Proceedings of the 26th Annual International Conference of the IEEE EMBS
San Francisco, CA, USA • September 1-5, 2004

A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching

Hsien-Yu Liao, Meng-Lai Yin, Yi Cheng
Electrical and Computer Engineering Department
California State Polytechnic University, Pomona
hsienyuliao@csupomona.edu, myin@csupomona.edu

Abstract—**Efficient biological sequence searching is an important and challenging task in bioinformatics. Among those fundamental sequence analysis algorithms, the Smith-Waterman algorithm that adopts the dynamic programming mechanism provides very high sensitivity. Unfortunately, the inefficiency in performance of this algorithm limits its applications in the real world. With the advances in the engineering technology, massive parallelism can be achieved using the FPGA[1]-based techniques. In this paper, a parallel implementation methodology of the Smith-Waterman algorithm is presented. This method provides magnificent speedup over the traditional sequential implementation, while maintain the same level of sensitivity.**

Keywords—**biological sequence alignment, massive sequences searching, parallel processing, Smith-Waterman algorithm.**

1. Introduction

Genetic sequence searching in general is highly computationally intensive and requires vast amounts of processing power. The most basic sequence analysis task is to ask if two sequences are related. Among those fundamental algorithms that address the above issue, the Smith-Waterman algorithm [1, 6] which is based on dynamic programming [7] provides a high sensitivity. The time complexity of this algorithm for comparing two sequences is *O(mn)*, where *m* and *n* are the lengths of the two sequences being compared. Although this computational complexity may not seem threatening, the growth in the genetic bio-sequence database is exponential. Thus, the complexity that concerns the real world applications is really *O(kmn)*, where *k* represents the exponential growth of the size in genetic databases. This exponential expansion of the genetic sequence databases motivates this research. Our goal is to develop methodologies that can best utilize the available computation power provided in the advanced FPGA technology to achieve high-sensitivity and high-efficiency genetic sequence searching. The Smith-Waterman algorithm is a good candidate for this research, due to its high-sensitivity characteristic.

2. Backgrounds

There are many existing tools for sequence alignment. Among those, FASTA[2] [2] and BLAST[3] [3] are two commonly used ones, where the time complexity has been reduced through some heuristic algorithms[4]. These heuristics algorithms obtain efficiency, however, at the expense of sensitivity. As a result, a distantly related sequence may not be found in a search using the above tools.

Researchers have been worked on this issue through different approaches. For example, Fa Zhang, Xiang-Zhen Qiao, and Zhi-Yong Liu [4] presented a parallel Smith-Waterman algorithm based on divide and conquer that can reduce running time and memory requirement. However, their method is also at the cost of losing sensitivity. Other methods [5] that apply standard computer systems such as high performance supercomputers and computer clusters with software solutions for conducting the Smith-Waterman algorithm, although can achieve high sensitivity and reduce running time, are with extremely high cost. With the advance technology in the FPGA, a cost-efficient parallel implementation for the Smith-Waterman algorithm can be obtained.

3. The Challenge

The Smith-Waterman algorithm, based on the dynamic programming technique, was used to compute

[1] FPGA stands for "Field Programmable Gate Arrays".

[2] FASTA stands for "Fast Alignment Sequence Tools - All".

[3] BLAST stands for "Basic Local Alignment search Tool".

[4] Heuristic algorithms are used to find solutions among all possible ones. They are faster and easier, but do not guarantee that the best will be found.

0-7803-8439-3/04/$20.00©2004 IEEE

the optimal local alignment of two sequences. The procedure consists of three steps:

1) Fill in the dynamic programming matrix.
2) Find the maximal value (score) in the matrix.
3) Trace back the path that leads to the maximal score to find the optimal local alignment.

Let $S = s_1\ s_2\ \ldots\ s_m$ of length m and $T = t_1\ t_2\ \ldots\ t_n$ of length n be the two sequences for comparison. The basic Smith-Waterman algorithm for calculating the best score between the two sequences S and T is to construct a $(m+1)\times(n+1)$ matrix D, named the dynamic programming matrix. D is indexed by i and j, where each index is for one sequence. In particular, a cell D_{ij} can be built recursively using the following recurrent relations.

$$For \quad 0 \le i \le m, 0 \le j \le n,$$
$$D_{i0} = D_{0j} = 0 \qquad (1)$$

$$For \quad 1 \le i \le m \ and \ 1 \le j \le n,$$
$$D_{ij} = \max\begin{cases} 0, \\ D_{(i-1)(j-1)} + Sbt(s_i, t_j), \\ D_{(i-1)j} - gap\cos t, \\ D_{i(j-1)} - gap\cos t. \end{cases} \qquad (2)$$

The *gap cost* in equation (2) is the penalty for inserting a gap character '-'. This could be s_i aligned to a gap ('-') or t_j aligned to a gap ('-'). The value $Sbt(s_i, t_j)$ of a substitution matrix is for scoring a match or a mismatch. Applying equation (2), all values of the matrix can be obtained. Note that this step takes $O(mn)$.

The second step in the procedure is to find the maximal value in the matrix. Once the maximal value is found, the third step can be taken, which traces back from the maximal value until a 0 value is reached.

		c	a	g	c	g	t	t	g
	0	0	0	0	0	0	0	0	0
a	0	0	2	0	0	0	0	0	0
g	0	0	0	4	2	2	0	0	2
g	0	0	0	2	3	4	2	0	2
t	0	0	0	0	1	2	6	4	2
a	0	0	2	0	0	0	4	5	3
c	0	2	0	1	2	0	2	3	4

Figure 1: The dynamic programming matrix and the tracing back path

As an example, suppose that one wishes to compare sequence $S = a\ g\ g\ t\ a\ c$ with sequence $T = c\ a\ g\ c\ g\ t\ t\ g$. Assume $Sbt(s_i, t_j) = \begin{cases} +2 & if(s_i = t_j) \\ -1 & else \end{cases}$ and *gap cost* = 2. Figure 1 illustrates the calculation of the dynamic programming matrix D and the path of tracing back. The best score found in the matrix is 6, and the corresponding optimal local alignment is

$$\begin{array}{cccccc} T: & a & g & c & g & t \\ S: & a & g & - & g & t \end{array}$$

The above steps show the procedure for comparing two sequences. As addressed before, the real challenge in the applications of the algorithm is on the exponential growth of the genetic database. In other words, the above procedure needs to be applied a great number of times, i.e. represented as k in our previous discussion which has an exponential growth. If the comparison is performed using the traditional sequential processing platform, the time complexity would be intolerable. With today's FPGA technology, massive parallelism can be provided with affordable price. How to take full advantages of the available parallelism is the key for success. In the following, we show the approach and the detailed method in achieving a cost-effective parallel implementation of the Smith-Waterman algorithm.

4. The Approach & the Methodology

The most interesting aspect of the parallel implementation here is not on each individual D_{ij}'s computation, but rather on the flow of computation for the entire matrix, and beyond. As will be explained later, the power of parallelization lays on the massive comparisons. Note that in our case, the number of comparisons grows exponentially, represented as 2^k in the previous discussion. When an unknown sequence is compared with different existing sequences, each comparison is independent and can be performed independently. This observation highlights the potential of massive parallelism existing in this particular application. Our method can achieve very high utilization of the parallelism and obtain great performance gain.

4.1. Single sequence pair comparison

For a single comparison, one dynamic programming matrix needs to be filled. This process can be done using certain level of parallelism. Note that the computation for each D_{ij} can be started once the data is available. In equation (2), the value of D_{ij} depends on the value of three other cells, *i.e.*, $D_{(i-1)(j-1)}$, $D_{(i-1)j}$, and $D_{i(j-1)}$, as shown in Figure 2.

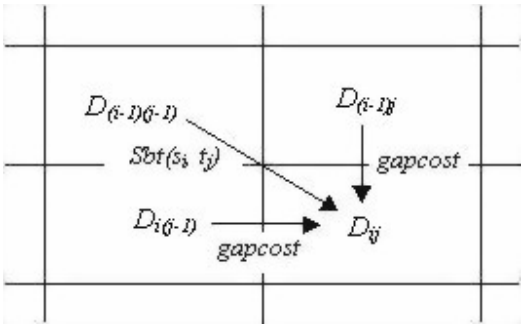

Figure 2: Data dependence in Equation (2)

Because of the data dependence shown above, only the elements on one positive slope diagonal entries in the matrix can be computed in parallel, as shown in Figure 3. The elements on a diagonal row cannot be computed unless the data from the previous diagonal row has been computed. Thus, the parallelism is limited to only one diagonal row at a time. This may seems a disadvantage, which inherited from the dynamic programming technique. However, when a large number of matrices are processed together, we can turn this disadvantage to be a great benefit.

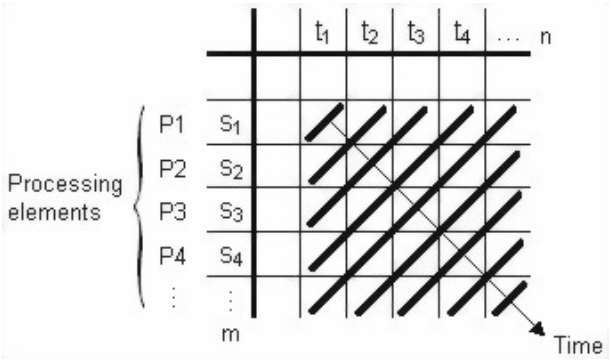

Figure 3: Wavefront propagation. Thick lines show entries which can be computed in parallel.

As shown in Figure 3, a number of processing elements are assigned to process different cells in parallel. In particular, one processing element is assigned per each row to perform ordered computations across the row. Note that the elements on each row have to be computed in order, but not simultaneously. Each processing element performs the computations from the left-most element to the rightmost one. Every time the processing element will detect the availability of the necessary data. As soon as the data is ready, the processing element activates, and then moves on to the next element on the same row.

If the lengths of the two sequences are denoted as *m* and *n,* as shown in Figure 3, then an *m* number of processing elements will be sufficient for the above described parallel processing. The computation for all elements on the first row can be completed in *n* time steps. The computation for the first element on the second row cannot be started until the first element on the first row is completed. Thus, the completion of the second row's elements is on time step *n+1,* etc. The overall computation of the entire matrix thus requires time steps *n+m-1*.

Figure 4 shows the implementation of the processing unit to realize the execution of equation (2). This processing element contains only simple circuits, such as adders and comparators (for finding the maximum).

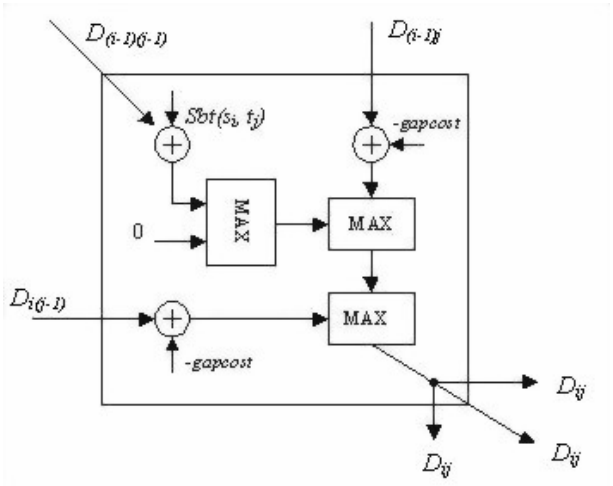

Figure 4: Processing element for implementing the Smith-Waterman algorithm

4.2 Multiple sequence pairs comparisons

In practice, an unknown sequence usually needs to compare with a large set of sequences from the existing database. To conduct the multiple sequence pairs comparisons using the same sets of processing elements, two approaches can be taken, e.g., synchronous and asynchronous. In the synchronous parallelism approach, when a processing element completes a row, it waits for all other processing elements to finish their rows. In this case, all processing elements will be assigned new rows for next sequence pair at the same time, i.e., at time step *n+m-1*, as shown in Figure 5.

Note that in the synchronous approach, the utilization of each processing element is low when *n* and *m* are on the same order of magnitude, *i.e.,* the utilization is *n/(n+m-1)*. For instance, the first processing element will be idle for the last *m-1* time steps. As stated before, since we have many matrices to be filled for different independent sequence comparisons, we can make the processing elements fully utilized by assigning each processing element to several matrices processing. This idea leads to the asynchronous approach.

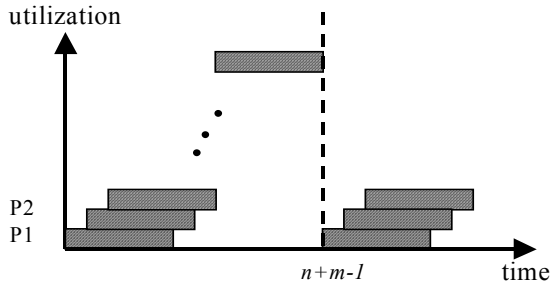

Figure 5: Synchronous approach for multiple sequence pairs comparison. All processing elements have to wait until the last row is finished, then the next sequence pair can be started.

In the asynchronous approach, when a processing element completes a row, it immediately grabs the next available row, which is typically in the next sequence pair. From another prospective, this represents a "pipeline" processing style that across different sequences comparison. Our goal is to achieve a highly cost-effective mechanism for multiple sequence pairs comparisons.

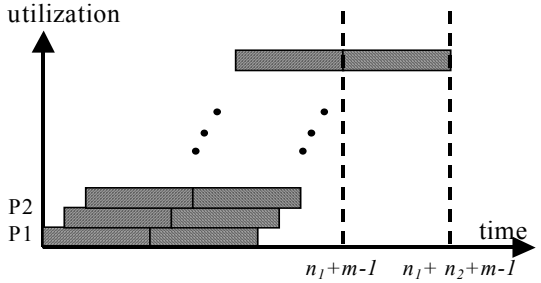

Figure 6: Asynchronous approach for multiple sequence pairs comparison. Upon completion of a row, a processing element immediately grabs the next available row.

Suppose that one wishes to compare an unknown sequence (S) with a large set of sequences from a database (T_1, T_2, …, T_k). Denote the lengths of S, T_1, T_2, …, T_k are m, n_1, n_2, …, n_k respectively.

With the synchronous approach, the first sequence pair can be completely computed in time $m+n_1-1$. Each additional sequence pair requires $m+n_i-1$ time steps, where $i = 2, 3, ..., k$. Comparison of the entire database requires time $(m+n_1-1)+(m+n_2-1)+...+(m+n_k-1)$ or $k \cdot m + \sum_{i=1}^{k} n_i - k$.

With an asynchronous parallel, the first sequence pair requires $m+n_1-1$ time steps to compute, but each additional sequence pair only requires n_i more time steps, where $i = 2, 3, ..., k$. Comparison of the entire database using asynchronous parallel approach thus requires only $(m+n_1-1)+n_2+n_3+ \ ... \ +n_k$ time steps or $m + (\sum_{i=1}^{k} n_i) - 1$. It is clear that the asynchronous parallel approach is $(k-1)\cdot(m-1)$ time steps faster than the synchronous parallel approach. Recall that k represents the size of the existing sequences in the database, which growth exponentially.

5. Conclusion

In this paper, we present a parallel implementation for multiple sequence-pairs comparison using the Smith-Waterman algorithm. This approach reduce the time complexity from *O(mn)* to *O(m+n)* for a single sequence pair comparison, and from *O(mnk)* to *O(m+nk)* for multiple sequence pairs comparison. In today's bioinformatics field, the size of the bio-sequences database, *i.e.*, *k*, grows exponentially. This highlights the urgent needs of cost-effective and efficient comparison mechanisms. The parallel implementation methodology presented here provides the framework of reducing time complexity, while maintaining the same level of sensitivity for this important task.

6. References

[1] T.F Smith and M.S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, 147(1), 195-197, 1981.

[2] W.R. Pearson, "Rapid and sensitive comparison with FASTP and FASTA", Methods Enzymol, 183, pp.63-98, 1990.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment search Tool", J. Mol. Biol., 215, pp.403-410, 1990.

[4] Fa Zhang, Xiang-Zhen Qiao, and Zhi-Yong Liu, "A Parallel Smith-Waterman Algorithm Based on Divide Conquer", *IEEE Computer Society*, 2002.

[5] Deshpande AS, Richards DS, and Pearson WR, "A Platform for Biological Sequence Comparison on Parallel Computers", Comput Appl Biosic, 7(2), 237-47, 1991.

[6] Gilles Brassard and Paul Bratley, "Algorithmics: Theory & Practice", Prentice Hall, April 1988.

[7] Eric V. Denardo, "Dynamic Programming: Models and Applications", Dover Pubns, April 23, 2003.