Homework 5

Due 11:59PM Dec. 6

This assignment is extra credit and optional. Students who have less than desired grades so far due to missed submission may benefit from completing it.

Exercise question 6.6 from Zybook:

**6.3** Many computer applications involve searching through a set of data and sorting the data. A number of efficient searching and sorting algorithms have been devised in order to reduce the runtime of these tedious tasks. In this problem we will consider how best to parallelize these tasks.

**6.3.1** [15] <COD §6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value *X* in a sorted N-element array A and returns the index of matched entry: 

BinarySearch(A[0..N−1], X) {

low = 0

high = N - 1

while (low <= high) {

mid = (low + high) / 2

if (A[mid] > X)

high = mid - 1

else if (A[mid] < X)

low = mid + 1

else

return mid // found

}

return −1 // not found

}

Assume that you have Y cores on a multi-core processor to run BinarySearch. Assuming that Y is much smaller than N, express the speedup factor you might expect to obtain for values of Y and N. Plot these on a graph.

**6.3.2** [15] <COD §6.2> Next, assume that Y is equal to N. How would this affect your conclusions in your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

Exercise question 6.6 from Zybook:

**6.6** Matrix multiplication plays an important role in a number of applications. Two matrices can only be multiplied if the number of columns of the first matrix is equal to the number of rows in the second.

Let's assume we have an *m × n* matrix *A* and we want to multiply it by an *n × p* matrix *B*. We can express their product as an *m × p* matrix denoted by *AB* (or *A · B*). If we assign *C = AB*, and *Ci, j* denotes the entry in *C* at position (*i, j*), then for each element *i* and *j* with 1 ≤ *i* ≤ *m* and 1 ≤ *j* ≤ *p* ci,j = ∑k=1n ai,k X bk,j. Now we want to see if we can parallelize the computation of C. Assume that matrices are laid out in memory sequentially as follows: a1,1, a2,1, a3,1, a4,1, …, etc.

**6.6.1** [25] <COD §6.5> Assume that we are going to compute *C* on both a single core shared memory machine and a 4-core shared-memory machine. Compute the speedup we would expect to obtain on the 4-core machine, ignoring any memory issues.

**6.6.2** [15] <COD §6.5> Repeat Exercise 6.6.1, assuming that updates to *C* incur a cache miss due to false sharing when consecutive elements are in a row (i.e., index *i*) are updated.

Exercise question 6.7 from Zybook:

**6.7** Consider the following portions of two different programs running at the same time on four processors in a *symmetric multicore processor* (SMP). Assume that before this code is run, both x and y are 0.   
  
Core 1: x = 2;   
Core 2: y = 2;   
Core 3: w = x + y + 1;    
Core 4: z = x + y;

**6.7.1** [15] <COD §6.5> What are all the possible resulting values of w, x, y, and z? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

**6.7.2** [15] <COD §6.5> How could you make the execution more deterministic so that only one set of values is possible?