

## CS-4234/CS-5234 Homework #4

### Problem 1

Reconsider the matrix multiplication problem. This time the computation should be performed using OpenMP. The assumption is that we work on a shared memory platform.

Implement the parallel matrix multiplication algorithm  $C = A \times B$  for  $n \times n$  matrices of real numbers. Define one task to be the multiplication of one row in  $A$  with one column in  $B$ , to obtain one entry in the result  $C$ . There are  $n^2$  tasks which need to be distributed to  $p$  threads.

You can take  $n$  to be a multiple of 960 (which divides evenly to  $p = 2, 3, 4, 5, 6, 7, 8$  threads). You can also use known, non-constant entries for  $A$  and  $B$  in order to check the correctness of the result.

Use the parallel loop in OpenMP. Test all six loop orderings ( $ijk$ , meaning that  $i$  is the outermost loop and  $k$  the innermost;  $jki$ , meaning that  $j$  is the outermost, etc)

Discuss how the parallelization strategy is different than the strategies we applied for distributed memory programming.

**Graduate students:** write a second version that uses nested parallelism. Specifically, you will need to parallelize two nested loop levels. Comment on how threads are generated and which threads execute which part of the nested loops.

### Problem 2

The serial sample sort algorithm sorts  $n$  integers ( $X = [x_1, \dots, x_n]$ ) with the following steps:

1. A sample of  $s$  ( $s \ll n$ ) elements  $[x_{i_1}, \dots, x_{i_s}]$  is selected from the  $n$ -element vector  $X$ . We sort the  $s$  elements and then choose  $p - 1$  “splitter values”  $a_1 < \dots < a_{p-1}$  that divide the sample vector  $[x_{i_1}, \dots, x_{i_s}]$  into  $p$  nearly equal sub-vectors.
2. The splitter values define  $m$  “buckets” as follows:  $B_1 = \{x | x < a_1\}$ ,  $B_i = \{x | a_{i-1} \leq x < a_{i+1}\}$ ,  $2 \leq i \leq p - 1$ ,  $B_p = \{x | a_{p-1} < x\}$ . During one pass through the vector  $X$  each element  $x_j$  is placed in the appropriate bucket according to its value (i.e., is copied into an array  $B_j$ ).
3. Each bucket is sorted independently.
4. The entire sorted vector is obtained by concatenating the contents of the sorted buckets  $[B_1 \dots B_p]$ .

Write a parallel sample sort algorithm for  $p$  processors. The algorithm goes as follows:

1. Each thread  $P_i$  is responsible for a sub-vector  $X_i$  with  $n/p$  elements.
2. Each thread selects  $k$  sample elements from its sub-vector and sends them to  $P_0$ . The process  $P_0$  now has a sample of  $s = kp$  elements which it uses to select the splitter values  $a_1 < \dots < a_{p-1}$ .

3. Each thread  $P_i$  goes through the local sub-vector  $X_i$  and splits it into  $p$  subsequences, such that elements in subsequence  $j$  belong to bucket  $j$  ( $X_{i,j} \in B_j$ ).
4. Each thread  $P_i$  communicates the sub-sequence  $X_{i,j}$  to thread  $P_j$ .
5. Thread  $P_j$  has all the elements of the original vector  $X$  that belong to bucket  $B_j$ . Each thread sorts locally the received sub-sequences (using e.g., quicksort).
6. The sorted subsequences from all threads are assembled into the sorted vector  $X$ .

Implement the algorithm in OpenMP. Run the parallel sorting algorithm for different data sizes (e.g.,  $n = 2^{10}$  and up) and on different number of processors (e.g.,  $p=1,2, \dots$  to 16). Time each of the runs. Plot the experimental speedup versus the number of processors. Discuss and draw conclusions.

### **Problem 3 (graduate students)**

Read section 10.4.2. in the textbook (pages 440-445).

Solve problem 10.6 (page 465).

Discuss the advantages and disadvantages of the 1-D data partitioning proposed in problem 10.8 (page 466).

Implement the parallel Floyd's algorithm using OpenMP (you choose which version of the algorithm to implement). Make several runs with different starting initial configurations and different numbers of threads. Report the results, including the speedup, and explain the behavior.