exercises corresponding to chapter 7

The exercises below are ordered chronologically, using a double numbering. The first element in this double numbering refers to the lecture. 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.

22.1) To test whether a number is prime, one could enumerate all possible divisors and check whether the remainder modulo each divisor is zero. Any even number is detected immediately, whereas it may take a long time to certify that a number is prime. Suppose we are given a sequence of n numbers and we can use p processors to verify whether every number is prime or not. Describe the algorithm to do this computation, using static load balancing. Analyze the communication and computation cost. Give an example of the worst possible input sequence for which there will be little or no speedup.

22.2) Use MPI to implement the algorithm of exercise 22.1. Experiment with sequence of random numbers and with especially chosen inputs to illustrate the worst case scenario with static load balancing.

23.1) Write the algorithm in pseudo code to do the primality testing of the previous exercise using dynamic load balancing. Make an analysis of the ratio between computation and communication cost and the potential speedup. Explain why dynamic load balancing will give a better speedup.

23.2) Use MPI to implement the dynamic load balancing algorithm for primality testing of exercise 23.1. Experiment with random numbers and especially chosen number sequences to illustrate the benefits of dynamic load balancing.

23.3) Suppose we are given one natural number and we want to test whether it is prime or not by enumerating all potential divisors and computing the remainder of the given number modulo this divisor. Describe how you could speed up this computation using p processors. Explain how superlinear speedup is possible, provided the termination problem is solved.

24.1) Consider again the primality testing of one natural number, for which we encountered the termination problem in exercise 23.3. Implement a solution for this termination problem, using MPI_Iprobe. Take as input a number which is a product of two primes to illustrate the superlinear speedup which may result from this solution.

24.2) In class we discussed the knapsack problem and a potential parallel implementation. Our setup involved a lower and an upper bound for the sum, fixed on input. Instead of these two bounds, assume we are given a target value for the sum. The goal of the program is to find the selection of items which meets best the given target value. Describe a parallel algorithm to solve this problem. Notice, that as soon as a better solution is found, every processor must notify all other processors.

24.3) Use MPI_Iprobe to implement the modification of the parallel knapsack algorithm of exercise 24.2. Give the output of a representative run, with enough intermediate output concerning the messages interchanged each time a better solution has been found.

25.1) The first parallel implementation of the root finder discussed in class did not really run in parallel. Correct the program shown in class and examine the speedup.

25.2) Use MPI_Iprobe to implement dynamic load balancing for the parallel root finder you corrected in exercise 25.1. Give empirical evidence of an improved speedup achieved by dynamic load balancing.

Homework collection is every Friday, at 1PM. Homework (unlike projects) must be completed individually. Feel free to contact me with questions about these assignments.