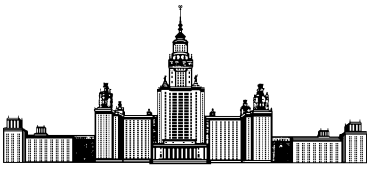


Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра Суперкомпьютеров и Квантовой Информатики

Исследование проблемы масштабируемости в MPI, вызванной пренебрежением отложенными запросами на взаимодействие

Выполнил:

студент 423 группы,

Имашев Владислав Родиславович

Москва, 2023

Содержание

1 Описание сути проблемы

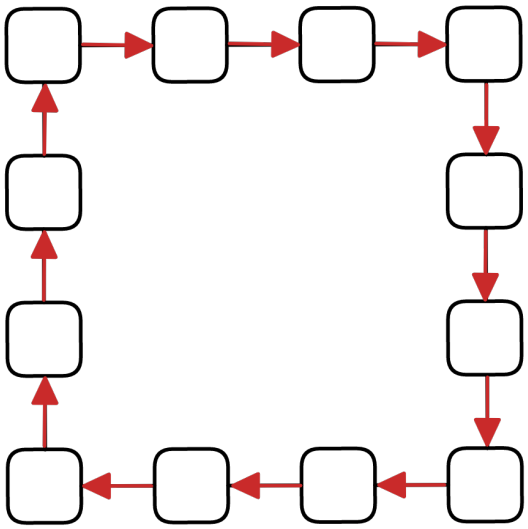
При многократном выполнении однотипных обменов сообщениями с одинаковыми параметрами тратится лишнее время на многократные повторные инициализации обмена сообщениями. В случае, если производится многократные однотипные пересылки, накладные расходы на инициализацию при каждой пересылке могут быть велики. Здесь на помощь приходят так называемые отложенные запросы на взаимодействия. Идея в том, что вместо того, чтобы в цикле вызывать команду, например, Send или Recv, в частности, асинхронные, при которых идет инициализация посылки сообщений, взаимодействие с коммуникационным оборудованием и только потом непосредственно физическая посылка сообщения, лучше сначала описать параметры соответствующего сообщения, фактически инициализировать посылку сообщений, а саму физическую посылку вызывать тогда, когда нужно.

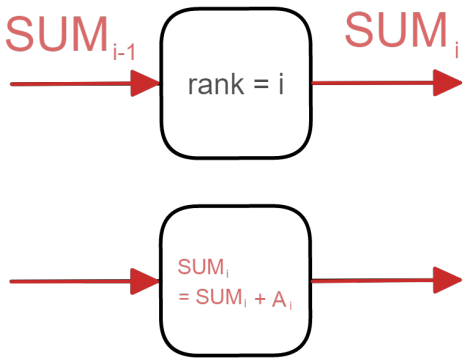
2 Модельная задача

2.1 Постановка задачи

На каждом процессе с рангом $rank$ есть матрица $A_{rank} \in R^{m \times n}$. Необходимо реализовать сложение матриц, находящихся в каждой процессе этой декартовой сетки, полученный результат сложения должен быть на каждом MPI-процессе.

Программная реализация представляет из себя цикл с количеством итераций, равному числу процессов (считаем, что процессы образуют собой топологию односвязного кольцевого списка). На каждой итерации процесс отсылает текущую матрицу-сумму следующему процессу и принимает от предыдущего процесса его результат. Полученную матрицу-резхультат каждый процесс обновляет, прибавляя матрицу, которая изначально хранилась на данном процессе.





2.2 Неоптимизированный вариант программы

В неоптимизированном варианте на каждой итерации цикла происходит инициализация однотипных пересылок:

```
for (int i = 0; i < size; ++i)
{
        if (rank == 0)
        {
                MPI_Send(sum,  matrix_size[0] * matrix_size[1], MPI_DOUBLE,
```

```
                (rank + 1) % size, 0, MPI_COMM_WORLD);
                MPI_Recv(buf, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
                size - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        } else if (rank % 2 == 0)
        {
                MPI_Send(sum,  matrix_size[0] * matrix_size[1], MPI_DOUBLE,
                 (rank + 1) % size, 0, MPI_COMM_WORLD);
                MPI_Recv(buf, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
                rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        else
        {
                MPI_Recv(buf, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
                rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                MPI_Send(sum,  matrix_size[0] * matrix_size[1], MPI_DOUBLE,
                (rank + 1) % size, 0, MPI_COMM_WORLD);
        }
        matrix_sum(buf, A, sum, matrix_size[0], matrix_size[1]);
}
```

2.3 Оптимизированный вариант программы

Оптимизированный вариант программы отличается от исходного тем, что тела однотипных обменов в каждом процессе инициализируются до основного цикла, а в самом цикле непосредственно запускаются обмены:

```
if (rank == 0)
{
        MPI_Recv_init(buf, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
        size - 1, 0, MPI_COMM_WORLD, &request[1]);
}
else
{
        MPI_Recv_init(buf, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
        rank - 1, 0, MPI_COMM_WORLD, &request[1]);
}
MPI_Send_init(sum, matrix_size[0] * matrix_size[1], MPI_DOUBLE,
(rank + 1) % size, 0, MPI_COMM_WORLD, &request[0]);
```

```
for (int i = 0; i < size; ++i)
{
        if (rank % 2 == 0)
        {
                MPI_Start(&request[0]);
                MPI_Start(&request[1]);
        }
        else
        {
                MPI_Start(&request[1]);
                MPI_Start(&request[0]);
        }
        MPI_Waitall(2, request, MPI_STATUS_IGNORE);
        matrix_sum(buf, A, sum, matrix_size[0], matrix_size[1]);
}
```

3 Сравнение двух реализаций

3.1 Метод сравнения

Производилось сравнение двух реализаций с помощью замера времени работы параллельной секции. Для этого использовалась функция $omp\_get\_wtime()$.

Все запуски производились на вычислительной системе IBM Polus, на 4 узле. Характеристики узла:

- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков) всего
- 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек

Компиляция программ производилась с помощью компилятора mpicc:

```
mpicc program.c -o program
```

Постановка задач на выполнения производилась с помощью планировщика LSF (через bsub, используя командный файл).

3.2 Результаты сравнения

Ниже представлены результаты работы программ для разных размеров входных данных (для матриц размером 1000x1000 и 10000x10000):

Число процессов	2	4	8	10	16	20	32
Время неоптимиз. (сек)	0.02	0.04	0.11	0.15	0.24	0.37	0.41
Время оптимиз. (сек)	0.02	0.04	0.11	0.13	0.22	0.35	0.39
Разница (%)	0	0	0	13.33	8.33	5.4	4.87

Таблица 1: Матрица 1000x1000

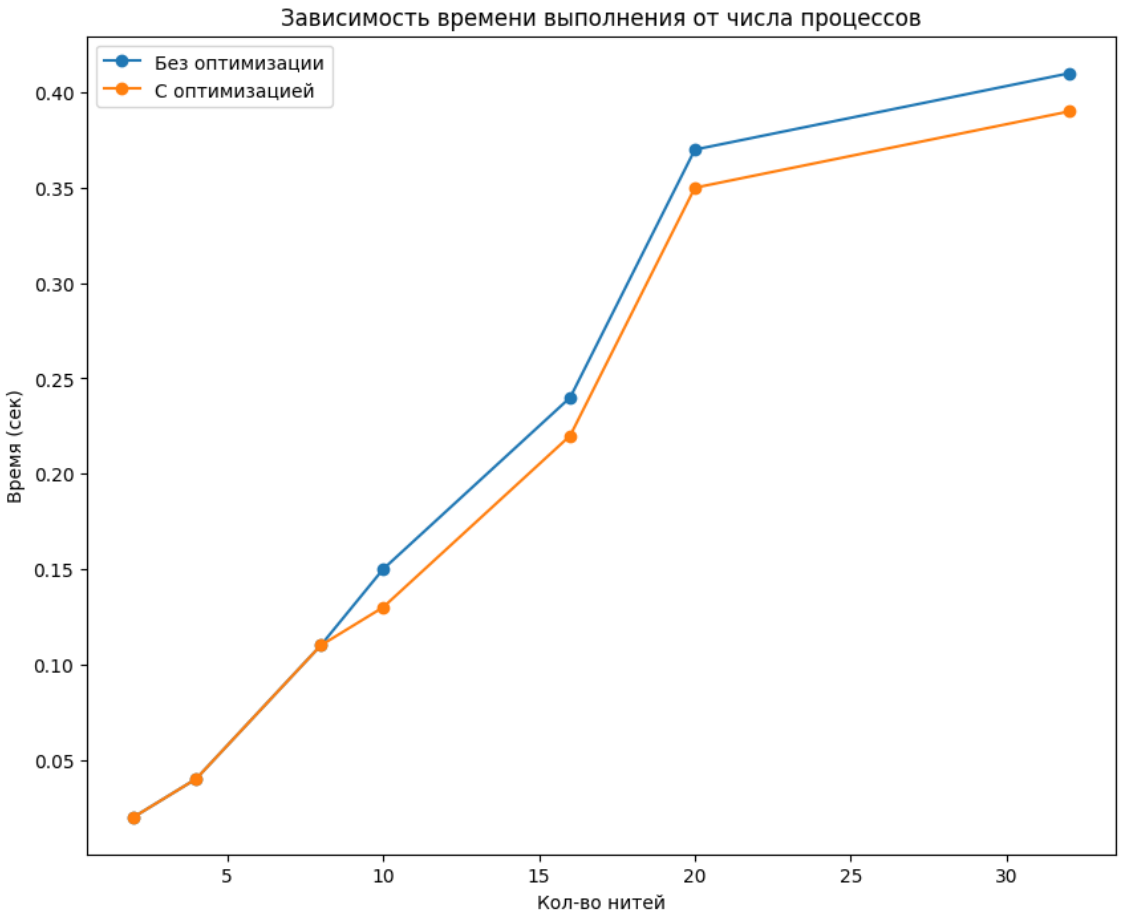


Рис. 1: Матрица 1000x1000

Число процессов	2	4	8	10	16	20	32
Время неоптимиз. (сек)	1.96	4.09	10.09	12.25	21.42	28.15	39.47
Время оптимиз. (сек)	1.79	3.72	8.26	11.05	18.03	23.00	32.32
Разница (%)	8.67	9.04	18.14	8.33	15.83	18.29	18.11

Таблица 2: Матрица 10000x10000

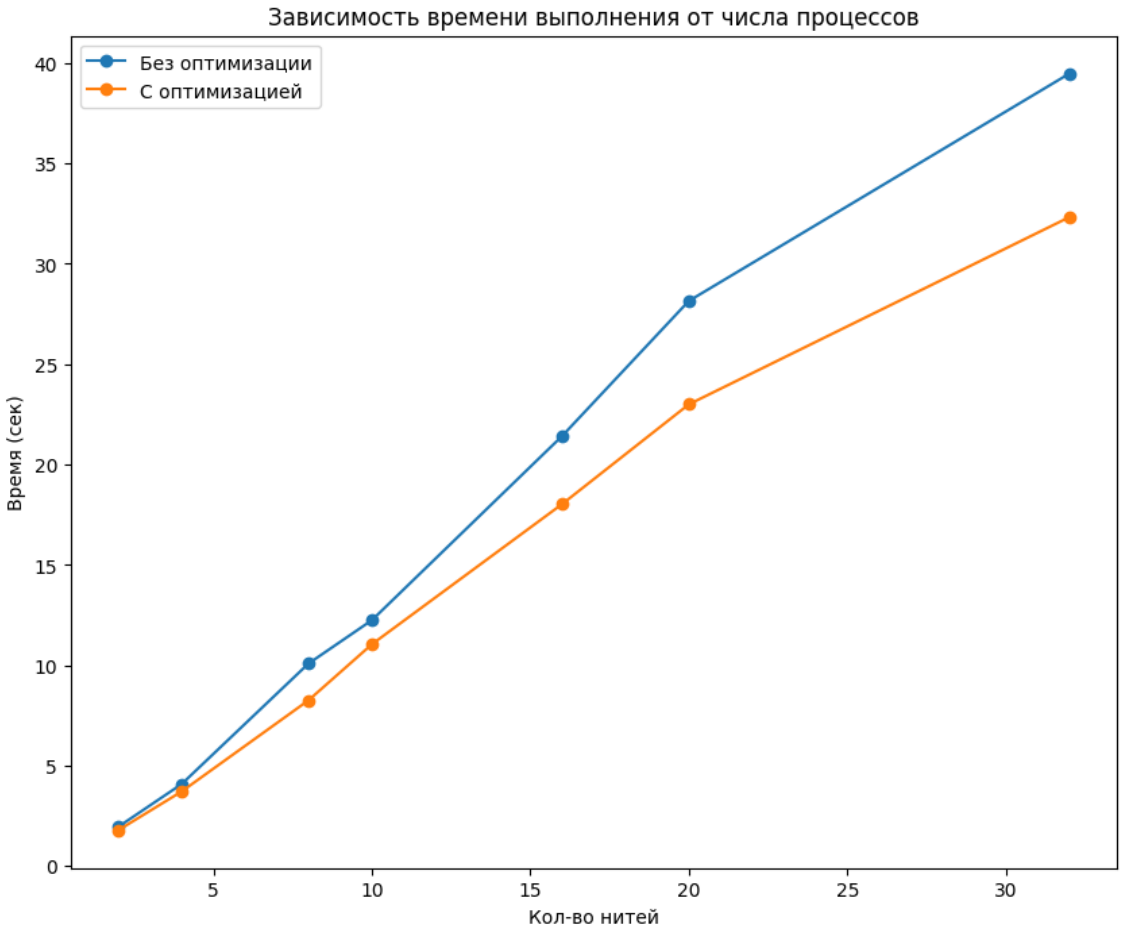


Рис. 2: Матрица весов 10000x10000