# Hardware Acceleration of Short Read Mapping

Corey B. Olson[1,2], Maria Kim[1], Cooper Clauson[1], Boris Kogon[1,2], Carl Ebeling[1], Scott Hauck[1], Walter L. Ruzzo[1,3]
[1]University of Washington, [2]Pico Computing Inc., [3]Fred Hutchinson Cancer Research Center
Seattle, WA
corey@picocomputing.com, {mbkim, cclauson, boko, ebeling, hauck, ruzzo}@uw.edu

***Abstract*—Bioinformatics is an emerging field with seemingly limitless possibilities for advances in numerous areas of research and applications. We propose a scalable FPGA-based solution to the short read mapping problem in DNA sequencing, which greatly accelerates the task of aligning short length reads to a known reference genome. We compare the runtime, power consumption, and sensitivity of the hardware system to the BFAST and Bowtie software tools. The described hardware system demonstrates a 250X speedup versus the original BFAST software version, and a 31X speedup versus Bowtie. Also, the hardware system is more sensitive than Bowtie, which aligns approximately 80% of the short reads, as compared to the 91% aligned by the hardware.**

***Keywords - bioinformatics; short reads; mapping; next-generation sequencing; reconfigurable hardware; FPGA;***

## I. INTRODUCTION

*Next-generation sequencing* (NGS) machines are revolutionizing many aspects of biology and medicine. These machines are dramatically lowering the cost and increasing the throughput of DNA sequencing. Their performance is on a trend line that has outstripped Moore's law for several years, with no end in sight. Furthermore, just as exponential progress in microelectronic technology has opened widespread and unexpected application areas; sequencing technology is being applied to a rapidly widening array of scientific and medical problems, from basic biology to forensics, ecology, evolutionary studies, agriculture, drug discovery, and the growing field of personalized medicine.

In brief, the sequencers determine the nucleotide sequence of short DNA fragments, typically a few tens to hundreds of bases, called *short reads*. This can be done in a massively parallel manner, yielding much higher throughput than older sequencing technologies – on the order of tens of billions of bases per day from one machine. For comparison, the human genome is approximately 3 billion bases in length.

Given the diversity of applications, there is no single workflow used for all NGS applications. However, in a dominant one, the short reads are derived from randomly fragmenting many copies of the genome of one organism for which a *reference* genome sequence is already known. In these cases, the key first step in the data analysis pipeline is the *short read mapping problem:* determining the location in the reference genome to which each read maps best. The problem is technically challenging for two reasons. First, speed is important simply due to the volume of data. For example, in human genetic studies, mapping a billion reads from one subject to the human reference genome is routine. Second, the achieved sensitivity of the algorithm, which is the ability to successfully map sequences that are not completely identical to the reference, is an important consideration. These differences exist both because of technical errors in the sequencing machines (a frequent and uninteresting case) and because of genetic differences between the subject and the reference genome. The latter case is rarer and much more interesting – indeed it may be the entire purpose of the experiment, as it may reveal the cause of some genetic disease. The cases are distinguishable because the sequencer errors are random while the genetic differences are not. Hence many mapped reads that consistently exhibit a difference with respect to the reference at a particular locus signal a genetic change, whereas occasional scattered differences are probably errors. This also drives the desire for more and more reads, since more data gives more accurate variant-calling. Figure 1 shows an example set of short reads mapped to a section of the reference genome and examples of both types of differences (sequencing errors and genetic variations).

Short read mapping has traditionally been performed by software tools such as Bowtie [1], BWA [2], MAQ [3], and BFAST [4], running on a cluster of processors. However, NGS improvements are moving the bottleneck of the genome sequencing workflow from the sequencing phase to the short read mapping software.

We describe using FPGAs to accelerate the short read mapping process by exploiting the parallelism of the task.

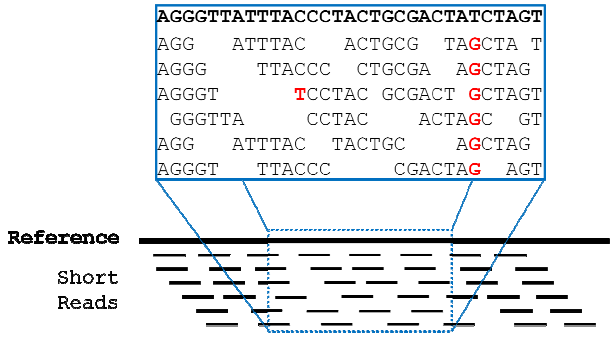


**Figure 1: Short read mapping aligns reads to a reference genome in the presences of errors and genetic variations. The highlighted 'T' is a sequencing error, whereas the highlighted 'G' is a genetic variation.**

The increased performance can be used to speed up genome sequencing, or to allow the use of more sensitive short read mapping algorithms, which are able to map a greater percentage of the reads to the reference genome.

## II. Related Work

### A. Software Short Read Mapping

There are many short read mapping software tools that tackle the problem of processing the enormous amount of data produced by the next-generation sequencing machines. These solutions tend to fall into two main algorithmic categories.

The first category of solution is based upon a block sorting data compression algorithm called the *Burrows-Wheeler Transform* (BWT) [5]. This solution uses the FM-index [6] to efficiently store information required to traverse a suffix tree for a reference sequence. These solutions can quickly find a set of matching locations in a reference genome for short reads that match the reference genome with a very limited number of differences. However, the running time of this class of algorithm is exponential with respect to the allowed number of differences; therefore BWT-based algorithms tend to be less sensitive than others. Bowtie [1] and BWA [2] are examples of programs based upon this algorithmic approach.

The second category of solution leverages the fact that individual genomes differ only slightly, meaning it is likely that some shorter subsequences of a short read will exactly match the reference genome. These shorter subsequences are called *seeds*. An *index* of the reference genome is compiled first, which maps every seed that occurs in the reference genome to the locations where they occur. To align a short read, all the seeds in the read are looked up in the index, which yields a set of *Candidate Alignment Locations (CALs)*. The short read is then scored against the reference at each of these CALs using the Smith-Waterman [7] string-matching algorithm. The location with the highest score is chosen as the alignment location for a short read. BFAST [4] is an example program based upon this algorithmic approach.

### B. FPGA-based Short Read Mapping

Multiple attempts have been made to accelerate short read mapping in hardware, but sequencing flows generally have yet to adopt the use of FPGAs. Previous efforts doing short read mapping using FPGAs have achieved at most an order of magnitude improvement compared to software tools. Also, previous solutions failed to produce a system that is well-suited to large-scale full-genome mapping.

Two attempts to accelerate short read mapping on FPGAs tried to use a brute-force approach to compare short sequences in parallel to an entire reference genome. They stream the reference genome through a system doing exact matching of the short reads to the reference [8] [9]. Reference [8] demonstrates a greater sensitivity to genetic variations in the short reads than Bowtie and MAQ, but the mapping speed was approximately the same as that of Bowtie. Also, this system demonstrated mapping short reads to only chromosome 1 of the human genome. Reference [9] demonstrates between 1.6x and 4x speedup versus RMAP [10] and ELAND [11] for reads with between 0 and 3 differences. This implementation was for the full human genome.

In both implementations, the number of short reads that can be aligned in a single pass of the reference genome is limited by the number of block RAMs on the FPGA. Scaling to a larger number of short reads (the previously cited works mapped only 100,000 50-base and 1,000,000 36-base reads respectively) would require multiple passes of the reference genome. This would greatly increase the runtime of the system.

## III. System Design

Our system design is based upon the algorithm used by the BFAST mapping software. Our current implementation uses seeds with 22 base-pairs, short reads with 76 base-pairs, and 2 bits per base encoding, but these are parameters which can be changed easily.

### A. Creating the Index

A pre-compiled index is used to map seeds of a short read to locations in the reference genome where the seed occurs. The index is implemented as a modified hash table, composed of a *pointer table* and a *CAL table*. The pointer table is directly indexed using only a portion of the seed. This means that many seeds map to each entry of the pointer table. The CALs for these seeds are stored along with the corresponding seed as a list in a second table called the CAL Table. Each entry in the pointer table contains a pointer to this list along with the size of the list. To look up the CALs for a seed, the pointer table is consulted to find the beginning and end of the relevant bucket in the CAL table. This bucket is searched for the CALs associated with the seed. Thus, looking up the CALs for a seed takes 2 DRAM reads, one for retrieving information from the pointer table, and another to bring in the actual CAL table bucket to be searched.

To compile the index for a reference genome, we walk along the reference genome and make a list of all the <seed, location> pairs in the genome. This list is then sorted by seed, and seeds occurring too frequently in the reference genome are removed. Removing these abundant seeds allows us to avoid aligning reads to repetitive regions of the reference genome without negatively affecting the quality of the alignments. We mirrored the BFAST default for this by removing seeds that occur more than 8 times in the reference genome. This sorted list comprises the *CAL table.* To complete the construction of the index, we create the previously described pointer table. An example of a fully constructed index for the reference string AGTACTGCGA can be seen in Figure 3.

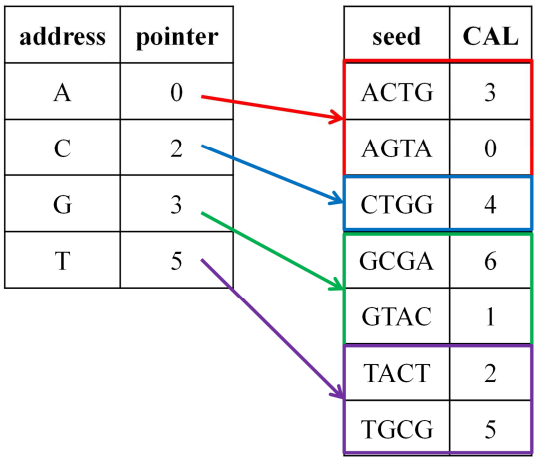


**Figure 3: The index consists of the pointer table (left) and the CAL table (right). The pointer table is an array of pointers to the start of hash table buckets in the CAL table. This example is constructed for the reference AGTACTGCGA.**

### B. *Finding CALs*

The first step in mapping reads to the reference genome is to extract all seeds from each short read. We have chosen the seed length to match the BFAST default seed length, which is 22 bases. To find CALs for a seed, we use the most-significant bits of the seed, which we call the *address* bits, to access the pointer table. The remaining bits of the seed are called the *key* bits. This yields a pointer to the correct CAL table bucket in the CAL table. We do a linear scan through the CAL table bucket, and CALs of entries with key bits matching those of the current seed are added to the set of CALs for the current short read. An example index access for a seed is shown in Figure 2 with the visited entries of the pointer table and CAL table highlighted.

The index must also address the problem that a short read might come from either the forward or reverse complement strand of the double-stranded DNA. There are three options for mapping reads to both the forward and the reverse complement strands.

The first option is to index only the forward reference strand, and lookup both the short read and its reverse complement in the index of the forward genome. This solution doubles the number of reads to be mapped and would thus double the runtime.

The second option is to index both the forward and the reverse complement genomes. Looking up the short read in this index would find the CALs for both the forward and the reverse complement genome. However, this index would require approximately double the memory footprint as compared to an index for just the forward reference.

Instead, we notice that each seed has two entries in an index constructed for both the forward and reverse complement genomes; one entry is for the forward strand and one for the reverse complement strand, but these entries have exactly the same location. Thus, we only need to keep one entry if we add a bit that indicates whether it is the entry for the forward or the reverse strand. When creating the index, we only add the lexicographically smaller of the forward and the reverse complement seeds to the index. An exception is made for seeds that are their own reverse complement. In this case both the forward and reverse complement seed must remain in the CAL table.

Now when looking up seeds in the index, we generate both the forward and reverse complement of each seed. We choose the lexicographically smaller of the two and use it to access the pointer table and CAL table. If the forward seed was used to access the index and it matches a CAL from the forward strand, or a reverse seed was used and it matched a CAL from the reverse strand, we do Smith-Waterman comparison on the forward reference. In the other cases, where only one of the seed and CAL were reverse-complemented, we compare the read to the reverse complement of the reference.

One problem that arises when deterministically keeping the lexicographically smaller seed is a non-uniform distribution of the CALs in the CAL table buckets. Since we keep the lexicographically smaller version of the seed, we will tend to have more seeds that begin with 'A' than seeds that begin with 'T'. This bias would cause the CAL table buckets towards the beginning of the CAL table to contain many more CALs than buckets at the end of the CAL table.

To address this issue, we redistribute the CALs throughout all CAL table buckets during the construction of

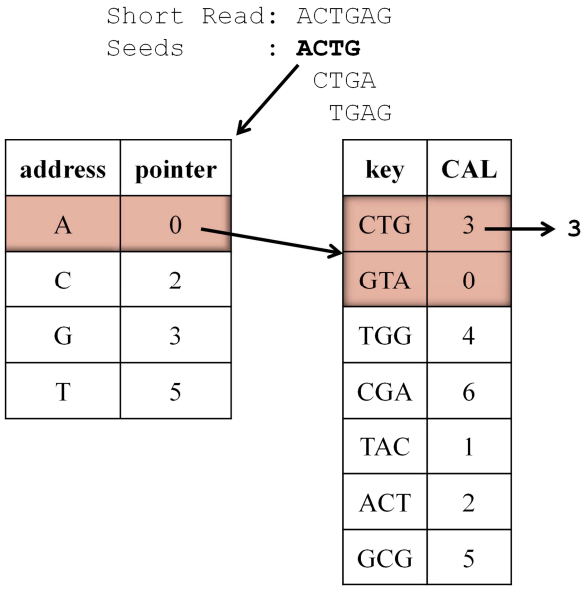


**Figure 2: An example of the pointer table and CAL table entries accessed for the seed ACTG. Note that the key CTG in the CAL table bucket matches that of the seed, so 3 is a valid CAL for this seed. Note that this example does not include the reverse complement strand.**

the index by hashing the lexicographically smaller seed before adding it to the table. We hash the seeds using a 1-1 hash function on the 44-bit seed. We then use the most significant 30 bits of the result to address the pointer table. We store the remaining 14 bits, which are sufficient to identify the seed, amongst the CALs in the CAL table. This produces a more uniform distribution of CALs through all CAL table buckets. During the mapping phase, we use the same hash on the lexicographically smaller seed before accessing the pointer table and CAL table.

### C. Binning and Filtering

Thus far we have described how to find the candidate alignments of seeds in a reference genome using an index. We must convert these locations, which give the location of the start of the seed, to the candidate location of the read containing that seed. We do this normalization by subtracting from the CAL the offset of the start of the seed with respect to the start of the short read. For example, if the seed starting at the third base of the short read finds a CAL of 34, we offset that CAL back to the start of the short read, location 32. This normalization gives us the exact location of the read in the reference if we assume that the genome being sequenced has no insertions or deletions with respect to the reference genome. To allow for up to N insertions and deletions, we must compare the short read to a section in the reference genome starting N bases before start of the short read's CAL (given by the index) and ending N bases after the end of the short read.

To simplify the hardware, we have divided the reference genome into *reference blocks* whose size is determined by the size of a single read from DRAM. All CALs are converted to the start of the reference block that contains the CAL, and the short read is aligned to the reference starting at the beginning of a block. This increases the size of the reference sequence compared against the short read, but the increase in time for this comparison is offset by much simpler hardware.

Many CALs are collected for a short read when looking up its seeds in the index. However, many of these CALs will refer to the same place in the reference, and the short read only needs to be compared once at each of these locations. For example, our system with 22-base seeds in 76-base short reads does 55 (76-22+1) index lookups, so an exact match to the reference will find the same CAL 55 times. We remove repeat CALs from the set for a short read by saving the CALs, along with their reverse complement bit, in a hash table (called the *CAL Filter*) as they are processed. If an entry already appears in the filter table, it is ignored. Each entry also stores a sequence number associated with each short read. This is incremented for each new short read, which implicitly clears the hash table between short reads.

### D. Smith-Waterman

The short read is then aligned to the reference genome at each CAL. This local alignment uses a combination of the Smith-Waterman and Needleman-Wunsch [12] string-matching algorithms. Instead of doing local alignment of a short read against the reference, or global alignment of the short read and reference, we globally align the short read against a local section of the reference. In other words, we want to score the alignment of the entire short read to a subsequence of the reference genome.

We use the affine gap model [13], which scores long insertion or deletion chains different from short insertions and deletions. In this model, the cost to begin an insertion or deletion chain is higher than the cost to continue an insertion or deletion chain. This model allows for longer insertion and deletion chains to appear in the actual alignment, which can be more biologically accurate. However, this model requires that we store 3 copies of the 2D scoring array. The first copy is the similarity score, and the other two are the current insertion and deletion scores at each cell respectively.

When using the affine gap model, the score for a cell is computed by solving (3), where $\alpha$ is a positive value that is the gap-open penalty, $\beta$ is a positive value that is the gap-extension penalty, $E(i,j)$ is the current read gap score for cell $(i,j)$, $F(i,j)$ is the current reference gap score for cell $(i,j)$, $\sigma(S[i],T[j])$ is the positive or negative score from the similarity matrix for read base $i$ and reference base $j$, and $V(i,j)$ is the final similarity score for cell $(i,j)$.

$$\mathrm{E(i,j)} = \max\begin{cases} V(i,j-1) - \alpha \\ E(i,j-1) - \beta \end{cases} \tag{1}$$

$$\mathrm{F(i,j)} = \max\begin{cases} V(i-1,j) - \alpha \\ F(i-1,j) - \beta \end{cases} \tag{2}$$

$$\mathrm{V(i,j)} = \max\begin{cases} E(i,j) \\ F(i,j) \\ V(i-1,j-1) + \sigma(S[i],T[j]) \end{cases} \tag{3}$$

The matrix is computed by sweeping $i$ from 1 to the length of the short read (N), and by sweeping $j$ from 1 to the length of the reference (M) being compared. Our scoring matrix is based upon that of BFAST, with $\alpha$ set to +2, $\beta$ to +1, and the similarity matrix $\sigma$ set such that matching bases are +2 and mismatching bases are -2. To guarantee that we globally align the short read to the local reference, we initialize the scoring matrices ($V$, $E$, and $F$) according to (4).

$$\mathrm{V(i,j)} = \begin{cases} 0 & i = 0, 1 \le j \le M \\ -\alpha - (i-1) * \beta & 1 \le i \le N, j = 0 \end{cases} \tag{4}$$

An example of a fully aligned read and the backtracking process can be seen in Figure 4.

| READ \ REFERENCE | - | A | G | T | A | C | T | G | C | G | A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | -2 | 2 | -2 | -2 | 2 | -2 | -2 | -2 | -2 | -2 | 2 |
| C | -3 | 0 | 0 | -2 | 0 | 4 | 2 | 1 | 0 | -1 | 0 |
| T | -4 | -1 | -2 | 2 | 0 | 2 | 6 | 4 | 3 | 2 | 1 |
| G | -5 | -2 | 1 | 0 | 0 | 1 | 4 | 8 | 6 | 5 | 4 |
| A | -6 | -3 | -1 | -1 | 2 | 0 | 3 | 6 | 6 | 4 | 7 |
| C | -7 | -4 | -2 | -2 | 0 | 4 | 2 | 5 | 8 | 6 | 5 |

**Figure 4: This example shows the similarity score matrix for the global alignment of the short read to a local section of the reference. The short read matches the reference perfectly with the exception of a single base "A". Only the similarity matrix of the affine gap model is shown here, but the insertion and deletion matrices are included in the computation.**

## IV. Hardware System

The target for this implementation was the full 3B base-pair human genome. Also, the read length was targeted as 76 bases, but both of these parameters can be tuned with slight changes to the design. With the exception of the memory system and the Smith-Waterman engines, the system operates on a 250 MHz source clock.

### *A. System Description*

The short read mapping system consists of multiple M-503 modules from Pico Computing [14]. Each M-503 contains a Xilinx Virtex-6 LX240T FPGA. It uses Gen2 x8 PCIe for external communication and contains 2x4GB DDR3 SODIMMs running at 400 MHz. The M-503 module connects to an EX-500 backplane, which plugs into a motherboard x16 PCIe slot. Three M-503s can plug into the EX-500 backplane, and they are connected together using a full-duplex x8 switched PCIe network. Eight EX-500s can plug into a motherboard, allowing up to 24 M-503s in a chassis. M-503s can send and receive traffic external to the EX-500 backplane using full-duplex Gen2 x16 PCIe.

Before the mapping begins, the index is streamed from the host system to the DDR3 of the M-503 via the PCIe, which takes roughly 72 seconds. If aligning multiple sets of reads to the same reference genome, which is common in a production setting, this step must only be done the first time.

Once the short read mapping phase begins, packets containing the short reads are streamed to the hardware system via PCIe. The hardware extracts each short read from the stream, and uses a shift register to extract all the seeds for the short read. The short read is then passed to the Smith-Waterman unit for use in the comparison step.

Three ports to the DDR3 system, operating on a 200 MHz clock, are required for the mapping. Each access to the memory system, whether to the pointer table, CAL table, or reference data, is to a random address. Heavy pipelining and out-of-order memory accesses are therefore required to attain the highest possible memory bandwidth for these memory accesses. The M-503 uses x64 DDR3 SODIMMs, which return at least 256 bits of data per read burst.

We efficiently pack the pointer table data into these 256-bit boundaries. Each CAL table entry requires 64 bits, for a total of four entries per 256-bit boundary of the DRAM. The reference is packed two bits per base, yielding 128 reference bases per DRAM boundary. After filtering out seeds that appear more than eight times in the reference genome, the total CAL table contains 2.44 billion entries, which requires 19.5 GB of DRAM to store the CAL table. The packed pointer table requires 2 GB of memory and the reference genome is stored in 0.77 GB of memory.

A CAL filter is implemented using block RAMs on the FPGA. The filter is implemented as a hash table with 1024 entries. CALs are therefore hashed to 10 bits to determine the target entry of the table. The CAL finding and filtering process all operates on a 250 MHz system clock.

The Smith-Waterman engines are implemented in the FPGA as a systolic array [15], as shown in Figure 5. Each cell computes the value of one entry in the 2D Smith-Waterman matrix each clock cycle. An array of cells computes the anti-diagonal of the Smith-Waterman matrix in parallel. In other words, the systolic array enables each base of the short read to be scored against a base of the reference simultaneously, which changes the runtime of the Smith-Waterman computation from O(MxN) for M length reference and N length short read to O(M+N).

Due to the many levels of logic in the unit cell of the systolic array, the Smith-Waterman system runs at a slower 125 MHz clock. One benefit of the systolic array is the

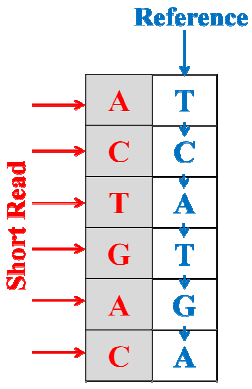


**Figure 5: Smith-Waterman compute unit is implemented as a systolic array. The short read is parallel-loaded before the alignment, and the reference bases are shifted through the array. All bases of a short read are scored against bases of the reference each clock cycle.**

ability to pipeline the alignment of a single short read to multiple reference buckets, which reduces the runtime to approximately equal the length of the reference bucket (M), instead of the reference bucket plus the read length (M+N). However, as described in the next section, we have partitioned the CAL table eight ways and are using a fairly long seed length, so there are approximately only 1.5 CALs per read in a given partition. This means we will not be able to pipeline many reference alignments.

Reference data is packed to 256-bit boundaries in DRAM, which should mean that we can align 76 base reads against 128 bases of reference. However, the short read alignment often will cross the 256-bit boundary of the DRAM. Therefore, we align the read to a 256-base (2 x 256 bits) section of the reference. When aligning against a 256-base reference bucket a single Smith-Waterman computation requires approximately 76+256=332 cycles. Since each computation takes so many cycles to complete, we instantiate multiple Smith-Waterman compute engines in the hardware, and round-robin the required computations among the instantiated compute units. The "right" number of Smith-Waterman engines to instantiate should be just enough to keep up with the pointer table, CAL table, and reference memory accesses. In our system, the memory interface (which benefits from partitioning and is explained later) requires approximately 45 cycles at the 200 MHz memory clock to complete the memory accesses for a single short read. This means approximately 18 Smith-Waterman engines are required to keep up with the memory system. A high-level block diagram of the described hardware system for the short read mapping system on a single M-503 is shown in Figure 6.

### *B. Partitioning*

To improve the performance of the system, we partition the index across multiple memories to allow concurrent memory operations and reduce the memory footprint required per node in the final system. To partition our index *N* ways, we partition both the pointer table and the CAL table by the first $log_2(N)$ bits of the seed (after the seed has been hashed). For example, when partitioning two ways, seeds in the index that begin with an A or C go into the index for partition 0, and seeds beginning with G or T go into the index for partition 1.

Each short read is streamed through each of the partitions. Each partition looks up only those seeds that occur in its partition of the index. Using the example above with two partitions, a seed beginning with an A will only be looked up by partition 0. Each partition collects the CALs for the short read and performs the Smith-Waterman comparison to the reference. When finished, it passes the short read along with the best alignment and score to the next partition. Each partition updates the best alignment and score in turn, with the last partition producing the final best alignment and score.

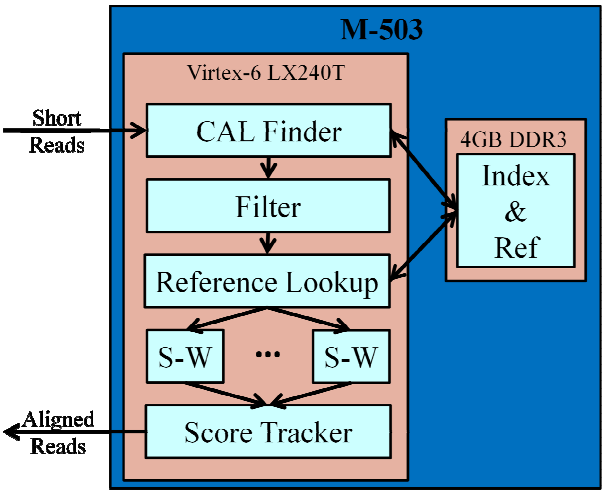


**Figure 6: High level view of the short read mapping system hardware on a single M-503 FPGA card. Short reads are streamed to the FPGA and aligned reads are streamed from the FPGA via PCIe. The index is stored in a 4 GB DDR3 memory, which is accessed by the CAL finding module and the reference lookup module. Multiple Smith-Waterman compute engines are instantiated to align reads in parallel.**

This aggregation is accomplished by creating a daisy-chain of the partitions using PCIe for communication. A partition replaces the best alignment information on the short read stream if it aligns that short read with a higher score than the current best alignment score. Short reads are passed along the daisy chain between FPGAs without the need for host processor control. The processor simply needs to send short reads to the first FPGA in the chain and receive results from the last FPGA in the chain.

Each read requires 32 B of data to be streamed through the system. Therefore, the total amount of data that must be streamed through the chain for 50 million reads is 1.5 GB. Gen2 x8 PCIe, which has a maximum sustained bandwidth of 3.2 GB/s, requires about 0.5 seconds to stream all the data for 50 million reads. Since the network is full duplex, reads can be streamed to the first FPGA and results streamed from the last FPGA concurrently.

The human genome requires an index of approximately 22 GB plus the reference data for each partition. By partitioning the index across 8 M-503s, each of which having 4 GB of DDR3 DRAM, we easily store the entire index and reference data. Each partition contains 0.25 GB of the pointer table, 0.77 GB of the reference data, and approximately 2.5 GB of the CAL table.

Two small problems exist with this partitioning method. First, the entire reference must now be stored with each partition. This is not a problem for the human genome, which can be stored in about 750 MB, but could become a problem for much larger genomes. Second, since the partitioning is not done by section of the genome, a short read may be aligned against the same reference bucket in multiple partitions. This inefficiency can be avoided by seeding the CAL filter at the beginning of each short read

with all of the CALs that have previously been aligned for that short read. This increases the data passed between partitions, but greatly reduces the number Smith-Waterman comparisons. However, this has yet to be implemented.

More details regarding our full human genome short read mapping implementation can be found in [16] and [17].

## V. Results

We compare the performance of this eight FPGA system, contained in a 4U Tyan chassis, to Bowtie and BFAST running on the two quad-core Intel Xeon E-5520 CPUs. To test the runtime and sensitivity of alignments, these software and hardware mappers were tested using 76 base short reads. Our benchmarks consisted of mapping subsets of 54 million short reads from one end of a paired-end Illumina GA IIx run on human exome data.

The tested system consumed 29% of the available slice registers, 62% of the slice LUTs, and 45% of the block RAMs on an LX240T-2 Virtex-6 FPGA. Due to routing congestion at the inputs and outputs of the Smith-Waterman modules, each FPGA was limited to eight Smith-Waterman compute engines.

Although our goal was to develop a high-performance implementation of the high-quality BFAST algorithm, we also compared the runtime of the FPGA system to that of the Bowtie software, which is a heuristic algorithm that trades quality for improved performance. BFAST and Bowtie were each tested using 8 threads, which was chosen because that number of threads saturated the memory bandwidth on the system and produced the fastest runtime for mapping 50 million reads, as shown in Figure 7. When measuring the runtime of the FPGA system, we do not include the time required to populate the index into memory and to configure the FPGAs, since this only needs to be done once per reference genome. However, when measuring the runtime of the BFAST and Bowtie software, we do include the time to load the index into memory, since it must be done for every set of short reads to be mapped.

The following are the commands that were used to run the matching local alignment phases of the BFAST software along with the command use to run the Bowtie software (after compilation of an index for both software tools):

- bfast match -f hg19.fasta -r reads1.fastq -A 0 -e 50000000 -w 0 -n 8 –t > reads1.bmf
- bfast localalign -f hg19.fasta -m reads1.bmf -A 0 -e 50000000 -n 8 -t > reads1.baf
- bowtie hg19 -q reads1.fastq -u 50000000 -v 3 -t -p 8 > reads1.bowtie_out

### A. Mapping Time

To demonstrate the speed of the FPGA system, we compared the time to map 50 million short reads using the FPGA system versus BFAST and Bowtie, as shown in Figure 7. The results show the FPGA system with a 250x speedup over BFAST, which mapped 50 million reads in 8,480 seconds (2 hours, 21 minutes, 20 seconds). Also the results show a 31x speedup over Bowtie, which mapped 50 million reads in 1,051 seconds (17 minutes, 31 seconds).

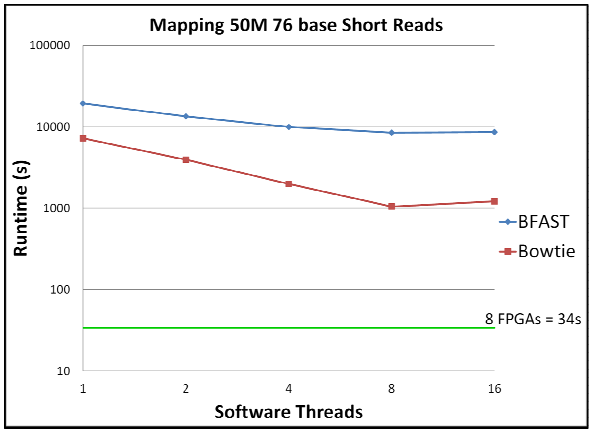


**Figure 7: Time required for the 8 FPGA system, BFAST, and Bowtie map 50 million 76 base short reads to the genome. Note the linear improvement in runtime until the memory system saturates at approximately 8 threads. Note the FPGA system here always uses 2 software threads.**

### B. Sensitivity

In addition to runtime, a key aspect to take into account is the sensitivity of each system. The FPGA system is able to map 91.5% of the 50 million reads, whereas Bowtie only maps 80.2% of the short reads. Bowtie is unable to handle reads that differ from the reference genome by more than three bases. Even when using the "best" option while running Bowtie (which slows down the runtime), only 85.8% of the reads were mapped. The differences between the reference genome and reads are often the data of most interest in a next-generation sequencing experiment, perhaps identifying the location of a genetic variant responsible for a heritable disease. Thus, the increase in sensitivity afforded by the FPGA system is of considerable importance.

The FPGA system can be tuned to be more sensitive by reducing the seed length. This finds more CALs per read, enabling more reads to be aligned to the reference genome, but it slows down the runtime of the system slightly. When testing species with high rates of genetic variation, or using the reference genome as a template for the assembly of reads from the genome of a closely related species, this option may be more important. Conversely, the Bowtie software cannot be tuned to compete with the sensitivity of the FPGA system.

### C. Energy

The previous sections stated the speed and sensitivity comparisons between the hardware and the software. However, power is another factor that must be considered. We measured the power drawn by the system while running the FPGA system, as well as the Bowtie and BFAST software, which is shown in the first row of Table 1.

TABLE 1: ENERGY REQUIRED TO MAP 50M READS

| | BFAST | Bowtie | FPGA |
|---|---|---|---|
| System Power (W) | 249 | 336 | 496 |
| 50M Reads (kW-hr) | 0.587 | 0.098 | 0.005 |
| Normalized Energy | 100% | 16.7% | 0.8% |

The FPGA system consumes more instantaneous power than the software, but it also maps reads much faster than the software. Therefore the FPGA system consumes much less energy when mapping 50 million short reads as compared to either software tools, as shown in Table 1. The normalized energy in the table is the energy required for the specified system (BFAST, Bowtie, or FPGA) normalized to the energy required for the BFAST system. The table shows that the FPGA system requires only about 0.8% of the energy required for the BFAST system and 4.7% of the energy required for the Bowtie system.

## VI. CONCLUSIONS

These results demonstrate the power of this FPGA short read mapping system. This system is able to map reads to the full genome faster, while using significantly less energy than both of the two tested software tools. We demonstrated a two order of magnitude speedup (250X) versus the BFAST software, and a one order of magnitude speedup (31X) versus the lower-quality Bowtie software running on two quad-core Intel Xeon processors.

These improvements in system runtime help to enable research that was not possible before. For example, researchers can use this to study sequencing technologies with higher read error rates, which require a very large read depth during mapping. Similarly, this technology enables researchers to study larger genomes that may not have been possible before, such as barley (5.3 Gbases). Also, this FPGA hardware system can be tuned for the speed versus sensitivity trade-off, which enables researchers to study genomes with large genetic variation among the species.

## ACKNOWLEDGMENT

We would like to thank the Washington Technology Center (WTC) and Pico Computing for sponsoring this research. We would also like to thank Professor Deborah Nickerson for providing the sample short read data used in our analysis.

## REFERENCES

[1] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, March 2009.

[2] Heng Li and Richard Durbin, "Fast and accurate short read alingment with Burrows-Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754-1760, July 2009.

[3] Heng Li, Jue Ruan, and Richard Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores, " *Genome Research*, vol. 18, no. 11, pp. 1851-1858, November 2008.

[4] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: An Alignment Tool for Large Scale Genome Resequencing," *PLoS ONE*, vol. 4, no. 11, p. e7767, November 2009.

[5] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digital Equipment Corporation, Palo Alto, CA, Technical report 124, 1994.

[6] Paolo Ferragina and Giovanni Manzini, "Opportunistic Data Structures with Applications," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, Washington, DC, 2000, p. 390.

[7] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195-197, March 1981.

[8] O. Knodel, T. B. Preusser, and R. G. Spallek, "Next-generation massively parallel short-read mapping on FPGAs," in *2011 IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2011, pp. 195-201.

[9] Edward Fernandez, Walid Najjar, Elena Harris, and Stefano Lonardi, "Exploration of Short Reads Genome Mapping in Hardware," in *2010 International Conference on Field Programmable Logic and Applications*, Milano, 2010, pp. 360-363.

[10] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of Solexa Read Mapping," *BMC Bioinformatics*, vol. 9, no. 128, pp. 1471-2105, February 2008.

[11] O. Cret, Z. Mathe, P. Ciobanu, S. Marginean, and A. Darabant, "A hardware algorithm for the exact subsequence matching problem in DNA strings," *Romanian Journal of Information Scient and Technology*, vol. 12, no. 1, pp. 51-67, 2009.

[12] Saul B. Needleman and Christian D. Wunsch, "General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," Journal of Molecular Biology, vol. 48, no. 3, pp. 443-453, March 1970.

[13] Peiheng Zhang, Guangming Tan, and Guang R. Gao, "Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform," in Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications, New York, 2007, pp. 39-48.

[14] Pico Computing. (2010) Pico Computing - the FPGA Computing Experts. [Online]. http://www.picocomputing.com/m_series.html.

[15] C. Yu, K. Kwong, K. Lee, and P. Leong, "A Smith-Waterman Systolic Cell," in New Algorithms, Architectures and Applications for Reconfigurable Computing, Patrick Lysaght and Wolfgang Rosenstiel, Eds. United States: Springer, 2005, ch. 23, pp. 291-300.

[16] Corey Olson, *An FPGA Acceleration of Short Read Human Genome Mapping*, University of Washington, Dept. of EE, MS Thesis, 2011.

[17] Maria Kim, *Accelerating Next Generation Genome Reassembly: Alignment Using Dynamic Programming Algorithms in FPGAs*, University of Washington, Dept. of EE, MS Thesis, 2011.