Alex Sanciangco

704050064

Lab 5 Report

1a) What are the item collections?

There are 3 item collections: the two input strings and the score matrix. The input strings are each of independent length N and M, and the scoring matrix is of dimensions N * M.

1b) What are the step collections?

This implementation only has one step, which is computeScore. The computeScore step will be essentially the body of the for-loop of the sequential Smith-Waterman algorithm.

1c) What are the dependencies and communications between collections?

The main dependency is between the computeScore step and the data it requires. Namely:

> Character 'i' of input string 1
> Character 'j' of input string 2
> Value of score[i-1][j-1], score[i-1][j], and score[i][j-1] where 'score' is the score matrix.

Then the value of score[i][j] is dependent on the output of the computeScore of i, j step. This dependency allows the program to "cascade" calculation through the score matrix. That is, if score[0][0] is in the top-left corner, then the calculations can only be done in the top left corner, since that's the only available data. But as more elements of the score matrix are computed, elements below, to the right, and at a down-right direction can be computed, creating the cascade effect.

2) How would you implement the kernel and main function?

The kernel (step function) would choose the maximum value of the 3 dependent cells (above, left, and upper-left) and their respective score value of either matching the letters or filling in a '-' character.

```
Kernel:
      score[i][j] = max(upper + upper-alignment-score,
                        left + left-alignment-score,
                        upper_left + upper_left-alignment-score)
```

The main function would do the same set-up as the sequential algorithm:

```
Main:
      Read input file(s)
      Store input strings from files
      Generate row 0 and col 0 of score matrix
      Use CnC implementation to populate score matrix
            Starting from score[1][1] // for dependencies to work
      Output result from lower-right corner of score matrix
```

3) Can you think of any advantage and disadvantage of CnC compared with MPI?

These two implementations are essentially opposite: CnC shows you what is dependent so the machine can parallelize everything else, while MPI specifies what can be parallelized and runs everything else sequentially. Depending on CnC's implementation, it will most likely run faster since the machine will parallelize everything that it can, whereas in MPI it's up to the programmer to find the best parallel implementation. This idea is similar to compilers that optimize code: the programmer might try to write code that's fast, but the compiler might move things around (loop unrolling, putting things in registers, etc.) that the programmer might have missed. Another disadvantage of MPI is the potential overhead of transferring data between processes.

A disadvantage of using CnC, however, is that the user has to write kernel code for each dependency. Similarly, the programmer must be completely aware of all dependencies in their program otherwise a CnC implementation will not work.