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

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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. In this solution, you may assume that the the number of processes P divides the matrix dimension 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.

  To simplify the program, you may assume that A is regular, i.e., $A(i,i)$ is different from 0 (if not, you may abort the computation).

- 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`).

Furthermore, perform a scalability analysis of the program (see Slide 13 of "Performance of Parallel Programs"), i.e.,

1. determine the isoefficiency function $f_E(P) = O(\ldots P \ldots)$, and
2. derive from this function the function $N = O(\ldots P \ldots)$ by which the matrix dimension must scale to keep the efficiency constant.

For this analysis, it suffices to use a simple communication model where sending a message of size n takes $O(n)$ time.

Finally, benchmark the program as follows:

1. Take the sequential solution and benchmark it with two appropriate values N_1, N_2 for the matrix dimension (one should run about a minute).
2. Benchmark the MPI version of the program for N_1 and N_2 and $P = 1,2,4,8,16$ processors.
3. Apply the result of the scalability analysis to scale one of N_1 or N_2 for $P = 1,2,4,8,16$ processors; benchmark for these values both the sequential and the parallel program.

Perform your benchmarks in three rounds and take the average values; report the absolute times, the speedup and the efficiency achieved, both in numerical and in graphical form (use logarithmic scales for the execution time and linear scales for speedup and efficiency).