Project 2: Multithread LCS Milestone 1 Report

December 5, 2017

Sabbir Ahmed

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 Initialization

The matrix is initialized to store the LCS before computation. Since the implementation utilized a two-dimensional array allocated on the heap to represent the matrix, initialization was possible in $\Theta(m)$ time. This method is faster than iterating through all the cells of the matrix to assign each to a placeholder. Algorithm 2.1 provides the implementation used to initialize the matrix.

Reading the sequences from the one-line input files, truncating them if necessary and storing them into buffers all take constant time operations.

Algorithm 2.1: Initialization of the Longest Common Subsequence Matrix

```
1  lcs_matrix = pointer array(m + 1);
2  for i = 0 to m + 1
3      lcs_i = pointer array(n + 1);
```

3 Serial Algorithm

Milestone 1 required implementation of the serial LCS length algorithm and its analysis. Algorithm 3.2 provides the pseudocode for computing the length of the LCS. The serial algorithm utilizes the memoization method to compute the LCS.

Algorithm 3.1: Serial Longest Common Subsequence Length

```
1  function LCS-LENGTH(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 upper-leftmost cell, content is 0
7              if (i == 0 or j == 0)
8                  lcs_{i,j} = 0
9              // if equal, content is the left diagonal value incremented by 1
10             else if (X_{i-1} == Y_{j-1})
11                 lcs_{i,j} = lcs_{i-1,j-1} + 1
12             // maximum between previous row and previous column values
13             else
14                 lcs_{i,j} = max(lcs_{i-1,j}, lcs_{i,j-1})
15     return lcs_{m,n} // the last value of the matrix is the length
```

3.1 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 3.2 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 $m = n$.

3.2 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 3.3 provides the pseudocode used to print the LCS.

Algorithm 3.2: Serial Longest Common Subsequence Printing

```
1  function SERIAL-LCS-PRINT(X, Y, m, n, lcs)
2      lcsstr = new string
3      cursor = lcs_{m,n} // cursor of the matrix
4      i = m, j = n // init from the bottom-rightmost cell
5      while (i > 0 and j > 0)
6          // if current character in X[] and Y[] are same
7          if (X_{i-1} == Y_{j-1})
8              lcsstr_{cursor-1} = X_{i-1} // result gets current character
9              i--, j--, cursor-- // decrement i, j and cursor
10         // find the larger of two and go to that direction
11         else if (lcs_{i-1,j} > lcs_{i,j-1})
12             i--
13         else
14             j--
15     print lcsstr
```

4 Parallel Algorithm

Milestone 2 required implementation of the parallel LCS length algorithm. Algorithm 4.4 provides the pseudocode for computing the length of the LCS. Unlike the serial implementation, the parallel version of the algorithm requires a variation

Algorithm 4.1: Parallel Longest Common Subsequence Length

```
1  function P-LCS-LENGTH(X, Y, m, n)
2      for i = 1 to n
3          parallel for j = 1 to i
4              if Y_{i-j} == X_{j-1}
5                  lcs_{i-j+1,j} = lcs_{i-j,j-1} + 1
6              else if lcs_{i-j,j} ≥ lcs_{i-j+1,j-1}
7                  lcs_{i-j+1,j} = lcs_{i-j,j}
8              else
9                  lcs_{i-j+1,j} = lcs_{i-j+1,j-1}
10     weight = 0
11     for i = 2 to m
12         if weight < (m - n)
13             weight++
14         parallel for j = i to (n + weight)
15             if Y_{n-j+i-1} == X_{j-1}
16                 lcs_{n-j+i,j} = lcs_{n-j+i-1,j-1} + 1
17             else if lcs_{n-j+i-1,j} ≥ lcs_{n-j+i,j-1}
18                 lcs_{n-j+i,j} = lcs_{n-j+i-1,j}
19             else
20                 lcs_{n-j+i,j} = lcs_{n-j+i,j-1}
```

5 Testing and Debugging

5.1 Milestone 1

The source code was initially developed in C++, but later translated to C. The Intel C++ Compiler, `icpc`, did not appear to properly compile the source code. The executable built without any warnings or errors, but the algorithm during its execution appeared to skip steps. The current implementation in C compiles with both `gcc` and the Intel C Compiler, `icc`.