1) Инструментарий: Язык: c++20. Компилятор: clang version 17.0.2 3)
   Результат работы написанной программы:

результаты работы написанной программы				
потоки	время	значение		
0 (без om	36,3338	35.9984		
1	35,4306	36.0003		
2	17,355	36.002		
4	10,5697	36.0037		
8	7,43018	36.0022		

   Наилучшее распределение потоков: 8
   Стандартный поток вывода: Time (8 thread(s)): 7.81159
   ms Вывод в файл: 36 36.0022 Процессор:

Intel Core i5 8265U

- **Ядер:** 4
- **L3-кэш:** 6МБ (общий)
- **TDP:** 15 Вт
- **Размер транзистора:** 14 нанометров

Core i5 8265U - процессор для ноутбуков от компании Intel для сокета BGA-1528, который имеет 4 ядра и 8 потоков. Его базовая частота – 1600 МГц, но поддержка технологии Turbo Boost позволяет автоматически разгоняться до 3900 МГц. Данный чип имеет интегрированную графику Intel UHD Graphics 620, а размер кэша 3-го уровня составляет 6 МБ.

2) Описание конструкций OpenMP: в моем коде я использовала:
   - `#pragma omp parallel (num_threads(threads))` – часть этого кода должна выполняться параллельно несколькими потоками. Дефолтно задается число потоков, равное количеству заявленных процессором (4 ядра по 2 потока). (8 стр документации см.источники)
   - `#pragma omp for` – Директива for идентифицирует итеративную конструкцию разделения работы, которая указывает, что итерации соответствующего цикла будут выполняться параллельно. Конструкция нужна для того, чтобы разбить цикл для потоков (11 стр) (Без указания данной инструкции цикл будет выполнен каждым потоком полностью). Также у “for” можно задать параметр schedule – распределение частей кода между потоками. Из используемых (протестированных) : static\dynamic с возможным значением chunk_size – блоков кода – или его отсутствием.
     - При schedule(static, chunk_size) итерации делятся на блоки размера, указанного chunk_size. Блоки статически назначаются потокам в команде в циклическом режиме в порядке числа потоков. Если chunk_size не указано, пространство итерации делится на блоки,

которые приблизительно равны размеру, с одним блоком, назначенным каждому потоку.

- При schedule(dynamic, chunk_size) итерации делятся на ряд блоков, каждый из которых содержит chunk_size итерации. Каждому блоку назначается поток, ожидающий назначения. Поток выполняет блок итерации, а затем ожидает следующего назначения, пока не будут назначены блоки. Последняя часть, назначаемая для назначения, может иметь меньшее количество итераций. Если chunk_size не задано, значение по умолчанию равно 1. (стр. 13)

- `#pragma omp atomic` – Директива “atomic” гарантирует, что определенная ячейка памяти обновляется в каждом потоке. Нужно для избежания коллизий (например, записи результата в одну глобальную переменную несколькими потоками одновременно) (19 стр.) Еще некоторые функции, связанные с omp:

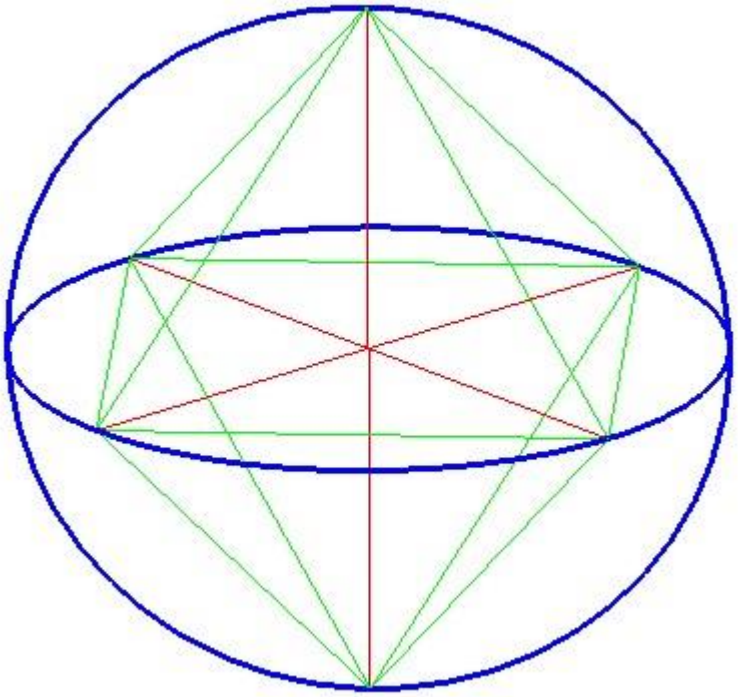
- `omp_get_thread_num()` – выводит номер текущего потока
- `omp_get_wtime()` – равен времени, прошедшему с некоторого «времени в прошлом». Фактическое «время в прошлом» произвольно, но оно гарантированно не изменится в течение выполнения программы.

3) Описание работы написанного кода:
   - С помощью fopen и fscanf – значения переменных из заданных аргументов и обработка ошибок
   - Объем фигуры считается с помощью метода Монте-Карло: Октаэдр помещается в более простое тело, случайным образом в этом теле выбираются точки и идет проверка: принадлежит ли эта точка октаэдру? Это помогает вычислить отношение между объемами фигур, и чем больше точек, тем выше точность. В качестве тела я выбрала куб, т.к. в него можно вписать правильный октаэдр и на нем просто задавать случайные числа, в силу того, что фигура имеет параллельные грани.

- Между тремя вершинами октаэдра обязательно найдутся две смежные (соединенные ребром), а также это будет минимально возможное

В *x*-*y*-*z* прямоугольной системе координат октаэдр с центром в точке (*a*, *b*, *c*) и радиусом *r* — это множество всех точек (*x*, *y*, *z*), таких, ч

$$|x-a|+|y-b|+|z-c|=r.$$



расстояние между любыми двумя вершинами правильной фигуры. Тогда между тремя вершинами найдем наименьшее расстояние – ребро правильного октаэдра. Найдем радиус описанной сферы = ребро\2 * sqrt(2) => теперь мы можем найти сторону куба = радиус * 2. Также я выбираю случайные точки, пометив центр тел в начало координат (без разницы, если ребро октаэдра известно). Тогда для каждой точки проверка следующая:

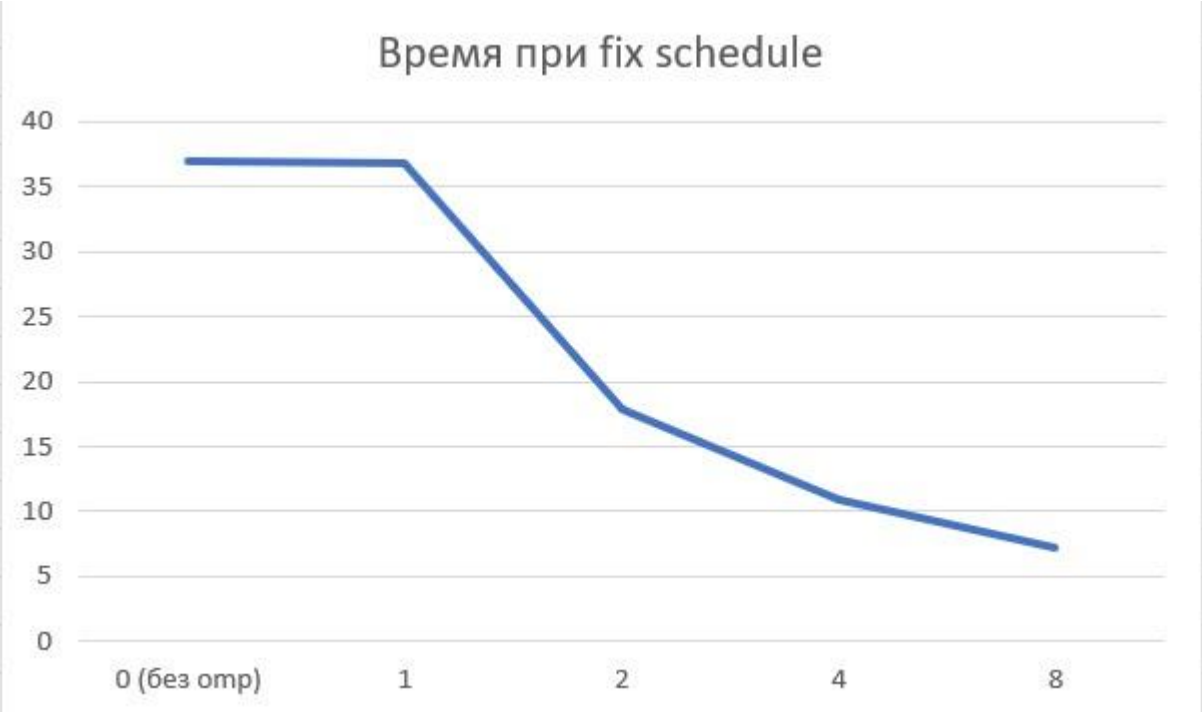
- Генерация “случайных” чисел: алгоритм xorshift. Этот алгоритм просто выполняет операцию Xor и сдвигает байт в несколько раз. У него будет прослеживаться паттерн для i-тых элементов последовательностей, поэтому при ненулевом значении потоков нужно задавать сид: изначальное число, от которого будет происходить генерация. Сид задается с помощью функции хэширования от номер текущего потока (`random_seed()`). Этот алгоритм является одним из наиболее быстрых и простых, а повторяющийся паттерн я постаралась компенсировать заданием сида. Также я выбрала именно 64-битную версию для создания большего периода генерации точек.

- После подсчета точек, попавших в октаэдр выводится полученное значение: `float result_random = V_cube * (float)oct / n;` где v_cube – объем куба, oct – количество попавших точек, n – количество точек.

4) Тестирование

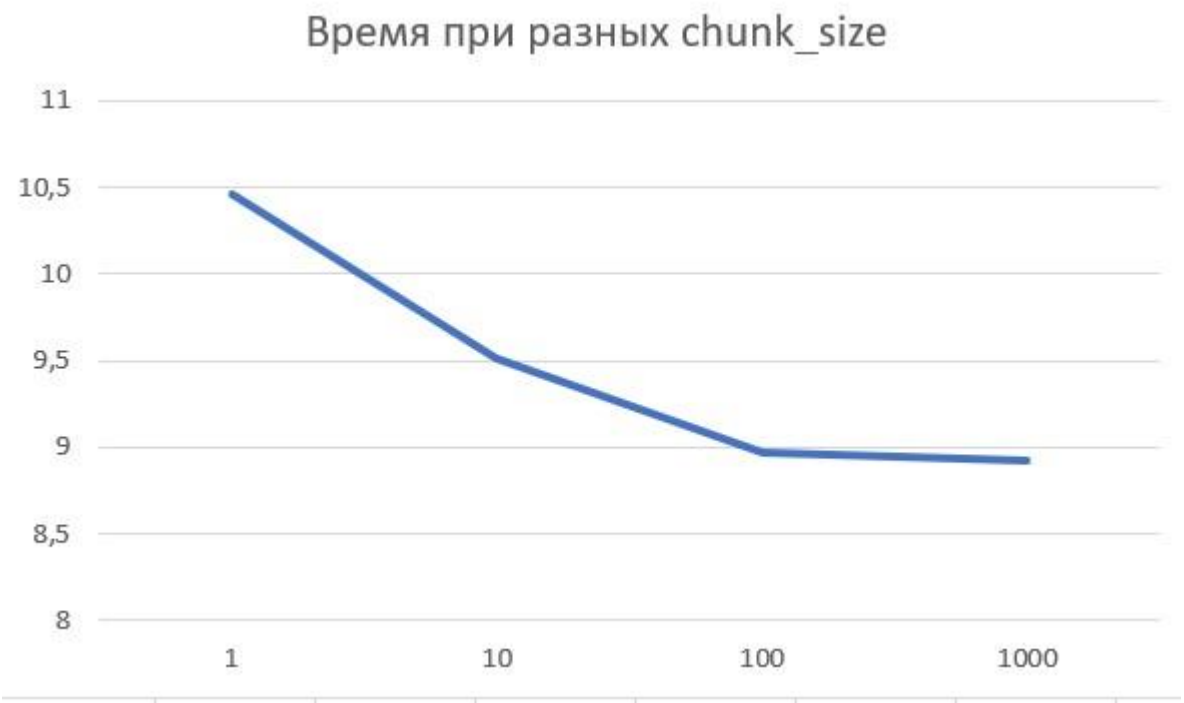
a при различных значениях числа потоков при одинаковом параметре (`#pragma omp for schedule (static)`)

0 (без omp	36,94713	36,9126	37,0077	36,9211
1	36,76717	35,4306	37,8874	36,9835
2	17,89547	17,355	18,7826	17,5488
4	10,98397	10,5697	11,673	10,7092
8	7,233467	7,43018	7,22173	7,04849



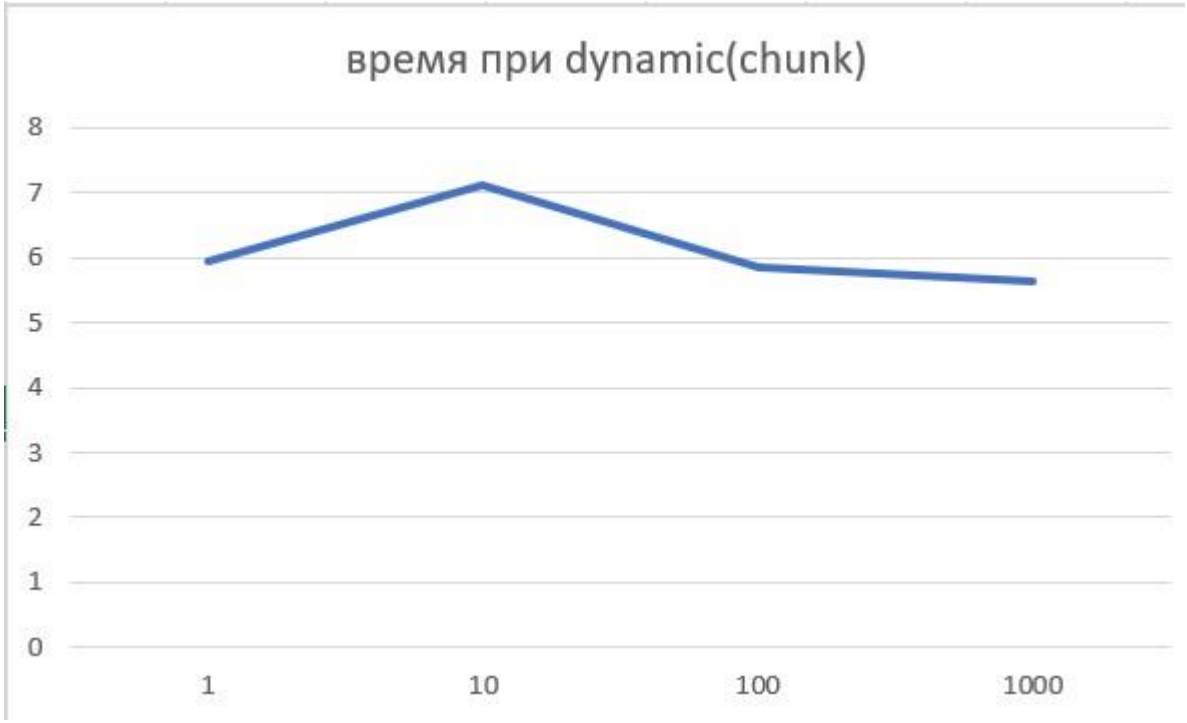
b при одинаковом значении числа потоков при различных параметрах (`#pragma omp for schedule (static,x)`). Потоков: 8

chunk_size	время	время1	время2	время3
1	10,4628	10,4745	10,3733	10,5405
10	9,5083	9,27534	9,31881	9,93077
100	8,9635	8,98452	8,97861	8,92741
1000	8,9284	8,83917	9,08222	8,8638



при одинаковом значении числа потоков при различных параметрах (`#pragma omp for schedule (dynamic,x)`). Потоков: 8

chunk_size	время	время1	время2	время3
1	5,941297	6,44773	5,62434	5,75182
10	7,128093	7,26087	7,26368	6,85973
100	5,856783	5,76336	5,80823	5,99876
1000	5,636017	5,6301	5,64183	5,63612



с с выключенным openmp и с включенным 1 потоком (по 3 попытки)

	время
0(без).1	35,0726
0(без).2	34,5407
0(без).3	35,6346
1 1	35,4306
1 2	37,8874
1 3	36,9835

5) Источники:

- https://www.openmp.org//wp-content/uploads/cspec20.pdf (документация openMP c\c++: конструкции openmp (стр 8, 11, 19) и schedule)
- https://ru.m.wikipedia.org/wiki/Xorshift (вики про алгоритм Xorshift)
- https://en.wikipedia.org/wiki/Xorshift (английская вики про Xorshift)
- https://habrcom.cdn.ampproject.org/v/s/habr.com/ru/amp/publications/574414/?amp_gsa=1&amp_js_v=a9&usqp=mq331AQIUAKwASCAAgM%3D#amp_tf=%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%3A%20%251%24s&aoh=17057017452690&referrer=https%3A%2F%2Fwww.google.com&ampshare=https%3A%2F%2Fhabr.com%2Fru%2Fcompanies%2Fvk%2Farticles%2F574414%2F (статья про оптимальные алгоритмы для случайных чисел)
- https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%82%D0%B0%D1%8D%D0%B4%D1%80 (октаэдр)
- https://learn.microsoft.com/ru-ru/cpp/parallel/openmp/2directives?view=msvc-170 (еще про schedule)
- https://studfile.net/preview/8896092/page:3/#9 (метод Монте-Карло)