Практическая работа № А-07

Методы высокопроизводительных вычислений

Альшаеб Басель, группа 24.M71-mm

04.12.2024

# 1- Задание

Написать программу вычисления выражения:

d = <B4 x, y>/<x, y> - <B3 x, y>/<x, y>,

где B – квадратная плотная матрица, элементы которой имеют тип double, элементы матрицы

задаются с помощью генератора псевдослучайных чисел, x и y – векторы (элементы задаются

псевдослучайными числами), <,> - скалярное произведение.

Распараллелить эту программу с помощью OpenMP (parallel, task).

Исследовать зависимость масштабируемости параллельной версии программы от ее

вычислительной трудоемкости (размера матриц).

Проверить корректность параллельной версии.

Проверка закона Амдала. Построить зависимость ускорение:число потоков для заданного примера.

# 2- Описание программно-аппаратной конфигурации

OS: Ubuntu 22.04.4 LTS

Kernel: 5.15.0-122-generic

Processer: Intel® Xeon® E-2136 CPU @ 3.30GHz

Cores: 12

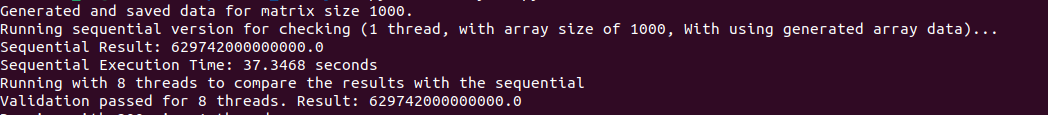
Architecture: x86\_64

Memory: 62 G

Disk Space: 6.3 G

# 3- Проверка правильности параллельного решения

Для проверки исправления я сгенерировал фиксированное количество данных для размера массива 1000, а затем запустил программу для 1 потока и 8 потоков и сравнил результат. Я приложу изображения для сравнения результата из логов.



# 4- Исследование зависимости ускорения от количества потоков и размера массива

Параллельный раздел посвящен распределению скалярного произведения и каждой операции умножения, и это будет распределено между потоками.

Каждый раз, когда мы добавляем больше потоков, мы можем распределить операции по разным потокам. В результате это сделает программу быстрее до определенной точки, когда размер массива будет соответствовать количеству потоков.

Эксперимент проводится на трех размерах массива: [200, 500, 1000] и потоках [1, 2, 4, 8, 16, 32, 64]

Array Size: 200x200

|  |  |  |
| --- | --- | --- |
| Threads | Execution Time | Speedup |
| 1 | 0.264327 | 1.0 |
| 2 | 0.132432 | 2.0 |
| 4 | 0.0681759 | 3.88 |
| 8 | 0.0674665 | 3.92 |
| 16 | 0.065614 | 4.03 |
| 32 | 0.0556225 | 4.75 |
| 64 | 0.0580132 | 4.56 |

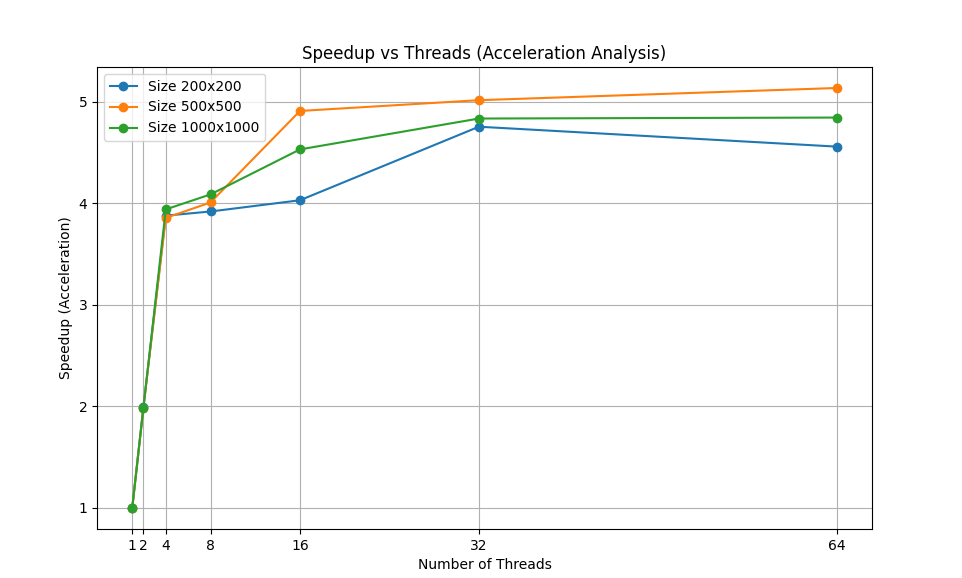
Array Size: 500

|  |  |  |
| --- | --- | --- |
| Threads | Execution Time | Speedup |
| 1 | 4.43893 | 1.0 |
| 2 | 2.23291 | 1.99 |
| 4 | 1.15158 | 3.85 |
| 8 | 1.10781 | 4.01 |
| 16 | 0.904546 | 4.91 |
| 32 | 0.885449 | 5.01 |
| 64 | 0.864754 | 5.13 |

Array Size: 1000

|  |  |  |
| --- | --- | --- |
| Threads | Execution Time | Speedup |
| 1 | 37.5727 | 1.0 |
| 2 | 18.9306 | 1.98 |
| 4 | 9.53971 | 3.94 |
| 8 | 9.19555 | 4.09 |
| 16 | 8.29615 | 4.53 |
| 32 | 7.77459 | 4.83 |
| 64 | 7.75921 | 4.84 |

Ниже вы можете увидеть график результатов по размеру, ускорению и потокам.



# 5- Объяснение результатов и выводы

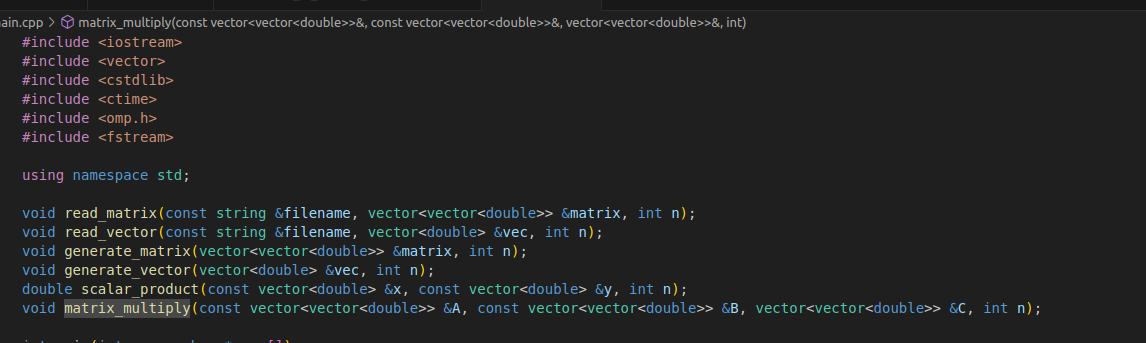
Из графика мы видим, что ускорение увеличивается с увеличением потоков, а размер массива также играет роль. При увеличении размера ускорение также меняется в зависимости от размера, потому что это квадратный массив.

- Из этого графика можно сказать, что при достижении 5-кратного ускорения ускорение останавливается и начинает сглаживаться

- Во всех случаях мы видим, что ускорение стабилизируется на 64 потоках.

# 6- Исходный код

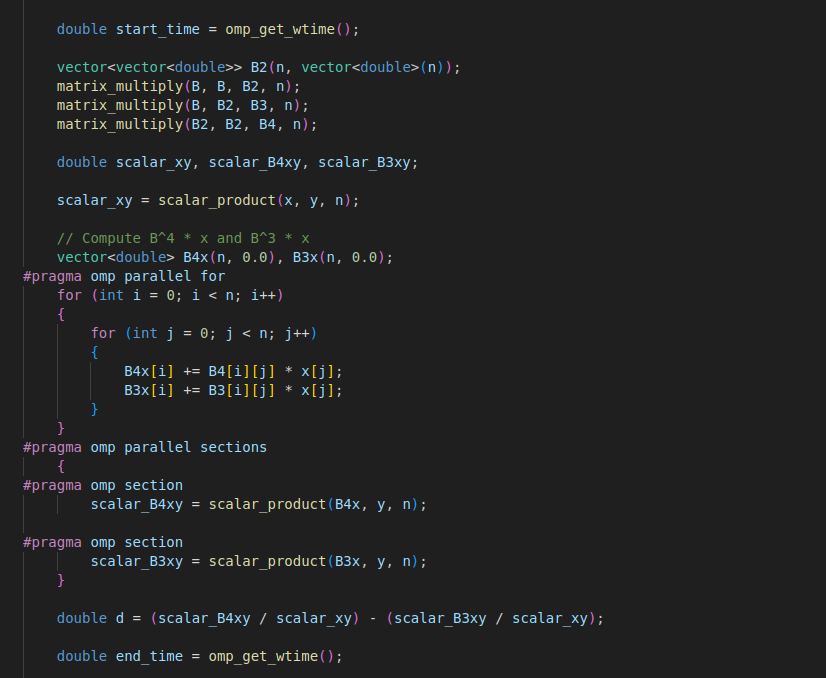
- Определение основных функций



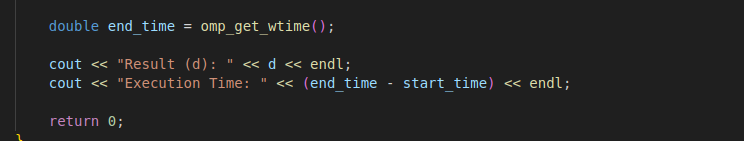
- определение основных переменных и массивов. и добавление параметров, которые помогут нам провести эксперимент. Итак, у нас есть "use-files", который говорит нам использовать предопределенный массив, "threads" для указания количества потоков, "size" для установки размера массива.



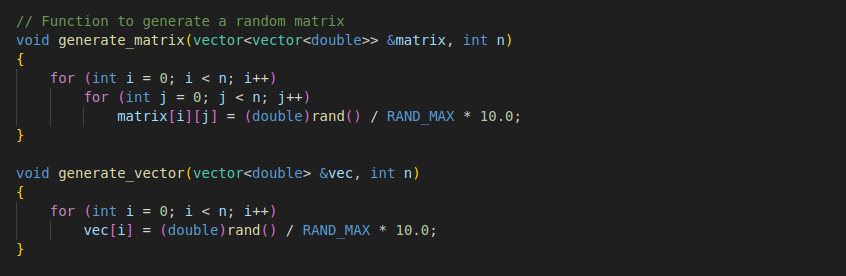
- Основная логическая область, где: 1- мы выполняем умножение массивов для создания B^2 и других, 2- мы вычисляем скалярное произведение. Для вычисления умножения векторов и массивов мы использовали «pragma omp parallel for», а для скаляра между вектором и массивом мы использовали «omp sections»



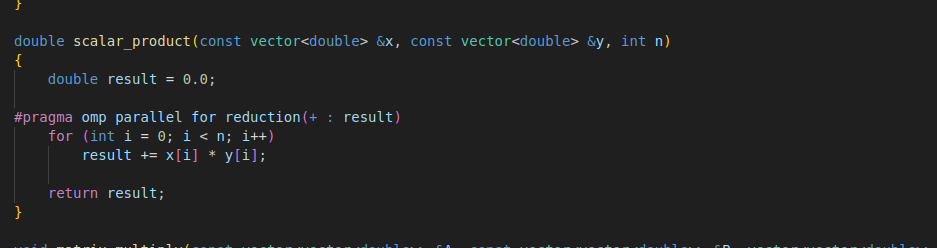
- Печать результата



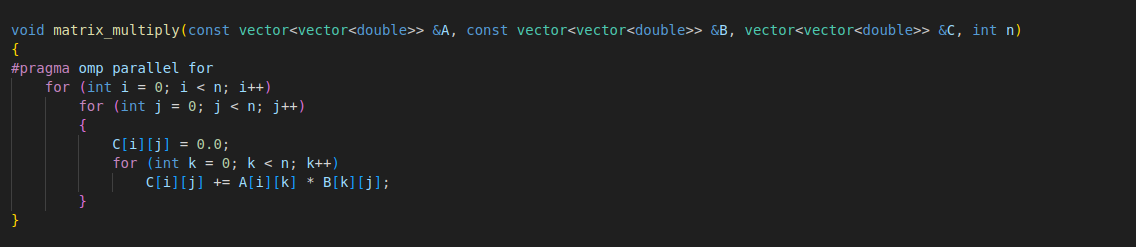
- Сгенерировать и заполнить массив и вектор случайными данными



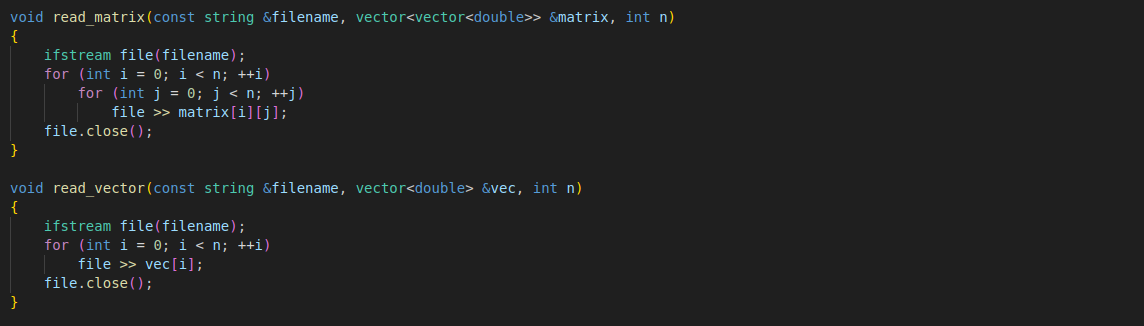
- Скалярное произведение двух векторных функций Я использовал «parallel for reduction (+ : result)»



- Функцию умножения матриц я использовал «omp parallel for»



- Функции чтения из файлов



- Код Python, в котором мы определяем все тесты и создаем статические файлы для проверки программы, а также генерируем график для сравнения.

