Project 2: Multithread LCS Milestone 1 Report

November 26, 2017

Sabbir Ahmed & Zafar Mamarakhimov

1 Description

The longest common subsequence (LCS) problem is to be implemented for the assignment. The objective of this project is to design, analyze, and implement a multithreaded version of the LCS length algorithm. A memoized or bottom-up approach. In either case, the goal is to design an algorithm that makes efficient use of multiple processors.

Analysis is required once the LCS length algorithm is designed. The work $T_1(m, n)$ and the span $T_\infty(m, n)$ are to be computed, where m and n are the lengths of the input sequences X and Y, respectively. Finally, the parallelism is to be computed along with an estimate of a range of parameters (m, n, and P) for which linear or near-linear speed-up may be expected.

The last part of the project is to implement the LCS length algorithm in C or C++ using OpenMP on UMBC's maya cluster and measure its performance empirically using 1, 2, 4, and 8 processors (optionally, it may be tested on 16 processors; most maya nodes have eight cores, but some have 16). Test data and LCS lengths of various sizes are to be generated to demonstrate the performance characteristics of the algorithm. The algorithm to recover the LCS must also be implemented, but this need not be multithreaded.

2 Serial Algorithm

Milestone 1 required implementation of the serial LCS algorithm and its analysis. Algorithm 2.1 provides the pseudocode for computing the length of the LCS. The serial algorithm utilizes the memoization method to compute the LCS.

Algorithm 2.1: Serial Longest Common Subsequence Length

```
1  function serial_lcs(X, Y, m, n)
2      // allocate (m + 1) × (n + 1) LCS matrix
3      lcs = new matrix[1, 2, .., m + 1][1, 2, .., n + 1]
4      for i = 0 to m
5          for j = 0 to n
6              if (i == 0 or j == 0) // upper-leftmost cell
7                  lcs_{i,j} = 0
8              else if (X_{i-1} == Y_{j-1})
9                  // incremented left diagonal cell value
10                 lcs_{i,j} = lcs_{i-1,j-1} + 1
11             else
12                 // maximum between previous row and previous column
13                 lcs_{i,j} = max(lcs_{i-1,j}, lcs_{i,j-1})
14     return lcs_{m,n} // the last value of the matrix is the length
```

2.1 Time Complexity

The running time of the serial algorithm provides the work, T_1, of the algorithm. Simple analysis of the algorithm provided in the snippet Algorithm 2.1 suggests a non- linear running time from the nested loop. The inner loop (line 5) iterates n times with several constant conditional checks, while the outer loop (line 4) iterates m times. The runtime is therefore:

$$T_1(m, n) = O(m \times n)$$

The work for the algorithm, $O(m \times n)$, is quadratic if $m = n$.

2.2 Printing The LCS

After computing the length, the matrix may be used to print the LCS itself. Printing the subsequence is done in a serial implementation since its analysis is not emphasized. Algorithm 2.2 provides the pseudocode used to print the LCS.

Algorithm 2.2: Serial Longest Common Subsequence Printing

```
function serial_lcs_print(X, Y, m, n, lcs)
    lcsstr = new string
    cursor = lcs_{m,n} // cursor of the matrix
    i = m, j = n // init from the bottom-rightmost cell
    while (i > 0 and j > 0)
        // if current character in X[] and Y[] are same
        if (X_{i-1} == Y_{j-1})
            lcsstr_{cursor-1} = X_{i-1} // result gets current character
            i--, j--, cursor-- // decrement i, j and cursor
        // find the larger of two and go to that direction
        else if (lcs_{i-1,j} > lcs_{i,j-1})
            i--
        else
            j--
    print lcsstr
```