Отчёт по 3-му заданию спецкурса "Параллельное программирование для высокопроизводительных вычислительных систем".

Подготовил: студент 428 группы Манушин Дмитрий Валерьевич.

1. Описание задачи.

В задании требовалось разработать параллельный алгоритм для пермножения произвольных, хранящихся в файлах по строкам плотных матриц и исследовать его эффективность.

2. Описание решения.

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

1) Матрицы разбиваются по строкам на n равных частей размером количество строк / n и остаток, каждый процесс загружает свою часть матриц.

2) В цикле от 0 до количества столбцов в B на каждом шаге выполняется сборка i-го столбца матрицы B всеми процессами из имеющихся у них его частей с помощью функции MPI_Allgatherv. Этот столбец умножается на имеющиеся у каждого процесса строки матрицы A с помощью описанного во 2-м задании умножения матрицы на вектор. После прохождения всех шагов, у каждого процесса получится матрица произведения своей части матрицы A и всей матрицы B, которая соответствует части строк матрицы результата.

3) Полученная часть матрицы результата записывается каждым процессом в результирующий файл по нужному смещению.

Описанный алгоритм реализуется в функции mpi_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. Запуск 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.

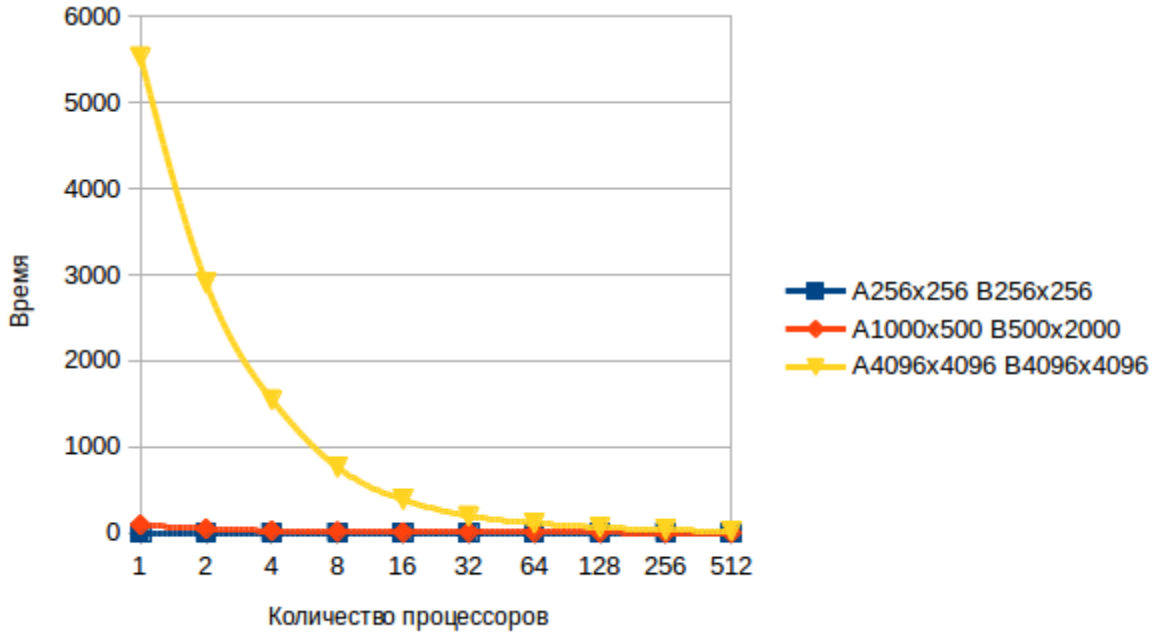
Проверка правильности алгоритма осуществлена с помощью функций умножения матриц на Python (results/matrix.py), запустив скрипт проверки results/check.py. Дополнительно можно проверить правильность работы с помощью сравнения результатов выполнения алгоритма на разных количествах процессоров (утилита cmp).

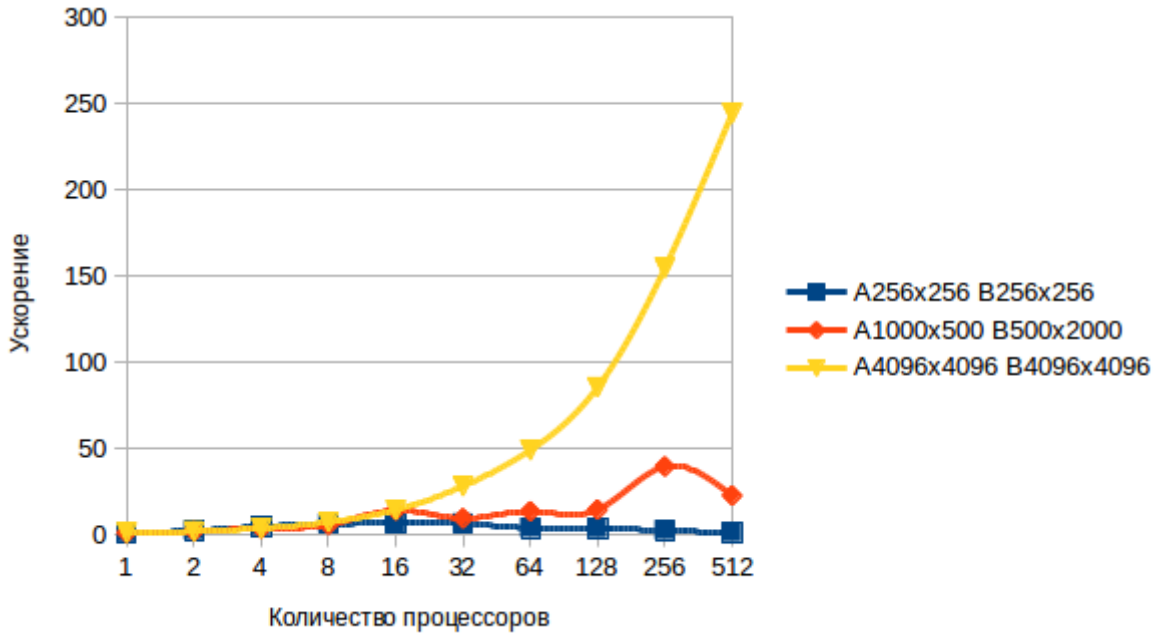
4. Результаты.

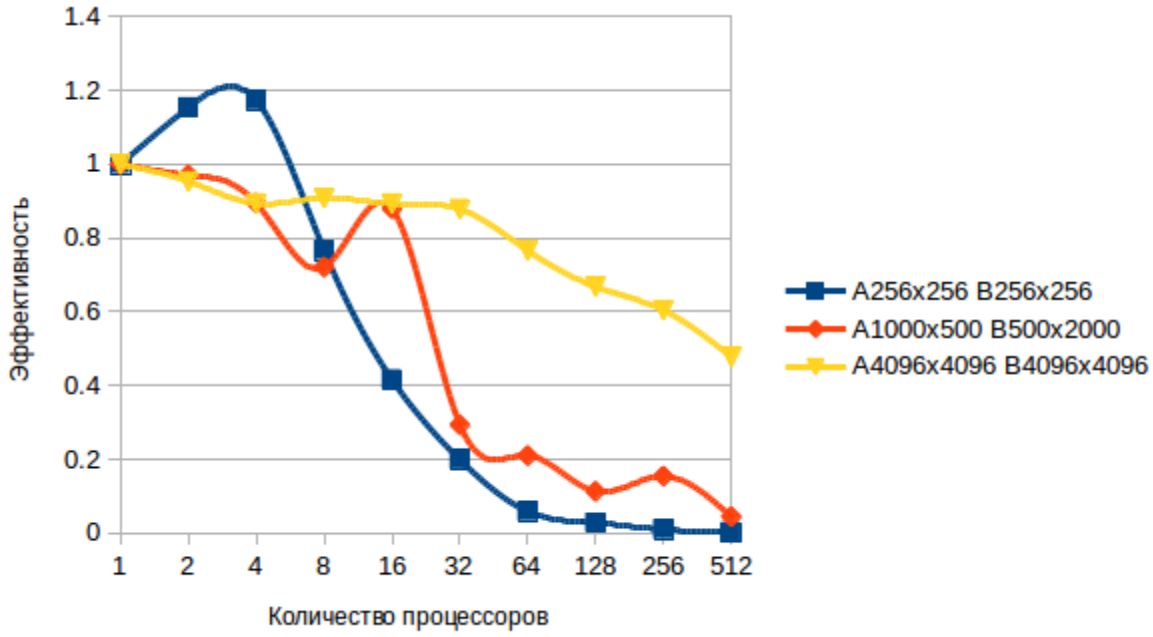
В ходе выполнения задания было замерено время выполнения алгоритма для

размеров матриц и разных количеств процессоров. Были построены графики времени выполнения, ускорения и эффективности.

Полученные графики для матриц:







5. Выводы.

Сложность представленного алгоритма O(N*M*K) при умножении матрицы размера N*M на матрицу размера M*K. Во время работы алгоритма выполняется пересылка M*K значений.

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

Ускорение обычно растёт и эффективность остаётся близко к уровню 1 до некоторой точки, после которой эффективность уменьшается и даже ускорение может падать. Это так называемая точка насыщения. Её можно объяснить превышением накладных расходов на пересылку и отдельную обработку малых частей матриц над ускорением за счёт параллельного выполнения.

Также можно заметить, что если для умножения матриц NxN использовать примерно N/64 процессоров, то эффективность будет держаться на уровне единицы. Увеличение количества процессоров также повлечёт увеличение ускорения, хотя и не такое значительное как до этой точки.

Таким образом, предложенный алгоритм позволяет довольно эффективно выполнять параллельное умножение матриц.