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 блочные параллельные алгоритмы Кеннона и Фокса для перемножения произвольных, хранящихся в файлах по строкам плотных матриц и исследовать их эффективность.

2. Описание решения.

A. Алгоритм Кеннона.

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 sqrt(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).

B. Алгоритм Фокса.

В ходе выполнения задания реализован параллельный алгоритм Фокса 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) Проводится sqrt(n) итераций. На i-ой итерации для каждой строки row процесс с координатами (row, (row + i) % sqrt(n)) делает широковещательную рассылку своего блока матрицы A всем процессорам своей строки (используется коммуникатор из MPI_Cart_sub) с помощью MPI_Bcast. После получения блока процесс выполняет его умножение на текущий блок матрицы B, добавляя результат к блоку C(i, j) матрицы C. Затем выполняется циклический сдвиг блоков матрицы B вверх по столбцам виртуальной решётки.

4) 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_fox_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, теперь принимающей 4 аргумента: путь к файлу с матрицей A, путь к файлу с матрицей B, количество процессов, алгоритм: --cannon или --fox.

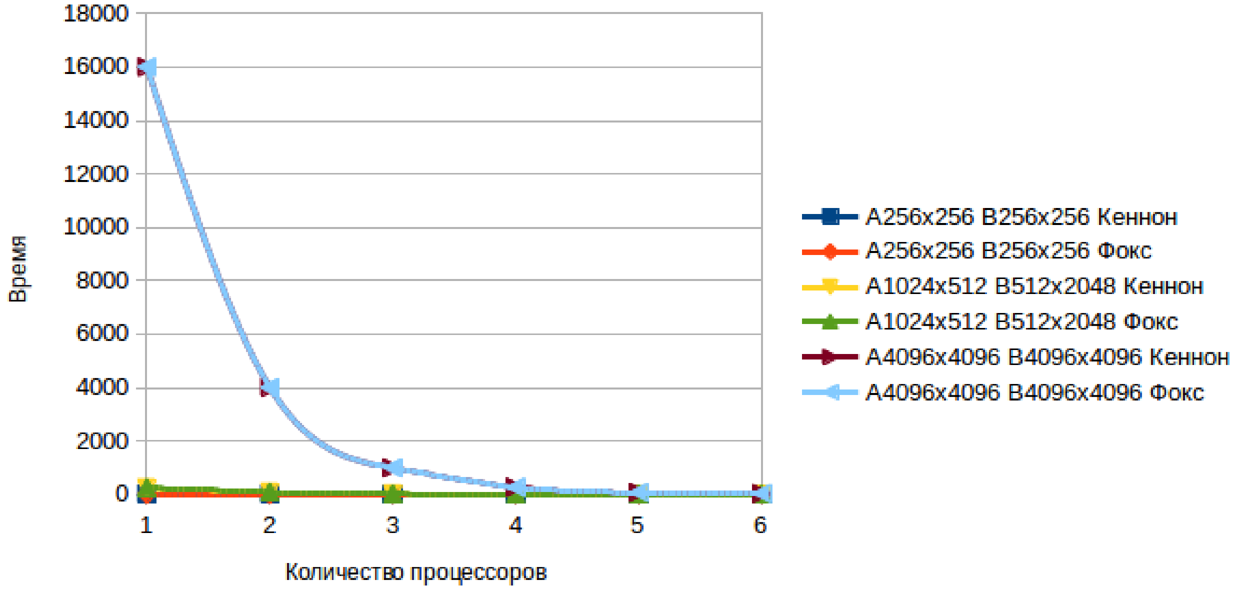
3. Проверка результатов вычислений.

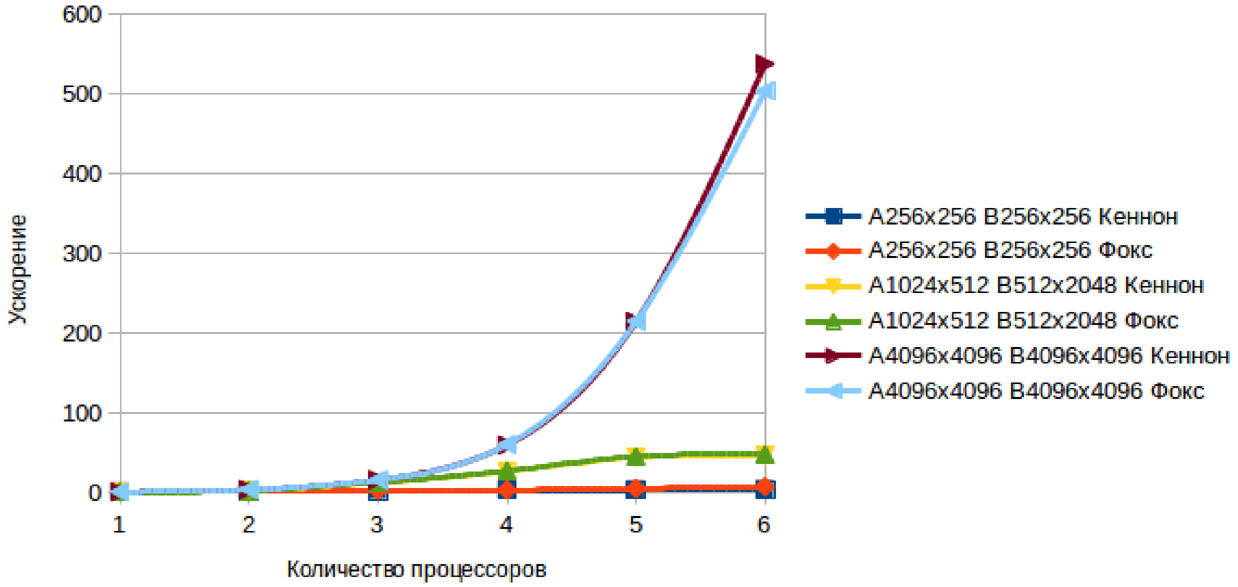
Проверка правильности работы алгоритмов осуществлена с помощью функций умножения матриц на Python (results/matrix.py). Скрипт проверки check.py находится в папке results.

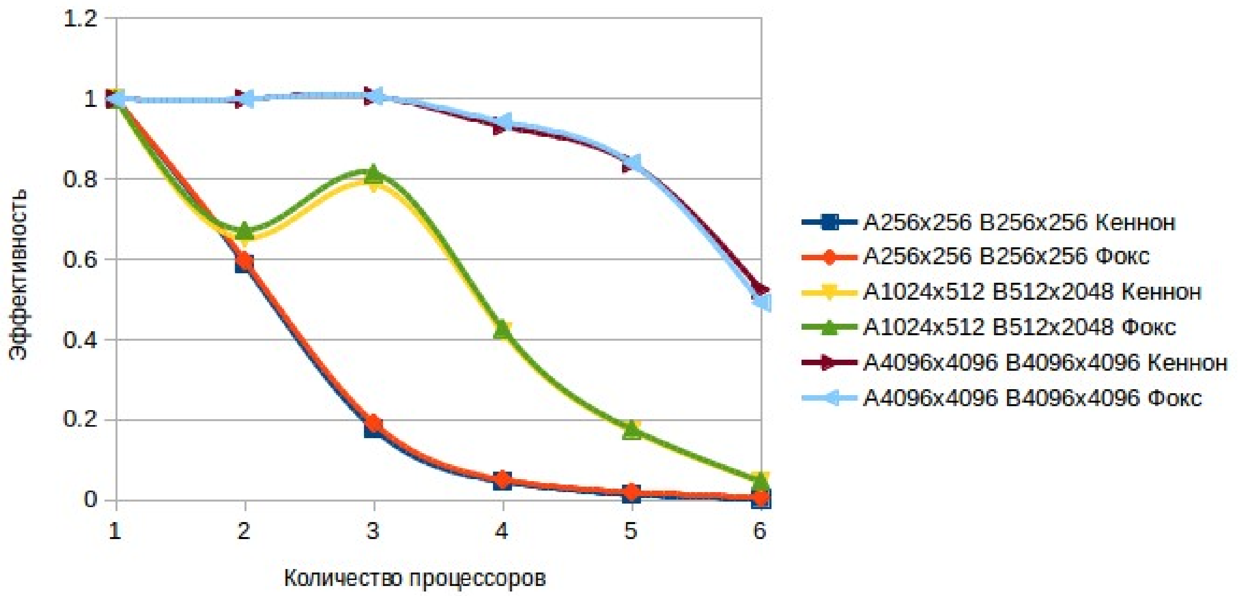
4. Результаты.

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

Полученные графики для матриц:


Время
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
Количество процессоров
A256x256 B256x256 Кеннон
A256x256 B256x256 Фокс
A1024x512 B512x2048 Кеннон
A1024x512 B512x2048 Фокс
A4096x4096 B4096x4096 Кеннон
A4096x4096 B4096x4096 Фокс


Ускорение
600
500
400
300
200
100
0
1
2
3
4
5
6
Количество процессоров
A256x256 B256x256 Кеннон
A256x256 B256x256 Фокс
A1024x512 B512x2048 Кеннон
A1024x512 B512x2048 Фокс
A4096x4096 B4096x4096 Кеннон
A4096x4096 B4096x4096 Фокс


Эффективность
1.2
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
Количество процессоров
A256x256 B256x256 Кеннон
A256x256 B256x256 Фокс
A1024x512 B512x2048 Кеннон
A1024x512 B512x2048 Фокс
A4096x4096 B4096x4096 Кеннон
A4096x4096 B4096x4096 Фокс

5. Выводы.

Сложность и алгоритма Фокса, и алгоритма Кеннона O(N*M*K) при умножении матрицы размера N*M на матрицу размера M*K. Упор в алгоритме Кеннона 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. То же самое можно сказать про алгоритм Фокса.

Различий по времени/ускорению/эффективности между алгоритмами Фокса и Кеннона практически нет. Скорее всего, такая ситуация наблюдается из-за того, что на суперкомпьютере BlueGene/P, благодаря применению различных топологий сети (дерево для коллективных коммуникаций и тор для коммуникаций точка-точка), все коммуникации выполняются достаточно быстро.

Таким образом, алгоритмы Кеннона и Фокса позволяют довольно эффективно выполнять параллельное умножение матриц, обеспечивая достаточно высокий уровень масштабируемости.