2013 26th IEEE Canadian Conference Of Electrical And Computer Engineering (CCECE)

Gigabyte-Scale Alignment of Biological Sequences: A Case Study of IO Bandwidth Reconfiguration for FPGA Acceleration

Theepan Moorthy*, Jan M. Correa†, and Sathish Gopalakrishnan*

*Department of Electrical and Computer Engineering, The University of British Columbia
†Department of Computer Science, University of Brasilia

Abstract

We expose the implementation challenges to sustaining acceleration speedups on FPGAs as the size of the data set to be processed scales. We examine the implementation of an FPGA platform for the processing of gigabyte scale biological sequences, and illustrate the significant design changes that must be made to achieve a successful implementation. In doing so, we demonstrate that conventional accelerator architecture design choices that focus on throughput speedup, in isolation of system level IO bandwidth feasibility, cannot sustain their throughput levels as the input data set scales. This is shown to be primarily due to currently unavailable high-bandwidth large-scale data storage and retrieval for FPGAs. As a solution to this problem, we propose a general FPGA based IO infrastructure to utilize high bandwidth hard-drive storage options, as means to achieving sustained throughput in the face of large data.

1 Introduction

Systolic-arrays are often adopted as the architecture of choice when designing accelerators that harness the inherent parallelism within FPGA fabrics [1] [2] [3]. The characteristics of the algorithm to be accelerated, and the typical size of the input data set that it aims to process, determine the number of Processing Elements (PEs) that a suitable systolic array architecture for it should contain. A systolic array offers optimal throughput performance when, ideally, all of its PEs can be synthesized within the available logic of a single FPGA. Any FPGA accelerator that uses a systolic array architecture is susceptible to performance degradation as soon as the input data set to be stored locally, in conjunction with the total number of PEs required to match such local data, exceeds its logic capacity. This problem has long been documented [4] [5]. However, published techniques to date rely on the assumption that the available on-chip memory capacity suffices to achieve their solution [6] to this problem. When scaling to stream in gigabytes of input data for processing, we illustrate with the case study presented herein, that this assumption quickly fails.

FPGAs used as customized accelerators for DNA alignment offer access to more parallel computing power at substantially lower costs, relative to computer cluster infrastructures. Work by Boukerche et al. [7] is an example of a successful implementation of the DIALIGN alignment algorithm on an FPGA platform. Our case study is an extension of that implementation, for the purpose of making it scalable to handle significantly larger sequences. We found that trying to scale the implementation to larger problem sizes dramatically increases the bandwidth requirements; here we will describe the difficulties associated with handling such increases in bandwidth, and explain how the problem arises. An outline for a low cost reconfigurable infrastructure, as a possible solution, to mitigate the increase in bandwidth without sacrificing FPGA throughput is presented.

The problem of increased IO bandwidth demands when scaling the size of input data sets is not unique to just bioinformatics applications. Thus the contributions of this article are not only applicable to the bioinformatics application presented in this case study, but more importantly, can be generalized to any systolic array based accelerator that is hindered by FPGA capacity limitations in the face of increased input data sizes.

2 FPGA Processing Architecture

Bioinformatics algorithms generally lend themselves well to hardware acceleration due to their inherent amount of parallelization. The Smith-Waterman algorithm [8] 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. The DIALIGN [9] algorithm is similar to the Smith-Waterman algorithm for local alignments, but enhances it to accommodate global alignment purposes as well by allowing for user controlled threshold levels, which can be used as a dial to vary whether optimal local or global alignments are desired.

The input data to DIALIGN is a series of 1-byte characters that represent either DNA or protein strands of query and reference sequences for comparison (cross comparisons between DNA strands and protein strands are never made). DNA representation requires only 4 distinct characters thus a 2-bit data implementation would suffice, however, protein variations require greater than 16 characters to represent. Thus an 8-bit data representation is favoured to accommodate these two types of input sequences.

DIALIGN takes an input query sequence against another input reference sequence, and aligns/matches segments of

978-1-4799-0033-6/13/$31.00 ©2013 IEEE

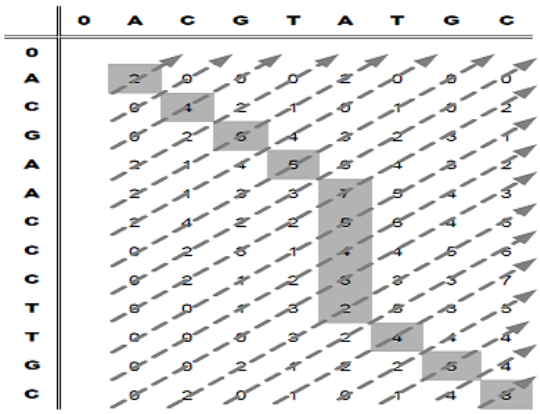

Figure 1: Parallel matrix scoring along the anti-diagonals (dashed arrows)

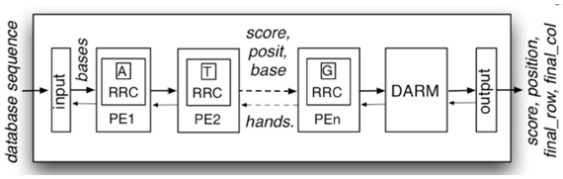

Figure 2: Array Architecture for Single-Pass processing [7]

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, similar to Smith-Waterman, 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 1Parallel matrix scoring along the anti-diagonals (dashed arrows)figure.1).

The systolic array architecture implemented by [7] (Figure 2Array Architecture for Single-Pass processing [7]figure.2), uses a daisy-chained series of PEs to compute the anti-diagonals of Figure 1Parallel matrix scoring along the anti-diagonals (dashed arrows)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 2Array Architecture for Single-Pass processing [7]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, generally only a 100 or so are synthesizable on current FPGAs. Increasing logic capacity on future FPGAs may fit more PEs by a factor-fold, but desired query-lengths are already longer by orders of magnitude [10] [6] [11]. Ideally,

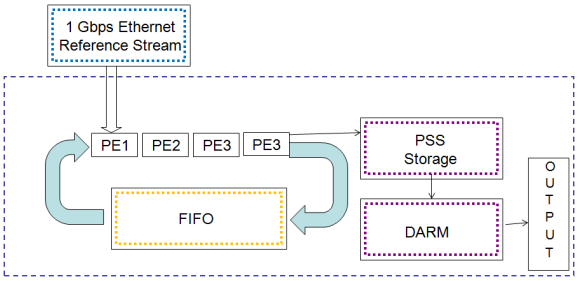

Figure 3: Inclusion of abstract FIFO and Partition-State-Storage (PSS) blocks.

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 ideal scenario will be examined first, before returning to the problem of scaling the architecture when the query-size exceeds the number of available PEs.

The reference sequence is at most 1 byte per character, and given normal accelerator operational speeds of 100 MHz, this requires only 100 MB/s of bandwidth. Current FPGA development boards offer direct SATA links to hard drives, along with the necessary SATA Controller IP infrastructure [12], reference sequences in the gigabyte ranges can be streamed in at the required 100 MB/s rate from a hard-drive database directly connected to the FPGA. Alternatively, a 1000 Mbps Ethernet stream could be used as well.

Given the streaming of the reference sequence, the PEs are self sufficient in calculating the scoring matrix. Once scoring calculations are complete, the DARM (Dialign Alignment Retrieval Module, Figure 2Array Architecture for Single-Pass processing [7]figure.2) retrieves score-data from the PEs in shift-register-fashion to compose alignment formation data as output.

3 Scaling Query Sequence Length

When the character length of the query sequence exceeds the number of PEs available for local storage and processing, the PEs must be time-multiplexed over consecutive partitions of the query sequence. This is referred to as the partitioning technique [7], but the paper goes no further in discussing the bandwidth challenges to implementation. Here we introduce the two new abstract hardware blocks (Figure 3Inclusion of abstract FIFO and Partition-State-Storage (PSS) blocksfigure.3), to the existing architecture of Figure 2Array Architecture for Single-Pass processing [7]figure.2, which partitioning functionality requires, and discuss the details of their implementation along with the bandwidth calculations to making them scalable.

3.1 Partition-State-Storage

Upon completion of matrix-scoring execution, each PE contains four pieces of trace-back data required by the DARM for further processing. Gigabyte scale reference and megabyte scale query sequences call for matrix row-column coordinates that require 32-bit representation. Thus each PE holds 16-bytes worth of post matrix processing data. To implement partitioning, the end state of each PE upon completion of each partition's worth of matrix-scoring must be stored.

When the total number of partitions required to fully process a query exceeds the available on-chip memory capacity to store PE states, off-chip DRAM must be used to prevent overflow.

The DARM used within the partitioning-unsupported architecture (Figure 2Array Architecture for Single-Pass processing [7]figure.2), experiences a maximum of n (number of PEs) on-chip-cycles of latency to retrieve data from the 1st PE in daisy-chained shift-register fashion. The DARM within the partitioning-supported architecture of Figure 3Inclusion of abstract FIFO and Partition-State-Storage (PSS) blocksfigure.3 would suffer from off-chip DRAM access latencies, to retrieve PSS data when query lengths call for off-chip storage. Although this does contribute to performance degradation in the face of partitioning induced latencies, since it is not strictly an IO bandwidth limitation, worst case DARM execution times when having to interface with off-chip DRAM are not analyzed any further.

3.2 Partition-to-Partition Data

The abstract FIFO block (Figure 3Inclusion of abstract FIFO and Partition-State-Storage (PSS) blocksfigure.3) provides the storage necessary to stitch the output data of one partition, as input data to the next partition. It must be capable of simultaneous read/write operations, since the last PE of the present partition will be writing to it while the 1st PE reads, from it, input data from the previous partition on each cycle.

Seven pieces of information flow from one PE to the next, during matrix scoring in this architecture. This totals to 28 bytes in a 32-bit implementation, which must be stored and read from the FIFO per cycle. For even a single gigabyte-length reference sequence, this equates to 28 gigabytes of data to be written and read, for each partition's stream of the reference sequence. Two or three gigabyte reference genomes lead to doubling and tripling this amount respectively. At this scale of generated intermediate data, it becomes clear that a storage drive must be inserted into the data-path at some point. Furthermore, the bandwidth required for such large scale storage, at 100 MHz, is placed at 5600 MB/s (read + write bandwidth).

4 System Infrastructure

An overview of competing PCI express based storage alternatives, and why they were ultimately rejected, can be found in the Discussion Section (Section 5).

Direct FPGA Storage: Systems with direct FPGA control of SATA based storage devices, in particular Solid State Drives (SSDs), have already been successfully implemented for high throughput large data acquisition applications in the medical imaging domains [13].

Our intended implementation board is the XUPV5 development board offered by Digilent. It utilizes the Xilinx Virtex 5, XC5VLX110T, device. This devices has two GTP RocketIO transceivers that can be utilized to implement two SATA 3 Gbps cores, for a theoretical one-way maximum bandwidth of 600 MB/s (given that each attached SSD can support equal read or write rates of 300 MB/s).

Although the theoretically available 600 MB/s supports neither the read nor write rate required by our application, it will however allow us to prototype and measure the system level degradation in accelerator performance that

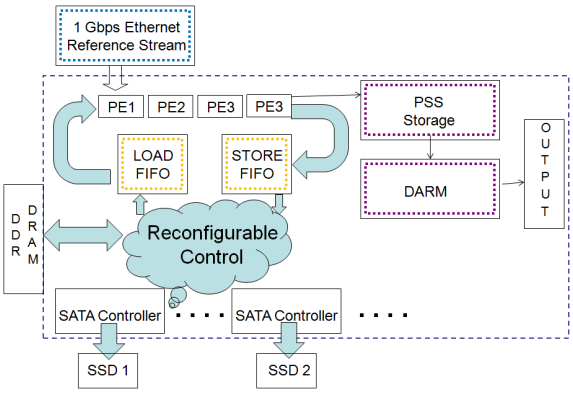

Figure 4: FPGA based Storage Interfaces for Application Partitioning

such a limitation exposes. Furthermore, we believe that this limitation will shortly be overcome by other - more expensive - Xilinx devices that implement 6 Gbps rather than 3 Gbps transceivers in conjunction with increasing the total number of available transceivers. Nonetheless, it is our intent to prototype this system with the available hardware to derive the actual percentage of achievable bandwidth from the 600 MB/s theoretical rate. Such data can then be extrapolated to a future system of 10 SATA 6-Gbs drives, where 5 SSDs can offer 3000 MB/s of write-bandwidth, while the remaining half is used for read-bandwidth. Consecutive partitions in the application can then toggle the read/write usage between the two halves of available drives.

The abstract FIFO (Figure 3Inclusion of abstract FIFO and Partition-State-Storage (PSS) blocksfigure.3) that was originally presented for our application's partitioning requirements is now illustrated (Figure 4FPGA based Storage Interfaces for Application Partitioningfigure.4) with some of the implementation details shown.

Reconfigurable Storage Interface: The reconfigurable flexibility of FPGA devices, in theory, allows for an abstract FIFO to be implemented as a storage system that supports not only a varying number of attached SSDs, but also accommodates itself to handle upgrades in future speeds of the physical SSD links.

Varying write latencies across the SSDs will inevitably require a buffer to be managed in DRAM [13] to prevent data loss. The data when stored on the SSDs is organised using the RAID protocol, thus a RAID controller which then drives the SATA-cores is also required. Control of the DRAM buffer, via the DRAM controller, the SATA-cores, via the RAID controller, is all accomplished through the Xilinx MicroBlaze 32-bit soft-core processor [13]. The goal of this work is to not only adopt this configuration for the need of our partitioning application, but to also make it scalable in handling a varying number of SSDs with varying SATA speeds.

At the hardware level, initially, manual synthesis of a varying number of SATA cores per number of SSDs used is required, along with their necessary data path wiring and port connections. However, at the embedded software level, fully flexible functionality to manage the DRAM buffer in the face of x number of SSDs used and their varying SATA-speeds supported, is envisioned.

5 Discussion

Competing Configurations for Storage: FPGA device-DRAM cannot be utilized, even if bandwidth requirements were met, since there is no data path available to a storage drive from device-DRAM (as exists on host-DRAM).

A PCI express 2.0 x8 (8000 MB/s) equipped FPGA-board affords enough bandwidth to transfer data to and from the FPGA at the required rates, however, the issue of high bandwidth storage to Hard Disk Drives (HDDs) still remains. To mitigate this issue, further Host Bus Adapter (HBA) hardware can be used to delegate the PCIe bandwidth across multiple HDDs [14]. However, direct peer-to-peer communication between the FPGA and HBA boards still may not be possible unless a dedicated peer-to-peer PCIe switch exists in combination with the integrated PCIe-root-complex of the Northbridge [15].

Achieving peer-to-peer communication from FPGA to HBA has another obstacle: commercial HBAs are designed for conventional software driver based operation. This then necessitates that the host OS and CPU become involved with the transfer of data to the HBA. Even when the use of DMA is considered, in this scenario, the drivers for the HBA card will most likely expect data to be flowing in from host memory, and not from the peer based FPGA board. Thus, the CPU may have to initiate a DMA transfer from the FPGA board (one memory page at a time) into host memory then back to the HBA for storage. This form of indirect data transfer, in the end, may not even support the 5600 MB/s of read+write bandwidth required by our FPGA application.

A direction considered, yet ultimately dismissed, is an architecture that eliminates a dedicated HBA card in favour of motherboard-integrated SATA controllers. There are motherboards that offer up to 12 independent SATA controllers, forming the basis for a peer-to-many-peers architecture. However, the same obstacles to efficient communication still apply in this case, as it did between the FPGA and HBA option. If the 8-lane FPGA board is placed on the Northbridge, a 2 GB/s DMI (Direct Media Interface) link, in itself will not have sufficient bandwidth to transfer FPGA data to the Southbridge. If a PCIe x8 port on the Southbridge is available, then the question becomes can the Southbridge PCI express switching fabric partition data fast enough across the slower 300 MB/s SATA drive links in parallel. The fact that these integrated SATA controllers are all OS-driver driven, meaning that they are conventionally designed to expect their data to flow in from host-DRAM, combined with the nature of slower PCIe switching fabric, does not facilitate high bandwidth peer-to-many-peers communication between an FPGA-board and such multiple host-SATA devices.

References

[1] Christos Kyrkou, Theocharis Theocharides, "A Flexible Parallel Hardware Architecture fro AdaBoost-Based Real-Time Object Detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, 2011.

[2] S. Natarajan, A. Mohan, P. Meher, "A Self-Configurable Systolic Architecture for Face Recognition System Based on Principal Component Neural Network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 8, pp. 1071–1084, 2011.

[3] Zhenyu Liu, Yiqing Huang, Yang Song, Satoshi Goto, Takeshi Ikenaga, "Hardware-Efficient Propagate Partial SAD Architecture for Variable Block Size Motion Estimation in H.264/AVC," *Proceedings of the 17th Great Lakes Symposium on VLSI*, pp. 160–163, 2007.

[4] S. Y. Kung, "VLSI Array Processors," *Englewood Cliffs, NJ: Prentice-Hall*, 1988.

[5] D. I. Moldovan, J. A. B. Fortes, "Partitioning and mapping of algorithms into fixed size systolic arrays," *IEEE Transactions on Computers*, vol. 35, pp. 1–12, 1986.

[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] 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.

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

[9] 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.

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

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

[12] 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.

[13] J. E. Breeding, W.F. Jones, J. H. Reed, T. Sangpaithoon, ""PETLINK Stream Buffer: Using an FPGA-based RAID controller with solid-state drives to achieve lossless, high count-rate 64-bit coincidence event acquisition for 3-D PET," *IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 3894–3900, 2011.

[14] HighPoint Technologies Inc., "A PCI 2.0 x8 HBA to 32 SATA-3 ports." [Online]. Available: http://www.highpoint-tech.com/USA$_new/cs-series_DC7280.htm$

[15] PLX Technology, "Express Apps." [Online]. Available: http://www.plxtech.com/files/pdf/apps/ExpApp49$_Dual-Graphics_07Mar07.pdf$