CS 677 High-performance Computing
Programming Assignment #2

Parallelized Local Sequence Alignment

Overview

The basic idea is to write a multi-threaded program that implements the Smith-Waterman dynamic programming algorithm for performing local sequence alignment of two DNA nucleotide sequences.

Background

Sequence alignment is sometimes referred to as the "longest common subsequence" problem. Fundamentally, it involves approximate pattern matching to find similar sequences in two (or more) strings of characters. Finding a similarity between a new, unknown sequence and a previously known sequence can provide biologically significant information about structural, functional or evolutionary relationships.

Specifications

The Smith-Waterman algorithm is a form of dynamic programming (see Sec. 12.1, 12.3 in our text), a technique in which the solution to a problem is recursively expressed as a function of similar subproblems at the preceding level. In this case, the subproblems are matching substrings.

The first phase of the SW algorithm calculates the *similarity* matrix, and the second phase retrieves the local alignments. This assignment is concerned primarily with parallelizing the first phase of the algorithm (the most time-consuming component).

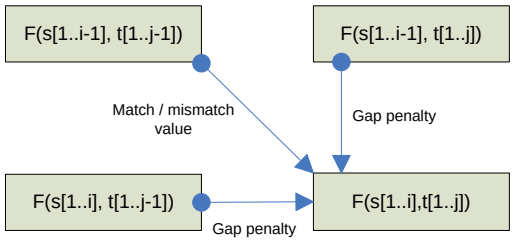
Input consists of sequences *s* and *t*, with |s| = m and |t| = n.
String prefix similarity measures (i.e. array entries) are calculated using the equation:

$$F(s[1..i], t[1..j]) = \max \begin{cases} F(s[1..i], t[1..j-1]) - 2, \\ F(s[1..i-1], t[1..j-1]) + (\text{if } s[i] = t[j] \text{ then } 1 \text{ else } -1), \\ F(s[1..i-1], t[1..j]) - 2, \\ 0 \end{cases}$$

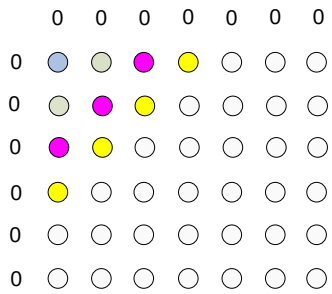
Note that because of the 0, negative scores are never generated. Note also that because each array entry must "look back" one row and/or one column, the first row and first column of the similarity matrix need to be initialized with zeros.

The basic algorithm: at each step, calculate the optimal alignment for a prefix that ends with a gap in s or a gap in t (and hence incurs a gap penalty), or has a match/mismatch. A gap results in a score of (previous value – 2); a match is scored as (previous value + 1); and a mismatch is (previous value – 1). Highest score wins.

The figure below illustrates the dynamic programming technique applied to the similarity scoring equation. The recursive nature of the equation is expressed in such a way that the value of a matrix element depends on the values of its neighbors to the north, west, and northwest.



This data dependency pattern suggests several different ways of allowing a computation to proceed. As seen below, the "wavefront" indicates those elements which are *available* for computation at each timestep.



At the first time step, only element (1,1) can be computed (the blue sphere). Once that result is available, elements (1,2) and (2,1) can be computed (the grey spheres), as the data they depend on is now available. Then elements (1,3), (2,2), and (3,1) can be computed (the pink spheres), and so on.

Devise a parallel decomposition and mapping that uses multiple threads to perform simultaneous computations on the similarity matrix.

Deliverables:

Create a multi-threaded program that calculates the similarity matrix between a known sequence *s* and an "unknown" sequence *t*. Note: a nucleotide value of '**?**' in a sequence means unknown; it should be treated as a wildcard (i.e. matches anything).

- You may use any programming language / library that supports true multi-threading (i.e. not Python).
- You must conduct a performance analysis of your code executing on a multiprocessor system, with particular focus on Speedup:

$$S = \text{Time}_{\text{Sequential}} / \text{Time}_{\text{Parallel}}$$

- Submit a hard-copy of your design document with analysis, source-code, and sample output. Be prepared to present your solution in class.