МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский ядерный университет «МИФИ»»

ЛАБОРАТОРНАЯ РАБОТА №4:

«Технология OpenMP. Особенности настройки»

Аверин Владислав

Группа Б19-505

Октябрь, 2021

Содержание

Характеристики лабораторного оборудования 3
Описание директив, опций и функций 4
Полученные настройки рабочей среды 6
Пример вычислительного алгоритма на основе механизма явных блокировок 7
Вычислительный эксперимент для разной настройки распределения нагрузки 9
Выводы 13

Характеристики лабораторного оборудования

Процессор: 11th Gen Intel Core i7-1185G7 3.00Ghz (8 CPUs)

RAM: 16Гб DDR4 3200МГц

Используемая версия _OpenMP: 201511 (December, 2015)

Операционная система: OS Linux Manjaro KDE Plasma 5.22.5; версия ядра: 5.10.68-1-MANJARO (64-бита), работа от сети

Редактор кода: Visual Studio Code 1.60.1

Компилятор: GCC 11.1.0

Параметры командной строки: -O0 -fopenmp (без оптимизации и отладочной информации для чистоты эксперимента)

Описание директив, опций и функций

- ***_OPENMP*** - переменная препроцессора, определяющая дату принятия используемого стандарта OpenMP; возвращает int переменную в виде yyyy:mm
- Опция ***schedule(type, chunk_size)*** определяет способ разделения итераций цикла for по нитям. Параметр type может принимать следующие значения (типы распределения):
  - ***static*** - блочно-циклическое распределение итераций цикла; размер чанка - chunk_size (default : кол-во итераций / кол-во нитей; большие по размеру чанки отдаются наипервейшим нитям).
  - ***dynamic*** - динамическое распределение итераций с фиксированным размером блока: сначала каждая из нитей получает chunk_size итераций (default : 1), далее та нить, которая закончила выполнение своей порции, получает следующую свободную порцию из chunk_size итераций. Так будет происходить до тех пор, пока все итерации не будут обработаны.
  - ***guided*** - так же динамическое распределение итераций, но с уменьшающимся размером чанка пропорционально оставшейся части цикла: начальное значение - зависит от реализации; конечное - chunk_size (default: 1);
  - ***auto*** - способ распределения выбирается компилятором и/или исполняющей системой;
  - ***runtime*** - способ распределения выбирается во время работы программы по значению переменной OMP_SCHEDULE. Параметр chunk_size при этом не задается (определяется дефолтным для данного способа);
- ***int omp_get_dynamic()*** - возвращает значение переменной *OMP_SCHEDULE*, которая показывает, разрешено ли системе динамически изменять кол-во нитей, используемых для выполнения параллельной области (например, для оптимизации использования ресурсов системы);
- ***double omp_get_wtick()*** - возвращает в вызывавшей нити разрешение таймера в секундах, что можно рассматривать как предельную точность таймера;
- ***int omp_get_nested()*** - возвращает значение переменной *OMP_NESTED*, которая показывает, разрешен ли вложенный параллелизм. По умолчанию (*OMP_NESTED* = 0) любой вложенный цикл выполняется последовательно соответствующей зашедшей в него нитью. Если же

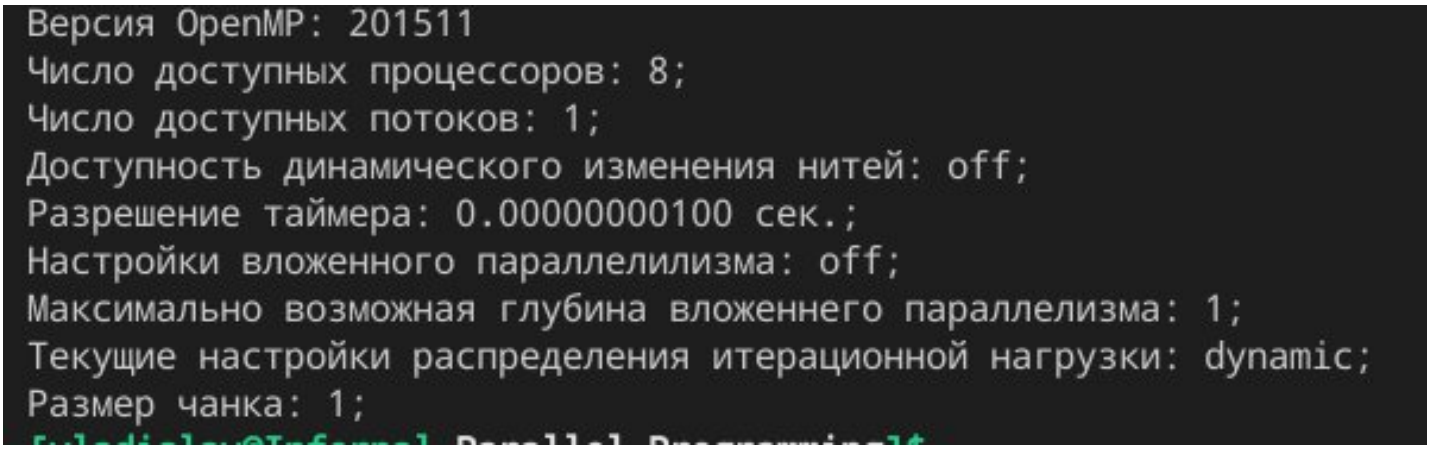
OMP_NESTED = 1, то эта нить породит свою группу нитей и станет в ней нитью-мастером.

- ***int omp get max active levels()*** - возвращает значение переменной *OMP_MAX_ACTIVE_LEVELS*, которая обозначает максимальную глубину возможного распаралелленногго цикла (default : 1)
- ***void omp get schedule(omp_sched_t* type, int* chunk_size)*** - возвращает обновленные значения chunk_size и type, соответствующие текущим дефолтным настройкам системы. Значения type определены в енаме библиотеки OpenMP:

```
/* schedule kind constants */
typedef enum omp_sched_t {
    omp_sched_static  = 1,
    omp_sched_dynamic = 2,
    omp_sched_guided  = 3,
    omp_sched_auto    = 4,
    omp_sched_monotonic = 0x80000000
} omp_sched_t;
```

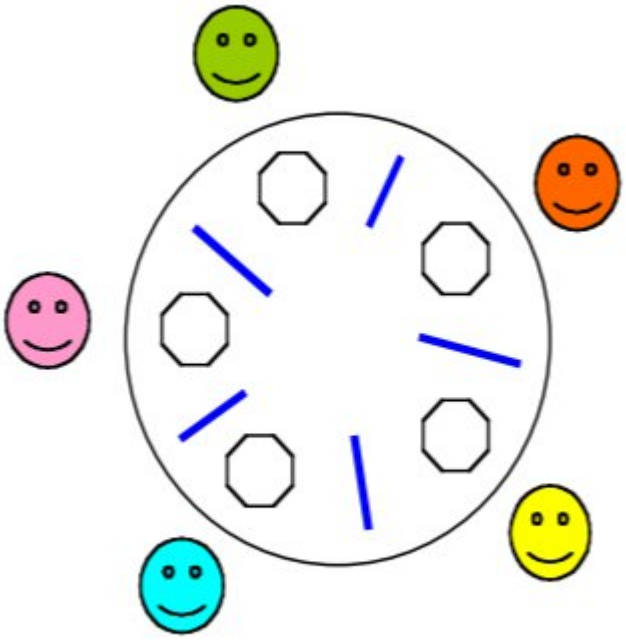
Полученные настройки рабочей среды

Программный вывод:



Пример вычислительного алгоритма на основе механизма явных блокировок

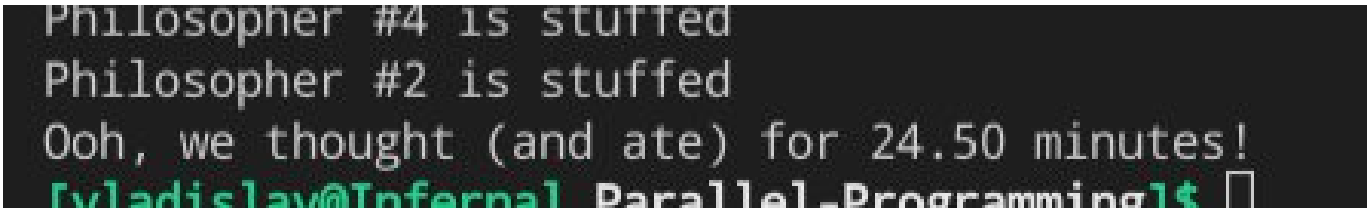
На самом деле, алгоритм не столько вычислительный, сколько интерактивный. Он решает одну конкретную задачу, но наглядно показывает возможности явных блокировок нитей. Задачу назовем ***«Филосовский обед».***



За круглым обеденным столом сидят N китайских философов. Перед каждым философом стоит тарелка с рисом - его обед. Чтобы начать трапезу, философу необходимы свободные левая и правая палочки, причем каждая палочка разделена между двумя соседними философами. Но как известно, китайским философам дай только повод пофилософсвовать. Таким образом, пока оба соседа получают вселенскую энергию от первозданности бытия и пребывания на триллионах таких же планет, как эта, философ, находящийся между ними, может придаться тихому и планоразмерному чревоугодию. После того, как он выполнил свою миссию по истреблению съестных запасов, перед ним возникает как на Скатерти-Самобранке следующая порция (Ма-а-agic), дабы не отвлекать его от гармонии со слиянием вечного с бесконечным и созерцания этого фрактального подобия. Продолжительность процесса философствования каждого из участников «голодных» игр определяется случайным образом, так же, как и время приема одной порции (но в два раза больше:)). Если при окончании периода раздумий перед философом не наблюдается полного набора палочек, он начинает снова думать (видимо, пытаясь материализовать новые палочки).

Необходимо посчитать время, за которое все философы за столом примут _MEALS порций.

Пример работы программы для значений N = 5 и _MEALS = 3:



Время, полученное таким образом, всегда будет разным (в каком-то интервале для конкретных значений), т.к. все процессы длятся разное время, и неизвестно, какой из элементов массива в данный момент залочет / разлочет «палочки».

Необходимость явных блокировок в данном примере заключается в том, что процессы потоков (философов) не независимые: каждый из потоков начинает новый процесс только тогда, когда обе палочки будут доступны (перестанут использоваться другим потоком). Эту уже задачу можно было бы сделать и без механизма замков, однако тогда бы она была последовательная и очень запутанная (либо же мы бы написали тот же самый принцип блокировки, просто черед костыли)

Вычислительный эксперимент для разной настройки распределения нагрузки

Будем применить разные опции schedule() для 1 лабы, где нужно было найти максимум / минимум в массиве.

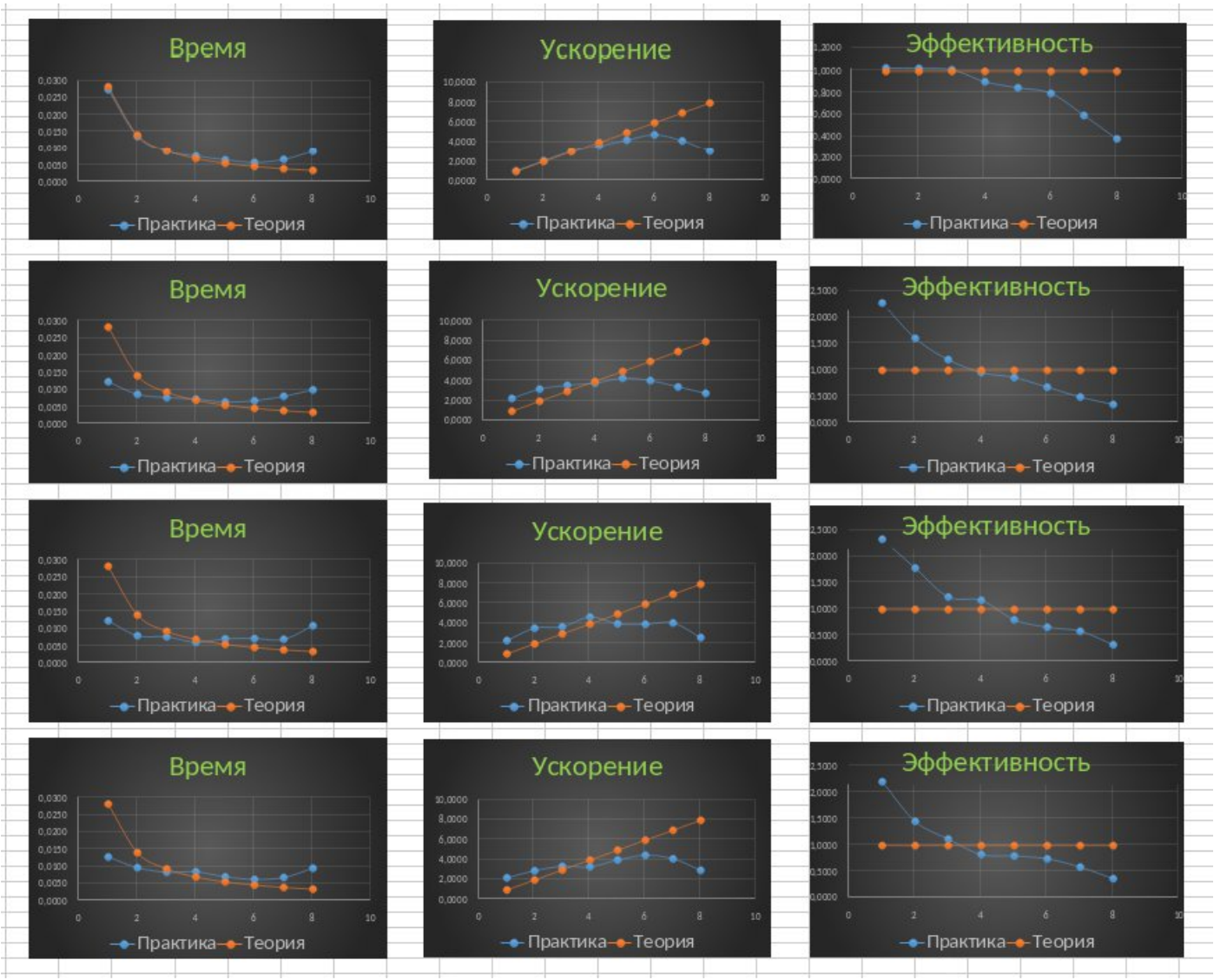
Будем проверять опции static, dynamic и guided (типы runtime и auto являются одними из этих трех типов). Для определенности chunk_size возьмем за 4 различных значения: дефолт (не указываем), P, P^2, P^3 (P - кол-во процессоров).

Сводная таблица:

consecutive	0,028436	0,028472	0,029302	0,028712	0,029556	0,028806	0,02895	0,029008	0,028502	0,02867	0,028639	0,028434
	Static				Dynamic				Guided			
	N/A	p	p^2	p^3	N/A	p	p^2	p^3	N/A	p	p^2	p^3
1	0,0276	0,0125	0,0125	0,0130	0,1108	0,1103	0,1101	0,1118	0,0275	0,0275	0,0275	0,0274
2	0,0138	0,0088	0,0082	0,0098	0,2622	0,1238	0,0862	0,0456	0,0137	0,0138	0,0138	0,0138
3	0,0093	0,0079	0,0079	0,0085	0,2774	0,1081	0,0394	0,0152	0,0075	0,0093	0,0092	0,0098
4	0,0079	0,0074	0,0062	0,0086	0,2645	0,0851	0,0198	0,0105	0,0062	0,0075	0,0073	0,0077
5	0,0067	0,0066	0,0073	0,0072	0,2349	0,0572	0,0120	0,0079	0,0058	0,0061	0,0063	0,0061
6	0,0059	0,0070	0,0074	0,0064	0,2109	0,0428	0,0091	0,0066	0,0055	0,0056	0,0058	0,0056
7	0,0068	0,0083	0,0071	0,0070	0,1975	0,0324	0,0090	0,0062	0,0055	0,0053	0,0065	0,0055
8	0,0093	0,0101	0,0111	0,0097	0,1998	0,0285	0,0118	0,0083	0,0094	0,0085	0,0078	0,0079
Acceleration	1,0306	2,2814	2,3372	2,2146	0,2667	0,2611	0,2629	0,2595	1,0350	1,0426	1,0407	1,0361
	2,0546	3,2197	3,5747	2,9200	0,1127	0,2328	0,3358	0,6364	2,0760	2,0730	2,0827	2,0667
	3,0439	3,6104	3,7190	3,3719	0,1066	0,2666	0,7351	1,9098	3,7977	3,0914	3,1146	2,8973
	3,6201	3,8325	4,6988	3,3258	0,1118	0,3384	1,4596	2,7658	4,5735	3,8380	3,9291	3,6817
	4,2505	4,3179	4,0206	4,0089	0,1258	0,5034	2,4177	3,6770	4,9184	4,6808	4,5786	4,6355
	4,7929	4,0814	3,9785	4,4848	0,1402	0,6731	3,1677	4,4152	5,1438	5,0761	4,9386	5,1039
	4,1824	3,4453	4,1074	4,1277	0,1496	0,8890	3,2267	4,6472	5,1747	5,4610	4,4114	5,1868
	3,0639	2,8087	2,6412	2,9704	0,1479	1,0094	2,4501	3,5034	3,0328	3,3757	3,6712	3,6002
Effeciency	1,0306	2,2814	2,3372	2,2146	0,2667	0,2611	0,2629	0,2595	1,0350	1,0426	1,0407	1,0361
	1,0273	1,6099	1,7874	1,4600	0,0564	0,1164	0,1679	0,3182	1,0380	1,0365	1,0413	1,0334
	1,0146	1,2035	1,2397	1,1240	0,0355	0,0889	0,2450	0,6366	1,2659	1,0305	1,0382	0,9658
	0,9050	0,9581	1,1747	0,8315	0,0279	0,0846	0,3649	0,6915	1,1434	0,9595	0,9823	0,9204
	0,8501	0,8636	0,8041	0,8018	0,0252	0,1007	0,4835	0,7354	0,9837	0,9362	0,9157	0,9271
	0,7988	0,6802	0,6631	0,7475	0,0234	0,1122	0,5280	0,7359	0,8573	0,8460	0,8231	0,8507
	0,5975	0,4922	0,5868	0,5897	0,0214	0,1270	0,4610	0,6639	0,7392	0,7801	0,6302	0,7410
	0,3830	0,3511	0,3302	0,3713	0,0185	0,1262	0,3063	0,4379	0,3791	0,4220	0,4589	0,4500

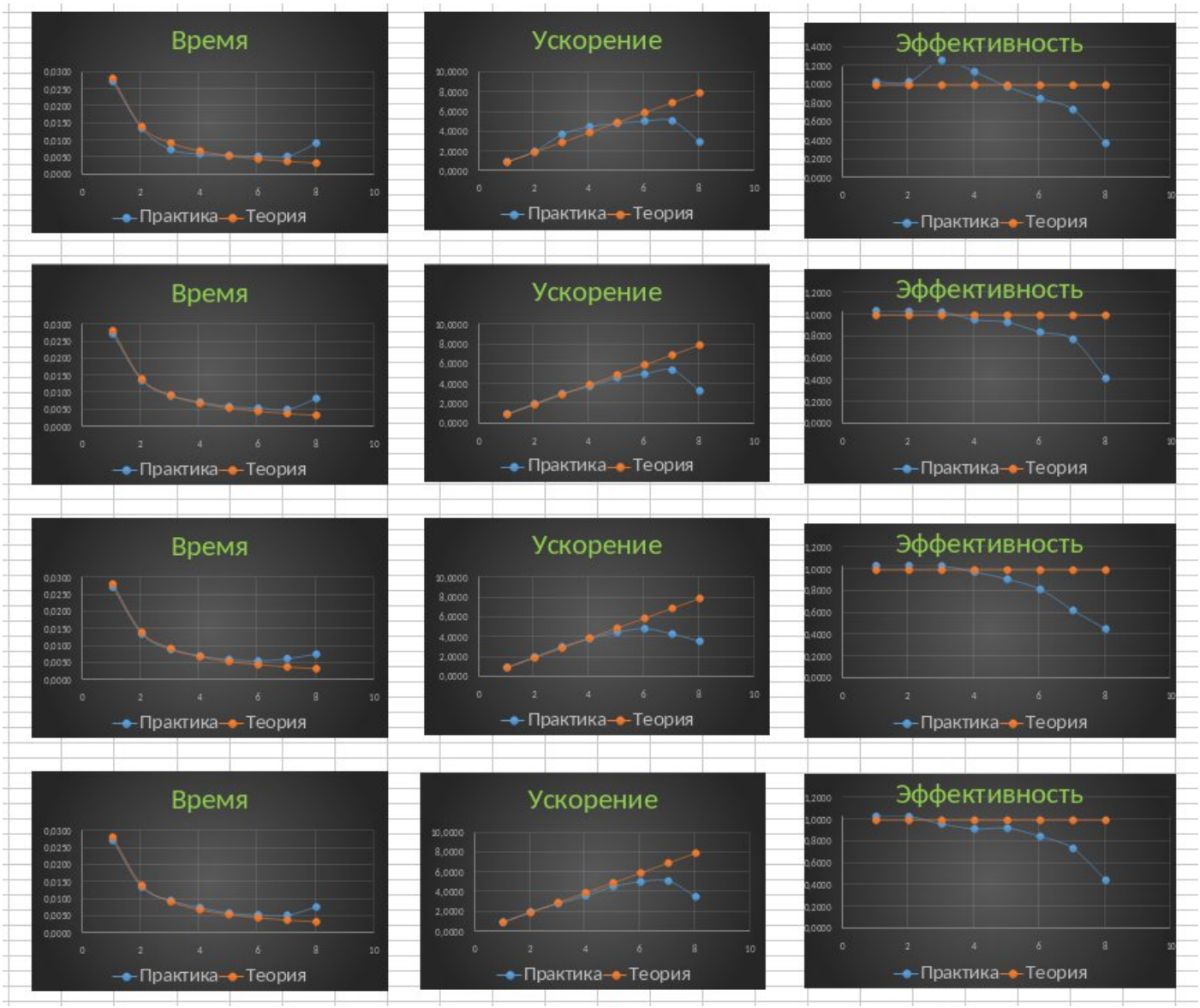
(Значения chunk_size идут в таком же порядке сверху-вниз.)

- *Для static:*



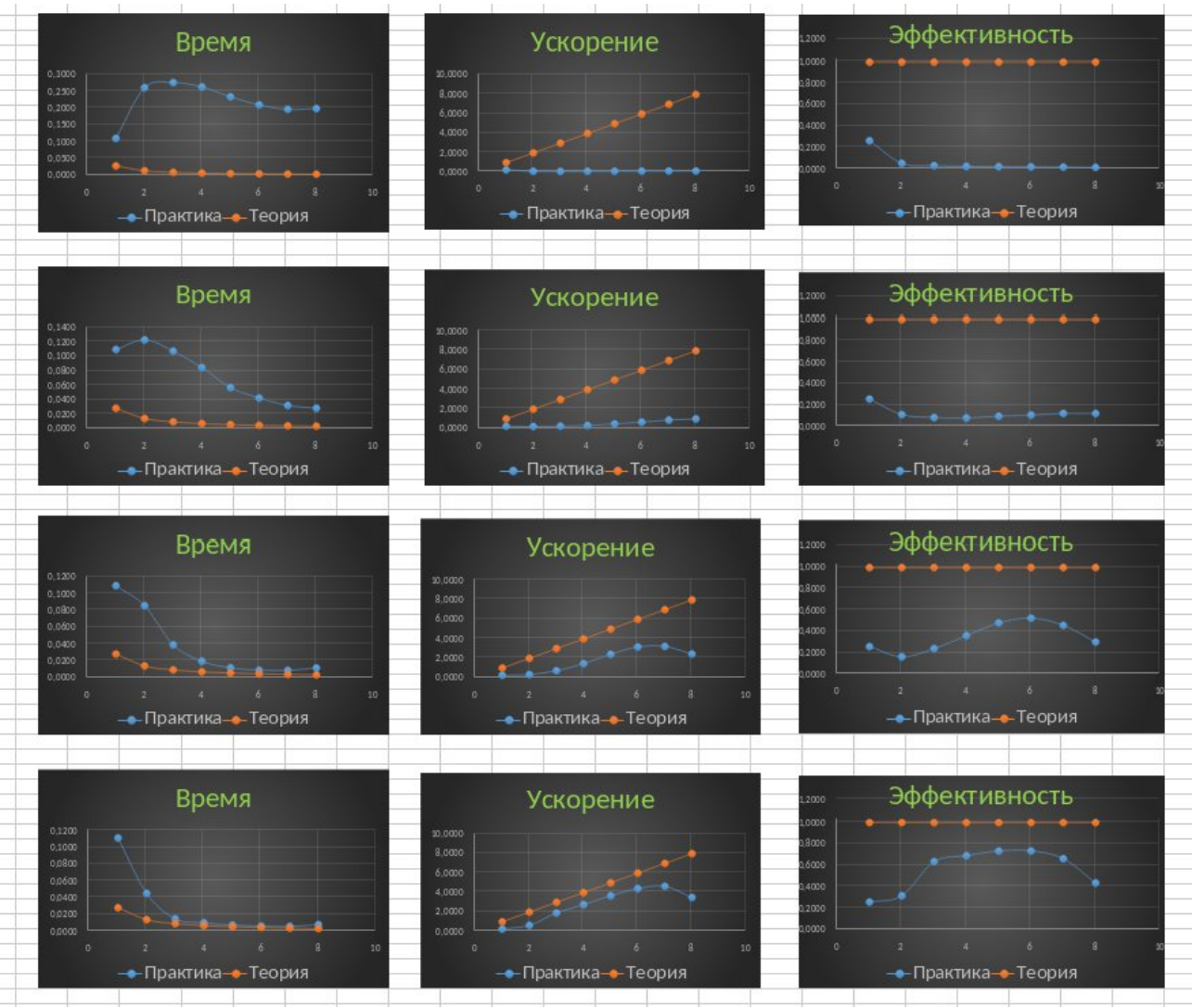
Как мы видим, ускорение почти идеально совпадает с теоретическим, а где-то даже аномально его опережает.

- *Для guided:*



Для guided все более адекватно и идеально, однако так же, как и в предыдущих лабораторных наблюдается просадка производительности на максимально возможном кол-ве сгенерированных потоков (8).

- А вот для *dynamic* все катастрофически плохо:



Затраты на передачу очередного блока освободившейся нити намного превышают время самой итерации сравнения. Используя бы мы эту опцию в более сложных конструкциях (например, в 3 лабораторной, где dynamic в моем случае в среднем ускорял работу в 2 раза независимо от кол-ва процессоров), мы бы получили более близкий к теории результат. Однако здесь наблюдается крайняя неэффективность распараллеливания с помощью динамического распределения.

Можно заметить еще один факт: чем больше само значение chunk_size, тем более лучше время стремится к нужным цифрам. Связано это с тем, что с увеличением размера чанка уменьшается кол-во новых «передач». То есть технически, увеличь бы мы размер чанка до count / P, мы бы получили максимальную эффективность. Что доказывают графики для guided-распределения: это такое же динамическое распределение итераций, однако в отличие от dynamic размер чанков уменьшается пропорционально оставшимся итерациям, гарантируя тем самым, что все нити будут нагружены примерно одинаково.

Выводы

В ходе данной лабораторной работы были изучены различные настройки стандарта OpenMP, в том числе: динамическое распределение нитей, опции распределения итераций цикла по потокам, возможности вложенного параллелизма и механизма явных блокировок. Для последнего была приведена в пример прикладная задача и ее решение с помощью распараллеливания и замков.

Что же касается настроек *chedule*, то были сделаны следующие выводы:

1. Хуже всего показало себя динамическое распределение dynamic, что указывает на то, что его нужно использовать лишь в исключительных ситуациях, т.к. время на присваивание очередного чанка освободившейся нити очень большое;
2. Программные коды предыдущих 3 лабораторных были оттаймированы на настройке schedule(dynamic), поэтому показывали не очень хорошие результаты; при изменении настройки на static производительность вычислений во всех 3 работах значительно возросла почти до теоретического значения.