PARALLEL LCS ALGORITHM FOR PAIRWISE SEQUENCE ALIGNMENT

Pich Tantichukaitikul[1], Sattara Hattirat[1] and Jonathan H. Chan[1,2*]

[1]Bioinformatics and Systems Biology Program, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
[2]School of Information Technology, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
*Corresponding Email: jonathan@sit.kmutt.ac.th

ABSTRACT

Multicore processors are gaining ground in the world of modern computers. With faster performance, it would allow DNA or protein alignment, which is a fundamental procedure in molecular biology, to be done faster, paving the way to faster multiple genome comparison. However, in order to harvest the advantages of multicore processors, corresponding parallel algorithms need to be developed and tested. This study aimed to develop a suitable parallel longest common subsequence (LCS) algorithm for pairwise alignment. The proposed parallel LCS performed approximately 30-40% faster than serial LCS, while yielding the same alignment results.

Index Terms—*Parallel algorithm; longest common subsequence; biological sequence alignment; computational biology*

1. INTRODUCTION

Biological sequence comparison has numerous applications and has been extensively studied by biologists. Sequence alignment deals with problems arising from processing DNA and protein sequence information: to find the homology between two or more organisms. The sequences of different organisms differ to some degree. Homologies in two different genomes suggest conserved biological structures and functions [1]. This could be used in further biological analyses on the organisms' ecological niche and evolution [2]. Sequence alignment, due to its usefulness and various applications, has become a fundamental operation in biology and bioinformatics [3].

To align biological sequences, different algorithms for finding the longest common subsequence (LCS) are used. The LCS problem for two sequences (2LCS) can be solved in O(*mn*) by using the dynamic programming technique where *m* and *n* are the lengths of the two input sequences [4]. Examples of common algorithms for solving LCS problems are dynamic programming and Hidden Markov Model. The former is known to be more accurate, providing optimal solutions, while taking longer time to run and requiring more computational resources. The latter, on the contrary, runs faster and uses less resource while providing acceptable results. In order to obtain a non-compromised result, we focus on dynamic programming algorithm for LCS and aim to take advantage of the current multiple-cored processor technologies to reduce the run-time of dynamic programming LCS.

Due to a number of physical limitations such as power consumption and heat dissipation [5], it is no longer possible to engineer increasingly more powerful chips by increasing the number of transistor components while reducing their sizes. Instead, manufacturers have turned to building chips with multiple processor cores. A single processor in a multicore processor architecture does not necessarily perform as fast as the latest designed single-core models. Nonetheless, they improve overall performance by handling more work in parallel [6]. It is estimated that a dualcore chip running multiple applications is approximately 1.5 times faster than a chip with just one comparable core [6].

However, there has been a lag on the development on parallelizing compilers. This could be due to the increased complications in designing a parallel algorithm. Also, parallelism works only for a restricted class of problems [5]. Consequently, many applications are not designed or have not been rewritten to be run in parallel architectures. Therefore, even though we are in the multicore era, the advantage of the multiple execution units of processors has not yet been fully materialized.

Despite its importance in biological arena, there have been very limited studies for the parallel sequence alignment problem. Biological sequence alignment with LCS dynamic algorithm could take longer time when the sequences become longer. This study aims to experiment on parallel pairwise alignment with LCS dynamic programming algorithm to find out if the problem of biological sequence alignment could be rewritten with parallelism and still yield optimal results. PLCS in this paper refers to parallel LCS.

2. PROBLEM FORMULATION

2.1. LCS Problem

To align two biological sequences, the letters which represent either amino acids or bases are shifted either right or left to align as many identical letters as possible. Gaps (denoted by "-") are inserted into the sequences to obtain better alignment [3]. Scoring functions are used to represent the degree of matching between each pair of the letters. An optimal alignment is one with the highest cumulative score. Note that it is possible for multiple optimal alignments; however, this is unlikely for increasingly longer sequences.

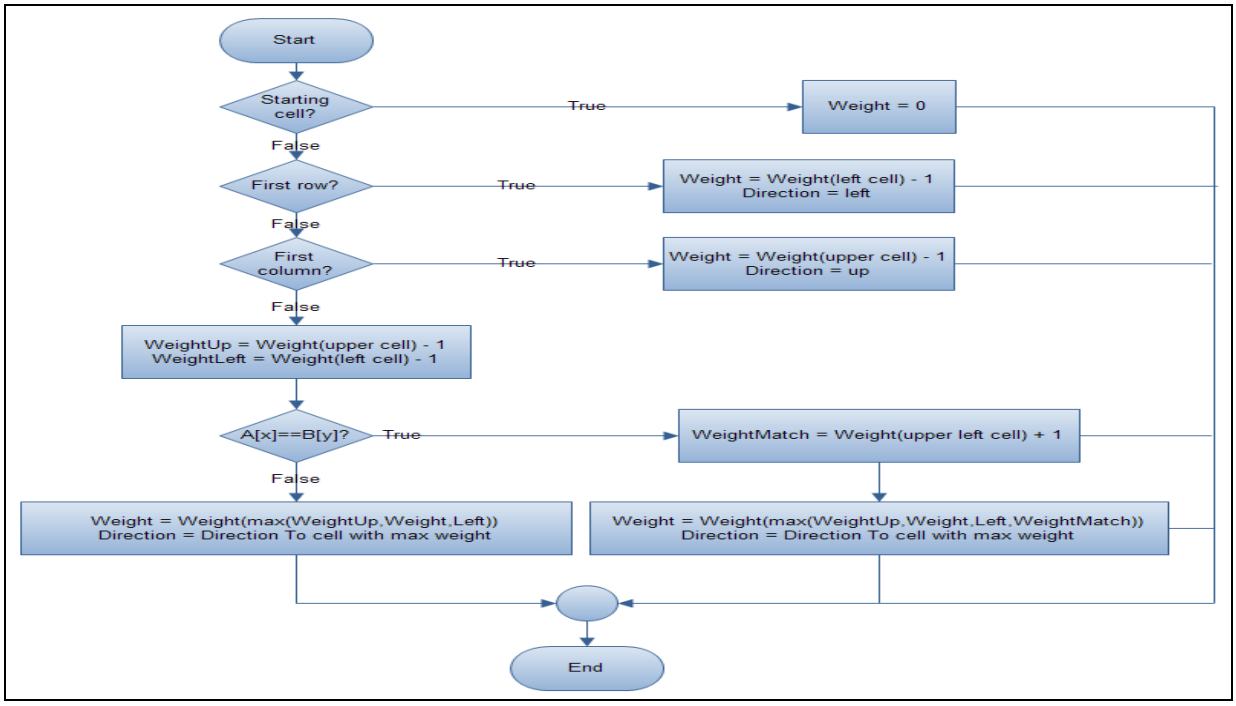

Figure 1. Flowchart of score filling in traditional LCS.

The flowchart in Figure 1 shows how traditional LCS fills similarity scores in the edit table during the dynamic programming process. Note that both vertical and horizontal cells are filled at the same time in a sequential manner.

2.2. PLCS Algorithm

Important things to concern when creating a serial algorithm are the accuracy of the program results and the resources (time and memory) required for the program to run [7]. In the case of parallel computing, result accuracy has to be taken more into account [5]. This is due to the data dependency between multi-concurrent threads. For example, one processor might retrieve the results for further calculation before the result is ready or before the job of the other processor is completed. In addition, deadlocks could occur when the first thread waits for the second thread while the second thread also waits for the first one, rendering the program to cease.

Another challenge in parallel programming is to balance the work load between CPUs in order to gain the optimal CPU allocation and a better run time. In order to obtain a better run time, CPU waiting should be minimized. That is, ideally, the CPU would not need to wait for the results from other processes before the next operation.

Taking the above considerations into account, we could write a parallel LCS algorithm as described and shown in Figure 2.

Referring to the figure, sequence A and sequence B have length *lA* and *lB*, respectively. The starting points to fill the edit table are *startX* and *startY*, both being at location 0. The variable *start* is for confirmation of the readiness of *startX* and *startY*. The status '*lock*' means the thread is in critical session. The initial status of *start* is '*unlock*'. Memory on RAM is allocated for WeightTable (for assigning the scores in each cell) and DirTable (for determining the direction of table filling). The vertical and horizontal threads are created using different filling patterns. When the processes in both threads are completed, every cell in the table is filled. The program will use the directions in filling of the table to traceback to obtain the optimal path for LCS.

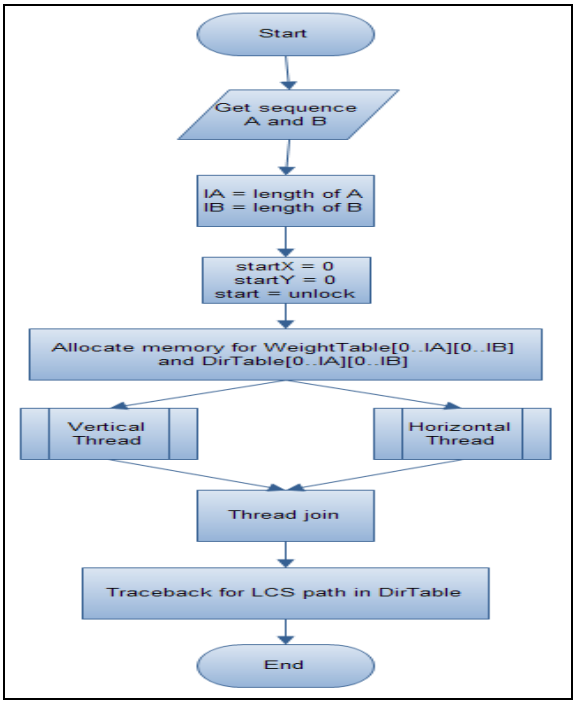

Figure 2. Parallel LCS flowchart.

The flowchart in Figure 3 shows how the table is filled by thread in the horizontal and vertical directions. Thread Vertical fills the similarity scores for the cells

along the vertical direction. Thread Horizontal fills in the cells along horizontal direction. Both threads work until the start point is outside the table reference position. They begin filling by changing the variable *start* from the *unlock* state to *lock* state. Next, the value *startX* and *startY* are used as reference variable for filling the table. Then the working position shifts. Thread Vertical shifts to $X+1$ position while Y remains unchanged. Thread Horizontal shifts to $Y+1$ while X remains unchanged. Next, variable *start* changes to *unlock*. Then both threads start filling the scores in their own directions until reaching the stopping criteria.

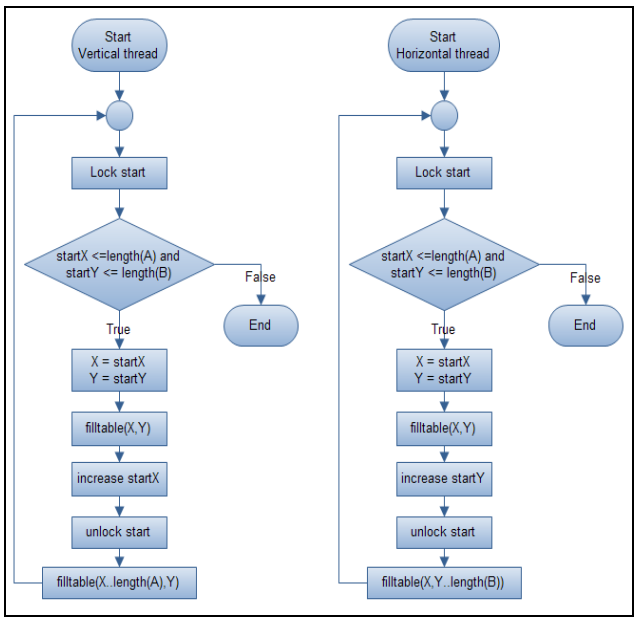

Figure 3. Flowchart of score filling in parallel LCS.

Figure 4 shows the topology of the order of filling for a sample pairwise alignment of DNA sequences ATCTGATC and TGCATAC. The vertical and horizontal threads are represented by the vertical and horizontal rectangles in the figure. Whereas the arrows denote the order of simultaneous data filling of the edit table from the top left to the bottom right by the two parallel threads.

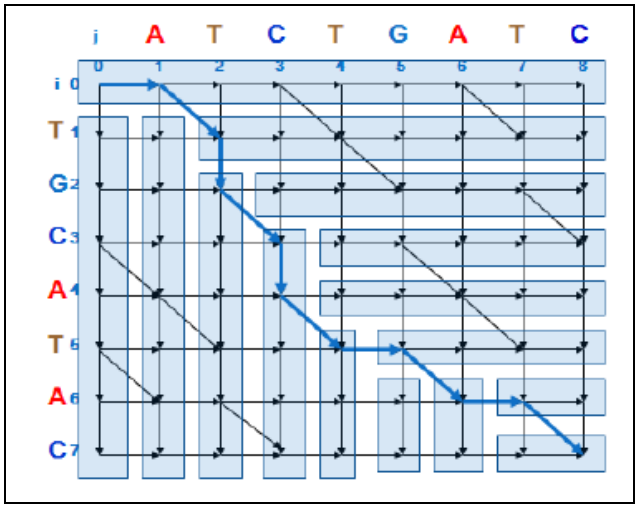

Figure 4. Topology of data filling in PLCS method.

3. METHODOLOGY

To evaluate the performance of parallel LCS algorithm compared to traditional LCS algorithm, it is necessary to control the factors which could affect the running time of the program. Both algorithms were implemented in C and were compiled using GNU Complier Collection (GCC). The random sequences of A, T, C, and G were generated for testing inputs of each program. Each program was then executed on the same computer, equipped with 2 Itanium CPUs which have a clock rate of 1.33GHz.

4. EXPERIMENTAL RESULTS AND DISCUSSION

The accuracy of parallel LCS and traditional LCS is the same. That is, the alignments from both algorithms are the same. However, the calculation time is improved by about 30% and 44% when tested with randomly generated 1,000-bases sequence alignment and 10,000-bases sequence alignment, respectively.

The means, standard deviations, minimal values and maximal values of the calculation time are shown in Table 1. The percentages of improvement are calculated from the medians, also shown in Table 1. The reduced running time is clearly shown in Figures 5 and 6 for 1,000 bases sequence alignment and 10,000 bases sequence alignment, respectively. The boxplot on the left of the figures represent the running time for the serial LCS while the right boxplot represents that of the parallel LCS. The boxes are very thin so they appear like grey horizontal lines. The whiskers represent standard deviations and the black dots are outliers.

Table 1. Comparison of the running time of serial and parallel LCS algorithms

	Running time (seconds)			
	1,000bases		10,000 bases	
	serial	parallel	serial	parallel
mean	0.087	0.069	7.505	5.209
SD	0.007	0.013	0.041	0.097
min	0.084	0.064	7.434	5.007
max	0.138	0.161	7.717	5.510
Q1	0.085	0.065	7.48075	5.1375
*median	*0.085*	*0.065*	*7.499*	*5.199*
Q3	0.086	0.066	7.52425	5.262

The standard deviations varied because of the random workload of the cluster computer used for running the programs. As the cluster is a public cluster and can be accessed by many remote users, it could be doing other jobs while performing both LCS and PLCS, resulting in CPU sharing and different running time results. The standard deviations of LCS performance were higher, as well as the spread of the outlier range. This was expected since in the case of parallel computing, all CPUs were used. CPU sharing, thus, affected the performance of PLCS more significantly. However, it could be seen that PLCS still performed faster regardless of the random incremental workloads by other users of the cluster.

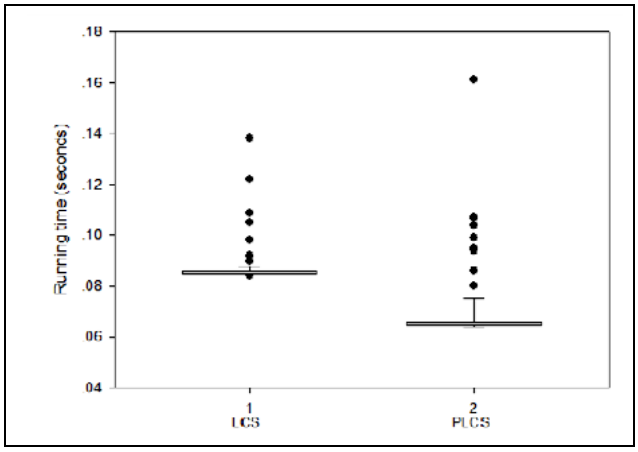

Figure 5. Boxplot comparing run time of serial LCS and parallel LCS for 1,000-bases sequence alignment.

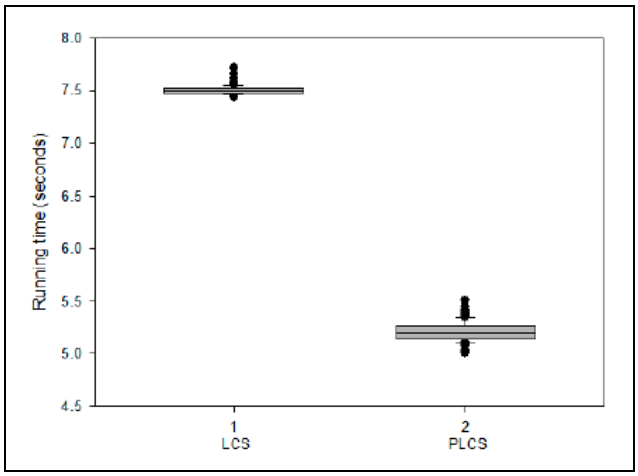

Figure 6. Boxplot comparing run time of serial LCS and parallel LCS for 10,000-bases sequence alignment.

The performance of PLCS was not much improved in 1,000-bases sequence alignment than in 10,000-bases alignment. This is because the time taken for the sequences in both alignment jobs to be uploaded into the cluster was not different, providing a constant initial running time for both LCS and PLCS. Therefore, PLCS showed greatly improved performance when the running time is sufficiently long.

5. CONCLUDING REMARKS

Parallel LCS could perform faster than LCS while yielding the same optimal solutions. There have been increasing studies on comparative genomics [8] which require longer running time for alignment due to the use of whole genome sequences. Such analyses, which require multiple genome comparison, have a wider range of applications [2]. There should be further development on parallel multiple biological sequence alignments to realize the increased power of multicore computing.

ACKNOWLEDGMENT

The authors would like to thank Itanium cluster, Large Scale Simulation Research Laboratory, NECTEC for the use of their cluster.

REFERENCES

[1] K-M Chao and L. X. Zhang, *Sequence Comparison: Theory and Method*. London: Springer-Verlag, 2009.

[2] C. M. Fraser, J. Eisen, R. D. Fleischmann, K. A. Ketchum, and S. Peterson, "Comparative genomics and understanding of microbial biology," *Emerging Infectious Diseases*, vol. 6, no. 5, pp. 505–512, 2000.

[3] A. Driga, P. Lu, J. Schaeffer, D. Szafron, K. Charter and I. Parsons, "FastLSA: A Fast, Linear-Space, Parallel and Sequential Algorithm for Sequence Alignment," *Algorithmica*, vol. 45, pp. 337-335, 2006.

[4] S. J. Shyu and C. Y. Tsai, "Finding the longest common subsequence for multiple biological sequences by ant colony optimization," *Computers and Operations Research*, vol. 36, pp. 73–91, 2009.

[5] A. Buttari, J. Langou, J. Kurzak and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, pp. 38-53, Jan. 2009.

[6] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11-13, 2005.

[7] M. A. Weiss, *Data Structures and algorithm analysis in C*, 2nd Ed. California: Addison-Wesley, 1997.

[8] T. T. Binnewies, Y. Motro, P. F. Hallin, O. Lund, D. Dunn, T. La, D. J. Hampson, M. Bellgard, T. M. Wassenaar and D. W. Ussery, "Ten years of bacterial genome sequencing: comparative-genomics based discoveries," *Functional and Integrative Genomics*, vol. 6, pp. 165-185, 2006.