Homework 5: due by 11:59pm on Friday, April 22, 2022

Instructors: Zoran Budimlić, Mackale Joyner.

Programming Checkpoint 1 due by 11:59pm on Friday, April 8, 2022

Total score: 100 points

All homework should be committed in the GitHub classroom repository at `https://classroom.github.com/a/4ZfCBR6c`. In case of problems committing your files, please contact the teaching staff at `comp322-staff@mailman.rice.edu` before the deadline to get help resolving for your issues.

Your solution to the written assignment should be submitted as a PDF file named `hw_5_written.pdf` in the hw_5 directory. This is important — you will be penalized 5 points if you place the file in some other folder or with some other name. The PDF file can be created however you choose. If you scan handwritten text, make sure that you use a proper scanner (not a digital camera) to create the PDF file. Your solution to the programming assignment should be submitted in the appropriate location in the hw_5 directory.

The slip day policy for COMP 322 is similar to that of COMP 321. All students will be given 3 slip days to use throughout the semester. When you use a slip day, you will receive up to 24 additional hours to complete the assignment. You may use these slip days in any way you see fit (3 days on one assignment, 1 day each on 3 assignments, etc.). Slip days will be tracked in Canvas.

Other than slip days, no extensions will be given unless there are exceptional circumstances (such as severe sickness, not because you have too much other work). Such extensions must be requested and approved by the instructor (via e-mail, phone, or in person) before the due date for the assignment. Last minute requests are likely to be denied.

If you see an ambiguity or inconsistency in a question, please seek a clarification on Piazza (remember not to share homework solutions in public posts) or from the teaching staff. If it is not resolved through those channels, you should state the ambiguity/inconsistency that you see, as well as any assumptions that you make to resolve it.

Finally, please note that the programming project for this homework is more challenging than in the past homework. It is important for you to start early, and to meet the intermediate checkpoint to ensure that you are on track to complete the entire homework before the final deadline.

Honor Code Policy: All submitted homework is 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 else's work as your own. If you use any material from external sources, you must provide proper attribution.

1 Written Assignment: Locality with Places and Distributions (25 points)

Submit your solutions to the written assignments as a PDF file named `hw_5_written.pdf` *in the hw_5 directory. Please note that you be penalized 5 points if you misplace the file in some other folder or if you submit the report in some other format. The written portion is due along with the final submission by April 22, 2022.*

The use of the HJLib `place` construct (Lecture 30) is motivated by improving locality in a computer system's memory hierarchy. We will use a very simple model of locality in this problem by focusing our attention on remote reads. A remote read is a read access on variable V performed by task T0 executing in place P0, such that the value in V read by T0 was written by another task T1 executing in place P1 $\neq$ P0. All other reads are local reads. By this definition, the read of A[0] in line 8 in the example code below is a local read and the read of A[1] in line 9 is a remote read, assuming this HJLib program is run with 2 places, each place with one worker thread.

```
1.        finish {
2.          place p0 = place(0); place p1 = place(1);
3.          double[] A = new double[2];
4.          finish {
5.            async at(p0) { A[0] = ... ; } async at(p1) { A[1] = ... ; }
6.          }
7.          async at(p0) {
8.            ... = A[0]; // Local read
9.            ... = A[1]; // Remote read
10.         }
11.       }
```

Consider the following variant of the one-dimensional iterative averaging example studied in past lectures. We are only concerned with local vs. remote reads in this example, and not with the overheads of creating `async` tasks.

```
1.      dist d = dist.factory.block([1:N]); // generate block distribution (Lecture 31)
2.      for (point [iter] : [0:M-1]) {
3.        finish for(int j=1; j<=N; j++)
4.          async at(d[j]) {
5.            myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
6.          } //finish-for-async-at
7.        double[] temp = myNew; myNew = myVal; myVal = temp;
8.      } // for
```

1. **(10 points)** Write an exact (not big-O) formula for the total number of remote reads in this code as a symbolic function of the array size parameter, N, the number of iterations, M, and the number of places P (assuming that the HJLib program was executed using P places, and 1 worker thread per place). Explain your answer.
2. **(10 points)** Repeat part 1 above if line 1 was changed to "`dist d = dist.factory.cyclic([1:N]);`". Explain your answer.
3. **(5 points)** What conclusions can you draw about the relative impact of block vs. cyclic distributions on the number of remote reads in this example?

2 Programming Assignment (75 points)

2.1 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 a COMP 182 homework assignment 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 $|X'| = |Y'|$ *i.e.,* X' and Y' have the same size,

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

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:

$$\begin{array}{ccccccccc} X' & = & \text{-} & \text{A} & \text{C} & \text{-} & \text{C} & \text{T} \\ Y' & = & \text{T} & \text{A} & \text{C} & \text{G} & \text{G} & \text{T} \end{array}$$

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:

$$\begin{array}{llllllllllll} X'' & = \text{A} & \text{C} & \text{C} & \text{T} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-} & \text{-} \\ Y'' & = \text{-} & \text{-} & \text{-} & \text{-} & \text{T} & \text{A} & \text{C} & \text{G} & \text{G} & \text{T} \end{array}$$

2.2 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 2.1 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 $4 \times M_{p,-} + 6 \times M_{-,q} = -32$.

2.3 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 $S[i,j]$, $0 \le i \le m$ and $0 \le j \le n$, 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 $\forall i,j \ge 1$:

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

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.

```java
for ( i = 1; i <= xLength; i++)
 for ( j = 1; j <= yLength; i++) {
   int i = point.get(0);
   int j = point.get(1);
   char xChar = X.charAt(i-1);
   char YChar = Y.charAt(j-1);
   int diagScore = S[i-1][j-1] + M[charMap(xChar)][charMap(YChar)];
   int topColScore = S[i-1][j] + M[charMap(xChar)][0];
   int leftRowScore = S[i][j-1] + M[0][charMap(YChar)];
   S[i][j] = Math.max(diagScore, Math.max(leftRowScore, topColScore));
 }
int finalScore = S[xLength][yLength];
```

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

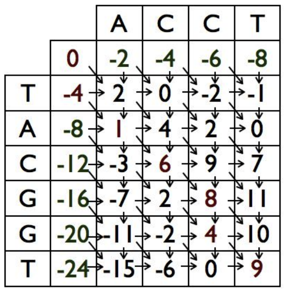

Figure 1: Dependences in Pairwise Sequence Alignment

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.

2.4 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.java` that you can use as a starting point. You should not use phasers for checkpoint 1. You should not use any parallel data constructs inside a constructor. Your homework deliverables are as follows:

1. [**Checkpoint 1 due on April 8, 2022: Ideal parallelism with abstract execution metrics (20 points)**] Examine the dependence structure for $S[i, j]$ defined in Section 2.3 and create an ideal parallel version called `IdealParScoring.java` that computes the same output as `SeqScoring.java`, 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, *e.g.,* by inserting a call to doWork(1) between lines 9 and 10 of Listing 1. You will need to insert this doWork() call at the appropriate location in the parallel solution you create in `IdealParScoring`. Your solution will be evaluated using HJLib's abstract metrics. You should not use phasers for Checkpoint 1. You may use phasers after Checkpoint 1.

   For this checkpoint, we have some unit tests in `Homework5Checkpoint1CorrectnessTest.java`.

   Hints based on common errors/omissions from past years: Remember to check that your solution passes all unit tests, and that you don't have any checkstyle errors.

2. [**Final submission due on April 22, 2022: Useful parallelism on NOTS compute nodes (40 points)**] Create a new parallel version of `SeqScoring.java` that is designed to achieve the smallest execution time using 16 cores on a dedicated NOTS compute node. Note that this is real execution time, not abstract metrics. Your code for this part will need to go into the `UsefulParScoring.java` file.

   For this part of the assignment, we recommend first debugging your solution on small strings for correctness (which can be done on any platform) using `Homework5Checkpoint2CorrectnessTest`, and then evaluating the performance of your implementation with pairs of strings of length $O(10^4)$ on dedicated NOTS compute nodes using `Homework5PerformanceTest`. You should be comfortable by now, having completed Labs 8 and 10 and HW4, with submitting jobs to the NOTS cluster.

   Even though each NOTS compute node has 32GB (or more) of memory, we will evaluate all homeworks using a maximum heap size of 8GB. The JVM heap size for tests running on NOTS is set in the provided pom.xml. Your submission will be evaluated with 8GB of heap, so changing this value in your pom.xml may result in incorrect test results. If you are running the JUnit tests locally through IntelliJ rather than using the provided pom.xml, you will want to add the following JVM command line argument to your Run Configurations to ensure that the JVM launched by IntelliJ is allowed to allocate up to 8GB of memory (in addition to the `-javaagent` argument that should already be there): `-Xmx8192m`

   However, note that some laptops do not have 8GB of physical memory, so running some of the larger tests locally may be prohibitively slow as your machine swaps memory pages out to disk as you exceed physical memory capacity.

   For this checkpoint, we have provided a set of unit tests in `Homework5Checkpoint2CorrectnessTest.java`. The `testUsefulParScoring` and `testUsefulParScoring2` tests in `Homework5PerformanceTest.java` also evaluate the performance of your `UsefulParScoring` implementation against the sequential version. We have provided a SLURM file under `src/main/resources` that can be used on NOTS to submit `Homework5PerformanceTest` for testing on a compute node. Note that you will need to edit this SLURM file to supply your e-mail for notification and to provide the correct path to your hw_5 folder on NOTS.

   Hints based on common errors/omissions from past years: Remember to check that your solution passes all unit tests, and that you don't have any checkstyle errors. Also, you should aim to get a speedup of $\geq 10\times$ for this part; you will get a 1-point deduction if the speedup is in the $[9, 10)$ range, a 2-point deduction if it is in the $[8, 9)$ range, etc.

3. [**Homework report (15 points)**] With the final submission you should submit a report file, formatted as a PDF file named hw5_report.pdf, summarizing the design of your parallel algorithms in `IdealParScoring.java` and `UsefulParScoring.java`, explaining why you believe that each implementation is correct and data-race-free. Your report should also include the following measurements for `UsefulParScoring.java` Execution time of `SeqScoring.java` and `UsefulParScoring.java` on a NOTS compute node with inputs of length 10,000. You can get these numbers from a run of the `Homework5PerformanceTest.testUsefulParScoring` test on NOTS (manually).