FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

MPI. Assignments $3 - 5$

Parallel algorithms for the analysis and synthesis of data

Performed by
Aleksandr Shirokov
J4133c
Accepted by
Petr Andriushchenko
Deadline: 19.12.21

St. Petersburg
2021

Contents

1 Assignments

1.1 Assignment 3.

1.1.1 Formulation of the problem

Compile and run ASSIGNMENT3.C program. Explain in detail how it works.

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

Code for **assignment 3** is here.
Compilation example: MPIC++ -O ./CPF/3.O ASSIGNMENT3.C
Launch example:

- Not in parallel: ./CPF/3.O

```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpic++ -o ./cpf/3.o Assignment3.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ ./cpf/3.o
Hello from process 0
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

- In parallel: MPIRUN –OVERSUBSCRIBE -NP 4 ./CPF/3.O (have a problem without –OVERSUBSCRIBE option.

```
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 4 ./cpf/3.o
Hello from process 0
Hello from process 1
Hello from process 2
Hello from process 3
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

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[]) {
    MPI_Init(&argc, &argv);
    int rank, n, i, message;
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        cout << "Hello from process " << rank << "\n";
        for (i  = 1; i < n; i++) {
            MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            cout << "Hello from process " << message << endl;
        }
    }
    else MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}

```

Assignment3 code

1. Line 5 - MPI_Init - initialisation, starting the parallel part with arguments of main function;
2. Line 6 - initialize variables, especially *rank* for rank of process and n for amount of process
3. Line 7 - creating MPI_status, variable status contains three attibutes of message:
   - MPI_source - number of the sending process;
   - MPI_tag - name of text message, identifier;
   - MPI_error - error code.
4. Line 7 - getting number of processes (n varialbe)
5. Line 8 - getting rank of the process (*rank* variable)
6. Line 10 - if statement
   - if rank == 0 - then we are printing 'Hello from process 0' and in range $i = [1..n]$ where i is number of process, waiting for incoming messages from any source (who is quicklier will be write earlier) with any tag using syntax of function MPI_Send. The information 'Hello from process %message' is printing after we get message on each iteration of cycle.
   - else - from processes with rank not 0 sending messages to process with rank 0. The message contains information about processes's rank.
7. Line 19 - Ending parallel part

I have analysed the first code with mpi, explain how it works, compile it and show results.

1.2 Assignment 4.

1.2.1 Formulation of the problem

1. Convert the code Assignment4.c to match your individual version of the assignment.
2. My task according to the table is 27.
3. If the number of processes in the task is not defined, then we can assume that this number does not exceed 8.
4. The *main process* is understood as a process of rank 0 for the communicator MPI_COMM_WORLD.
5. For all processes of nonzero rank in the tasks, the common name of the subordinate processes is used.
6. If the task does not define the maximum size of a set of numbers, then it should be considered equal to 10.

Variant 27:

1. Each subprocess is given four integers.
2. Send these numbers to the main process using one call to the MPI_Send function for each sending process, and display them in the main process.
3. The resulting numbers should be displayed in ascending order of the ranks of the processes that sent them.

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

Code for **assignment 4** is here.

Compilation example: mpic++ -o ./cpf/4.o Assignment4.c

Launch example:mpirun –oversubscribe -np 8 ./cpf/4.o (have a problem without –oversubscribe option.

```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 8 ./cpf/4.o
Start main process 0
Print numbers from process 1: 18 2 11 20
Print numbers from process 2: 4 18 3 4
Print numbers from process 3: 16 13 10 4
Print numbers from process 4: 8 11 14 4
Print numbers from process 5: 5 7 6 9
Print numbers from process 6: 14 7 1 5
Print numbers from process 7: 19 13 18 20
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

Works correctly - four integers in each subprocess was send to main process and displayed in ascending order of the ranks

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

```cpp
#include <stdio.h>
#include "mpi.h"
#include <ctime>
using namespace std;

int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv);
    int rank, n, i;
    int message[5];
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        cout << "Start main process " << rank << endl;
        for (i = 1; i < n; i++)
        {

            MPI_Recv(&message[0], 5, MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            cout << "Print numbers from process " << message[4] << ": ";
            for (int j = 0; j < 4; j++)
            {
                cout << message[j] << ' ';
            }
            cout << endl;
        }
    }
    else
    {
        srand(time(NULL) + rank);
        int sending[5];
        for (int i = 0; i < 4; i++)
        {
            sending[i] = 1 + rand() % 20;
        }
        sending[4] = rank;

        MPI_Send(&sending[0], 5, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

Assignment4 code

1. Line 7 - MPI_Init - initialisation, starting the parallel part with arguments of main function;
2. Line 8 - initialize variables, especially *rank* for rank of process and n for amount of process
3. Line 9 - initializing int array of size 4 for input messages from subprocesses;
4. Line 10 - creating MPI_status, variable status contains three attibutes of message:
    - MPI_source - number of the sending process;
    - MPI_tag - name of text message, identifier;
    - MPI_error - error code.
5. Line 11 - getting number of processes (n varialbe)
6. Line 12 - getting rank of the process (*rank* variable)
7. Line 13 - if statement

- IF RANK == 0 - then we are printing 'Start main process 0' and in range $i = [1..n]$ where i is number of process, waiting for incoming messages from each i source (from 1 to n) with any tag using syntax of function MPI_SEND. When message is recieved, the array of 5 integers is displayed in process 0 from process i - first 4 is integers that we need to display and the last - rank of process (something like check that program works correctly). Using this algrotihm we print results of process in ascending order of the ranks.
- else - from processes with rank not 0 sending messages to process with rank 0. Generate int array of size 5 with different number for each process using line 29 - first 4 elements are random integers from 1to 20 and the last is rank of process. The message contains information about generated array using interface MPI_SEND - start with first element of array SENDING, 4 elements of size int to process 0.

8. Line 19 - Ending parallel part

I have implemented an algorithm from task and display results of processes in ascending order.

1.3 Assignment 5.

1.3.1 Formulation of the problem

1. Compile and run Assignment5.c program.
2. Explain in detail how it works.
3. Determine the execution time of the program from the previous task.

   Task from previous lab: variant 27.

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

1.3.2.1 Part I. Explain Assignment 5.c $

Code for **assignment 5** is here.

Compilation example: mpic++ -o ./cpf/5.o Assignment5.c

Launch example:mpirun –oversubscribe -np 8 ./cpf/5.o (have a problem without –oversubscribe option.

```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 8 ./cpf/5.o
processor improfeo, process 6time = 4.4e-08
processor improfeo, process 4time = 4.5e-08
processor improfeo, process 5time = 4.5e-08
processor improfeo, process 2time = 4.4e-08
processor improfeo, process 0time = 8.1e-08
processor improfeo, process 3time = 6.66e-07
processor improfeo, process 1time = 4.3e-08
processor improfeo, process 7time = 4.4e-08
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

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

```cpp
#include <iostream>
#include "mpi.h"
#define NTIMES 100
using namespace std;
int main(int argc, char **argv)
{
    double time_start, time_finish;
    int rank, i;
    int len;
    char *name = new char;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &len);
    time_start = MPI_Wtime();
    for (i = 0; i < NTIMES; i++)
        time_finish = MPI_Wtime();
    cout << "processor " << name << ", process " << rank << "time = " << (time_finish - time_start) / NTIMES << endl;
    MPI_Finalize();
}
```

Assignment5 code

The main purpose of the problem is to count average time consumption of every process. This program takes as an input parameter an amount of processes, for NTIMES run the program and count the average time of each process works, also the processor and the rank of process is displayed.

1.3.2.2 Part II. Determine the execution time of the program from the previous task

Code for **assignment 5 partII** is here.

Compilation example: MPIC++ -O ./CPF/5.1.O 5.1.C

Launch example:MPIRUN –OVERSUBSCRIBE -NP 8 ./CPF/5.1.O (have a problem without –OVERSUBSCRIBE option.

```
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 8 ./cpf/5.1.o
processor improfeo, process 3 time = 2.62e-05
processor improfeo, process 1 time = 1.91e-05
processor improfeo, process 2 time = 5.41e-05
processor improfeo, process 4 time = 2.4e-05
processor improfeo, process 5 time = 2.36e-05
processor improfeo, process 6 time = 2.05e-05
processor improfeo, process 7 time = 1.79e-05
Start main process 0
Print numbers from process 1: 7 20 1 18
Print numbers from process 2: 11 13 19 2
Print numbers from process 3: 11 6 10 19
Print numbers from process 4: 10 13 9 10
Print numbers from process 5: 15 10 15 5
Print numbers from process 6: 14 16 8 10
Print numbers from process 7: 17 7 5 3
processor improfeo, process 0 time = 0.0005722
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

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

```cpp
#include <stdio.h>
#include "mpi.h"
#include <ctime>
using namespace std;

int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv);
    int rank, n, i, len;
    char *name = new char;
    double time_start, time_finish;
    int message[5];
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &len);
    time_start = MPI_Wtime();
    if (rank == 0)
    {
        cout << "Start main process " << rank << endl;
        for (i = 1; i < n; i++)
        {

            MPI_Recv(&message[0], 5, MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            cout << "Print numbers from process " << message[4] << ": ";
            for (int j = 0; j < 4; j++)
            {
                cout << message[j] << ' ';
            }
            cout << endl;
        }
    }
    else
    {
        srand(time(NULL) + rank);
        int sending[5];
        for (int i = 0; i < 4; i++)
        {
            sending[i] = 1 + rand() % 20;
        }
        sending[4] = rank;

        MPI_Send(&sending[0], 5, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    time_finish = MPI_Wtime();
    cout << "processor " << name << ", process " << rank << " time = " << time_finish - time_start << endl;
    MPI_Finalize();
    return 0;
}
```

Assignment5 part II code

I've implemented counting working time of each process as it was implemented in Assignment5.c. As we expected, the working time of process 0 is the biggiest, because main process waiting for other processes's results.

1.4 Appendix

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