Практическая работа № А-07 v2 with MPI

Методы высокопроизводительных вычислений

Альшаеб Басель, группа 24.M71-mm

09.12.2024

# 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 MPI (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 22.04.5 LTS

Kernel: 6.8.0-49-generic

Processer: AMD Ryzen 7 5700G with Radeon Graphics

Cores: 16

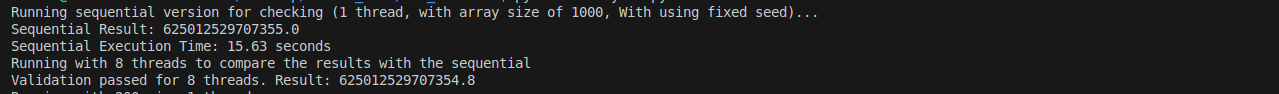
Architecture: x86\_64

Memory: 14 G

Disk Space: 1.5 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 строк массивов, и каждый поток может выполнять вычисления для тех же строк, а затем сводить все результаты вместе.

Каждый раз, когда мы добавляем больше потоков, мы можем распределять операции по разным потокам. В результате это сделает программу быстрее до определенного момента, а затем перегрузка коммуникации замедлит программу.

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

Array Size: 200x200

|  |  |
| --- | --- |
| Threads | Speedup |
| 1 | 1.0 |
| 2 | 1.76 |
| 4 | 1.71 |
| 8 | 2.5 |
| 16 | 1.27 |
| 32 | 0.03 |
| 64 | 0.01 |

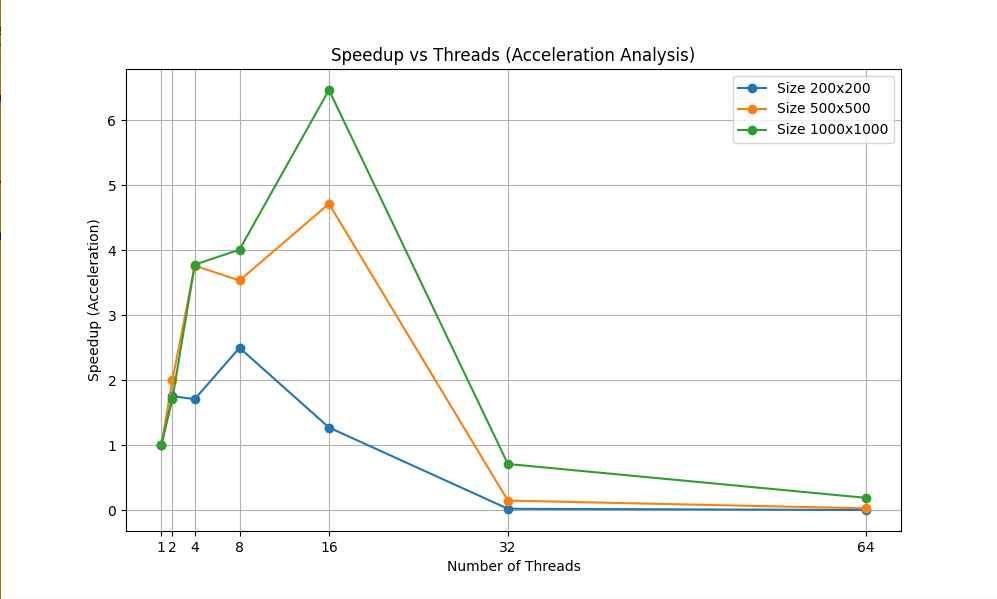
Array Size: 500

|  |  |
| --- | --- |
| Threads | Speedup |
| 1 | 1.0 |
| 2 | 2.0 |
| 4 | 3.76 |
| 8 | 3.53 |
| 16 | 4.71 |
| 32 | 0.15 |
| 64 | 0.03 |

Array Size: 1000

|  |  |
| --- | --- |
| Threads | Speedup |
| 1 | 1.0 |
| 2 | 1.72 |
| 4 | 3.78 |
| 8 | 4.01 |
| 16 | 6.46 |
| 32 | 0.71 |
| 64 | 0.19 |

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



# 5- Объяснение результатов и выводы

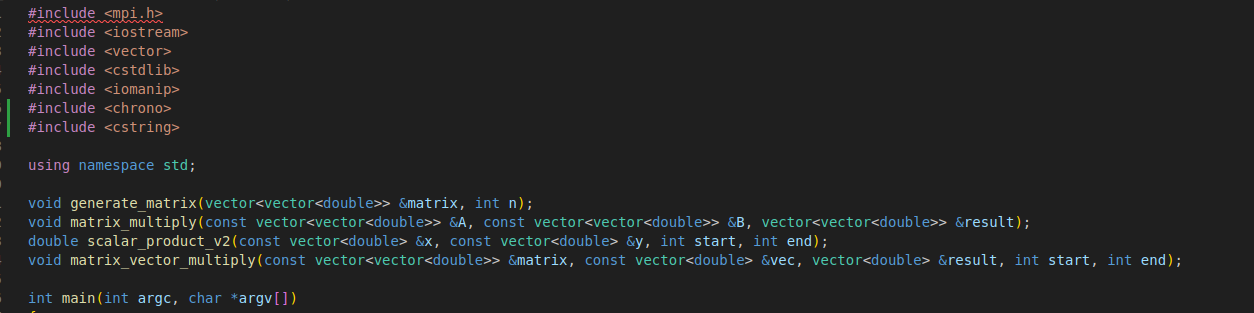
Из графика видно, что ускорение увеличивается с потоками, размер массива также играет роль. По мере увеличения размера ускорение также меняется в зависимости от размера, поскольку это квадратный массив.

- Из этого графика можно сказать, что чем больше потоков мы добавляем, тем большую нагрузку мы добавляем к коммуникации, и это снижает качество решения.

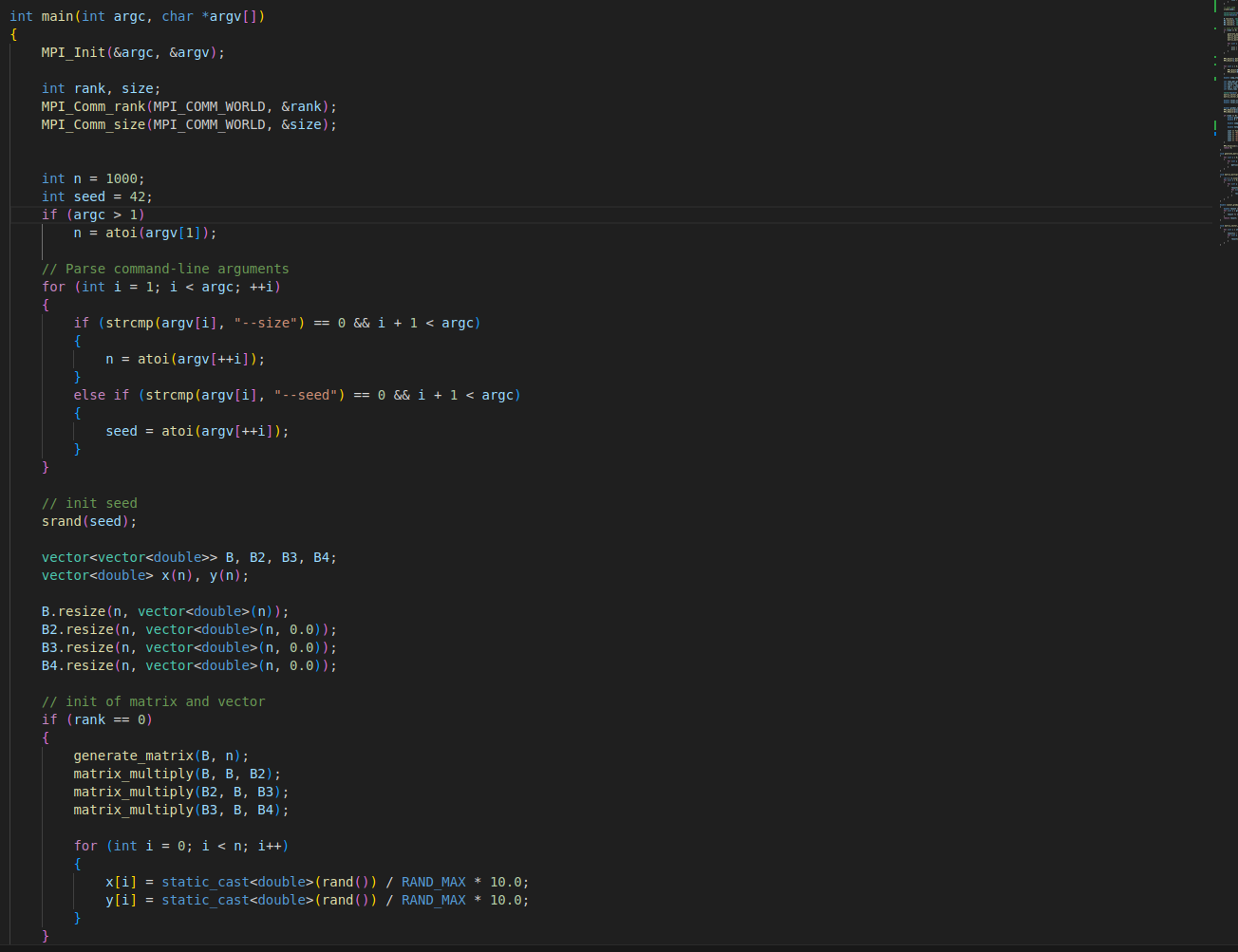
- Во всех случаях бывает так, что ускорение снижается после определенного количества потоков.

# 6- Исходный код

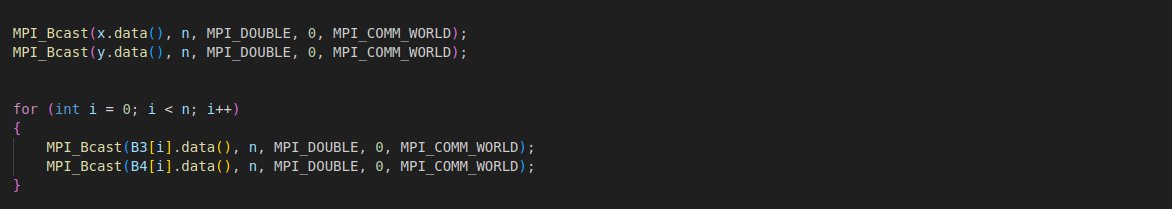
- Определение основных функций



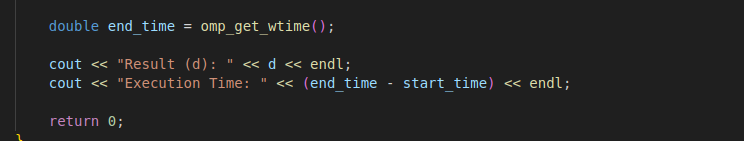
- определение основных переменных и массивов. и добавление параметров, которые помогут нам провести эксперимент. Итак, у нас есть «seed», который говорит нам использовать предопределенный массив, «size» для установки размера массива.



- Сначала я отправляю массив и векторы в MPI\_Bcast, а затем отправляю основной массив построчно, чтобы уменьшить нагрузку на связь. Я также могу распределить только необходимые строки, но безопаснее отправлять весь массив и код более читаемый.



- Печать результата

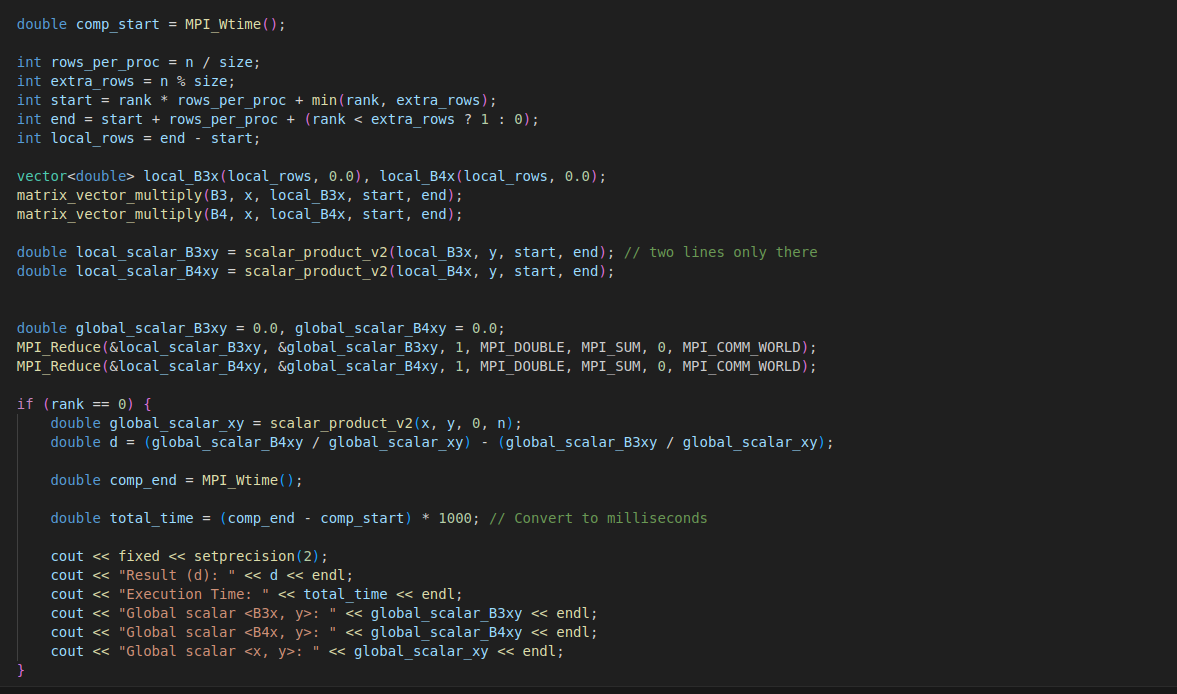


- Основная логика здесь: я делю строки на количество потоков и добавляю дополнительные строки для каждого процесса. Затем вычисляю диапазон, в котором каждый поток будет выполнять работу внутри.

- После этого мы выполняем умножение внутри этого диапазона и используем результат в скалярном произведении также в том же диапазоне.

- Наконец, я уменьшаю все результаты для конечных скаляров, а затем в корневом потоке выполняю конечную операцию.

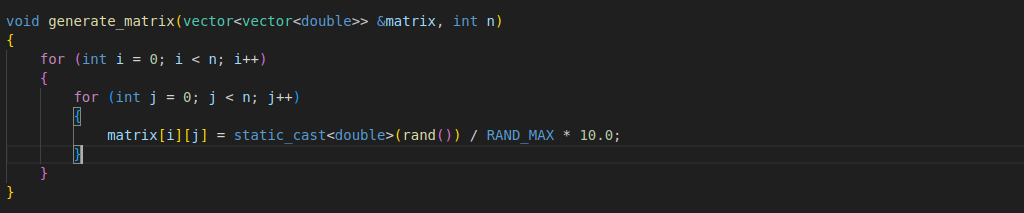
Для расчета времени я использовал MPI\_Wtime.



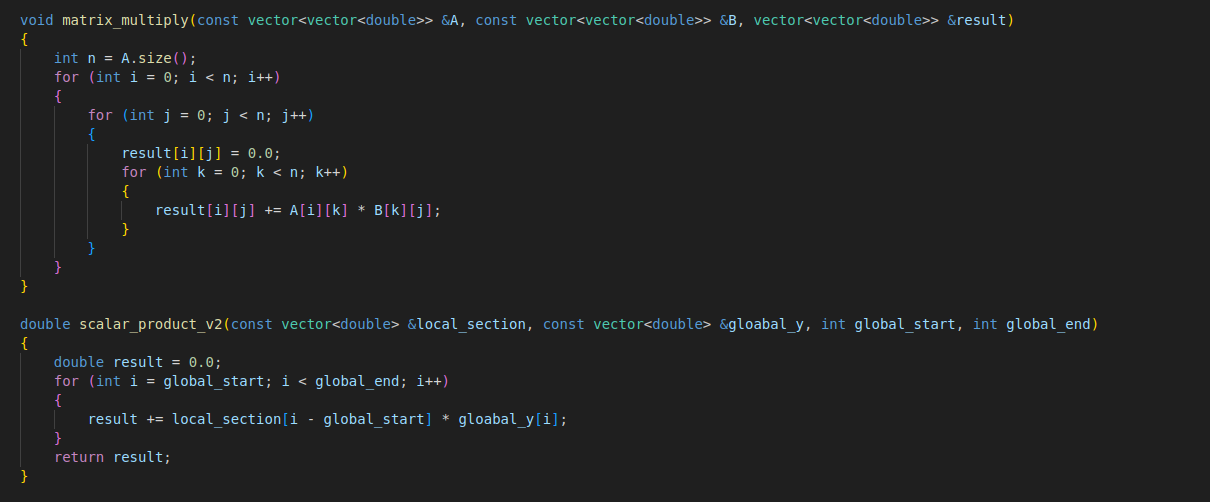
- В конце закрываем MPI с помощью MPI\_Finalize



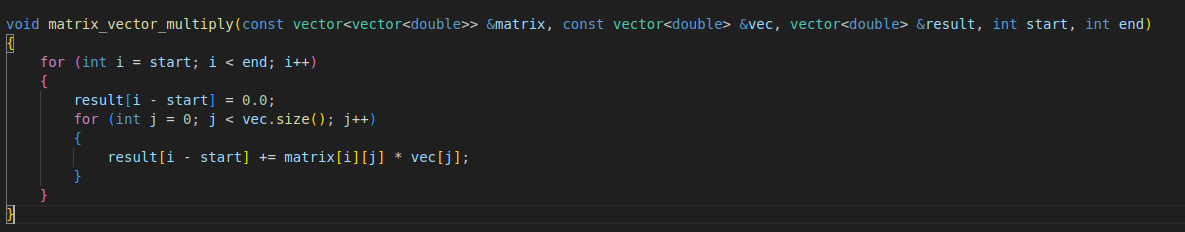
- Сгенерировать и заполнить массив случайными данными



- Умножение матриц выполняется так же, но скаляр использует диапазонные операции.



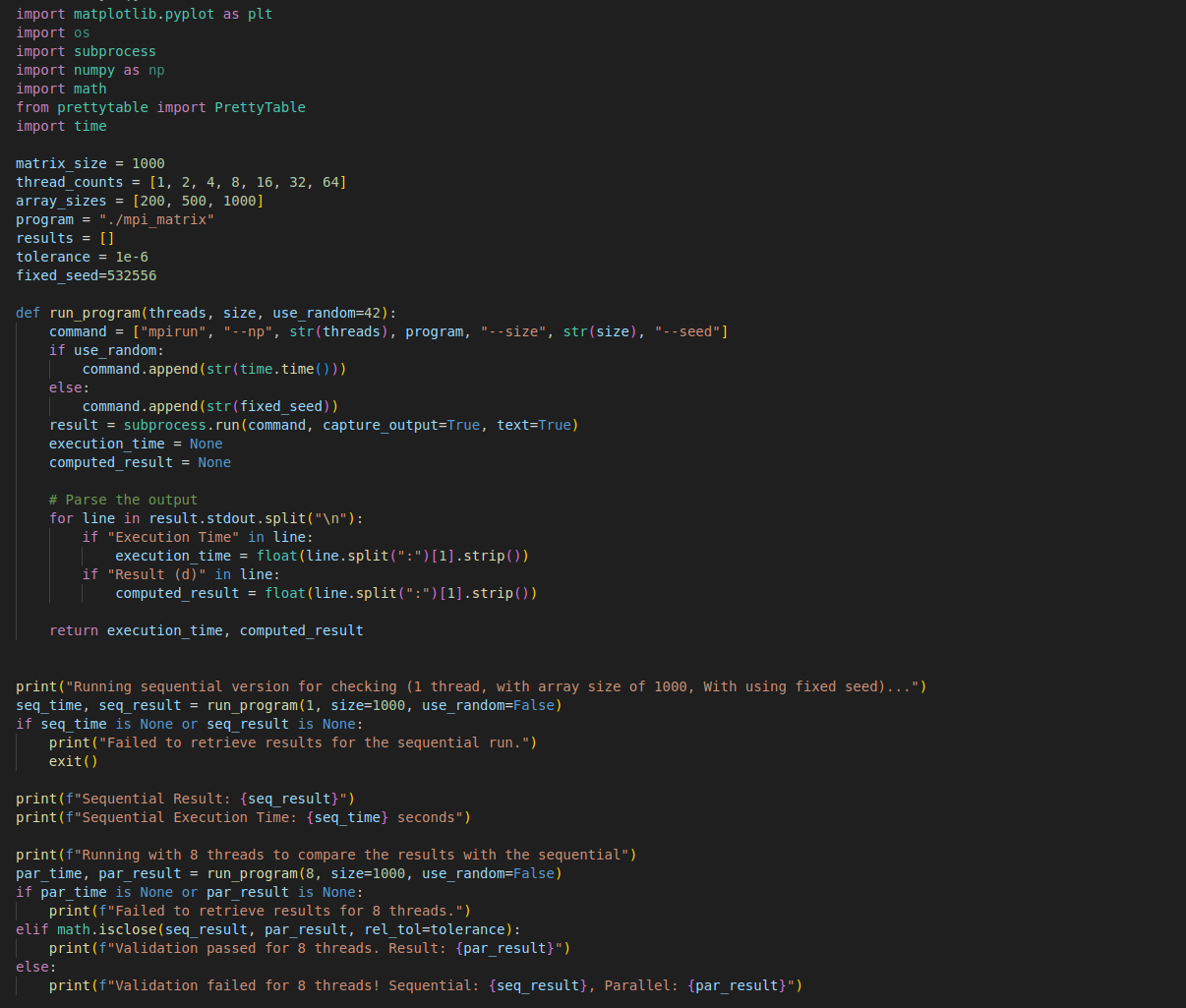
- Умножение вектора на матрицу.



- Код Python, где мы запускаем программу и передаем размер и начальное число, как нам нужно.

- Там же мы извлекаем время выполнения операции из журнала.

Запуск последовательной и параллельной версии, чтобы мы могли сравнить результаты



- Расчет ускорения раздела с массивом и построением диаграммы

