exercises corresponding to chapter 5

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.

15.1) Consider the evaluation of a polynomial f of degree n given by its coefficient vector $(a_0, a_1, a_2, \ldots, a_n)$ at some x, i.e.: compute $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$. Give the pseudo code of an algorithm to compute $f(x)$ by a p-stage pipeline. Illustrate the algorithm with a space-time diagram for $n = 32$ and $p = 4$. What is the speedup for this case?

15.2) Consider the evaluation of a polynomial f of degree n given by its coefficient vector $(a_0, a_1, a_2, \ldots, a_n)$ at some x, using Horner's method, e.g., for $n = 4$: $f(x) = (((a_4x + a_3)x + a_2)x + a_1)x + a_0$. Give the pseudo code of an algorithm to compute $f(x)$ by a p-stage pipeline. Illustrate the algorithm with a space-time diagram for $n = 32$ and $p = 4$. What is the speedup for this case?

15.3) Modify the program `pipe_ring.c` program as follows: the manager node reads a string of at most 20 characters (which could be your name) and passes it through the processors. Every node replaces one character (at a random position) by a random character, before passing the string on to the next processor. Give the program and its output.

16.1) In class we showed that the speedup to sum n numbers by a p-stage pipeline is $n/(n/p + p - 1)$ which approaches p when $n \gg p$. The code presented in class is limited for $n = p(p + 1)$. Extend this code so that it runs for n equal to any multiple of p. Demonstrate for summing the first n numbers, where $n = 10p$.

16.2) With insertion sort, a p-stage pipeline can sort p numbers in $2p$ pipeline cycles. Extend this pipelined sorting algorithm to sort two sequences of length p. How many pipeline cycles does it take to merge the two sorted sequences? Illustrate for $p = 4$.

16.3) Bubble sort is like insertion sort based on exchange operations. A pipeline is a beneficial infrastructure to implement the bubbling. Describe the pipelined bubble sort algorithm and illustrate for $p = 6$. Analyze the number of comparisons and give its speedup.

16.4) Implement the pipelined bubble sort algorithm you developed in exercise 16.3 using message passing. Give the program and the output for one typical run.

16.5) We skipped section 5.3.3 on the prime number generation using the Sieve of Eratosthenes. According to the analysis in the book, a pipelined implementation of this prime number generation algorithm can be quite effective. In addition, since weeding out all numbers divisible by a prime in one stage of the pipeline can be computationally very expensive, this parallel algorithm can be very good for a message passing implementation. Implement this parallel sieve of Eratosthenes algorithm and experiment with the speedup, comparing the execution times for 2, 4, and 8 processors.

17.1) Design a pipeline algorithm to compute the matrix vector product Ax, for A an n-by-n matrix and x a vector of length n, using an n-stage pipeline. Assume the matrix A is accessible to all processors in the pipeline. Describe the algorithm in pseudo code and illustrate for $n = 4 = p$. Analyze the speedup. Is it possible to perform Ax in $2n$ pipeline cycles? Describe the extension of the algorithm when $n = 2p$.

17.2) Write a program to simulate an n-stage pipeline to solve an n-by-n triangular system $Ly = b$ as we discussed in class. Initially, the manager scatters the lower triangular matrix L (with ones on its diagonal). As in the algorithm we discussed in class, the vector b is passed through the processors in the pipeline. Show the run on 10 processors for random data.

17.3) Lecture 17 ended by showing the speedup of the algorithm to solve an n-by-n triangular system by a p-stage pipeline, for $p = n/2$, which is better than for $p = n$. For $n = 2^p$, show that the speedup of a p-staged pipelined algorithm tends to p, for $n \gg p$. Your proof must contain the pseudo code of the p-staged algorithm, along with the operation count.

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