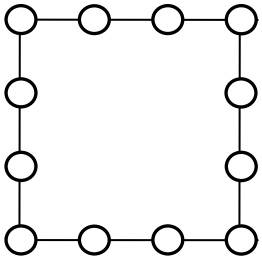
CS 140 Midterm 2 -- 3 March 2010

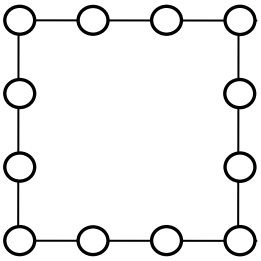
Name Perm#

Problem 1 [20 points total] This problem is about maximal independent sets in the graph shown at right, which has 12 vertices connected in a cycle with 12 edges. For each part, you are to first identify a particular maximal independent set (or MIS), and then label the vertices to show how one of the algorithms we studied in class could find that particular MIS.



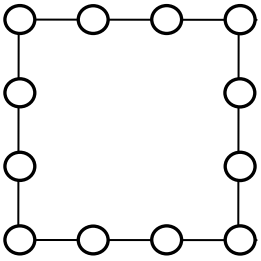
(1a) [5 points] On the copy of the graph below, identify a *largest possible* maximal independent set by coloring in the vertices of that set.

(1b) [5 points] On the same graph below, label the vertices with the integers 1, 2, …, 12 in such a way that the *sequential MIS algorithm* would find the same MIS you've colored in.



(1c) [5 points] On the copy of the graph below, identify a *smallest possible* maximal independent set by coloring in the vertices of that set.

(1d) [5 points] On the same graph below, label the vertices with 2-digit numbers (like 3.1, 2.7, 1.9) in such a way that, if the *randomized parallel MIS algorithm* happened to choose those random numbers, it would take just one round to find the same MIS you've colored in.



Name Perm#

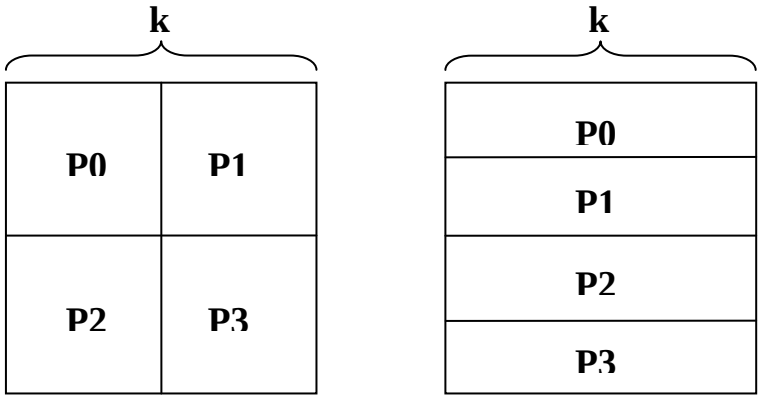
Problem 2 [20 points total] Recall that we have two ways to measure communication cost:

(1) *Communication volume* **v** counts the total number of data words that move from any processor to any other in the computation. In this measure, we add up the sizes of all the messages to get **v**.

(2) *Message-passing time* $\mathbf{t_{comm} = \alpha + \beta w}$ counts the time to send each message with **w** words. Here **α** is the startup time and **β** is the per-word time; the values of **α** and **β** depend on the machine architecture but not on the algorithm or the problem. In this measure, we assume that different messages can happen at the same time, provided each processor does only one send or one receive at a time.

Here we consider one iteration of the Jacobi stencil method for the temperature problem, on a square **k**-by-**k** grid of unknowns with $\mathbf{n = k^2}$ nodes. As usual, the effect of the iteration is to replace the value at each node with the average of its four neighboring values. We are only looking at one iteration, so ghost nodes are *not* an issue.

You have four processors. You are to compute the communication cost for one Jacobi iteration, in both measures (volume **v** and time $\mathbf{t_{comm}}$), for each of two possible data layouts, by blocks (shown at left below) or by rows (shown at right below). You may omit lower-order terms.



(2a) [5 points] What is the communication *volume* **v** for one Jacobi iteration on 4 processors using the *block* data layout? Your answer should be in terms of **k**.

(2b) [5 points] What is the communication *volume* **v** for one Jacobi iteration on 4 processors using the *row* data layout? Your answer should be in terms of **k**.

(2c) [5 points] What is the communication *time* $\mathbf{t_{comm}}$ for one Jacobi iteration on 4 processors using the *block* data layout? Your answer should be in terms of **k**, **α**, and **β.**

(2d) [5 points] What is the communication *time* $\mathbf{t_{comm}}$ for one Jacobi iteration on 4 processors using the *row* data layout? Your answer should be in terms of **k**, **α**, and **β.**

Name Perm#

Problem 3 [30 points total] The "sieve of Eratosthenes" is an algorithm that finds all the prime numbers up to a limit **n**. It uses an array of "marks" to mark off numbers from **2** to **n** that can't be prime because they're multiples of earlier primes; at the end, anything that didn't get marked off is prime. You are to use Cilk++ to parallelize each of the three steps in the following sequential code.

```
int count_primes (int n) {
        char marked[n+1];

        // Step One:  Set all the candidates to "unmarked".
        for (int i = 2; i <= n; i++)  marked[i] = 0;

        // Step Two:  Mark off everything that can't be prime.
        for (int number = 2; number <= sqrt(n); number++)
                if (marked[number] == 0) {
                        // Mark off all the multiples of "number"
                        for (int multiple = 2*number; multiple <= n; multiple += number)
                                marked[multiple] = 1;
                }

        // Step Three:  Count the numbers that didn't get marked off.
        int nprimes = 0;
        for (int i = 2; i <= n; i++)
                if (marked[i] == 0)
                        nprimes++;

        return nprimes;
}
```

You don't have to write detailed Cilk++ code unless you want to, but your description must be complete enough for us to be sure you know exactly what to do. You can continue your answers on the back if you want (though actually there is enough space below!).

(3a) [10 points] How can you parallelize **Step One** in Cilk++? (Hint: this is really easy.)

(3b) [10 points] How can you parallelize **Step Two** in Cilk++, to make the span roughly **sqrt(n)**? (Hint: this is simple *if* you do it the right way. Notice the bounds and the increments in the loops!)

(3c) [10 points] How can you parallelize **Step Three** in Cilk++?

Name Perm#

Problem 4 [30 points total] Short answers.

(4a) [10 points] Briefly describe two important differences between Cilk++ and MPI.

(4b) [10 points] Briefly describe one difficulty in getting good performance on graph algorithms on a distributed-memory parallel computer, and say how a massively multithreaded architecture (like the Cray XMT) tries to overcome that difficulty.

(4c) [10 points] Why doesn't everyone use shared-memory parallel computers instead of distributed-memory parallel computers?