Brigham Young University
BYU ScholarsArchive

Faculty Publications

2009-01-01

Hardware Accelerated Sequence Alignment with Traceback

Scott Lloyd

Quinn O. Snell
snell@cs.byu.edu

Follow this and additional works at: https://scholarsarchive.byu.edu/facpub

Part of the Computer Sciences Commons

Original Publication Citation

Hardware Accelerated Sequence Alignment with Traceback, Scott Lloyd and Quinn O. Snell, International Journal of Reconfigurable Computing, vol. 29, Article ID 762362, 1 pages, 29. doi:1.1155/29/762362.

BYU ScholarsArchive Citation

Lloyd, Scott and Snell, Quinn O., "Hardware Accelerated Sequence Alignment with Traceback" (2009). *Faculty Publications*. 868.
https://scholarsarchive.byu.edu/facpub/868

This Peer-Reviewed Article is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Faculty Publications by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 762362, 10 pages
doi:10.1155/2009/762362

Research Article

Hardware Accelerated Sequence Alignment with Traceback

Scott Lloyd and Quinn O. Snell

Department of Computer Science, Brigham Young University, Provo, UT 84602, USA

Correspondence should be addressed to Scott Lloyd, gscott@ieee.org

Received 15 March 2009; Revised 4 August 2009; Accepted 13 October 2009

Recommended by Cesar Torres

Biological sequence alignment is an essential tool used in molecular biology and biomedical applications. The growing volume of genetic data and the complexity of sequence alignment present a challenge in obtaining alignment results in a timely manner. Known methods to accelerate alignment on reconfigurable hardware only address sequence comparison, limit the sequence length, or exhibit memory and I/O bottlenecks. A space-efficient, global sequence alignment algorithm and architecture is presented that accelerates the forward scan and traceback in hardware without memory and I/O limitations. With 256 processing elements in FPGA technology, a performance gain over 300 times that of a desktop computer is demonstrated on sequence lengths of 16000. For greater performance, the architecture is scalable to more processing elements.

Copyright © 2009 S. Lloyd and Q. O. Snell. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Searching and comparing biological sequences in genomic databases are essential processes in molecular biology. The collection of genetic sequence data is increasing exponentially each year and consists mostly of nucleotide (DNA/RNA) and amino acid (protein) symbols. Approximately 3 billion nucleotide pairs comprise the human genome alone. Given the large volume of data, sequence comparison applications require efficient computing methods to produce timely results.

Biologists and other researchers use sequence alignment as a fundamental comparison method to find common patterns between sequences, predict protein structure, identify important genetic regions, and facilitate drug design. For example, sequence alignment is used to derive flu vaccines [1] and by the nation's BioWatch [2] program in identifying DNA signatures of pathogens. Sequence alignment consists of matching characters between two or more sequences and positioning them together in a column. Gaps may be inserted in regions where matches do not occur to reflect an insertion or deletion evolutionary event. A count of the matching characters results in a measure of similarity between the sequences. Pairwise alignment involves two sequences (see Figure 1) and multiple alignment considers three or more sequences. Finding the optimal multiple sequence alignment is NP-hard in complexity. As a first step, multiple alignment algorithms [3, 4] often compute a pairwise alignment between all the sequences.

Global and local pairwise alignments are the two most common alignment problems. Global alignment [5] considers both sequences from end to end and finds the best overall alignment. Local alignment [6] identifies the sections with greatest similarity and only aligns the subsequences. Both alignment problems are typically solved with dynamic programming (DP), which fills a two-dimensional matrix with score or distance values in a forward scan from upper left to lower right, followed by a traceback procedure. Traceback occurs from a designated lower right position following a path to upper left, thereby determining the best alignment.

The computational cost for an optimal sequence alignment increases exponentially with the length of each sequence and with the number of sequences. This complexity poses a challenge for sequence alignment programs to return results within a reasonable time period as biologists compare greater numbers of sequences. Using current methods, an alignment program may run for days or even weeks depending on the number of sequences and their length.

Unlike most acceleration methods that focus on sequence comparison, this research describes and evaluates a space-efficient, global sequence alignment algorithm and

- - TTCT - - T - - TAGATTC
CCTTCTACTGCTA - CTTC

Figure 1: Example pairwise alignment.

architecture that includes traceback for implementation on reconfigurable hardware. Given a pair of sequences, the accelerator returns a list of edit operations constituting the optimal alignment. A library of accelerator functions is easily incorporated into multiple sequence alignment programs that run on platforms equipped with reconfigurable hardware.

2. Related Work

Most efforts to accelerate biosequence applications with hardware have focused on database searches. Ramdas and Egan [7] compare several of these architectures in their survey. Given a query sequence, an entire genetic database is scanned looking for other sequences that are similar. Searching a genetic database for matches with a biosequence is similar in nature to a search of the web that returns "hits" sorted by relevance. Accelerating a database search is a simpler problem than alignment. Only the score for the comparison is computed by hardware in the forward scan; whereas alignment requires traceback in addition to the forward scan. The sequence comparison problem can be mapped to a linear systolic array of processing elements (PEs) requiring $O(\min(m, n))$ space, where m and n are the lengths of the sequences. However, global alignment necessitates extra storage for traceback pointers and a traceback procedure, which are not addressed by sequence comparison solutions.

Traceback support in hardware has the most benefit when the traceback path spans a significant portion of the DP matrix. Global alignment applications realize the greatest performance gain because the traceback path extends across the entire DP matrix; whereas local alignment applications with a shorter path show less benefit. After a forward scan in hardware, any alignment in software must recompute the DP matrix and traceback pointers for the section of interest before determining an optimal traceback path. For instance, accelerated database search applications may compute an alignment in software only between high-scoring matches and the query sequence after the comparison phase. These search applications usually run in acceptable time with relatively short query sequences; however, comparative genomic applications commonly align long sequences at greater computational cost and stand to benefit from accelerated alignment. Examples include whole genome alignment [8], whole genome phylogeny [9], and computation of pathogen detection signatures [10].

The predominant, nonparallel algorithms for global sequence alignment are described by Gotoh [11] and Myers and Miller [12]. Both algorithms execute in $O(mn)$ time. The algorithm presented by Gotoh requires $O(mn)$ space, while the algorithm of Myers-Miller needs only $O(\log m + n)$ space, but it incurs a factor of 2 time penalty. Most of the space is used to hold values of the DP matrix and the traceback pointers. Saving all traceback pointers in an array requires only one forward scan through the DP matrix followed by one traceback pass. Otherwise, multiple passes through the DP matrix are required if not saving all the traceback pointers. The downside of saving all the traceback pointers is the $O(mn)$ space requirement, which can be significant for longer sequence lengths or prohibitive when limited by FPGA memory.

A few efforts propose hardware methods for accelerating pairwise alignment and traceback. The work presented by Hoang and Lopresti [13] describes an FPGA architecture which consists of a linear systolic array of PEs that output traceback data. However, the type of sequences is limited to only DNA and the sequence length is limited by the number of PEs on the accelerator (a couple of hundred nucleotides). The works by Jacobi et al. [14] and VanCourt and Herbordt [15] suggest accelerated traceback methods, but with few details. The sequence length accommodated by their accelerators is also limited by the number of PEs on the accelerator like the one described by Hoang. Another limitation of the Hoang and VanCourt methods is that traceback cannot be overlapped with another forward scan since the systolic array is used for both scan and traceback.

The methods presented by Yamaguchi et al. [16] and Moritz et al. [17] allow longer sequences by partitioning the sequences through the pipeline of PEs. Nevertheless, the traceback data must be saved to external memory, since the size of the data exceeds the amount of available internal FPGA memory. Hence, the traceback performance of both methods is limited by the FPGA bandwidth to external memory. The design described by Benkrid et al. [18] also partitions sequences, but the size of FPGA memory ultimately limits the length of sequences that are aligned with hardware acceleration. Operating at 100 MHz, a systolic array with 256 PEs requires at least 6.4 GB/s of memory bandwidth to store 2-bit traceback data from each PE. As PE densities and clock frequencies increase, the external memory bandwidth is easily exceeded. Internal FPGA memory has sufficient bandwidth, but even modest sequence lengths of 16 K require 64 MB of traceback store, which far exceeds current FPGA internal memory capacities.

The global alignment algorithm presented in this paper overcomes the memory size and bandwidth limitations of FPGA accelerators and does not limit the sequence length by the number of PEs. Long sequences of DNA and protein are accommodated by the algorithm through a space-efficient traceback procedure that is accelerated in hardware. Traceback may occur in parallel with the next forward scan since it is implemented in a separate process from the systolic array.

3. Algorithm

The general algorithm is described first followed by the FPGA architecture in the next section. The algorithm is based on dynamic programming (DP), but partitions the problem into slices for the FPGA hardware. A description of the general sequence alignment problem is also found in [5, 11].

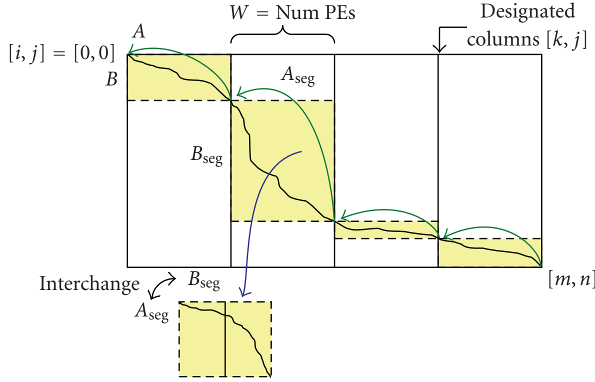

Figure 2: Forward scan and traceback.

Given a pair a *sequences* $A = a_1a_2 \dots a_m$ and $B = b_1b_2 \dots b_n$ of *length* $|A| = m$ and $|B| = n$ from the finite alphabet Σ, a *sequence alignment* is obtained by inserting *gap characters* "-" into A and B. The aligned sequences A' and B' from the extended alphabet $\Sigma' = \Sigma \cup \{\text{"-"}\}$ are of equal length such that $|A'| = |B'|$. Let the function $s : \Sigma \times \Sigma \to \mathbb{Z}$ determine the similarity of symbol a_i with b_j, and let the constant α represent the cost of inserting/deleting a gap. Let H denote the DP matrix and the element $H[i, j]$ the similarity score of sequences $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$. An optimal alignment is obtained by maximizing the score in each element of H. The values of H are determined by the following recurrence relations for $1 \le i \le m$ and $1 \le j \le n$:

$$\begin{aligned} H[0,0] &= 0, \\ H[i,0] &= H[i-1,0] + \alpha, \\ H[0,j] &= H[0,j-1] + \alpha, \\ H[i,j] &= \max\begin{cases} H[i-1,j-1] + s(a_i, b_j), \\ H[i-1,j] + \alpha, \\ H[i,j-1] + \alpha. \end{cases} \end{aligned} \tag{1}$$

The matrix fill occurs in a scan from upper left to lower right because of dependencies from neighboring elements. During the forward scan, a pointer $p \in \{\text{DIAG, ABOVE, LEFT}\}$ indicates the current selection of the max function in (1). Given a tie, fixed priority resolves the selection. The value of p is saved to the traceback matrix T, thus $T[i, j] = p$. Following the forward scan, traceback proceeds from $T[m, n]$ to $T[0, 0]$, thereby determining the best alignment. The result is a list of edit operations $e \in \{\text{SUBSTITUTE, INSERT, DELETE}\}$.

The scan algorithm presented here builds upon the space-saving concepts described by Edmiston et al. [19], and the divide-and-conquer scheme of Guan and Uberbacher [20]. Since sequence lengths are often longer than the number of PEs available in a systolic array, the problem is often partitioned [21]. The forward scan consists of two fundamental scan procedures ScanPartial and ScanFull. The Partial and Full descriptors refer to the amount of traceback data saved by the procedures. ScanPartial partitions the DP matrix H into slices of width W. The slices are processed iteratively. The result of processing each slice is a column of traceback pointers $R[k, j]$ that refer to a row in a prior slice (see Figure 2). The designated columns k are given by $k \in \{c \mid c \bmod W = 0 \vee c = m\}$. The row pointers form a partial traceback path through H that link only the right-most columns of each slice. Given that p indicates the heritage of element $H[i, j]$, the following recurrences for $1 \le i \le m$ and $1 \le j \le n$ determine R

If $i \bmod W = 1$, then

$$R[i,j] = \begin{cases} j-1 & \text{if } p = \text{DIAG}, \\ j & \text{if } p = \text{LEFT}, \\ R[i,j-1] & \text{if } p = \text{ABOVE}, \end{cases}$$

else

$$R[i,j] = \begin{cases} R[i-1,j-1] & \text{if } p = \text{DIAG}, \\ R[i-1,j] & \text{if } p = \text{LEFT}, \\ R[i,j-1] & \text{if } p = \text{ABOVE}. \end{cases} \tag{2}$$

Only the designated columns of R are actually stored, which correspond to the right-most columns of a slice. The values for the other columns are retained temporarily with a vector variable that follows the wavefront of the scan. In contrast, the ScanFull procedure does not partition the DP matrix and produces a full matrix T of traceback pointers that refer to adjacent elements of H.

The TracePartial procedure differs from TraceFull in that the partial set of traceback pointers from R are followed instead of the full set from T. The row pointers, from $R[m, n]$ to $R[0, 0]$ in designated columns, identify waypoints on the optimal path through the DP matrix. Since the row pointer in $R[k, j]$ refers to a row in a prior slice, a block between the columns is identified, along with corresponding segments of A and B. The segments of A and B are passed to ScanFull and TraceFull to determine the full path from $[k, j]$ back to $[k_{\text{prev}}, R[k, j]]$. The alignment results from each block are concatenated and thereby form a complete path from $[m, n]$ to $[0, 0]$.

Since the vertical height of a block (the length of a B segment) is unbounded, the traceback space available to the Full procedures may be exceeded. To avoid this case, a vertical threshold Y is defined such that if exceeded, the Partial procedures are called instead, with the segments of A and B interchanged in the calls. Algorithm 1 shows the procedure that is central to bounding the memory required for traceback. TracePartial is called recursively a maximum of once. Any segments passed to the Full procedures will not exceed W and Y in length because of the partitioning done by ScanPartial. In the worst case, the length of sequence A is bounded by the first call to ScanPartial and the length of B is bounded by the second call.

```
procedure TracePartial(A, B, m, n, R, E)
{
  x2 ← m, y2 ← n
  while (x2 > 1) do
    x1 ← ⌊(x2 − 1)/W⌋ · W + 1
    y1 ← (x1 > 1 ∧ y2 ≥ 1) ? R[x2, y2] + 1 : 1
    xlen ← x2 − x1 + 1, ylen ← y2 − y1 + 1
    if (ylen = 0) then
      Add xlen DELETE operations to E′
    else if (ylen ≤ Y) then
      ScanFull(A_x1, B_y1, xlen, ylen, T)
      TraceFull(A_x1, B_y1, xlen, ylen, T, E′)
    else // interchange A and B
      ScanPartial(B_y1, A_x1, ylen, xlen, R′)
      TracePartial(B_y1, A_x1, ylen, xlen, R′, E′)
      ∀e ∈ E′: replace DELETE ⇔ INSERT
    end if
    E ← E ∪ E′
    x2 ← x1 − 1, y2 ← y1 − 1
  end while
}
```

Algorithm 1: Procedure for TracePartial.

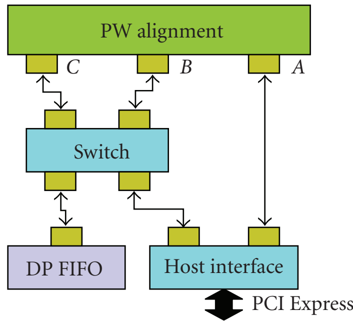

Figure 3: System architecture.

4. Architecture

The global alignment accelerator is implemented using Qnet [22], an open-source packet-switched network architecture similar to DIMEtalk [23]. Qnet components interconnect the host and other FPGA accelerator modules in the system. The architecture facilitates system design with reusable modules that encapsulate sharable devices or resources. Qnet encourages parallelism by offering concurrent, high-performance data paths between modules. Figure 3 shows the alignment system constructed with Qnet modules and components. A few specifics of Qnet are given before describing the alignment accelerator module and system operation.

4.1. Qnet Components. The basic network components consist of a switch, Qports, and Qlinks. As the central figure in the network, the switch provides a path for communicating packets to other modules. Qports are the interface between modules and the network, and are the addressable endpoints of communication. Qports are connected by Qlinks, which

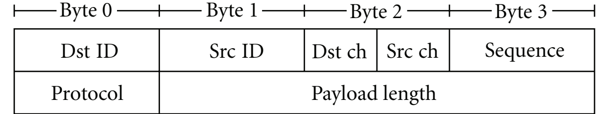

Figure 4: Qpacket header.

consist of paired, unidirectional, point-to-point signaling channels that are each 32-bits wide in this system, but may be implemented with other bit widths. Each Qport has word-based flow control that will apply back-pressure on a link, delaying communication until the port is ready to receive. Hence, packets are not arbitrarily discarded, and the requirement to buffer an entire packet at the input of a module is removed while still maintaining performance. Qnet communication performance has been shown to be very near the theoretical max bandwidth between modules on the FPGA while also maintaining latencies very near theoretical minimums.

Qnet reliably transfers data packets between endpoints through a simple protocol that requires minimal FPGA resources. Packets consist of a small header (see Figure 4) and a payload of variable size. The header specifies the source and destination endpoints with unique port identifiers and also indicates the payload length. When a packet header enters the switch, the output port is determined from the destination endpoint and remains the same for all following words of the packet. With a cut-through packet forwarding method, the full packet is not buffered in the switch. Packets that enter the switch simultaneously with different destinations pass through concurrently. This architecture allows parallel data transfer on all ports of an accelerator module.

4.2. System Modules. Host Interface. The host computer communicates with the FPGA accelerator through the PCI Express [24] module, which contains DMA engines and translates PCI packets into Qnet packets. Two ports on this module allow both sequences to be sent in parallel to the accelerator.

DP Matrix FIFO. If the length of sequence A is longer than the number of PEs in the accelerator, the DP matrix H must be processed in slices of width W = (num. PEs) as described in Section 3. After processing a slice, the right column of DP matrix values will exit the pipeline of PEs. These H values are sent in a packet to the DP matrix FIFO and retained for processing the next slice through the pipeline. Any packet sent to the DP matrix FIFO will be returned to the originating Qport, as indicated by the packet header, thus cycling the pipeline output to the input. The FIFO may be implemented with any memory technology of sufficient bandwidth and size to handle the stream of data from the PE pipeline. Since only one H value exits the pipeline each clock cycle, the bandwidth requirement is not excessive.

Pairwise Alignment Module. The compute intensive portions of the alignment algorithm are performed by the pairwise

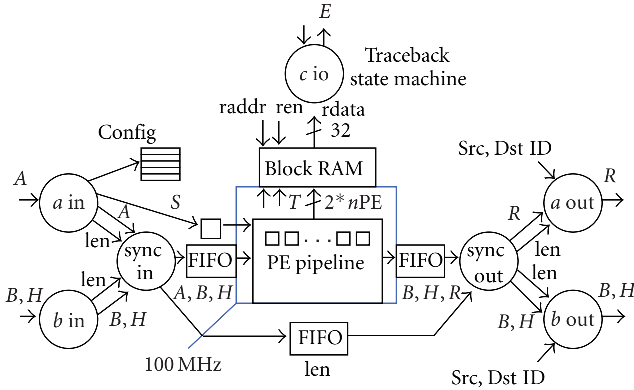

Figure 5: Pairwise alignment module.

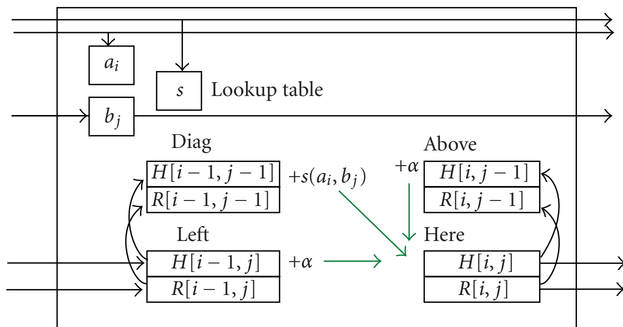

Figure 6: Processing element architecture.

alignment module, which contains the pipeline of PEs. This module has 3 Qports through which the sequences are provided and results are returned (see Figure 3). In parallel, Sequence A is input on port A and sequence B is input on port B, while the traceback results are returned on port C.

Figure 5 shows the internal architecture of the alignment module. The front-end of the pipeline synchronizes the A and B streams of symbols, and the back-end sends the partial traceback results R out on port A and the H values on port B. The symbols of sequence B that flow through the pipeline are merged with the H values on output, since they will also be needed in processing additional slices. Merged B and H values that exit the pipeline are sent in a packet to the DP matrix FIFO. As sequence A is fed into the pipeline, merged B and H values from the end of the pipeline flow from the alignment module through the DP matrix FIFO and back into the front-end of the pipeline at port B. This cycle occurs for each slice of the scan, except for the last.

Most systems commonly load a segment of A into the pipeline and then shift in B; whereas this system enters A and B in parallel [25]. Sequence B is shifted in as usual, but A is bussed to each PE and latched when the first symbol of B reaches a PE in the pipeline (see Figure 6). The recurrence equations described in Section 3 are calculated by the PEs each time a pair of symbols enter the pipeline. As a forward scan proceeds from upper left to lower right, the pipeline of PEs operates in parallel along an antidiagonal wavefront through the DP matrix. Figure 7 shows the progression of symbols in the pipeline and shows the mapping of PEs to DP matrix cells over several cycles.

Both of the forward Scan procedures are implemented by the pipeline of PEs. ScanPartial enables the R (partial row pointer) output, while ScanFull enables the T (full traceback pointer) output. Configuration bits in the packet header of sequence A determine which pointer type is enabled. For each slice processed by ScanPartial, a column of R is returned to the host in a packet. ScanFull will only process one slice, while saving the full traceback data in FPGA block RAM, which has the bandwidth to store pointers from every PE in parallel. The vertical threshold Y, as described in Section 3, is determined by the depth of FPGA block RAM allocated to full traceback.

A state machine implements the TraceFull procedure that follows the pointers saved in block RAM by ScanFull. To initiate a full traceback, a request packet is sent to Port C of the pairwise alignment module from the host. The results, a list of edit operations $e \in E$, are returned to the host from Port C. TracePartial is implemented in software on the host, but calls the Full procedures for most of the work (see Algorithm 1).

Access to traceback pointers $T[i, j]$ in block RAM requires a skewed addressing scheme because of the storage method used in the forward scan. Storing a diagonal wavefront of pointers as a row in block RAM skews the traceback matrix T in memory (see Figure 8). A full traceback begins with a request packet that contains the cell address of $T[1, 1]$ and the lengths of sequences A and B. The address of $T[1, 1]$ is saved at the start of a full forward scan and will always be the lowest address in a row (leftmost). From the address of $T[1, 1]$ and the width W of block RAM in cells, the address of $T[m, n]$ is calculated

$$\begin{aligned} m' &= m - 1, \\ n' &= n - 1, \\ \text{addr}_{T[m,n]} &= \text{addr}_{T[1,1]} + W(m' + n') + m'. \end{aligned} \tag{3}$$

Traceback proceeds from $T[m, n]$ to $T[0, 0]$ following the pointer in each accessed cell. Given a traceback pointer p from the current cell, the following equation determines the address of the next cell in block RAM

$$\text{addr} = \begin{cases} \text{addr} - (2W + 1) & \text{if } p = \text{DIAG}, \\ \text{addr} - (W + 1) & \text{if } p = \text{LEFT}, \\ \text{addr} - W & \text{if } p = \text{ABOVE}. \end{cases} \tag{4}$$

Since block RAM is dual-ported, traceback reads can occur while the next forward scan concurrently saves pointers in another portion of the traceback memory. Address calculations into block RAM wrap around when the range is exceeded.

4.3. System Parameters. Most system parameters are implemented with VHDL generics. For example, symbol width, number of PEs, traceback memory depth, and various register sizes are all specified at a high level in the module

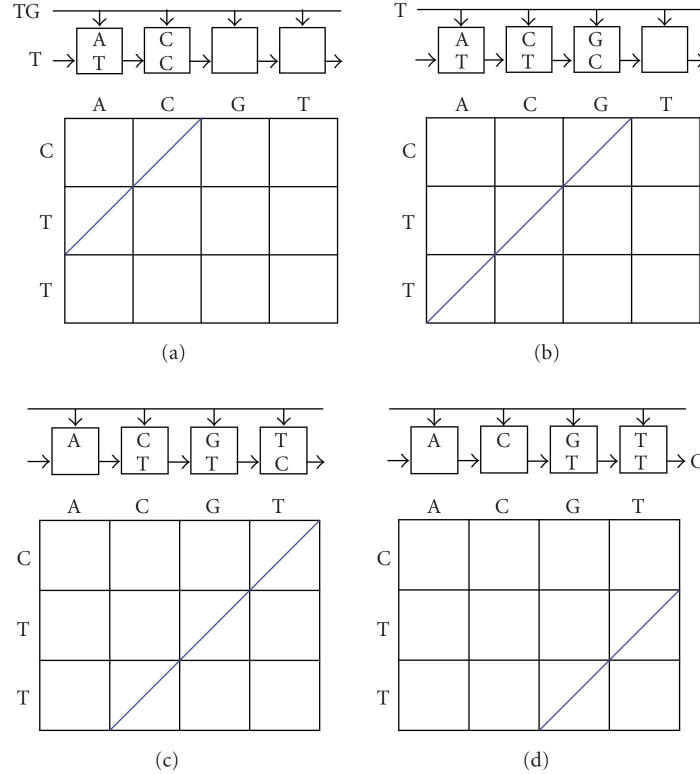

Figure 7: Symbol flow and the corresponding DP matrix wavefront for sequential cycles of the PE pipeline.

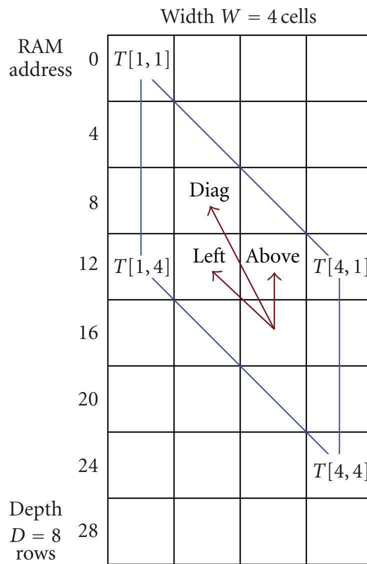

Figure 8: The traceback matrix T is skewed in memory. The pointers show how to address neighboring cells during traceback in the skewed matrix.

hierarchy and passed as generics to lower modules. This allows different configurations of the accelerator with minimal changes to the source. Protein sequences require 5-bit symbols and DNA sequences require at least 2-bit symbols. Mega-length sequences may be handled by the architecture and algorithm by setting system constants and rebuilding a system. The number of PEs is scalable to match the target hardware resources.

Several system parameters affect the maximum sequence length $L_{\max}$ that can be processed by the accelerator. As mentioned previously, the DP matrix FIFO must be deep enough to hold the merged B symbols and H values that come from the end of the pipeline. The FIFO length limit is determined by $L_F = N_{\text{FIFO}}/N_{BH}$, where N_{BH} denotes the number of bytes for a single B-H pair and N_{FIFO} denotes the DP matrix FIFO size in bytes. Also, the substitution and gap costs combined with the H register size affect the maximum sequence length. Each stage of the pipeline increments an H value by the gap cost α or the result of the similarity function $s(a_i, b_j)$. To avoid H register overflow, the H length limit is $L_H = (2^{N_H-1} - 1)/I_{\max}$, where N_H denotes the number of bits in H registers, and $I_{\max}$ denotes the maximum absolute value of the gap cost α or the similarity function s. In conjunction with the other parameters, the R register size affects the maximum sequence length. A register for R must hold an index into sequence B without overflow. Given N_R, the number of bits in R registers, the R length limit is $L_R = 2^{N_R} - 1$. From the contributing length limits, the maximum sequence length is determined by $L_{\max} = \min(L_F, L_H, L_R)$.

5. Timing Model

A timing model is presented for the sequence alignment algorithm and architecture described in Sections 3 and 4.

First, constants for the system are defined with the values in parenthesis being specific to the evaluation system:

$$
\begin{aligned}
W &: \text{number of PEs } (256),\\
Y &: \text{threshold for length of sequence } B \ (768),\\
C_{\text{pad}} &: \text{cycles to pad pipeline } (8),\\
t_s &: \text{communication startup } (1.5\,\mu\text{s}),\\
t_h &: \text{host overhead } (3\,\mu\text{s}),\\
t_{\text{clk1}} &: \text{period of clock 1 } \left(\frac{1}{100\,\text{MHz}}\right),\\
t_{\text{clk2}} &: \text{period of clock 2 } \left(\frac{1}{150\,\text{MHz}}\right).
\end{aligned}
\tag{5}
$$

Timing varies as a function of the following variables:

$$
\begin{aligned}
l &= |A'| = |B'|, \text{ aligned length},\\
m &= |A|, \text{ length of sequence } A,\\
n &= |B|, \text{ length of sequence } B,\\
N_{\text{slice}} &= \lceil m/W \rceil, \text{ number of slices}.
\end{aligned}
\tag{6}
$$

The time for processing a slice is determined by the length of B or the length of the pipeline plus padding, whichever is greater. Flush time depends on how much of sequence B is left in the pipeline after processing a slice and is calculated from the length of B minus padding (zero limited) or the length of the pipeline, whichever is less

$$
\begin{aligned}
t_{\text{slice}} &= t_{\text{clk1}}\left[\max\left(n, W + C_{\text{pad}}\right) + 1\right],\\
t_{\text{flush}} &= t_{\text{clk1}} \min\left(W, \max\left(0, n - C_{\text{pad}}\right)\right).
\end{aligned}
\tag{7}
$$

Based on the previous definitions, execution times for the Scan and Trace procedures are

$$
\begin{aligned}
t_{\text{scan}F} &= t_{\text{slice}} + t_{\text{flush}} + 4t_s,\\
t_{\text{trace}F} &= t_{\text{clk2}}(2l + 4) + 2t_s,\\
t_{\text{scan}P} &= N_{\text{slice}}(t_{\text{slice}} + t_s) + t_{\text{flush}} + 4t_s,\\
t_{\text{trace}P} &= N_{\text{slice}}(t_{\text{scan}F} + t_{\text{trace}F}).
\end{aligned}
\tag{8}
$$

Finally, the time to perform a global sequence alignment is given by:

$$
t_{\text{align}} =
\begin{cases}
t_{\text{scan}F} + t_{\text{trace}F} + t_h & \text{if } m \le W \wedge n \le Y,\\
t_{\text{scan}P} + t_{\text{trace}P} + t_h, & \text{else}.
\end{cases}
\tag{9}
$$

This analytical model matches experimental results and predicts the scalability and performance of the architecture under various system configurations.

6. Experimental Setup

Application. Three global alignment implementations are tested in the evaluation: (1) as a baseline, a software-only version of the algorithm presented in this paper; (2) a version accelerated by the FPGA; and (3) an implementation of the Myers-Miller global alignment algorithm for an additional point of reference. The host computer is used to evaluate the software only versions of the algorithms. Seq-Gen [26] produced varying lengths of test sequences ranging from 128 to 16383 symbols for the evaluation. The applications use a gap cost of −2, a substitution score of 1, and a match score of 2.

Host. The host platform consists of a desktop computer with a 2.4 GHz Intel Core2 Duo processor running Fedora 6 Linux as the operating system. All benchmark applications execute in a single thread and are compiled with gcc using −O3 optimization. For accurate timing, the processor's performance counters are used.

Accelerator. An 8-lane PCI Express add-in card with a Xilinx Virtex-4 FX100 FPGA provides the hardware acceleration. To conserve FPGA resources, only 4 of the 8 PCI Express lanes are used in the experimental system. All of the components are implemented in VHDL. As shown in Figure 3, a 4-port switch connects the three FPGA modules using 32-bit Qlinks that run at 150 MHz. For simplicity and minimal latency, the switch is implemented with a fixed address table and a fixed port priority resolution scheme. The DP matrix FIFO uses 64 KB of FPGA block RAM, which is enough to hold 16 K entries of B symbols and H values. Driven by a 100 MHz clock, the pipeline consists of 256 PEs placed in a tiled pattern. DNA and protein sequences are accommodated with 5-bit symbol values. An 8-bit look-up table that requires one block RAM per PE implements the similarity function $s(a_i, b_j)$. Each PE outputs a 2-bit traceback pointer p that is stored in traceback memory, which is instantiated in 64 KB of block RAM with a width of 512 bits and a depth of 1024. The traceback memory depth determines the Y threshold. Within the system, DP matrix values H and row pointer values R both require 16-bits.

Through the use of constraints and floor planning, 90% slice utilization is achieved. First, an area shape and size constraint for one PE is determined, in this case, by repeated place and route trials. Then, given this shape and size, a simple (75 line) Perl script tiles the PEs in a programed pattern by generating area constraints for each PE. Keep-out areas are also given to the Perl script. The text output from the Perl script is pasted into the user constraints file for use by the place and route tools along with the other constraints. Only slice resources are constrained for the PEs, since the block RAM needed for each PE may not reside within the area constraint. To meet timing, the first and last PEs of the pipeline are kept closer to the Qport interfaces of the switch and alignment module, which is shown in Figure 9 along with the tiling pattern. The traceback block RAMs are constrained to a centrally located area of the FPGA to minimize path lengths from distant PEs. For proximity to the traceback memory, the traceback state machine is also centrally located. Table 1 shows the relative resource usage of the various components.

Table 1: Resource usage.

Component	Slices	FPGA percentage
PCI Express	6175	14.6%
Host interface	1221	2.9%
4-port switch	448	1.1%
Traceback	283	0.7%
DP FIFO	192	0.5%
PE (one)	111	0.3%

Table 2: Speedup between implementations.

Sequence length	t_{FPGA} μs	$\frac{t_{\text{Myers}}}{t_{\text{FPGA}}}$	$\frac{t_{\text{Host}}}{t_{\text{FPGA}}}$
511	64	131	107
1023	128	171	124
2047	327	264	181
4095	969	357	236
16383	11696	471	304

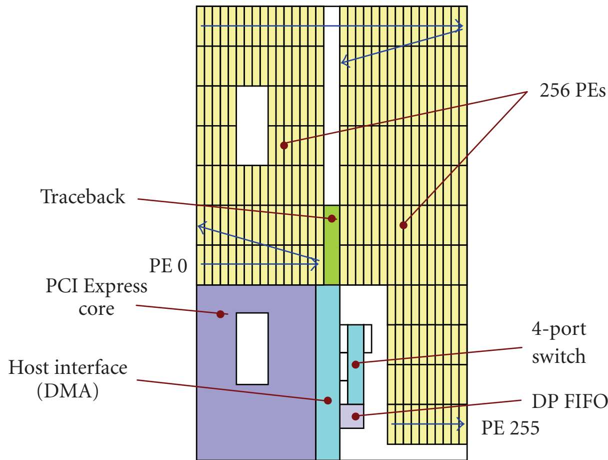

Figure 9: FPGA floorplan.

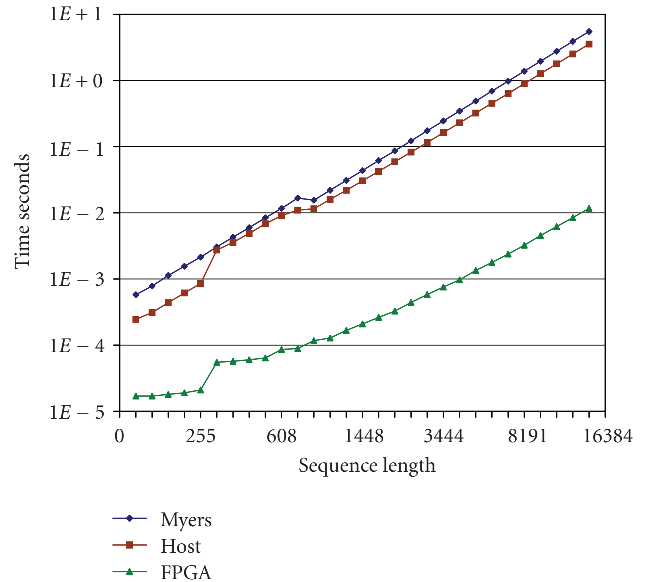

Figure 10: Global alignment execution time.

7. Results

Figure 10 shows the performance of the three global sequence alignment implementations with varying lengths of sequences and Table 2 compares the speedup between the implementations. The host-only version averages a speedup of 1.6 over the Myers-Miller implementation and the accelerated version achieves a max speedup of 304 over the host version. During the forward scan, the accelerator reaches a peak dynamic programming rate of 25.6×10^9 cell updates/s (CUPS). Traceback occurs at a peak rate of 75×10^6 pointers/s. Figure 11 shows the actual performance compared with the timing model from Section 5. For longer sequences, the actual performance is near the theoretical peak. The timing model suggests a high degree of scalability for the presented algorithm and architecture. For example, performance predicted by the model gives a speedup of 580 with 512 PEs operating at 100 MHz on a larger FPGA.

Supported by the low communication overhead of Qnet, sequences of length 10 or greater are aligned faster on the accelerator. Sending a single packet from the host to the accelerator takes minimally 1.5 μs. The demonstration system takes a minimum of 14 μs for an alignment with most of the time being attributed to the overhead of several packets, since only 2.65 μs is required for a single pipeline fill and flush once sequences are ready at the front-end of the pipeline.

Sequences shorter than W have a lower bound on alignment time, because unused PEs must be filled with null symbols. Longer sequences realize greater performance on the accelerator because the pipeline does not require a flush between adjacent slices. Adjacent slices need only 1 cycle of spacing in the pipeline. Longer sequences are also more efficient because of proportionately less time spent in the traceback. The average traceback time relative to the forward scan can be visualized in Figure 2 as the area of the sub-blocks relative to the area of the whole matrix.

Even though the algorithm presented here requires $O(mn)$ space, the traceback memory is reduced by a significant constant. For example, given sequences with 100 K symbols, saving all the traceback data requires 2.5 GB. By saving the partial traceback pointers in a system with 256 PEs, the traceback data is reduced to 78 MB. Perhaps more importantly, the necessary memory bandwidth to store the partial traceback pointers is reduced to a practical level that is achievable between the host computer and the FPGA accelerator. With the pipeline running at 100 MHz and 16-bit R values, the partial traceback data rate is only 200 MB/s.

Qnet provides communication bandwidth up to 600 MB/s per link in each direction between modules, which exceeds the rate needed by the alignment module to maintain maximum throughput in the pipeline. With excess

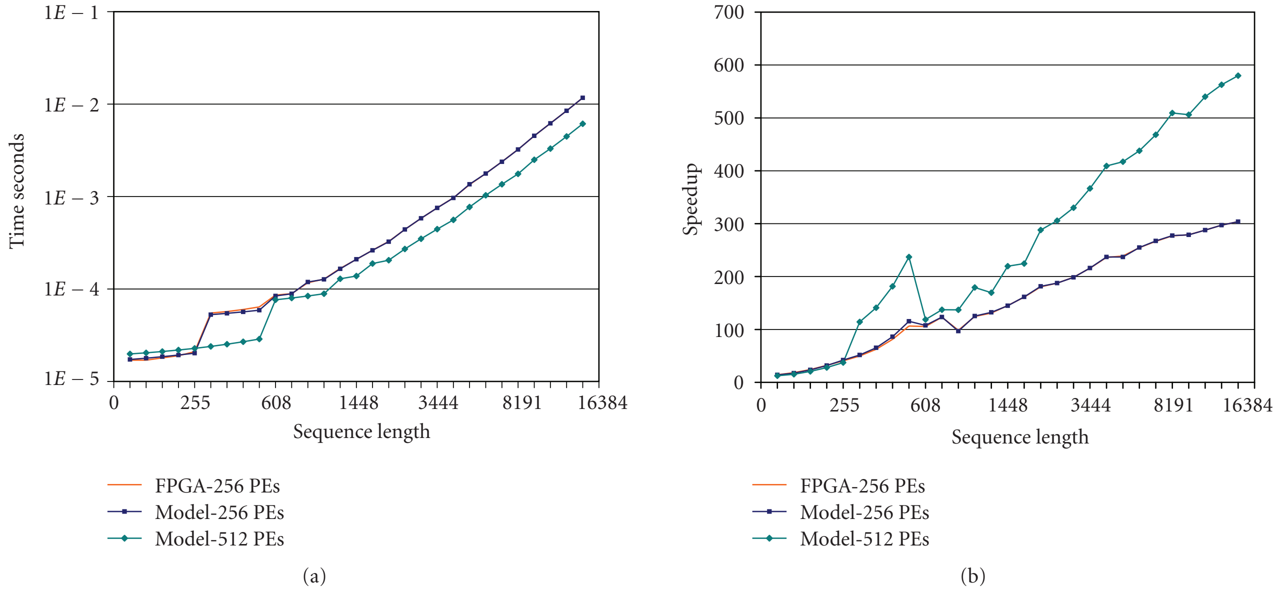

Figure 11: Timing model compared with actual FPGA performance. The model is nearly indistinguishable from the FPGA time. (a) Sequence alignment execution time, (b) speedup relative to the host-only version.

bandwidth at each end of the pipeline, stalls occur infrequently. Sequences enter the alignment module on ports A and B at a rate of 100 MB/s. Concurrently, partial traceback pointers exit port A at 200 MB/s destined for the host, and merged *B*-*H* values exit port B at 400 MB/s destined for the DP FIFO.

Notice that the presented algorithm does not limit the sequence length by the number of PEs or by the amount of full traceback memory. Matching system parameters, such as the number of PEs and the size of traceback memory, to the available FPGA resources maximizes performance. The experimental results and timing model together demonstrate the scalability of the algorithm without memory bandwidth limitations.

8. Conclusion

With the presented algorithm and architecture, long sequences are globally aligned with supercomputing performance on reconfigurable hardware. A speedup over 300 is achieved with the example implementation on FPGA technology when compared to a desktop computer. The architecture is scalable to larger capacity FPGAs for a further increase in performance. Beyond sequence comparison, the full alignment of long sequences is accelerated without memory and I/O bottlenecks through a space-efficient algorithm. After executing traceback in hardware, the accelerator returns a list of edit operations to the host, which constitutes an optimal alignment. Other global alignment acceleration methods only address sequence comparison, limit the sequence length, or exhibit memory and I/O bottlenecks.

The key features of the algorithm are the bounded space requirement for full traceback memory and the reduced space for partial traceback memory. These space reductions enable high-performance alignment of long sequences on a reconfigurable accelerator and are a match for FPGA memory capacities and bandwidth. Only 64 KB of FPGA block RAM is used for full traceback in the demonstrated implementation. Partial traceback data sent to the host at a rate of 200 MB/s is supported by commodity FPGA boards.

Future work includes combining coarse-grain parallel methods [27] with the fine-grain parallelism of this method for multiplied performance gain on reconfigurable computing clusters. Also, the advantages of the presented method are applicable to accelerating local alignment. A general-purpose accelerated alignment library that consists of both local and global methods may be applied to multiple sequence alignment codes with minimal effort.

Acknowledgment

An earlier version of this paper appeared as “Sequence Alignment with Traceback on Reconfigurable Hardware.” In *Proceedings of the 2008 International Conference on ReConFigurable Computing and FPGAs (ReConFig’08)*, Pages 259–264, December 2008.

References

[1] C. Macken, H. Lu, J. Goodman, and L. Boykin, “The value of a database in surveillance and vaccine selection,” in *Options for the Control of Influenza IV*, vol. 1219 of *International Congress Series*, pp. 103–106, October 2001.

[2] S. N. Gardner, M. W. Lam, N. J. Mulakken, C. L. Torres, J. R. Smith, and T. R. Slezak, “Sequencing needs for viral diagnostics,” *Journal of Clinical Microbiology*, vol. 42, no. 12, pp. 5472–5476, 2004.

[3] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[4] C. Notredame, D. G. Higgins, and J. Heringa, "T-coffee: a novel method for fast and accurate multiple sequence alignment," *Journal of Molecular Biology*, vol. 302, no. 1, pp. 205–217, 2000.

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[6] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[7] T. Ramdas and G. Egan, "A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology," in *Proceedings of the IEEE Region 10 Annual International Conference (TENCON '05)*, pp. 1–6, Melbourne, Australia, November 2005.

[8] P. Chain, S. Kurtz, E. Ohlebusch, and T. Slezak, "An applications-focused review of comparative genomics tools: capabilities, limitations and future challenges," *Briefings in Bioinformatics*, vol. 4, no. 2, pp. 105–123, 2003.

[9] F. Delsuc, H. Brinkmann, and H. Philippe, "Phylogenomics and the reconstruction of the tree of life," *Nature Reviews Genetics*, vol. 6, no. 5, pp. 361–375, 2005.

[10] T. Slezak, T. Kuczmarski, L. Ott, et al., "Comparative genomics tools applied to bioterrorism defence," *Briefings in Bioinformatics*, vol. 4, no. 2, pp. 133–149, 2003.

[11] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982.

[12] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Computer Applications in the Biosciences*, vol. 4, no. 1, pp. 11–17, 1988.

[13] D. T. Hoang and D. P. Lopresti, "FPGA implementation of systolic sequence alignment," in *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, H. Grünbacher and R. W. Hartenstein, Eds., pp. 183–191, Springer, Berlin, Germany, 1992.

[14] R. P. Jacobi, M. Ayala-Rincón, L. G. Carvalho, C. H. Llanos, and R. W. Hartenstein, "Reconfigurable systems for sequence alignment and for general dynamic programming," *Genetics and Molecular Research*, vol. 4, no. 3, pp. 543–552, 2005.

[15] T. VanCourt and M. C. Herbordt, "Families of FPGA-based accelerators for approximate string matching," *Microprocessors and Microsystems*, vol. 31, no. 2, pp. 135–145, 2007.

[16] Y. Yamaguchi, T. Maruyama, and A. Konagaya, "High speed homology search with FPGAs," in *Proceedings of the 7th Pacific Symposium on Biocomputing (PSB '02)*, pp. 271–282, Lihue, Hawaii, USA, January 2002.

[17] G. L. Moritz, C. Jory, H. S. Lopes, and C. R. E. Lima, "Implementation of a parallel algorithm for protein pairwise alignment using reconfigurable computing," in *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig '06)*, pp. 99–105, San Luis Potosi, Mexico, September 2006.

[18] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 561–570, 2009.

[19] E. W. Edmiston, N. G. Core, J. H. Saltz, and R. M. Smith, "Parallel processing of biological sequence comparison algorithms," *International Journal of Parallel Programming*, vol. 17, no. 3, pp. 259–275, 1988.

[20] X. Guan and E. C. Uberbacher, "A multiple divide-and-conquer (MDC) algorithm for optimal alignments in linear space," Tech. Rep. ORNL/TM-12764, Oak Ridge National Lab., June 1994.

[21] R. Lipton and D. Lopresti, "Comparing long strings on a short systolic array," in *Systolic Arrays*, W. Moore, A. McCabe, and R. Urquhart, Eds., pp. 363–376, Hilger, Bristol, UK, 1987.

[22] S. Lloyd and Q. Snell, "Qnet: a modular architecture for reconfigurable computing," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '08)*, pp. 259–265, Las Vegas, Nev, USA, July 2008.

[23] C. Sanderson, "Simplify FPGA application design with DIMEtalk," *Xcell Journal*, vol. 51, pp. 104–107, 2004.

[24] PCI-SIG, "PCI Express," http://www.pcisig.com/.

[25] P. Faes, B. Minnaert, M. Christiaens, et al., "Scalable hardware accelerator for comparing DNA and protein sequences," in *Proceedings of the 1st International Conference on Scalable Information Systems (INFOSCALE '06)*, ACM, Hong Kong, May-June 2006.

[26] Seq-Gen, http://tree.bio.ed.ac.uk/software/seqgen/.

[27] S. Rajko and S. Aluru, "Space and time optimal parallel sequence alignments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 12, pp. 1070–1081, 2004.