

# CS 475 Fall 2010 Midterm I

Oct 6 2010

Name \_\_\_\_\_

*Please read these instructions completely before proceeding. Do not turn this page until you are asked to. This is a open book/notes exam. There are three questions, with multiple parts. Please answer all questions. Use the points (which all add up to 80) as a guideline to budget your time.*

Prob. No.	Max score	Your Score
1	20	
2	30	
3	30+10	
Total	80+10	

**Problem 1:**

[20 pts]

Parallelize each of the following code fragments using OpenMP pragmas, or explain why it cannot be parallelized.

**1a:**

[10 pts]

```
for (i=1; i<n; i++){
    for (j=20; j<m; j++){
        a[i][j] = foo(a[i-1][j], a[i][j-20]); // foo has no side-effects
    }
}
```

[10 pts]

```
for (i=0; i<n; i++){
    for (j=0; j<i; j++){
        x = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = x;
    }
}
```

[30 pts]

Quinn claims that in a certain code fragment, the inner  $j$  loop can be executed in parallel, so the following ( $a$  is an  $n \times n$  array,  $k$  is an integer, and  $tmp$  is a float variable, all appropriately initialized) is a legal parallelization.

```
for (i=0; i<m; i++){
    #pragma omp parallel for
    for (j=0; j<n; j++)
        a[i][j] = MIN(a[i][j], a[i][k]+tmp);
}
```

Explain why this is incorrect

[10 pts].

Give the correct parallelization of the inner loop.

[10 pts].

Despite the answer to the previous question, are there some conditions when Quinn is correct, or is it *never* legal to parallelize as given above? Justify. [10 pts].

**Problem 3. Collective Communications:** [30+10 pts]

We have a  $P$ -processor machine where the processors are arranged in an “octopus-accountant” grid, i.e., each processor can communicate (simultaneously, and bidirectionally) to all four of its neighbors (the boundary processors may have fewer neighbors). The time for a single communication *does not* depend on the message volume, it is just  $\lambda$ , and the arithmetic cost is  $\chi$ . Assume that the grid is square,  $P = p^2$  and that  $p$  is even. There is a data packet of size  $b$  already distributed on each processor. We want to perform the following computation: Each processor performs an operation `foo` on its packet, producing an answer packet of size  $b$ . Then these packets are combined using a special reduction operator, `bar` that takes two packets of size  $b$  and produces an answer packet of size  $b$ . However, we want the final result to be available at all the processors (i.e., an AllReduce operation). On any processor, the time to do either one of `foo` or `bar` is  $b^2\chi$ .

A simple algorithm to solve the problem would be (1) perform `foo`; (2) do the reduce operation to processor  $\langle 0, 0 \rangle$ ; (3) who then broadcasts to everyone. Describe the algorithm to do step 2. [5 pts].

Derive a formula for the execution times for each of the 3 steps. [2+4+4=10 pts].

To improve this algorithm, we choose a different processor  $\langle a, b \rangle$  as the destination of the reduction in the step 2. Explain how each step changes and re-derive the formula for the total execution time (it will be a function of  $a$  and  $b$ ). [10 pts].

Based on this, what is the fastest algorithm for the problem. [5pts]

d. **Extra Credit:** The machine is now modified to be a torus, i.e., the connections “wrap around” the edge of the grid—the west border is a neighbor of the east border and the north border is similarly adjacent to the south border. Describe how to further improve the algorithm, and derive a formula for its execution time. [10 pts].