Лабораторная работа №5

1.

ФИО	Завьялов Никита Аркадьевич
Группа	M3138
Название работы	OpenMP

2. https://github.com/skkv-itmo/itmo-comp-arch-2023-omp-NEKAfk

3. Работа была выполнена с использованием среды разработки VSCode, язык программирования C++, компилятор g++ (Debian 12.2.0-14) 12.2.0.

4. Тесты проводились с использованием процессора:
AMD Athlon Silver 3050U with Radeon Graphics

Input:
1000000000
(-2.37648 -1.28132 1.64637)
(2.94148 1.12998 0.266185)
(-0.818226 2.66201 1.49807)
Output: stdout: Time (2 thread(s)): 25188.6 ms
file: 36 36.002

5. Для распараллеливания комманд были использованы конструкции
#pragma omp parallel if(condition) num_threads(threads) – распараллеливает участок, если выполняется условие condition на количество потоков равное threads, если значение threads <= 0, то берется значение по умолчанию.

#pragma omp for schedule(kind, chunk_size) – разрезает цикл на блоки размера chunk_size и распределяет их в зависимости от значения kind. Static – блоки статически присваиваются потокам в команде по круговому принципу в порядке номера резьбы, dynamic – распределяет блоки в зависимости от времени освобождения потока, guided – распределяет блоки, уменьшая их размер экспоненциально до chunk_size.

#pragma omp single – выполнение данного блока комманд производится только одним потоком.

#pragma omp critical – блок выполняется одним потоком за раз.

6. Работа кода
На рис.1 представлен основной участок кода.
Сначала при помощи omp parallel обозначаем параллельный участок кода. В нем узнаем сколько всего потоков используется и создаем локальный счетчик lc.

Дальше нарезаем цикл при помощи omp for
Заметим, что нам не важно настоящее положение октаэдра, нам нужен только его размер. Также в силу симметрии можно найти объем только 1/8 части октаэдра.
То есть задача упрощается: мы можем взять 1/8 октаэдра и ограничить его кубом. Благодаря этому мы будем смотреть только на одно уравнение плоскости вместо 8, что ускорит нашу работу при больших N.

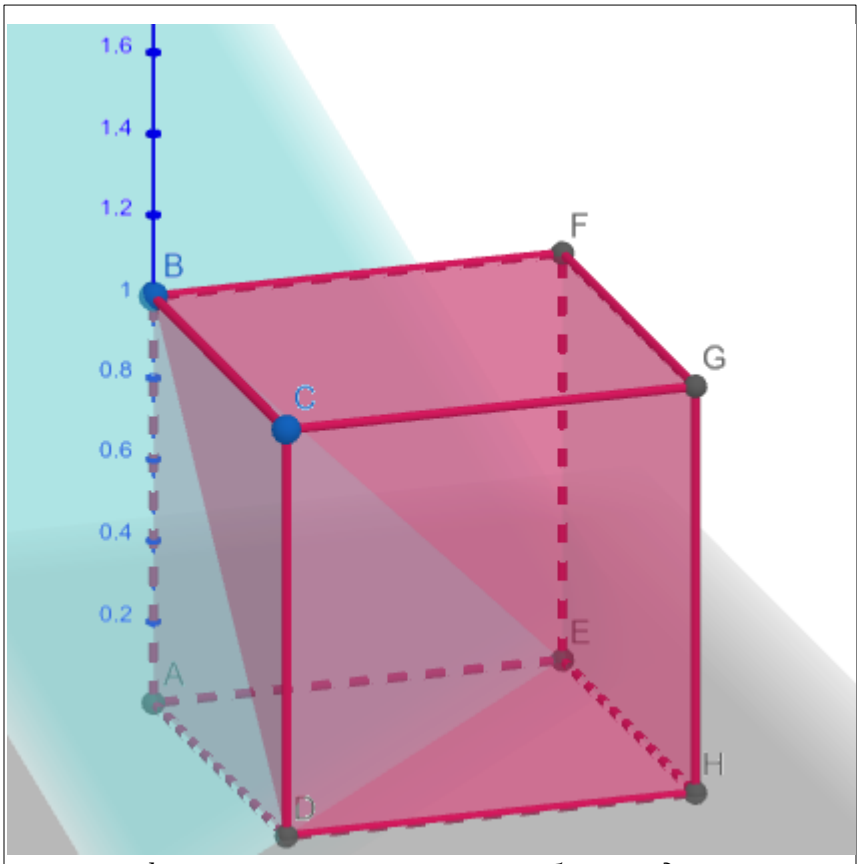


Рис.1 Графичесская интерпретация работы кода

```cpp
long long c = 0;
int mod = 1e9 + 7;
random_device rd;

double start = omp_get_wtime();
#pragma omp parallel if(threads != -1) num_threads(threads)
{
    #pragma omp single
    {
        thr = omp_get_num_threads();
    }
    long long lc = 0;
    uint32_t state = (rd() % mod) + 9;


    #pragma omp for schedule(static, N / thr)
    for (long long i = 0; i < N; i++) {
        double x = (double) xorshift(state) / numeric_limits<uint32_t>::max();
        double y = (double) xorshift(state) / numeric_limits<uint32_t>::max();
        double z = (double) xorshift(state) / numeric_limits<uint32_t>::max();
        if (underSurface(x, y, z)) {
            lc++;
        }
    }

    #pragma omp critical
    {
        c += lc;
    }
}
double end = omp_get_wtime();
```

```cpp
bool underSurface(double x, double y, double z) {
    return (x+y+z-1 <= 0);
}
```

```cpp
uint32_t xorshift(uint32_t& state) {
    uint32_t x = state;
    x ^= x << 13;
    x ^= x >> 5;
    x ^= x << 17;
    state = x;
    return x;
}
```

Пусть дана точка (x, y, z), принадлежащяя кубу со стороной d/2, где d – диагональ октаэдра. Чтобы она была ниже грани октаэдра(плоскости, пересекающей куб), надо чтобы выполнялось неравенство x + y + z <= d/2. Поделим на d/2: x/(d/2) + y/(d/2) + z/(d/2) <= 1. Если мы поместим октаэдр в центр координат и ограничим его часть кубом со стороной d/2, то x, y, z будут принимать значения [0, d/2] =>

x/(d/2), y/(d/2) и z/(d/2) будут принимать значения [0, 1]. Для генерации был использован генератор xorshift, код которого можно увидеть выше. В качесве сида было генерированно число, которой приводилось в ненулевое состояние, так как в противном случае генератор не будет работать.

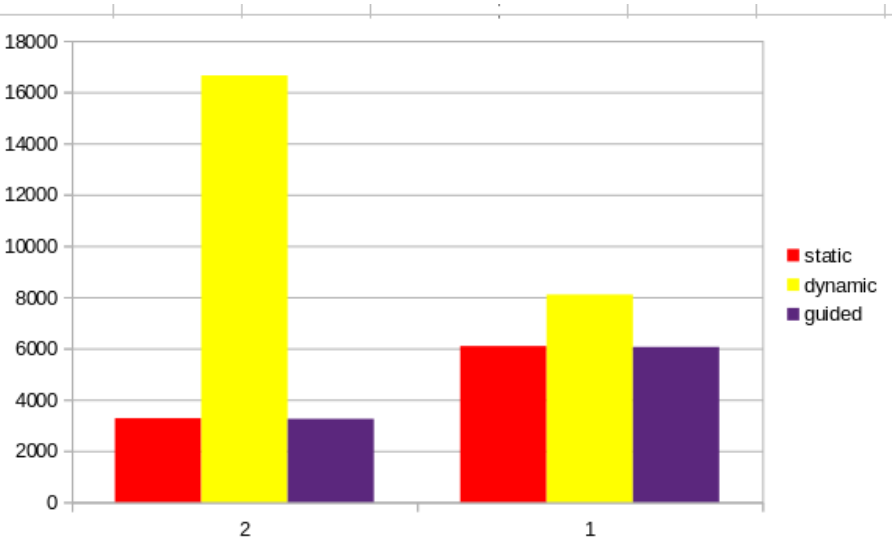
В конце используется omp critical, чтобы прибавить локальный счетчик к глобальному. Благодаря этому не случится проблем с одновременной записью в переменную, так как += не атомарная операция.

7. Результаты работы

При сравнении времени работы было взято N = 1e9. Значения усреднялись по 20 запускам.

1. Сравнение работы при разном количестве потоков с chunk_size взятым по умолчанию.

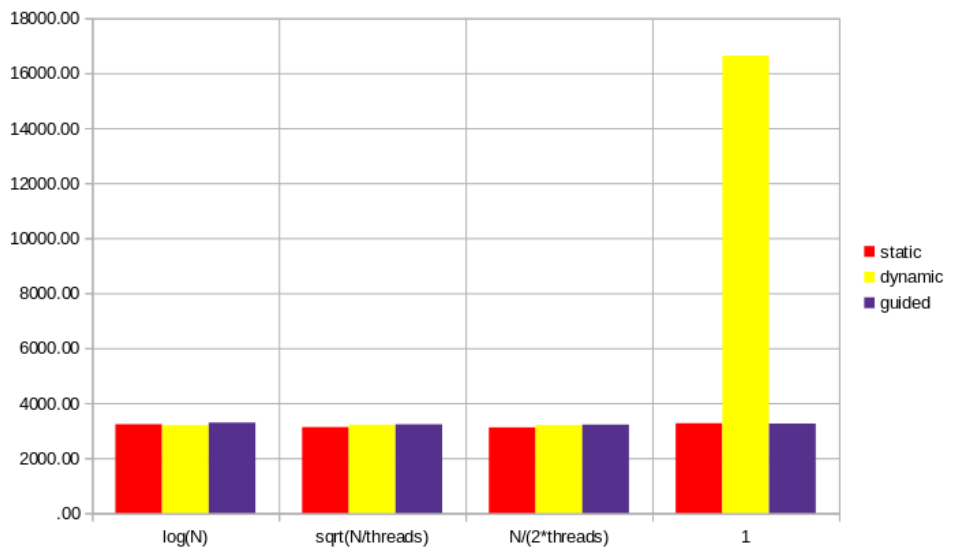
Schedule	Threads	Time
static	2	3271.56
static	1	6086.89
dynamic	2	16647.92
dynamic	1	8097.36
guided	2	3245.67
guided	1	6052.33



Static и guided работают примерно одинаково, а при использовании dynamic тратится лишнее время на распределение задач и поэтому при 2-ух потоках время выполнения даже больше, чем при одном.

2. Разные chunk_size(threads = 2)

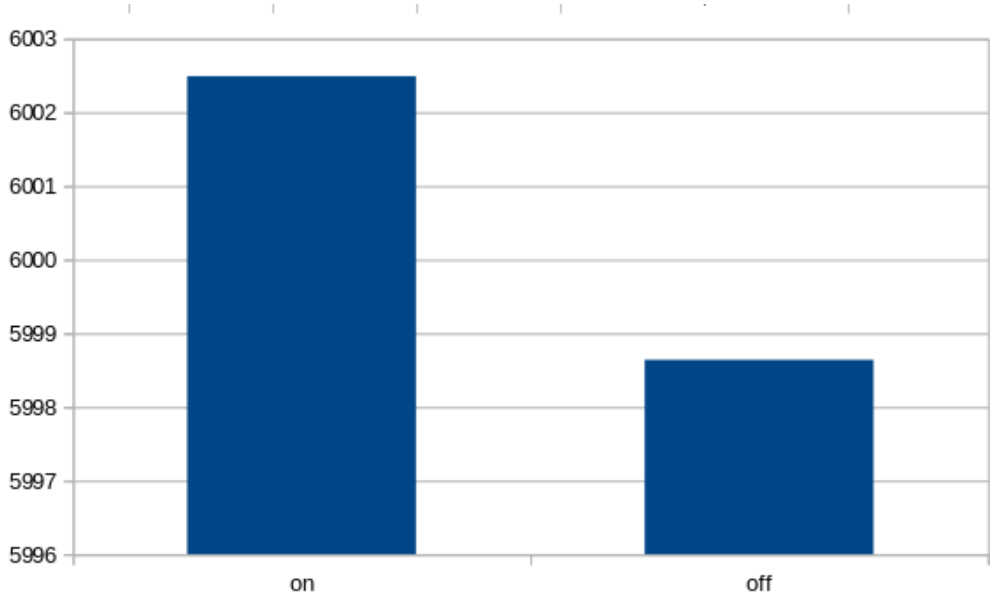
Schedule type	chunk_size	time
static	log(N)	3229.86
static	sqrt(N/threads)	3124.74
static	N/(2*threads)	3113.12
static	1	3266.01
dynamic	log(N)	3196.09
dynamic	sqrt(N/threads)	3213.43
dynamic	N/(2*threads)	3195.72
dynamic	1	16633.22
guided	log(N)	3289.76
guided	sqrt(N/threads)	3225.84
guided	N/(2*threads)	3213.91
guided	1	3251.62



Видно, что здесь аналогичная проблема с dynamic при chunk_size=1 очень много времмени уходит на распределение задач. Также расмотрены несколько случаев. log(N) рассмотрен, так как guided экспоненциально уменьшает размер блока, sqrt(N/threads), так как интересно посмотреть распределение при котором размер блоков равен их количеству, N/(2*threads) рассмотрен, так как захотелось рассмотреть линейное распределение.
Видно, что лучше всего работает static с chunk_size=N/(2*threads).

3. Сравнение однопоточного выполнения с использованием openmp и без его использования

openmp	Time
on	6002.49
off	5998.64



При использовании openmp с одним потоком происходит замедление выполнения программы. Вероятно тратится время на установление потока мастером и другие системные процессы.

8. Список литературы

1. OpenMP C and C++ Application Program Interface v2.0
2. https://en.wikipedia.org/wiki/Xorshift