CS 140 Sample Midterm 2 Questions -- March 2010

You may use your textbook and notes, but no other books or computers.

Problem 1 [30 points total]: Short answers.

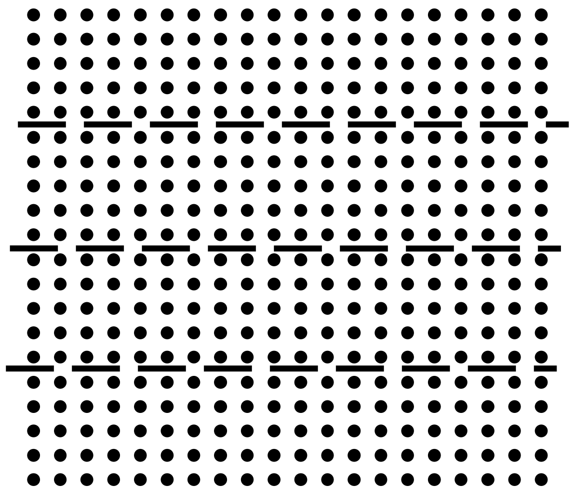
(1a) [10 points] Draw a graph that has two maximal independents sets of *different* sizes, and identify both maximal independent sets. (Your graph should have 5 vertices or less.)

(1b) [10 points] Give two reasons why many computations on large graphs perform poorly on conventional distributed-memory parallel computers.

(1c) [10 points] Why doesn't everyone use shared-memory parallel computers instead of distributed-memory parallel computers?

Problem 2 [35 points total]

This problem is about Jacobi iteration to solve the temperature problem on an **k**-by-**k** grid (with $\mathbf{n} = \mathbf{k}^2$ points), using a message-passing distributed memory machine. As usual, the effect of one iteration is to replace each value $\mathbf{x_i}$ with the average of its four neighboring values, using the boundary values if the point $\mathbf{x_i}$ is next to the edge of the region. For this problem, each processor has **k/p** rows of the grid. Assume **k** is divisible by **p**. Here's a picture for **k = 20** and **p = 4**:



To analyze the computation and communication time, use the following assumptions:

- Computation time only counts +, -, *, and / operations on floating point numbers, and each arithmetic operation takes one unit of time.

- Communication time counts both startup and data time, so the time to send or receive a message with **w** words is $\alpha + \beta * \mathbf{w}$. (The values of $\alpha = t_{startup}$ and $\beta = t_{data}$ depend on the machine architecture, but not on the algorithm or the problem.)

- Any number of messages can happen at the same time, provided each processor is only doing one send or one receive at a time.

- The total number of iterations is 1000.

Give your answers to the questions on the next page in terms of **k**, **p**, **α**, and **β**. You may ignore lower-order terms, but I want you to give the correct constant multipliers; in other words, an answer like **1000*(k/p * β + α)** is okay but an answer like **O(k/p)** is not.

The idea in all the questions is to compare the various complexity costs for 1000 iterations using one layer of ghost points, and 1000 iterations using two layers of ghost points.

(2a) [7 points] What is the computation time $\mathbf{t_{comp}}$ for 1000 iterations, as a function of **k** and **p**, if you use one layer of ghost points? (By this I mean the time needed by the processor with the most work, if they're different.)

(2b) [7 points] What is the computation time $\mathbf{t_{comp}}$ for 1000 iterations, as a function of **k** and **p**, if you use two layers of ghost points? (By this I mean the time needed by the processor with the most work, if they're different.)

(2c) [7 points] What is the communication time $\mathbf{t_{comm}}$ for 1000 iterations, as a function of **k**, **p**, $\mathbf{t_{startup}}$, and $\mathbf{t_{data}}$, if you use one layer of ghost points? (By this I mean the time needed by the processor that does the most communication, if they're different.)

(2d) [7 points] What is the communication time $\mathbf{t_{comm}}$ for 1000 iterations, as a function of **k**, **p**, $\mathbf{t_{startup}}$, and $\mathbf{t_{data}}$, if you use two layers of ghost points? (By this I mean the time needed by the processor that does the most communication, if they're different.)

(2e) [7 points] If you were writing code for this algorithm on a specific machine for which you knew the speed of floating-point arithmetic and the parameters **α** and **β**, how would you decide whether to use one or two levels of ghost points? Please answer both with a short English explanation and with a simple mathematical expression.

Problem 3 [15 points] You (still) want to use cilk++ to parallelize your **accumulate()** function from the first midterm, which computes the sum of elements in an array of size **n = 2^k**. The serial version of your code looks like this:

```
double accumulate (double * array, int n) {
        double sum = 0;
        for (int i = 0; i < n; i++) {
                sum += array[i];
        }
        return sum;
}
```

You will recall that your friend who told you to just replace the **for** loop with the **cilk_for** keyword was wrong, because that would cause a data race.

Now, another friend has suggested that you should parallelize the **accumulate()** function by rewriting it in a divide-and-conquer fashion, and shows you the following cilk++ code:

```
void recursive_accum (double * array, int start, int end, double * sum) {
        int size = end – start;
        if (size > THRESHOLD) {
                cilk_spawn recursive_accum (array, start, start+size/2, sum);
                recursive_accum (array, start+size/2, end, sum);
                cilk_sync;
        }
        else
                for (int i = start; i < end; i++)
                        *sum += array[i];
}

double accumulate (double * array, int n) {
        double sum = 0;
        recursive_accum (array, 0, n, &sum);
        return sum;
}
```

Do you think this code will run successfully in parallel? Why or why not? If you think the code is erroneous for some reason, propose a way to fix it.

Problem 4 [20 points] (Essay question.) This is an open-ended question. Your answer should be approximately two paragraphs long. Your score will be based both on your technical insight and on how well you communicate the point of your argument.

Describe why the question "where's the data?" is important in parallel programming. What are the trends in the development of computer architecture that are making this question either more important, or less important, as time goes on, and why? (If you want you can talk about trends in both directions.)