Homework 3: Parallel Scoring in Pairwise Sequence Alignment

Instructor: Vivek Sarkar

Assigned February 3, 2012, due by 11:55pm on Wednesday, February 22, 2012
(1 programming problem totaling 100 points)

All homeworks should be submitted in a directory named "hw_3" using the turn-in script. It is important that you start early on this homework to meet the deadline. Sections 1–3 contain background information. Your assignment is in Section 4.

Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone elses work as your own. If you use any material from external sources, you must provide proper attribution.

1 Pairwise Sequence Alignment

In this homework, we will focus on the *pairwise sequence alignment* problem in evolutionary and molecular biology, and how parallelism can help in solving this problem. (This homework is adapted from Homework 7 from the Spring 2011 offering of COMP 182 by Prof. Luay Nakhleh.)

Let X and Y be two sequences over alphabet Σ (for DNA sequences, $\Sigma = \{A, C, T, G\}$). An *alignment* of X and Y is two sequences X' and Y' over the alphabet $\Sigma \cup \{-\}$, where X' is formed from X by adding only dashes to it, and Y' is formed from Y by adding only dashes to it, such that

1 $|X'| = |Y'|$,

2 there does not exist an i such that $X'[i] = Y'[i] = -$, and

3 X is a subsequence of X', and Y is a subsequence of Y'.

This alignment is also referred to as *global pairwise alignment* (as opposed to *local pairwise alignment*, which is used to align selected regions of sequences X and Y).

Sequence alignment helps biologists make inferences about the evolutionary relationship between two DNA sequences. Aligning two sequences amounts to "reverse engineering" the evolutionary process that acted upon the two sequences and modified them so that their characters and their lengths differ. As an example, one possible alignment of the two sequences $X = ACCT$ and $Y = TACGGT$ is as follows:

X'	=	-	A	C	-	C	T
Y'	=	T	A	C	G	G	T

As you may imagine, there may be multiple alignments for the same pair of sequences. For example, a trivial alternate alignment for X and Y is as follows:

X''	= A	C	C	T	-	-	-	-	-	-
Y''	= -	-	-	-	T	A	C	G	G	T

2 Scoring in Pairwise Sequence Alignment: Optimality Criterion

As discussed above, a number of alignments exist for a given pair of sequences; therefore, we define a *scoring scheme* that gives higher scores to "better" alignments. Once the scoring scheme is defined, we seek an

alignment with the highest score (among all feasible alignments). For DNA, a scoring scheme is given by a 5×5 matrix M, where for $p, q \in \{A, C, T, G\}$, $M_{p,q}$ specifies the score for aligning p in sequence X' with q in sequence Y', $M_{p,-}$ denotes the penalty for aligning p in sequence X' with a dash in sequence Y', and $M_{-,q}$ denotes the penalty for aligning q in sequence Y' with a dash in sequence X'. Assuming $|X'| = |Y'| = k$, the score of the alignment is

$$\sum_{i=1}^{k} M_{X'[i],Y'[i]}. \tag{1}$$

For this assignment, we will assume the following scoring scheme: $M_{p,p} = 5$, $M_{p,q} = 2$ (for $p \neq q$), $M_{p,-} = -2$ and $M_{-,q} = -4$.

For this scoring scheme. the score of the (X', Y') alignment in Section 1 is

$$M_{-,T} + M_{A,A} + M_{C,C} + M_{-,G} + M_{C,G} + M_{T,T} = (-4) + 5 + 5 + (-4) + 2 + 5 = 9$$

and the score of the (X'', Y'') alignment is $4 \times M_{p,-} + 6 \times M_{-,q} = -32$.

3 Sequential Algorithm to compute the Optimal Scoring for Pairwise Sequence Alignment

In this problem, we introduce a sequential dynamic programming algorithm (called the Smith-Waterman algorithm) to compute the Optimal Scoring for Pairwise Sequence Alignment. For two sequences X and Y of lengths m and n, respectively, denote by $S[i, j]$, $0 \leq i \leq m$ and $0 \leq j \leq n$, the score of the best alignment of the first i characters of X with the first j characters of Y. The boundary values are, $S[i, 0] = i * M_{p,-}$ and $S[0, j] = j * M_{-,p}$. It has been shown that this optimal scoring can be defined as follows $\forall i, j \geq 1$:

$$S[i, j] = \max \begin{cases} S[i-1, j-1] + M_{X[i],Y[j]} \\ S[i-1, j] + M_{X[i],-} \\ S[i, j-1] + M_{-,Y[j]} \end{cases}. \tag{2}$$

The above definition directly leads to a sequential dynamic programming algorithm that can be implemented as shown in Listing 1. Assume that the input sequences are represented as Java strings, and the scoring matrix, S, is represented as a 2-dimensional array of size (*X.length()+1*) $\times$ (*Y.length()+1*). After the algorithm terminates, the final score is available in *S[X.length()][Y.length()]*.

The dependence structure of the iterations in Listing 1 is shown in Figure 1. The cells in the figure correspond to $S[i, j]$ values, and the arrows show the dependences among the $S[i, j]$ computations.

```
1  for (point[i,j] : [1:X.length(),1:Y.length()] ){
2     char xChar = X.charAt(i-1);
3     char YChar = Y.charAt(j-1);
4     int diagScore = S[i-1][j-1] + M[charMap(xChar)][charMap(YChar)];
5     int topColScore = S[i-1][j] + M[charMap(xChar)][0];
6     int leftRowScore = S[i][j-1] + M[0][charMap(YChar)];
7     S[i][j] = Math.max(diagScore, Math.max(leftRowScore, topColScore));
8    }
9  }
10 int finalScore = S[X.length()][Y.length()];
```

Listing 1: Sequential implementation of Smith-Waterman Algorithm for Optimal Scoring for Pairwise Sequence Alignment

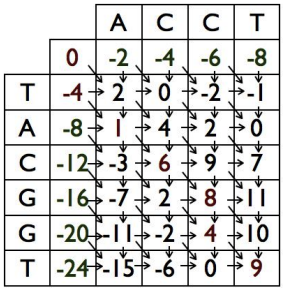

Figure 1: Dependences in Pairwise Sequence Alignment

This homework focuses on computing the optimal score for pairwise sequence alignment, not on the alignment itself. Though a biologist is ultimately interested in seeing the alignment, there are many applications where the score alone is of interest. For example, in multiple sequence alignment, the most commonly used approach is called progressive alignment, where an evolutionary tree is first built based on the scores of pairwise alignments, and then the tree is used as a guide for doing the multiple sequence alignment. In this case, the pairwise alignments are performed solely for the sake of obtaining scores, and the alignments themselves are not needed. However, it is important to compute the scores as quickly as possible when exploring alignments of large DNA sequences.

4 Your Assignment: Parallel Optimal Scoring for Pairwise Sequence Alignment

Your assignment is to design and implement parallel algorithms for optimal scoring for pairwise sequence alignment. We have provided a sequential implementation of the algorithm in `SeqScoring.hj` that you can use as a starting point. Your homework deliverables are as follows. All programs that you submit should take two command-line arguments, *string1* and *string2*, as in `SeqScoring.hj`.

1. [**Ideal parallelism with abstract execution metrics (20 points)**] Examine the dependence structure for $S[i, j]$ defined in Section 3 and create an ideal parallel version called `IdealParScoring.hj` that computes the same output as `SeqScoring.hj`, and delivers the maximum ideal parallelism ignoring all overheads. For analysis of ideal parallelism, assume that computing a single element of $S[i, j]$ takes one unit of time. You can reason about the ideal parallelism using a paper-and-pencil analysis, or by measuring abstract execution metrics (WORK,CPL) as in Homework 2. However, note that abstract metrics are only supported for the finish, async, and future construct (not data-driven futures). We recommend testing your solution on pairs of strings of length ≤ 10 for this part of the assignment.

2. [**Useful parallelism on Sugar compute nodes (30 points)**] Create a new parallel version of `SeqScoring.hj` that is designed to achieve the smallest execution time using 8 cores on a dedicated Sugar compute node, and call it `UsefulParScoring.hj`. This version need not reuse code from `IdealParScoring.hj`, but it's also fine if it does so. For this part of the assignment, we recommend first debugging your solution on small strings for correctness (which can be done on any platform), and then evaluating the performance of your implementation with pairs of strings of length $O(10^4)$ for performance evaluations on dedicated Sugar compute nodes (as explained in Lab 4).

   Since each Sugar compute node has 16GB of memory, it is recommended that you increase the maximum heap size to 8GB by using the -mx option when running HJ programs on a Sugar compute node (see Lab4 handout): "chj -mx 8000m -places 1:8 UsefulParScoring *string1 string2* ".

3. **[Sparse memory version and useful parallelism on Sugar compute nodes (30 points)]**

   The sequential algorithm outlined in Listing 1 and `SeqScoring.hj` allocates and uses a two-dimensional matrix which requires $O(n^2)$ space when processing strings of size $O(n)$. The goal of this part of the assignment is to create a *sparse* memory version of the program that can process strings of length $O(10^5)$ or greater by using space that's less than $O(n^2)$. The key idea to think about is what data really needs to be retained as the computation advances. For example, in the sequential version, row 1 of the S matrix can be freed (set to null and garbage-collected, or reused elsewhere) when the computation reaches row 3, since computation of row 3 only needs row 2 and not row 1.

   You will need to design and implement an analogous approach to reducing the space requirements of the parallel version. This will require reworking the data structure for matrix S, and may even require using a different algorithm from `UsefulParScoring.hj`. Call this version `SparseParScoring.hj`. As before, we recommend first debugging your solution on small strings for correctness, and then testing with pairs of strings of length $O(10^5)$ for performance evaluations on dedicated Sugar compute nodes. These runs may last a few minutes, so be sure to run these computations only on Sugar compute nodes, and not on the login node. Also, there may be be some impact of Java's garbage collection (GC) on the performance you observe. Please contact a teaching staff member if you believe that GC overheads are interfering with your performance measurements.

4. **[Homework report (20 points)]** You should submit a brief report summarizing the design of your parallel algorithms in `IdealParScoring.hj`, `UsefulParScoring.hj`, `SparseParScoring.hj`, explaining why you believe that each implementation is correct and data-race-free, and maximally parallel.

   Your report should also include the following measurements for parts 1–3 above:

   (a) Abstract CPL and WORK metrics for `IdealParScoring.hj` with input strings of length 10, assuming that the computation of each S[i,j] element takes unit time. These metrics can be obtained by paper-and-pencil analysis or by using HJ's abstract metrics.

   (b) Performance of `SeqScoring.hj` and `UsefulParScoring.hj` on a Sugar compute node with inputs of length (approximately) 10,000. `UsefulParScoring.hj` should be executed with the "-places 1:8" option to run with 8 workers (so as to use all 8 cores).

   (c) Performance of sequential and parallel versions of `SparseParScoring.hj` with inputs of length (approximately) 100,000. As before, `SparseParScoring.hj` should be executed with the "-places 1:8" option. The sequential HJ version can usually (but not always) be obtained by removing all parallel keywords (async, finish, etc.)

5 Generation of Test Data

You are welcome to generate any test data that you choose to debug your programs. Just keep in mind that they need to be strings of characters in $\{A, C, T, G\}$.

This is optional, but if you are interested in generating pairs of DNA sequences under realistic models of evolution, you can use a free web service available at `http://bibiserv.techfak.uni-bielefeld.de/rose/submission.html`. You should use the following options for this web service (with default values for everything else):

- *Sequence type:* Select DNA.
- *Number of sequences:* Enter 2.
- *Average length of sequences:* Enter whatever length you need e.g., 10, 10000, 100000, etc.
- *Average pairwise distance:* This parameter impacts the number of gaps that you are likely to see in the alignment (a larger alignment will lead to more gaps). We recommend entering 250 for sequences of length $O(10^5)$.