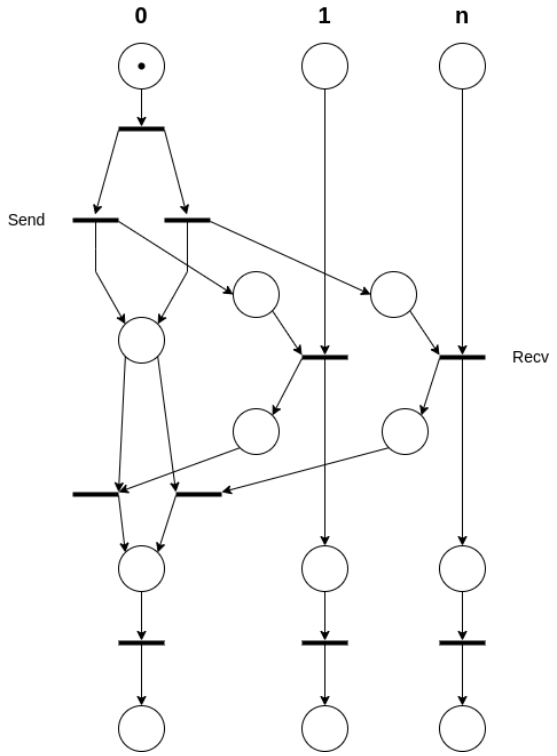
Задание

Выполнить задачу умножения двух квадратных матриц A и B размера m × m, результат записать в матрицу C. Реализовать последовательный и параллельный алгоритм (блочный алгоритм Фокса) и провести анализ полученных результатов. Все числа в заданиях являются целыми. Матрицы должны вводиться и выводиться по строкам.

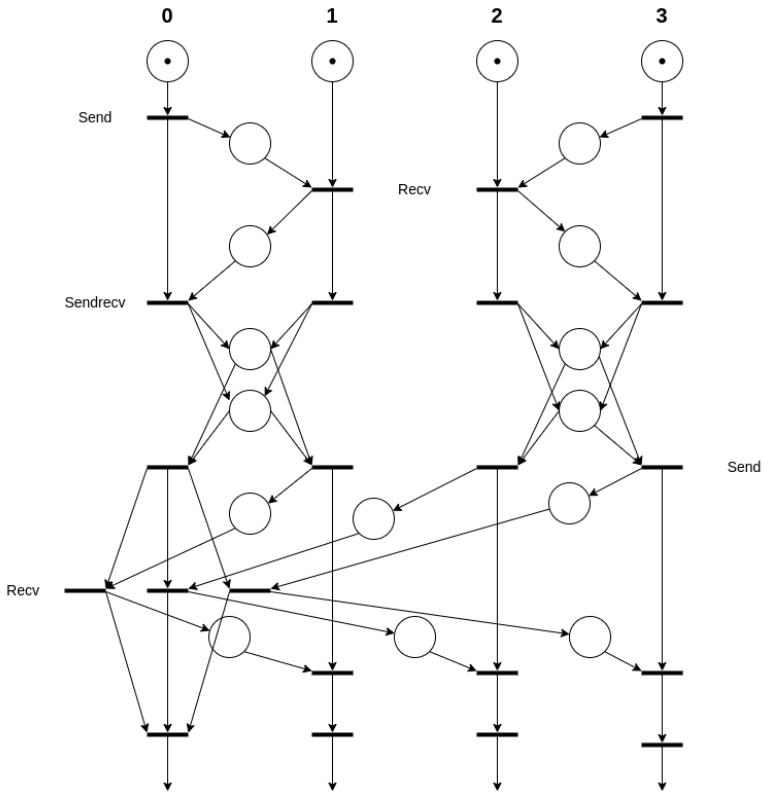
Алгоритм решения задачи

Для всех процессов определяется декартовая топологию в виде матрицы GridComm. После чего, используя функцию MPI_Cart_sub, матрица процессов расщепляется на одномерные столбцы и строки ColComm и RowComm соответственно. Итерация алгоритма: в коммуникаторах RowComm текущий блок A рассылается на все процессы с помощью MPI_Bcast, рассчитывается блок C, в коммуникаторах ColComm по кругу отправляются блоки B в соседние процессы с помощью MPI_Sendrecv. Для сбора получения результата функцией MPI_Gather внутри MPI_COMM_WORLD из процессов собираются блоки C в процессе 0.

Считывание матриц из файла происходит в процессе 0. Процесс 0 разбивает матрицу на блоки и рассылает их в процессы в соответствии с рангами внутри GridComm вызовами функции MPI_Send. Во всех процессах кроме нулевого вызываются MPI_Recv. Сеть Петри для рассылки блоков матриц в процессы:



Формальное описание алгоритма с использованием аппарата сетей Петри (последняя итерации + сбор результатов MPI_Gather):



Сравнительный анализ эффективности алгоритмов

Асимптотика последовательного алгоритма – $O(n^3)$ (три вложенных цикла for: по строкам левой матрицы, по столбцам правой матрицы, по элментам).

Асимптотика алгоритма Фокса – $O(n^3/p)$ (умножение блоков матриц в каждом процессе отдельно).

Выполнение работы

Для решения поставленной задачи были написаны вспомогательные программы: генератор матриц и программа, реализующая последовательное умножение матриц.

Далее была написана программа для параллельного умножения матриц алгоритмом Фокса. Для всех процессов определяется декартовая топологию в виде матрицы размера q x q (q – корень из числа процессоров) GridComm. Считывание матриц из файла происходит в процессе 0. Процесс 0 разбивает матрицу на блоки и рассылает их в процессы в соответствии с рангами внутри GridComm вызовами функции MPI_Send. Во всех процессах кроме нулевого вызываются MPI_Recv. После чего, используя функцию MPI_Cart_sub, матрица процессов расщепляется на одномерные столбцы и строки ColComm и RowComm соответственно. В каждом процессе инициализируется алгоритм – блок C заполняется нулями. Блок A запоминается в переменной temp_block_A. Далее реализуется сам алгоритм. Одна итерация: в коммуникаторах RowComm текущий блок A рассылается на все процессы с помощью MPI_Bcast, рассчитывается блок C, в коммуникаторах ColComm по кругу отправляются блоки B в соседние процессы с помощью MPI_Cart_shift и MPI_Sendrecv. Для сбора получения результата функцией MPI_Gather из

процессов собираются блоки C в процессе 0. Процесс 0 выводит полученную матрицу.

Тестирование программы:

```
polina@polina:~/Documents/pa/lab6$ bash script.sh 1 4 4
sequential:
157 145 122 134
100 90 76 78
105 92 71 58
84 84 69 39

fox:
157 145 122 134
100 90 76 78
105 92 71 58
84 84 69 39

polina@polina:~/Documents/pa/lab6$ bash script.sh 1 2 4
sequential:
12 39
2 17

fox:
12 39
2 17
```

Для тестирования программ написан bash-скрипт, который запускает все программы.

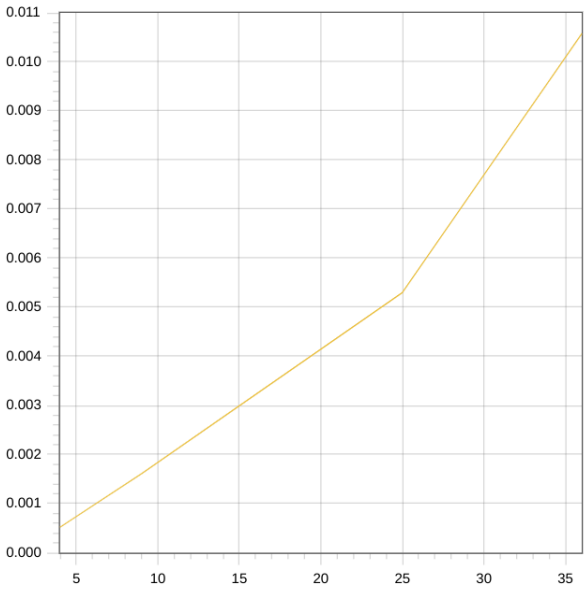
Результаты экспериментов:

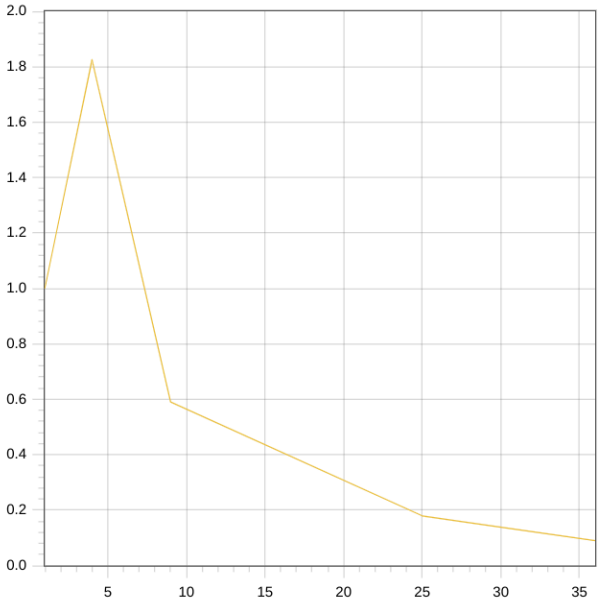
размерность матрицы	последовательный алгоритм	4 процесса	9 процессов	25 процессов	36 процессов
30	0.000949	0.000520	0.001608	0.005301	0.010572
90	0.00417	0.004898	0.007428	0.020752	0.062684
300	0.086238	0.036229	0.122440	0.129539	0.198237
600	0.806417	0.498884	0.555108	0.602285	0.837111
810	3.315671	*	2.453735	2.294095	1.516398

* – невозможно протестировать из-за необходимости выделения слишком больших областей памяти.

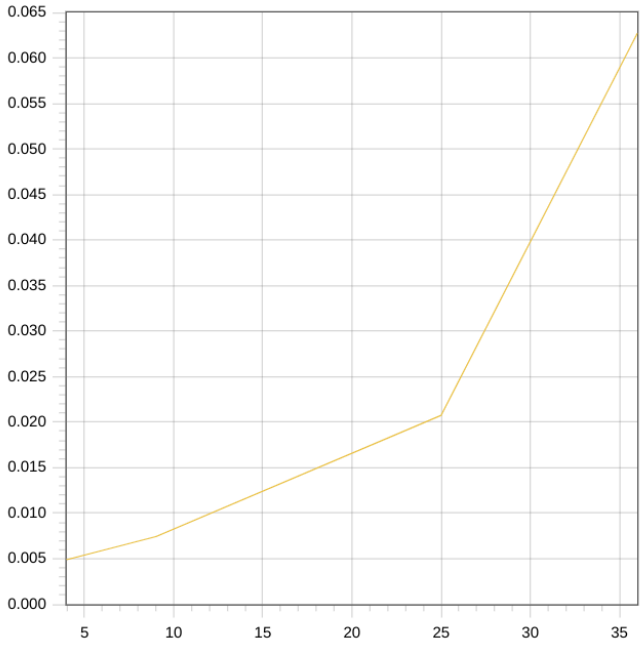
Графики зависимости времени выполнения программы от числа процессов и ускорения для разного объема исходных данных:

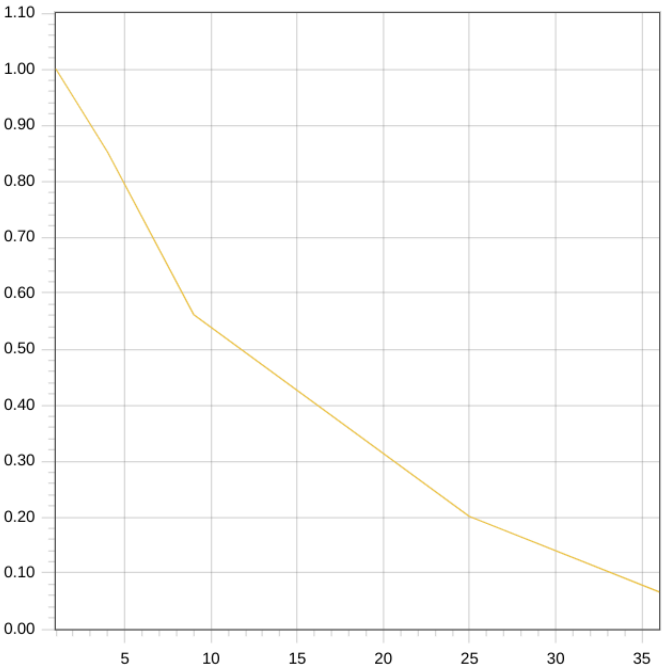
1) размерность матрицы – 30 элементов



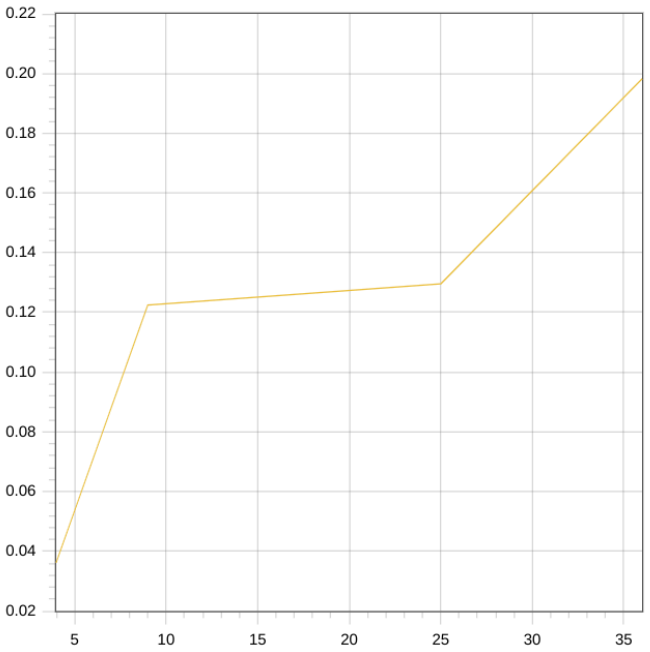


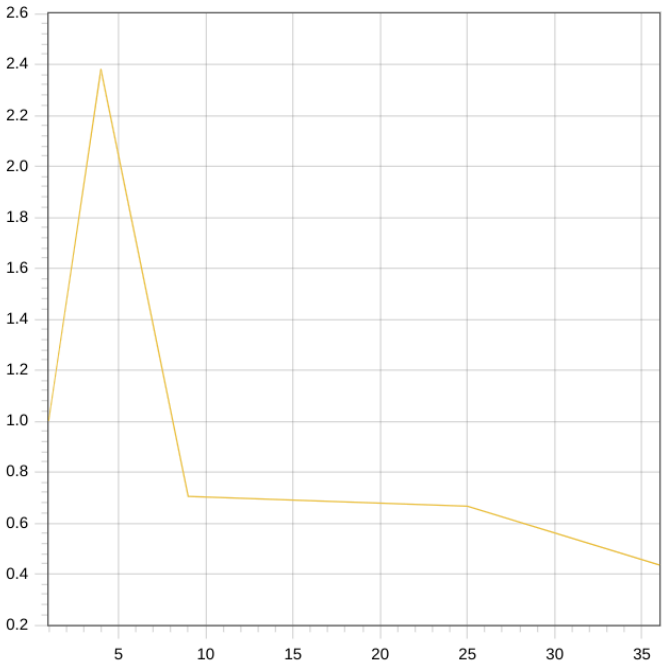
2) размерность матрицы – 90 элементов


0.065
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
5
10
15
20
25
30
35

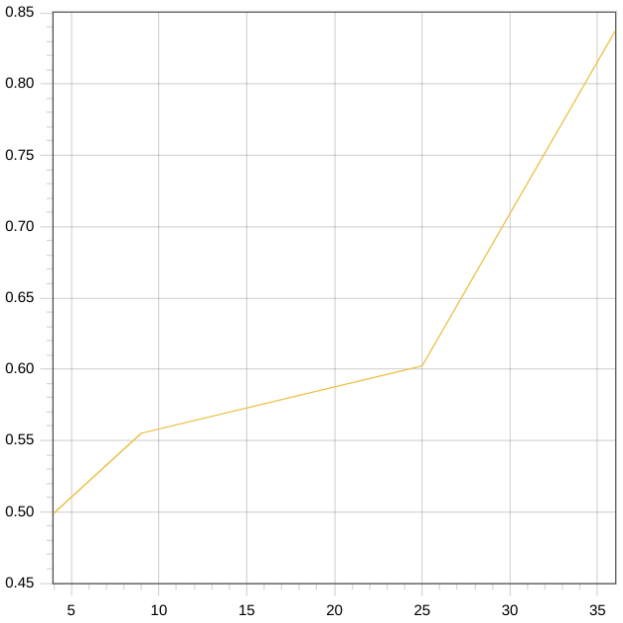

1.10
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
5
10
15
20
25
30
35

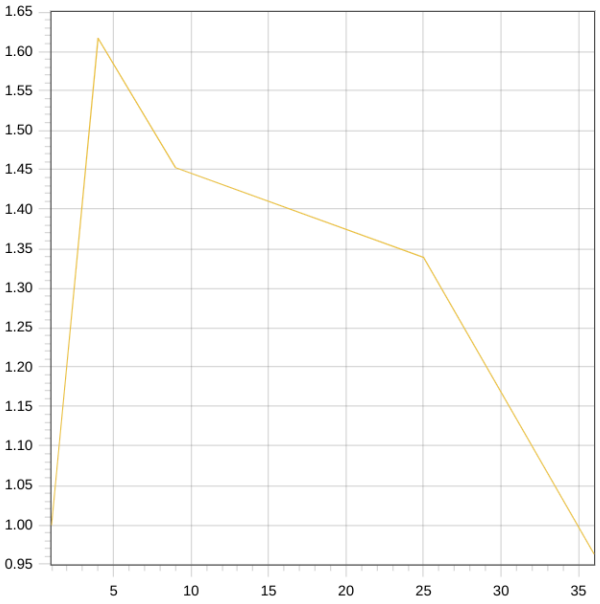
3) размерность матрицы – 300 элементов



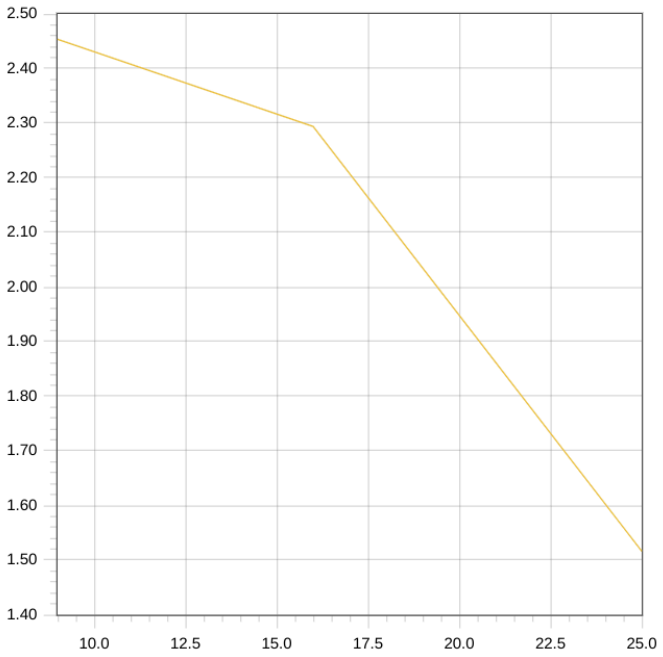


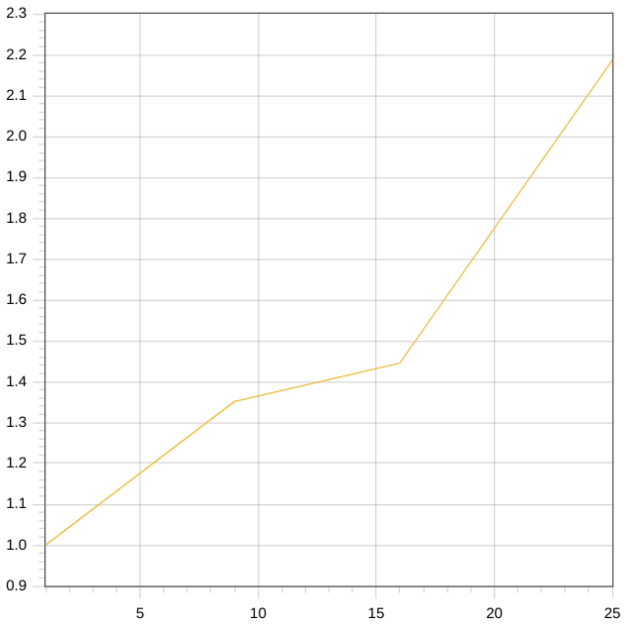
4) размерность матрицы – 600 элементов





5) размерность матрицы – 810 элементов





Вывод

Как видно из графиков ускорения в данной задаче можно достичь при правильной комбинации объема входных данных и количества процессов. Если для данного объема входных данных процессов слишком много, то накладные расходы на взаимодействие между процессами перевешивают выигрыш во времени самих вычислений. Если для данного

объема входных данных процессов слишком мало, то количество итераций последовательного алгоритма растет, и время выполнения параллельной программы стремится ко времени выполнения последовательной.

Учитывая асимптотики алгоритмов время выполнения программ должно было бы отличаться в p (количество процессов) раз. Но на практике это невозможно, так как добавляются накладные расходы на взаимодействие между процессами.