разных вариантах расписания: `static`, `dynamic`, `guided`, `auto`.
- Измерения проводятся для 50 случайно сгенерированных массивов длиной 10 000 000 элементов.
- Находится среднее время работы для конкретного варианта расписания и конкретного числа потоков.

Построение линейного алгоритма

Был реализован линейный алгоритм поиска максимального элемента в массиве. Среднее время работы составляет 0.033946с для массива длиной 10 000 000 элементов.

Построение параллельного алгоритма

На основе линейного алгоритма был построен параллельный алгоритм поиска максимального элемента. При этом использовались различные типы расписаний. Ниже приведены таблицы среднего времени работы алгоритма на различном числе потоков при различном типе расписаний.

Static, default chunk size

Number of threads	Exec. time
1	0.033946
2	0.023875
3	0.019082
4	0.014214
5	0.007936
6	0.006933
7	0.006466
8	0.007191
9	0.008469
10	0.007761
11	0.007329
12	0.006821
13	0.006481
14	0.006468
15	0.006398
16	0.006362
17	0.006779
18	0.006601
19	0.006633
20	0.006348

Static, 1000 chunk size

Number of threads	Exec. time
1	0.033946
2	0.018103
3	0.011884
4	0.009368
5	0.007886
6	0.006983
7	0.006319
8	0.006824
9	0.008678
10	0.008119
11	0.007624
12	0.007281
13	0.007117
14	0.006958
15	0.006513
16	0.006849
17	0.007141
18	0.006950
19	0.006892
20	0.006741

Dynamic, 100 chunk size

Number of threads	Exec. time
1	0.033946
2	0.022464
3	0.017470
4	0.013894
5	0.011174
6	0.010016
7	0.008910
8	0.009183
9	0.008729
10	0.008824
11	0.008823
12	0.008629
13	0.008518
14	0.008852
15	0.008686
16	0.008623
17	0.008578
18	0.008530
19	0.008554
20	0.008607

Dynamic, 1000 chunk size

Number of threads	Exec. time
1	0.033946
2	0.018660
3	0.012512
4	0.009615
5	0.007852
6	0.006773
7	0.006241
8	0.005875
9	0.005924
10	0.006070
11	0.005856
12	0.005750
13	0.005803
14	0.005893
15	0.005785
16	0.005970
17	0.005707
18	0.005807
19	0.005808
20	0.005799

Guided, default chunk size

Number of threads	Exec. time
1	0.033946
2	0.021913
3	0.012838
4	0.010382
5	0.007196
6	0.006212
7	0.005691
8	0.005876
9	0.005989
10	0.005764
11	0.005647
12	0.006053
13	0.005869
14	0.005759
15	0.005893
16	0.005900
17	0.005699
18	0.005639
19	0.005705
20	0.006076

Guided, 1000 chunk size

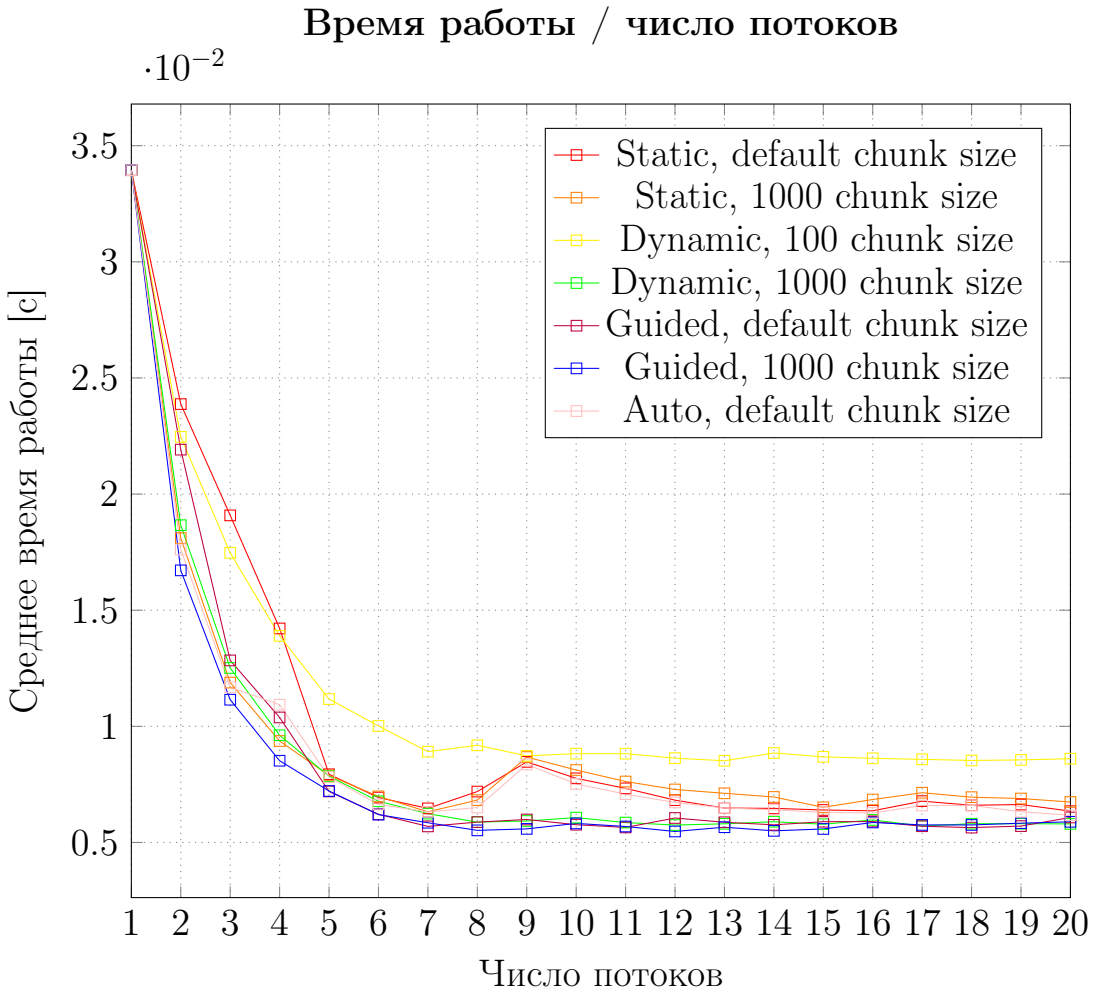
Number of threads	Exec. time
1	0.033946
2	0.016715
3	0.011146
4	0.008519
5	0.007216
6	0.006199
7	0.005849
8	0.005521
9	0.005584
10	0.005820
11	0.005687
12	0.005473
13	0.005656
14	0.005500
15	0.005581
16	0.005861
17	0.005755
18	0.005754
19	0.005830
20	0.005888

Auto, default chunk size

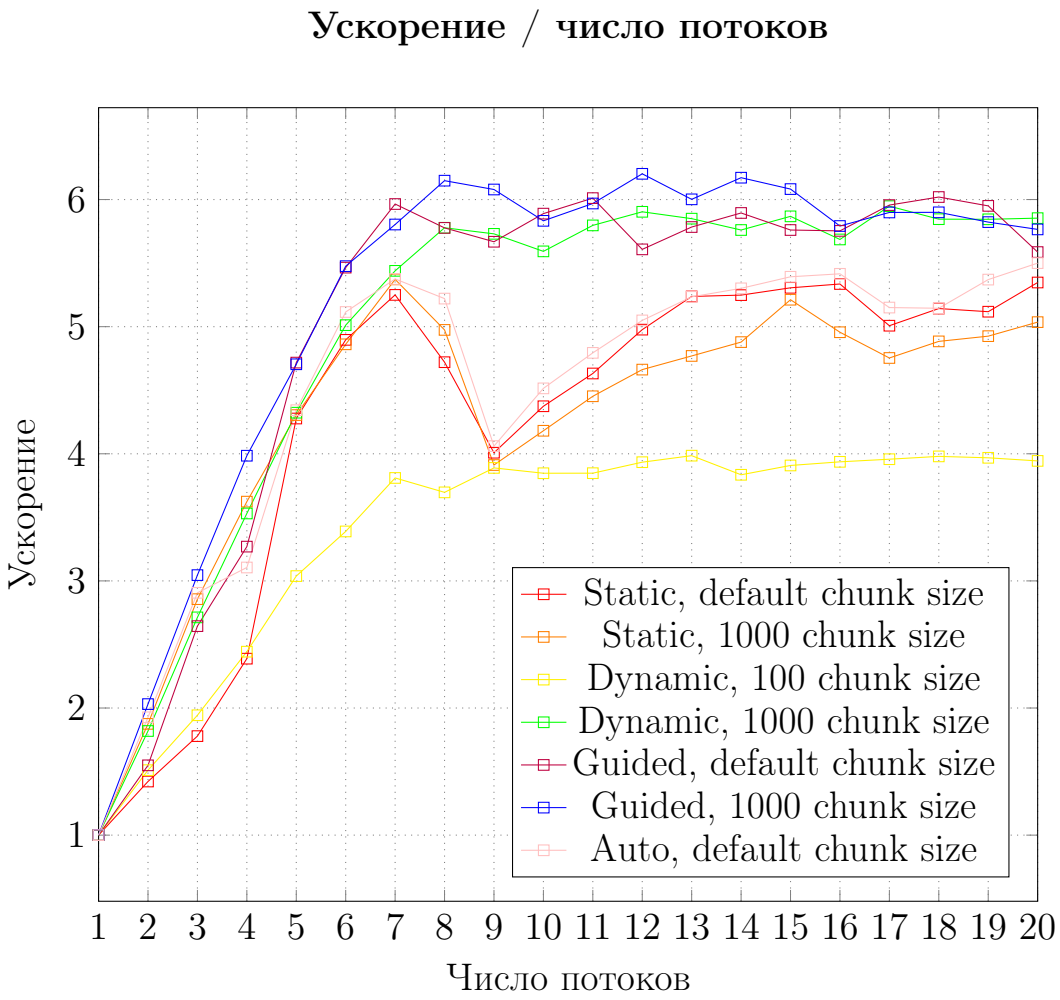
Number of threads	Exec. time
1	0.033946
2	0.017616
3	0.011680
4	0.010931
5	0.007816
6	0.006636
7	0.006315
8	0.006502
9	0.008360
10	0.007519
11	0.007081
12	0.006723
13	0.006485
14	0.006403
15	0.006295
16	0.006268
17	0.006591
18	0.006598
19	0.006321
20	0.006172

Графики

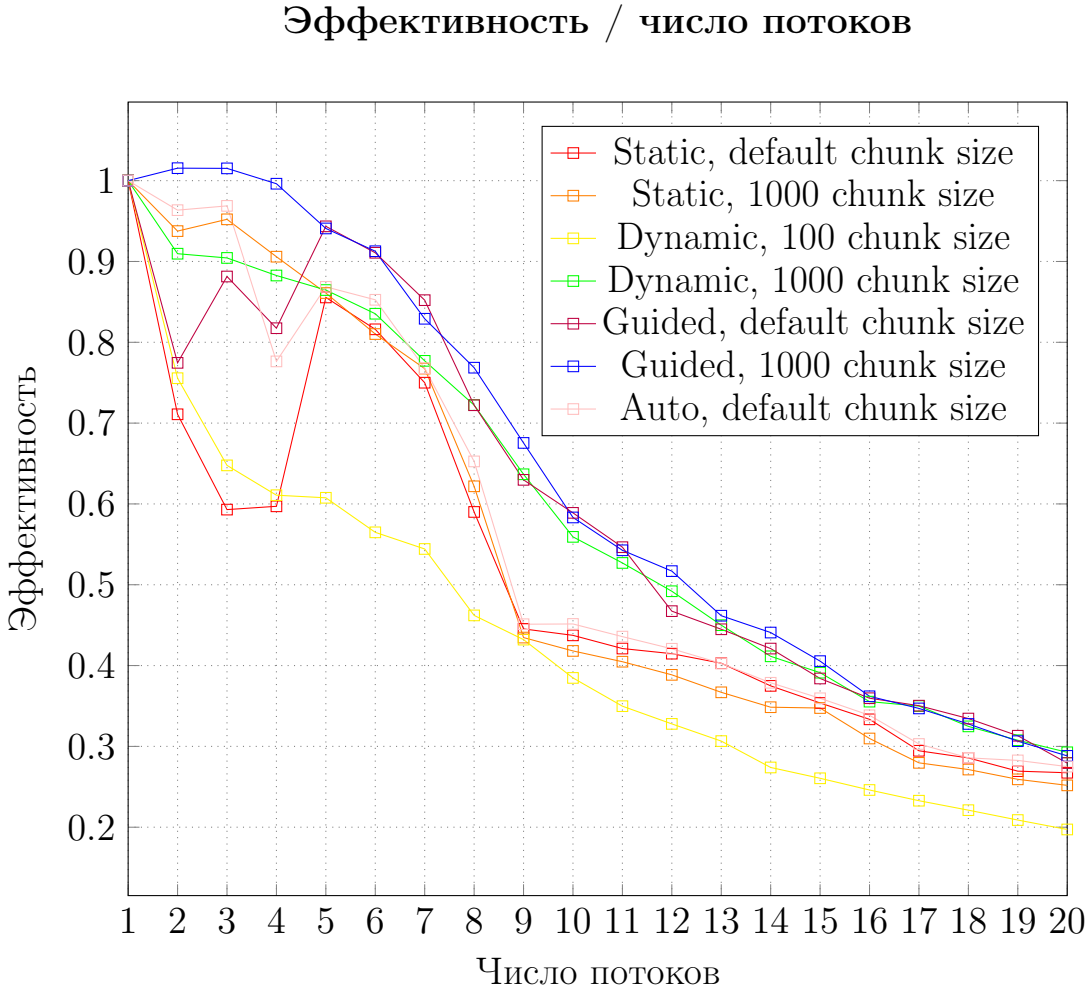
Среднее время работы



Ускорение



Эффективность



5. Заключение

В ходе лабораторной работы были получены текущие настройки OpenMP в используемой системе.

Была реализована программа, демонстрирующая необходимость использования механизма явных блокировок.

Было изучено влияние использования различных типов расписаний на производительность алгоритма поиска максимального элемента в массиве:

- Самым неэффективным оказался Динамический тип расписания с небольшим размером секции (100). Максимальное ускорение при данном типе расписания достигается на 18 потоках и составляет 3.97. Эффективность при этом составляет лишь 0.22.
- Наибольшие значения ускорения показал тип расписаний Guided. При стандартном размере секции максимальное значение ускорения составляет 6.02 на 18 потоках, при размере секции 1000 - 6.20 на 12 потоках. Так же хорошие значения ускорения показал динамический тип расписания с размером секции 1000 - 5.90 на 12 потоках.
- Типы расписаний Static и Auto вели себя примерно одинаково независимо от размера секции у Static. Кривые ускорения и среднего времени работы имеют очень похожий вид. Также у всех трех кривых наблюдается просадка производительности на 9 потоках: ускорение составляет примерно 4.00, хотя на 8 потоках было выше 4.70.

6. Приложение

Код программы, таблицы расположены на github

Запуск программы: `make all`