Лабораторная работа №5
OpenMP

Халили Алина Ниязовна M3136

https://github.com/skkv-itmo2/itmo-comp-arch-2023-omp-AlinaK12345

Язык: C++ 20, Visual Studio 2019

Ноутбук: 4 ядра, 8 – логических процессоров

1) Задача:
   Аналитическое решение:
   -Из любых 2 точек октаэдра, хотя бы 2 лежат на одной грани, и так как фигура правильная, то мы можем найти длину ребра – наименьшее расстояние из всех, которые можно получить между 3 точками.
   -По длине ребра правильный октаэдр восстанавливается однозначно. Формула объёма V = (a*a*a*(2)**0,5) / 3

   Метод Монте-Карло:
   -Зная ребро, мысленно сдвинем фигуру в удобные координаты(так что она стала симметрично относительно (0,0,0) и есть ребра || осям. Тогда будем генерировать точки в прямоугольнике (-1/2**0.5 * a, -1/2**0.5 * a, -1/2**0.5 * a) (1/2**0.5 * a, 1/2**0.5 * a, 1/2**0.5 * a)
   (x, y, z) принадлежит фигуре, если |x| + |y| + |z| <= (радиуса, т.е 1/2**0.5 * a)
   -генерируем много точек (n) , считаем count_in – сколько попали внутрь фигуры
   Объём будет равен = (in_count / n) * (a*2**0.5)**3
   (так как равновероятно выпадали точки, значит соотношения объёмов = соотношению попаданий точек при больших n)

   Алгоритм генерации – **Линейный** конгруэнтный метод(LCG)

   $X_{i+1} = (aX_i + c) \% m$
   Где важно подобрать a, m, c так, что бы длина периода была как можно дольше:

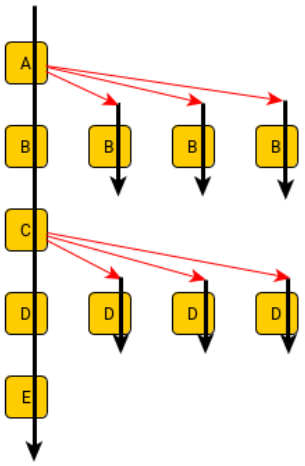
1. Числа c и m взаимно простые;
2. a-1 кратно p для каждого простого p, являющегося делителем m;
3. Если m кратно 4, то и a-1 должно быть кратно 4.

   Это необходимые требования для этого.
   Я воспользовалась уже готовыми, подобранными числами.
   (https://habr.com/ru/companies/vk/articles/574414/)
   Потом я перевела полученное псевдослучайное число в нужный диапазон и использовала для координат.
   (все считаю в float)

С помощью библиотеки OpenMP можно распараллелить программу, что уменьшит время вычислений.

```
#include "omp.h"

int main() {
  // A - single thread
#pragma omp parallel
  {
    // B - many threads
  }
  // C - single thread
#pragma omp parallel
  {
    // D - many threads
  }
  // E - single thread
}
```



(https://pro-prof.com/archives/4335?ysclid=lrm3357cz7820110413)

Используя #parallel можно управлять потоками.

Я использовала `#pragma omp parallel for` которое само распределяет итерации цикла между потоками. Но надо следить за общими переменными, поэтому для каждого потока я создала локальную, и в конце с помощью `#pragma omp atomic` (потоки могут обращаться только по очереди) их сложила в общую.

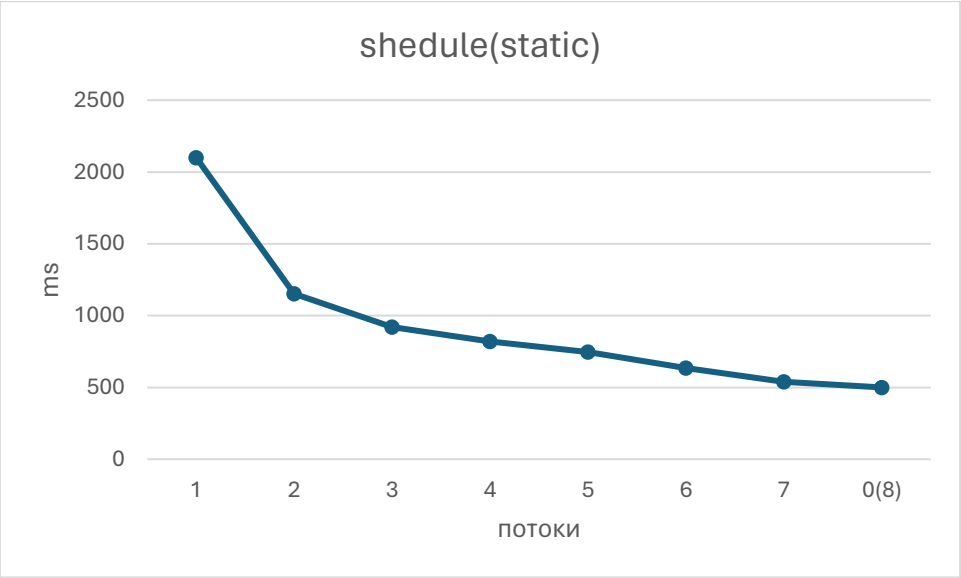
При использовании schedule(static) итерации цикла будут поровну поделены между потоками

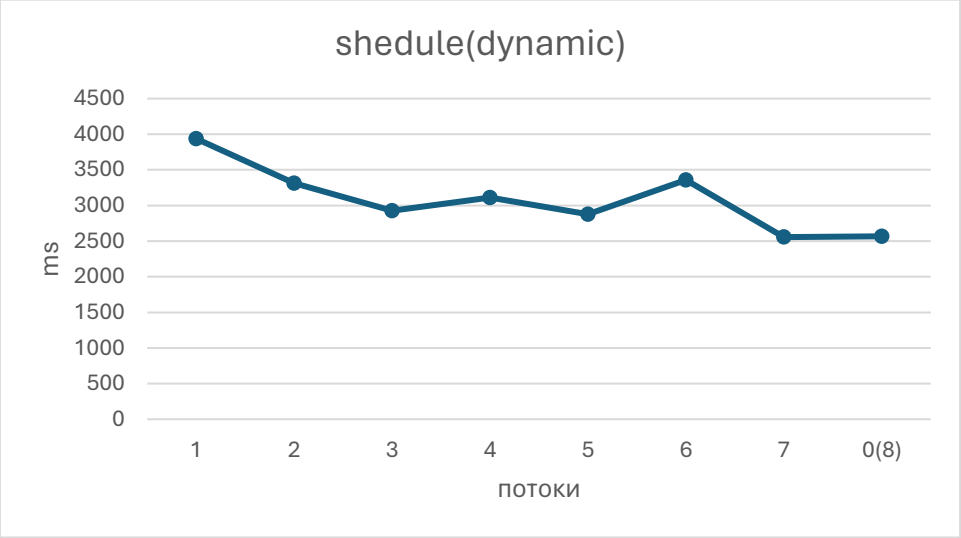
schedule(static, ch_size) блочно-циклическое распределение итераций.
Каждый поток получает заданное число итераций в начале цикла = ch_size

schedule(dynamic, ch_size)– динамическое планирование. По умолчанию параметр опции равен 1. Каждый поток получает заданное число итераций, выполняет их и запрашивает новую порцию.

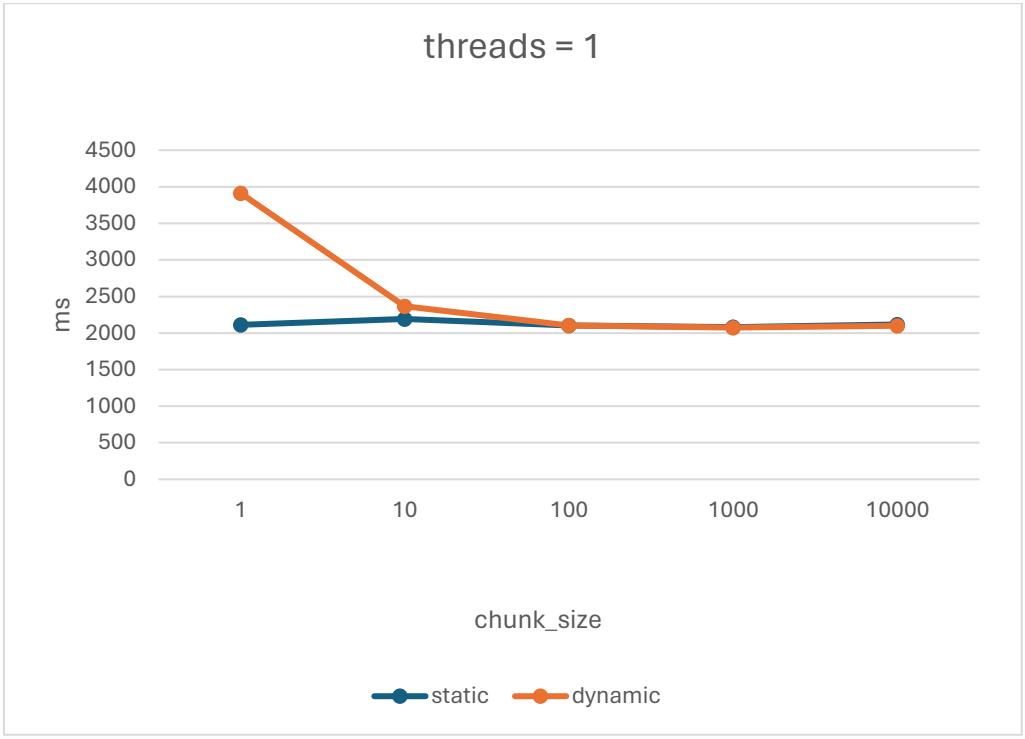
Для точности я брала среднее по трем измерениям, и получила следующие графики:

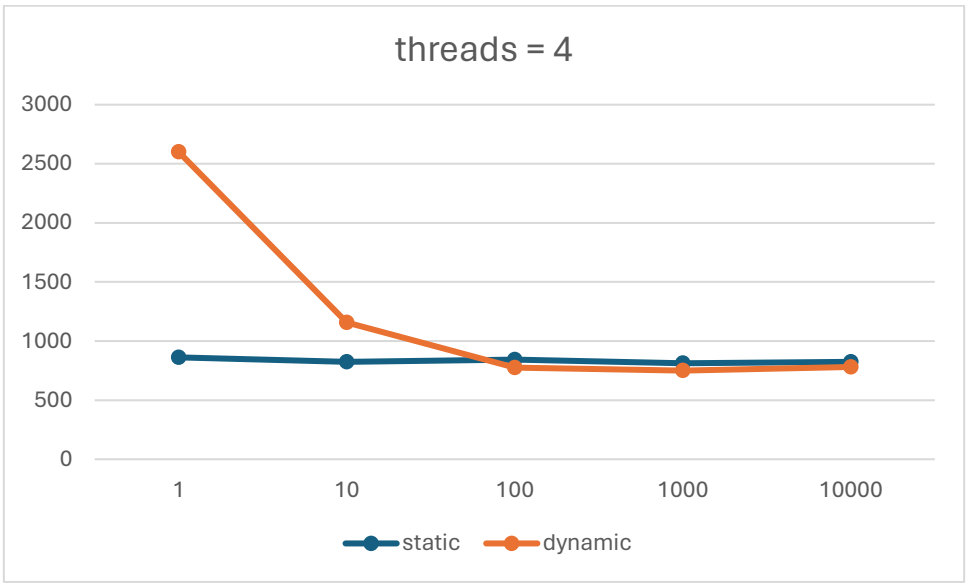
1) Сравнение времени работы при разделение на разное число потоков

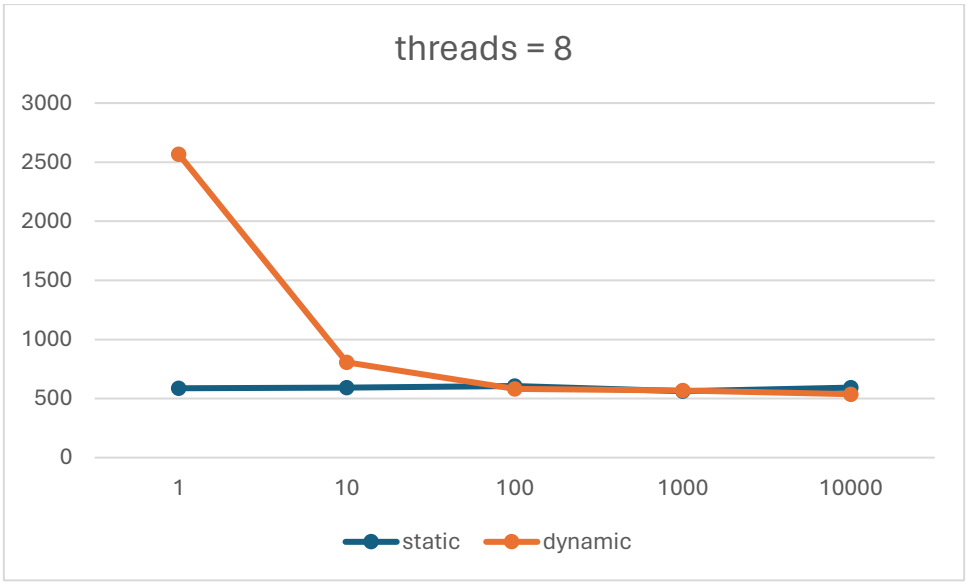




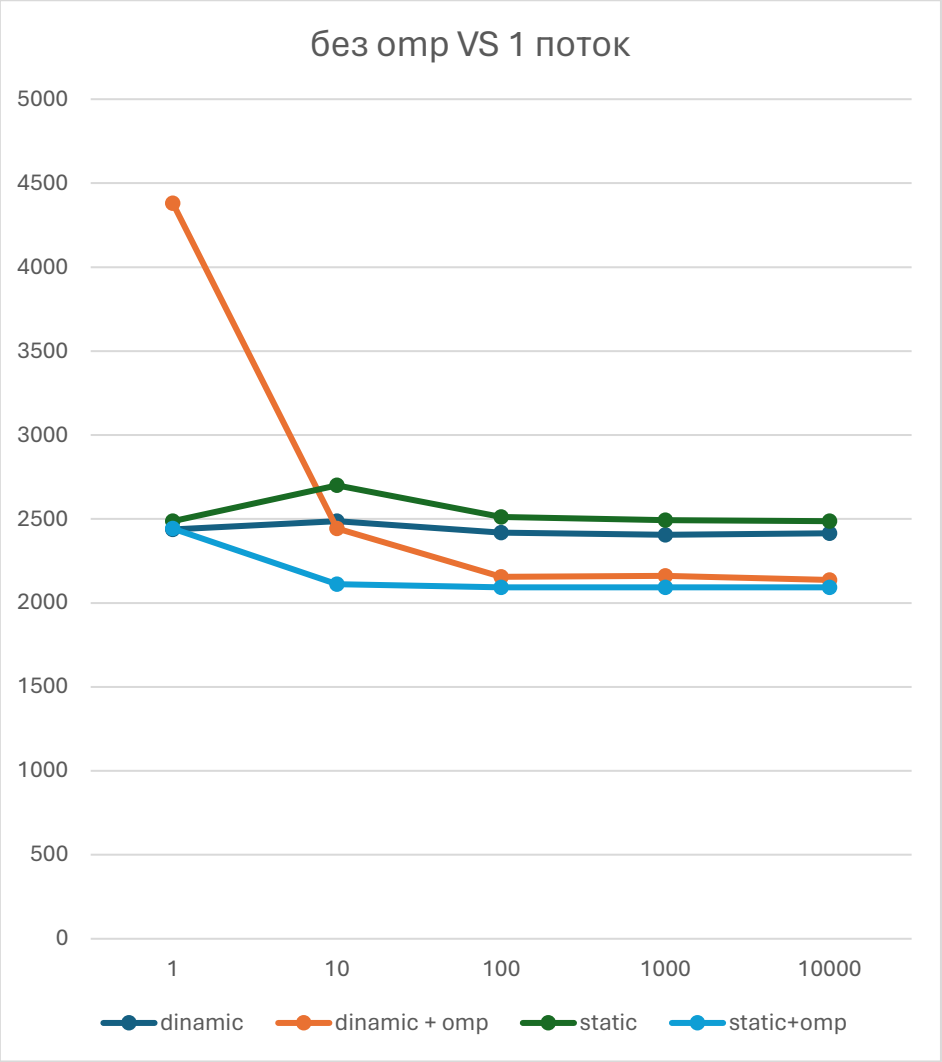
2) Сравнение static(chank_size) и dynamic(chank_size) при одинаковом количестве потоков


threads = 1
4500
4000
3500
3000
2500
2000
1500
1000
500
0
ms
1
10
100
1000
10000
chunk_size
static
dynamic


threads = 4
3000
2500
2000
1500
1000
500
0
1
10
100
1000
10000
static
dynamic


threads = 8
3000
2500
2000
1500
1000
500
0
1
10
100
1000
10000
static
dynamic

2)



Из графиков видно что делить на потоки выгоднее по времени, причем если делить на 8 потоков – то время уменьшается в среднем в 4 раза. Так же самые быстрый это режим static

(Все на примере из условия работы)

Лучшее :

```
Time( 8 thread(s)) : 4875.03 ms
```

schedule(static)

потоки	Время, с				
-1	25.2500	25.9375	23.7500	23.5625	
0	5.5000	4.7500	5.0000	4.875	
1	21.0625	20.8125	20.8750	20.7500	
2	11.4375	11.9375	12.0625	12.0000	
3	8.8125	9.6875	9.0625	9.4375	
4	8.3125	8.0625	8.1875	8.5	
5	7.6875	7.375	7.5	7.25	
6	6.25	6.3125	6.375	6.4375	
7	5.4375	5.375	5.375	5.4375	

Код:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <omp.h>
#include <cmath>
#include <algorithm>
#include <iostream>
#include <string>
using namespace std;

float distance(float* a, float* b) {
   return std::sqrt(std::pow(a[0] - b[0], 2) + std::pow(a[1] - b[1], 2) + std::pow(a[2] - b[2], 2));
}

const float eps = 0.000001;

bool equal_d(float a, float b) {
   return std::abs(a - b) < eps;
}


long long r_max = 4294967296;
float random_f(float a, long seed) {
```

```
    float f = (seed % r_max) * (2 * (a)) / (r_max - 1) - a;

    return f;
}
long a1 = 2147483647;
const long m1 = 16807;
long new_seed(long seed) {

    seed = 214013 * seed + 2531011;
    seed ^= seed >> 15;

    return seed ;

}

int main(int argc, char** argv)
{
    if (argc != 4) {
        std::cerr << "Not correct input" << std::endl;
        exit(1);
    }

    FILE* fin = fopen(argv[2], "rb");
    if (fin == NULL) {
        std::cerr << "No file" << std::endl;
        exit(1);
    }
    long long int n;
    //cout << "ok";

    int threads = std::stoi(argv[1]);//to number
```

```
float coords[3][3];
float x, y, z;
fscanf(fin, "%lld\n", &n);
//cout << n << '\n';

for (int i = 0; i < 3; i++) {
   fscanf(fin, "(%f %f %f)\n", &x, &y, &z);
   coords[i][0] = x;
   coords[i][1] = y;
   coords[i][2] = z;
}
//cout << coords[0][0]<<" " << coords[0][1];
fclose(fin);
//cout << "ok" << '\n';
//analitic:
float l1 = distance(coords[0], coords[1]);
float l2 = distance(coords[2], coords[1]);
//float l3 = distance(coords[0], coords[2]);
float a = std::min(l1, l2);
float ans1 = a*a*a * std::sqrt(2) / 3;
//cout << l1 << '\n';
//

float a_2 = std::sqrt(2) * a / 2;

float tstart = omp_get_wtime();

//cout << tstart << '\n';
int sum = 0;
int count_in = 0;
int inside = 1;
```

```
  int output_threads = 0;

  if (threads == -1) {
    unsigned long seed = 1;
    float x1, y1, z1;
    for (int i = 0; i < n; ++i)
    {
      x1 = random_f(a_2, seed);
      seed = new_seed(seed);
      //cout << i << " " << seed << ' ' << x1 << '\n';
      //cout << omp_get_thread_num() << '\n';
      y1 = random_f(a_2, seed);
      seed = new_seed(seed);
      z1 = random_f(a_2, seed);
      seed = new_seed(seed);
      if (abs(x1) + abs(y1) + abs(z1) <= a_2) {
        count_in += 1;
      }

    }
  }
  else {
#ifdef _OPENMP
    if (threads > 0) {
      omp_set_num_threads(threads); // count of threads
    }
#endif

#pragma omp parallel
    {
#ifdef _OPENMP
      unsigned long seed = omp_get_thread_num() + 1;
```

```
        //cout << omp_get_thread_num()<<'\n';


        int local_in = 0;
        float x1, y1, z1;
#pragma omp for nowait schedule(static)
        for (int i = 0; i < n; ++i)
        {
          //cout << seed << '\n';
          x1 = random_f(a_2, seed);
          seed = new_seed(seed);
          //cout << i << " " << seed << ' ' << x1 << '\n';
          //cout << omp_get_thread_num() << '\n';
          y1 = random_f(a_2, seed);
          seed = new_seed(seed);
          z1 = random_f(a_2, seed);
          seed = new_seed(seed);
          if (abs(x1) + abs(y1) + abs(z1) <= a_2) {
            local_in += 1;
          }


        }
#pragma omp atomic
        count_in += local_in;
        output_threads += 1;


#endif
      }
    }
    float ans2 = a * (2 * sqrt(2) * a * a* count_in) / n;


    float tend = omp_get_wtime();
```

```
    FILE* fout = fopen(argv[3], "w");
    fprintf(fout, "%g %g\n", ans1, ans2);
    fclose(fout);
    //cout << ans1<<" "<<ans2 << '\n';
    //printf("Time (sec): %f\n", tend - tstart);

    printf("Time(% i thread(s)) : %g ms\n", output_threads, (tend - tstart)*100);
    return 0;
}
```