CS 590V: Parallel Computing
Spring 1998: Final Exam, May 6, 1998
Name:
ID #:

1. (5 + 5 + 5 + 5 + 5 + 5 points, 20 mins)

   (a) It takes 1 ns to add two numbers. What is the maximum speedup you can get on a program (on any machine) for adding 256 numbers?

   (b) A computer has 400 MB/s of bandwidth to DRAM. What is the peak FLOPS rate of this processor for adding two matrices?

   (c) A parallel computer has a message startup of 10 microseconds. What is the fastest broadcast time of a 1 word message on 256 processors (ignore t_w).

(d) If all communication in a program is between neighboring processors, what is the difference in the runtime of a program on a 256 processor cut-through routed machine as opposed to a store-and-forward machine.

(e) An all-to-all personalized communication takes 100 microseconds on a 256 processor hypercube. Assuming t_s to be negligible, how long would it take on a mesh?

(f) An all-to-all personalized communication takes 100 microseconds on a 256 processor hypercube. Assuming t_s to be negligible, how long would it take on a linear array?

2. (10 points, 10 mins) What is the FLOPS to memory access ratio (in FLOPS/byte) for multiplying two 256×256 matrices (Assume 4 byte words)? If the processor has a peak rating of 200 MFLOPS and the memory has a bandwidth of 400 MB/s, what is the peak computation rate?

3. (15 points, 15 mins) The parallel time of an algorithm is given by

$$T_p = \frac{n^3}{p} t_c + n^2 t_w$$

and the serial time by

$$T_s = n^3 t_c$$

The total memory requirement is n^2.

- If the memory available per processor is constant and you are always operating at peak memory, what is the variation of efficiency with number of processors.

- What is the Isoefficiency of this algorithm?

4. (15 points, 10 mins) Consider 5 threads, each of which must "eat" its food. Each thread has a

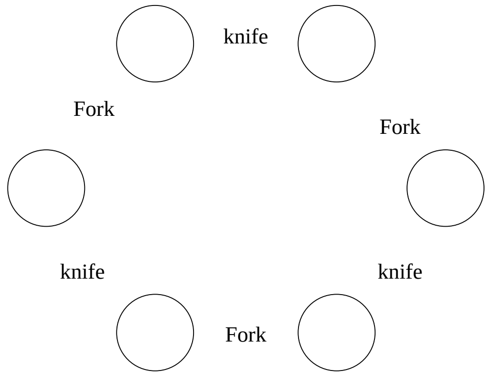

Figure 1: Illustration of forks, knives and diners.

fork and a knife on either side as shown in Figure 1. Each thread executes the following fragment of code:

```
lock(left)
lock(right)
eat()
unlock(right)
unlock(left)
```

What is wrong with this piece of code? How do you fix this?

5. (10 points, 10 mins) Consider the following:

```
        do 100, i = 1, 50
                sum = sum + a(i)
 100    continue
```

Using constructs do, doall (do each iteration of the loop in parallel) and critical (the next statement is critical) write a program that can run efficiently on a parallel computer (i.e. you can expect speedup from it).

6. (20 points, 20 mins) One way of detecting termination in dynamically load balanced programs is using a ring termination scheme. Each processor executes the following loop:

```
start_termination_detection();
repeat
{
        while (work_available())
        {
                work_a_bit ()
                if (pending_work_request())
                {
                        work_sent = 1;
                        send_some_work();
                }
        }
}
until(terminated() == TRUE);
```

The termination detection algorithm works as follows: processor 0 initiates a token of color white and sends it to the next processor (processor 1). Each intermediate processor in its terminated() routine checks to see if a token is present. If there is no token, terminated returns FALSE. If it is present, and if the processor has not sent any work to another processor, it sends the token unchanged to the next processor. If this token happened to be red, it also signals termination locally. If the processor had sent work to another processor, it changes the token to black and sends it to the next processor. On getting a white token back from processor $p-1$, processor 0 assumes that all work is done and sends a red token to processor 1; otherwise it sends a white token.

Is this a valid termination detection algorithm? Write the terminated() routine. Make sure you account for the differences in processor 0s tasks compared to everyone else. Use only the primitives send(TOKEN, processor), receive(TOKEN, processor), and pending_message(processor) (to see if there is a message pending from the processor).