FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

MPI. Assignments 12 – 13

Parallel algorithms for the analysis and synthesis of data

Performed by
Aleksandr Shirokov
J4133c
Accepted by
Petr Andriushchenko
Deadline: 22.12.21

St. Petersburg
2021

Contents

1 Assignments

1.1 Assignment 12. MPI. Delayed interactions. Scheme of an iterative method with exchange over a ring topology using pending requests.

1.1.1 Formulation of the problem

MPI allows for non-blocking operations to form whole packets of requests for communication operations MPI_Send_init and MPI_Recv_init , which are started by the MPI_Start or MPI_Startall functions.

Checking for completion of execution is performed by conventional means using the functions of the **WAIT** and **TEST** families.

Find and fix errors in Assignment12.c, add the for loop. When should you use a loop?

1.1.2 Example of launch parameters and output. Detailed description of solution

Code for **assignment 12** is here.

Compilation example: mpic++ -o ./cpf/12.o Assignment12.c

Launch example: mpirun –oversubscribe -np 10 ./cpf/12.o

Let's move to the the code and explain how it works.

```cpp
#include <iostream>
#include "mpi.h"

using namespace std;
int main(int argc, char **argv)
{
    int rank, size, prev, next;
    float sbuf[2], rbuf[2];

    MPI_Init(&argc, &argv);

    MPI_Request reqs[4];
    MPI_Status stats[4];

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    prev = rank - 1;
    next = rank + 1;

    if (rank == 0) prev = size - 1;
    if (rank == size - 1) next = 0;

    MPI_Recv_init(&rbuf[0], 1, MPI_FLOAT, prev, 5, MPI_COMM_WORLD, &reqs[0]);
    MPI_Recv_init(&rbuf[1], 1, MPI_FLOAT, next, 6, MPI_COMM_WORLD, &reqs[1]);
    MPI_Send_init(&sbuf[0], 1, MPI_FLOAT, prev, 6, MPI_COMM_WORLD, &reqs[2]);
    MPI_Send_init(&sbuf[1], 1, MPI_FLOAT, next, 5, MPI_COMM_WORLD, &reqs[3]);

    for (int i = 0; i < 2; i++) sbuf[i] = float(rank);

    MPI_Startall(4, reqs);
    MPI_Waitall(4, reqs, stats);

    cout << rbuf[0] << " (previous)" << " -> " << rank << " (current)" << " -> " << rbuf[1] << " (next)" << '\n' << endl;

    for (int i = 0; i < 4; i++) MPI_Request_free(&reqs[i]);

    MPI_Finalize();
}
```

Assignment12 code

Firstly i wrote a whole program that can be compiled and works correctly as a ring topology as we can see on picture. As fixing errors in lines $24 - 27$ i have add by symbol & address of variable RBUF and SBUF (also i have created float arrays with the same names), also do the same for REQS. In lines $26 - 27$ we can see that there is no values in array SBUF, so for loop is for mapping float value of rank to each piece of array (line 29). Inside for loop there is no need to start sending and recieving messages on each iteration - we should do it only ones as i did it in lines $31 - 32$. After that i displayed information in ring topology and used for loop for function MPI_REQUEST_FREE to free inactive persistent requests created with either MPI_RECV_INIT or MPI_SEND_INIT and friends. All that i said is the answer on question when should we use loop: free inactive requests and may be for initialzing data to send. That's all, program works correctly and errors are fixed.

1.2 Assignment 13. MPI. Collective process interactions. Barrier.

1.2.1 Formulation of the problem

Find out which process will perform the multiplication of two 500×500 square matrices faster.

Complete the code Assignment13.c. You can use the necessary code from the previous assignments.

1.2.2 Example of launch parameters and output. Detailed description of solution

Code for **assignment 13** is here.

Compilation example: mpic++ -o ./cpf/13.o Assignment13.c

Launch example: mpirun –oversubscribe -np 5 ./cpf/13.o 500

```
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 5 ./cpf/13.o 500
process 2, execution time=2.03871
process 4, execution time=2.37145
process 3, execution time=2.41065
process 0, execution time=2.42486
process 1, execution time=2.48914
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

Let's move to the the code and explain how it works.

```cpp
#include <mpi.h>
#include <stdio.h>
#include <ctime>
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char **argv)
{
    int matrix_size = atoi(argv[1]);
    int rank, size;
    MPI_Init(&argc, &argv);

    double start_time, end_time;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //matrix initialization by each process

    int A[matrix_size][matrix_size], B[matrix_size][matrix_size], C[matrix_size][matrix_size];

    for (int i = 0; i < matrix_size; i++)
    {
        for (int j = 0; j < matrix_size; j++)
        {
            A[i][j] = 1 + rand() % 10;
            B[i][j] = 1 + rand() % 10;
            C[i][j] = 0;
        }
     }


    MPI_Barrier(MPI_COMM_WORLD); //barrier process synchronization

    // start timing for each process

    start_time = MPI_Wtime();

    // matrix multiplication

    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            for (int k = 0; k < matrix_size; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    // end of timing

    end_time = MPI_Wtime();

    // output the execution time of matrix multiplication at each process

    cout << "process " << rank ;
    cout << ", execution time=" << (end_time - start_time);
    cout << '\n' << endl;
    MPI_Finalize();
}
```

Assignment13 code

What should we need was written in file by comments, by job was to fill the gaps as code. Firstly i initialize matrices A and B with the same seed for each process with random integers from 1 to 10 and matrix C by null values. Next there are a barrier function which is waiting when all processes initialize their matrices, after that we start timing for eahc process, do known matrix multiplication algorithm, end timing and show execution time of matrix multiplication at each process and understand which process performed faster. In our example it is process with rank 2. Program works correcly and algorithm is explained.

1.3 Appendix

The link to the sourse code which is placed on my github.