Technische Universität Graz
Institut für Angewandte Mathematik
Assoc. Prof. Dr. Günther Of

Numerics and Simulation
Elective subject mathematics (DDM)

Exercise sheet 5, May 18, 2022

Exercise 17: Compile the MPI program `helloworld.c` discussed in the lecture and run it with several processes. Report the output. Modify the program such that process 0 can receive the messages in arbitrary order. Report the related output.

Exercise 18: Write an MPI program realizing ping-pong, where two processes send and receive a message (of `NR` double values) alternatingly:

- Process 0 sends a message to process 1 (ping).
- After receiving the message process 1 sends a message to process 0 (pong).

Implement this operation in a loop of 50 iterations and enhance your program by an internal time measurement. Provide a table of the communication times for message lengths `NR` $\in \{1, 10, 100, 1000, 10000, 100000\}$. Run the program several times for each value to get reliable numbers. Rate these numbers.

Exercise 19: Consider a given vector $\underline{x} \in \mathbb{R}^n$, which is distributed blockwise by

$$\underline{x} = \begin{pmatrix} \underline{x}_1 \\ \vdots \\ \underline{x}_p \end{pmatrix}$$

to p processes, where $\underline{x}_i \in \mathbb{R}^m$ and $pm = n$. Compute the norms $\|\underline{x}\|_1$, $\|\underline{x}\|_2$ and $\|\underline{x}\|_\infty$ in a parallel program using `MPI_Allreduce`. To this end parallelize the sequential code `norm.c`. Test your program with several processes. Verify and document its correctness by comparison to the results of the sequential program.

Exercise 20: Implement an example for every collective MPI operation and print the results. Use 3 processes and the two prescribed integer values for each process. If a collective MPI operation requires different data structures, choose their lengths meaningful and the values such that they can be identified in the output. Use

```
printf("NameOp %d Input ...", myrank, ...);
printf("NameOp %d Output ...", myrank, ...);
fflush(stdout);
sleep(1);
MPI_Barrier(MPI_COMM_WORLD);
```

for a clear output, see `kollektiv.c`. Record and provide the output.

Exercise 21: Implement a MPI version of the finite difference method discussed in the lecture. Start from the sequential code `bvp.c`. Test your implementation for correctness. Provide computational times for different numbers of processes and a meaningful problem size. Choose a sufficiently small time step size. Which speedup do you observe?

Exercise 22: Parallelize the matrix generation and the matrix vector multiplication in the sequential program `simplebem.c` by MPI. For simplicity, run the CG method on process 0 only. Test your implementation for correctness: Provide the approximate solution for 16 boundary elements for the sequential and the parallel version.
Provide a table of computational times of the matrix generation and the solution of the linear system for a meaningful problem size. Which speedup do you observe?