Мини-отчёт

Рудаков Максим, M3137

20 января 2024 г.

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

Инструментарий: clang version 17.0.6.

1 Результат работы написанной программы

stdout:

```
Time (16 thread(s)): 1318.21 ms
```

output.txt:

```
36 36
```

ЦП:

```
13th Gen Intel(R) Core(TM) i5-13500H

Базовая скорость: 2,60 ГГц
Сокетов: 1
Ядра: 12
Логических процессоров: 16
Виртуализация: Включено
Кэш L1: 1,1 МБ
Кэш L2: 9,0 МБ
Кэш L3: 18,0 МБ
```

2 OpenMP

В работе я использовал одну конструкцию:

```
#pragma parallel for
```

Конструкция позволяет распараллеливать выполнение цикла в программе. Цикл разбивается на части, и каждая часть выполняется параллельно на своём потоке.

OpenMP автоматически управляет созданием и уничтожением потоков, а также распределением работы между ними. Количество потоков, которые будут использованы для выполнения

цикла, а также параметры распределения, определяются константами и специальными аргументами конструкции, например, schedule. Эта настройка позваоляет указать один из видов планирования разделения работы между потоками. В лабораторной я подробно исследовал влияние на время программы установки таких параметров, как: static, dynamic и guided.

При использовании параметра static итерации цикла разделяются поровну между потоками, делясь на порции по chunk_size итераций.

При использовании dynamic итерации делятся на порции по chunk_size итераций, а затем по порядку даются первому свободному потоку. Как только поток обрабатывает порцию, он делает запрос и получает следующую. Хочу отметить, что этот параметр крайне неэффективен при использовании малых chunk_size (как ясно видно на графиках), так как после каждой итерации происходит запрос, что, конечно, очень тормозит работу. Так, при размере chunk_size = 1 (по умолчанию) время работы программы на 16 потоках сравнится со временем без использования OpenMP вообще.

При использовании guided порции делятся на уменьшающиеся порции. То есть если задан chunk_size, то порции будут уменьшаться от размера, примерно равного N / number_of_threads до chunk_size.

3 Описание работы кода

В коде используются простые вспомогательные структуры Point и Vector, которые я использовал для геометричесих вычислений.

Программа получает аргументы командной строки и парсит их. Первым делом программа переводит строку, содержащую кол-во потоков в число.

Затем программа открывает файл и читает из него кол-во точек и точки, задающие октаэдр.

Здесь программа засекает время.

Дальше программа вычисляет положение октаэдра и его высоту. Для того, чтобы тремя точками единственным образом задать октаэдр, 3 этих точки не должны лежать на одной его стороне. Но тогда две точки будут лежать на симметричных относительно центра фигуры вершинах.

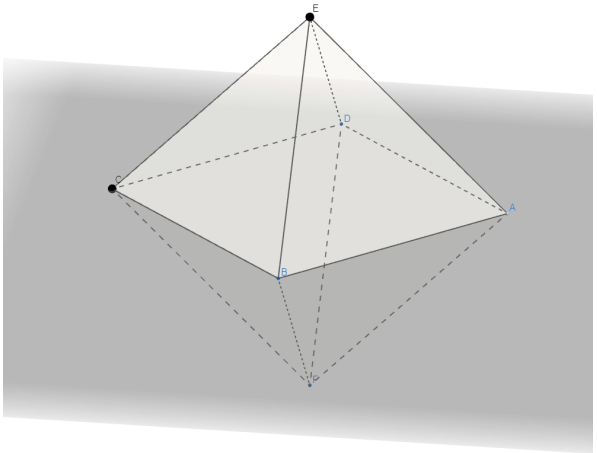


Рис. 1: иллюстрация к теореме

Теорема.

Если мы взяли противоположные точки, мы уже победили. Иначе мы берем две точки, лежащие на 1 ребре (других вариантов нет) и фиксируем их (остальные варианты будут приводимы к этому путём поворотов и отражений). Теперь мы можем взять две из четырёх оставшихся точек в качестве третьей, так как D и B будут лежать на одной грани окаэдра и не будут задавать его однозначно. Взяв точку A, она же будет противоположна C. F будет противоположна E. Так мы в любом случае получим две противоположные точки (чтд).

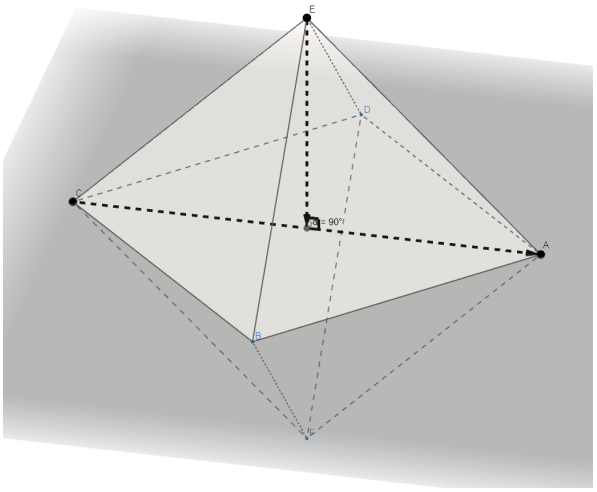


Рис. 2: https://www.geogebra.org/calculator/sxzxgejm

Отталкиваясь от этого, программа проверяет каждые пары точек на то, на противоположных ли они вершинах. Для этого она берет третью точку и строит вектор к середине прямой, проходящей через первые две точки. Если вектор получился перпендикуляром к прямой (вычисляет скалярное произведение), то эти точки симметричны относительно центра.

После этого код разветвляется. Я не понял, как нужно запустить программу без omp, так что просто скопировал цикл и вставил его без конструкций параллельного исполнения.

Итак, если кол-во потоков -1, выполняется один цикл, иначе устанавливается нужное кол-во потоков и выполняется второй цикл. Вместо использования #pragma atomic я решил сделать массив значений по количеству потоков и записывать сумму подходящих точек для каждого потока отдельно, а после цикла проходить по массиву и суммировать значения. Для генерации случайных чисел я использую самописную PRNG функцию, которая по числу генерирует следующее случайное число. Она вызывается сначала от счётчика цикла i, а затем от полученного на предыдущем шаге числа для генерации каждой новой координаты точки.

Далее программа считает объем по полученному соотношению точек в кубе и объем окаэдра по формуле из интернета (Источник 2), найдя сторону через высоту домножением на sqrt(2).

Здесь программа фиксирует время окончания работы.

После этого программа выводит ответ в файл output и информацию о потоках и времени работы.

4 Тестирование

Ох и устал я тестировать свою программу. Результаты замеров приведены в google-таблице и доступны для ознакомления. Каждый замер повторялся 5 раз для уточнения результата (без

omp - 15). В графиках используются средние значения по каждой группе замеров. Графика времени "без omp"я строить не стал, так как график из одной точки не имеет смысла. Как втиснуть это значения на остальные графики я не придумал. Вместо этого для наглядности предлагается условное форматирование в таблице.

https://docs.google.com/spreadsheets/d/1n3GqCg6AHcrjLawLq-XgwStrFepbWOy5jdJcrpHhNFI/edit?usp=sharing

Для тестирования я написал несложный скрипт на Питоне (Приложение 1). Во время всех замеров из приложений на компьютере был запущен только PyCharm и Excel. Графики приведены в Приложении 2.

5 Ссылки на источники

1. Технология параллельного программирования OpenMP - вся информация по OpenMP, за исключением нескольких посиков в интернете. Особо перечитываемы были страницы 13-20, 35-49.
https://cs.petrsu.ru/ kulakov/courses/parallel/lect/openmp.pdf

2. Объем октаэдра.
https://studwork.ru/spravochnik/matematika/obemy-figur/obem-oktaedra

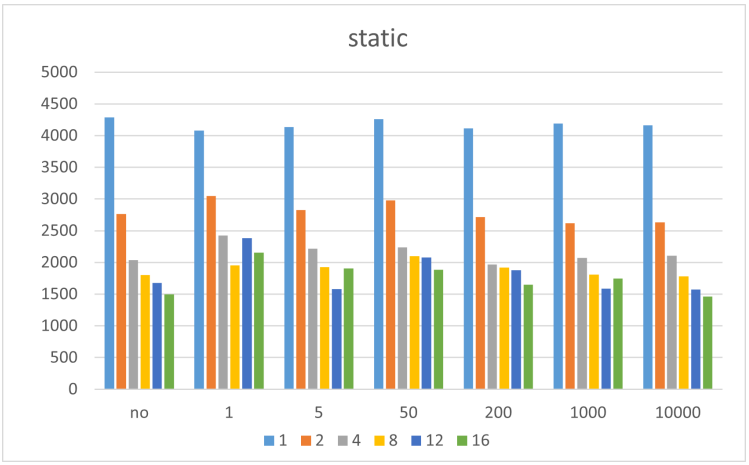
3. Google-таблица с замерами. К сожалению, графики остались у меня локально в Excel.
https://docs.google.com/spreadsheets/d/1n3GqCg6AHcrjLawLq-XgwStrFepbWOy5jdJcrpHhNFI/edit

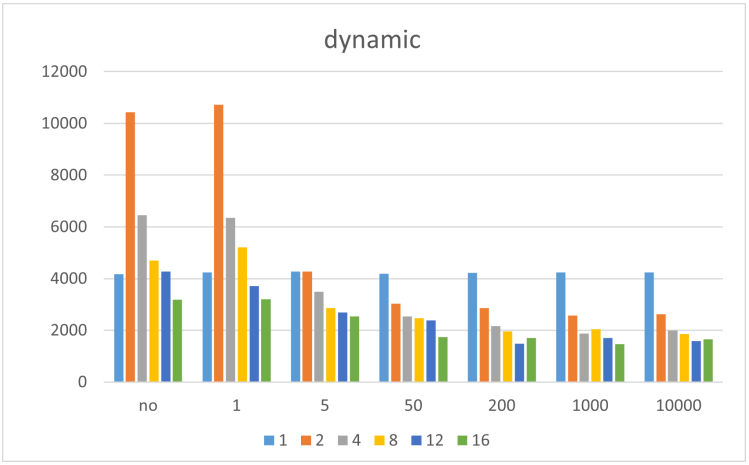
6 Приложение 1

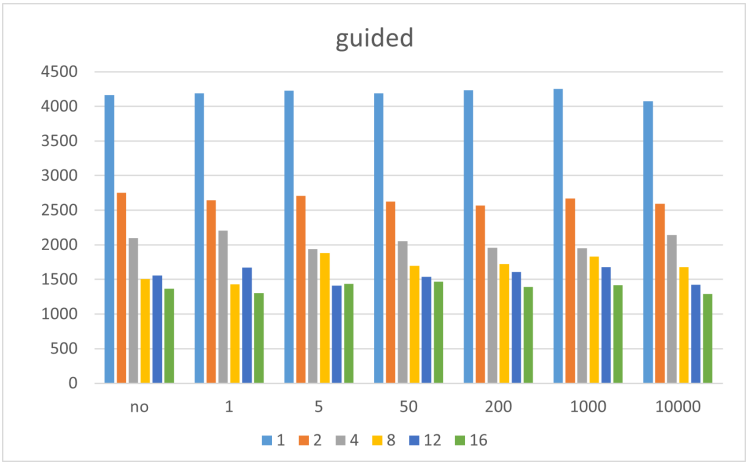
```
import os
from time import sleep

os.system('clang++ -std=c++20 -D _CRT_SECURE_NO_WARNINGS -D _USE_MATH_DEFINES '
          '-O2 -fopenmp normal.cpp -o normal.exe')

for chunk_size in [1, 5, 50, 200, 1000, 10000, 100000]:
    for thread_num in [1, 2, 4, 8, 12, 16]:
        b = []
        for _ in range(5):
            os.system(f'.\\normal.exe {thread_num} input.txt output.txt {chunk_size}')

            while not os.path.exists('stat.txt'):
                sleep(0.2)

            with open('stat.txt', 'r') as file:
                b.append(file.read().replace('.', ','))
            os.remove('stat.txt')
        print(*b, sep='\t')
```

7 Приложение 2

Графики зависимости времени от числа потоков при фикс. chunk_size (группировка по chunk_size):







Графики зависимости времени от парметра chunk_size при фикс. кол-ве потоков (группировка по кол-ву потоков):