

FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

MPI. Assignments 18 — 19

Parallel algorithms for the analysis and synthesis of data

Performed by
Aleksandr Shirokov

J4133c

Accepted by
Petr Andriushchenko

Deadline: 25.12.21

St. Petersburg

2021

Contents

1	Assignments	2
1.1	Assignment 18. MPI. Operations with communicators. Renumbering processes. . .	2
1.1.1	Formulation of the problem	2
1.1.2	Example of launch parameters and output. Detailed description of solution .	2
1.2	Assignment 19. MPI. Dynamic process control. Client-server communication. . . .	5
1.2.1	Formulation of the problem	5
1.2.2	Example of launch parameters and output. Detailed description of solution .	5
1.3	Appendix	7

1 Assignments

1.1 Assignment 18. MPI. Operations with communicators. Renumbering processes.

1.1.1 Formulation of the problem

To complete the task, you need to create and compile two programs: Master (MASTER.O) and Slave (SLAVE.O). The Master should start the worker, so be careful with the names of the executable files.

Launch the master via the mpiexec command for one process.

Startup example: `MPIEXEC -N 1 ./MASTER.O`.

Understand the new functions in `ASSIGNMENT18_MASTER.C` and `ASSIGNMENT18_SLAVE.C` and explain programs execution.

Add a third process, which will transfer from the slave processes to the master the number of running processes, the master should receive and display

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

Code for **assignment 18** are [here](#)(master) and [here](#)(slave).

Compilation example:

1. `MPIC++ -O ./CPF/18_MASTER.O ASSIGNMENT18_MASTER.C`
2. `MPIC++ -O ./CPF/18_SLAVE.O ASSIGNMENT18_SLAVE.C`

Launch example: `MPIRUN -OVERSUBSCRIBE -NP 1 ./CPF/18_MASTER.O`

```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpic++ -o ./cpf/18_master.o Assignment18_master.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpic++ -o ./cpf/18_slave.o Assignment18_slave.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 1 ./cpf/18_master.o
slaves 0 and 1 are working
slave process 2: total amount of processes is 3
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

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

```

1  #include "mpi.h"
2  #include <iostream>
3
4  using namespace std;
5
6
7  int main(int argc, char **argv)
8  {
9      int size, rank1, rank2, total;
10     MPI_Status status;
11     MPI_Comm intercomm;
12     char slave[20] = "./cpf/18_slave.o";
13     MPI_Init(&argc, &argv);
14     MPI_Comm_size(MPI_COMM_WORLD, &size);
15     MPI_Comm_spawn(slave, MPI_ARGV_NULL, 3,
16         MPI_INFO_NULL, 0, MPI_COMM_SELF,
17         &intercomm, MPI_ERRCODES_IGNORE);
18     MPI_Recv(&rank1, 1, MPI_INT, 0, 0, intercomm, &status);
19     MPI_Recv(&rank2, 1, MPI_INT, 1, 1, intercomm, &status);
20
21     //Display "Slaves rank1 and rank2 are working",
22     //instead of the words rank1 and rank2 their values should be displayed.
23
24     cout << "slaves " << rank1 <<;
25     cout << " and " << rank2 << " are working" << '\n' << endl;
26
27     MPI_Recv(&total, 1, MPI_INT, 2, 2, intercomm, &status);
28     cout << "slave process " << status.MPI_SOURCE;
29     cout << ": total amount of processes is " << total << endl;
30
31     MPI_Finalize();
32     return 0;
33 }

```

Assignment 18 - master code

```

1  #include "mpi.h"
2  int main(int argc, char **argv)
3  {
4      int rank, size;
5      MPI_Comm intercomm;
6      MPI_Init(&argc, &argv);
7      MPI_Comm_get_parent(&intercomm);
8      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9      MPI_Comm_size(MPI_COMM_WORLD, &size);
10
11     if (rank < 2) MPI_Send(&rank, 1, MPI_INT, 0, rank, intercomm);
12     else MPI_Send(&size, 1, MPI_INT, 0, rank, intercomm);
13     MPI_Finalize();
14     return 0;
15 }

```

Assignment 18 - slave code

In this lab there are two programs - master program and slave program. In master program

there is a new function:

`MPI_COMM_SPAWN(`

- IN `const char *command` - name of program to be spawned (string, significant only at root). In now code our slave program is located in folder `./CPF/18_SLAVE.O`.
- IN `char *argv[]` - arguments to command (array of strings, significant only at root)
- IN `int maxprocs` - maximum number of processes to start (integer, significant only at root)
- IN `MPI_Info info` - a set of key-value pairs telling the runtime system where and how to start the processes (handle, significant only at root)
- IN `int root` - rank of process in which previous arguments are examined (integer)
- IN `MPI_Comm comm` - intracommunicator containing group of spawning processes (handle)
- OUT `MPI_Comm * intercomm` - intercommunicator between original group and the newly spawned group (handle)
- OUT `int array_of_errcodes[]` - one code per process (array of integer)

) which spawn up to `MAXPROCS` instances of a single MPI application. In our program the `maxprocs = 3`. After that the slave program starts, in which there are such a simple logic - if rank of processes is lower than 2 these processes are sending their rank, else process send information about maximum amount of processes in this start. For communication with parent process slaves processes have to use `MPI_COMM_GET_PARENT` function which returns the parent's process communicator. In master program we expected the results from each processes in variables `RANK1`, `RANK2` and `TOTAL` and after that the results are displayed (as shown on picture higher) - as we expected. Two programs works correctly and understandable.

1.2 Assignment 19. MPI. Dynamic process control. Client-server communication.

1.2.1 Formulation of the problem

To complete the task, you need to create and compile two programs: server and client. In one window of the SSH client, a server is launched for one process, which gives out the port name.

An example of a command to start the server: `MPIEXEC -N 1 ./SERV.O`

Then the client is launched in another window, specifying the port name separated by a space in single quotes (example command: `MPIEXEC -N 1 ./CLIENT.O 'PORT NAME'`).

Understand the new functions in `ASSIGNMENT19_SERV.C` and `ASSIGNMENT19_CLIENT.C` and explain programs execution.

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

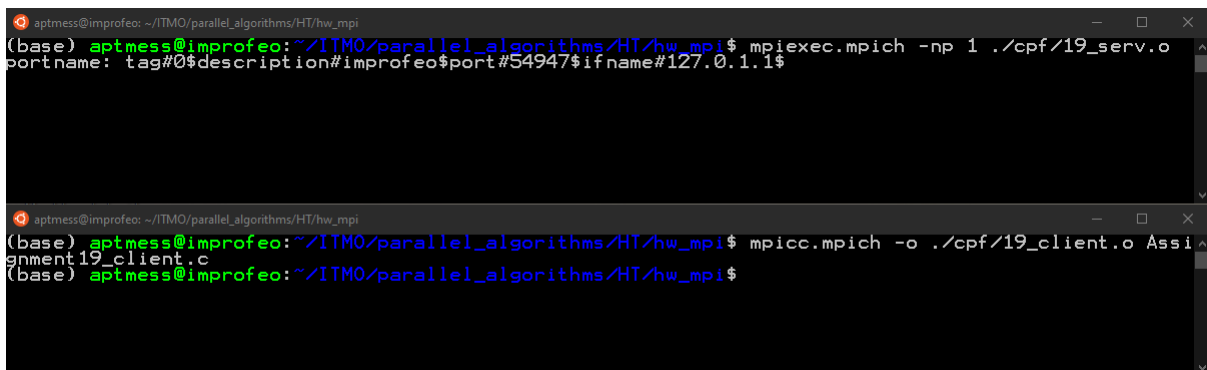
Code for **assignment 18** are [here](#)(server) and [here](#)(client).

Compilation example:

1. `MPICC.MPICH -O ./CPF/19_SERV.O ASSIGNMENT19_SERV.C`
2. `MPICC.MPICH -O ./CPF/19_CLIENT.O ASSIGNMENT19_CLIENT.C`

Firstly we compile two programs with server and client and start a server by command:

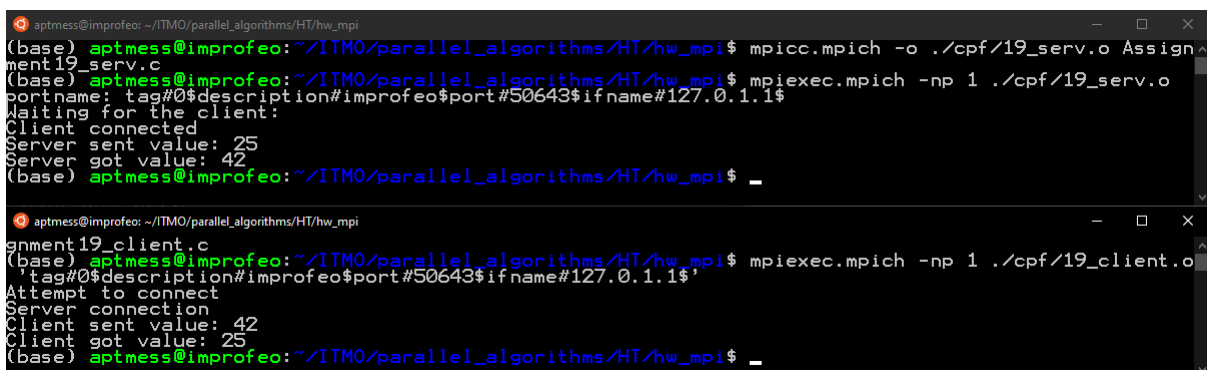
`MPIEXEC.MPICH -NP 1 ./CPF/19_SERV.O` - firstly this, then copy the port to client program



```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpiexec.mpic -np 1 ./cpf/19_serv.o
portname: tag#0$description#improfeo$port#54947$ifname#127.0.1.1$

aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpicc.mpic -o ./cpf/19_client.o Assign
gment19_client.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

After that we are executing client with port as a parameter: `mpiexec.mpic -np 1 ./cpf/19_client.o 'tag#0$description#improfeo$port#50643$ifname#127.0.1.1$'` and got results:



```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpicc.mpic -o ./cpf/19_serv.o Assign
ment19_serv.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpiexec.mpic -np 1 ./cpf/19_serv.o
portname: tag#0$description#improfeo$port#50643$ifname#127.0.1.1$
Waiting for the client:
Client connected
Server sent value: 25
Server got value: 42
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ _

aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpiexec.mpic -np 1 ./cpf/19_client.o
'tag#0$description#improfeo$port#50643$ifname#127.0.1.1$'
Attempt to connect
Server connection
Client sent value: 42
Client got value: 25
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ _
```

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

```
1  #include <stdio.h>
2  #include "mpi.h"
3  int main(int argc, char **argv)
4  {
5      int got, send = 25;
6      MPI_Init(&argc, &argv);
7      char port_name[MPI_MAX_PORT_NAME];
8      MPI_Status status;
9      MPI_Comm intercomm;
10     MPI_Open_port(MPI_INFO_NULL, port_name);
11     printf("portname: %s\n", port_name);
12     printf("Waiting for the client: \n");
13     MPI_Comm_accept(port_name, MPI_INFO_NULL, 0, MPI_COMM_SELF, &intercomm);
14     printf("Client connected \n");
15     MPI_Recv(&got, 1, MPI_INT, 0, 0, intercomm, &status);
16     MPI_Send(&send, 1, MPI_INT, 0, 0, intercomm);
17     MPI_Comm_free(&intercomm);
18     MPI_Close_port(port_name);
19     printf("Server sent value: %d\n", send);
20     printf("Server got value: %d\n", got);
21     MPI_Finalize();
22     return 0;
23 }
```

Assignment 19 - server code

```
1  #include <stdio.h>
2  #include <string.h>
3  #include "mpi.h"
4  int main(int argc, char **argv)
5  {
6      int got, send = 42;
7      MPI_Init(&argc, &argv);
8      char port_name[MPI_MAX_PORT_NAME];
9      MPI_Status status;
10     MPI_Comm intercomm;
11     strcpy(port_name, argv[1]);
12     printf("Attempt to connect \n");
13     MPI_Comm_connect(port_name, MPI_INFO_NULL, 0, MPI_COMM_SELF, &intercomm);
14     printf("Server connection \n");
15
16     MPI_Send(&send, 1, MPI_INT, 0, 0, intercomm);
17     printf("Client sent value: %d\n", send);
18
19     MPI_Recv(&got, 1, MPI_INT, 0, 0, intercomm, &status);
20     printf("Client got value: %d\n", got);
21
22     MPI_Finalize();
23     return 0;
24 }
```

Assignment 19 - client code

In this lab there are a server-client pretty architecture, where firstly in server program with function `MPI_OPEN_PORT` establish an address that can be used to establish connections between

groups of MPI processes, after that using `MPI_COMM_ACCEPT` accept a request to form a communicator and start waiting for the request from client. Then we go to client command and there we establish communication between client and server using `MPI_COMM_CONNECT` by input port and then we send by client a message with value 42 to server. Server got a request, displayed information about it and send to client server's value - 25 which is also displayed on client's side. After that the port is closing and the server and client's end's their programs. The results are shown in picture higher and it is expected by us and explained - two programs works correctly.

1.3 Appendix

The link to the source code which is placed on my [github](#).