Amirkabir University of Technology

(Tehran Polytechnic)

Department of Computer Engineering,
Multicore Programming (Spring 2024),
Assignment 4

Question 1: The following function shuffles the elements of the given array. Based on this implementation, answer the following questions

```
void shuffle (int* array, int len){
    omp_lock_t locks[len];
    #pragma omp parallel
    {
        unsigned int a, b;
        unsigned int seed = omp_get_thread_num();

        //Create Locks
        #pragma omp for
        for (int i = 0; i < len; i++)
            omp_init_lock(&locks[i]);

        // Shuffle
        for (int i = 0; i < 1000; i++) {
            a = rand_r(&seed) % len;
            do {
                b = rand_r(&seed) % len;
            } while(a == b);

            omp_set_lock(&locks[a]);
            omp_set_lock(&locks[b]);
            swap(&array[a], &array[b]);
            omp_unset_lock(&locks[a]);
            omp_unset_lock(&locks[b]);
        }

        // Delete Locks
        #pragma omp for
        for (int i = 0; i < len; i++)
            omp_destroy_lock(&locks[i]);
    }
}
```

a) What is the main reason for the deadlock in multi-core programming?
b) Explain why is there a chance of a deadlock happening in this function.
c) What is the maximum number of threads that can be used in this function for it to have a less than 10% chance of deadlock? (for simplicity assume that all threads are synchronized in the shuffle loop as in they have the same loop index at the same time)
d) Rewrite this code and remove any potential for a deadlock.

Question 2: A longest common subsequence (LCS) is the longest subsequence common to all sequences in a set of sequences (often just two sequences). For example, the two strings BCDAACD and ACDBAC have the largest common subsequence CDAC. Using dynamic programming, LCS is solvable in polynomial time by filling a matrix row by row. You are provided with a serial implementation of LCS that works on two 32Kb strings. Write a parallel implementation of LCS using OpenMP tasks. For cache friendlier memory access patterns, break the matrix into 128x128 chunks, each chunk becoming a task. Use the least amount of synchronization between tasks and try to have minimal cache misses.

Question 3: Timsort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data. It was invented by Tim Peters in 2002 for use in the Python programming language. The insertion sort, which is inefficient on large data, but very efficient on small data (say, five to ten elements), is used as the final step, after applying another algorithm, which is in this case, merge sort. Merge sort is asymptotically optimal on large data, but the overhead becomes significant if applying it to small data, hence the use of a different algorithm at the end of the recursion. You are supposed to implement a parallelized Timsort algorithm using C/C++ and OpenMP.

1. Implement a parallel version of Timsort using OpenMP, assuming that the number of final subarrays (arrays that are to be sorted using insertion sort) is divisible by the number of threads. Resulted code should have very high utilization (close to 100%) during all major operations.
2. Complete the following table (using the maximum number of threads you have available). What is the optimal transition point between merge sort and insertion sort?

Insertion sort transition point	Array Size (4 Bytes per element)			
	4KB	256KB	16MB	1GB
4				
8				
16				
32				