Parallelization of the Needleman-Wunsch Algorithm

Klasing, Blake
University of Central Florida
Orlando, FL
BKlasing@knights.ucf.edu

Gramajo, Stacy
University of Central Florida
Orlando, FL
SGramajo@knights.ucf.edu

1. Abstract

In Bioinformatics, one of the well-known problem is understanding evolutionary relationships through sequence alignments that can be found in a number set of data: protein, DNA, RNA, or genes. A fundamental method used for global sequence alignment is Needleman-Wunsch algorithm. It solves the problem of finding the optimal alignment among two sequences, based on a simple scoring system. We propose to examine this algorithm and implement a parallelized program that can decrease the execution time for analyzing two sequences without sacrificing correctness and time complexity.

2. Introduction

The Needleman-Wunsch(NW) algorithm is global sequence alignment method that exemplifies the relationship between sequences using a scoring system. Using a dynamic programming approach, there are two parts to the algorithm in order to retrieve the optimal score: creating a matrix and traceback. The traceback method is somewhat trivial that can be accomplished in linear time. However, the creation of the scoring matrix poses a more complex problem that we will be focusing on.

Overall, the NW algorithm achieves a $O(n \cdot m)$ time complexity and $O(n \cdot m)$ space complexity, where n and m are the lengths of the input sequences to be aligned. Although this is not an issue if the sequences are small, as they increase, the size of the scoring matrix as well as computing it grows in a quadratic fashion. For huge datasets that are the norm within the computational biology field, this poses an issue. For this reason, speeding up this algorithm can have great affects within the bioinformatics community, if not at least provide some insight into how to parallelize such an algorithm.

We propose to parallelize the method that computes the scoring matrix within the NW algorithm in order to improve the overall execution time. Our main goal is to achieve reasonable speedup versus the original NW algorithm by utilizing current consumer grade multicore processing architectures.

3. Related Work

The Needleman-Wunsch algorithm is widely known for solving this fundamental biology problem of global RNA sequence alignment. An algorithm proposed by Temple F. Smith and Michael S. Waterman is also very well known for solving a very similar problem, local RNA sequence alignment [3]. Both of these algorithms solve the problem of optimal RNA alignment by dynamic programming, reaching the same time complexity. There are many other implementations and concepts of solving the problem of non-structured optimal RNA alignment, but we will focus on the NW dynamic programming solution.

Some parallelized implementations of NW take advantage of the hardware being used to not only speed up the original non-parallelized algorithm, but also push this hardware to its limits in order to squeeze as much computation as possible out of it with a parallelized algorithm. A group of researchers at the University of Delaware create a highly parallelized implementation based on fine grain multithreaded execution and architecture model [2].

The majority of research around parallelizing the NW algorithm focuses on speeding up the calculation of the scoring matrix, which is where most of the computation takes place. However, some researchers focus on both the method of creating the scoring matrix, as well as the backtracking method [1]. Researchers at the University of Puerto Rico at Mayaguez bring a more complete approach of parallelizing the NW algorithm

by offering a parallel implementation of the backtracking method, in addition to the scoring matrix method. This group of researchers take advantage of properties of symmetry between the matrix D and D', where D is a matrix based on the original sequences being aligned, and D' is a matrix based on the same sequences in reverse order.

Other approaches include dividing the input sequences into several smaller sequences, calculating the matrices, backtracking to find the optimal score, and combining the results from each set of sequence alignments. This method is based on finding local optimal sequence alignment. This implies that it is not guaranteed to find the optimal global sequence alignment score. However, it may provide insight into possible solutions to parallelizing this global optima algorithm [4].

4. Background

4.1. Needleman-Wunsch

To compute the NW, the algorithm needs two sequences as the input. Assume in this case that sequence 1 is m and sequence 2 is n. Using the sequences, a table is created where the number of rows equal to length of m+1 and the number of columns equal to the length of n+1. After the creation of the matrix S, the first row and column in precomputed. Then the matrix takes each of the remaining cells in the matrix and computes the maximum of three equations:

$$S(i,j) = max \begin{cases} S(i-1,j-1) + c(i,j) \\ S(i-1,j) + c(i,-) \\ S(i,j-1) + c(-,j) \end{cases} \quad (1)$$

Once S has been filled, the final score is taken by the last item that was inputted. In order to print out the string containing gaps, mismatches, or matches, NW uses S to trace back starting at S [m+1, n+1].

The scoring system that is used to determine the overall score consists of three types of variables used: Gap, Mismatch, and Match. Gap is represented as the - character symbol that will appear in either/both of the sequences as the output solution. Match is the number added in the score if the characters of index i equal each other. Mismatch is usually a negative value that decrease the final score.

4.2. Multiprocessor Programming Concepts

In both the lock-based and lock-free implementation, it uses an unbounded pool as one of its architecture design. A bounded or unbounded item determines if the structure is limited to a specific capacity. If the object is unbounded, then it is not confined to a size; whereas, bounded is visa-versa. A pool is an object that contains methods where producer threads is able to store items and consumer threads are able to retrieve items to work on.

The lock-based method will contain a few mutexes. Mutexes are locks used to ensure mutual exclusion on critical sections. The lock-free does not use synchronization primitives, such as the mutex. It is a program that must be thread-safe when accessing a shared data. When constructing a lock-free program, primitives like the CAS (compare-and-swap) is used. The CAS is a type of instruction that has three parameters: the address of object x, value it is expected e, and value that it needs to change to v. When it is executed, CAS checks if x is equal to e. If true, then the value is updated to v and returns true; otherwise, x is not altered and returns false.

5. Overview

Our implementation program being build is a locked based system that contains a 2D array for the matrix, a pool, and mutexes. In the lock-based, the first row/column is first computed. The first thread begins calculated the first cell. When it is finished, the thread adds two items into the pool where other threads can retrieve and start computing its score. This cycle continues until the matrix has been complete. The lock in the program is used every time a thread retrieves an item in the pool to provide mutual exclusion. It is critical that two threads do not receive the same item from the pool that it would need to compute on. Additionally, there will be a second mutex that will increment every time a cell has been completed. The purpose of this variable is to ensure the program knows when to stop. If time permits, a second program will be constructed to be a lock-free based algorithm using the same idea presented in the first method. Unlike the first, the second algorithm will consist of compare-and-set operations with a status array to determine if a cell is Busy, Waiting, or Undefined. It will have similar concepts of the elimination stack presented in Chapter 10.

6. Experimental Results

-TBD-

7. Conclusion

-TBD-

8. Appendix

8.1. Challenges

- Lock based implementation
- acquiring implementations of related work that will compared against our algorithm.

8.2. Completed Tasks

- Implementation of sequential Needleman-Wunsch algorithm
- Abstract/Introduction/Related Work/Background complete within report
- Dataset acquired to test our algorithms

8.3. Remaining Tasks

- Complete parallel implementation of algorithm
- Create testing suite to ensure results are correct
- Indentify concurrency validation software to use
- Gathering results for comparing with our algorithm
- Complete report

References

[1] Carlos Torres Jaime Seguel. Paralellization of needleman-wunsch string alignment method. *BIOCOMP*, 1:239–244, 2011.

[2] W. S. Martins, J. B. Del Cuvillo, F. J. Useche, K. B. Theobald, and G. R. Gao. A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. *Pacific Symposium on Biocomputing*, 6:311322, 2001.

[3] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195197, 1981.

[4] F. Zhang, XZ Qiao, and ZY Liu. Parallel divide and conquer bio-sequence comparison based on smith-waterman algorithm. *Science in China Series F-Information Sciences*, 47(2):221231, 2004.