Introduction to Parallel and Distributed Computing Exercise 4 (June 25)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

May 25, 2010

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- a PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the source code of the sequential program,
  - the demonstration of a sample solution of the program,
  - the source code of the parallel program,
  - the demonstration of a sample solution of the program,
  - a benchmark of the sequential and of the parallel program in the style of Exercise 2.
- the source (`.c`/`.f`) files of the sequential program and of the parallel program.

Exercise 4: MPI Parallelization

The goal of this exercise is to develop a MPI-based parallel version of the Gaussian Elimination program elaborated in Exercise 2 in C/C++ or in Fortran.

First, take the sequential solution and benchmark it with appropriate values for the matrix dimension N.

Next, develop a MPI version of the program, benchmark this program as in Exercise 2 and report the corresponding results. You may choose the matrix dimension N and the number of processes P such that P divides N exactly.

The program starts by distributing the system A, b *row-wise* among the P processes *in a round-robin fashion* (i.e. process 0 receives rows $0, P, 2P, \ldots$, process

1, receives rows $1, P+1, 2P+1, \ldots$, and so on). For this purpose, process 0 constructs a correspondingly permuted version A', b' of the system to scatter the values among all processes (`MPI_SCATTER`).

For performing the triangulization, the program runs in N iterations, where in iteration i process $p = i\%P$ computes the pivot-value at $A(i, i)$ and broadcasts this value to all other processes (`MPI_BCAST`). Each process then uses this value to update all the rows of the system for which it is responsible.

For performing the back-substitution, the program runs in N iterations where in each iteration the process $p = N\%i$ that holds the newly computed result $x[i]$ broadcasts this value to all other processes (`MPI_BCAST`). Each process then uses this value to update all the rows of the system for which it is responsible.

Finally, process 0 gathers the result vector x (`MPI_GATHER`).