A Parallel Non-Alignment Based Approach to Efficient Sequence Comparison using Longest Common Subsequences

S. Bhowmick 1, M. Shafiullah 1, H. Rai and D. Bastola 2

1. Department of Computer Science, University of Nebraska at Omaha
2. School of Interdisciplinary Informatics, University of Nebraska at Omaha

Email dkbastola@unomaha.edu

Abstract. Biological sequence comparison programs have revolutionized the practice of biochemistry, molecular and evolutionary biology. Pairwise comparison is the method of choice for many computational tools developed to analyze the deluge of genetic sequence data. Unfortunately, a comprehensive study of the strengths and weaknesses of different alignment algorithms applied to different biological problems is not currently available. In this study we evaluate an alignment-free method of sequence comparison and present the results from the implementation of a scalable parallel algorithm for computing Longest Common Subsequence (LCS) between genetic sequences.

1. Introduction
A fundamental operation in bioinformatics involves the comparison of genetic (DNA) sequences. Similarity between genetic sequences is a strong indicator of evolutionarily preserved characteristics. This property has been successfully used in determining pathologically important bacteria, viruses and fungi [1-3].

Among the many sequence comparison tools for mining genetic information, an extremely common technique includes the alignment-based methods. These involve aligning the entire (global alignment, Needleman-Wunsch[4]) or smaller sections (local alignment, Smith-Waterman [5]) of the genetic sequences. The choice of global or local alignment is based on the type of analysis desired. However, both these methods are heavily dependent on the quality of sequence data. A slight discrepancies resulting from experimental or technical limitations, can significantly affect the comparison results.

Although, the fine granularity of comparative analysis is desirable when analyzing specific biological properties such as the single nucleotide polymorphism (SNP), an alternative approach of sequence analysis is becoming increasingly important in dealing with the exponential growth of genetic sequence data, classification and the grouping of organisms based on these sequences. Such alternative approaches include the alignment-free methods, which match the relative (as opposed to the exact) order of the base pairs in the sequence [6-9].

Advancements in sequencing technology have provided a deluge of genetic data. The Genbank, a public repository of genetic sequence data, reported 120604423 sequence records in its 178th release in June 15, 2010. Analyzing such large datasets on uniprocessor machines is an extremely time consuming process. It is imperative therefore, to harness the power of high performance computing to facilitate our understanding of this high throughput data.

In this paper, we present an efficient parallel non-alignment method for sequence comparison, based on finding the longest common subsequence (LCS), across genetic sequences. Our preliminary results demonstrate that this technique can be used to identify “signature-sequences” across multiple strains within different species of mycobacterium.
2. Sequence Comparison Using LCS
The longest common subsequence algorithm finds the longest subsequence between two strings. In contrast to the substring, the subsequence denotes a series of letters from the string which while being in order, need not be consecutive. For example, between ATCG and CTCAG, the longest common substring is TC, while the longest common subsequence is TCG.

LCS can help identify the key nucleotides across genetic sequences and is considerably less affected by the occasional sequencing error. This method is also useful for identifying potential regions of small mutations by analyzing the portions of the string not present in the LCS.

2.1. Computing the LCS
Our algorithm for computing LCS is inspired by the FAST_LCS method described in [10]. The algorithm involves creating position pairs of identical letters in the sequence and combining them to potential LCS strings as long as their relative order is maintained across the pairs. Given two position pairs (i,j) and (l,m), the corresponding letters would be a potential candidate for LCS only if i<l and j<m.

In the sequences given above; the position pairs are; A:(0,3); C:(2,0) and (2,2); T:(1,1) and G: (3,4). Some strings preserving the relative order are: (i) AG : (0,3)->(3,4); (ii) TG: (1,1)->(3,4); and (iii) TCG: (1,1)->(2,2)->(3,4)

The algorithm consists of obtaining these position pairs and then creating a tree where the vertices represent the position pairs and the edges, their order in the subsequences. The diameter of the tree provides the LCS, as shown in the Figure1.

![Figure 1.](image)

In order to improve the time to traverse the tree, [10] suggested pruning techniques, whereby some vertices can be deleted from their original positions and later added at a new position. This helps remove redundant edges that do not contribute to the LCS. For example though both TG and TCG are
subsequences, the vertex representing the position pair of G can be later reconnected to the position
pair of C, instead of being connected to T.

However, pruning is a computationally expensive algorithm since (i) it does not eliminate the
initial redundant additions, (ii) deletion of edges requires more operations than additions and (iii)
identifying the position of the redundant vertices requires periodic tree traversals.

To avoid these excess operations, we order the pairs such that a vertex appearing later in a string is
not visited earlier. In the above example the ordered position pairs will be A:(0,3), T:(1,1), C:(2,0),
C:(2,2), and G:(3,4). Therefore, TG will not be connected before TC. Furthermore, we ensure that a
new letter is connected only to the longest candidate string, thereby reducing the number of edges to
traverse.

Let the length of a string with L different symbols be given by;

\[ M = \sum_{i=1}^{L} m_i \]

where \( m_i \) is the number of occurrences of the symbol i.

The complexity of computing LCS, for two strings of length \( M = \sum_{i=1}^{L} m_i \) and \( N = \sum_{i=1}^{L} n_i \), is
proportional to the number of vertices and is given by \( \alpha \sum (m_i \times n_i) \).

2.2. Parallel Implementation of LCS

We have designed a scalable parallel version of the LCS algorithm as outlined in Figure 2. First, we
divide each string across the processors and the LCS of the substrings in each processor (LCS1 and
LCS2) are computed. Then we interchange the portions of strings (grey areas) that were beyond the
first and last positions in the LCS. We compute LCS for these previously unused strings. Finally,
we combine the respective portions to obtain the complete LCS.

![Figure 2. A Schematic Diagram of the Parallel LCS Algorithm](image-url)

**Figure 2.** A Schematic Diagram of the Parallel LCS Algorithm
3. Experimental Results
We conducted three sets of experiments. In the first of experiments we focused on verifying that LCS clustering did indeed mimic the clustering as obtained by MSA. In our second set of experiments we investigated how parallel computing can improve the execution time for obtaining LCS. Finally, the third set of experiments involved modifications of the parallel algorithm to further improve the execution time without reducing the accuracy of clustering.

Figure 3a. Clustering of sequences using multiple sequence alignment. The different colors indicate the different species of mycobacterium

Figure 3b. Clustering of sequences using longest common subsequences across species. The different colors indicate the different species of mycobacterium. The species *intercellulare* and *fouruitum* have multiple LCS due to internal mutations.
3.1. Comparison of LCS and MSA Clustering

In our first set of experiments we computed the LCS of a portion of the genomic sequences of five species of *Mycobacterium* on a sequential program. We observed that the number of distinct LCS produced per sequence pair is proportional to their variance in their base pair composition. It is observed that the sequence with low variance in composition (avium, kansasii, gordone) show a single LCS while divergent sequences (fortuitum, intercellulare) show multiple-LCS fragments.

We compared the phylogenetic clustering of the *Mycobacterium* using only the LCS sequence (Figure 3b) from each group of species with that obtained through multiple sequence alignment (MSA) (Figure 3a) of all sequences across all the species. Our results indicate that the LCS clustering follows the MSA clustering very closely.

3.2. Parallel Implementation of LCS

We implemented LCS based on the parallel algorithm sketched in Figure 2. Our parallel experiments were conducted on the Firefly 1,151 node linux cluster with distributed memory architecture connected through infiniband [11]. We use MPI to implement the distributed memory parallel program.

We performed a pairwise comparison over the four genome sequences of *Mycobacterium tuberculosis*, with an average length of 1500 base pairs. The genome sequences used for this study were; *Mycobacterium_tuberculosis_CDC1551, Mycobacterium_tuberculosis_F11, Mycobacterium_tuberculosis_H37Ra and Mycobacterium_tuberculosis_H37Rv.*

Each pair of sequences (total six pairs) was divided into a set of processors ranging from 16 to 128. In the current version of the program we are comparing only the portions of the sequence that are allocated to the same processor (Step 1 in Figure 2). The total length of the LCS is obtained by adding the partial LCS obtained from each processor. Table 1 shows the length of the LCS across the six sequence pairs. There is a slight variation in the values across processors, because we are yet to accommodate LCS values that fall across processors (Step 2 in Figure 2). However, it is interesting to note that the variance amongst the LCS per sequence pair across the processors (variance per column of Table 1) is quite low. These results lead us to conjecture that we can obtain an accurate clustering of genome sequences by just obtaining this partial value of the LCS.

**Table 1.** The length of the LCS (over 16 to 128) for pairwise comparison of Genome Sequences from *Mycobacterium tuberculosis*.

| Processors | LCS of Sequence 1 | LCS of Sequence 1 | LCS of Sequence 1 | LCS of Sequence 2 | LCS of Sequence 2 | LCS of Sequence 3 |
|------------|------------------|------------------|------------------|------------------|------------------|------------------|
| 16         | 1161             | 1118             | 926              | 1161             | 850              | 833              |
| 32         | 1128             | 1100             | 914              | 1087             | 843              | 830              |
| 64         | 1103             | 1098             | 914              | 1025             | 846              | 817              |
| 128        | 1111             | 1105             | 971              | 1008             | 894              | 857              |

Figure 4, gives the lower bound on the execution time of the parallel LCS. As can be seen from the results, that the current version of the algorithm (implementing only step 1) is highly scalable.

Note that due to the availability of over a thousand nodes and the mutual independence between the LCS computations of any two pairs of sequences, we could execute the parallel experiments for pairwise comparison simultaneously. That is, when the LCS of one pair of sequences is being computed over *n* processors, the LCS of another pair can be computed at the same time over other *n* processors. Thus, we are employing a two level parallelism. In the first level we use the embarrassingly parallel
nature of pair wise comparison to allocate each pair over a group of processors. In the second level we use a more tightly coupled distributed memory paradigm to compute the actual LCS.

In the next version of the paper we will show the scalability results for computing the entire LCS across strings (Steps 1 to 4 in Figure 2). We will also show that if the end regions of each substring (A, B, C, D, E and F in Figure 2) are small enough then Step 1 is sufficient to generate a reliable LCS. We will verify our claim by comparing the clustering obtained by these partial LCS with the complete LCS.

4. Conclusions
We have developed a scalable parallel algorithm for computing LCS between strings. Our results demonstrate that longest common subsequence can be used as an effective non-alignment based technique for genetic sequence comparisons. Our parallel experiments show that LCS is indeed very scalable and can be used to compare entire genomic sequences. In future, we intend to investigate the relation between occurrences of multiple LCS across the same species to intra-species mutations.

5. References
[1] Rispail N, Soanes D.M., Ant C, Czajkowski R, Grünler A, Huguet R, Perez-Nadales E, Poli A, Sartorel E, Valiante V, Yang M, Beffa R, Brakhage AA, Gow NA, Kahmann R, Lebrun MH, Lenasi H, Perez-Martin J, Talbot NJ, Wendland J and Di Pietro A 2009 Comparative genomics of MAP kinase and calcium-calci neurin signalling components in plant and human pathogenic fungi. *Fungal Genet Biol.* **46** 287-98

[2] Dong QJ, Wang Q, Xin YN, Li N and Xuan SY 2009 Comparative genomics of Helicobacter pylori. *World J Gastroenterol* **15** 3984-91

[3] Ma LJ, van der Does HC, Borkovich KA, Coleman JJ, Daboussi MJ, Di Pietro A, Dufresne M, Freitag M, Grabherr M, Henrissat B, Houterman PM, Kang S, Shim WB, Woloshuk C, Xie X, Xu JR, Antoniw J, Baker SE, Bluhm BH, Breyspear A, Brown DW, Butchko RA, Chapman S, Coulson R, Coutinho PM, Danchin EG, Diener A, Gale LR, Gardiner DM, Goff S, Hammond-Kosack KE, Hilburn K, Huan-Van A, Jonkers W, Kazan K, Kodira CD, Koehrsen M, Kumar L, Lee YH, Li L, Manners JM, Miranda-Saabedra D, Mukherjee M, Park G, Park J, Park SY, Proctor RH, Regev A, Ruiz-Roldan MC, Sain D, Sakthikumar S, Sykes S, Schwartz DC, Turgeon BG, Wapinski I, Yoder O, Young S, Zeng Q, Zhou S, Galagan J, Cuomo CA, Kistler HC and Rep M 2010 Comparative genomics reveals mobile pathogenicity chromosomes in Fusarium. *Nature.* **464** 367-73

[4] Needleman SB and Wunsch CD 1970 A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **48** 443-53

[5] Smith TF and Waterman MS 1981 Comparison of biosequences. *Adv. Appl. Math.* **2** 482–89

Figure 4. The execution time of parallel LCS algorithm as the number of processors are increased.
[6] Domazet-Loso M and Haubold B 2009 Efficient estimation of pairwise distances between genomes. *Bioinformatics.* 25 3221-7

[7] Elemento O and Tavazoie S 2007 Fast and systematic genome-wide discovery of conserved regulatory elements using a non-alignment based approach. *Genome Biology* 6 R18

[8] Quest D, Tapprich W and Ali H 2007 A grammar based methodology for structural motif finding in ncRNA database search. *Comput Syst Bioinformatics Conf.* 6 215-25

[9] Jia C, Liu T, Zhang X, Fu H and Yang Q 2009 Alignment-free comparison of protein sequences based on reduced amino acid alphabets. *J Biomol Struct Dyn.* 26 763-9

[10] Chen Y, Wan A and Liu W 2006 A fast parallel algorithm for finding the longest common sequence of multiple biosequences. *BMC Bioinformatics* 7 S4

[11] Holland Computing Center. [http://hcc.unl.edu/firefly](http://hcc.unl.edu/firefly)

**Acknowledgement**

This project was supported by the NIH grant number P20 RR016469 from the INBRE Program of the National Center for Research Resources.