FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

MPI. Assignments 9

Parallel algorithms for the analysis and synthesis of data

Performed by
Aleksandr Shirokov
J4133c
Accepted by
Petr Andriushchenko
Deadline: 21.12.21

St. Petersburg
2021

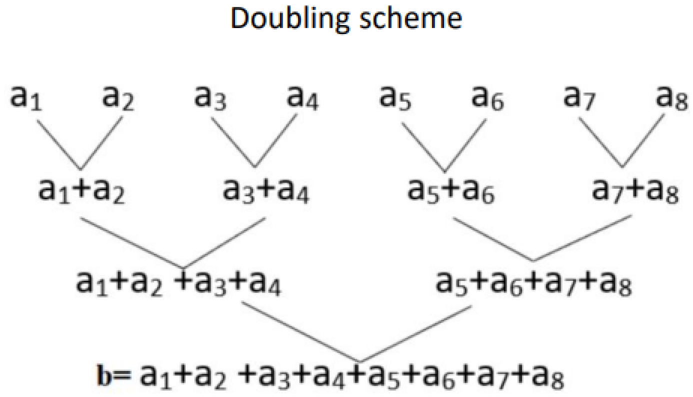
Contents

1 Assignments

1.1 Assignment 9. MPI. MPI_Reduce.

1.1.1 Formulation of the problem

1. Write an MPI program in which the global vector addition operation is modeled by a doubling (cascade) scheme using point-to-point data transfers.
2. Compare the execution time of such a simulation using the MPI_REDUCE procedure on as many processes as possible. Each process stores an array of $1,000,000$ elements equal to 1.



MPI_Reduce

The MPI_REDUCE function concatenates the input buffer entries of each process in a group using the **op** operation and returns the concatenated value to the root process's output buffer.

int **MPI_Reduce** (

- IN void ***sendbuf** - address of the beginning of the input buffer;
- OUT void ***recvbuf** - address of the beginning of the result buffer (used only in the receiving process root);
- IN int **count** - the number of elements in the input buffer;
- IN MPI_Datatype **sendtype** - the type of elements in the input buffer;
- IN MPI_Op **op** - the operation by which the reduction is performed;
- IN int **root** - number of the receiving process of the operation result;
- IN MPI_Comm **comm** - communicator

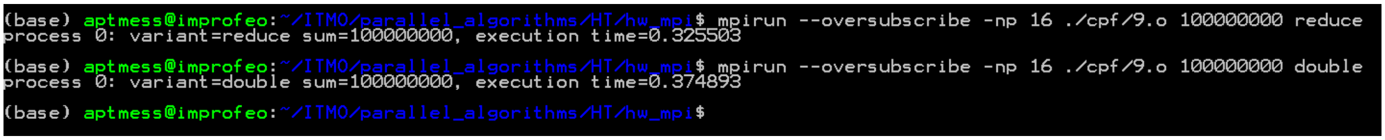
)

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

Code for **assignment 9** is here.
Compilation example: mpic++ -o ./cpf/9.o Assignment9.c
Launch example, there are two options:

1. mpirun –oversubscribe -np 16 ./cpf/9.o 100000000 double

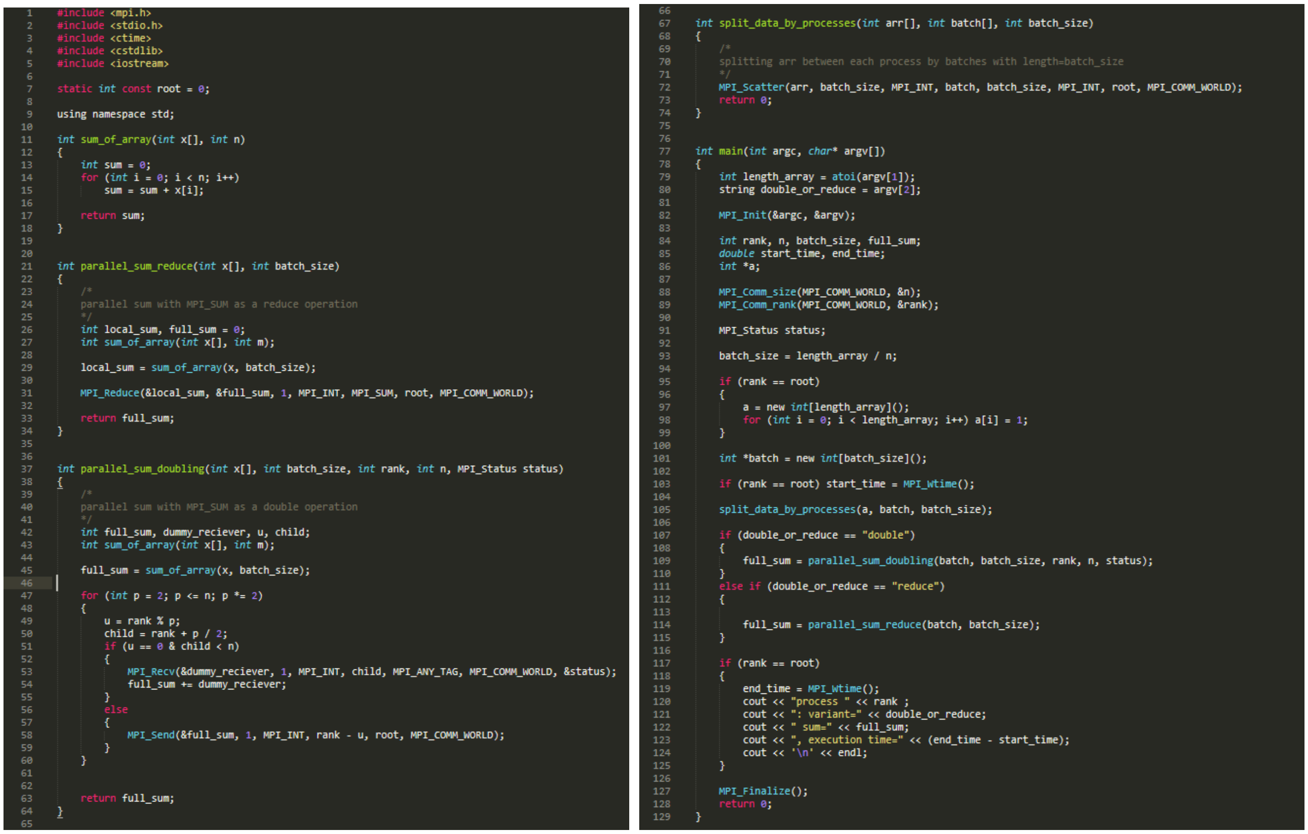
2. mpirun –oversubscribe -np 16 ./cpf/9.o 100000000 reduce



```
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 16 ./cpf/9.o 100000000 reduce
process 0: variant=reduce sum=100000000, execution time=0.325503
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$ mpirun --oversubscribe -np 16 ./cpf/9.o 100000000 double
process 0: variant=double sum=100000000, execution time=0.374893
(base) aptmess@improfeo:~/ITMO/parallel_algorithms/HT/hw_mpi$
```

Results

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>

static int const root = 0;

using namespace std;

int sum_of_array(int x[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum = sum + x[i];

    return sum;
}


int parallel_sum_reduce(int x[], int batch_size)
{
    /*
    parallel sum with MPI_SUM as a reduce operation
    */
    int local_sum, full_sum = 0;
    int sum_of_array(int x[], int m);

    local_sum = sum_of_array(x, batch_size);

    MPI_Reduce(&local_sum, &full_sum, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD);

    return full_sum;
}


int parallel_sum_doubling(int x[], int batch_size, int rank, int n, MPI_Status status)
{
    /*
    parallel sum with MPI_SUM as a double operation
    */
    int full_sum, dummy_reciever, u, child;
    int sum_of_array(int x[], int m);

    full_sum = sum_of_array(x, batch_size);

    for (int p = 2; p <= n; p *= 2)
    {
        u = rank % p;
        child = rank + p / 2;
        if (u == 0 & child < n)
        {
            MPI_Recv(&dummy_reciever, 1, MPI_INT, child, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            full_sum += dummy_reciever;
        }
        else
        {
            MPI_Send(&full_sum, 1, MPI_INT, rank - u, root, MPI_COMM_WORLD);
        }
    }


    return full_sum;
}
```

```cpp

int split_data_by_processes(int arr[], int batch[], int batch_size)
{
    /*
    splitting arr between each process by batches with length=batch_size
    */
    MPI_Scatter(arr, batch_size, MPI_INT, batch, batch_size, MPI_INT, root, MPI_COMM_WORLD);
    return 0;
}


int main(int argc, char* argv[])
{
    int length_array = atoi(argv[1]);
    string double_or_reduce = argv[2];

    MPI_Init(&argc, &argv);

    int rank, n, batch_size, full_sum;
    double start_time, end_time;
    int *a;

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

    MPI_Status status;

    batch_size = length_array / n;

    if (rank == root)
    {
        a = new int[length_array]();
        for (int i = 0; i < length_array; i++) a[i] = 1;
    }

    int *batch = new int[batch_size]();

    if (rank == root) start_time = MPI_Wtime();

    split_data_by_processes(a, batch, batch_size);

    if (double_or_reduce == "double")
    {
        full_sum = parallel_sum_doubling(batch, batch_size, rank, n, status);
    }
    else if (double_or_reduce == "reduce")
    {

        full_sum = parallel_sum_reduce(batch, batch_size);
    }

    if (rank == root)
    {
        end_time = MPI_Wtime();
        cout << "process " << rank ;
        cout << ": variant=" << double_or_reduce;
        cout << " sum=" << full_sum;
        cout << ", execution time=" << (end_time - start_time);
        cout << '\n' << endl;
    }

    MPI_Finalize();
    return 0;
}
```

Assignment9 code

In this code there are two functions base functions - parallel_sum_reduce and parallel_sum_doubling. In first function we are for each process and for their part of input array compute sum and send the result to toor process using syntax of MPI_Reduce, in parallel_sum_doubling function we are splitting our workers as a tree (amount of processes should be a degree of 2 because of implemenation condition). After that if there are a way to split worker to more workers less than amount of processes we are splitting and waiting for result of each child process, else we are sending sum - the structure as on picture in the previous subsection. In main functions we initialise array of ones, with function MPI_Scatter send for each process their own part of array and depending on input parameter counting the sum. After some expereiements I mentioned that reduce operation worke quicklier than doubling (as we can see in picture Results).

1.2 Appendix

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