P. Koumoutsakos
S. Martin
ETH Zentrum, ML H44
CH-8092 Zürich

Fall semester 2021

Set 6 -MPI

Issued: December 10, 2021
Hand in (optional): December 24, 2021 12:00

Question 1: MPI Bug Hunt (16 points)

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(&important_value, 1, MPI_DOUBLE, 1, 123, MPI_COMM_WORLD);
else
    MPI_Send(&important_value, 1, MPI_DOUBLE, 0, 123, MPI_COMM_WORLD);

MPI_Recv(
    &important_value, 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;
```

Question 2: Implementing a distributed reduction (45 points)

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 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.

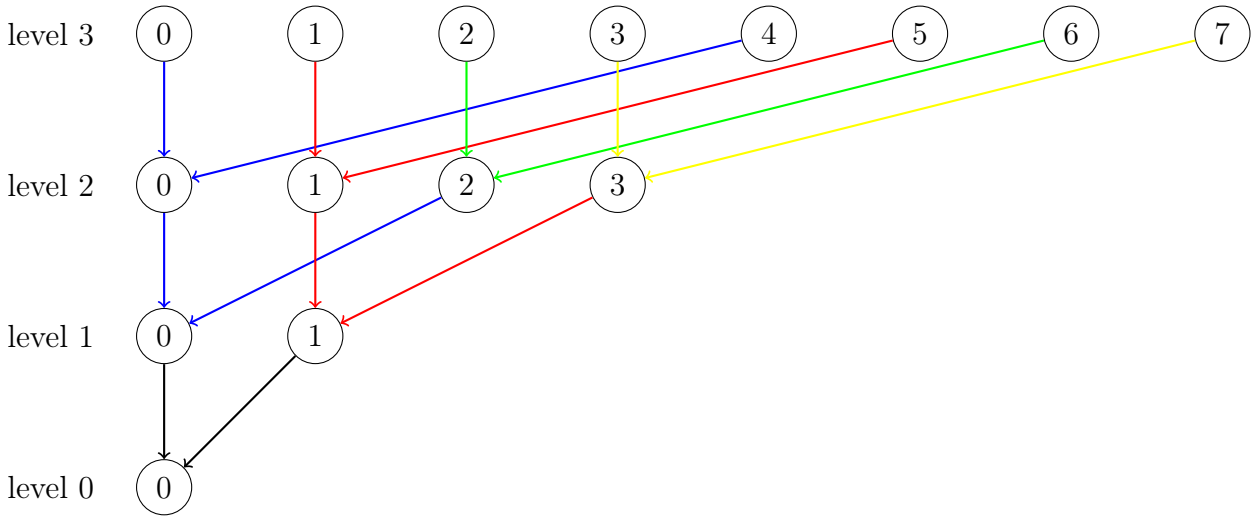

Figure 1: The communication pattern of a tree-like reduction. Each circle represents a rank, the number inside is the rank ID. Communication takes place along the arrows.

e) First Perform the reduction using MPI blocking collectives in the skeleton code. Then, 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 2: Left: Wake of an airplane visualized by condensation behind the engines. The vorticity sheet is generated at the trailing edge of the wings. Right: Vortex sheet at $t = 1$ from the simulation.

Question 3: Roll-up of a vortex line (30 points)

We want to simulate the evolution of a two-dimensional vorticity sheet similar to the wake of an airplane (Figure 2). We use N particles moving with time t, each particle i is located at $(x_i(t), y_i(t))$ and carries a constant value Γ_i (circulation). The velocity field defined as

$$u(x_j, y_j, t) = \sum_{i=0}^{N-1} \frac{\Gamma_i}{2\pi} \frac{-[y_j(t) - y_i(t)]}{\varepsilon^2 + [x_j(t) - x_i(t)]^2 + [y_j(t) - y_i(t)]^2},$$
$$v(x_j, y_j, t) = \sum_{i=0}^{N-1} \frac{\Gamma_i}{2\pi} \frac{[x_j(t) - x_i(t)]}{\varepsilon^2 + [x_j(t) - x_i(t)]^2 + [y_j(t) - y_i(t)]^2}$$

is an approximate solution of Euler equations and corresponds to the vorticity field

$$\omega_j = \frac{\partial v(x_j, y_j, t)}{\partial x_j} - \frac{\partial u(x_j, y_j, t)}{\partial y_j} = \sum_{i=0}^{N-1} \Gamma_i \delta_\epsilon\big(x_j(t) - x_i(t), y_j(t) - y_i(t)\big)$$

where $\delta_\epsilon(x, y) = \frac{1}{\pi} \frac{\epsilon^2}{(\epsilon^2 + x^2 + y^2)^2}$ approximate a Dirac-delta function for a small ϵ. A particle moves with the velocity at its location

$$\frac{dx_i(t)}{dt} = u(x_i(t), y_i(t), t), \quad \frac{dy_i(t)}{dt} = v(x_i(t), y_i(t), t),$$

Initially, particles are placed at $y_i = 0$ and

$$x_i = -\frac{1}{2} + \frac{i + 1/2}{N}, \quad i = 0, ..., N - 1$$

and have circulation $\Gamma_i = \frac{1}{N} \frac{4x_i}{\sqrt{1 - 4x_i^2}}$

a) Implement the interaction function `computeVelocities` in `q3/serial.cpp`. You may check your results by visualizing the csv files with paraview.

b) Parallelize your code using MPI by filling in the TODOs in `q3/mpi.cpp`. Each MPI rank must contain an equal number of particles. The parallelization of the `computeVelocities` can be done using a multi-pass communication, as described in the table:

		process p				
		0	1	$\cdots$	P-2	P-1
	0	D_0	D_1	$\cdots$	D_{P-2}	D_{P-1}
	1	D_{P-1}	D_0	$\cdots$	D_{P-3}	D_{P-2}
pass q	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	P-1	D_1	D_2	$\cdots$	D_{P-1}	D_0

the data necessary to compute the velocities (x_i, y_i and Γ_i) must be communicated to every rank in a cyclic manner until every rank has computed the interactions between its own particles with every particles in the simulation. Parallelize the function `dumpToCsv` by gathering the data on the root.

c) Use non blocking MPI routines to overlap the communication with computation. Write your solution in `q3/mpi_non_blocking.cpp`.