***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](#_Toc1)

[*Описание директив, опций и функций* 4](#_Toc2)

[*Полученные настройки рабочей среды* 6](#_Toc3)

[*Пример вычислительного алгоритма на основе механизма явных блокировок* 7](#_Toc4)

[*Вычислительный эксперимент для разной настройки распределения нагрузки* 9](#_Toc5)

[*Выводы* 13](#_Toc6)

## *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

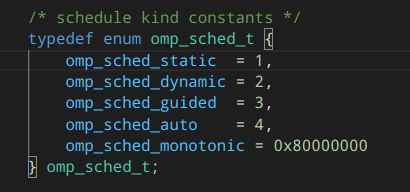
*Редактор кода*: Visual Studio Code 1.60.1

*Компилятор*: GCC 11.1.0

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

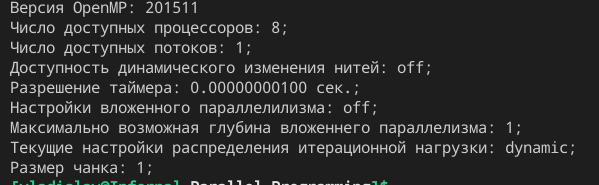
## *Описание директив, опций и функций*

* ***\_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:



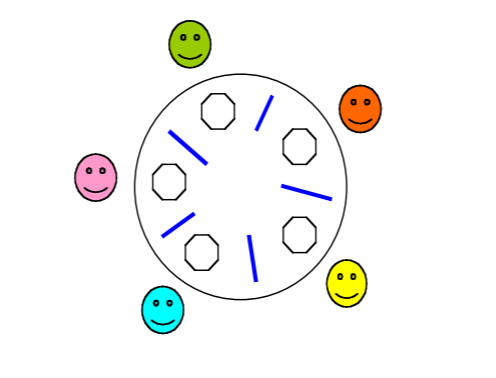
## *Полученные настройки рабочей среды*

Программный вывод:



## *Пример вычислительного алгоритма на основе механизма явных блокировок*

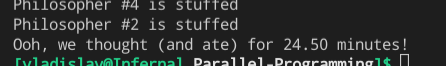
На самом деле, алгоритм не столько вычислительный, сколько интерактивный. Он решает одну конкретную задачу, но наглядно показывает возможности явных блокировок нитей. Задачу назовем ***«Филосовский обед».***



*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 да й 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 (Ma-a-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».

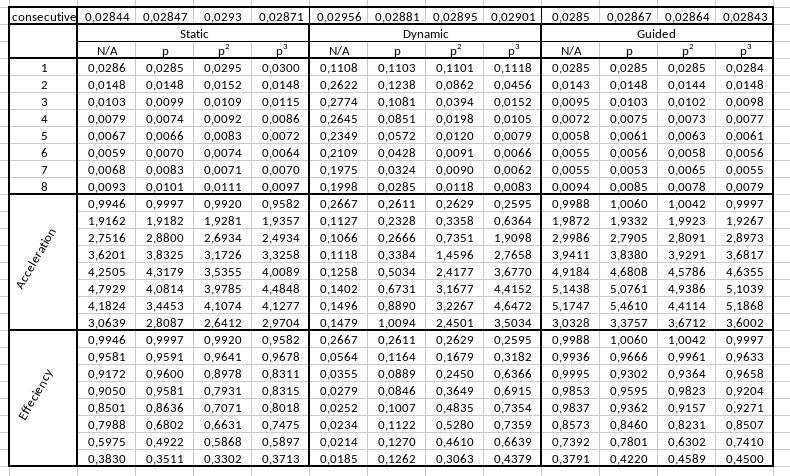
***Необходимость явных блокировок*** в данном примере заключается в том, что процессы потоков (философов) не независимые: каждый из потоков начинает новый процесс только тогда, когда обе палочки будут доступны (перестанут использоваться другим потоком). Эту уже задачу можно было бы сделать и без механизма 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.

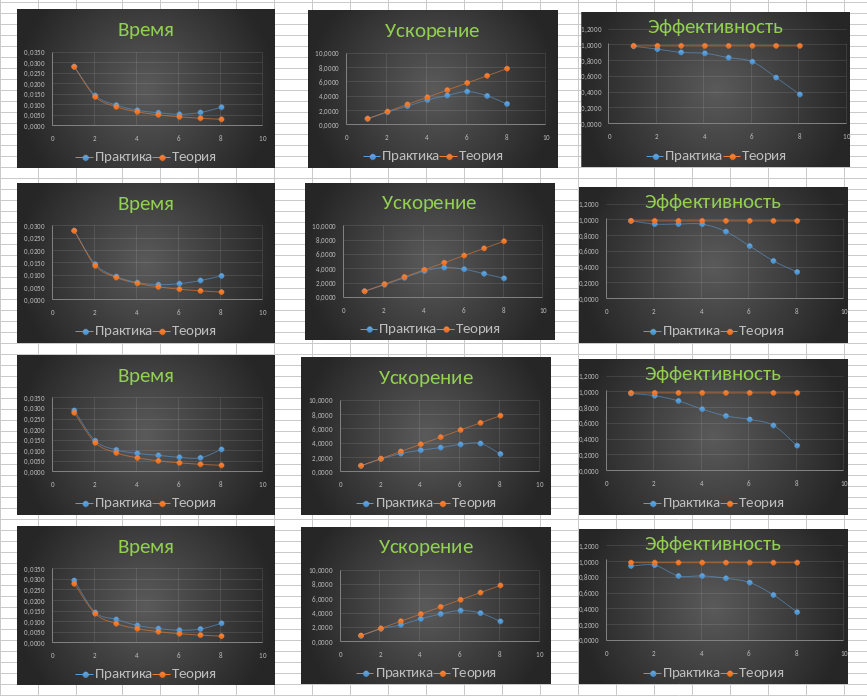
Будем проверять опции static, dynamic и guided (типы runtime и auto являются одними из этих трех типов). Для определенности chunk\_size возьмем за 4 различных значения: дефолт (не указываем), P, P2, P3 (P - кол-во процессоров).

*Сводная таблица:*



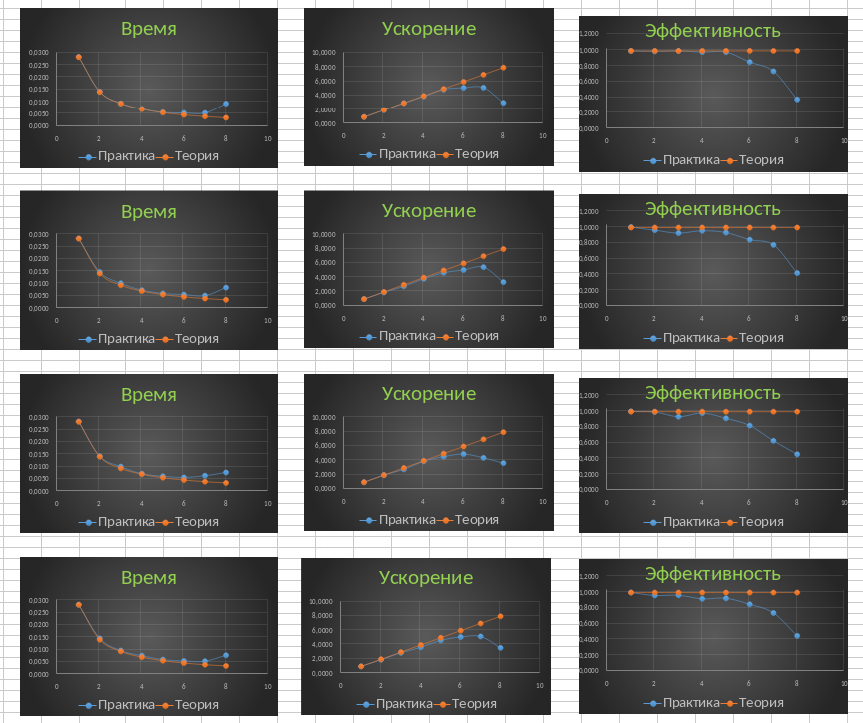
(Значения chunk\_size идут в таком же порядке сверху-вниз.)

* *Для static:*



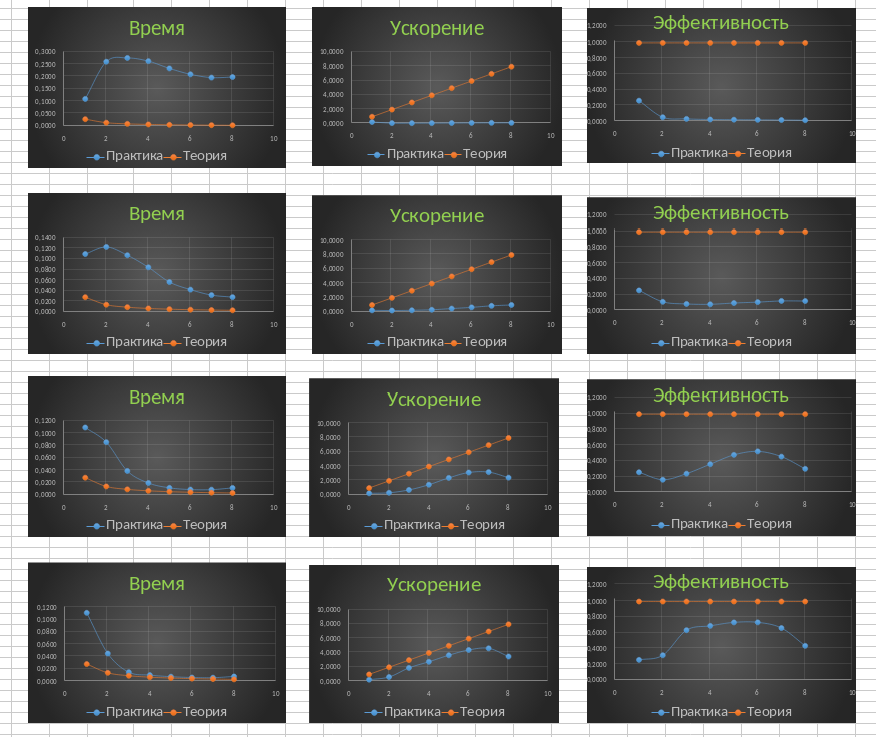
Как мы видим, ускорение почти идеально совпадает с теоретическим, постепенно теряя в эффективности (с увеличением числа потоков).

* *Для guided:*



Для guided все еще более идеально, однако так же, как и в предыдущих лабораторных наблюдениях видна 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, настроенный по дефолту 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.