P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2018

Set 8 - MPI: Blocking Communication

Issued: November 16, 2018
Hand in (optional): November 26, 2018 23:59

Question 1: Implementing a distributed reduction

In this question, you will use MPI to calculate the following sum:

$$x_{\text{tot}} = \sum_{n=1}^{N} n = 1 + 2 + 3 + \ldots + (N-1) + N \tag{1}$$

a) Fill in the missing part in the Makefile in order to compile the skeleton code with MPI support.

b) Validation of HPC code is an important subject. For example, there is an analytic formula for the above sum. Use this to check if your implementation is correct.
To this end, implement the function `exact(N)`.
Hint: A young C.F. Gauss found the formula in elementary school.

c) Initialize and finalize MPI by filling the corresponding gaps in the skeleton code.

d) Each rank performs only a part of the sum. Distribute the work load reasonably in order to guarantee load balancing. Each rank should calculate the subsum

$$\text{sum}_{\text{rank}} = N_{\text{start}} + (N_{\text{start}} + 1) + \ldots + N_{\text{end}}, \tag{2}$$

where N_{start} and N_{end} are the corresponding variables in the skeleton file.

e) Finally, implement your own reduction. This can be done in a tree-like way as depicted in Fig. 1. Your task is to implement this scheme for the special case that the number of ranks is a power of 2, i. e.

$$|\text{ranks}| = 2^l, l \in \mathbb{N}_0 \tag{3}$$

f) What is the advantage of this scheme compared to the naive reduction? Name 2 advantages and quickly justify your answer.
Hint: In the naive approach, every rank sends its elements directly to the master. The master then reduces all obtained elements by repeatedly applying the operation, in our case the sum.

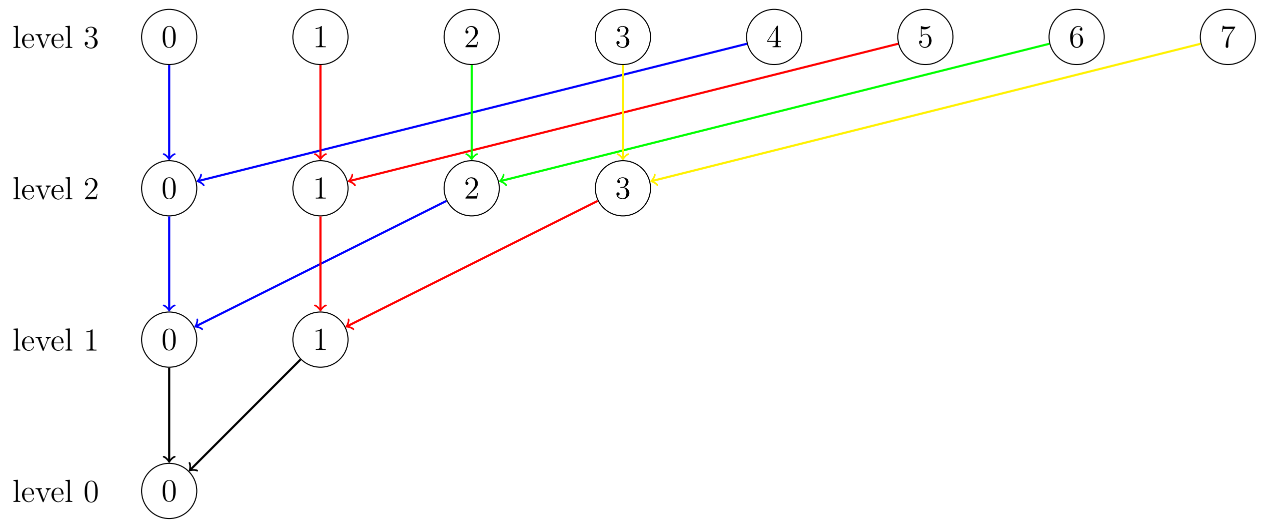

Figure 1: The communication pattern of a tree-like reduction. Each circle represents a rank, the number inside is the rank ID. Communication takes place along the arrows.

Question 2: MPI Bug Hunt

Find the bug(s) in the following MPI code snippets and find a way to fix the problem!

a)
```cpp
const int N = 10000;
double* result = new double[N];
// do a very computationally expensive calculation
// ...

// write the result to a file
std::ofstream file("result.txt");

for(int i = 0; i <= N; ++i){
    file << result[i] << std::endl;
}

delete[] result;
```

b)
```cpp
// only 2 ranks: 0, 1
double important_value;

// obtain the important value
// ...

// exchange the value
if(rank == 0)
    MPI_Send(&sum, 1, MPI_DOUBLE, 1, 123, MPI_COMM_WORLD);
else
    MPI_Send(&sum, 1, MPI_DOUBLE, 0, 123, MPI_COMM_WORLD);

MPI_Recv(
    &sum, 1, MPI_INT, MPI_ANY_SOURCE,
    MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);

// do other work
```

c) What is the output of the following program when run with 1 rank? What if there are 2 ranks? Will the program complete for any number of ranks?

```
MPI_Init(&argc, &argv);

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

int bval;
if (0 == rank)
{
    bval = rank;
    MPI_Bcast(&bval, 1, MPI_INT, 0, MPI_COMM_WORLD);
}
else
{
    MPI_Status stat;
    MPI_Recv(&bval, 1, MPI_INT, 0, rank, MPI_COMM_WORLD, &stat);
}

cout << "[" << rank << "] " << bval << endl;

MPI_Finalize();
return 0;
```