Introduction to Parallel and Distributed Computing Exercise 4 (June 27)

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 1.
- the source (`.c`) 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++. In this solution, you may assume that 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). By this distribution, we ensure that the workload is evenly shared in the later phases of the triangulation (when the non-zero part of A becomes small).

  To distribute A, process 0 constructs a correspondingly permuted version A', b' of the system to scatter the values among all processes (by a single call of `MPI_Scatter`).

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

  To simplify the program, you may assume that $A(i,i)$ is different from 0 (if this should not be the case, 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 remove one unknown from all the rows of the system for which it is responsible.

- Finally, since process 0 has received all values that were broadcast during back-substitution, it can determine the result x.

Furthermore, perform a scalability analysis of the program, i.e.,

1. determine the workload function $w(N)$,
2. determine the overhead function $h(N,P)$,
3. determine the solution $N = \Omega(f(P))$ of the constraint

$$w(N) = \Omega(h(N,P))$$

   Thus the solution $f(P)$ determines how much for growing processor number P the matrix dimension N must at least grow (asymptotically) to keep the efficiency of the program constant (see Slide 13 of "Performance of Parallel Programs").

For determining $h(N,P)$, you just need to consider the communication overhead. Here it suffices to use a simple communication model where sending a message of size m takes time m; furthermore assume that broadcasting a message to p processors is implemented by sending p individual messages to each processor.

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 $P = 1,2,4,8,16,32$ processors.
3. Apply the result of the scalability analysis to scale one of N_1 or N_2 for $P = 1,2,4,8,16,32$ processors; benchmark for these values both the sequential and the parallel program. Do the benchmark results confirm the results of the scalability analysis (i.e., is the efficiency preserved at a high level)?

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.