SeqHive: A Reconfigurable Computer Cluster for Genome Re-sequencing

Kristian Stevens*, Henry Chen§, Terry Filiba§, Peter McMahon† and Yun S. Song‡
*Department of Computer Science, University of California Davis, Davis, California 95616
Email: kastevens@ucdavis.edu
§Berkeley Wireless Research Center, 2108 Allston Way, Berkeley, California 94704
†Department of Electrical Engineering, Stanford University, Stanford, California 94305
‡Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, California 94720

***Abstract*—We demonstrate how Field Programmable Gate Arrays (FPGAs) may be used to address the computing challenges associated with assembling genome sequences from recent ultra-high-throughput sequencing technologies. Advances in sequencing technology allow researchers to generate immense amounts of raw data in the form of short reads with high error rates. A prerequisite to effectively utilizing this data for most applications is accurate alignment to a reference genome. While dynamic programming (DP) alignment algorithms are generally avoided on conventional architectures due to their computational complexity, they can be tailored for efficient implementation on systolic architectures. We describe and implement the first system capable of assembling large genomes using DP. We implemented application-specific DP algorithms for aligning data from ultra-high-throughput sequencers in a reconfigurable computing cluster. To obtain the necessary throughput while maintaining scoring integrity, we extended the compact encoding scheme of Lipton and Lopresti for our application. Each FPGA is capable of rapidly aligning multiple reads in parallel against a long reference genome. The reconfigurable cluster proves to be scalable and capable of processing real world datasets with a sustained performance of 11 tera cell updates per second. We examine the advantages and practicality of our system by benchmarking real genomic data from a large sequencing project. Our exhaustive validation confirms that application specific computing hardware can provide more accurate results than current heuristic methods and remain practical. While directly addressing the important problem of genomic assembly, particularly in circumstances where error rates or evolutionary divergence is high, the methods presented are also relevant to many other current applications for this type of data.**

I. Introduction and Background

In this paper, we examine a role for reconfigurable computing hardware in the compelling scientific challenge of obtaining a complete genome sequence. The current approach is Whole-Genome Shotgun (WGS), in which many identical copies of a genome are randomly fragmented into substrings of manageable length, and a deep random sample from this pool is then sequenced. The computationally challenging assembly part of WGS sequencing involves stitching together overlapping substrings (*reads*) to reconstruct the superstring genome. Assembly is substantially aided if there exists a reference genome with high sequence similarity. This is almost always from the same species as the sample being sequenced. Genome sequencing that uses a reference genome for assembly is referred to as *resequencing*.

The central computational problem we accelerate with reconfigurable hardware is the accurate alignment of a very large number of reads to a reference genome. A correct alignment positions each read at the *homologous* location where the read and the reference genome share a common evolutionary history. Importantly, reads differ from their corresponding reference genome substrings by errors and evolutionary events such as substitutions and indels (insertions and deletions). Therefore, we must use an alignment algorithm that can compensate for, if not accurately model, the potential differences between the two strings. The classic sequence alignment algorithms are based on *dynamic programming* (DP) and take $O(mn)$-time, where m is the length of the read and n the length of the reference genome. The scale of our problem is impressive and no system has been implemented to accomplish it using DP. The *Drosophila melanogaster* genome (sequenced 10 fold) requires 8×10^{17} recursions be computed, while the comparable depth in the human genome requires 9×10^{19}.

Here we describe the first system capable of resequencing a genome as large as Human within the time required to obtain the input reads. To accomplish this, it was necessary to implement an architecture that can efficiently pack multiple systolic arrays onto a single FPGA to exploit both the available coarse- and fine-grained parallelism. The reconfigurability provided by the FPGAs is critical for our system since it allows us to handle different length reads and different scoring functions. Importantly, we demonstrate the practicality of our approach by benchmarking on real genomic data from a large high-throughput sequencing project [http://www.dpgp.org]. Our system has a number of advantages over current heuristic methods.

Our current multi-FPGA prototype uses the widely available open source Berkeley Emulation Engine 2 (BEE2) system, described in Section II-B1. We have also ported and benchmarked our design to the Xilinx Virtex-5 architecture, anticipating wide availability of the next-generation BEE3 system. We show how the larger LUTs on the Virtex-5 can be used to extend the algorithm without a performance penalty.

1) Ultra-High-Throughput Sequencing: Genome sequencing technology has recently entered a revolutionary phase, opening vast new opportunities for biological sciences and related fields [1]. The fast and cost effective generation of immense amounts of raw, unassembled sequence data

can now be obtained from commercially available platforms. Large-scale resequencing projects are underway. Of particular importance to public health is the Human *1000 Genomes* Project [http://1000genomes.org]. Concurrently, other projects are sequencing a similar number of genomes of scientifically important model organisms, such as the *Drosophila Population Genomics Project* [http://www.dpgp.org]. The scale of data being generated is prompting the computational community to respond with faster algorithms and larger computational infrastructure to assemble and analyze the data. The methodology described here is directly relevant to these projects.

Several competing ultra high-throughput sequencing platforms are commercially available: the 454 sequencer from Roche Diagnostics; SOLiD from Applied Biosystems; and Illumina's Genome Analyzer. In this paper we focus on the Illumina platform because it is the primary technology used in the aforementioned projects and currently the most popular. Illumina's sequencer currently delivers millions of reads in a single run. The reads are of uniform length between 25 and 100 bp per run. Each run takes a few days to complete, depending on read length. For a system to be practical, it must first be computationally powerful enough to keep up with this rate of data production.

Single molecule technologies from companies such as Helicos Biosciences and Pacific Biosystems promise to offer a second wave of improvements, but currently suffer from higher error rates [2]. Notably, both 454 and Helicos also have insertion and/or deletion errors [2], [3]. Aligning reads under these error processes is therefore an important consideration.

2) DP Sequence Alignment: Our implementation of alignment is an application-specific hybrid between local (Smith Waterman) and global (Needleman-Wunsch) alignment algorithms [4]. In the context of resequencing, we expect a given read to be a substring of the reference genome, with differences arising from mutation and error. Our scoring methodology is similar to global alignment, but we do not charge a penalty for the gaps at the beginning or the end of the read. Unlike local (Smith Waterman) alignment, we align all read characters to the genome.

To be more precise, let $r_1, \ldots, r_m$ denote a read and $g_1, \ldots, g_n$ a reference genome, where $m \ll n$. Optimal alignments are found by filling in an $(m+1)$-by-$(n+1)$ DP table $C = (C_{ij})$, where $i = 0, \ldots, m$ and $j = 0, \ldots, n$ (see Figure 1). Each cell stores the score of the optimal alignment running through it. With the first row and the first column appropriately initialized, the following recursion computes the optimal score for the other cells:

$$C_{ij} = \min \begin{cases} C_{i,j-1} + \delta(-, g_j), \\ C_{i-1,j} + \delta(r_i, -), \\ C_{i-1,j-1} + \delta(r_i, g_j). \end{cases} \quad (1)$$

The penalty for a character deletion in the read is given by $\delta(-, g_j)$, while the penalty for a character insertion in the read is given by $\delta(r_i, -)$. The substitution penalty is given by $\delta(r_i, g_j)$. All penalties are non-negative. We examine the last row of the table C_{mj} for optimal alignment ending at position j in the reference.

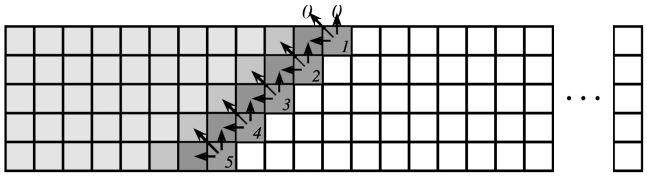

Fig. 1. Illustration of DP table fill by anti-diagonal with arrows used to indicate data dependencies. The cells along an anti-diagonal are filled simultaneously, each being assigned to a systolic processing element (PE). Communication is only required between adjacent (dependent) PEs which need only store the current and last result.

Figure 1 illustrates how the dependency structure allows for the DP table to be filled by anti-diagonal for maximal parallelism [5]. To exploit this fine-grained parallelism we map the recursion in (1) to a systolic array corresponding to the anti-diagonal. Our implementation chains m *processing elements* (PEs), corresponding to each position of the read, as illustrated in Figure 2. In this configuration, communication only happens between adjacent PEs. On the FPGA this results in a simple layout procedure with low fan-out signals traveling short distances.

Moreover, because the reads are a fixed short length, well under the number of PEs that can be synthesized on a single FPGA, we are able to exploit a coarser level of parallelism by synthesizing multiple systolic arrays to align multiple reads simultaneously. Particular attention is also paid to the scoring function to achieve a high number of PEs for our application. While the clock rate is an order of magnitude less than that of modern conventional processors, the parallelism obtained on an FPGA more than compensates for this.

Previous Work) The pioneering paper of Lipton and Lopresti [5] introduced both a systolic array architecture for DNA sequence alignment and an efficient encoding scheme to minimize the datapath width for scoring. Specifically, their encoding scheme allows the use of fewer bits for distance score computations by sacrificing scoring flexibility. The encoding scheme requires a specific scoring function (with mismatch penalty $= 2$ and indel penalty $= 1$) that allows for significant logic reduction in the recursion logic, with the added complexity of an accumulator. We extend their method to two alternative scoring functions.

Systems implementing their encoding scheme offer the most relevant comparison to our approach. Lipton and Lopresti targeted their architecture to a custom VLSI chip. The SPLASH [6] architecture later demonstrated the power of the emerging FPGA platforms for accelerating alignment. Two current surveys [7], [8] provide a detailed discussion of the development of the state-of-the-art in hardware-accelerated sequence alignment. We also highlight the prominent contemporary FPGA implementation of Yu et al. [9] which showed how the Lipton and Lopresti systolic cell could be efficiently implemented in modern FPGA logic slices.

Our design goals differ in fundamental ways from what has previously been demonstrated in the literature. First, we aimed to produce a design that is optimized for aligning many short-reads in parallel. Second, we aimed to build a cluster implementation and demonstrate the expected performance

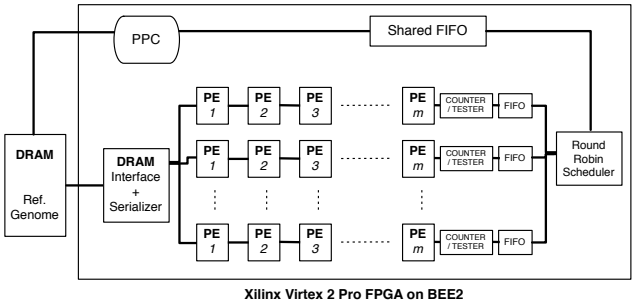

Fig. 2. Systolic array within a single FPGA. For aligning k length-m reads in parallel, there are k length-m systolic arrays of processing elements (PEs).

increase from taking advantage of the coarse parallelism available in the short-read alignment problem. Third, we wish to have the flexibility to report multiple alignment locations. Lastly, we motivate and implement application specific scoring using a compact encoding.

II. Implementation

We describe our reconfigurable cluster for accelerated high throughput genome resequencing. We begin with how the DP recursion is implemented in a processing element (PE) and then discuss the system's larger-scale architecture.

A. Processing Element Design

We mapped our DP alignment algorithm into a systolic array architecture where a processing element is dedicated to each position in a read sequence. To fully exploit available hardware, multiple systolic arrays must be synthesized on a single FPGA due to the short length of the reads (see Figure 2). A primary goal of our design was to reduce the size of the PE to maximize parallelism.

The greatest influence on the size of a PE in the FPGA implementation is the bitwidths of its data paths. PE size is thus primarily driven by the the precision of the scoring system employed and the value of the largest quantity that the processing element must add or compare. Naively our DP alignment has an implementation space complexity of $O(\log(m|\delta|))$ where m is the length of the read and $|\delta|$ is the number of bits needed to represent $\delta(.,.)$. An attractive aspect of short reads is that the value of m is typically small. However, we must do better. It was demonstrated in a classic paper by Lipton and Lopresti [5] that the dependence of the PE size on m can be eliminated in certain circumstances.

We label the three inputs and one output (a quartet) of the DP recursion scanning from left to right a, b, c, d, where a is the upper left input and d is the output. In their paper, Lipton and Lopresti proved that in any quartet of the DP table there are only 2 possible offsets from the value in a, $a-1$ and $a+1$, for the scoring function: $\delta(r,g)=2$, $\delta(r,-)=1$, and $\delta(-,g)=1$ (See Table I). If the range of possible relative values in a quartet is bounded to k, the relative values can be coded using $\lceil \lg(k) \rceil$ bits. We still determine the full score, but only for the last row of the DP table. This is done at the last PE by initializing a counter to the value of $C_{m,0}$. This counter

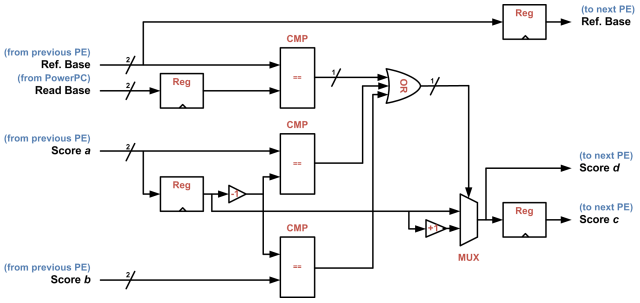

Fig. 3. Example PE logic diagram for the (1,1,1) scoring function.

TABLE I
Three Compact Offset Encoding Schemes.

Lipton and Lopresti's Scoring:
$\delta(r,g)=2$, $\delta(r,-)=1$, $\delta(-,g)=1$
Base case:

0	1
1	0 or 2

Induction:

a	$b = a-1, a+1$
$c = a-1, a+1$	$d = a, a+1$

Uniform Scoring:
$\delta(r,g)=1$, $\delta(r,-)=1$, $\delta(-,g)=1$
Base case:

0	1
1	0 or 1

Induction:

a	$b = a-1, a, a+1$
$c = a-1, a, a+1$	$d = a, a+1$

Rare Indel Scoring:
$\delta(r,g)=1$, $\delta(r,-)=2$, $\delta(-,g)=2$
Base case:

0	2
2	0 or 1

Induction:

a	$b = a-2, a-1, a, a+1, a+2$
$c = a-2, a-1, a, a+1, a+2$	$d = a, a+1, a+2$

is then incremented by the value of $C_{m,j} - C_{m,j-1}$ which is computed from the encoded scores.

Because under many circumstances their scoring scheme, henceforth $(2,1,1)$, is not optimized for our problem, we applied the same inductive reasoning to compactly encode two alternative scoring functions. The uniform $(1,1,1)$ scoring system, also known as edit distance, can be motivated because it assumes no prior knowledge about the relative frequency of insertions, deletions, and substitutions. Likewise, since in our data inserted and deleted bases are observed to be rarer than substitutions, the $(1,2,2)$ scoring function is of utility because it captures our prior information about rare indels in a reasonably compact design. Table I illustrates all three scoring functions mentioned here. In the $(1,1,1)$ case we show that a recursion contains only 3 possible relative values $(a, a-1, a+1)$ implying only 2 bits are required to propagate the score. In the $(1,2,2)$ case we show that a recursion contains only 5 possible relative values implying only 3 bits are required to propagate the score.

B. Reconfigurable Cluster Implementation

Our multi-FPGA platform is implemented on a cluster of eight BEE2 systems, each connected via a TCP/IP switch to a central control server. The control server handles loading the reference genome and read sequences into the systolic arrays,

setting control parameters, and reading back the results. On each BEE2 a control FPGA acts as an interface between the control server and four user FPGAs on which the systolic arrays are implemented. On each user FPGA, all the arrays are supplied with a reference genome from DRAM in parallel and share a common reporting mechanism, but each systolic array has an independently loadable read sequence register.

1) Berkeley Emulation Engine: As mentioned before, we utilized the Berkeley Emulation Engine (BEE) system, which is an FPGA-based computing platform with a wide range of applications. The second-generation BEE platform, called the BEE2 [10], is designed with high memory and I/O bandwidth. A single BEE2 system has 5 Xilinx XC2VP70 Virtex-II Pro FPGAs, each containing over 74,000 logic cells, 328 Block SelectRAM (BlockRAM or BRAM) memory blocks of size 18kb, and 2 PowerPC 405 cores. Each FPGA is supplemented with 4 DDR2 DIMM sockets. BEE2 systems can make use of 10/100 Mbps Ethernet connectivity.

2) FPGA Implementation: The systolic arrays within a single FPGA are shown in Figure 2. For simultaneously aligning k length-m reads, we have k length-m systolic arrays of PEs. Each PE computes the recursion of (1) implemented in Verilog HDL. The details of this computation are shown in Figure 3. The inputs to each PE's combinational logic are a fixed read character, a shifting reference character, and the scores from neighboring cells. The recursion is computed in a PE's combinational logic. The interdependency of the scoring function, bus-width, and register size require that the scoring function be implemented at compile time. Scores are 2-bits each for the $(1, 1, 1)$ scoring function. The read and reference characters are encoded using 2-bits to represent the four letter nucleotide alphabet. Each cell update computation is done in one clock cycle. The systolic array is fed reference sequence data from a dual ported BRAM that is loaded periodically from DRAM (described in Section II-B3). Each PE passes several pieces of data to the next PE in the array: the reference sequence character, and the current and previous scores. The last PE in the array computes the score in the last row (i.e., row m) of the DP table. Currently we implement a few options for alignment reporting. The position of the alignments are always reported; reporting the score is optional. Our design can also report the best alignment or all locations below a score threshold T. The latter requires additional buffering mechanisms at the output illustrated in Figure 2. In this mode the score is compared with a programmable threshold T, and if the score is $\leq T$, then the column position is written into a FIFO. On a single FPGA, k reads are processed in parallel, so there are k per-read FIFOs in each FPGA (see Figure 2). A round-robin scheduler reads the candidates from all k per-read FIFOs into a shared FIFO which is memory mapped to the PowerPC. Per-read FIFOs are used because many reads may simultaneously find match candidates, causing contention for the shared FIFO.

3) DRAM: The 2VP70 FPGAs on the BEE2 system have approximately 6Mb of on-chip BRAM, which is insufficient for storing a long reference genome, and therefore off-chip memory is required. We use the DDR2-SDRAM DIMMs

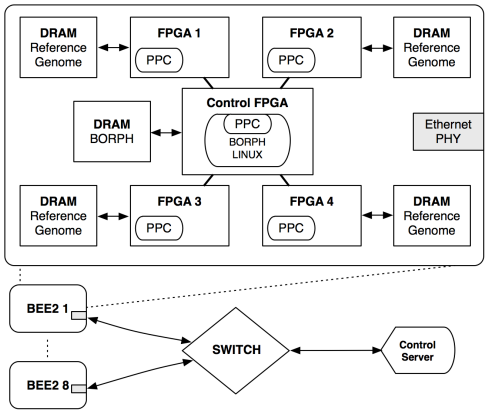

Fig. 4. Layout of a single BEE2 board and our eight board cluster.

available to each FPGA on the BEE2, with an addressable 2GB. In the application considered here, a single DIMM run at 200MHz is used by each FPGA with a 2-bit encoding to store a reference sequences as long as the Human genome with room to spare.

Our design implements a generic DDR2 DRAM interface that works with 144-bit data transfers. Due to the overhead of initiating DRAM accesses, multiple 144-bit words are read from DRAM and buffered internally in the FPGA. This puts several hundred reference characters into BRAMs and masks the non-constant access latencies of DRAM. In addition, the dual-ported BRAMs on Xilinx Virtex-II Pro FPGAs allow different aspect ratios, i.e., an 18kb BRAM can be written into as a 64×256 memory, but read out as a 2×8192 memory. The additional BRAM buffer eliminates the need for frequent DRAM accesses and provides an easy way of breaking up the data into individual reference characters.

4) BORPH Control Mechanism: The design is implemented on a BEE2 running BORPH, a custom Linux distribution that supplies OS-level access to FPGA resources [11]. By using BORPH, loading the DRAM with a reference sequence can be done by simply writing into a file representation of the DRAM's memory space. Likewise, 32-bit control and status registers, as well as the FIFOs for reporting results, are accessible as Linux files. A detailed description of BORPH and the memory interface was published in [12].

5) Overall Cluster Architecture: Shown in Figure 4 is the overall architecture of our cluster consisting of eight BEE2 boards. A standard 1 Gbps Ethernet switch is used to connect the BEE2s to a control server. The server is used to start and stop computation runs on the BEE2s, and copies the input data to and the results data from each BEE2. Each BEE2 board contains 4 FPGAs and each FPGA simultaneously aligns k length m reads. The implementations tested in Section III are focused on exploring various read length instantiations aligned with the $(1, 1, 1)$ scoring scheme with all alignment locations above threshold tracked. At maximum computational density, the cluster contains $81,920$ total PEs. At the maximum practical read parallelism the cluster contains $1,536$ length 31 systolic arrays using a total of $47,616$ PEs in parallel.

6) Virtex-5 Target: Anticipating the wider availability of the multi-FPGA BEE3 platform, we also ported our design to

TABLE II
PERFORMANCE IN GCUPS OF OUR CLUSTER PER BEE2 FPGA, INCLUDING AMORTIZED RECURRENT OVERHEAD.

# Reads Aligned	Time (s)	GCUPS per FPGA
900	18.7	366
1400	30.0	354
1800	38.2	358
2800	59.6	357
5600	119	359
11200	236	360
22400	477	357
44800	944	360

a Virtex-5 FPGA. The larger amount of logic hardware on the XC5VLX155T, specifically the 6-input LUTs, allow us to add functional extensions without a performance loss.

A compelling piece of functionality is to encode many genomes into a single *extended reference*. This allows for a more truthful representation of the underlying multiple alignment problem and reduces the potential for scoring bias arising from alternate reference characters. Our framework for doing this is to use a 4-bit encoding where each bit is assigned to one of the four nucleotide characters with its logical value indicating presence or absence. This is also useful for encoding the semantics of the empty set character, used frequently for masking repeats in biological databases.

Our synthesized systolic arrays achieved a clock rate of 300MHz. In addition to the higher clock frequency, we also were able to implement the extended reference with essentially no increase in slice occupancy (less than 0.1%). We also synthesized the $(1, 2, 2)$ scoring alternative with only a 4% increase in slice occupancy over the $(1, 1, 1)$ scoring function.

III. RESULTS

A. Hardware Performance

To benchmark the performance of our implementation, we used a cluster of eight BEE2 boards to align reads from the previously mentioned Drosophila Population Genomics Project against the 100 Mbp reference genome. We present performance results using the problem-size-independent throughput measure, billions of *Cell Updates Per Second* (GCUPS). A "cell update" refers to the calculation of a single DP recursion.

Table II shows the performance of our alignment implementation on a single FPGA on a BEE2. The FPGA implementation uses a fixed amount of time to find alignments for a set of k simultaneous reads; this time is directly proportional to the length of the reference sequence. There is also a fixed cost at startup to load the reference sequence into DRAM. Specifically, our 100 Mbp reference took approximately 53 seconds to be loaded from the filesystem on the BEE2 into DRAM. There is also overhead associated with loading the reads into the FPGA registers, reading results out from the FPGA and writing those results to the filesystem. While the former is essentially amortized away, the latter is not and is incorporated into our GCUPS measurements.

We ran multiple jobs in parallel on the cluster. There is a startup cost associated with distributing reads to the BEE2s, and loading the reference sequence for each of four FPGAs on each BEE2. The startup time is approximately 240 seconds, and is dominated by the reference sequence loading time.

TABLE III
DEPENDENCY OF RESOURCE UTILIZATION ON READ LENGTH m WHEN PACKING AS MANY READS AS POSSIBLE k ONTO A SINGLE FPGA.

Read length m	# Reads / FPGA k	# PEs / FPGA $m \times k$	Speed (GCUPS)
36	44	1592	318
48	38	1824	365
64	32	2048	410
76	28	2128	426
96	24	2304	461
256	10	2560	512
512	4	2048	410
1024	2	2048	410

Excluding the startup time (based on the valid assumption that the run time will dominate the startup time for all practical applications), we measured the sustained performance of the cluster to be 11.5×10^{12} CUPS including non-startup overhead. The cluster can align the reads associated with a 10 fold redundant Drosophila genome in $2.5h$ and a Human genome in $9.5d$.

We tested the scalability of our architecture as follows. To investigate how resource utilization depends on the length m of reads, we varied m and packed as many reads as possible per FPGA. This is an important study to carry out since future advances in base-calling algorithms and sequencing technology will likely increase read length. A summary of our results is shown in Table III. With smaller m, one can pack larger number k of reads, due to the finer granularity and smaller incremental cost of instantiating additional systolic arrays. However, the scaling bottoms out due to the overhead associated with individual arrays. Conversely, though larger m causes coarser granularity, instantiating fewer systolic arrays means less overhead is incurred.

We benchmarked a software implementation of our DP algorithm on a dual-processor 2.0 GHz quad-core Intel Xeon server with 8 GB of RAM. Each read was run as a single thread on one CPU core. On average, the serial code had a throughput of 0.1 GCUPS per core. In our SeqHive implementation, each FPGA yields over 1000X speedup over the comparable quad-core implementation for reads of 64 bp and longer. We considered benchmarking our system against the fastest SIMD implementation, which reported 1 GCUPS per 2.0 GHz core [13], but it does not handle long reference sequences nor does it report coordinates.

Direct comparisons between our work and previous related efforts are difficult, due to differences in design goals and platform. However, we note that our PEs are implemented at least as efficently as those reported by Yu et al. [9]: our PEs each use two 4-input LUTs and eight FFs in our Virtex 2 Pro 70 implementation, whereas Yu et al. reported effectively mapping a PE to six 4-input LUTs and six FFs on a Xilinx XCV1000E chip. Yu et al. [9] reported achieving a performance of 136 GCUPS, with 4032 PEs on their Xilinx XCV1000E-6 (Virtex) FPGA, running at 202MHz. Our implementation fitted 2560 PEs on a single FPGA, but running at 200MHz, could achieve performance in excess of 360 GCUPS per FPGA. Yu et al.'s performance was hampered by limited IO bandwidth, and our ability to fit more PEs onto each FPGA was limited by the need to instantiate a DDR2 controller on each chip.

TABLE IV
ALIGNMENT RESULTS FOR TEST DATA CONSISTING OF ONE MILLION READS ALIGNED TO HUMAN DNA CLONE BCX98J21.

Nominal 36 bp Illumina GA1 Reads ($T = 6$)

Evo. Rates	% True Positives			% False Positives			% False Negatives		
θ, γ, ν	Bowtie	BWA	FPGA	Bowtie	BWA	FPGA	Bowtie	BWA	FPGA
0%	81.20	81.30	92.91	1.20	1.10	0.95	17.60	17.60	6.13
.1%	76.30	80.20	92.44	1.20	1.10	1.33	22.50	18.60	6.21
.2%	71.60	79.00	91.93	1.20	1.20	1.71	27.20	19.90	6.35
1%	43.00	63.20	87.96	1.00	1.50	4.61	56.00	35.30	7.42
2%	22.40	41.00	82.52	0.70	1.60	7.92	76.90	57.50	9.54

Nominal 76 bp Illumina GA2 Reads ($T = 12$)

0%	93.80	93.70	97.82	1.20	1.30	0.75	5.10	5.00	2.17
.1%	88.20	92.80	97.40	1.20	1.30	1.15	10.60	5.90	2.59
.2%	83.00	91.70	96.98	1.20	1.30	1.55	15.80	7.00	3.01
1%	50.50	75.10	93.64	1.10	1.80	4.62	48.30	23.20	6.35
2%	26.60	49.40	89.12	0.80	1.90	8.33	72.60	48.70	10.87

Lower Cost, Higher Density 76 bp GA2 Reads ($T = 12$)

0%	34.00	59.80	91.74	0.00	0.00	0.04	66.00	40.20	8.2
.1%	28.70	57.00	91.19	0.00	0.00	0.39	71.30	42.90	8.41
.2%	24.20	53.80	90.61	0.00	0.10	0.74	75.70	46.10	8.63
1%	6.10	26.80	85.74	0.00	0.50	3.42	93.90	72.70	10.83
2%	1.10	8.30	78.34	0.00	0.40	6.17	98.90	91.40	15.48

B. Accuracy

We addressed the relative performance of our FPGA implementation of DP alignment to heuristic methods with a number of different datasets and use cases in mind. We compared to the popular fast heuristic method, Bowtie [14], and a more recent one, BWA, that allows for indel events [15].

1) Data simulation: In order to evaluate the implemented alignment algorithms, we created validation datasets, with the error profile of experimental data, where the correct alignment for each read is known. We started with experimentally determined reads obtained from multiple sequencing runs of a DNA standard with accurately estimated empirical quality scores. We simulated evolution using a three parameter model. The substitution parameter θ is the probability that two nucleotides in aligned sequences chosen at random will differ. Indels are controlled by γ, the probability that a base in the read is deleted, and ν, the probability that a base in the read is inserted. For each read a simulator was employed as follows. 1) A starting location is randomly chosen for the read. 2) Using the three parameter model of evolution above we mutate the reference substring. 3) Using the read's quality scores as an error profile, the substring is further mutated at each position with probability determined by the quality score.

2) Results: In Table IV we survey the alignment accuracy for different datasets and applications. Three tiers of evolutionary divergence were simulated: (i) $\theta = \gamma = \nu = 0$. (ii) $\theta = \gamma = \nu = 0.1\%$ or 0.2%, roughly corresponding to values seen within a population. (iii) $\theta = \gamma = \nu = 1\%$ or 2%, roughly corresponding to values seen between closely related species. Three datasets were tested from the Illumina GA1 and GA2. Principal differences were read length and the density at which reads were packed in the imaged field. Higher density implies lower costs but higher error rates. A read was considered correctly mapped (true positive) if its alignment coordinates were identical to the results known by the simulator.

The FPGA implementation of alignment performed optimally across the board in terms of sensitivity measured by true positives and false negatives. The heuristic programs, by nature, have very high specificity as demonstrated by the low number of false positives. Remarkably though, the FPGA implementation outperforms them in many circumstances, especially when the optimal T was chosen using an ROC analysis (not shown). Arguably, sensitivity is more important than specificity since redundancy can be used to filter incorrectly aligned reads using consensus. The ability to incorporate indels in the alignment model is critical when indels are present in the data, hence we see a large accuracy difference between the non-indel method and the two methods incorporating indels.

IV. DISCUSSION

We have demonstrated that reprogrammable hardware can play a significant role in the development and utilization of high throughput resequencing data. The primary purpose of the FPGA system we created is to act as an accelerator for the most computationally intensive aspect of an exhaustive resequencing algorithm, namely the alignment of reads by DP to a reference genome. With the throughput attained, we can comfortably keep up with current data production rates for even large genomes. We directly addressed the utility of this new data by enabling analyses that would be impractical using conventional hardware. The enhanced ability of our platform to align divergent sequences results in increased data utility and lower cost. Our platform also opens the door to applications currently unsuited to existing heuristic methods, for example, using the genome from another species as a reference.

Acknowledgments: We thank Venkatesh Akella, Dan Burke, Chen Chang, Charles Langley, Vinayak Nagpal, and John Wawrzynek for their input during this project. This research was supported by NIH R01-HG002942.

REFERENCES

[1] D. Bentley *et al.*, "Accurate whole human genome sequencing using reversible terminator chemistry," *Nature*, vol. 456, no. 7218, p. 53, 2008.
[2] J. Bowers *et al.*, "Virtual terminator nucleotides for next-generation DNA sequencing," *Nat Meth*, 7 2009.
[3] M. Margulies *et al.*, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, p. 376, 2005.
[4] D. Gusfield, *Algorithms on strings, trees, and sequences*. Cambridge Univ Press, 1997.
[5] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Chapel Hill Conference on VLSI*, pp. 363–376, 1985.
[6] D. Hoang, "A systolic array for the sequence alignment problem," *Technical Report, Dept. of Computer Science, Brown University*, 1992.
[7] T. Ramdas and G. Egan, "A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology," *Proceedings of TENCON*, pp. 1–6, 2005.
[8] L. Hasan, Z. Al-Ars, and S. Vassiliadis, "Hardware acceleration of sequence alignment algorithms-an overview," in *Proc. Design & Technology of Integrated Systems in Nanoscale Era*, 2007, pp. 92–97.
[9] C. Yu *et al.*, "A Smith-Waterman systolic cell," in *Proc. 13th Int. Conf. on Field-programmable Logic and Applications*, 2003, pp. 375–384.
[10] C. Chang, J. Wawrzynek, and R. Brodersen, "BEE2: a high-end reconfigurable computing system," *IEEE Design & Test of Computers*, vol. 22, no. 2, pp. 114–125, Mar 2005.
[11] H. K.-H. So and R. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH," *ACM Trans. on Embedded Computing Sys.*, vol. 7, no. 2, pp. 1–28, 2008.
[12] H. So, "Borph: An operating system for fpga-based reconfigurable computers," Ph.D. dissertation, University of California, Berkeley, 2007.
[13] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, pp. 156–161, 2007.
[14] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.
[15] H. Li and R. Durbin, "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform," *Bioinformatics*, 2009.