Отчёт по 4-му заданию спецкурса "Параллельное программирование для высокопроизводительных вычислительных систем".

Подготовил: студент 428 группы Манушин Дмитрий Валерьевич.

1. Описание задачи.

В задании требовалось реализовать блочный параллельный алгоритм Кеннона для перемножения произвольных, хранящихся в файлах по строкам плотных матриц и исследовать его эффективность.

2. Описание решения.

В ходе выполнения задания реализован параллельный алгоритм Кеннона для умножения матриц. При запуске на n процессах алгоритм умножения матрицы A на матрицу B состоит в следующем.

1) Организуем процессы в виртуальную двумерную декартову топологию (матрицу с соседними краями), используя функцию MPI_Cart_create.

2) Матрицы разбиваются на n равных для одной матрицы прямоугольных блоков. Каждый процесс в соответствии со своими координатами в декартовой топологии считывает свои блоки матриц A и B.

3) Строки i решетки подзадач блоки матрицы A сдвигаются на (i - 1) позиций влево, для каждого столбца j решетки подзадач блоки матрицы B сдвигаются на (j - 1) позиций вверх с помощью функций MPI_Cart_shift и MPI_Sendrecv_replace.

4) Проводится n итераций, во время которых сначала происходит перемножение блоков по методу трех вложенных циклов, и произведение складывается с текущим значением результирующего блока. Затем выполняется циклический сдвиг блоков матрицы A вдоль строк решетки влево и циклический сдвиг блоков матрицы B вверх по столбцам виртуальной решетки.

5) После выполнения всех итераций в процессе с координатами (i, j) получится блок C(i, j) результирующей матрицы C. После этого выполняется запись блока C(i, j) в файл матрицы C по нужным смещениям.

Описанный алгоритм реализуется в функции mpi_cannon_multiply_and_save_matrix(char *A_matrixname, char *B_matrixname, char *resultname).

Также при выполнении данного задания в соответствии с требованиями задачи изменены программы и скрипты, описанные в предыдущем отчёте. Умножения матриц по-прежнему можно запустить с помощью программы main, принимающей 3 аргумента: путь к файлу с матрицей A, путь к файлу с матрицей B, количество процессов.

3. Проверка результатов вычислений.

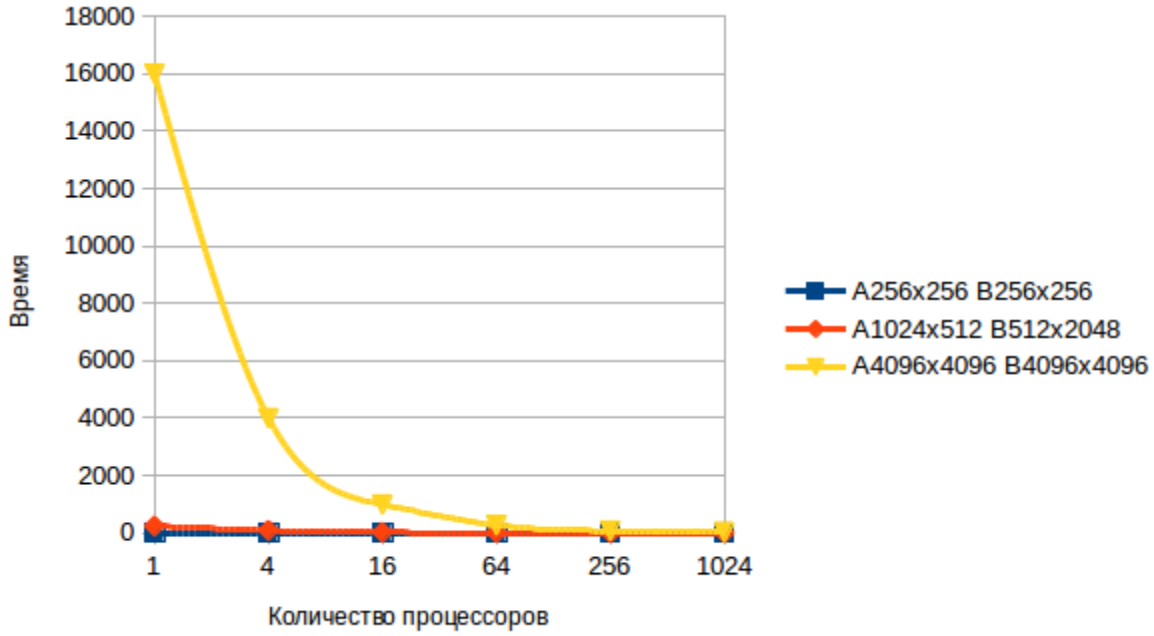
Проверка правильности алгоритма осуществлена с помощью функций умножения

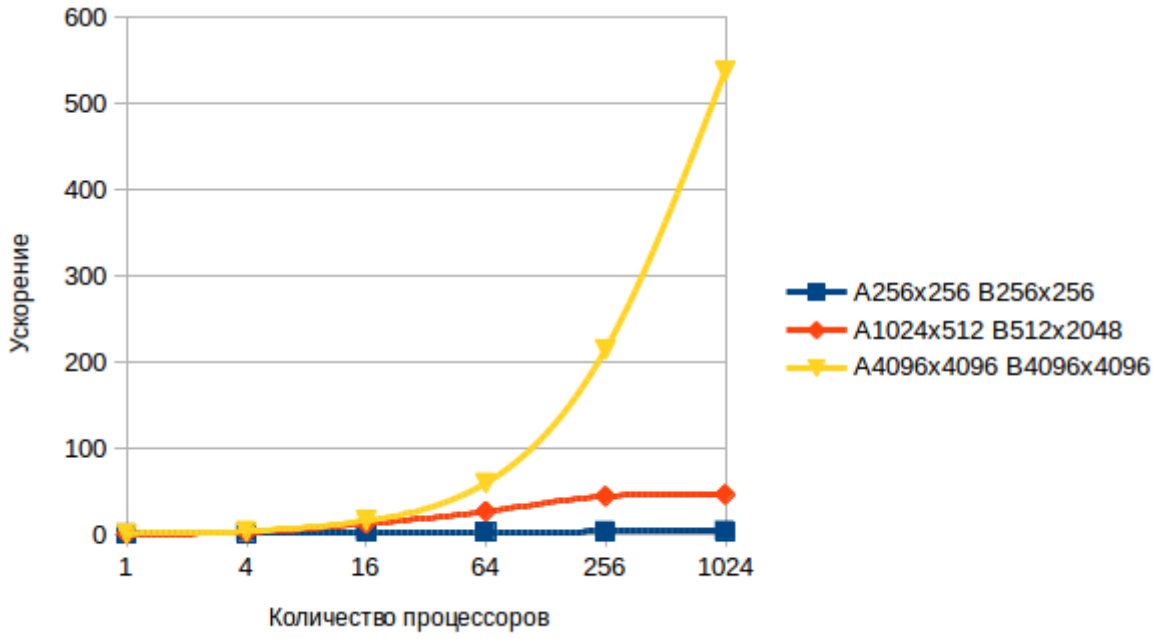
матриц на Python (results/matrix.py). Скрипт проверки check.py находится в папке results.

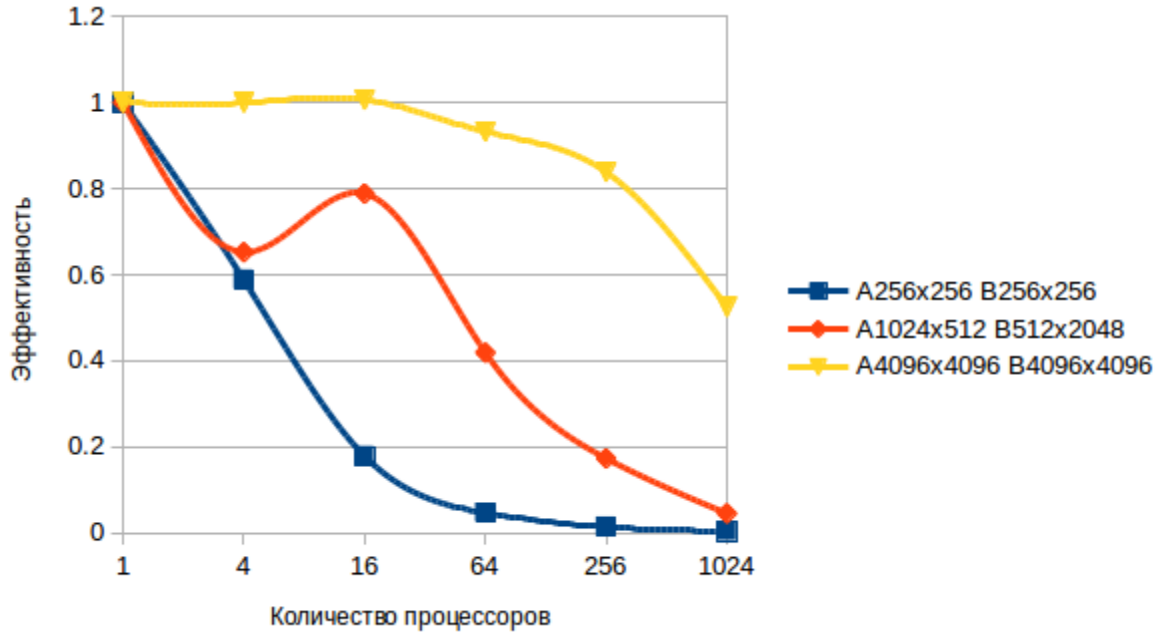
4. Результаты.

В ходе выполнения задания было замерено время выполнения алгоритма для размеров матриц и разных количеств процессоров. Были построены графики времени выполнения, ускорения и эффективности.

Полученные графики для матриц:







5. Выводы.

Сложность алгоритма Кеннона O(N*M*K) при умножении матрицы размера N*M на матрицу размера M*K. Упор в алгоритме делается на упрощение и максимальное распараллеливание коммуникаций между процессорами.

По графикам можно заметить, что использование нескольких процессоров для умножения матриц позволяет значительно ускорить процесс умножения.

Исходя из данных замеров, в сравнении с ленточным алгоритмом, алгоритм Кеннона проявляет большую масштабируемость. При количестве процессоров, большем некоторого значения для данной матрицы, ускорение ленточного алгоритма снижается из-за больших накладных расходов. При этом ускорение в алгоритме Кеннона только увеличивается.

Однако, стоит заметить, что при маленьких количествах процессоров ленточный алгоритм показывает немного лучшие результаты. Но, в целом, результаты обоих алгоритмов сопоставимы.

Таким образом, алгоритм Кеннона позволяет довольно эффективно выполнять параллельное умножение матриц, обеспечиваю достаточно высокий уровень масштабируемости.