|  |
| --- |
| 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

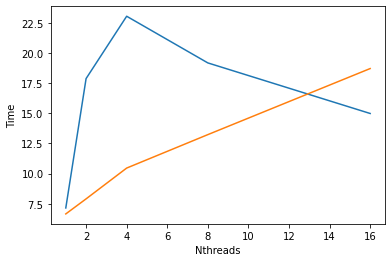
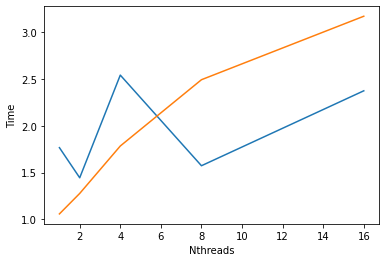
Где **i** – количество процессов, на которых будет запускаться программа, а **j** – размер матрицы, на которой будут производиться вычисления.

Оценка результативности изменений проводилась по средством измерения времени с помощью функции gettimeofday(&Tp, NULL).

# Результаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Size** | **Nthreads** | **Time\_Normal** | **Time\_Problem** | **diff** |
| **0** | 10000 | 1 | 1.7664849999999999 | 1.058165 | -0.7083199999999998 |
| **1** | 10000 | 2 | 1.444425 | 1.2770325 | -0.16739250000000006 |
| **2** | 10000 | 4 | 2.5417525 | 1.7839775 | -0.7577749999999999 |
| **3** | 10000 | 8 | 1.57318 | 2.4924850000000003 | 0.9193050000000003 |
| **4** | 10000 | 16 | 2.3740075000000003 | 3.1709050000000003 | 0.7968975 |
| **5** | 25000 | 1 | 7.1447775 | 6.6493825 | -0.49539500000000025 |
| **6** | 25000 | 2 | 17.8770275 | 7.8975275 | -9.979500000000002 |
| **7** | 25000 | 4 | 23.048907500000002 | 10.4532325 | -12.595675000000002 |
| **8** | 25000 | 8 | 19.1776575 | 13.2277 | -5.949957499999998 |
| **9** | 25000 | 16 | 14.98165 | 18.709 | 3.7273499999999995 |

Рисунок . 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