CSc 422/522 — Homework 1

due Tuesday, February 6

Problems 1 through 4 are worth five points each. Problems 5 through 7 are worth 10 points each. Graduate students are to solve all problems (50 points); undergraduates are to solve any combination of problems that adds up to 40 points.

You may discuss the meanings of questions with classmates, but the work you turn in must be yours alone. Explain clearly and succinctly what you are doing; don't just give an answer.

For the programming assignments, turn in well-commented listings of your programs. We will not be using electronic turnin for this assignment. Every program should have a descriptive header comment that includes your name. See also the other Presentation Points on the class Web page.

1. MPD book, Exercise 2.10.

2. MPD book, Exercise 2.15.

3. MPD book, Exercise 2.16.

4. MPD book, Exercise 2.19.

5. If `M` is an `n` $\times$ `n` matrix, the transpose of `M` is a matrix `T` such that `M[i,j] = T[j,i]` for all `i` and `j`. Write an iterative parallel program in MPD to compute the transpose of a matrix *in place.* Do *not* use an extra matrix. Use `PR` processes and assume that `n` is a multiple of `PR`. The values of `n` and `PR` should be command-line arguments. Initialize the matrix to any values you wish; I suggest using values that make it easy for you to check that your answer is correct.

6. Consider the parallel recursive quadrature program described in the text and given in MPD program `quad.co.mpd`.

(a) Modify the program so that it approximates the value of π using either of the functions in Exercise 1.6. The program should have the same three command-line arguments and it should write the same output.

(b) Modify your answer to (a) as described in Exercise 1.7, part (a). The threshhold value `T` should be a fourth command-line argument.

(c) Modify your answer to (b) so that it uses an array of `T` processes. Divide the interval from `a` to `b` into `T` parts, have each process approximate the area of one part, then add the results. Use the sequential recursive algorithm within each part.

Execute all three programs using the same arguments and report the results. Vary `epsilon` and `T` to see how they affect the accuracy of the result and the execution time.

7. Consider the following simple problem: Given two input files, output a *perfect shuffle* of the two files. A perfect shuffle is an interleaving that contains the first line from input file 1, then the first line from input file 2, then the second line from input file 1, then the second line from input file 2, and so on. (If the input files are the same, this has the effect of duplicating every line.) If one file is longer than the other, append the extra lines to the end of the output.

(a) Write a sequential MPD program to solve this problem.

(b) Modify your program to use the "`co` inside `while`" style as described in Section 2.2 of the textbook. The three independent activities are reading from `filename1`, reading from `filename2`, and writing to standard output. Use *double buffering* for each input file—i.e., read and write different buffers, then swap their roles.

If `shuffle` is the name of your executable file, then both programs should be invoked with two command-line arguments:

```
shuffle filename1 filename2
```

The programs should write to standard output.

Note: The MPD implementation by default lets a process run until it blocks, then it executes another process, and so on. This makes execution pretty deterministic on a single processor. You can force MPD to reschedule a process—and hence get a better simulation of true concurrency—in either of two ways. One is put calls of `nap(0)` in the bodies of loops. The second is to use the `-L` option with the MPD linker (`mpdl`). In particular, if `shuffle` is the name of your main (only) resource, first compile your program, then execute "`mpdl -L 1 shuffle`". The executable now resides in `a.out`.