Отчет

Программирование на системах с распределенной памятью при помощи MPI

Выполнил:

Студент группы 22.Б15-пу, Колосков Виктор

Задание 1

Формулировка

Разработайте программу для нахождения минимального (максимального) значения среди элементов вектора

Решение

При помощи коллективной операции Broadcast вектор отправляется на все процессы. Далее вычисляется индексы начала и конца набора элементов, в котором соответствующим процессом ищется максимальный и минимальных элемент. Когда всеми процессами найдены локальные минимумы и максимумы, при помощи операции редукции вычисляются глобальные.

Анализ решения

На рисунке 1.1 представлен график зависимости времени выполнения алгоритма от количества процессов. На рисунке 1.2 - график зависимости ускорения программы от количества процессов.

Из графиков видно, что при увеличении количества процессов время работы алгоритма только растет. Начиная с 14 процессов некоторые процессы начинают переходить на второй узел, вследствие чего время работы увеличивается приблизительно в 2 раза.

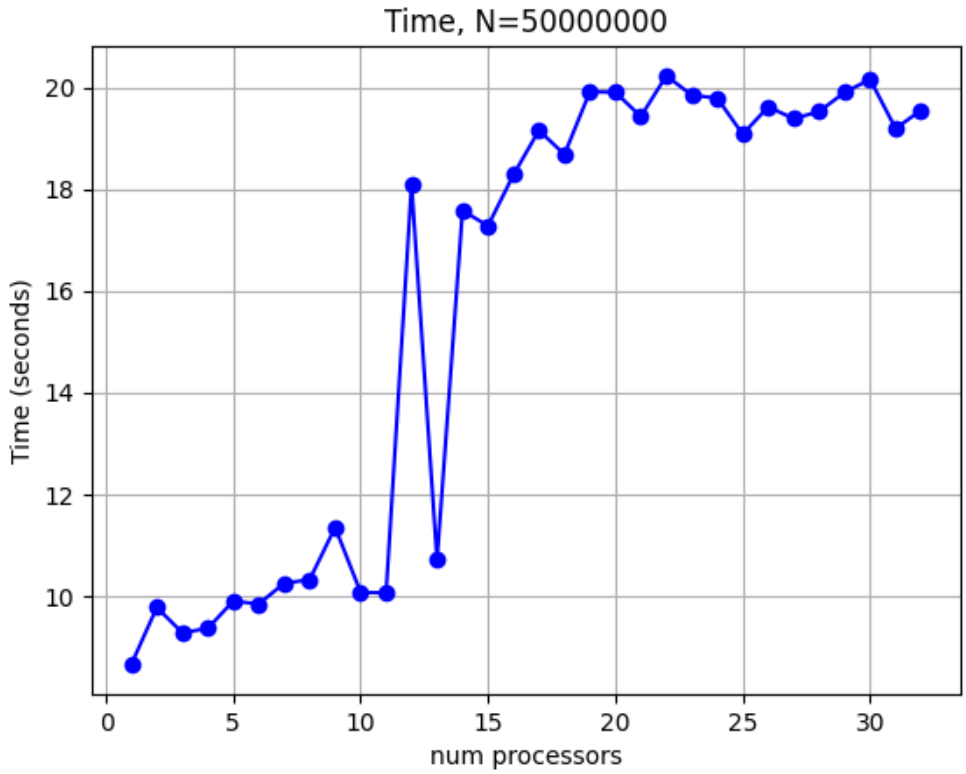

Рисунок 1.1. График зависимости времени работы алгоритма от количества процессов

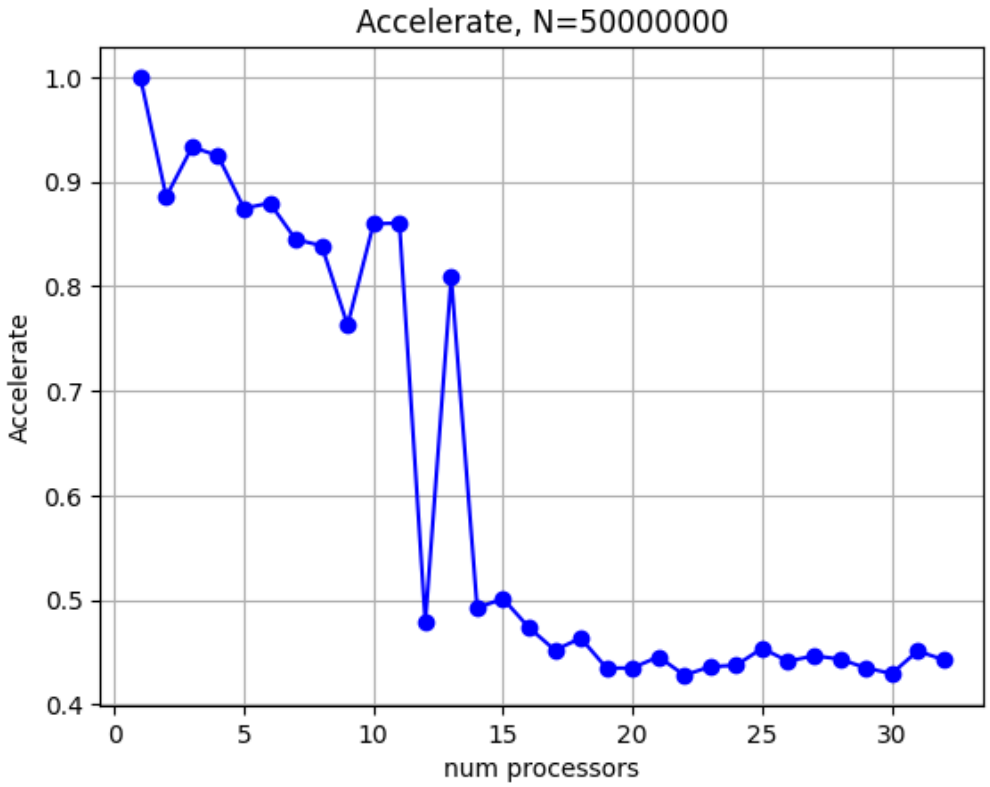

Рисунок 1.2. График зависимости ускорения алгоритма от количества процессов

Задание 2

Формулировка

Разработайте программу для вычисления скалярного произведения двух векторов

Решение

При помощи коллективной операции, отправляем каждому процессу необходимые данные для вычисления части скалярного произведения на каждом процессе. Затем при помощи операции редукции получаем ответ.

Анализ решения

Для решения данной задачи можно воспользоваться для отправки данных либо операцией Broadcast, либо Scatter.

На рисунке 2.1 представлен график зависимости времени выполнения алгоритмов от количества процессов. На рисунке 2.2 - график зависимости ускорения программ от количества процессов. В точках, где ускорения и время работы равно 0 алгоритм не смог дать ответ в связи с нехваткой памяти.

Из графиков видно, что программа, использующая Scatter работает во всех случаях эффективнее, при этом в тестах не было проблем с нехваткой памяти, в отличие от Broadcast. Однако все равно не распараллеленная версия программы работает эффективнее. Стоит также отметить, что при достижении 14 процессов часть вычислений переходит на второй узел. Однако время из-за этого увеличивается не так сильно, как в задании 1.

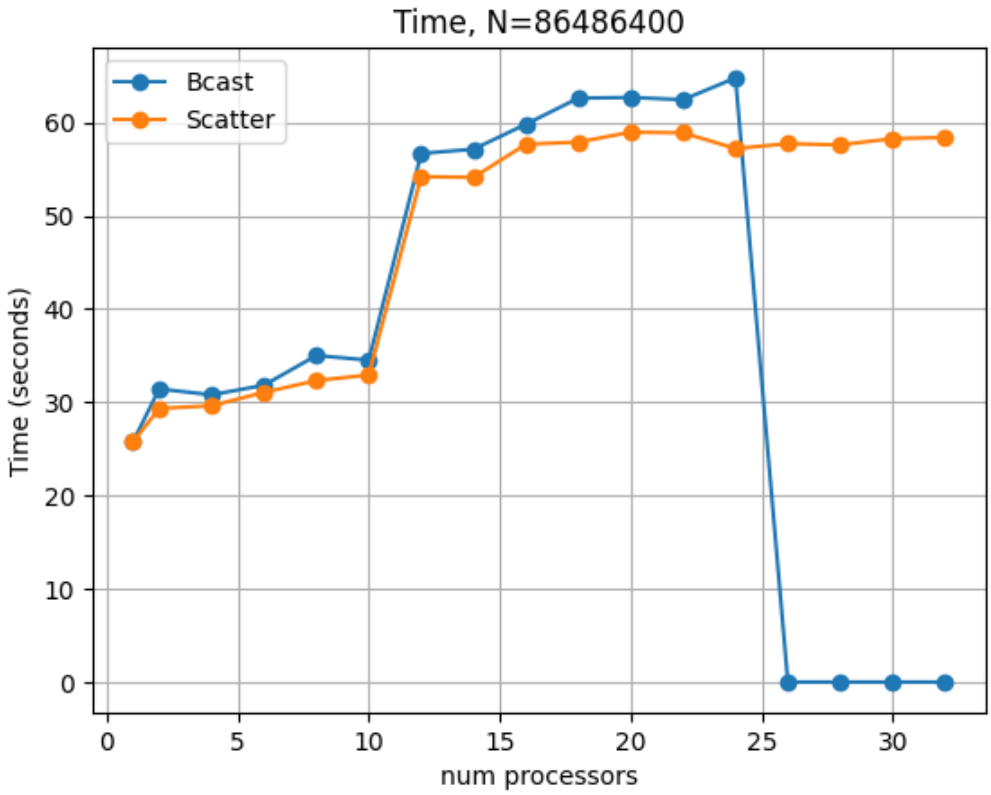

Рисунок 2.1. График зависимости времени работы алгоритма от количества процессов.

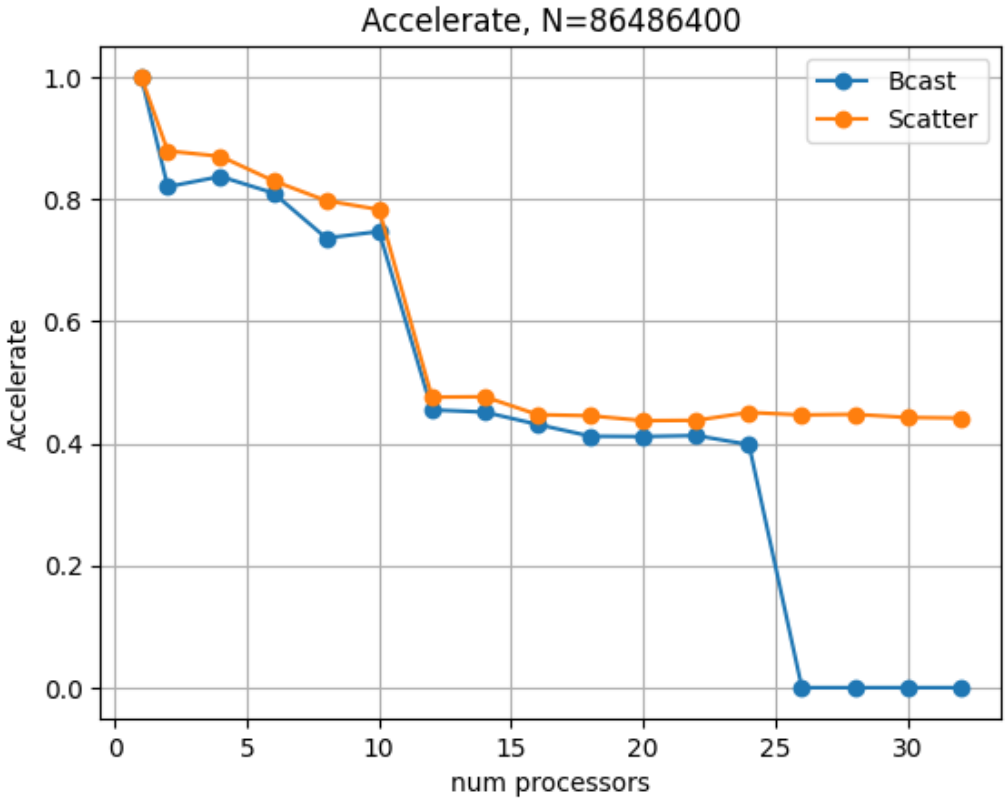

Рисунок 2.2. График зависимости ускорения алгоритма от количества процессов

Задание 3

Формулировка

Разработайте программу, в которой два процесса многократно обмениваются сообщениями длиной n байт. Выполните эксперименты и оцените зависимость времени выполнения операции передачи данных от длины сообщения.

Решение

Создается вектор, который будет передаваться между процессами. Затем от одного процесса вектор передается другому и наоборот. Повторяется так k раз.

Анализ решения

На рисунке 3.1 представлен график зависимости времени выполнения алгоритма от размерности вектора. На рисунке 3.2 - график зависимости ускорения программы от размерности вектора.

Из графика 3.1 можно сделать вывод, что, в целом время передачи вектора линейно зависит от его размера. Стоит также заметить, что при размере вектора: 100, 1000, 10000 наблюдается сильный всплеск ускорения. Это может быть связано с особенностями работы MPI_Send. При достаточно малом размере вектора, он сначала копируется в буфер, а только потом отправляется другому процессу. Если вектор довольно большой, то он может не поместиться в буфер и тогда передача происходит напрямую, без копирования, вследствие чего время работы меньше.

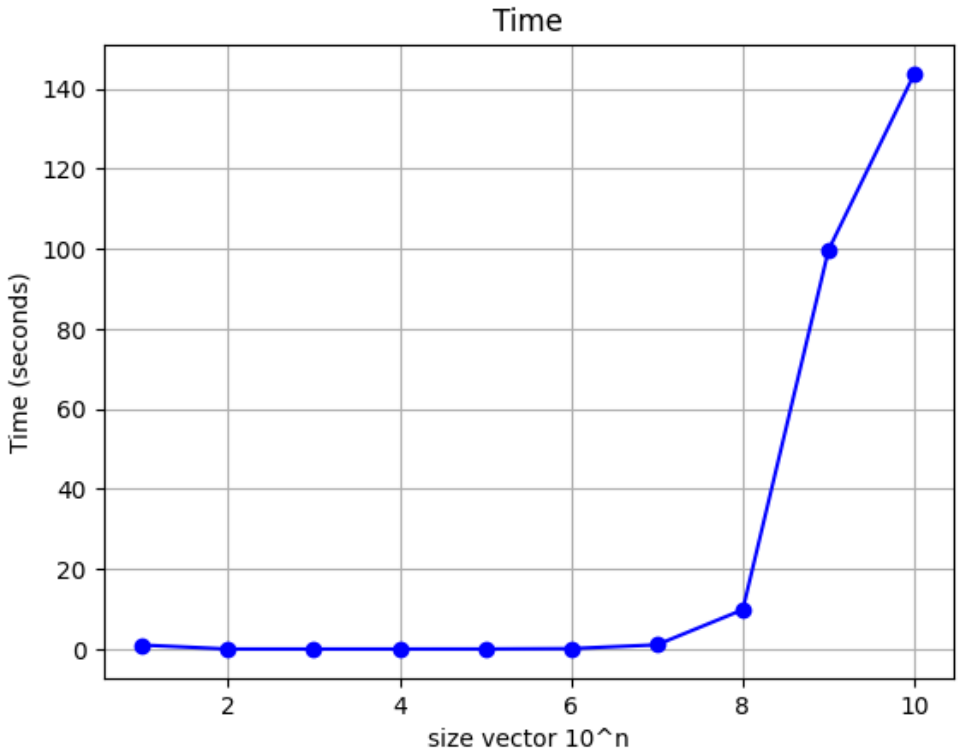

Рисунок 3.1. График зависимости времени работы алгоритма от размерности вектора.

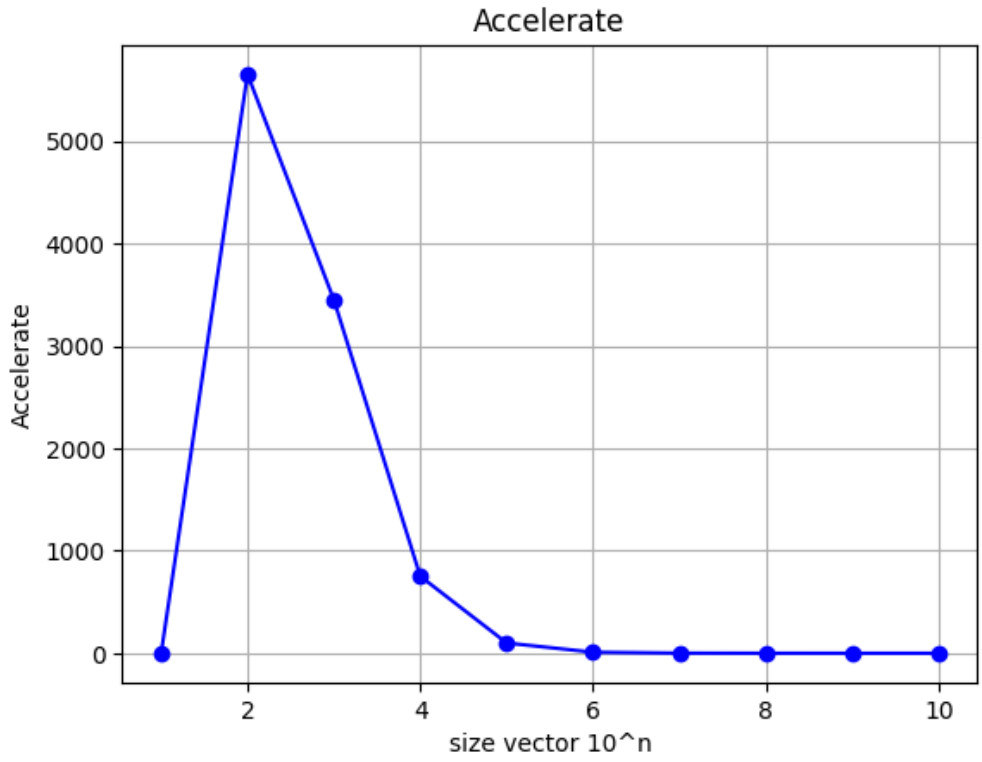

Рисунок 3.1. График зависимости ускорения алгоритма от размерности вектора.

Задание 4

Формулировка

Реализуйте алгоритмы умножения матриц: 1) простейший (ленточный); 2) более сложный на выбор (алгоритм Фокса, алгоритм Кэннона или иной). Сравните производительность работы программ на основе разных алгоритмов

Решение

В качестве более сложного алгоритма взят стандартный алгоритм Фокса (http://www.hpcc.unn.ru/?dir=1034). Для ленточного алгоритма применена небольшая модификация: вторая матрица считывается как транспонированная. Благодаря этому происходит умножение не строки на столбец, а строки на строку, вследствие чего упрощается передача данных между процессами.

Анализ решения

На рисунке 4.1 представлен график зависимости времени выполнения алгоритмов от количества процессов. На рисунке 4.2 - график зависимости ускорения программ от количества процессов.

Из графиков можно сделать вывод, что в целом ленточный алгоритм работает быстрее и лучше, чем алгоритм Фокса. Однако при увеличении количества процессов ускорения ленточного алгоритма падает быстрее, чем алгоритма Фокса, что ускорения подчиняется практически линейному закону. Вследствие чего, возможно, что при более высокой размерности матрицы и большем количестве процессов алгоритм Фокса будет работать быстрее.

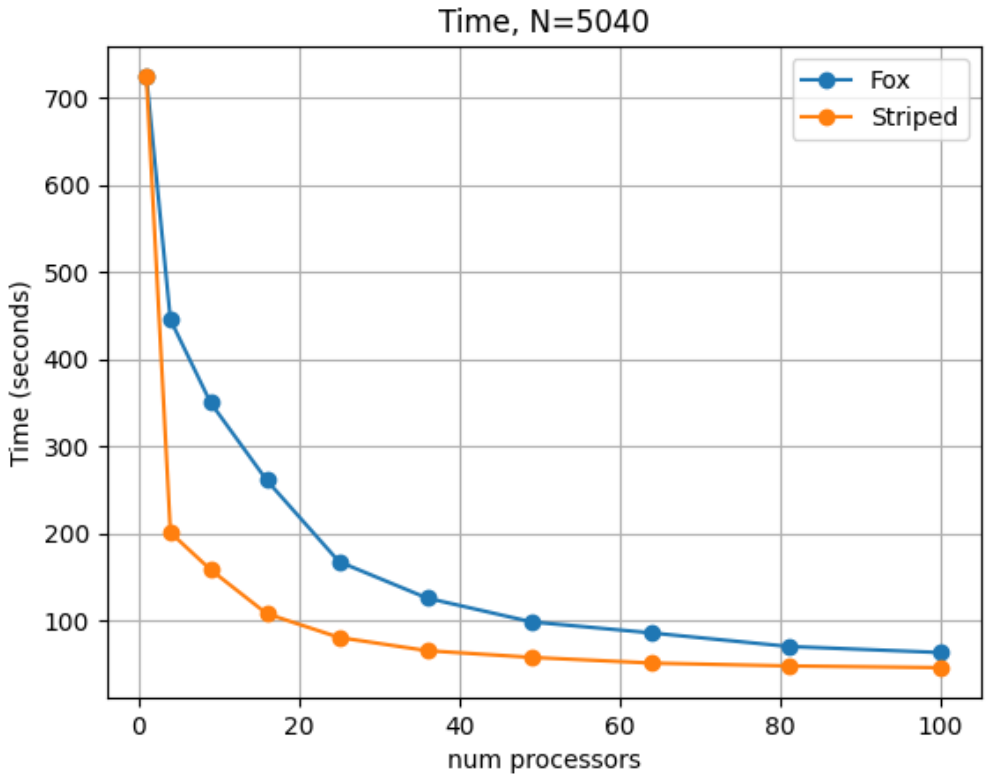

Рисунок 4.1. График зависимости времени работы алгоритма от количества процессов.

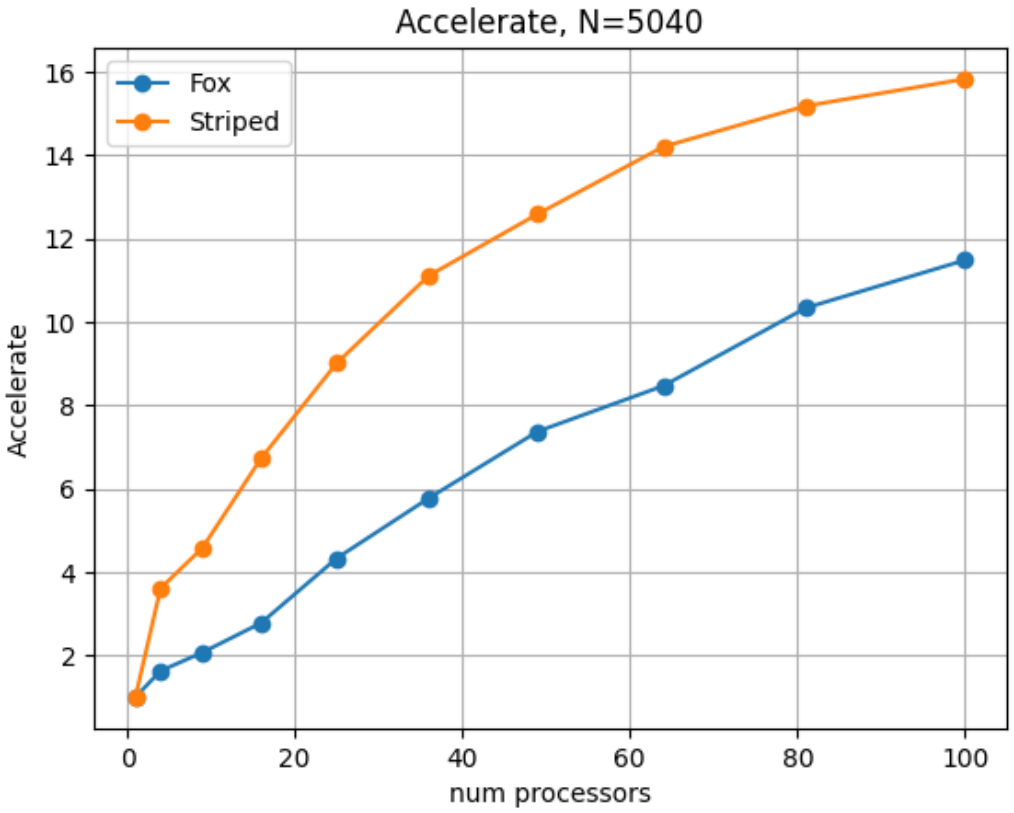

Рисунок 4.2. График зависимости ускорения алгоритма от количества процессов

Задание 5

Формулировка

Реализуйте параллельную программу с конфигурируемым объемом вычислений и коммуникаций (разные объемы вычислений можно эмулировать, например, с помощью вызова sleep на разное количество микросекунд; объемы коммуникаций регулировать размером и количеством пересылаемых сообщений). Исследуйте производительность (ускорение) программы при различном балансе вычислений и коммуникаций

Решение

В качестве имитации вычислений используется функция, в которая замораживает процесс на определенное количество секунд. Для имитации коммуникаций используется операция Broadcast. Для подсчета ускорения необходимо выбрать эталонный элемент, относительно которого рассчитывается ускорение. Таковым элементом является элемент с объемом вычислений в 1 секунду и объемом коммуникаций в 2*10^6 элементов.

Анализ решения

На рисунках 5.1, 5.2, 5.3, 5.4 представлены графики зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 8, 16, 24, 32 процессах соответственно.

Из графиков можно сделать вывод, что при небольшом количестве процессов на скорость работы программы в основном влияет объем вычислений. Однако, чем больше число процессов, тем более сильнее влияют коммуникационные задержи.

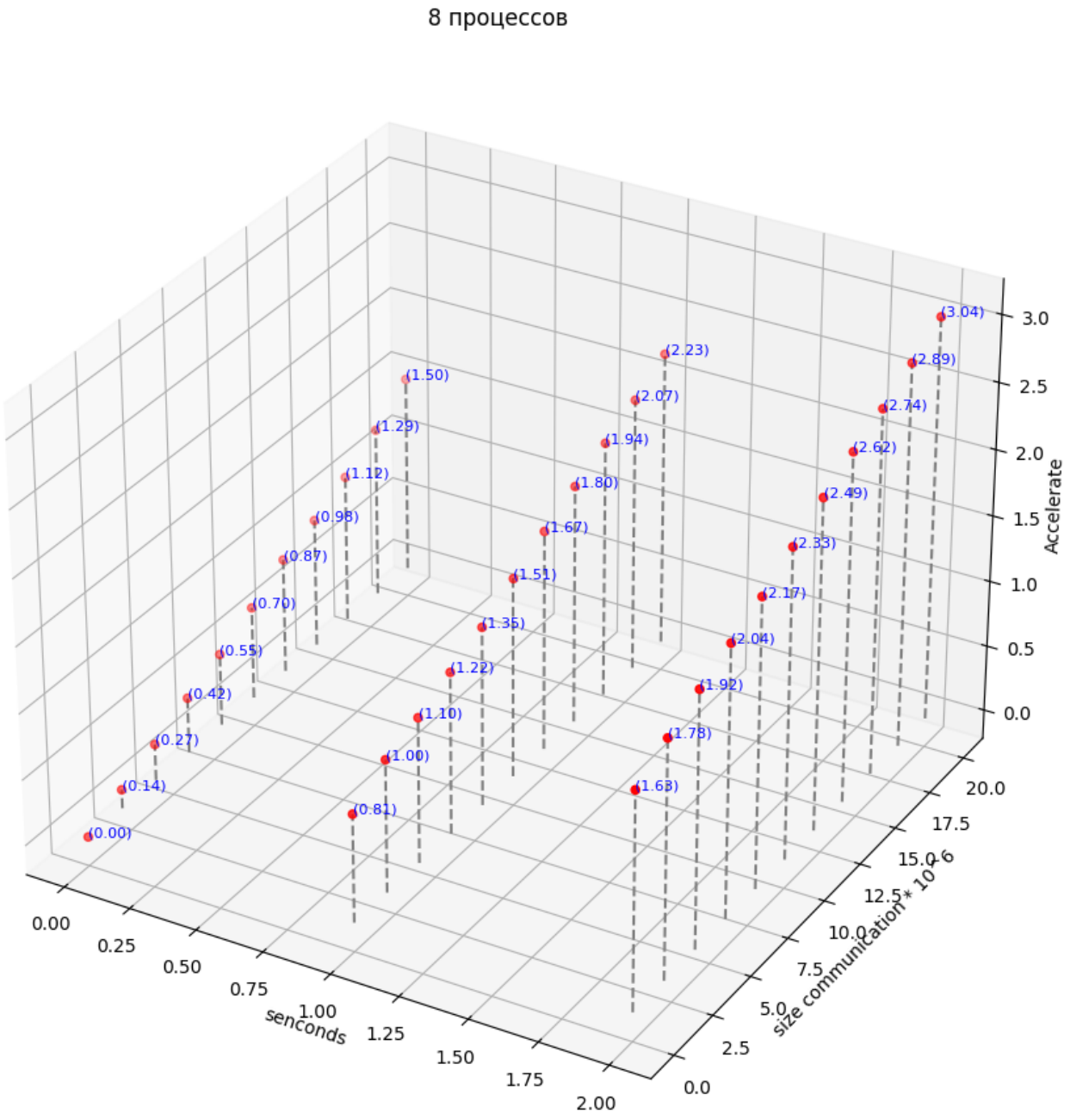

Рисунок 5.1 график зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 8 процессах.

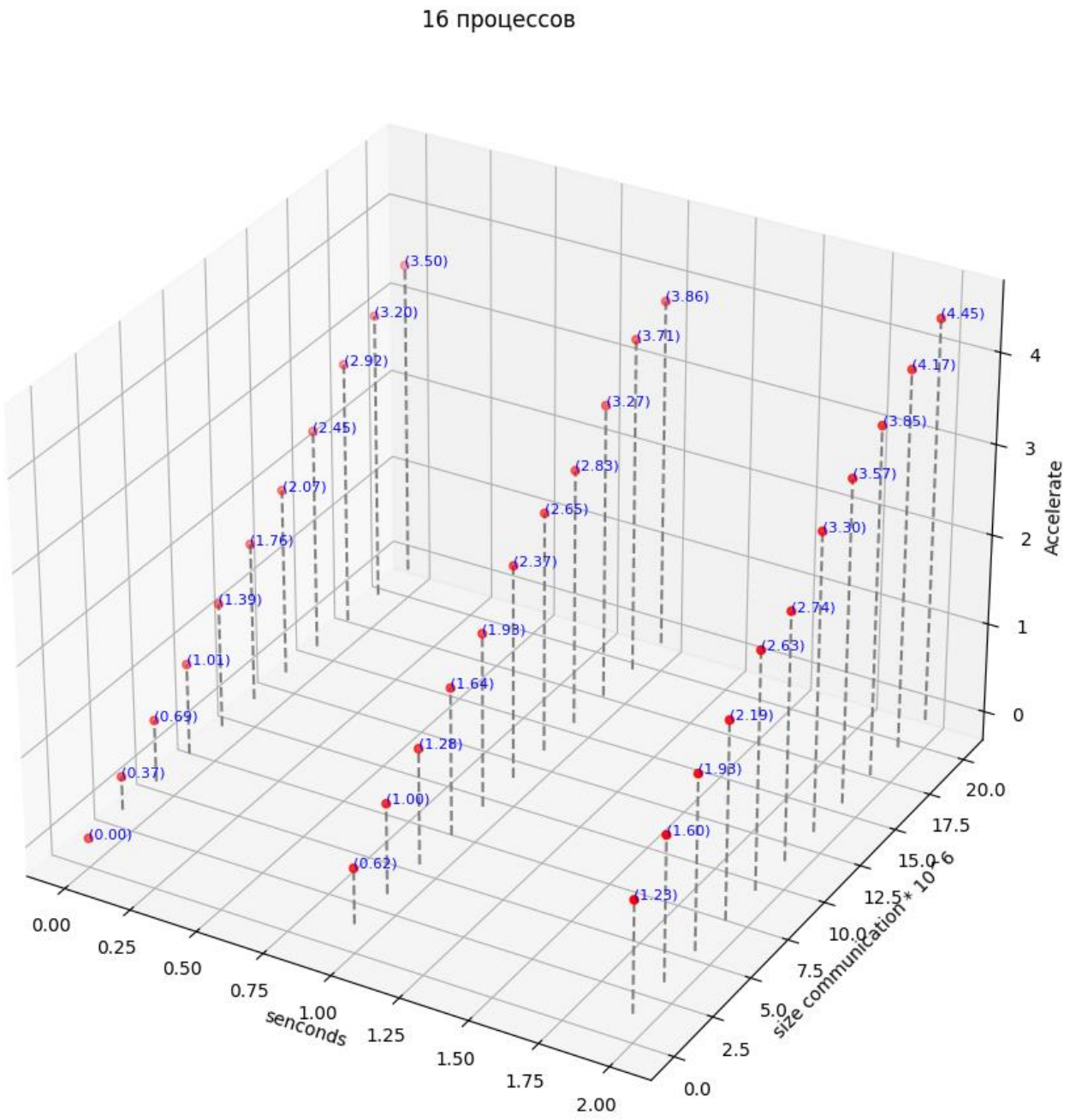

Рисунок 5.2 график зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 16 процессах.

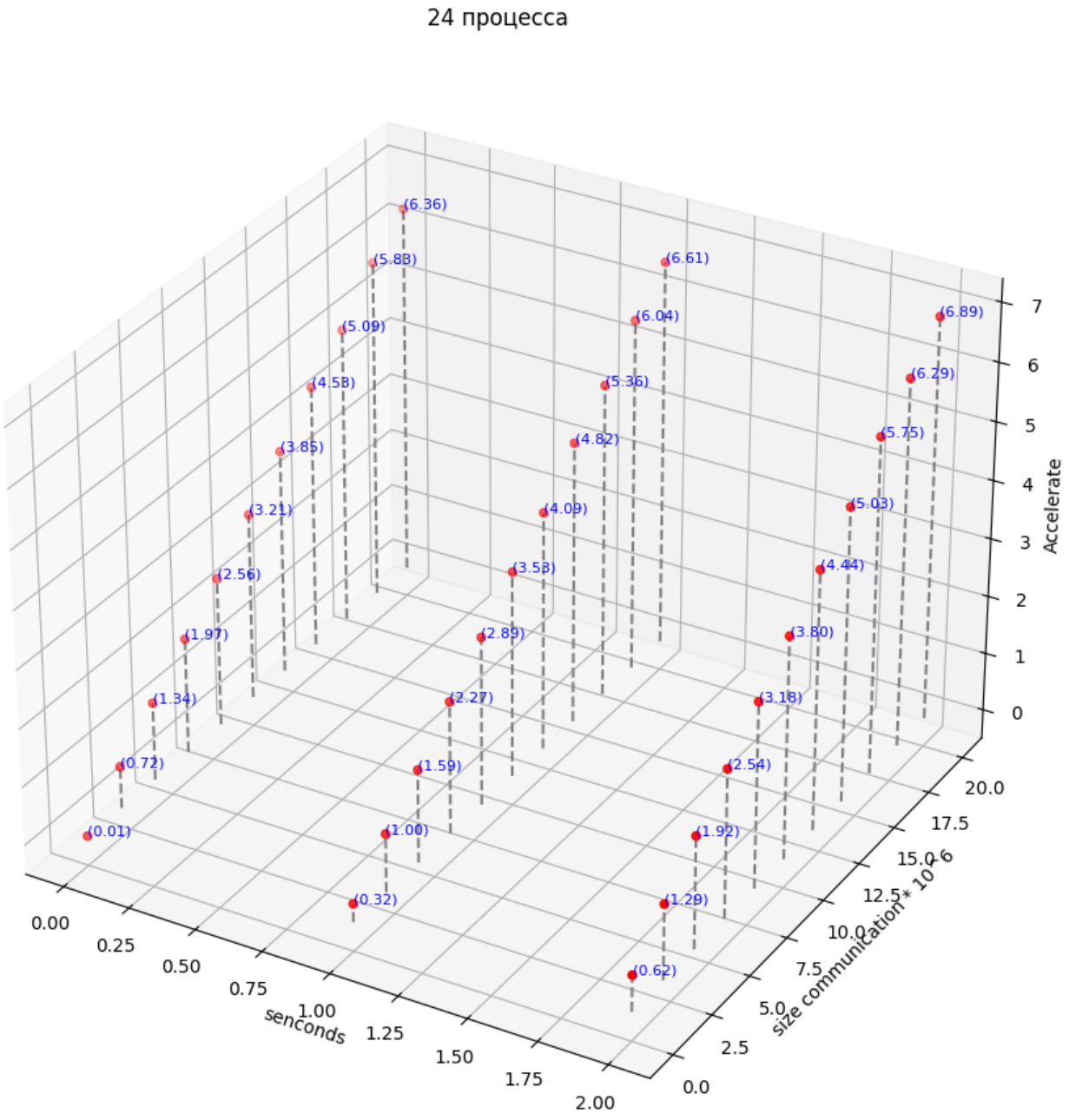

Рисунок 5.3 график зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 24 процессах.

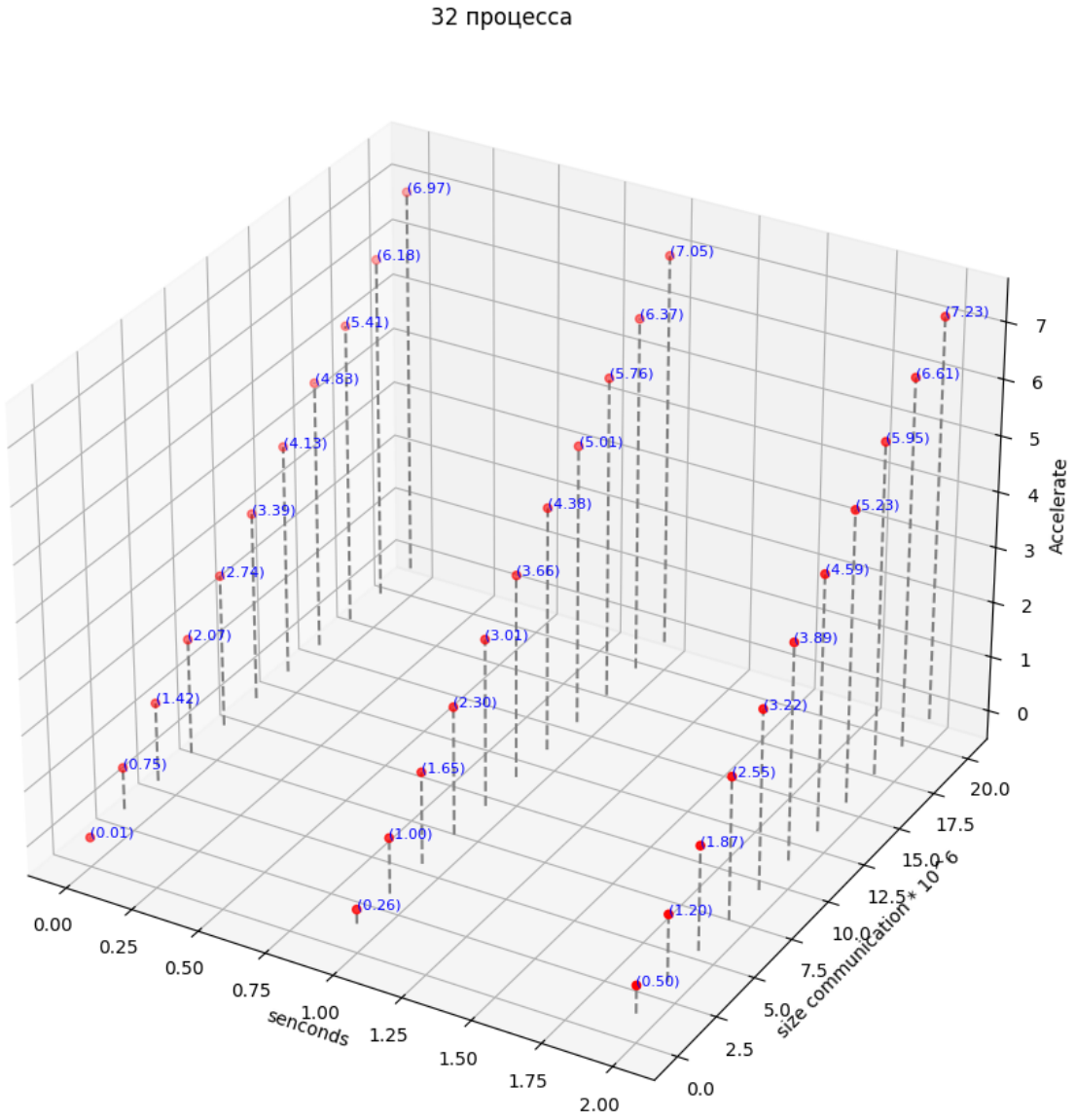

Рисунок 5.4 график зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 32 процессах.

Задание 6

Формулировка

Подготовьте варианты ранее разработанных программ (*достаточно одной из программ, например из задания 4*) с разными режимами выполнения операций передачи данных (синхронный, по готовности, буферизованный). Сравните времена выполнения операций передачи данных при разных режимах работы.

Решение

В качестве программы взят ленточный алгоритм из задания 4. В нем, так, где необходимо заменены операции MPI_Send на соответствующие. Для буферизованного режима найден оптимальный объем буфера путем запуска программы с различным размером буфера на 2-х узлах по 8 процессов на каждом. Оптимальным оказался буфер размером: 3 * (количества процессов * размер матрицы на 1-м процессе * размер матрицы на всех процессах * размера типа данных) байт

Анализ решения

На рисунке 4.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 4.2 - график зависимости ускорения программ от количества процессов.

Из графиков можно сделать вывод, что буферизованный режим является наименее эффективным в данном случае. Это может быть связано с тем, что передаваемые сообщения слишком большие. Синхронный и простой режимы практически одинаковы по времени работы, однако исходя из графика ускорения, можно сделать вывод, что простой режим более эффективен.

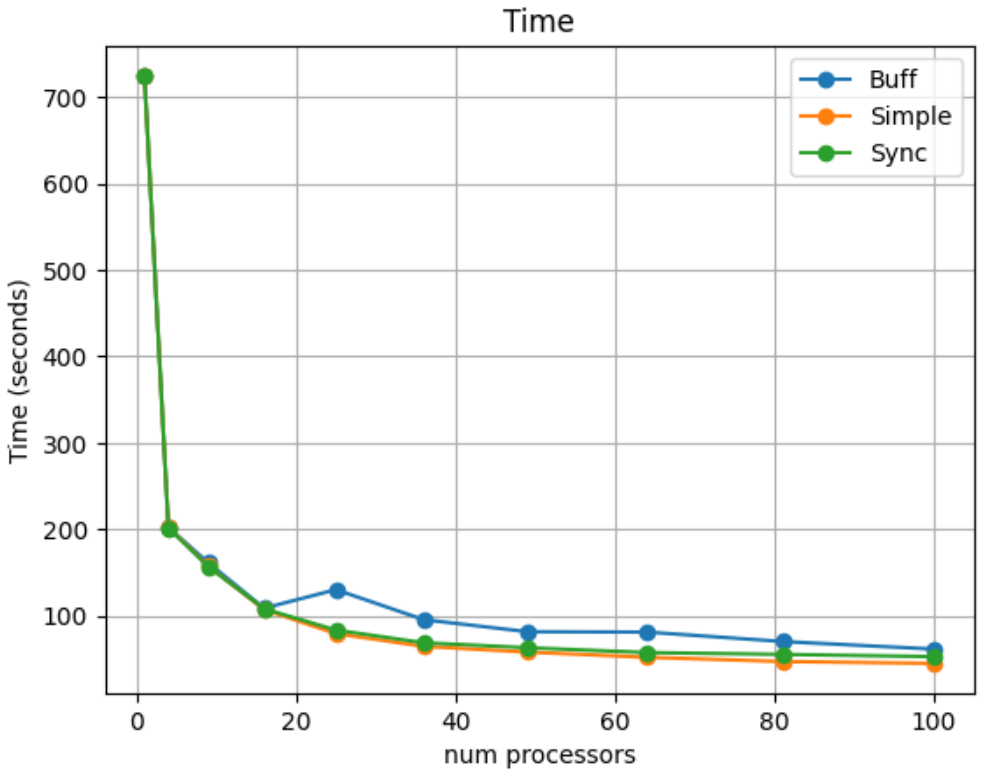

Рисунок 6.1. График зависимости времени работы алгоритма от количества процессов.

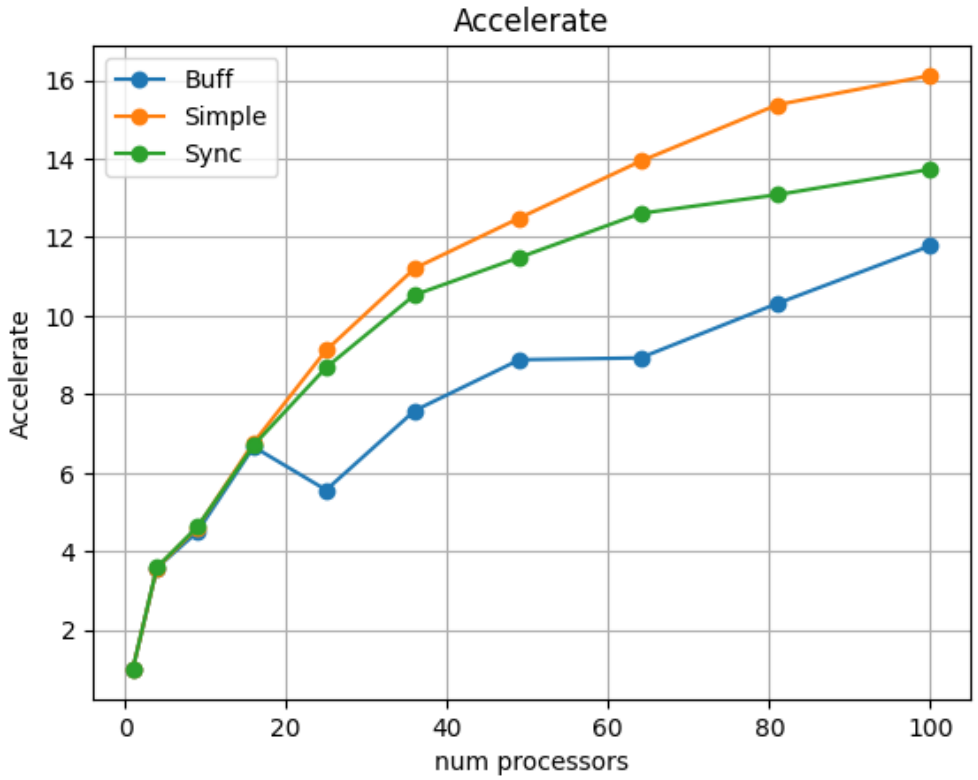

Рисунок 4.2. График зависимости ускорения алгоритма от количества процессов

Задание 7

Формулировка

Подготовьте варианты ранее разработанных программ (*достаточно одной из программ, например из задания 5*) с использованием неблокирующего способа выполнения операций передачи данных. Оцените необходимое количество вычислительных операций для того, чтобы полностью совместить передачу данных и вычисления.

Решение

В качестве программы взята реализация алгоритма из задания 5, в которой MPI_Send и MPI_Recv заменены на MPI_Isend и MPI_Irecv соответственно.

Анализ решения

На рисунках 7.1, 7.2, 7.3, 7.4 представлены графики зависимости ускорения работы программы от объема вычислений и объема коммуникаций при 8, 16, 24, 32 процессах соответственно.

Из графиков можно сделать вывод, что при небольшом количестве процессов на скорость работы программы в основном влияет объем вычислений. Однако, чем больше число процессов, тем более сильнее влияют коммуникационные задержи.

Количество вычислительных операций определяется по формуле:

$$N = \left[\frac{T_{data}}{T_{comp}}\right] (1)$$

Где [ ] - округление вверх, T_{data} - время затрачиваемое на пересылку данных и ▯ - время, затрачиваемое на вычисления. Оценим для $T_{comp} = 1$ сек и объеме вычислений 2*10^6 на 24процессах, $T_{data} = 2.54$ секТогда $N = 3$

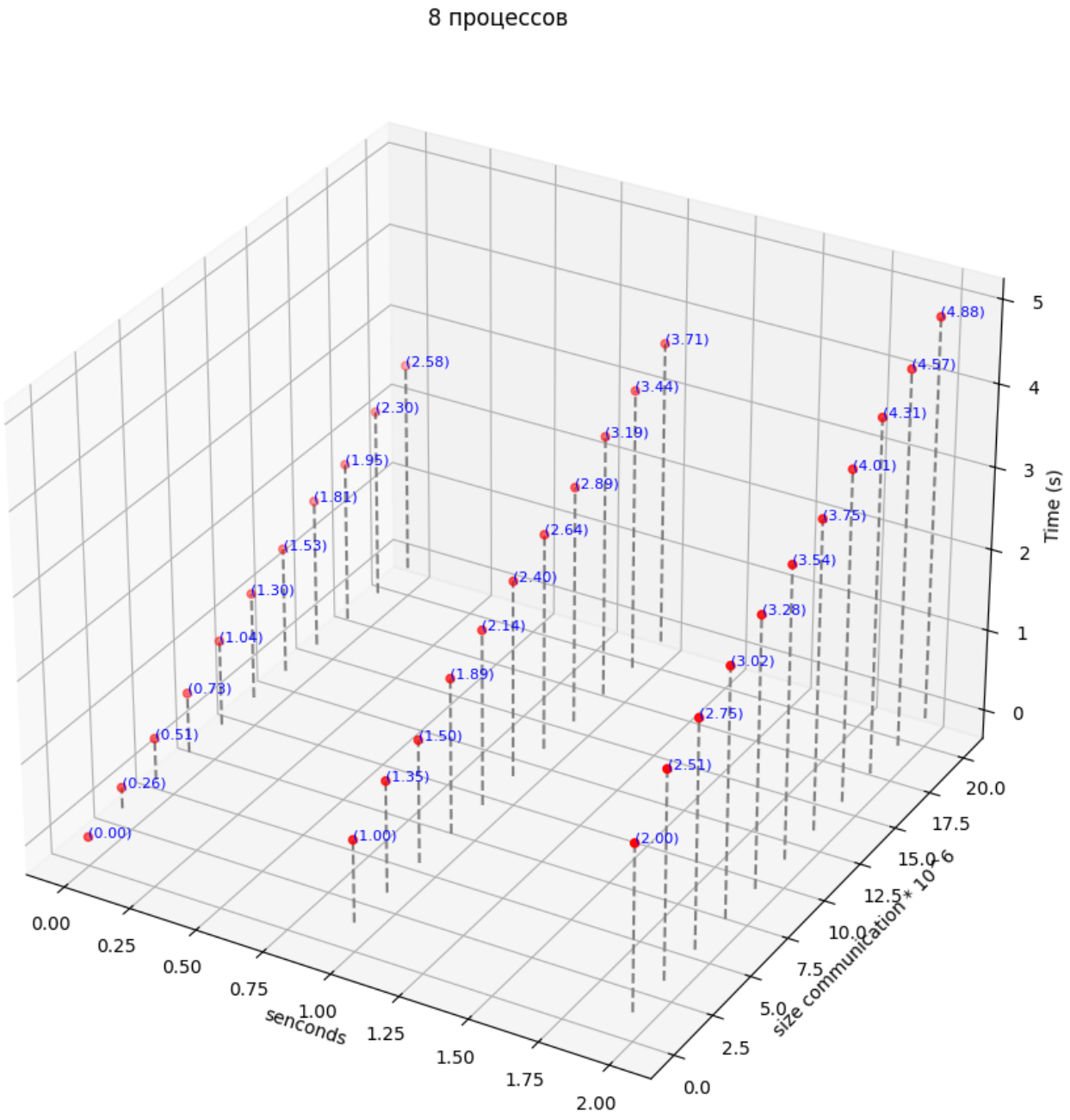

Рисунок 5.1 график зависимости времени работы программы от объема вычислений и объема коммуникаций при 8 процессах.

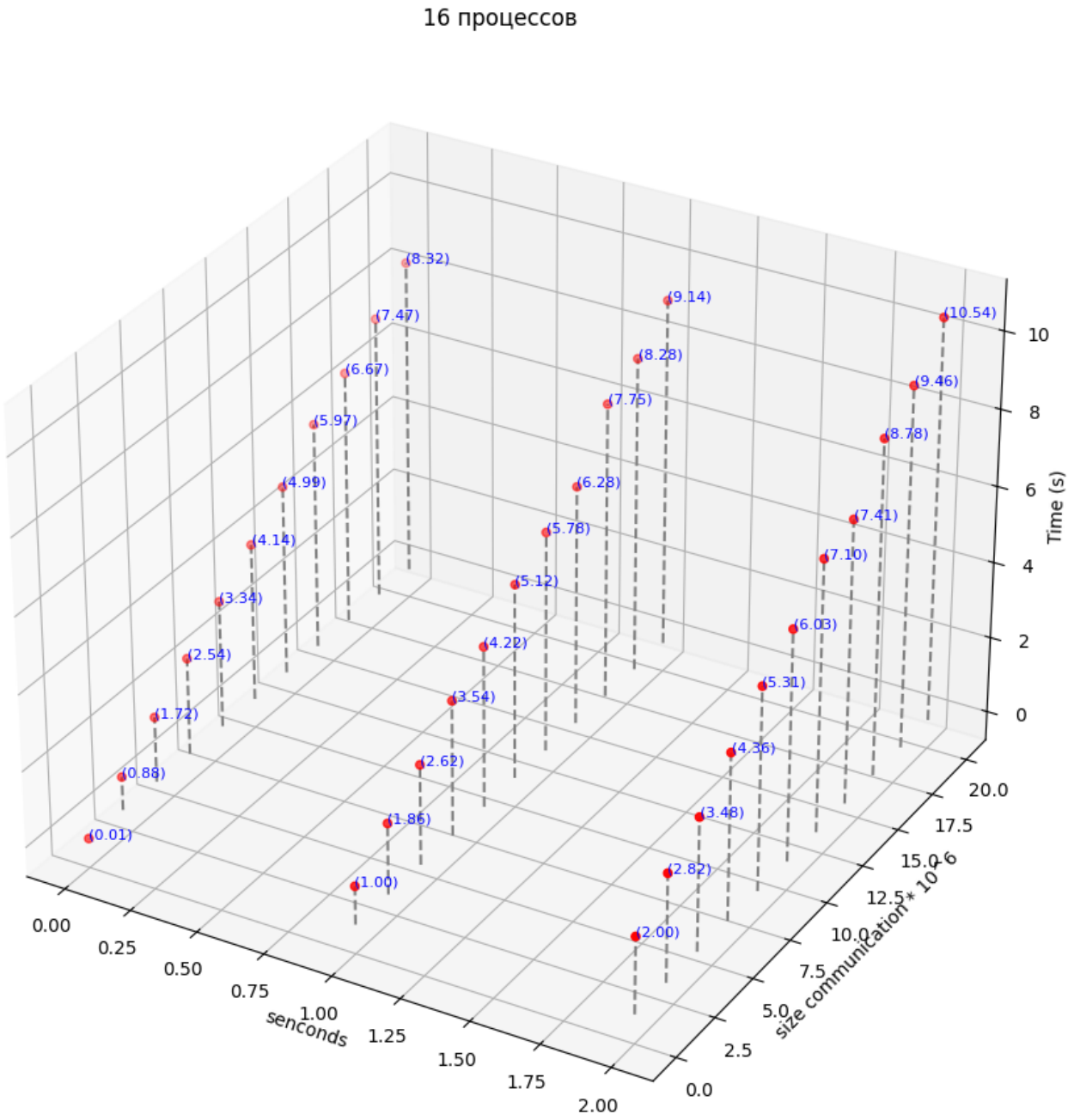

Рисунок 5.2 график зависимости времени работы программы от объема вычислений и объема коммуникаций при 16 процессах.

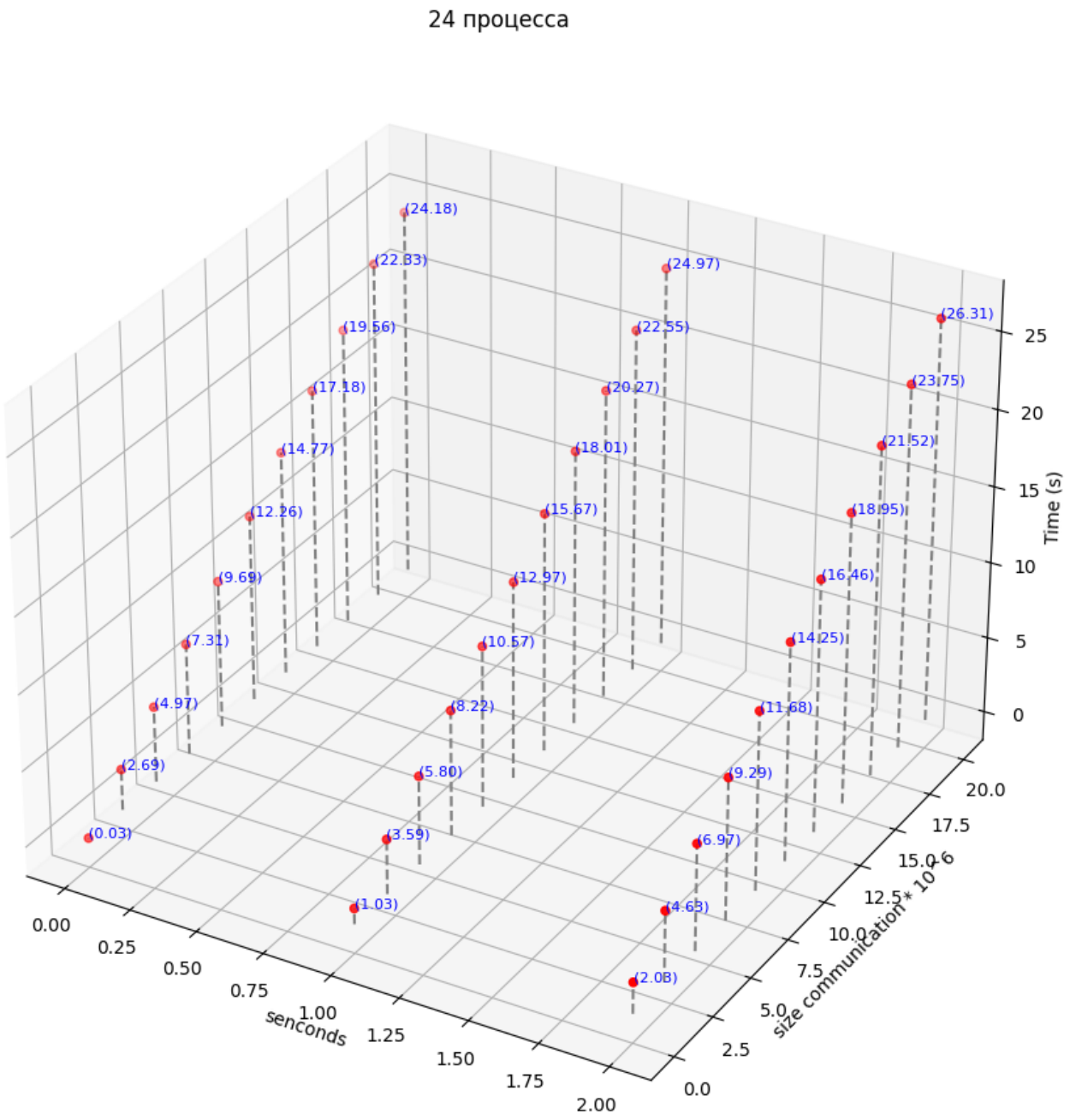

Рисунок 5.3 график зависимости времени работы программы от объема вычислений и объема коммуникаций при 24 процессах.

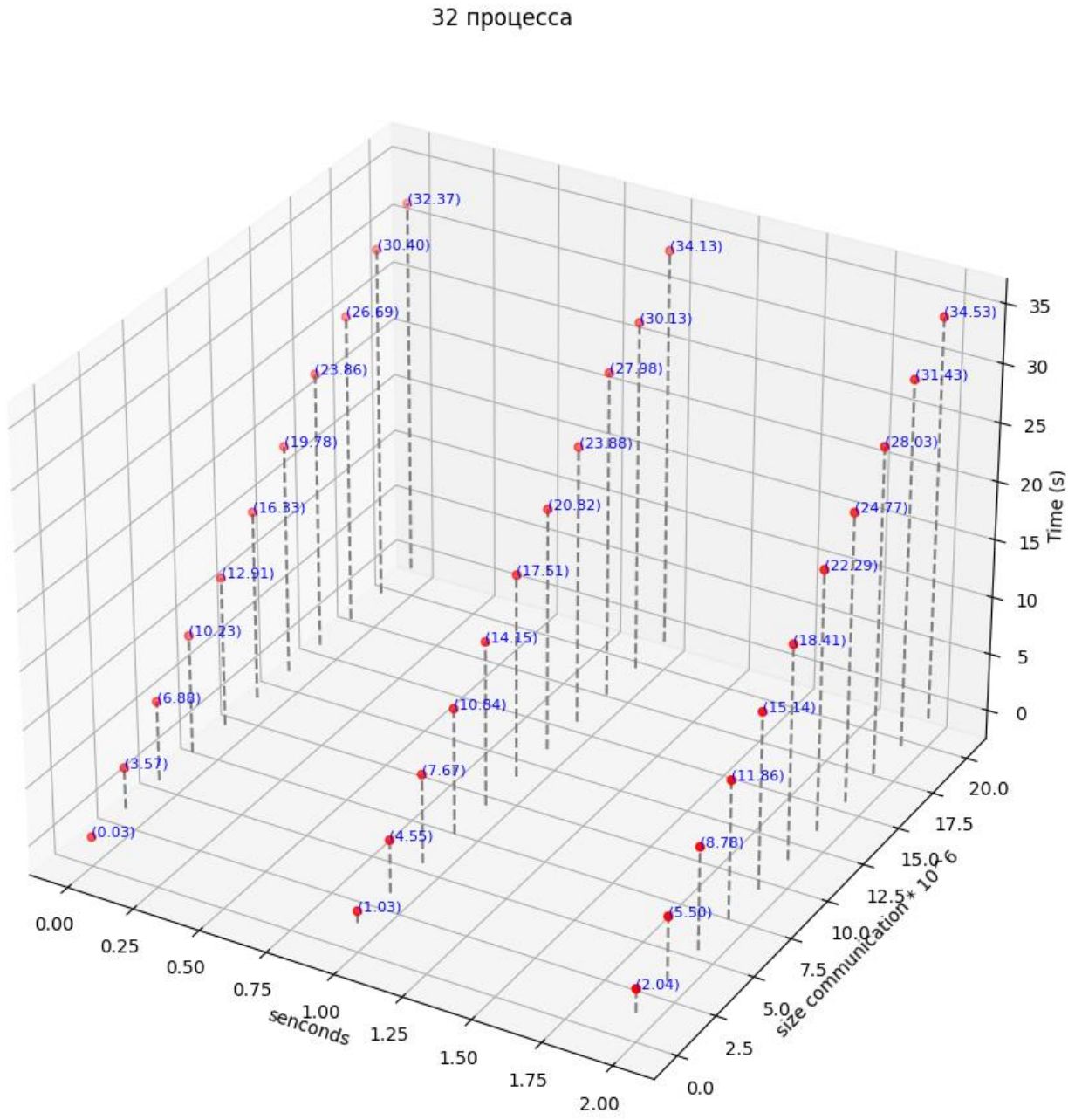

Рисунок 5.4 график зависимости времени работы программы от объема вычислений и объема коммуникаций при 32 процессах.

Задание 8

Формулировка

Модифицируйте программу из задания 3, используя операцию одновременного выполнения передачи и приема данных. Сравните результаты вычислительных экспериментов

Решение

Там, где это необходимо, в программе 3 две операции: MPI_Send и MPI_Recv заменяются на одну: MPI_Sendrecv или MPI_Sendrecv_replace.

Анализ решения

На рисунке 8.1 представлен график зависимости времени выполнения программ от количества процессов.

Из графика времени можно сделать вывод, что при небольшом объеме вектора сильной разницей между операциями нет, однако при больших векторах наиболее эффективной будет MPI_Sendrecv_replace. К тому же при этой операции памяти расходуется меньше так-как используется для Send и Recv один и тот же буфер.

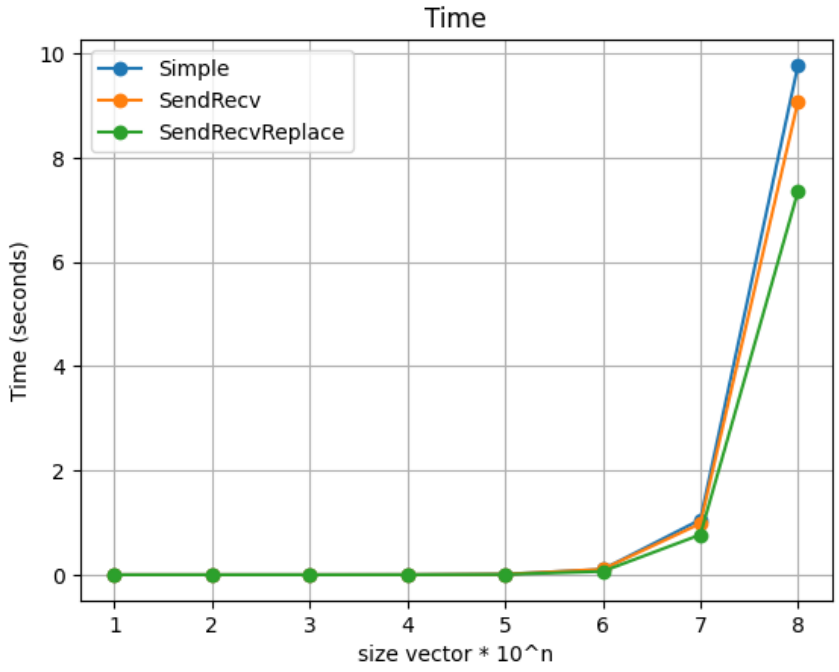

Рисунок 1.1. График зависимости времени работы программ от количества процессов

Задание 9

Формулировка

Разработайте реализации коллективных операций (Broadcast, Reduce, Scatter, Gather, AlltoAll, AllGather) при помощи парных обменов между процессами. Выполните вычислительные эксперименты и сравните времена выполнения разработанных программ и функций MPI для коллективных операций.

Решение

1.Broadcast

При помощи неблокирующих операций инициализируем операции отправки сообщений, тем самым значительно уменьшая задержки. Остальные процессы принимают данные.

2.Reduce

Пусть на каждом процессе у нас есть вектор. Все векторы одного размера и для них нужно поэлементно сделать операцию редукции. При помощи реализованной операции Broadcast на каждом ппоцессе оказываюстя все векторы. Затем на каждом процессе для каждой группы элементов производится операция редукции. На последнем шаге все полученные элементы отправляются на процесс root.

3.Scatter

Инициализируются неблокирующие отправки на процессе root. В каждую отправку передается один и тот же буфер с необходимым смещением на начальный элемент. На остальных процессов инициализируется получение.

4.Gather

На процессе root инициализируются неблокирующие операции получения. В них передается один и тот же буфер с необходимым

смещением на начальный элемент. На остальных процессах инициализируются операции отправки.

5.AlltoAll

В каждом из процессов инициализируются неблокирующие операции отправки и получения. На каждом процессе для всех операций отправки и получения используются только по одному буферу. Передаются они в функции со смещением на начальный элемент.

6.AllGather

Всего n (количество процессов) этапов отправки и получения. На этапе i с процесса с рангом k передается данные при инициализации на процесс с рангом (k + i) % n. И принимаются данные от процесса с рангом (k + n – i) % n.

Анализ решения

1.Broadcast

На рисунке 9.1.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.1.2 - график зависимости ускорения программ от количества процессов.

Из графиков можно сделать вывод, что моя реализация Broadcast работает быстрее при небольшом количестве процессов, однако затем ее эффективность резко падает.

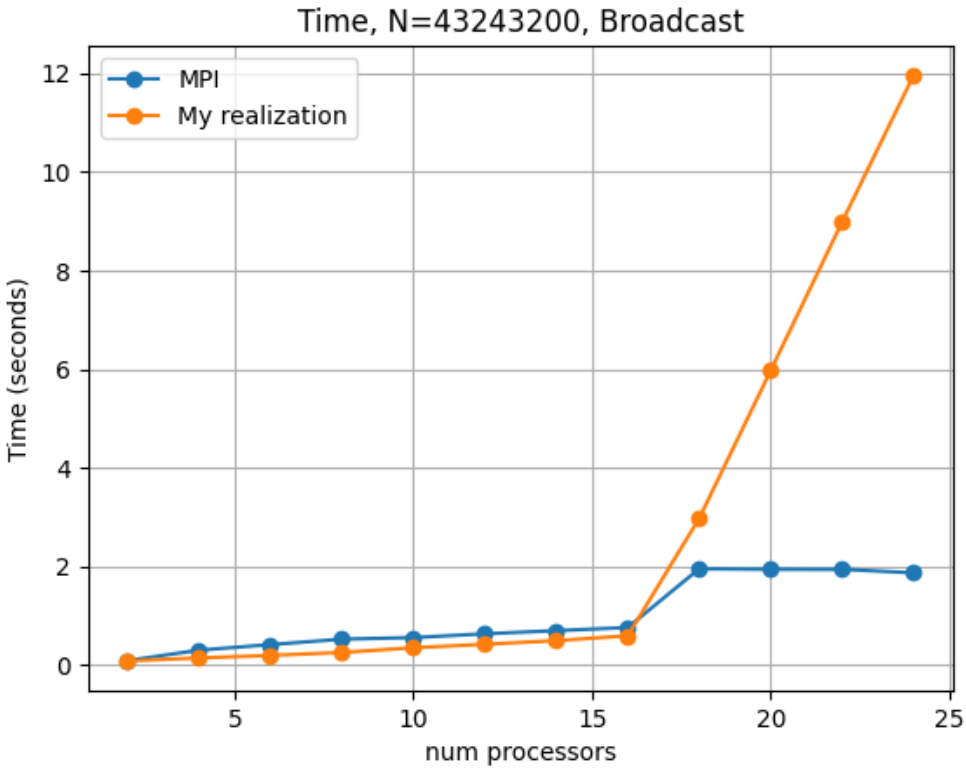

Рисунок 9.1.1. График зависимости времени работы программ от количества процессов

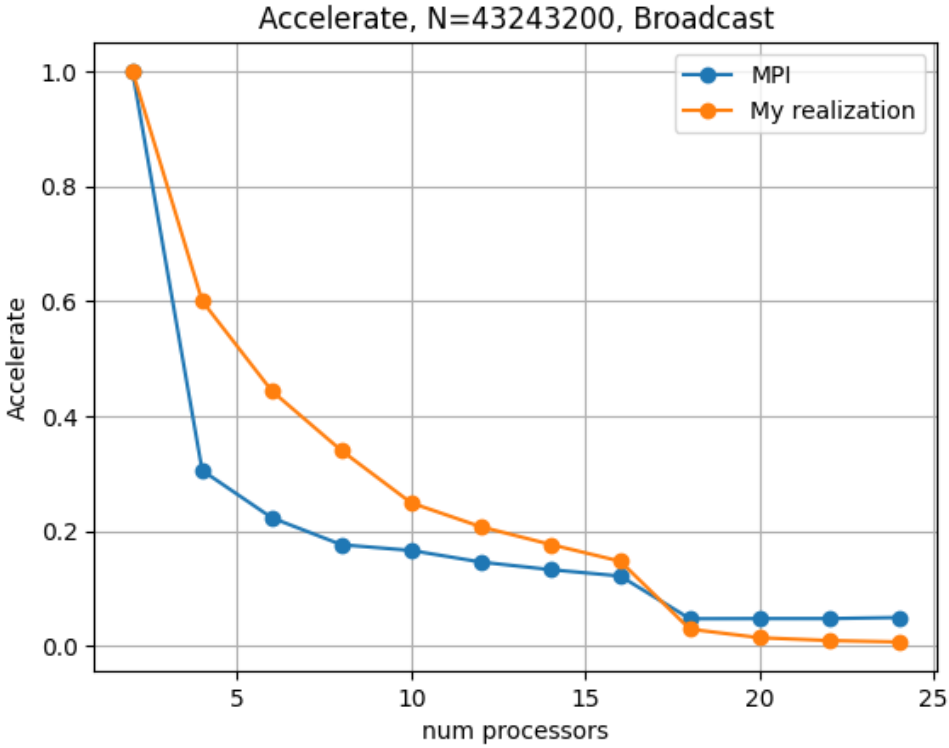

Рисунок 9.1.2. График зависимости ускорения программ от количества процессов

2.Reduce

На рисунке 9.2.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.2.2 - график зависимости ускорения программ от количества процессов.

Из графиков можно сделать вывод, что моя реализация неэффективна при любом количестве процессов.

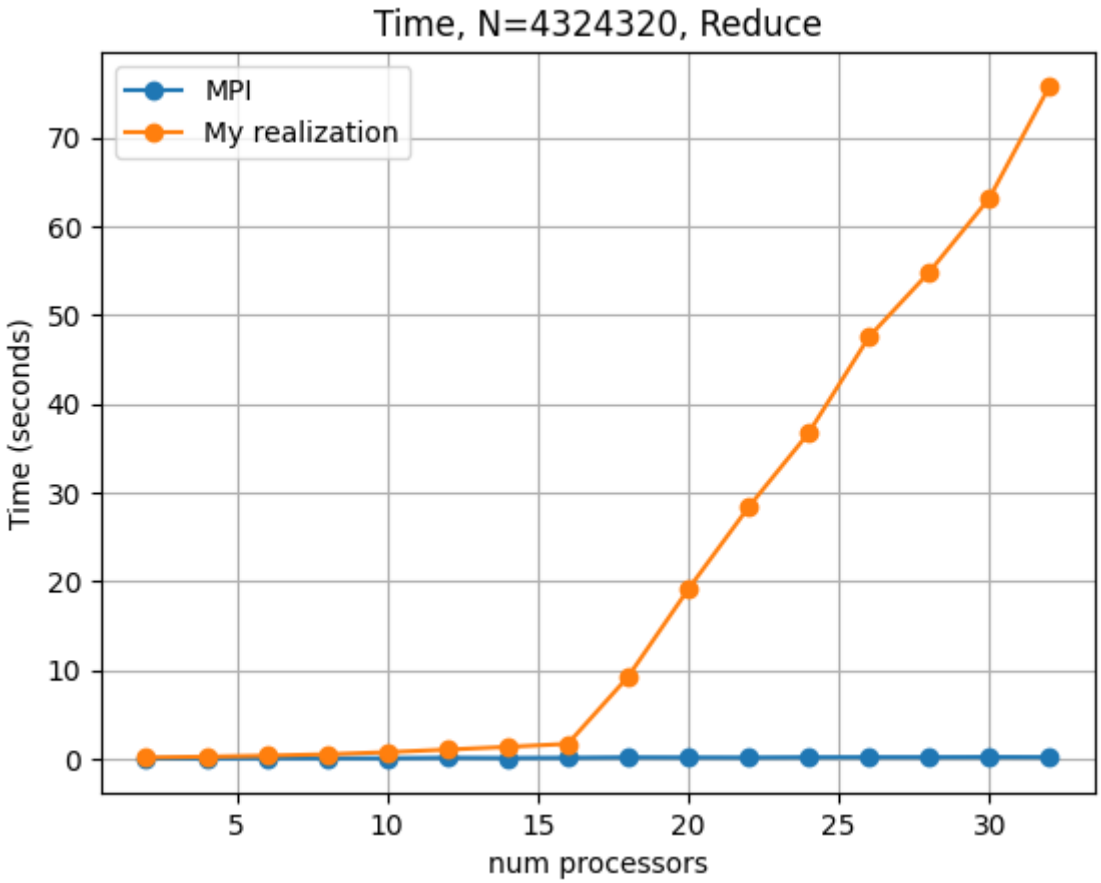

Рисунок 9.2.1. График зависимости времени работы программ от количества процессов

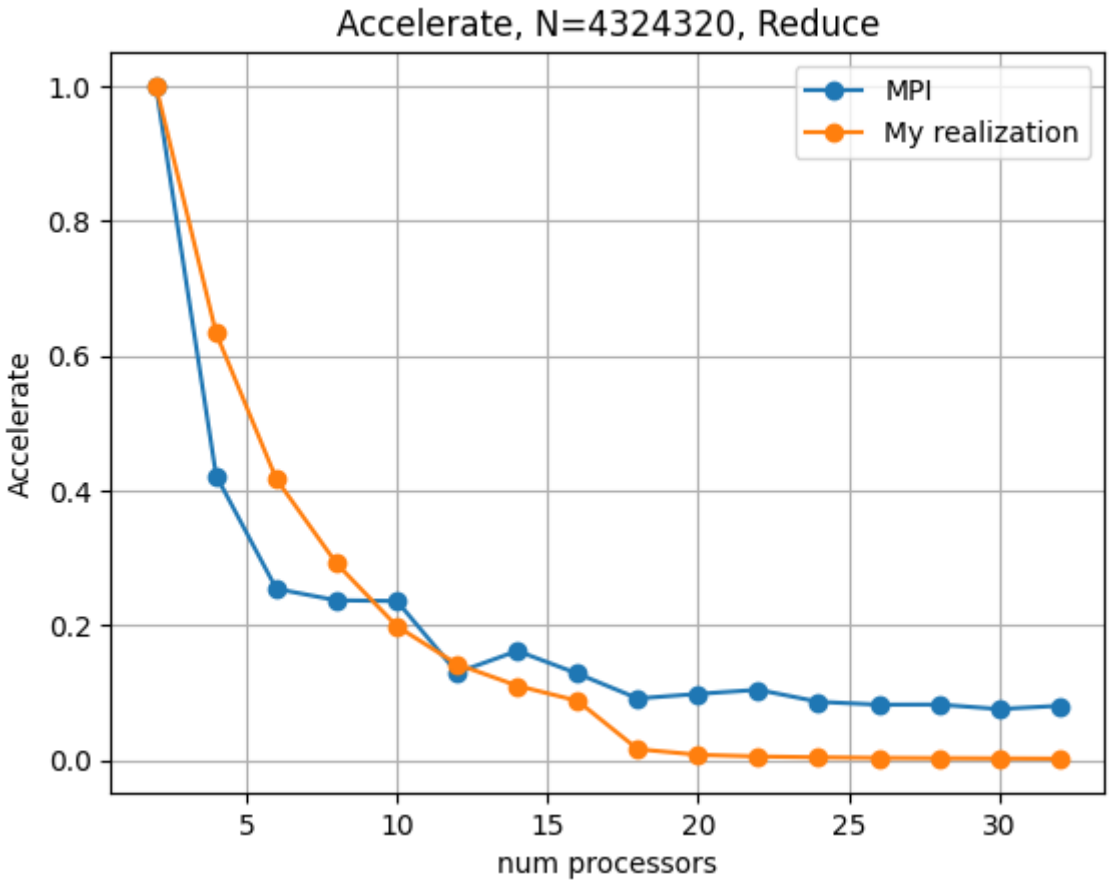

Рисунок 9.2.2. График зависимости ускорения программ от количества процессов

3.Scatter

На рисунке 9.3.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.3.2 - график зависимости ускорения программ от количества процессов.

Из графика ускорения, что при небольшом количестве процессов моя реализация достигает ускорения практически в 2 раза. Затем ускорения быстро падает и при большом количестве процессов немного выше, чем у реализации MPI. При этом, как показано на рисунке 9.3.1 практически во всех тестах время работы меньше.

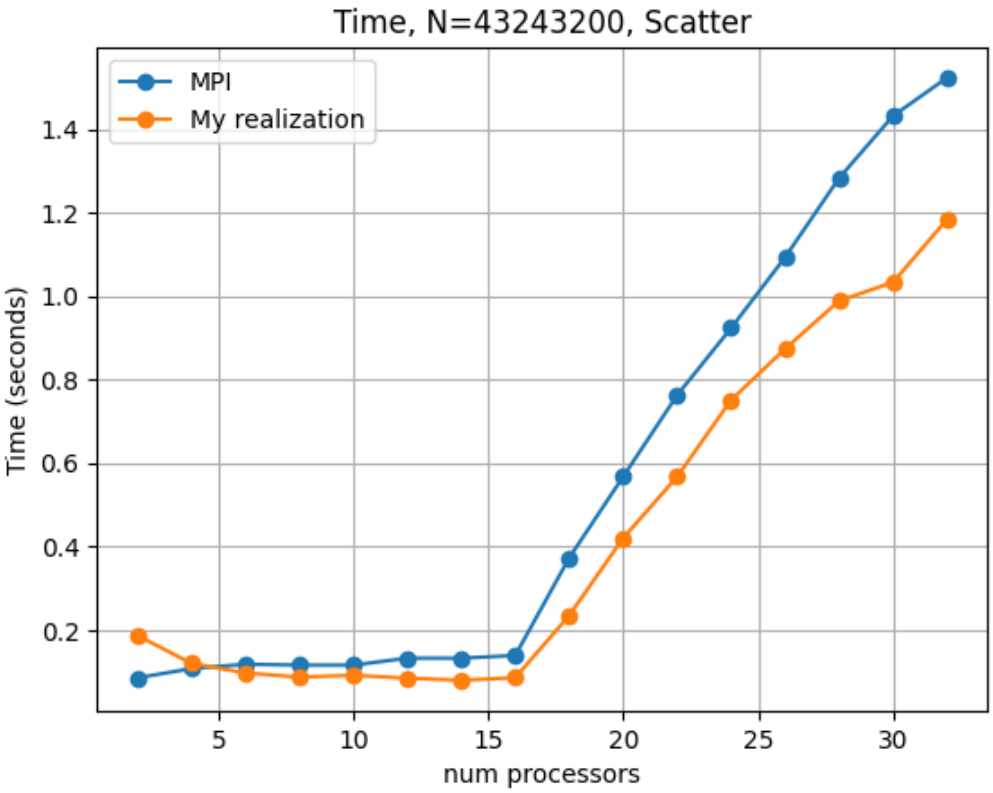

Рисунок 9.3.1. График зависимости времени работы программ от количества процессов

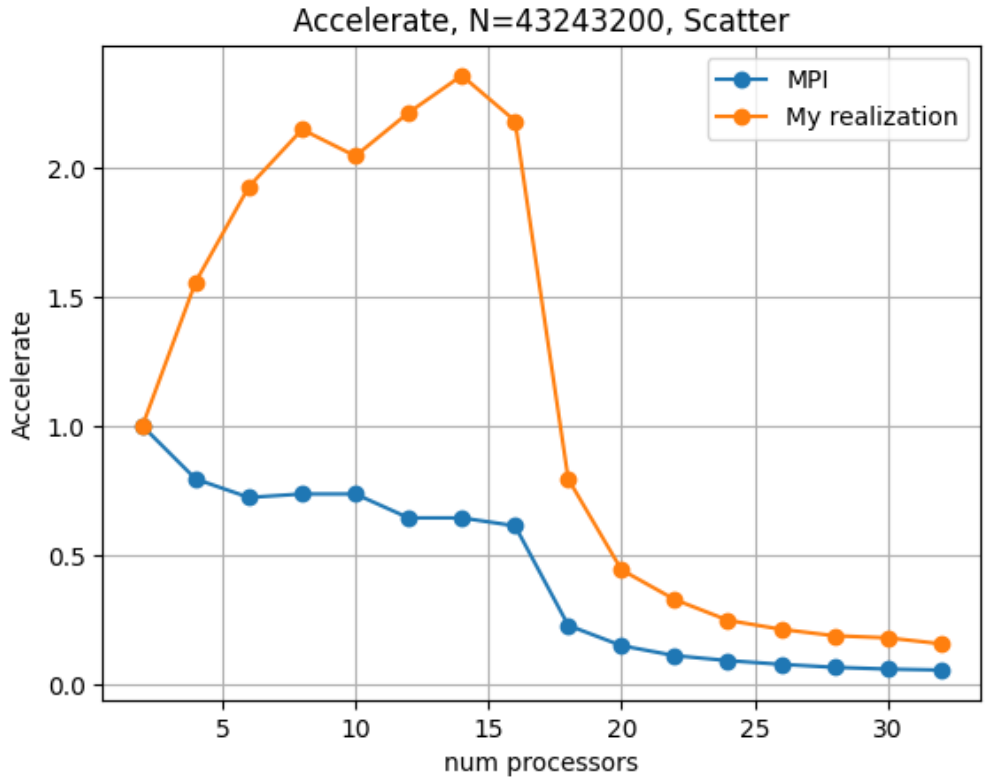

Рисунок 9.3.2. График зависимости ускорения программ от количества процессов

4.Gather

На рисунке 9.4.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.4.2 - график зависимости ускорения программ от количества процессов

Из графика времени работы можно сделать вывод , что при небольшом количестве процессов эффективнее реализация MPI. При количестве процессов от 18 до 26 эффективнее моя реализация. При количестве от 26 до 32 обе реализации примерно одинаковы. При этом стоит отметить, что ускорения моей реализации практически всегда выше.

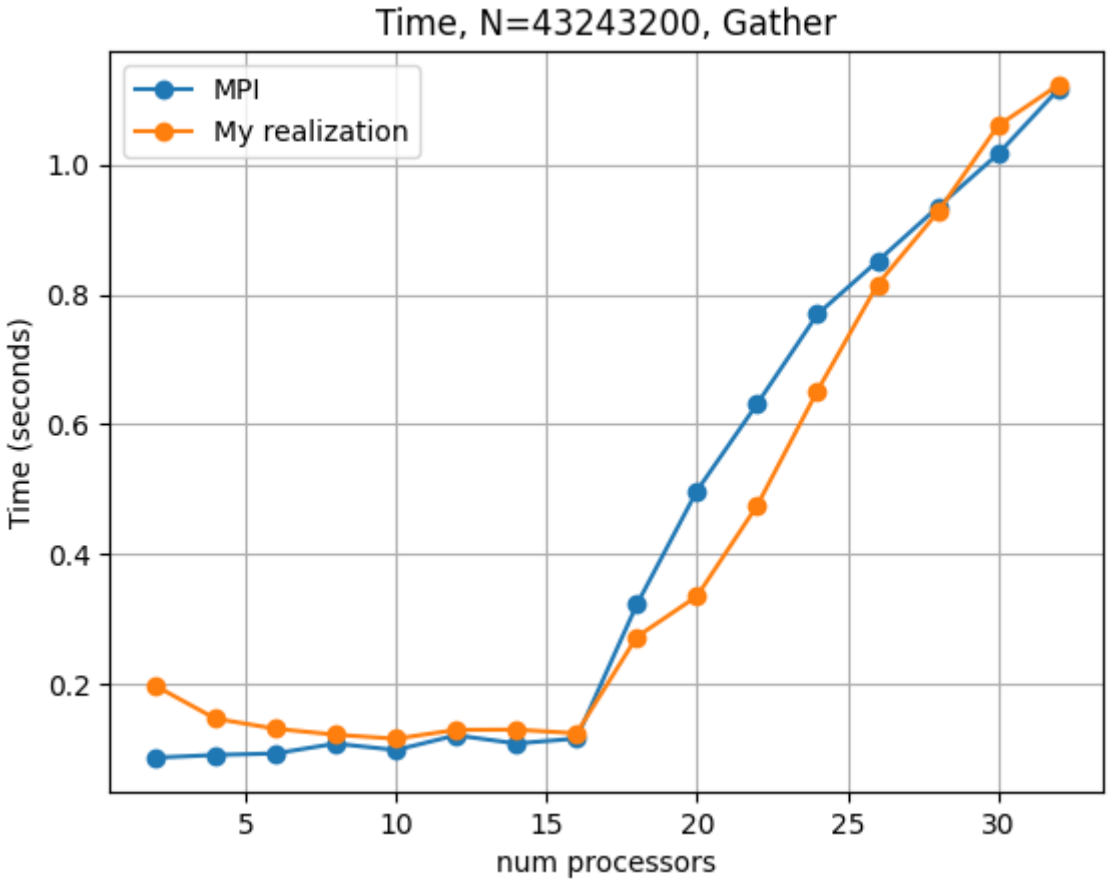

Рисунок 9.4.1. График зависимости времени работы программ от количества процессов

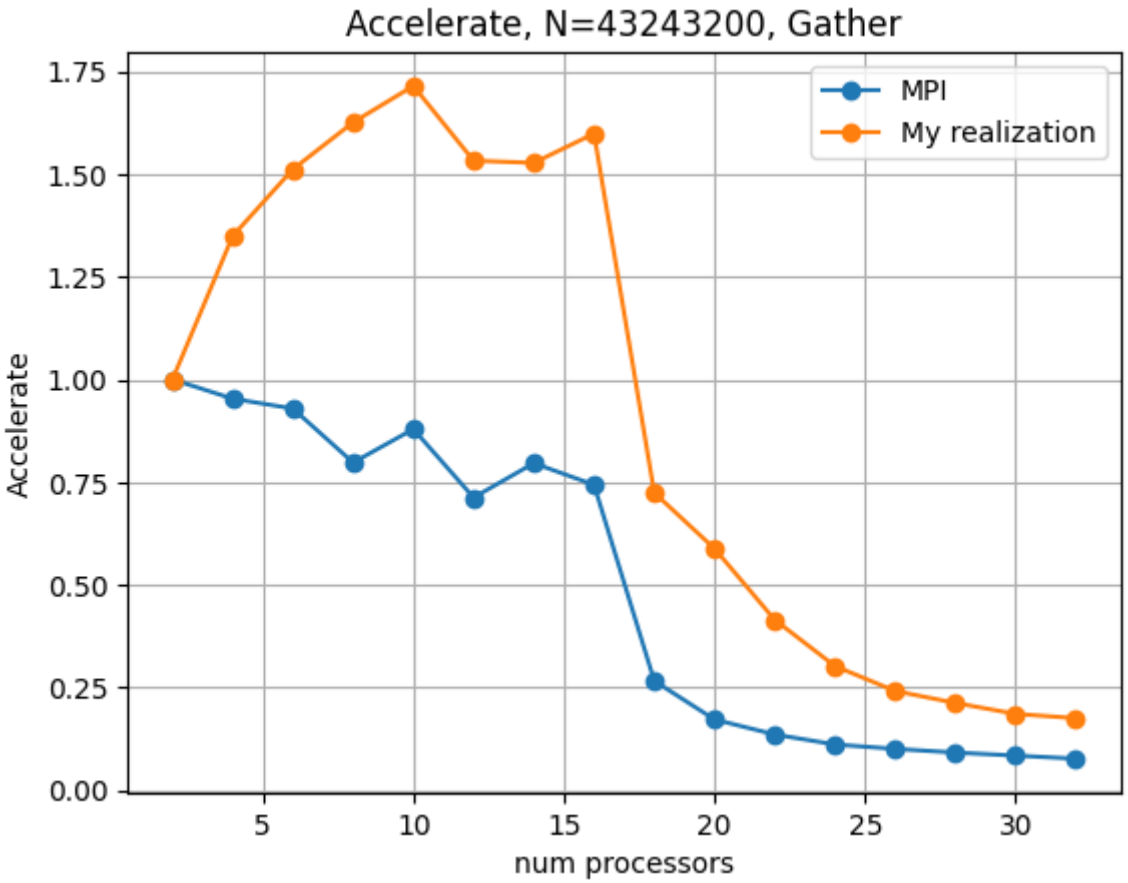

Рисунок 9.4.2. График зависимости ускорения программ от количества процессов

5.AlltoAll

На рисунке 9.5.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.5.2 - график зависимости ускорения программ от количества процессов

Из графика времени работы можно сделать вывод , что при небольшом количестве процессов обе реализации примерно одинаковы по эффективности. Однако при увеличении процессов, реализация MPI оказывается лучше. При этом стоит отметить, что ускорения моей реализации практически всегда выше.

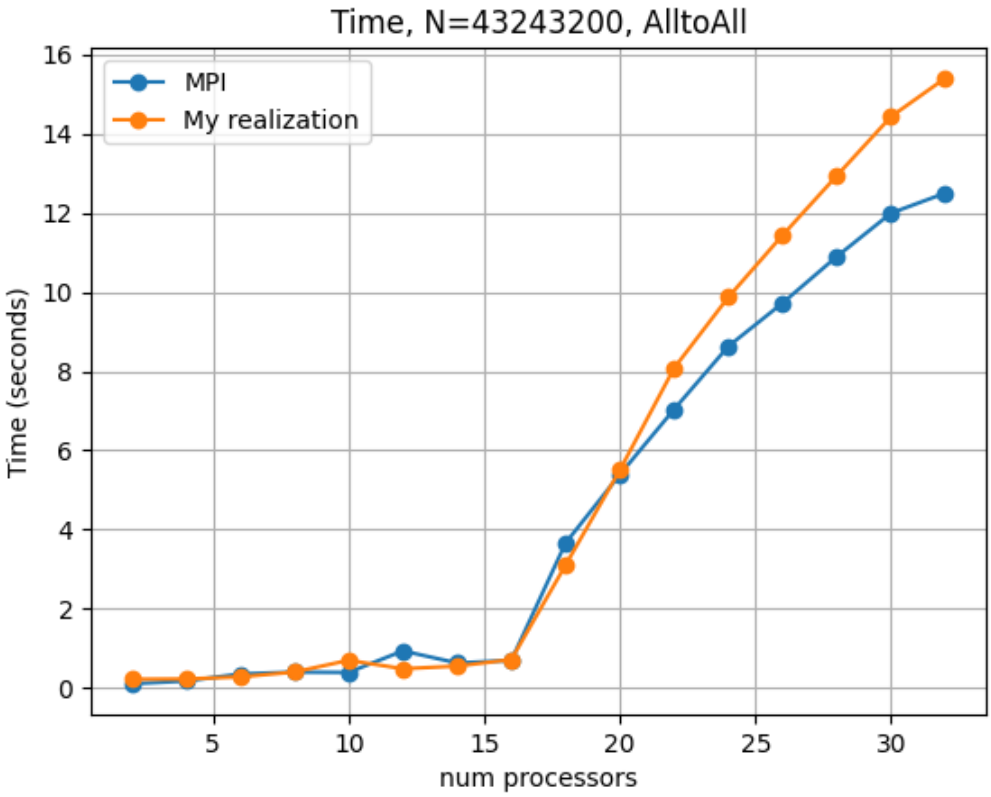

Рисунок 9.5.1. График зависимости времени работы программ от количества процессов

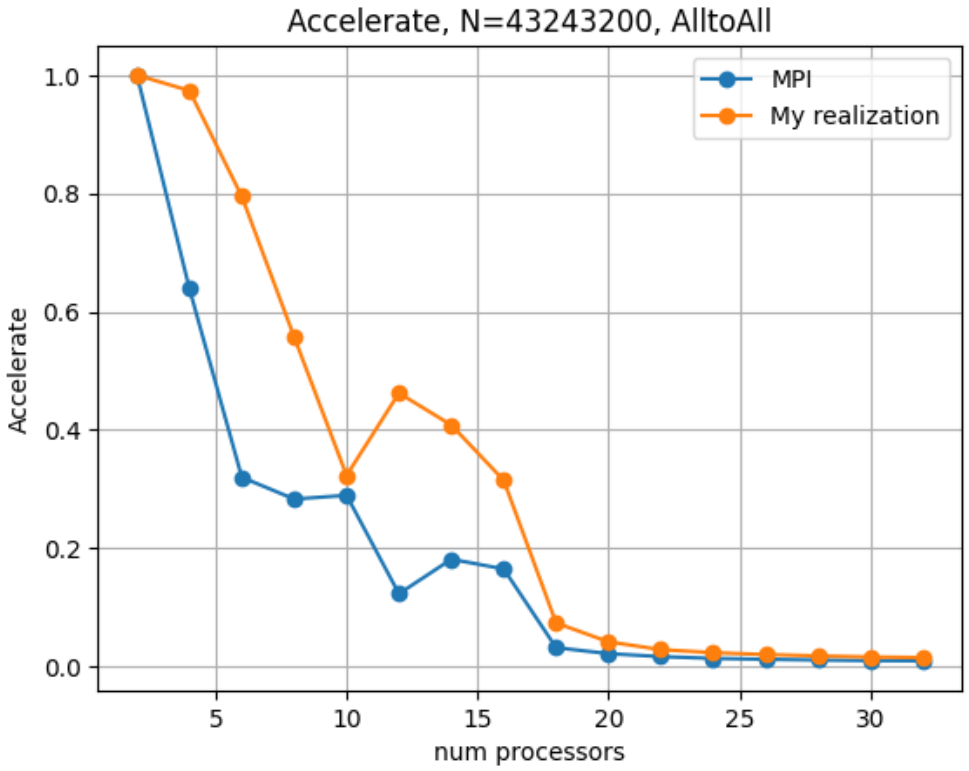

Рисунок 9.5.2. График зависимости ускорения программ от количества процессов

6.AllGather

На рисунке 9.6.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 9.6.2 - график зависимости ускорения программ от количества процессов

Из графика времени работы можно сделать вывод , что при небольшом количестве процессов обе реализации примерно одинаковы по эффективности. Однако при увеличении процессов, реализация MPI оказывается лучше. При этом при небольшом количестве процессов ускорение моей реализации выше, однако при увеличении количества процессов оно стремительно падает.

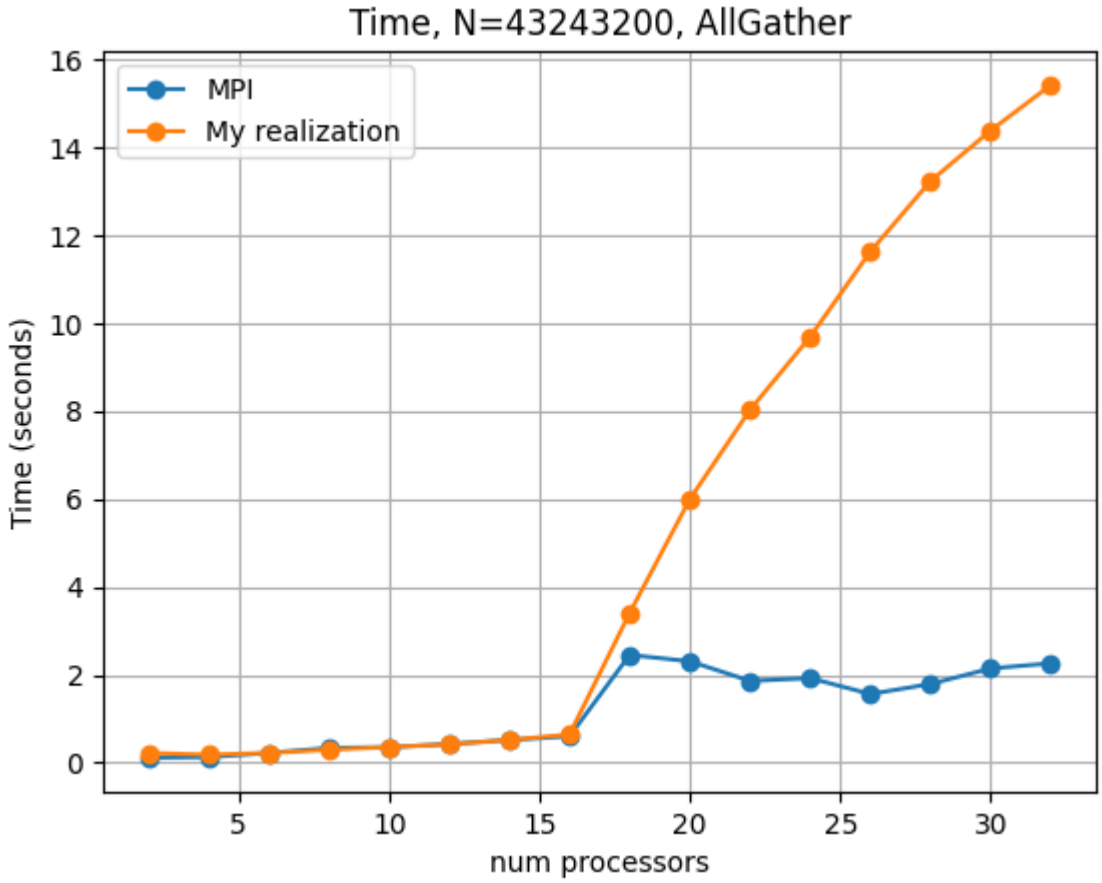

Рисунок 9.6.1. График зависимости времени работы программ от количества процессов

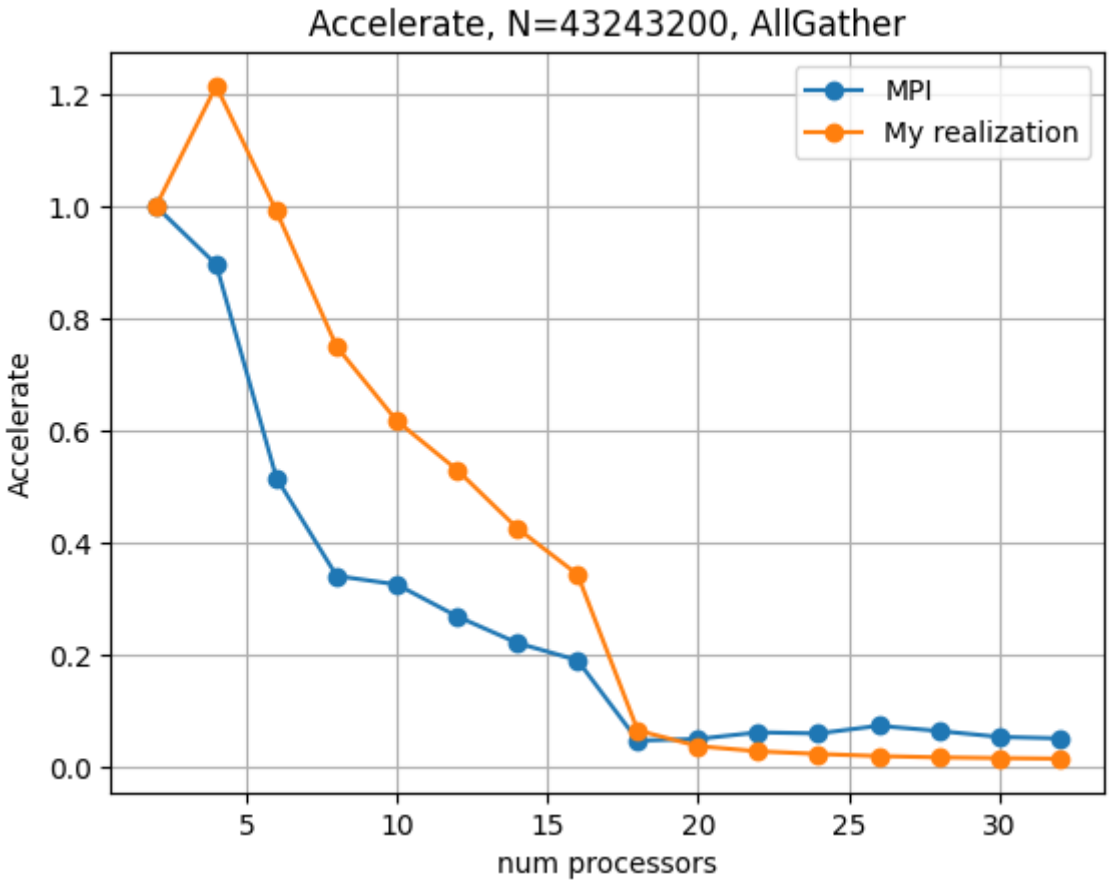

Рисунок 9.6.2. График зависимости ускорения программ от количества процессов

Задание 10

Формулировка

Разработайте программу для конструирования производных типов данных, для использования упаковки и распаковки производных типов данных, проведите эксперименты, сравните результаты этих вариантов при передаче данных

Решение

В качестве структуры для упаковки используется структура с двумя значениями типа int, одним значением типа double и восемью значениями типа char. При этом будет передавать не один элемент, а вектор из таких эелеметов.

При использовании производного типа данных каждый элемент упаковывается отдельно и добавляется в буфер для отправки

При использовании упаковки и распаковки элементы упаковываются в буфер один за другим

При неиспользовании ни того, ни другого, создается 3 буфера для каждого типа данных и элементы один за другим упаковываются в эти буферы. Каждый буфер передается отдельно.

Анализ решения

На рисунке 10.1 представлен график зависимости времени выполнения программ от количества процессов. На рисунке 10.2 - график зависимости ускорения программ от количества процессов.

Из графиков можно сделать вывод, что наиболее эффективным методом передачи для данного случая является создание производного типа данных.

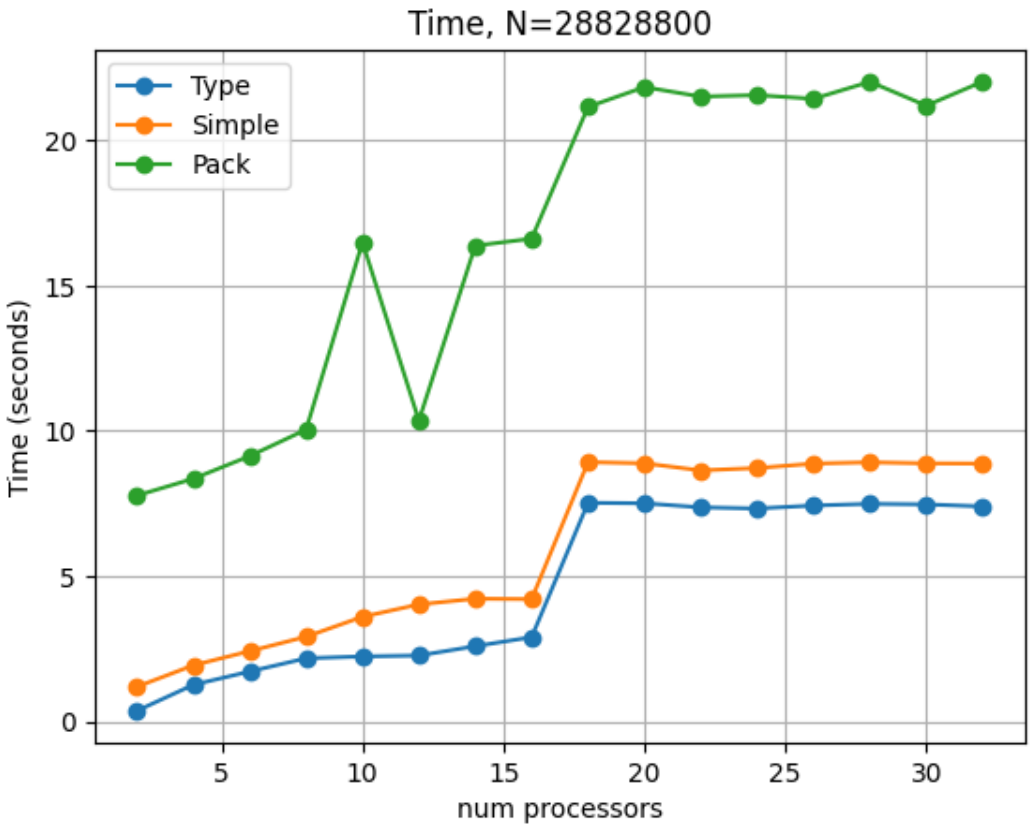

Рисунок 10.1. График зависимости времени работы программ от количества процессов

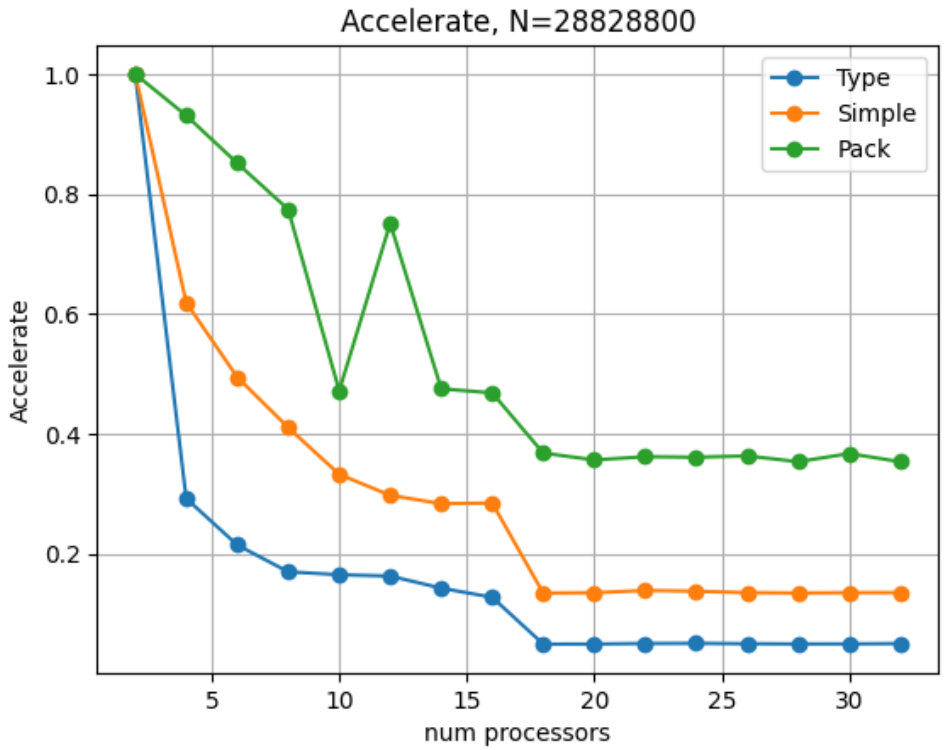

Рисунок 10.2. График зависимости ускорения программ от количества процессов

Задание 11

Формулировка

Разработайте программу для представления множества процессов в виде прямоугольной решетки. Создайте коммуникаторы для каждой строки и каждого столбца процессов. Выполните коллективную операцию для всех процессов и для одного из созданных коммуникаторов. Сравните время выполнения операции

Решение

При помощи MPI_Cart_create создаем представление процессов в виде прямоугольной решетки. При помощи MPI_Cart_split создаем коммуникаторы для столбцов и строк.

Анализ решения

На рисунке 11.1 представлен график зависимости отношения времени передачи всем процессов в коммуникаторе строки к времени передачи все процессам от количества строк и столбцов. Из графика можно сделать вывод, что при небольшом количестве строк их время примерно одинаково или у декартова коммуникатора ниже. При увеличении количества строк время работы декартова коммуникатора заметно увеличивается.

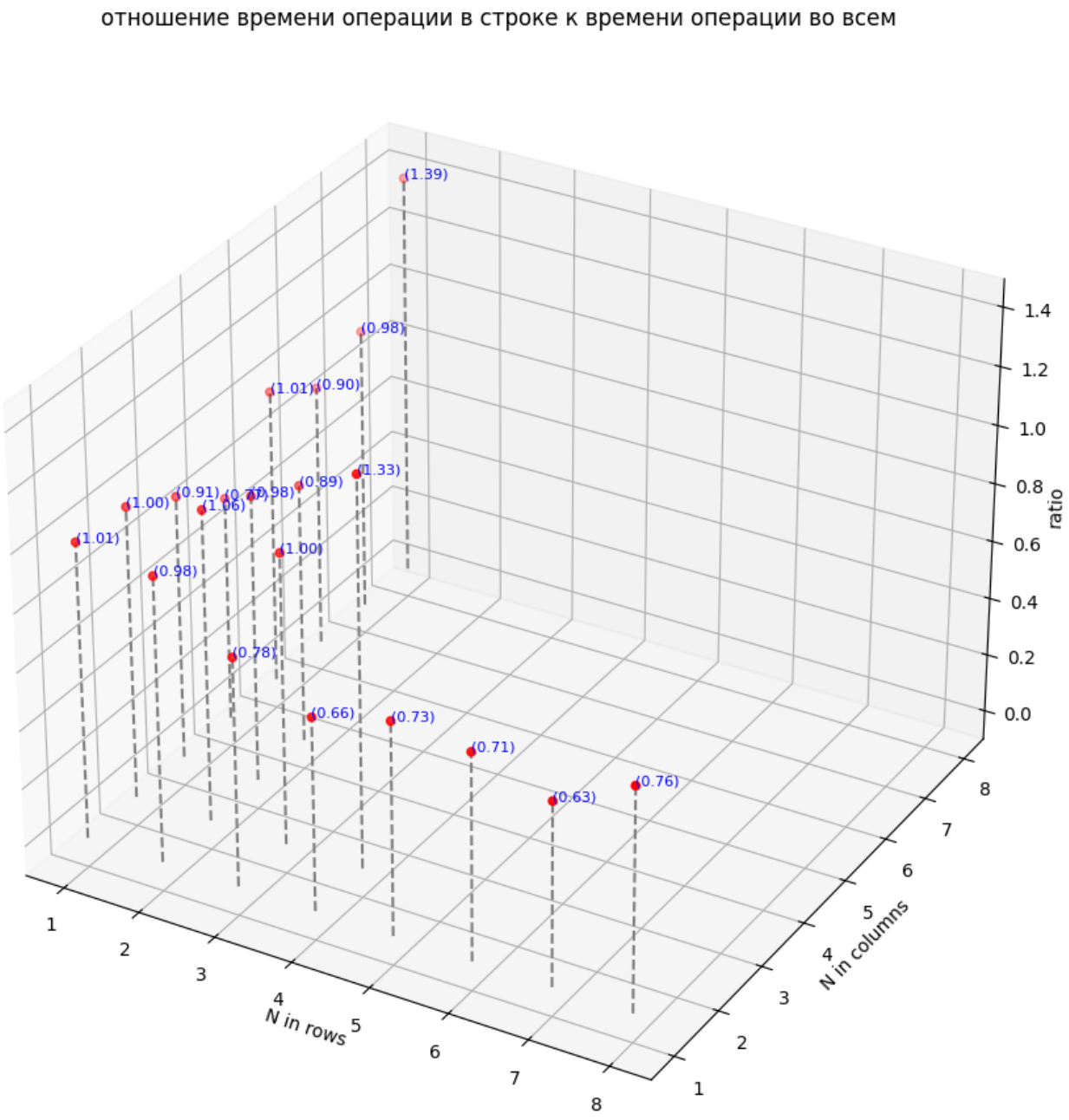

Рисунок 11.1 график зависимости отношения времени передачи всем процессов в коммуникаторе строки к времени передачи все процессам от количества строк и столбцов

Задание 12

Формулировка

Разработайте программу для создания декартовой топологии, топологии тора, топологии графа, топологии звезды

Решение

Для создания декартовой топологии можно воспользоваться функцией MPI_Cart_create.

Для создания топологии n-мерного тора необходимо в N-мерной решетке соединить граничные процессы, что также можно сделать при помощи MPI_Cart_create

Для создания топология звезды необходимо соединить процесс root со всеми остальными процессами.