

Лабораторная работа №5

1.

ФИО	Завьялов Никита Аркадьевич
Группа	М3138
Название работы	OpenMP

2. <https://github.com/NEKAfk/comp-ach>

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 команд 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 команд 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 .

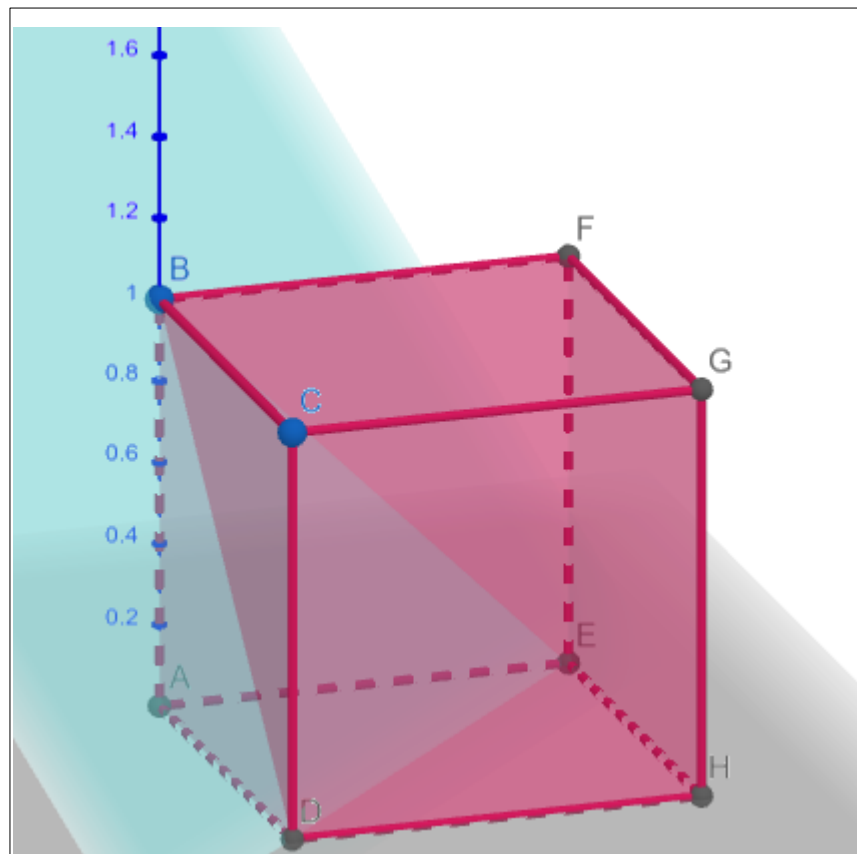


Рис.1 Графическая интерпретация работы кода

Рис. 2. Использование OpenMP

```

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();

```

```

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

```

You 2 weeks ago • new method

```

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

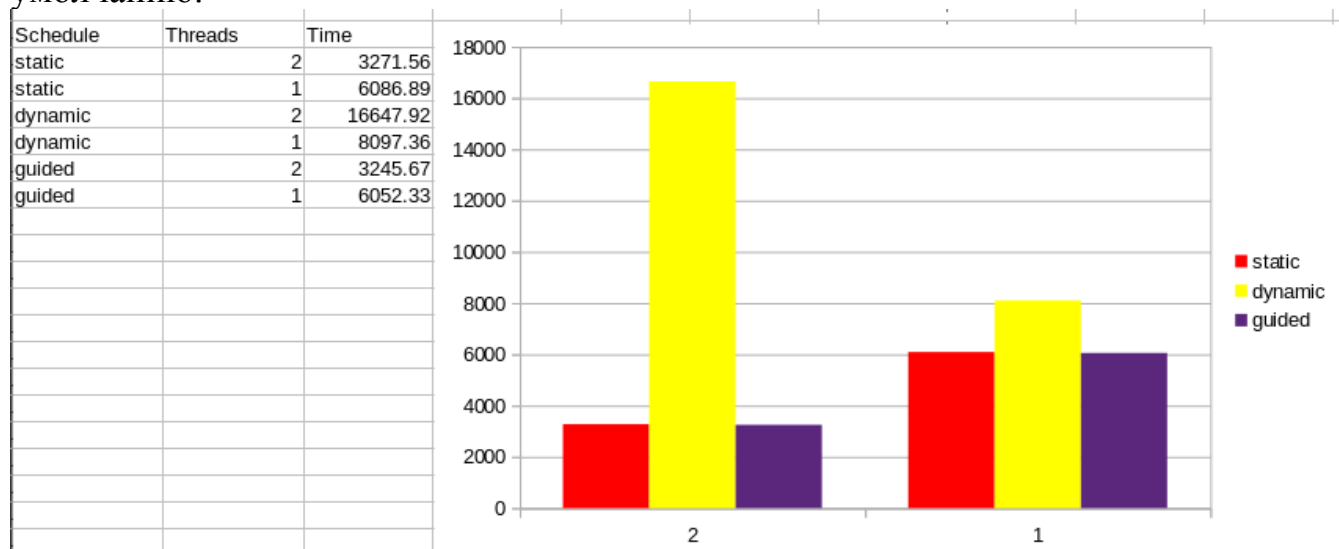
$x/(d/2)$, $y/(d/2)$ и $z/(d/2)$ будут принимать значения $[0, 1]$. Для генерации был использован генератор xorshift, код которого можно увидеть выше. В качестве seeds было генерировано число, которое приводило в ненулевое состояние, так как в противном случае генератор не будет работать.

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

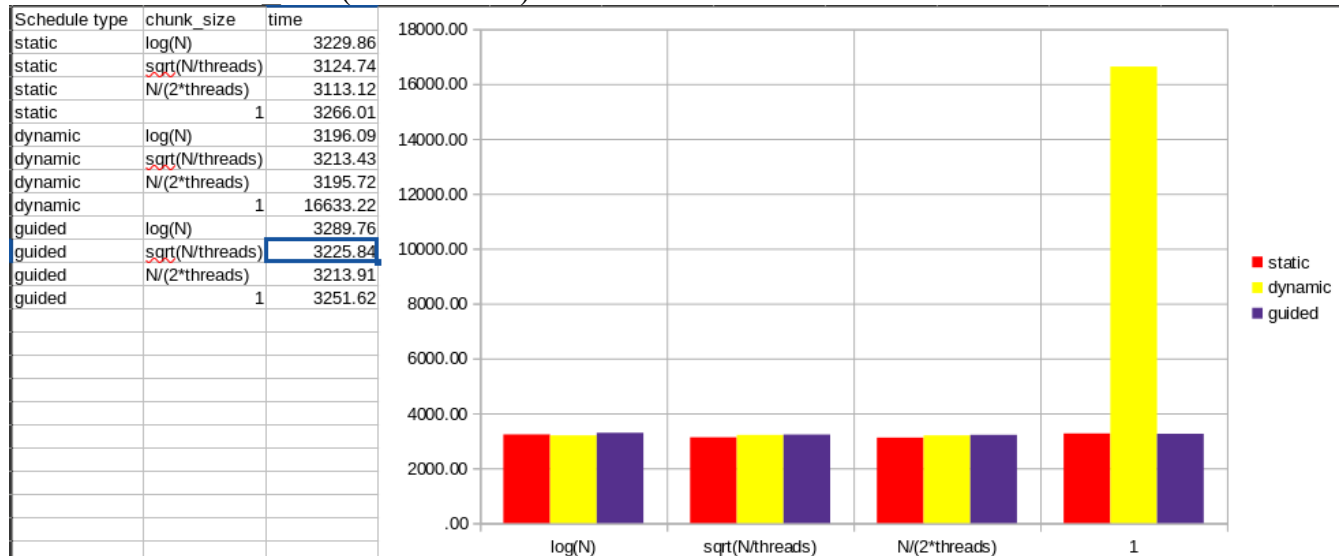
При сравнении времени работы было взято $N = 1e9$. Значения усреднялись по 20 запускам.

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



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.

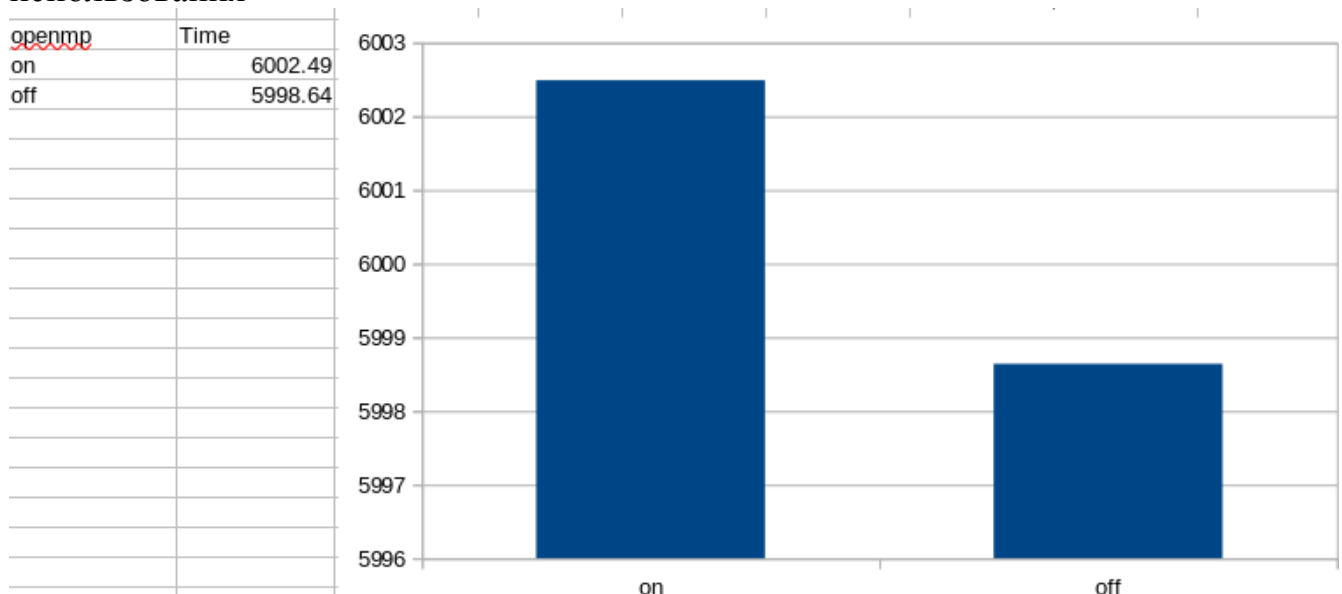
2. Разные chunk size(threads = 2)



Видно, что здесь аналогичная проблема с dynamic при $\text{chunk_size}=1$ очень много времени уходит на распределение задач. Также рассмотрены несколько случаев. $\log(N)$ рассмотрен, так как guided экспоненциально уменьшает размер блока, $\sqrt{N}/\text{threads}$, так как интересно посмотреть распределение при котором размер блоков равен их количеству, $N/(2*\text{threads})$ рассмотрен, так как захотелось рассмотреть линейное распределение.

Видно, что лучше всего работает static с $\text{chunk_size}=N/(2*\text{threads})$.

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



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>