ETH zürich **High Performance Computing for Science and Engineering I**

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

Fall semester 2017

Set 10 - MPI I/O, Hybrid MPI and OpenMP

Issued: December 01, 2017
Hand in (optional): December 08, 2017 8:00am

Question 1: Diffusion and MPI I/O

In the provided skeleton code `Diffusion/diffusion2d_mpi.cpp`, you find a simplified version, with slightly modified initial conditions, of the MPI code for the 2D diffusion problem.

a) Use MPI I/O in `write_density_mpi()` to generate a single **binary** file for all density values.

Hints:

- You do not need to store the two rows that contain the ghost data of each rank.
- Given that you write only doubles in the output file, you can use a binary reader or the printdata.sh script to check your implementation. Adjust the problem size and the number of ranks to simplify your checks.
- Verify your solution for larger number of processes by comparing the output binary file with that generated by the single process experiment (but for the same problem size and number of steps).

Question 2: Hybrid Quadratic Form

In a previous exercise (#5), we computed the quadratic form

$$Q = \vec{v}^T \cdot A \cdot \vec{w} = \sum_{ij} v_i A_{ij} w_j \tag{1}$$

in parallel employing shared memory parallelism with OpenMP. The OpenMP code is also provided in `QuadraticForm/main_shared.cpp`.

a) Parallelize the code for a distributed memory architecture. Using a block-row distribution every process must initialize only the needed portion of the objects v, w, A. Assume that n is a multiple of the number of processes.

   Put your solution in `QuadraticForm/main_distributed.cpp` and extend the Makefile accordingly.

b) Combine shared and distributed memory parallelism in an hybrid program. Put your solution in `QuadraticForm/main_hybrid.cpp` and extend the Makefile accordingly.

Hints:

- Initialize the MPI library with the appropriate thread safety level.
- Block-row distribution: distribute the rows of the matrix to the available processes (ranks), as performed in the MPI diffusion code.
- As already mentioned, allocate and initialize only the portion of data needed by each rank.
- Take care of the different range of loop indices when you perform data initialization.
- Optimize MPI communication whenever this is possible.
- The application must produce the same output messages regardless of the implementation approach (i.e. only one rank prints).

Question 3: Non-blocking MPI Barrier

MPI_Ibarrier() is a non-blocking version of MPI_Barrier(). By calling MPI_Ibarrier(), a process notifies that it has reached the barrier. The call returns immediately, independently of whether other processes have called MPI_Ibarrier().

An example of MPI_Ibarrier() is given in the following code:

```
MPI_Ibarrier(MPI_COMM_WORLD, &request);
work(); /* computation and possibly MPI communication */
MPI_Wait(&request, &status);
```

Use OpenMP to implement an equivalent version of the above code for cases where only the blocking version of MPI_Barrier() is provided by the MPI library. The initial single-threaded code is as follows:

```
MPI_barrier(MPI_COMM_WORLD);
work(); /* computation and possibly MPI communication */
```

Write your solution code and show or state any other requirements that must be satisfied.

Hints:

- Ensure that your solution does not rely on assumptions that if not satisfied can introduce correctness issues.