exercises corresponding to the first ten lectures

The exercises below are ordered chronologically. A good understanding of the lecture notes (what was written on the blackboard in class) should suffice to find the answers.
The programs we have seen in class are available on the class web site.

The first element in the double numbering refers to lecture. Exercises with zero number are meant as an addition to your culture and will not be collected.

1.0) Visit www.top500.org.

1.1) How many processors whose clock speed runs at 3.0GHz does one need to build a supercomputer which achieves a theoretical peak performance of at least 4 Tera Flops? Justify your answer.

1.2) Suppose we have a program where 2% of the operations must be executed sequentially. According to Amdahl's law, what is the maximum speedup which can be achieved using 64 processors? Assuming we have an unlimited number of processors, what is the maximal speedup possible?

2.1) Benchmarking of a program running on a 64-processor machine shows that 2% of the operations are done sequentially, i.e.: that 2% of the time only one single processor is working while the rest is idle. Use Gustafson's law to compute the scaled speedup.

2.2) Derive a formula for the number of links in a hypercube with $p = 2^k$ processors, for some positive number k.

3.1) Consider a network of 16 nodes, organized in a 4-by-4 mesh with connecting loops to give it the topology of a torus (or doughnut). Can you find a mapping of the nodes which give it the topology of a hypercube? If so, use 4 bits to assign labels to the nodes. If not, explain why.

3.2) In class we derived an omega network for eight processors (see figure 1.14 on page 21 of the textbook). Give an example of a configuration of the switches which is blocking, i.e.: a case for which the switch configurations prevent some nodes from communicating with each other.

4.0) Run the programs we have seen in class on the cluster.

4.1) Adjust hello world so that you type in your name and every node salutes you, using the name you typed in.

5.1) Modify the program we have seen in class to sum 100 numbers to work with a variable number of processors. Instead of 100, add the first n positive natural numbers where $n = 10p$.

5.2) Rewrite the program to sum 100 numbers using MPI_Send and MPI_Recv instead of MPI_Scatter and MPI_Gather.

5.3) Rewrite the program to square p numbers using MPI_Scatter and MPI_Gather.

5.4) Modify the program for a parallel sum replacing the MPI_Gather by MPI_Reduce to sum up the results of the partial sums.

6.1) Rewrite the `cost_scatter.c` (available from the course web site) using MPI_Send and MPI_Recv. Execute for #processors $= 2, 3, \ldots, 10$, and make a table with wall times. Are the times as bad as for the scatter?

6.2) The second program with fan-out broadcast does not do the same as the scatter because it sends the same block to all processors. Modify the fan-out broadcast program `cost_fan.c` so that both programs achieve the same effect. Check the wall times for eight processors.

6.3) Run the program `cost_fan.c` for a varying number of processors, e.g.: #processors $= 2, 3, \ldots, 8$, and compute start up time for communication via interpolation.

9.1) Download the MATLAB/Octave script `mtbf.m` to solve a simple instance of the mean time between failures problem using the Monte Carlo method. Translate `mtbf.m` into C and create a parallel version of the program. Use MPI_Walltime to measure the speedup. Do you get more accurate answers as you increase the number of simulations?

10.0) Visit `http://sprng.cs.fsu.edu/`.

10.1) We say that a is a primitive element mod m if $\#\{\ x \mid x = a^i \bmod m,\ i = 0, 1, \ldots, m-1\ \} = m-1$. If the multiplier a in the congruential generator $x_{n+1} = a * x_n \bmod m$ is a primitive element mod m, for a prime modulus m, what is the period of the numbers x_n? Does the period depend on the seed x_0? Justify your answers, either by mathematical proof or/and simulations.

10.2) Write an algorithm to compute, for a given m, the sequence of numbers a of primitive elements mod m. Implement the algorithm in C and create a parallel version of the program.

The first homework collection will happen on Friday 10 February, at 1PM.

Homework (unlike projects) must be completed individually.

Feel free to come to my office or send email for questions about these assignments.