Gigabyte-Scale Alignment Acceleration of Biological Sequences via Ethernet Streaming

Theepan Moorthy and Sathish Gopalakrishnan

Electrical and Computer Engineering, The University of British Columbia

***Abstract*—We describe the design of a PC-to-FPGA data streaming platform that enables hardware acceleration of gigabyte scale input data. Specifically, the acceleration is an FPGA implementation of the Dialign Algorithm, which performs both global and local alignment of query biological sequences against relatively larger reference strands of biological sequences. Earlier implementations of this algorithm could not be scaled to handle gigabyte-length reference sequences, nor megabyte-length query sequences, due to the inherent limitations of available memory and logic on single-FPGA platforms. We solve these issues via the design of an Ethernet channel to stream the reference sequence, and describe the novel use of SATA based Solid State Drives (SSDs) to time multiplex the FPGA logic into handling larger query sequences as well. In doing so, this paper also presents a general method to achieve gigabyte-depth FIFOs on commercially available FPGA development boards. This benefits data-intensive acceleration even outside of the bioinformatics application domain. Through the development of our acceleration logic and careful coupling of the required IO peripherals, we have successfully demonstrated a processing time of 28.61 minutes for a 200 base-pair query-sequence aligned against a 1 GB reference-sequence, a rate that is limited only by SATA 2 SDD write speeds. The present runtime offers a 38x speedup (18.36 hours down to 28.61 minutes) compared to standalone PC based processing.**

I. Introduction

As the logic density of FPGAs continues to scale the opportunity to accelerate new applications through hardware concurrency will also multiply. In addition, as parallel programming steadily continues to become more of a necessity than a luxury for new applications, the ease of adopting algorithms to run on parallel hardware will also rise. In this wave of transition to parallel programming, we believe that the greater challenge to hardware designers lies not in developing concurrent architectures to support acceleration, but in high throughput data delivery to fully support the acceleration itself. This shifting of efforts in terms of development time to design Input/Output (IO) interfaces to handle "big data" applications creates need for unconventional memory hierarchies as well as the need for new Electronic Design Automation (EDA) tools to help design and connect such resources.

II. The DIALIGN Algorithm

Bioinformatics algorithms generally lend themselves well to hardware acceleration due to their inherent amount of parallelization. The Smith-Waterman algorithm [1] is one such well known algorithm that offers optimal results (via brute-force processing) for local alignment purposes. Local alignment refers to the task of matching a query DNA sequence against a *single* region of a reference DNA sequence, which offers the highest degree of similarity. In contrast, other global alignment algorithms, try to match the query sequence against the entire/global length of the reference sequence, by allowing for the query sequence itself to be segmented across *multiple* regions of the reference sequence. Alternatively, an algorithm such as DIALIGN [2] can be used to perform either local or global alignment along with pairwise or multiple sequence alignment as well; the drawback being that the increased functionality also increases the computational demand.

In an effort to reduce the runtimes associated with the increased computational load of DIALIGN, previous work has had success on parallel processors [3]. However, the parallel processors approach for large genomic sequences uses heuristics to reduce runtimes. This is similar to BLAST and other popular algorithms that also make use of heuristics to trade off optimality in order to provide tolerable runtimes (minutes instead of hours). Boukerche et al. [4] advanced DIALIGN processing speeds even further by providing the first hardware based implementation for it, with the added benefit of not having to rely on heuristic methods to do so.

DIALIGN takes an input query sequence against another input reference sequence, and aligns/matches segments of the query sequence to the regions of the reference sequence that offer the highest (optimal) degree of similarity. The output of DIALIGN is a mapping of the query sequence coordinates to various reference sequence coordinates (i.e. regions).

DIALIGN, uses a scoring matrix, with the characters of the query sequence represented as columns and that of the reference sequence placed as rows. Dynamic programming is then often used to compute each score value of the matrix, once finished, a trace-back procedure based on the best scores is used to retrieve the optimal alignment. Acceleration in hardware is achieved by computing several scores of the matrix in parallel along sequential anti-diagonals of the matrix on each clock cycle (Figure 1).

The systolic array architecture implemented by Boukerche et al. [4] (Figure 2), uses a daisy-chained series of PEs to compute the anti-diagonals of Figure 1 per clock cycle. It is successful in hardware primarily due to its innovations in using a linear memory model, to solve what otherwise would require a 2-dimentional memory space to fill in the matrix in software.

The systolic array architecture (Figure 2) stores each character of the query sequence within each PE, then streams the reference sequence across the chain of PEs to compute the matrix scoring scheme. Given the complexity of each PE, only 50 PEs were synthesizable on our present Virtex 5 (XC5VLX110T) FPGA implementation (XUP 5 Development board). Increasing logic capacity on future FPGAs may fit

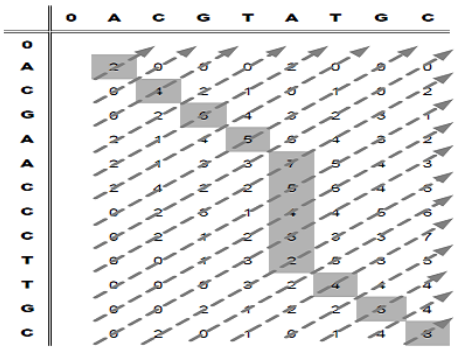

Fig. 1. Parallel matrix scoring along the anti-diagonals (dashed arrows)

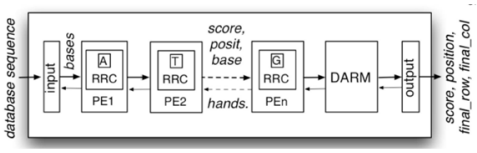

Fig. 2. Array Architecture for Single-Partition processing [4]

more PEs by a factor-fold, but desired query-lengths are already longer by orders of magnitude [5] [6] [7]. Ideally, this architecture performs best when all of the characters of the query sequence can entirely fit into the total n number of synthesizable PEs. This work addresses the problem of scaling the architecture when the query-size exceeds the number of available PEs.

III. Reference-Sequence Streaming

This section describes our solution to allow for gigabyte reference sequences to be processed by the FPGA.

Simple Interface for Reconfigurable Computing (SIRC) [8] is a software and matching hardware API solution that provides for an abstracted level of communication between a Windows based host PC and an FPGA device. SIRC was created to allow for rapid integration of physical IO channels on an FPGA via standard and user friendly APIs, thus obviating manual configuration of hardware IP. Simplicity and ease of hardware integration with a user-circuit were the driving motives behind SIRC's architectural design choices. Therefore their interface works on a batch-mode data transfer mechanism where the host sends the FPGA a set number of bytes as input, the FPGA performs computations and then transfers a set amount of bytes back to the host upon completion. Appropriate handshaking protocols and synchronization are the fine grain accomplishments that make such transfers successful within their framework.

Despite the methods suggested by SIRC to achieve streaming being conceptually straightforward, their actual implementation, however, requires careful attention to how the single EMAC core is multiplexed in time between a pair of simultaneously active hardware APIs. This involves modifications to the Hardware API itself to include a new hardware Arbitration Module, in addition to crucial modifications to the accelerator pipeline such that any and all delays in the stream of input data does not corrupt the functional correctness of the accelerator-output. These last two points are the major contributions of

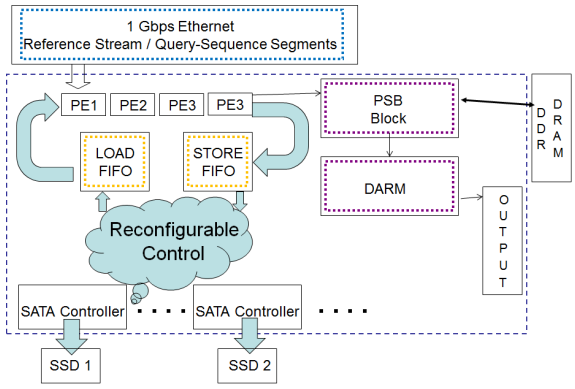

Fig. 3. FPGA based Storage Interfaces for Application Partitioning

this work in adopting and implementing SIRC to provide for a real-time reference sequence stream that successfully feeds DIALIGN acceleration on the FPGA.

IV. Scaling Query-Sequence Size

Relative to the difficulties of host communication to stream gigabyte *reference* sequences to the FPGA, the local storage and logic required to support the processing of large *query* sequences on a single FPGA poses greater challenges. We now move to describing the greater contribution of increasing acceleration support for longer query sequence lengths locally on the FPGA side.

Traditionally, local-data registered within the PEs themselves did not tax FPGA capacity. However, as the length of the locally stored operand data (e.g. megabyte or longer query sequences) continues to grow in DNA alignment applications, the number of PEs that will fit within an FPGA device to store the data locally may not be enough. For example, if a search query sequence is 200 characters long, and a particular FPGA device can only fit 50 PEs—then the 50 PEs have to be time multiplexed 4 times over to process all 200 characters of query data across the streamed reference data. This multiplexing is commonly referred to as partitioning the query sequence into multiple passes of operation across the workload.

Given that partition switches across the PEs are necessary to support increasing lengths of query-sequences, the demand that this process places on intermediate storage requirements when gigabyte reference sequences are streamed is now described. Three partition switches across a systolic-array architecture of 50 PEs daisy chained together, effectively creates the logical equivalent of 200 PEs being used in a similar daisy chain fashion. Thus, all of the data being produced by the last (50th PE) at each clock cycle of a single partition's operation must be collected and stored. This stored data will then be looped back and fed as contiguous input to the 1st PE when the next partition is ramped up, thereby creating the required illusion that 200 PEs are actually linked together (Fig. 3).

By making use of a memory hierarchy that goes beyond what is available on-chip, this paper puts forth the use and management of a SATA controller accessed by the FPGA to provide for gigabyte-depth levels of SSD based FIFO storage, and presents the challenges to keeping the latency of said FIFO at minimal levels. Upcoming sections will now break down

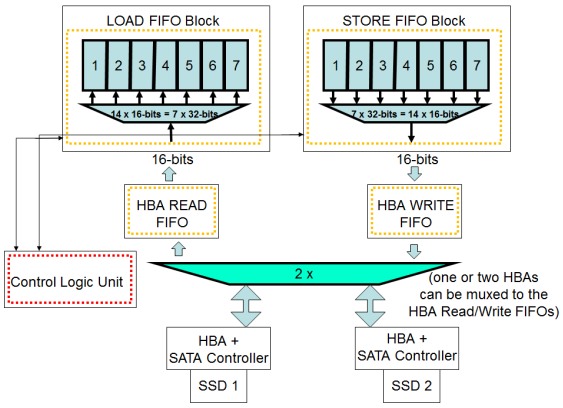

Fig. 4. Data flow to and from the Load/Store FIFOs to SSDs (Not all connections to the CLU are depicted)

the "Reconfigurable Data-Flow and Control" cloud depicted within the abstract representation of our hardware (Figure 3), and describe the select features of it in more detail.

A. *Stitching Partitions Together*

During the systolic-array based operation of the PEs, there are seven pieces of intermediate data that flow from one pipeline-stage to the next. Each piece of data, or 32-bit variable, equals 4 bytes of data flow. Thus 28-bytes of data in total, flow from one PE to the next, causing the last terminal PE in the chain to also output 28-bytes per cycle. A 1-gigabyte reference-sequence will correspondingly require at least 1 giga-cycles to stream its characters across the PEs, and thus 28-gigabytes of intermediate storage is required. This storage, captures the total per cycle output of one partition's worth of processing. Our synthesized system can be clocked at 67.689 MHz, thereby generating 1.876 gigabytes of data per second. In other words, a write bandwidth of 1876 MB/s for intermediate storage is required, if the accelerator is to be sustained at full throughput.

The actual SATA 2 SSD hardware available to us is not enough to sustain such demanding levels of write bandwidth. However, even as a low-cost proof of concept prototype, it still manages to harness accelerated performance relative to single-PC processing alternatives.

The abstract Load/Store FIFO Blocks (Fig. 3) are implemented as 7 banks of 32-bit wide FIFOs (Figure 4), each bank capturing 1 of the 7 values of output from the terminal-PE. We have arbitrarily set the depth of each bank to be 512 lines deep. Thus, each abstract Load/Store FIFO Block, when full, retains 14.336 KB of data (512 x 28-bytes).

We employ, Groundhog [9] an open-source SATA Host Bus Adapter (HBA) for FPGAs, to connect an SSD to our accelerator. Groundhog supports basic sequential read/write SATA commands, as well as more advanced Native Command Queuing (NCQ) SSD features. In this work, only sequential read/writes to the SSD were utilized, and NCQ optimizations were not attempted.

The HBA has a 16-bit data line, and allows for a minimum of 512 Bytes (1 sector) to be read/wrote at a time. Our accelerator has 7 banks of 32-bit FIFOs that need to be drained into the SSD. Therefore, we have designed a FIFO_TO_DISK_MUX module, to convert the accelerator's 224-bit (i.e. 7 banks by 32-bits) data line into yet another 16-bit HBA_WRITE_FIFO (Figure 4). Once the HBA_WRITE_FIFO reaches a full-threshold of 512 bytes, its almost-full line is asserted to trigger a sector-write to transfer the data into the HBA's own internal buffer. Similar HBA_READ_FIFO and DISK_TO_FIFO_MUX modules are also implemented to perform the inverse process of first collecting the 16-bit pay load data read from the SSD and then combining them to form single 224-bit payload writes into the Load-FIFO-Block.

During initial accelerator execution within the first partition, only the Store-FIFO-Block (SFB) is active. And in turn, the HBA is busy slowly writing the SFB data to disk. The HBA has been benchmarked to provide for an average sequential write-bandwidth of 66.324 MB/s (details of the connected SSD will be provided within the Experimental Results section). This is obviously significantly lower than the 1876 MB/s of data bandwidth being outputted by the accelerator into the SFB. Thus whenever the SFB is full, waiting for the HBA_WRITE_FIFO to drain it, the accelerator pipeline is intentionally paused/stalled.

Once all 28 gigabytes of captured intermediate data from the first partition execution has been written to the SSD, the system then begins to fully fill the Load-FIFO-Block in preparation for the next partition's worth of processing. Once the HBA_READ_FIFO has filled the LFB to capacity, the PEs are triggered to resume their pipeline flow. Our control logic mandates that the LFB must always be filled to capacity before any partition (apart from the very first one) can commence. This ensures that an accelerator pipeline stall is always only triggered by a full-line assertion from the SFB versus an empty-line assertion from the LFB. The sequential read bandwidth of SSDs is generally much greater than their write-bandwidth (our HBA is benchmarked to have a sequential read-rate of 272.964 MB/s and a sequential write-rate of 66.324 MB/s), this facilitates ease of control logic design in having the accelerator's PEs being paused and unpaused based solely on the full-line assertion and de-assertion of the SFB respectively.

When the 2nd partition pass now starts, *both* the Load-FIFO and Store-FIFO Blocks will be active. The LFB will be feeding the 1st PE with data from the previous partition, and the SFB will be capturing the data produced by the terminal-PE for this partition. The HBA_WRITE_FIFO and HBA_READ_FIFO, connected to the SFB and LFB respectively, are also now simultaneously active. However, only a full-line assertion from the HBA_WRITE_FIFO triggers any SSD activity, an empty-line assertion from the HBA_READ_FIFO does nothing. This protocol of control logic setup is well suited to our current implementation, which only utilizes a single SSD. A future implementation, which instantiates dual-HBAs connected to two independent SSDs, would have the empty-line and full-line triggering SSD reads and writes independently.

V. Experimental Results

Our system was implemented on the Digilent XUP-V5 development board [10], which features a 1-Gbps physical Ethernet line, two SATA header/connectors, and the Xilinx XC5VLX110T Virtex 5 FPGA. Our system consumed 88% of the device logic with the accelerator and IO interfacing resources combined, and the accelerator utilizing 50 PEs. The system is capable of operation at 67.689 MHz. The SSD connected to our board is the Intel SSD_SA2_MH0_80G_1GN model, with a capacity of 80 GBs. It offers SATA 2 (3 Gbps) line rates, and states 70 and 250 MB/s sequential write and read bandwidths respectively under technical specifications by the manufacturer. However, our benchmarked rates with the Groundhog HBA [9] (independent of any of our DIALIGN accelerator and interfacing logic) revealed maximum sequential write and read rates of 66.324 and 272.964 MB/s respectively.

A 1-gigabyte-reference and 200-character-query synthetic sequence was generated on the Host side. Timing is measured to start once the Software API begins to send over the first 50-character segment of the query-sequence to be loaded onto the FPGA, and stops once the PC console has printed all of the calculated DIALIGN fragments from the FPGA. A wall-clock time of 28.61 minutes was measured on average for the FPGA to align the 200-character query sequence against the 1-gigabyte reference sequence. The breakdown for the measured time can be roughly accounted for as follows.

Each partition's worth of execution needs to write and read 28 Gigabytes worth of data. The write-time for 28 gigabytes given our HBAs benchmarked rate of 66.324 MB/s is 7.20 minutes (28 x 1.024 GB / 66.324 MB / 60s). The read-time at an HBA read rate of 272.964 MB/s is 1.75 minutes. Given that only a single SSD is in use, these two time requirements cannot be overlapped, thus the total SSD time per partition's worth of execution becomes 8.95 minutes.

A. Comparison

The original DIALIGN hardware implementation [4], was measured against an optimized C model in software that captured the innovations in linear-memory processing that their novel wave-front hardware architecture provided. This was done so that the speedup comparisons between FPGA and PC runtimes did not unfairly disadvantage the PC by having it run the conventional DIALIGN algorithm [2], which does not run in linear memory space. The PC used in their experiments was a Pentium 4 3 GHz, and all disk read operations were eliminated by retaining all data in main memory. When the largest reference-sequence used in the experiments was 10 MB in length, 200 characters was set as the maximum length for the query-sequence. On the hardware side, since their system was not implemented to support partitioning, 200 PEs were synthesized to process the entire query length in one pass. This work around was accomplished by leaving only bare minimal hardware modules for synthesis, to squeeze all 200 PEs onto their Stratix 2 device.

Their PC experiments report runtimes for aligning the 200-character query-sequence against the 10 MB reference-sequence (Table 4 in [4]) to be 661.39 seconds or 11.02 minutes with the *optimized* C-model.

Modern PC systems can easily accommodate 4 Gigabytes or more of main-memory, so a reference sequence of 10 MB scaled to a 1 GB reference sequence should not unproportionally affect runtime for the better. From this we make the following assumption. The runtime on the Host PC to align the same 200-character query-sequence against a much longer 1-gigabyte reference sequence using the optimized C model should be no shorter than their runtime of 11.02 minutes factored by 100 (1-GB/10-MB), for a total time of 1102 minutes or 18.36 hours. This result is in line with why clustered computing resources are a must for bioinformatics applications when one cannot afford a half a day or more to obtain results from a single PC.

Our FPGA speed up for the 200-character query-sequence fully aligned against our 1-gigabyte reference-sequence, streamed in real-time, can now be stated as 38x (1102 minutes / 28.61 minutes). This is not as impressive as the 343x speedup reported by the original Boukerche et. al. DIALIGN implementation [4], but of course our added functionality of being able to support much larger query and reference sequences demands entirely different hardware resources. Nonetheless, we have cut roughly 18 hours of single PC processing down to less than half an hour via a low-cost prototype system.

References

[1] T. F. Smith, M. S. Waterman, "Identification of Common Molecular Subsequences," *J Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] B. Morgenstern, K. Frech, A. Dress, and T. Werner, "DIALIGN: Finding Local Similarities by Multiple Sequence Alignment," *Bioinformatics*, vol. 14, no. 3, pp. 290–294, 1998.

[3] Martin Schmollinger, Kay Nieselt, Michael Kaufmann, Burkhard Morgenstern, "DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors," *BMC Bioinformatics*, vol. 5, no. 128, 2004.

[4] Azzedine Boukerche, Jan M. Correa, Alba Cristina M.A. de Melo, Ricardo P. Jacobi, "A Hardware Accelerator for the Fast Retrieval of DIALIGN Bilogical Sequence Alignments in Linear Space," *IEEE Transactions on Computers*, vol. 59, no. 6, pp. 808–821, 2010.

[5] E. Sotiriades, C. Kozanitis, A. Dollas, "FPGA based architecture for DNA sequence comparison and database search," *Parallel and Distributed Processing Symposium*, p. 8, 2006.

[6] Khaled Benkrid, Ying Liu, AbdSamad 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, 2009.

[7] O. Storaasli, W. Yu, D. Strenski, and J. Maltby, "Performance evaluation of FPGA-based biological applications," *Proc. Cray User Group (CUG'07)*, 2007.

[8] Ken Eguro, "SIRC: An Extensible Reconfigurable Computing," *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2010.

[9] Louis Woods, Ken Eguro, "Groundhog - A Serial ATA Host Bus Adapter (HBA) for FPGAs," *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 220–223, 2012.

[10] Xilinx, "Xilinx University Program XUPV5-LX110T Development System," *http://www.xilinx.com/univ/xupv5-lx110t.htm*.