

# CS 475/GRAD 510 Fall 2008 Midterm

Nov 20 2008, 2:00 pm

Name \_\_\_\_\_

*Please read these instructions completely before proceeding. Do not turn this page until you are asked to. This is a closed book exam, but you are allowed to bring in one page of notes. There are 4 questions of the weight shown. You need to answer all the problems. Although the total score is 100, the weight of the problem corresponds roughly to the time you should spend on it (problem 1 should actually take you only 10 minutes or less).*

Prob. No.	Max score	Your Score
1	20	
2	30	
3	25	
4	25	
Total	100	

**Problem 1:** [20 pts]

For each of the following code fragments, use OpenMP pragmas to make the loop parallel, or explain why it is not suitable for parallel execution. If it is not parallelizable, suggest, if possible how the program could be rewritten for parallelization.

**1a:** [10 pts]

```
for (i=1; i<n; i++)
  for (j=0; j<m; j++) {
    X[i] = bar(X[i], A[i-1], A[i]); /* bar has no side-effects */
  }
```

**1b:** [10 pts]

```
for (i=1; i<n; i++)
  for (j=0; j<m; j++) {
    X[i] = bar(X[i], X[i-1]); /* bar has no side-effects */
  }
```

**Problem 2: Knapsack**

[30 pts]

The *unbounded knapsack problem* is a variant of the knapsack problem in which each item can be selected any number of times. A similar recurrence to the 0/1 case can be set up for this problem too that defines  $F(i, k)$  the best profit that can be achieved with a knapsack of capacity  $k$ , while choosing out of the first  $i$  objects.

$$F(i, k) = \begin{cases} k = 0 & : 0 \\ i = 0 & : 0 \\ 0 < k < w_i; i > 0 & : F(i-1, k) \\ k \geq w_i; i > 0 & : \max \begin{pmatrix} F(i-1, k) \\ p_i + F(i, k - w_i) \end{pmatrix} \end{cases} \quad (1)$$

In the last line, the value added to  $p_i$  comes from “the left by  $w_i$ ,” as before, but now it comes from the *same* row rather than the previous row. This is the only difference.

Write a simple sequential code fragment (pseudo code is all we need—we just need to see the logic) that computes this table in a row wise, memory efficient manner. Assume that `curr[0...C]` and `prev[0...C]` have been declared, as have `weights[0...N]` and `profits[0...N]`, and appropriately initialized. [15 pts]

Parallelize this code in OpenMP.

[15 pts]

**Problem 3:**

[25 pts]

Consider the following fragment similar to the 1D Gauss Seidel code that was provided to you for HW4 (assume that `tid`, `nthreads` and all other variables have been appropriately initialized and have their standard meanings). The code updates the values based only on the west and south iterations, and so there is only one barrier.

```
#pragma omp parallel ...
// Standard initialization code, omitted for brevity
for (prolog = 0; prolog < tid; prolog++){           // Prolog
#pragma omp barrier
}
for (t=0; t<MAX_ITERATION; t++){
    for (i=first; i<=last; i++) A[i] = (A[i-1] + A[i])/3.0;
#pragma omp barrier                               // Note this comment
}
for (epilog = tid; epilog < nthreads-1; epilog++){ // Epilog
#pragma omp barrier
}
```

Before checking this program in, we inadvertently deleted the barrier within the innermost loop (the line with the “Note this comment” was removed). The modified program will (circle one) [10 pts]

1. hang.
2. terminate but produce different results from the original.
3. terminate and produce exactly the same results.

If you answered (1) to the previous question, explain precisely why [10 pts] and a different way to fix it (other than reinserting the missing line) [5 pts]. [15 pts]

If you chose (2) or (3), describe how the execution time will be affected, regardless of the values produced. Will it run (circle one): (i) much slower; (ii) slightly slower; (iii) exactly the same (iv) slightly faster; or (v) significantly faster than the sequential code [5 pts]. Justify your answer [10 pts]. [15 pts]

**Problem 4:**

[25 pts]

Consider the following program that computes an output array B from an input array A.

```
B[0,0] = A[0,0]
for (i=1; i<n; i++){ // Initialization at the boundaries
    B[i,0] = B[i-1,0] + A[i,0];
    B[0,i] = B[0,i-1] + A[0,i];
}
for (i=1; i<n; i++){ // Main loop
    for (j=1; j<n; j++)
        B[i,j] = B[i,j-1] + B[i-1,j] - B[i-1,j-1] + A[i,j];
}
```

Parallelize this program in OpenMP.

[10 pts]

[Challenge]

[15 pts]

Parallelize the program as efficiently as you can. [Hint: it can be done with almost *no communication/synchronization*. You need to think a bit outside the box.]