Отчёт по 2-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 и исследовать его эффективность для задачи умножения матрицы на вектор для двух типов матриц:
1) Матрица плотная;
2) Матрица разреженная.

2. Описание решения.

В ходе выполнения задания разработан параллельный алгоритм для умножения матрицы на вектор в случае как плотной, так и разреженной матрицы.

Реализована библиотека функций для работы с плотными (lib/matrix_lib.h) и разреженными (lib/sparse_lib.h) матрицами и векторами. В ней есть структуры матриц и векторов, а также функции по их созданию, случайной генерации, загрузке, сохранению, выводу на экран, удалению. Также здесь реализованы функции умножения матрицы на вектор и скалярного произведение векторов. Матрицы (и плотные, и разреженные) хранятся в файлах по строкам для более удобной загрузки. При умножении (mpi_multiply, mpi_sparse_multiply) каждый процесс вычисляет произведение своей части строк матрицы на вектор с помощью функций multiply или sparse_multiply, которые в свою очередь используют функции скалярного произведения.

В папке data лежат плотные матрицы и векторы для тестирования. В целях уменьшения размера архива оставлены только матрица 100x100 и вектор 100x1. Также здесь находится скрипт view, принимающий два параметра - имя файла с матрицей и имя файла с вектором и позволяющий их просмотреть.

В папке gen находятся скрипты и программы для генерации плотных матриц и векторов. Для генерации матрицы и соответствующего ей вектора необходимо запустить скрипт gen с одним параметром - размер N. Скрипт сгенерирует матрицу NxN и вектор Nx1.

В папке datasparse лежат разреженные матрицы и соответствующие им плотные векторы для тестирования. В целях уменьшения размера архива оставлены только матрица 100x100 и вектор 100x1. Также здесь находится файл view.py, в котором находится функция load_matrix(filename), позволяющая загрузить матрицу в интерпретаторе python и просмотреть её (интерпретатор следует запускать командой python -i view.py).

В папке gensparse находятся скрипты и программы для генерации разреженных матриц и соответствующих им плотных векторов. Для генерации матрицы и соответствующего ей вектора необходимо запустить скрипт gen_sparse_3param с 3 параметрами - размер N, коэффициент c, количество процессоров для генерации. Скрипт сгенерирует матрицу NxN и вектор Nx1. Коэффициент c определяет разреженность матрицы - в матрице в среднем будет c * N ненулевых элементов

(вероятность ненулевого элемента будет c / N). Генерация устроена параллельно - при запуске на n процессорах, все процессы, кроме 0-го, генерируют случайные разреженные векторы с заданной вероятностью ненулевого элемента, а 0-й процесс собирает эти векторы и записывает в файл матрицы. Для оптимизации генерации векторов с заданной вероятностью ненулевого элемента используется шаг, определяемый геометрическим распределением geom_random (более подробно см. функцию sparse_vector_gen).

В папках results и resultssparse хранятся результаты вычислений.

Файл main.c выполняет умножение матриц. Программа принимает 3 параметра - размер матрицы, количество процессов, --sparse или --dense. При этом будет произведено умножение матрицы соответствующего размера из соответствующей папки data или datasparse на соответствующий вектор. Для запуска можно пользоваться скриптами ri (для плотных) и risparse (для разреженных), принимающими 2 параметра - количество процессов и размер матрицы.

Дополнительные скрипты show.py и runall предназначены для сбора статистики по времени выполнения и запуска нескольких задач за один раз соответственно.

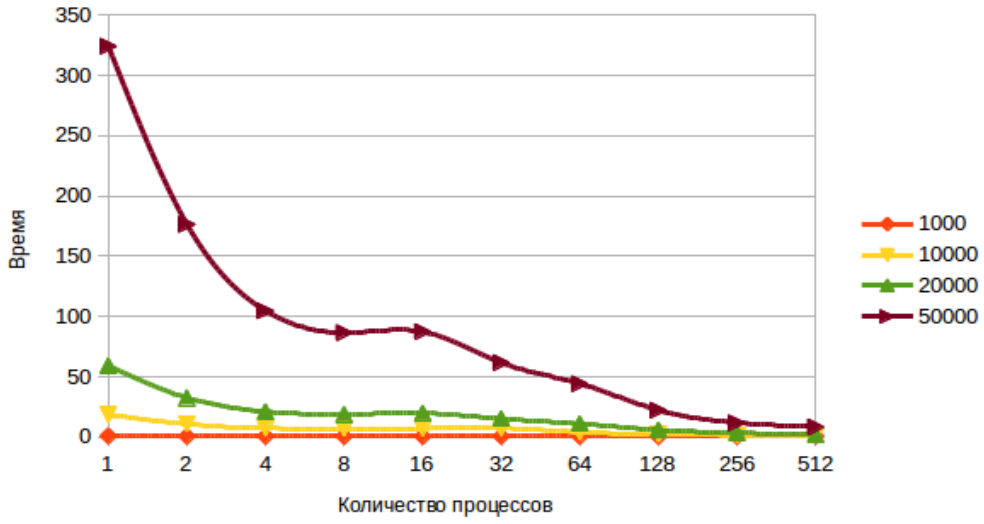
3. Проверка результатов вычислений.

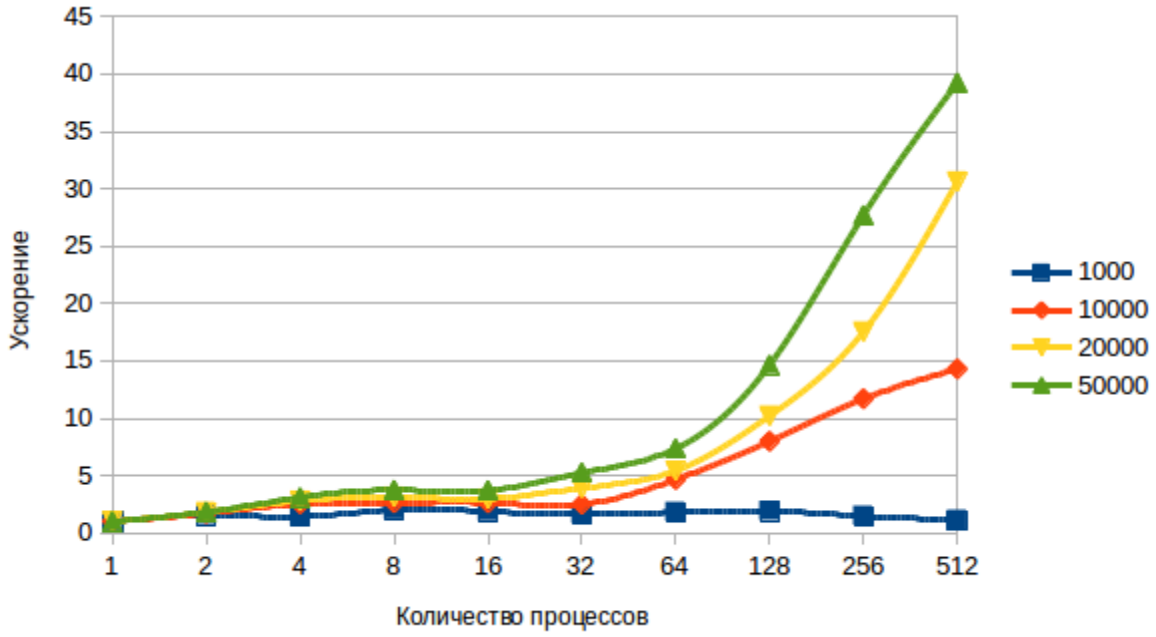
Проверка правильности алгоритма осуществлена с помощью функций умножения матриц на Python (файлы gen/generator.py, gen/check.py), а также с помощью сравнения результатов выполнения алгоритма на разных количествах процессов (утилита cmp).

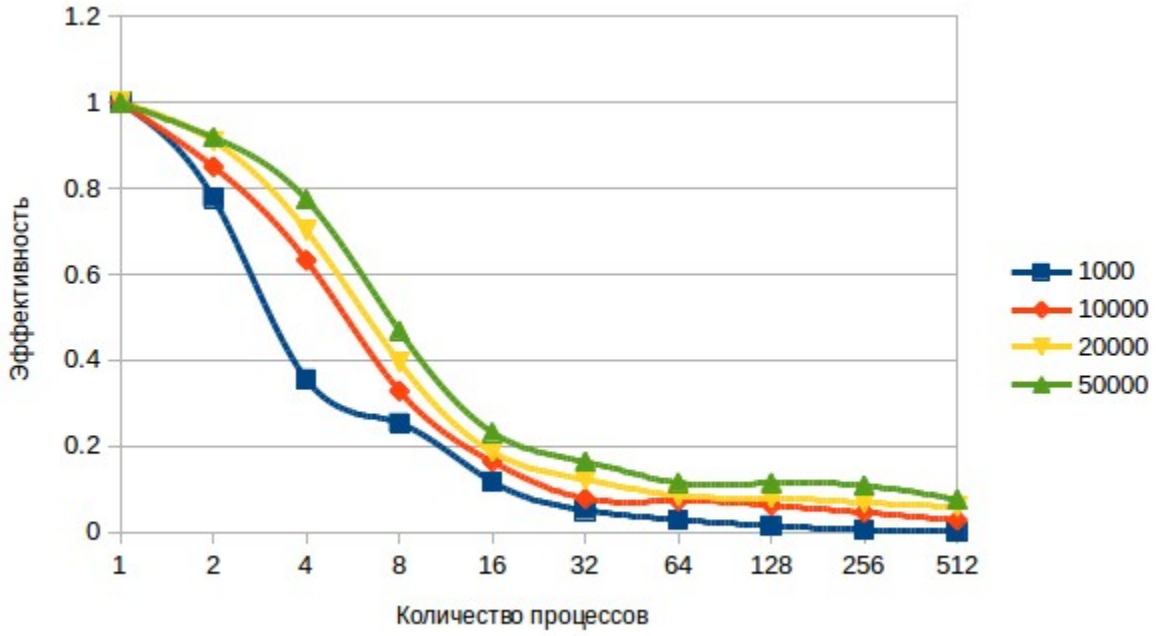
4. Результаты.

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

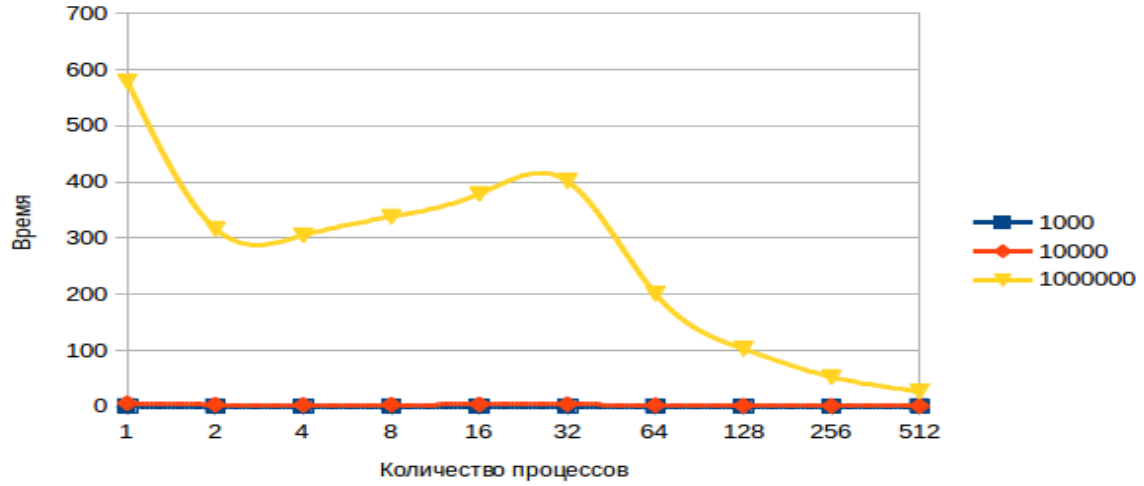
Полученные графики для плотных матриц:

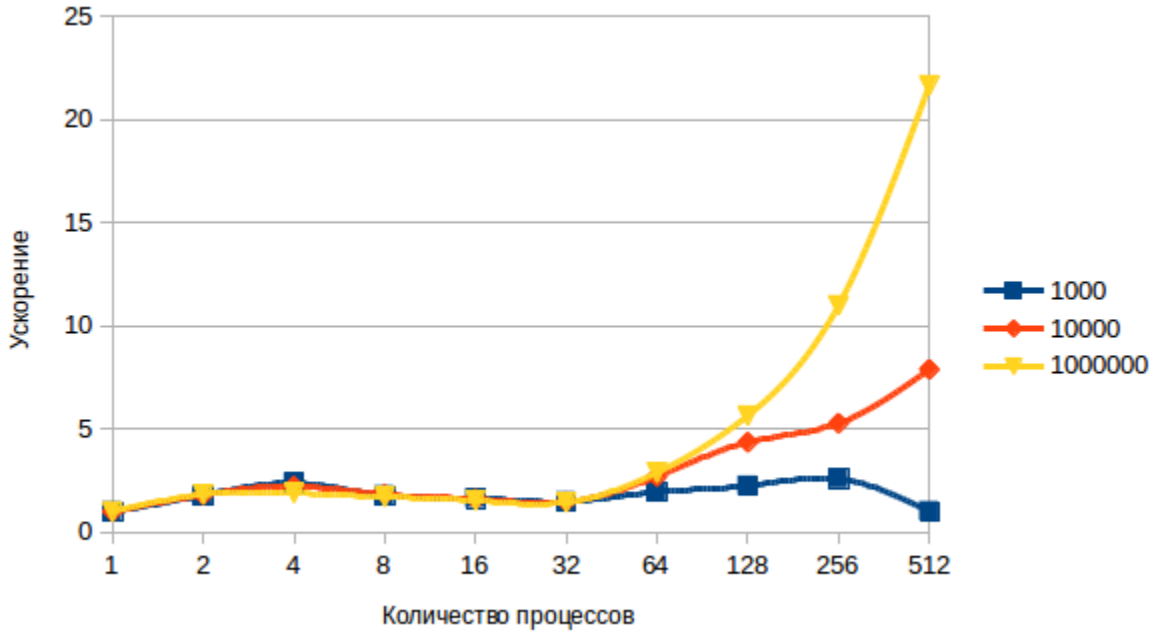


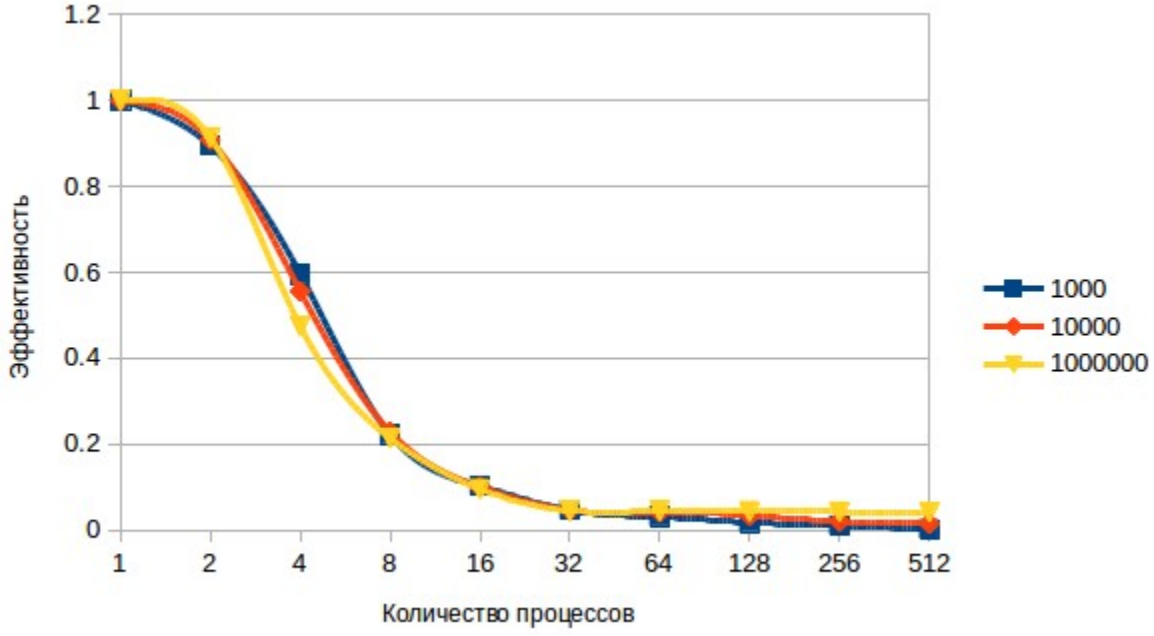




Полученные графики для разреженных матриц:


700
600
500
400
300
200
100
0
Время
1
2
4
8
16
32
64
128
256
512
Количество процессов
1000
10000
1000000


25
20
15
10
5
0
Ускорение
1
2
4
8
16
32
64
128
256
512
Количество процессов
1000
10000
1000000


1.2
1
0.8
0.6
0.4
0.2
0
Эффективность
1
2
4
8
16
32
64
128
256
512
Количество процессов
1000
10000
1000000

5. Выводы.

Сложность представленного алгоритма O(N^2), где N - количество строк матрицы, что в целом соответствует графикам.

Как видно из графиков, использование нескольких процессов для умножения матрицы на вектор позвлояет достаточно сильно ускорить процесс умножения.

Однако, стоит заметить, что ускорение не равно количеству процессов, на которых запущен алгоритм, а меньше его, что можно видеть из графика эффективности. Это можно связать с тем, что при большом количестве процессов возникают дополнительные накладные расходы (в основном на пересылку данных между ними).