FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

MPI. Assignments 16 – 17

Parallel algorithms for the analysis and synthesis of data

Performed by
Aleksandr Shirokov
J4133c
Accepted by
Petr Andriushchenko
Deadline: 24.12.21

St. Petersburg
2021

Contents

1 Assignments

1.1 Assignment 16. MPI. Operations with communicators. Renumbering processes.

1.1.1 Formulation of the problem

In the MPI_Comm_split function (Assignment16.c), replace the color parameter with (rank% 2), (rank% 3), look at how many groups the processes are split into, depending on the specified attribute of division into groups.

int **MPI_Comm_split** (

- IN MPI_Comm **comm** - parent communicator
- IN int **color** - a sign of division into groups
- IN int **key** - parameter defining numbering in new groups
- OUT MPI_Comm ***newcomm** - new communicator

)

The function splits the group associated with the parent communicator into non-overlapping subgroups, one for each value of the color subgroup attribute. **Color** must be non-negative. Each subgroup contains processes with the same color value. The **key** parameter controls the ordering within the new groups: a lower **key** value corresponds to a lower process ID value. If the **key** parameter is equal for multiple processes, the ordering is performed according to the order in the parent group

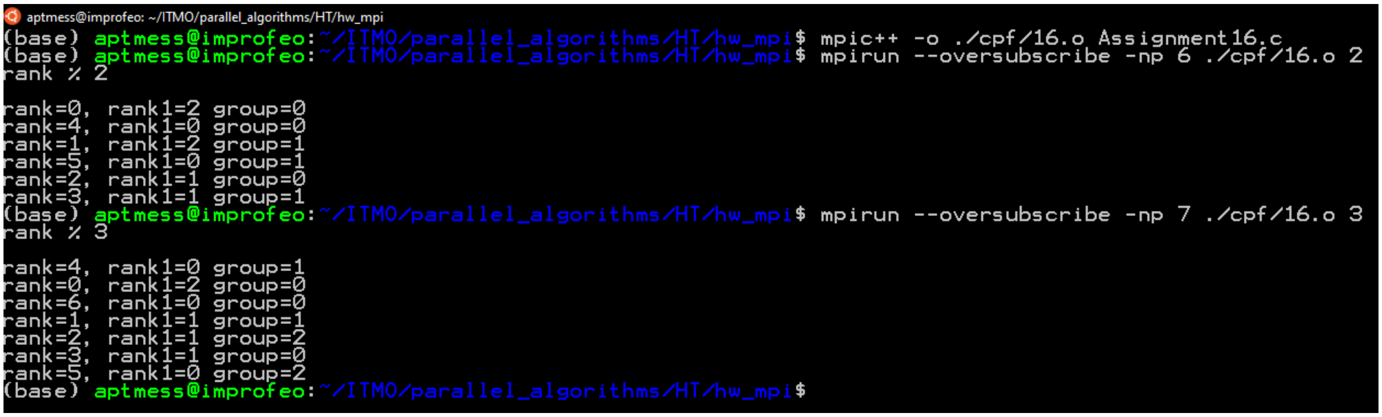
1.1.2 Example of launch parameters and output. Detailed description of solution

Code for **assignment 16** is here.

Compilation example: mpic++ -o ./cpf/16.o Assignment16.c

Launch example:

1. mpirun –oversubscribe -np 4 ./cpf/16.o 2
2. mpirun –oversubscribe -np 4 ./cpf/16.o 3

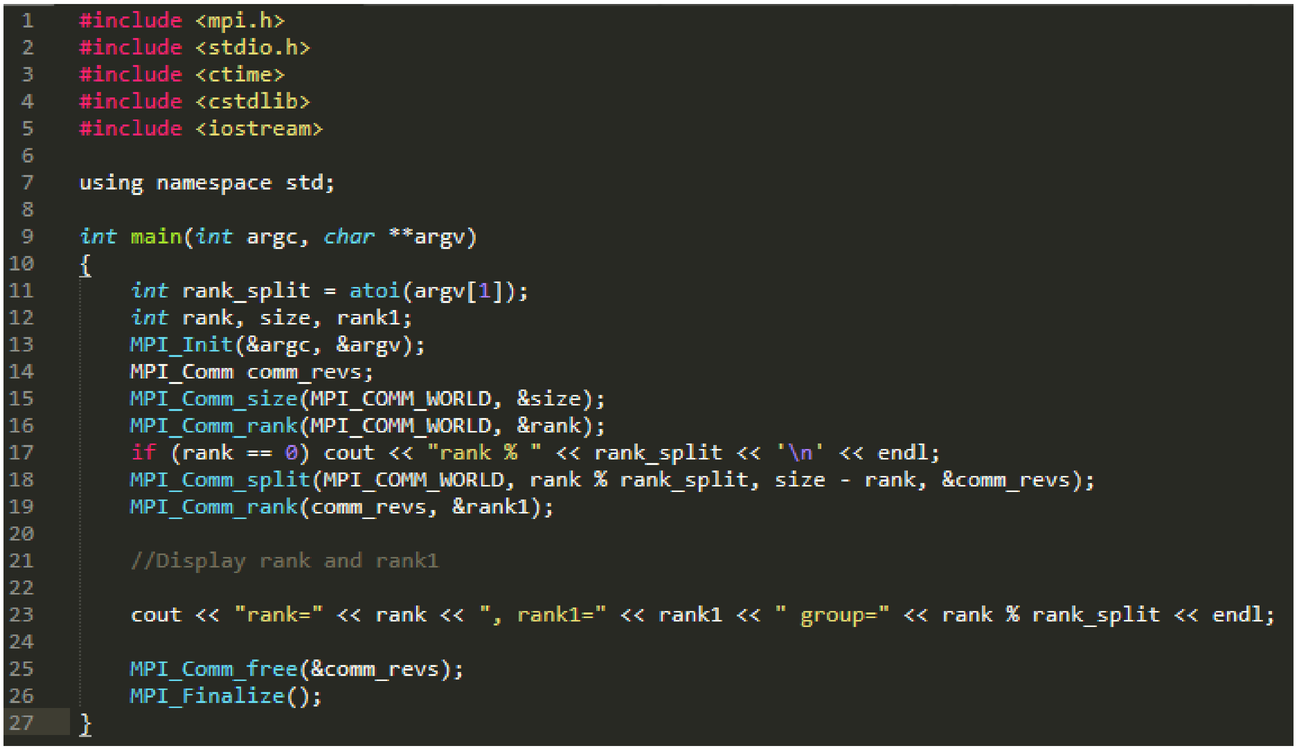


```
aptmess@improfeo: ~/ITMO/parallel_algorithms/HT/hw_mpi
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpic++ -o ./cpf/16.o Assignment16.c
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 6 ./cpf/16.o 2
rank % 2

rank=0, rank1=2 group=0
rank=4, rank1=0 group=0
rank=1, rank1=2 group=1
rank=5, rank1=0 group=1
rank=2, rank1=1 group=0
rank=3, rank1=1 group=1
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 7 ./cpf/16.o 3
rank % 3

rank=4, rank1=0 group=1
rank=0, rank1=2 group=0
rank=6, rank1=0 group=0
rank=1, rank1=1 group=1
rank=2, rank1=1 group=2
rank=3, rank1=1 group=0
rank=5, rank1=0 group=2
(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 rank_split = atoi(argv[1]);
    int rank, size, rank1;
    MPI_Init(&argc, &argv);
    MPI_Comm comm_revs;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) cout << "rank % " << rank_split << '\n' << endl;
    MPI_Comm_split(MPI_COMM_WORLD, rank % rank_split, size - rank, &comm_revs);
    MPI_Comm_rank(comm_revs, &rank1);

    //Display rank and rank1

    cout << "rank=" << rank << ", rank1=" << rank1 << " group=" << rank % rank_split << endl;

    MPI_Comm_free(&comm_revs);
    MPI_Finalize();
}
```

Assignment16 code

This program works clearly - all processes are splitted into groups based on some condition such as color = rank %2 or color = rank %3 and the new rank in group is calculated as size $-rank$ as a RANK1 variable with process rank in new group. For example for second run example, with initial 7 processes they are splitted on three groups: $0 : \{0, 3, 6\}, 1 : \{1, 4\}, 2 : \{3, 5\}$ and new ranks are $0 : \{2, 1, 0\}, 1 : \{1, 0\}, 2 : \{1, 0\}$ as on a screen higher. The program is tested and works correctly.

1.2 Assignment 17. MPI. Data packing. Sending packed data.

1.2.1 Formulation of the problem

Understand the new functions in Assignment17.c and explain program execution.

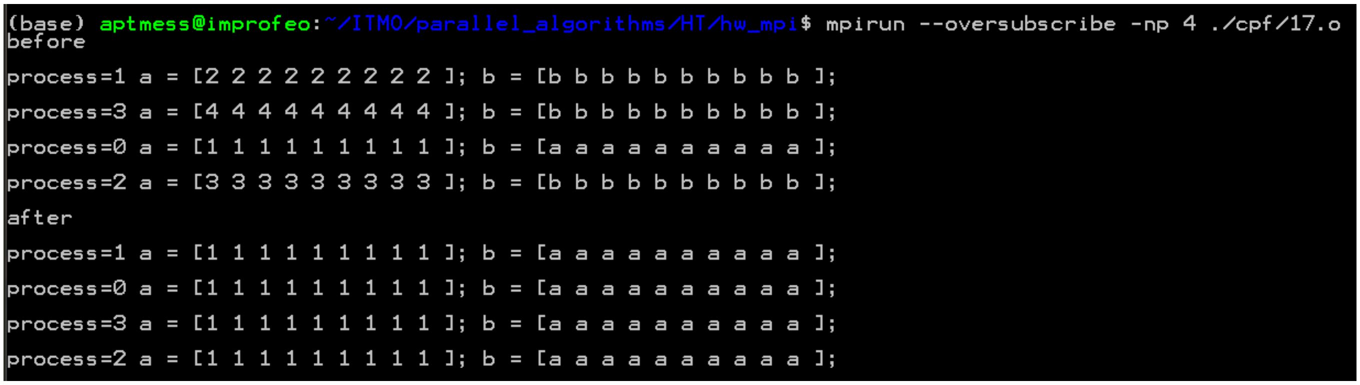
Display the values of the process number and arrays $a[i]$, $b[i]$, before packing and distribution, and after. See how broadcasting works.

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

Code for **assignment 17** is here.

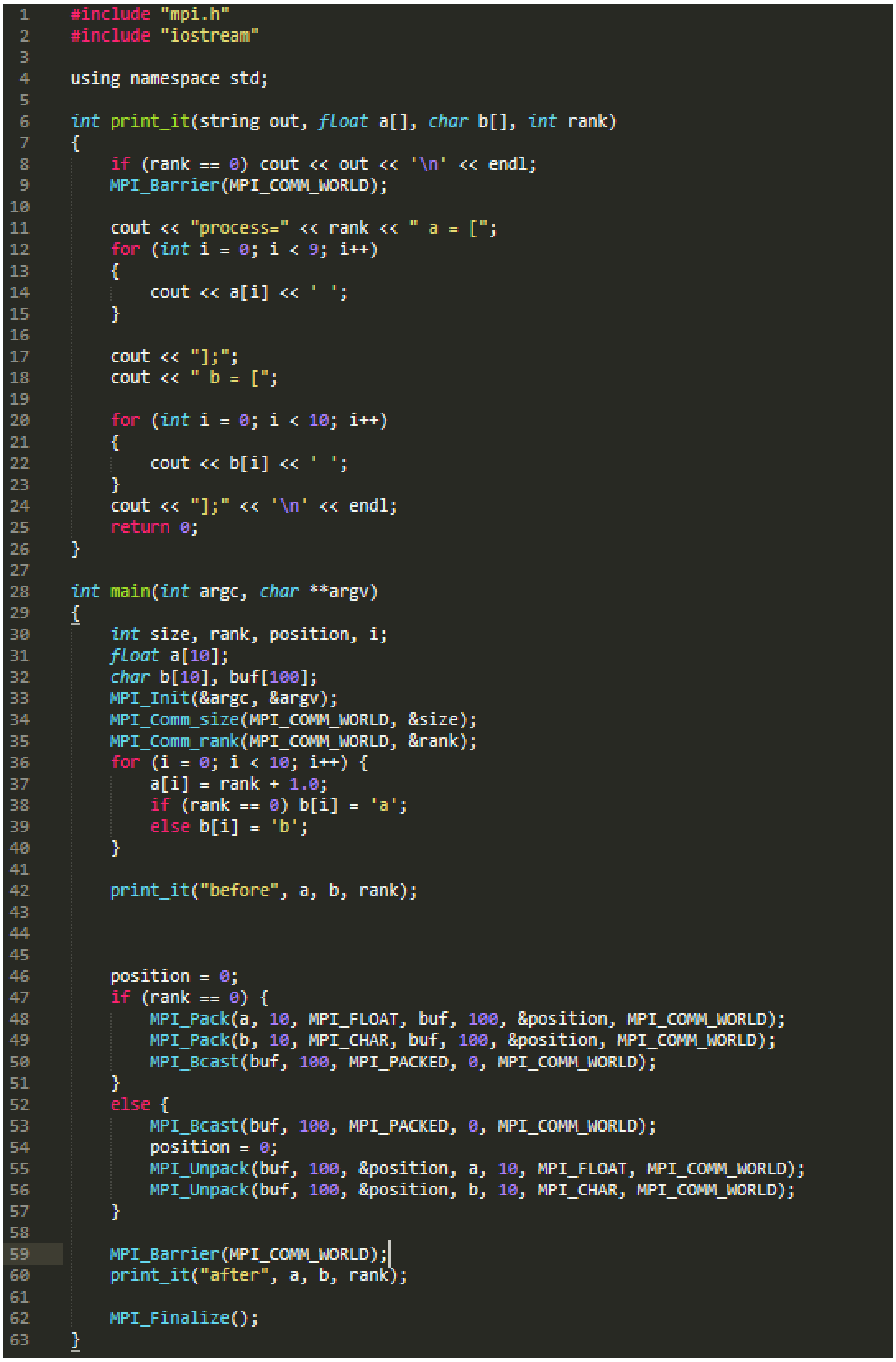
Compilation example: mpic++ -o ./cpf/17.o Assignment17.c

Launch example: mpirun –oversubscribe -np 4 ./cpf/17.o



```
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 4 ./cpf/17.o
before
process=1 a = [2 2 2 2 2 2 2 2 2 ]; b = [b b b b b b b b b b ];
process=3 a = [4 4 4 4 4 4 4 4 4 ]; b = [b b b b b b b b b b ];
process=0 a = [1 1 1 1 1 1 1 1 1 ]; b = [a a a a a a a a a a ];
process=2 a = [3 3 3 3 3 3 3 3 3 ]; b = [b b b b b b b b b b ];
after
process=1 a = [1 1 1 1 1 1 1 1 1 ]; b = [a a a a a a a a a a ];
process=0 a = [1 1 1 1 1 1 1 1 1 ]; b = [a a a a a a a a a a ];
process=3 a = [1 1 1 1 1 1 1 1 1 ]; b = [a a a a a a a a a a ];
process=2 a = [1 1 1 1 1 1 1 1 1 ]; b = [a a a a a a a a a a ];
```

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



```
#include "mpi.h"
#include "iostream"

using namespace std;

int print_it(string out, float a[], char b[], int rank)
{
    if (rank == 0) cout << out << '\n' << endl;
    MPI_Barrier(MPI_COMM_WORLD);

    cout << "process=" << rank << " a = [";
    for (int i = 0; i < 9; i++)
    {
        cout << a[i] << ' ';
    }

    cout << "];";
    cout << " b = [";

    for (int i = 0; i < 10; i++)
    {
        cout << b[i] << ' ';
    }
    cout << "];" << '\n' << endl;
    return 0;
}

int main(int argc, char **argv)
{
    int size, rank, position, i;
    float a[10];
    char b[10], buf[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    for (i = 0; i < 10; i++) {
        a[i] = rank + 1.0;
        if (rank == 0) b[i] = 'a';
        else b[i] = 'b';
    }

    print_it("before", a, b, rank);



    position = 0;
    if (rank == 0) {
        MPI_Pack(a, 10, MPI_FLOAT, buf, 100, &position, MPI_COMM_WORLD);
        MPI_Pack(b, 10, MPI_CHAR, buf, 100, &position, MPI_COMM_WORLD);
        MPI_Bcast(buf, 100, MPI_PACKED, 0, MPI_COMM_WORLD);
    }
    else {
        MPI_Bcast(buf, 100, MPI_PACKED, 0, MPI_COMM_WORLD);
        position = 0;
        MPI_Unpack(buf, 100, &position, a, 10, MPI_FLOAT, MPI_COMM_WORLD);
        MPI_Unpack(buf, 100, &position, b, 10, MPI_CHAR, MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
    print_it("after", a, b, rank);

    MPI_Finalize();
}
```

Assignment17 code

Theare are three new functions in the code:

- int MPI_Pack that packs a datatype into contiguous memory(
    - const void *inbuf - input buffer start (choice)

  - int incount - number of input data items (non-negative integer)
  - MPI_Datatype datatype - datatype of each input data item (handle)
  - OUT - void *outbuf - output buffer start (choice)
  - int outsize - output buffer size, in bytes (non-negative integer)
  - IN/OUT int *position - current position in buffer, in bytes (integer)
  - MPI_Comm comm - communicator for packed message (handle)

- int MPI_Unpack that unpack a buffer according to a datatype into contiguous memory(
  - const void *inbuf - input buffer start (choice)
  - int insize - size of input buffer, in bytes (integer)
  - IN/OUT int *position - current position in buffer, in bytes (integer)
  - OUT - void *outbuf - output buffer start (choice)
  - int outcount - number of items to be unpacked (integer)
  - MPI_Datatype datatype - datatype of each output data item (handle)
  - MPI_Comm comm - communicator for packed message (handle)

- MPI_Bcast broadcasts a message from the main process ($rank = 0$) to all other processes of the communicator

Due to this our function works like this - there are initliazation of two arrays, first array a is filled by formula 'current rank + 1' and the other b contains char 'a' if it is root process and 'b' if not. After that i am printing array and if it is root process array a and b due to function MPI_Pack are packing into one contiguous memory and after packing thia arrays are cast to other processes using MPI_Bcast. In others processes using MPI_Unpack function we are unpacking a buffer from process 0 accodring to datatypeto contigious memory and as we expected the messages in non root processes in arrays a and b are overwritten by root values in this arrays. The proram is explained and works correctly.

1.3 Appendix

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