МГУ им. М. В. Ломоносова, факультет ВМК

Задание 2 Проблемы масштабируемости в MPI. Пренебрежение отложенными запросами на взаимодействие .

Арбузов Николай Романович
группа 423

Проблема

При реализации параллелизма в решении задач с помощью MPI пренебрежение отложенными запросами на взаимодействие повышает время выполнения задачи без влияния на качество выполнения программы.

Алгоритм

В качестве показательного алгоритма для этой проблемы был выбран алгоритм обмена массивами между процессами:
Проблемный код:

```
for (int i = 0; i < msg_size; i++)
    {

        MPI_Isend(send_buf, msg_size, MPI_INT, (rank + size - 1 - (i % (size - 1))) %
size, MSG_TAG, MPI_COMM_WORLD, &req_send);
        MPI_Irecv(&recv_buf[msg_size * i], msg_size, MPI_INT, (rank + 1 + (i % (size -
1))) % size, MSG_TAG, MPI_COMM_WORLD, &req_recv);

        MPI_Wait(&req_send, MPI_STATUS_IGNORE);
        MPI_Wait(&req_recv, MPI_STATUS_IGNORE);
        for (int j = 0; j < msg_size; j++)
        {
            if (recv_buf[msg_size * i + j] != (rank + 1 + (i % (size - 1))) % size)
            {
                std::cerr << "Error: received value " << recv_buf[msg_size * (i < rank
? i : i - 1) + j] << " from process " << i << std::endl;
                break;
            }
        }
    }
```

Код без проблемы:

```
for (int i = 0; i < msg_size; i++)
    {

        MPI_Send_init(send_buf, msg_size, MPI_INT, (rank + size - 1 - (i % (size -
1))) % size, MSG_TAG, MPI_COMM_WORLD, &send_reqs[i]);
        MPI_Recv_init(&recv_buf[msg_size * i], msg_size, MPI_INT, (rank + 1 + (i %
(size - 1))) % size, MSG_TAG, MPI_COMM_WORLD, &recv_reqs[i]);
    }

    MPI_Startall(msg_size, send_reqs);
    MPI_Startall(msg_size, recv_reqs);

    MPI_Waitall(msg_size, send_reqs, statuses);
    MPI_Waitall(msg_size, recv_reqs, statuses);

    for (int i = 0; i < msg_size; i++)
    {
        for (int j = 0; j < msg_size; j++)
        {
```

```
            if (recv_buf[msg_size * i + j] != (rank + 1 + (i % (size - 1))) % size)
            {
                std::cerr << "Error: received value " << recv_buf[msg_size * i + j] <<
" from process " << (rank + i + 1 + (i + 1) / size) % size << std::endl;
                break;
            }
        }
    }
```

Компиляция и запуск

Все вычисления производились на машине Polus.
Сама программа написана на языке C++ и состоит из файлов:

- main.cpp / main_problem.cpp (в зависимости от того, как мы хотим запустить с без отложенных запросов на взаимодействие или с ними)

Компилировалась с использованием Makefile:

```
all: main

problem: *.cpp *.h
    mpicxx problem_main.cpp -o prog -std=c++11

main: *.cpp *.h
    mpicxx main.cpp -o prog -std=c++11

clean:
    rm -rf ./prog
```

Запуск производился постановкой в очередь с помощью планировщика mpisubmit.pl:

```
mpisubmit.pl -p $i -w 00:30 --stdout ./out_files/$j/$i.out --stderr
./err_files/$j/$i.err ./prog -- polus $j
```

Где **i** – количество процессов, на которых будет запускаться программа, а **j** – размер матрицы, на которой будут производиться вычисления.
Оценка результативности изменений проводилась по средством измерения времени с помощью функции gettimeofday(&Tp, NULL).

Результаты

	Size	Nthreads	Time Normal	Time Problem	diff
0	10,000	1	0.32	0.26	-0.05
1	10,000	2	0.74	0.32	-0.42
2	10,000	4	0.68	0.57	-0.11
3	10,000	8	0.82	0.74	-0.08

4	10,000	16	0.96	1.56	0.61
5	25,000	1	1.43	1.14	-0.29
6	25,000	2	2.99	1.44	-1.56
7	25,000	4	2.14	3.02	0.88
8	25,000	8	2.86	3.83	0.97
9	25,000	16	3.48	6.43	2.95

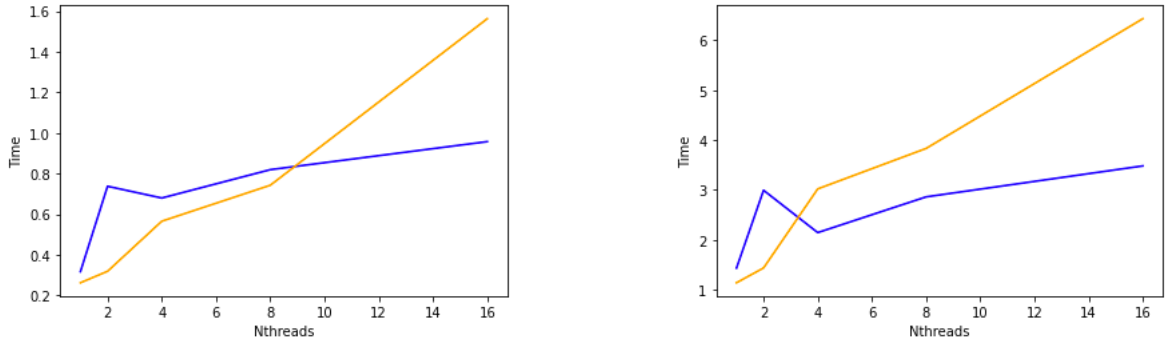

Рисунок 1. Графики времени выполнения программы: Синим цветом без использования операторов инициации пересылки, оранжевым цветом – с ними. Графика по порядку для матриц размером 10000 и 25000.

По графикам видно, что с увеличением числа процессов программа без использования отложенных запросов работает менее эффективно, это связано с тем, что при большем числе процессов инициируется больше запросов на взаимодействие