Fast Sequence Database Search Using Intra-Sequence Parallelized Smith-Waterman Algorithm

Chung-E Wang
Department of Computer Science
California State University, Sacramento
Sacramento, CA 95819-6021
wang@csus.edu

Abstract - In this paper, we describe a fully pipelined VLSI architecture for sequence database search using Smith-Waterman algorithm. The architecture makes use of the principles of parallelism and pipelining to the greatest extent in order to take advantages of both intra-sequence and inter-sequence parallelization and to obtain high speed and throughput. First, we describe a parallel Smith-Waterman algorithm for general SIMD computers. The parallel algorithm has an execution time of $O(m+n)$, where $m$ and $n$ are lengths of the two biological sequences to be aligned. Next, we propose a VLSI implementation of the parallel algorithm. Finally, we incorporate a pipeline architecture in the proposed VLSI circuit and result in a pipeline processor that can do sequence database searches at the speed of $O(m+n+L)$, where $L$ is the number of sequences in the database.

Keywords: Sequence alignment, sequence database search, Smith-Waterman algorithm, parallel algorithm, VLSI circuit, pipelined architecture.

1. Introduction

A sequence alignment is a way of matching two biological sequences to identify regions of similarity that may be a consequence of functional, structural or evolutionary relationships between the two biological sequences. Sequence alignment is a fundamental operation of many bioinformatics applications such as genome assembly, sequence database search, multiple sequence alignment, and short read mapping.

The Smith-Waterman algorithm [1] is the most sensitive but slow algorithm for performing sequence alignment. Here sensitivity refers to the ability to find the optimal alignment. Smith-Waterman algorithm requires $O(m^2n)$ computational steps, where $m$ and $n$ are lengths of the two sequences to be aligned. Smith-Waterman algorithm was later improved by Gotoh [2]. Gotoh’s algorithm can find an optimal alignment of two biological sequences in $O(mn)$ computational steps. It was a great improvement for aligning two sequences. However, it’s not fast enough for sequence database searches. A sequence database search is to compare a query sequence with a database of sequences, and identify database sequences that resemble the query sequence above a certain threshold.

Due to substantial improvements in multiprocessing systems and the rise of multi-core processors, parallel processing became a trend of accelerating Smith-Waterman’s algorithm and sequence database searches. Many enhancements of Smith-Waterman algorithm based on the idea of parallel processing have been presented [3-18]. However, all these enhancements can only speed up Smith-Waterman algorithm by a constant factor. That is, all these enhancements still require an execution time of $O(mn)$ to align two biological sequences.

Besides parallel processing, heuristic methods are other commonly used approaches for speeding up sequence database searches. A heuristic method is a method which is able to produce an acceptable solution to a problem but for which there is no proof that it’s an optimal solution. Heuristic methods are intended to gain computational performance, potentially at the cost of accuracy or precision. Popular alignment search tools such as FASTA [19], BLAST [20] and BLAT [21] are in this category. They did successfully gain some speed. However, the sensitivity is compromised. For sequence database searches, sensitivity refers to the ability to find all database sequences that resemble the query sequence above a threshold.

Without sacrificing any sensitivity, in this paper, we first describe a parallel Smith-Waterman algorithm for general SIMD (Single Instruction stream - Multiple Data stream) computers. This parallel algorithm requires an execution time of $O(m+n)$ to align two biological sequences. Second, we propose a VLSI (Very-Large-Scale Integration)
implementation of the parallel algorithm. Finally, we use the pipeline technique to overlap the execution times of alignment checking of database sequences. The resulting pipeline has a throughput rate of $O(1)$ execution time per database sequence. Consequently, the time complexity of the proposed pipeline processor is $O(m+n+L)$, where $L$ is the number of sequences in the database.

2. The Smith-Waterman Algorithm

The Smith-Waterman algorithm is used to compute the optimal local-alignment score. Let $A = a_1 \ldots a_n$ and $B = b_1 \ldots b_n$ be the two sequences to be aligned. A weight $w(a_i, b_j)$ is defined for every pair of residues $a_i$ and $b_j$. Usually $w(a_i, b_j) \leq 0$ if $a_i \neq b_j$, and $w(a_i, b_j) > 0$ if $a_i = b_j$. The penalties for starting a gap and continuing a gap are defined as $g_{init}$ and $g_{ext}$ respectively. The optimal local alignment score $S$ can be computed by the following recursive relations:

$$E_{i,j} = \max \{ E_{i,j-1} - g_{ext}, H_{i,j-1} - g_{init} \} \quad (1)$$

$$F_{i,j} = \max \{ F_{i+1,j} - g_{ext}, H_{i+1,j} - g_{init} \} \quad (2)$$

$$H_{i,j} = \max \{ 0, E_{i,j}, F_{i,j}, H_{i+1,j-1} + w(a_i, b_j) \} \quad (3)$$

$$S = \max \{ H_{i,j} \} \quad (4)$$

The values of $E_{i,j}$, $F_{i,j}$, and $H_{i,j}$ are 0 when $i<1$ and $j<1$. Smith-Waterman algorithm is a dynamic programming algorithm. A dynamic programming algorithm is an algorithm that stores the results of certain calculations in a data structure, which are later used in subsequent calculations. Smith-Waterman algorithm uses a $m \times n$ matrix, called alignment matrix, to store and compute $H$, $E$, and $F$ values column by column.

3. An Intra-sequence Parallelized Smith-Waterman Algorithm For SIMD Computers

Intra-sequence parallelization means the parallelization is within a single pair of sequences, in contrast to inter-sequence parallelization where the parallelization is carried out across multiple pairs of sequences.

Figure 1 shows the computational dependencies of Smith-Waterman alignment matrix. Most of the existing parallelized Smith-Waterman algorithms in the literature are originated from this dependency structure. Since cells on a bottom-left to top-right diagonal have the same sum of indices, we number those diagonals with their sums of indices. Noticeably cells of a diagonal don’t dependent on cells of the same diagonal and thus can be computed simultaneously. Furthermore, cells of a diagonal only depend on cells of the previous two diagonals. So, the basic idea of the parallel algorithm is to fill the matrix diagonal by diagonal starting from the top-left corner. Moreover cells of a diagonal are filled simultaneously with multiple processors.

![Fig. 1. Computational dependencies in the Smith-Waterman alignment matrix.](image)

The pseudo code of the parallel algorithm is given in Figure 2. Note that in the pseudo code, we use the in-parallel statement to indicate things to be done by processors simultaneously. The in-parallel statement has the following syntax:

In parallel, all processor i, lo<=i<=hi do {
    ...
}

Only processors with processor numbers between lo and hi are activated. Also note that in the pseudo code, $j$ is a variable for processor $i$ only. In other words, different processors have different $j$’s. Furthermore, array $maxH$ is used for processors to keep track of the largest $H$. Since there are $m+n-1$ diagonals, the loop repeats $m+n$ times and thus the parallel algorithm has an execution time of $O(m+n)$.

![Fig. 2. The pseudo code](image)
4. A VLSI Implementation

First we design a small processing element according to the recursive relations (1), (2) and (3). Processing elements will be used to implement cells of Smith-Waterman’s alignment matrix. Thus, each processing element will be numbered with the indices of its corresponding cell in Smith-Waterman’s alignment matrix. As follows, processing element PE\(_{i,j}\) will be used to compute values \(E_{i,j}, F_{i,j}\) and \(H_{i,j}\). As shown in Figure 3, processing element PE\(_{i,j}\) consists of 3 registers \(E_{i,j}, F_{i,j}\) and \(H_{i,j}\) and several simple combinational circuits such as adders and comparators (for finding maximum of two or three values). Since \(H_{i,j}\) depends on \(E_{i,j}\) and \(F_{i,j}\), processing element PE\(_{i,j}\) has two computational states and thus requires two clock signals to complete its computation of values \(E_{i,j}, F_{i,j}\) and \(H_{i,j}\). Since the longest data path in the processing element consists of an adder and a comparator, the clock period, i.e. time between each clock signal, only needs to be set to the time delay caused by an adder and a comparator.

Each processing element PE\(_{i,j}\) has 5 input ports and 3 output ports. Input ports A and B are for inputting residues \(a_i\) and \(b_j\) from the two sequences to be aligned. Input ports \(E_{in}, F_{in}\) and \(H_{in}\) are for inputting corresponding values from cells on which PE\(_{i,j}\) depends. Output ports \(E_{out}, F_{out}\) and \(H_{out}\) are for exporting values to cells depending on PE\(_{i,j}\).

Additionally, in order to keep track of the largest \(h_{i,j}\) value, register \(H_{i,j}\) of processing element PE\(_{i,j}\) is connected to register \(maxH\) as shown in Figures 3.

Next, we map our algorithm into a VLSI circuit. Figure 4 depicts our VLSI circuit at the register level for \(m=4\) and \(n=6\). Processing elements and registers \(maxH\) are connected together according to recursive relations (1), (2), (3) and (4). As shown in Figure 4, processing elements are arranged into levels such that processing elements corresponding to cells of a diagonal are placed on the same level and thus will compute their values at the same time. For the clarity of the logic of the VLSI circuit, in Figure 4, levels are numbered with their corresponding diagonal numbers.

Fig. 3. Processing element PE\(_{i,j}\) and register max\(H_i\)

Fig. 4. The VLSI circuit, \(m=4, n=6\).
5. A Pipeline Architecture for Sequence Database Searches

Since sequence database search is different from sequence alignment, first, we modify our processing element PE<sub>i,j</sub> as shown in Figure 5. Instead of keeping track of largest h<sub>i,j</sub>, the new processing element PE<sub>i,j</sub> will generate a SELECT signal when its H<sub>i,j</sub> value is above a threshold.

![Modified processing element PE<sub>i,j</sub>]

**Fig. 5.** Modified processing element PE<sub>i,j</sub>.

To speedup sequence database searches, a pipelined architecture is incorporated in the VLSI circuit. Pipelining is a natural concept for increasing the throughput of a system when processing a stream of data, even though pipelining cannot speed up the process of a single datum. The space-time diagram in Figure 6 reveals the advantages of the pipelined architecture in processing database sequences. The diagram shows the succession of the levels in the pipeline with respect to time. From the diagram one can observe how independent sequences are processed concurrently in the pipeline.

![Pipeline space-time diagram]

**Fig. 6.** Pipeline space-time diagram

Since there are m+n-1 levels in the VLSI circuit, it's very natural to organize the entire architecture as a linear pipeline with m+n-1 stages. To do so, as shown in Figure 7, we add m+n-1 registers to the VLSI circuit to hold database sequences one for each stage, i.e. level. Additionally, each database sequence register has a S flag which will be used to indicate whether the database sequence is selected or not. As shown in Figure 7, processing elements' select signals are connected to S flags of corresponding

![The single-chip pipeline processor, m=4, n=6.]

**Fig. 7.** The single-chip pipeline processor, m=4, n=6.
shown for generation of intra-sequence alignments. Since a processing element $PE_{i,j}$ needs two clock signals to compute its values, our pipeline takes 2 clock signals to move from one pipeline stage to another.

Apparently as soon as the first database sequence completes its alignment checking, every one stage there is a database sequence completes its alignment checking. Consequently, the pipeline processor has a throughput rate of one database sequence per two clock signals. In other words, the pipeline has a time complexity of $O(1)$ time per database sequence. Moreover, since the first sequence takes $O(m+n)$ time to completes its alignment checking, the total time complexity of the pipeline processor is $O(m+n+L)$, where $L$ is the number of sequences in the database.

For most bioinformatics applications, $m$ and $n$ are in thousands and thus $mn$ is in millions. Since there are $m \times n$ processing elements in our VLSI circuit, the pipeline processor requires millions of combinational circuits and registers. As of today, a VLSI microchip can have billions of transistors. Since a register or a simple combinational circuit such as adder or comparator doesn’t need thousands of transistors, billions of transistors should be enough for millions of our processing elements. As a result, it’s possible to implement the pipeline processor in a single VLSI microchip.

For some applications such as genome assembly, the length of the query sequence may be more than thousands and thus requires multiple microchips to implement the pipeline processor. Figure 8 demonstrates the scalability of the pipeline processor. In circuit design, scalability refers to the ability to be expanded to cope with increased use. As shown in Figure 8, we decompose the pipeline circuit into three sub-circuits each of which can be implemented in a microchip. Type 1 chip is for the head of a pipeline. Type 3 chip is for the tail of a pipeline. Type 2 chip is for the middle part of a pipeline. To handle long query sequences, we simply use more type 2 chips in the middle to lengthen the pipeline.

6. Conclusion

In this paper, we have described a $O(m+n)$ time intra-sequence parallelized Smith-Waterman algorithm for general SIMD computers, where $m$ and $n$ are lengths of the two sequences to be aligned. We have shown a VLSI implementation of the parallel algorithm. We have also shown that by incorporating a pipelined architecture into the VLSI circuit, we can speed-up sequence database searches without sacrificing the sensitivity. The resulting pipeline processor can do sequence database searches at the speed of $O(m+n+L)$, where $L$ is the number of sequences in the database. Moreover, we have demonstrated the scalability of the pipeline processor.

7. References

[14] Y Liu, DL Maskell, B Schmidt, CUDA++:


**Fig. 8.** A multiple-chip pipeline processor, m=3, n=6.