2008-12-01

Sequence Alignment with Traceback on Reconfigurable Hardware

Scott Lloyd

Quinn O. Snell
snell@cs.byu.edu

Sequence Alignment with Traceback on Reconfigurable Hardware

Scott Lloyd and Quinn O. Snell
Brigham Young University, Dept. of Computer Science, Provo, UT 84602
{gslloyd, snell}@cs.byu.edu

Abstract

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.

1. Introduction

Searching and comparing biological sequences in the 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 [10] and by the nation's BioWatch [3] 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 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 [20], [14] often compute a pairwise alignment between all the sequences.

Global and local pairwise alignment are the two most common alignment problems. Global alignment [13] considers both sequences from end to end and finds the best overall alignment. Local alignment [19] 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 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 bio-sequence applications with hardware have focused on database searches. Ramdas and

978-0-7695-3474-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ReConFig.2008.30

Egan [17] 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 and a traceback procedure, which is not addressed by sequence comparison solutions.

The predominant, non-parallel algorithms for global sequence alignment are described by Gotoh [4] and Myers-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 or 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 [6] describes an FPGA architecture which consists of a linear systolic array of PEs that output traceback data. However, the type of sequences are limited to only DNA and the sequence length is limited by the number of PEs on the accelerator (a couple of hundred nucleotides). The work by Jacobi et al. [7] and VanCourt-Herbordt [21] 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. [22] and Moritz et al. [11] 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. Operating at 100 MHz, a systolic array with 256 PEs requires 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 can handle the memory bandwidth, but even modest sequence lengths of 16K 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 [13], [4].

Given a pair a *sequences* $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ 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 $s : \Sigma \times \Sigma \rightarrow \mathbb{Z}$ determine the similarity of symbol a_i with b_j, and 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 $a_1a_2...a_i$ and $b_1b_2...b_j$. 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 $1 \leq i \leq m$ and $1 \leq j \leq n$:

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

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 Equation 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

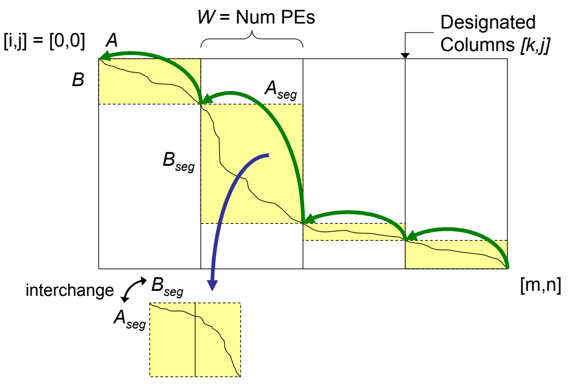

Figure 1. Forward scan and traceback

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. [2], and the divide-and-conquer scheme of Guan and Uberbacher [5]. Since sequence lengths are often longer than the number of PEs available in a systolic array, the problem is often partitioned [8]. 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 1). The designated columns k are given by $k \in \{c \,|\, c \bmod W = 0 \vee c = m\}$. 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

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

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.

```
procedure TracePartial($A, B, m, n, R, E$)
{
  $x_2 \leftarrow m,\ y_2 \leftarrow n$
  while ($x_2 > 1$) do
    $x_1 \leftarrow \lfloor (x_2 - 1)/W \rfloor \cdot W + 1$
    $y_1 \leftarrow (x_1 > 1 \wedge y_2 \ge 1)\ ?\ R[x_2, y_2] + 1 \ :\ 1$
    $xlen \leftarrow x_2 - x_1 + 1,\ ylen \leftarrow y_2 - y_1 + 1$
    if ($ylen = 0$) then
      Add $xlen$ DELETE operations to $E'$
    else if ($ylen \le Y$) then
      ScanFull($A_{x_1}, B_{y_1}, xlen, ylen, T$)
      TraceFull($A_{x_1}, B_{y_1}, xlen, ylen, T, E'$)
    else // interchange $A$ and $B$
      ScanPartial($B_{y_1}, A_{x_1}, ylen, xlen, R'$)
      TracePartial($B_{y_1}, A_{x_1}, ylen, xlen, R', E'$)
      $\forall e \in E'$ : replace DELETE ⇔ INSERT
    end if
    $E \leftarrow E \cup E'$
    $x_2 \leftarrow x_1 - 1,\ y_2 \leftarrow y_1 - 1$
  end while
}
```

Figure 2. Algorithm for TRACEPARTIAL

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 $[k_{prev},\ R[k,j]]$. 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. Figure 2 shows the algorithm, which 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.

4. Architecture

The global alignment accelerator is implemented using Qnet [9], an open-source packet-switched network architecture similar to DIMEtalk [18]. Qnet components in-

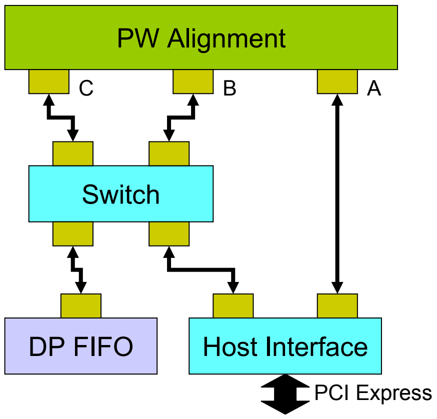

Figure 3. System architecture

terconnect 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 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.

4.2. System Modules

Host Interface. The host computer communicates with the FPGA accelerator through the PCI Express [15] 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.

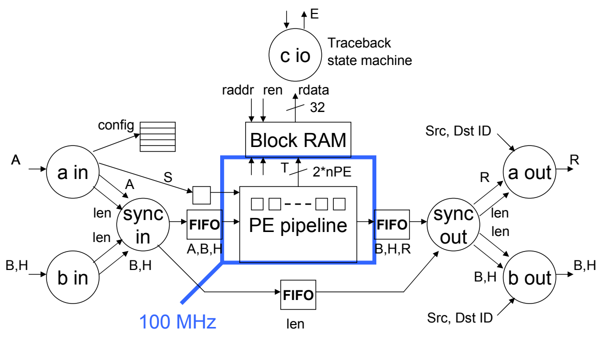

Figure 4. Pairwise alignment module

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 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. The recurrence equations described in Section 3 are calculated by the PEs each time a pair of symbols enter the pipeline.

Figure 4 shows the internal architecture of the alignment module. The front-end of the pipeline synchronizes the A and B streams of symbols. The pipeline 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. Both of the merged B and H values are sent in a packet to the DP matrix FIFO. As the sequence A is fed into the pipeline, the 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.

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 Figure 2).

5. 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 [1] produced varying lengths of test sequences ranging from 128 to 16383 symbols for the evaluation.

Host. The host platform consists of a desktop computer with a 2.4 GHz Intel® Core™2 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. The DP matrix FIFO uses 64 KB of FPGA block RAM, which is enough to hold 16K entries of B symbols and H values. With the use of floor planning, 256 PEs are instantiated within the pipeline and clocked at 100 MHz. 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.

Table 1. Speedup between implementations

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

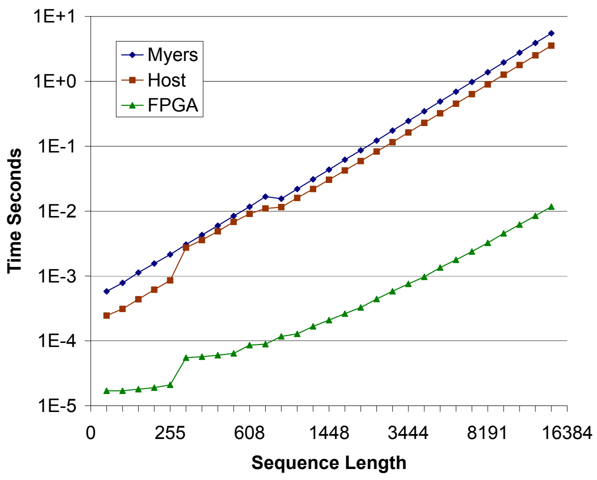

Figure 5. Global alignment execution time

6. Results

Figure 5 shows the performance of the three global sequence alignment implementations with varying lengths of sequences and Table 1 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. For longer sequences, the actual performance is near the theoretical peak. A 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.

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 1 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 100K 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 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.

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.

7. 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 [16] 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.

References

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

[2] 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*, 17(3):259–275, 1988.

[3] 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*, 42(12):5472–5476, December 2004.

[4] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

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

[6] D. T. Hoang and D. P. Lopresti. FPGA implementation of systolic sequence alignment. In H. Grünbacher and R. W. Hartenstein, editors, *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, pages 183–191. Springer-Verlag, Berlin, 1992.

[7] 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*, 4(3):543–552, September 2005.

[8] R. Lipton and D. Lopresti. Comparing long strings on a short systolic array. In W. Moore, A. McCabe, and R. Urquhart, editors, *Systolic Arrays*, pages 363–376. Hilger, 1987.

[9] S. Lloyd and Q. Snell. Qnet: A modular architecture for reconfigurable computing. In *Procedings of the 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'08)*, pages 259–265, July 2008.

[10] C. Macken, H. Lu, J. Goodman, and L. Boykin. The value of a database in surveillance and vaccine selection. *International Congress Series*, 1219:103–106, October 2001.

[11] 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 *Reconfigurable Computing and FPGA's (ReConFig)*, pages 99–105. IEEE, September 2006.

[12] E. W. Myers and W. Miller. Optimal alignments in linear space. *Comput. Appl. Biosci.*, 4(1):11–17, 1988.

[13] 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*, 48(3):443–453, 1970.

[14] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, September 2000.

[15] PCI-SIG. PCI Express. http://www.pcisig.com/.

[16] S. Rajko and S. Aluru. Space and time optimal parallel sequence alignments. *IEEE Transactions on Parallel and Distributed Systems*, 15(12):1070–1081, December 2004.

[17] T. Ramdas and G. Egan. A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology. In *TENCON 2005 IEEE Region 10*, pages 1–6, November 2005.

[18] C. Sanderson. Simplify FPGA application design with DIMEtalk. *Xcell Journal*, Winter(51):104–107, 2004.

[19] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[20] 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*, 22(22):4673–4680, 1994.

[21] T. VanCourt and M. C. Herbordt. Families of FPGA-based accelerators for approximate string matching. *Microprocessors and Microsystems*, 31(2):135–145, March 2007.

[22] Y. Yamaguchi, T. Maruyama, and A. Konagaya. High speed homology search with FPGAs. In *Pacific Symposium on Biocomputing*, pages 271–282, 2002.