WHITE PAPER

Accelerating Genomics Research with OpenCL™ and FPGAs

Authors

Chris Rauer (Software Engineer), George S. Powley (Software Engineer), Mir Ahsan (Software Engineer), and Nicholas Finamore, Jr. (Sales & Marketing Associate)

Table of Contents

Abstract

With the rapid decrease in gene sequencing costs due to the emergence of second-generation sequencing equipment, the availability of genome sequence data is increasing dramatically. The ability to correlate the variations among genomes is enabling advances in a wide range of medical research and personalized care. Because each human genome comprises more than three billion base pairs, whole genomic sequencing requires significant processing power, storage capacity, and network bandwidth. In particular, variant calling is extremely computationally intensive. The Genome Analysis Toolkit (GATK) is a software package developed at the Broad Institute to analyze high-throughput sequencing data. This paper describes the acceleration of the GATK's HaplotypeCaller algorithm using Intel's field programmable gate array (FPGA) devices, programmed using the Intel® FPGA SDK for OpenCL™ technology.

Keywords: Genome, Genomics, Haplotype, PairHMM, OpenCL, FPGA, Broad Institute, Intel® Arria® 10 FPGA, heterogeneous computing

1 Introduction

Genomic variant discovery may appear to be a straightforward problem that consists of mapping reads to a reference sequence and at every position, counting the mismatches and construing the genotype variants. However, multiple sources of error in the sequence data make this process much more complex than it at first appears. These potential errors include amplification biases that may occur during wet lab preparation, machine errors during library sequencing, and software errors and mapping artifacts during read alignment. "A good variant calling workflow must involve data preparation methods that correct or compensate for these various error modes." [1] These diverse errors make variant discovery a computationally intensive undertaking. Modern variant caller algorithms can take up to several days of computation time using standard microprocessors.

2 Heterogeneous Computing and the OpenCL™ Computing Language

In the field of high-performance computing, heterogeneous computing systems are emerging to solve a wide range of challenges. A standard CPU with an attached accelerator device such as a GPU or field programmable gate array (FPGA) can be used to accelerate a wide range of functions including data search, image processing, and financial or seismic simulations. With the emergence of these heterogeneous systems, programming standards have emerged to allow easier adaptation (and acceleration) of algorithms from standard systems.

OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, digital signal processors (DSPs), FPGAs, and other multicore processors. It includes a language based on standard ANSI C99 for programming these devices, and APIs to control the platform and execute programs on the compute devices. The OpenCL standard is managed by the Khronos Group, a non-profit organization. [2] The portability of programs among accelerator devices from different vendors is a significant advantage of OpenCL. Several vendors, including Intel, AMD, and NVIDIA, provide OpenCL-conformant compilers. In order to claim conformance to the OpenCL standard, the vendor's compiler must accurately compile and execute a suite of more than 8,500 OpenCL programs. [3]

2.1 FPGA Technology

FPGAs are reconfigurable integrated circuits that consist of programmable routing networks linking together logic array blocks, embedded memory blocks, and DSP blocks, in contrast to the fixed data paths and topologies found in CPUs and GPUs that process program instructions. FPGA resources may be configured and linked together to create custom instruction pipelines through which data is processed. "Dynamically creating custom pipelines to process each target application increases throughput, performance, and power efficiency by reducing the amount of superfluous functional units in silicon." [4] The FPGA architecture can be used to solve certain types of computing problems efficiently.

2.2 Programming with FPGAs

Hardware developers have traditionally designed and verified digital circuits on FPGAs at the register-transfer level (RTL) using hardware description languages such as Verilog and VHDL. While these traditional methods are effective for ensuring efficient use of the devices, they are impractical for the implementation of complex algorithms such as gene sequencing. In early 2012, Altera Corporation introduced the Altera SDK for OpenCL, which allows use of the OpenCL programming language to program Intel's FPGAs as computing accelerator devices. The Altera SDK was rebranded as the Intel® FPGA SDK for OpenCL technology following Intel's acquisition of Altera in 2015. In late 2014, Xilinx Corporation, another leading FPGA vendor, announced development of a compiler for OpenCL as well.

2.3 The Intel® FPGA SDK for OpenCL Technology

The Intel FPGA SDK for OpenCL technology has been used for a wide array of algorithms in a variety of computing fields. This SDK uses the same programming model as other vendors' compilers. Figure 1 illustrates the SDK's programming flow. The system requires an FPGA-based card designed for the OpenCL SDK, available from a variety of vendors. No additional RTL-level programming is required; the FPGA is programmed entirely using OpenCL.

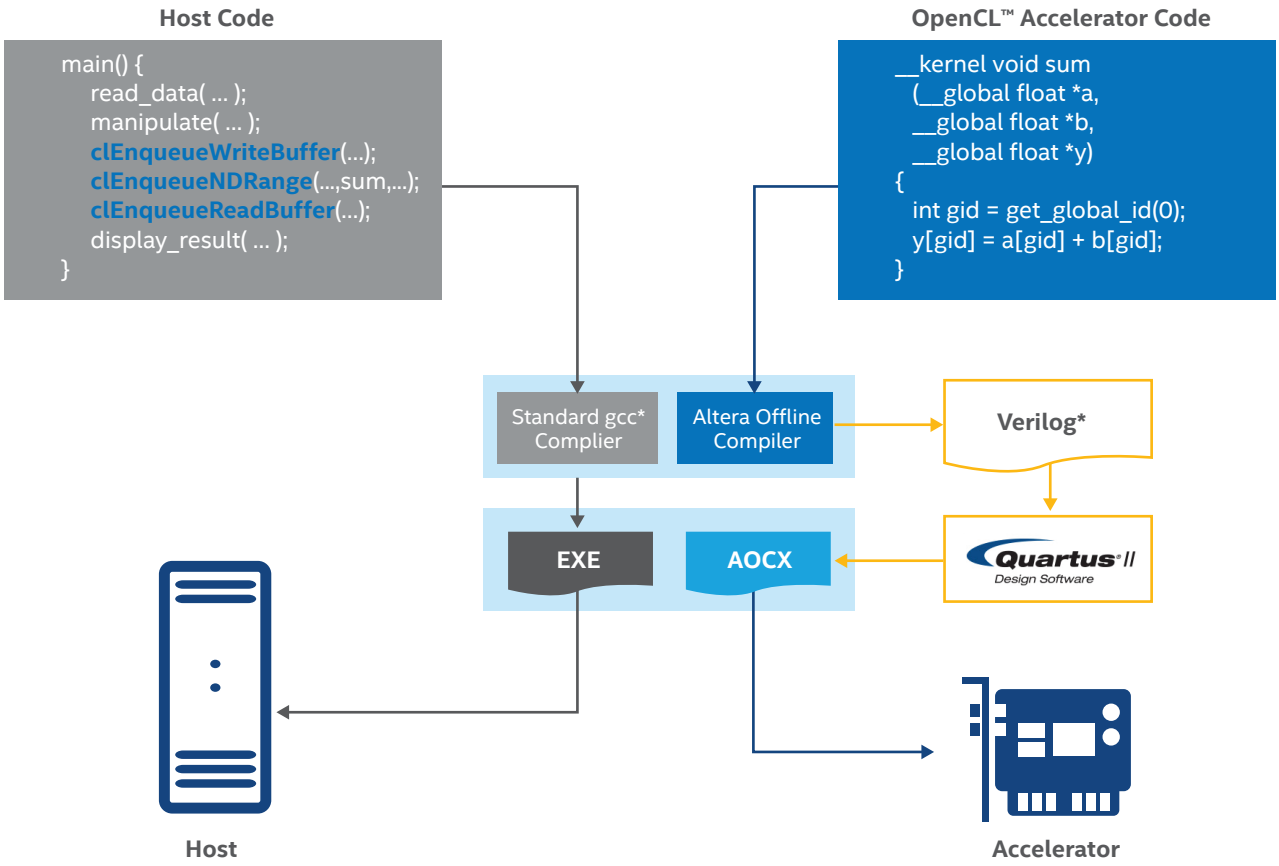

Figure 1. Intel® FPGA SDK for OpenCL™ technology usage model.

The OpenCL program consists of host code that is intended to run on a standard CPU, and the kernel code, which is intended to run on the accelerator (in this case, the FPGA). Using the standard IDE and GCC* compiler, a programmer writes and compiles host code but uses the OpenCL API to communicate with the OpenCL kernel. In a separate .cl file, the programmer writes with OpenCL C following the appropriate optimization guidelines for the FPGA. The OpenCL kernel file is compiled using the Intel FPGA offline compiler and actually runs Quartus* in the background to produce the .aocx file. At runtime, the Intel FPGA offline compiled executable is downloaded to the FPGA. All of the tools and processes typically used by FPGA designers are abstracted away, because this entire process occurs in the familiar software programmer's environment. [5]

2.4 FPGA Device Targets

For the purposes of this experiment, the HaplotypeCaller code is partitioned to run on both the host and the FPGA, to optimize performance. The OpenCL complier was targeted to build the code for an Intel® Arria® 10 FPGA. The Intel Arria 10 FPGA is part of Intel's high-end family of devices and is built using 20nm silicon process technology. This relatively new process technology enables the Intel® Arria® 10 device to have more logic elements, DSPs, and memory, as well as run at higher frequencies than if it were manufactured using older technologies. It has advanced, hardened floating-point elements that make floating-point functions more efficient than those implemented using standard logic. These features make it possible to fit more computational blocks, optimizing performance and performance per watt for the HaplotypeCaller algorithm.

3 Genome Analysis

3.1 Genome Variant Discovery

The process of identifying differences between DNA sequences is called variant discovery. The ability to identify variations in DNA has become essential for medical research and personalized medical care. Research projects that seek to compare hundreds or thousands of sequences are often stifled by the amount of compute time and resources required. Therefore, many in the medical and high-tech communities are pursuing acceleration of variant discovery.

Using a robust calling algorithm that not only compares sequences but also leverages meta-information such as base qualities scores variant discovery can be performed on the appropriately processed data. To minimize errors, it is desirable to include as many potential variants as possible. "Once a highly-sensitive call set has been generated, appropriate filters can be applied to achieve the desired balance between sensitivity and specificity." [6]

3.2 The Genome Analysis Tool Kit (GATK)

The GATK is a software package developed at the Broad Institute to analyze high-throughput sequencing data. It offers a wide variety of tools, with a primary focus on variant discovery and genotyping as well as a strong emphasis on data-quality assurance.

The HaplotypeCaller function within the GATK is the variant discovery algorithm. The main algorithm to compare sequences is called the PairHMM. It is used to call SNPs and indels simultaneously via local re-assembly of haplotypes in an active region. The basic operation of the HaplotypeCaller defines ActiveRegions, determines haplotypes by reassembly of the ActiveRegion, determines likelihoods of the haplotypes given the read data, and assigns sample genotypes. [6]

By using significant variation evidence, the areas to be further analyzed are identified as the ActiveRegions. The program then creates a De Bruijn-like graph to reassemble the ActiveRegions and detect the possible haplotypes, which are then realigned using the Smith-Waterman algorithm. Using the PairHMM algorithm, the ActiveRegions are pairwise aligned against each haplotype to produce a matrix of likelihoods of haplotypes based on the read data. This is then relegated to create the likelihoods of alleles for each potential variant site. [6]

3.3 PairHMM Algorithm Overview

The process of comparing two gene sequences is not as simple as comparing two regular strings, because each sequence may contain insertions, deletions, and mutations. The hidden Markov models in the PairHMM algorithm are used to calculate the probability of a match with these possible changes. Because the exact alignment is not known, a comparison must be performed with each alignment.

The input to the algorithm requires two gene sequences. The first is the read sequence, which contains the gene string and some quality factors based on how it was read in. The second sequence is the haplotype sequence, which is simply a gene string without any additional data. Each sequence is compared with the PairHMM hidden Markov model equation, and the result is passed to the next diagonal. The next diagonal compares the same two sequences again but with a different alignment. The different alignment is just a shift in sequence by one for each diagonal. Figure 2 shows two tiny sequences and the iteration of diagonals with the shifting of the alignment.

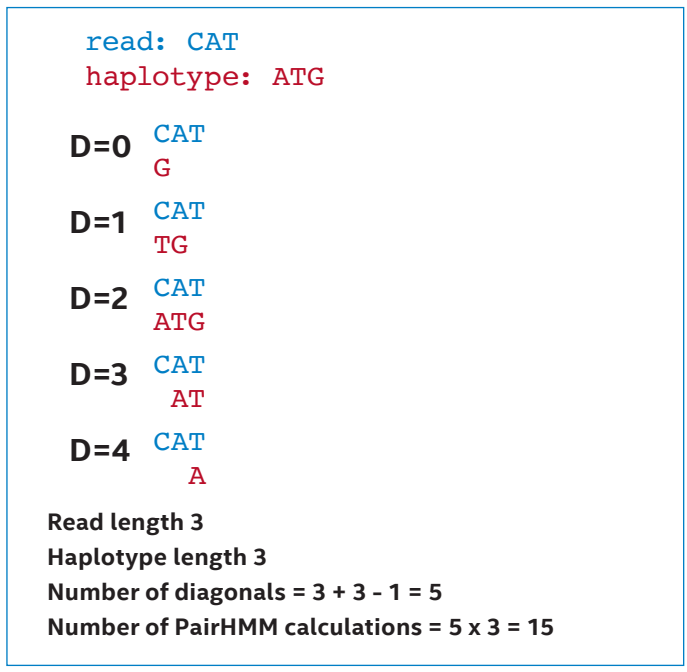

Figure 2. The process of comparing each alignment of two small sequences.

The hidden Markov comparison result is a single probability score, but the floating-point math used is computationally expensive. If the length of each sequence is ***n***, the computation requirement is ***O(n^2)***. Also, each calculation depends on previous calculations from the previous row and diagonal. Figure 3 shows these dependencies. The probability at the end of each diagonal is added up to give an overall score for all the alignments of each sequence. This score determines how well the two sequences match with all alignments. Each box in Figure 3 shows how the result of the PairHMM calculation for two sequence characters is used in subsequent calculations. This type of structure is called a systolic array and readily lends itself to being mapped to an FPGA fabric.

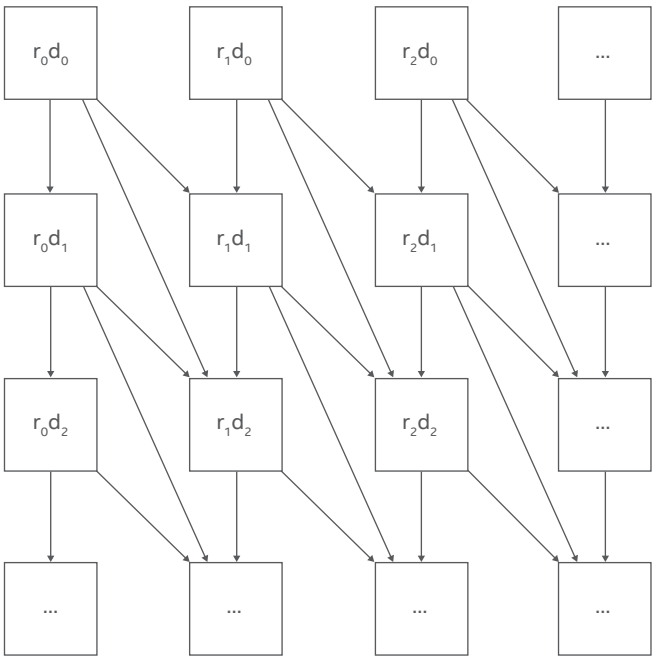

Figure 3. The dependencies among calculations in a Hidden Markov comparison.

3.4 PairHMM FPGA Implementation

This algorithm can be substantially optimized on a CPU using vector instructions. If the sequences are small, the comparison can be done entirely using the CPU's internal Level 1 cache. However, with larger sequences, the external memory bandwidth of the CPU may become a performance limiter, particularly because each calculation must be broken up into separate instructions. These types of algorithms also work well on FPGAs. An FPGA typically has a lower clock speed than a CPU but can take advantage of pipelined parallelism to do hundreds of complex calculations in parallel, running one after another each cycle inside the pipeline. Intel's FPGA SDK for OpenCL compiler analyzes the code and builds these pipelines automatically. The lower clock speed of the FPGA typically enables applications to consume far less power.

4 Experiment

The HaplotypeCaller Algorithm within the GATK was initially written in Java*. The Broad Institute then converted the algorithm to C++. For our experiment, we ported the PairHMM algorithm that was originally written by the Broad Institute [6] from C++ to OpenCL. The fact that OpenCL is a C-based language made it fairly straightforward to port the algorithm. Also, the code was well optimized, and constant values that were needed were already precalculated. The code was tested for functional correctness with the emulator that is part of the Intel FPGA SDK for OpenCL technology, and we used the test cases that came with the Broad Institute's C++ source code.

The code was then targeted to an Intel Arria 10 device on a reference development platform. The compiler produced an optimized .aocx file, which was loaded into the Intel Arria 10 device. Measurements of runtime performance were taken.

4.1 Choosing an Optimal Compute Grid Size

The FPGA accelerates this algorithm effectively because the algorithm can be mapped to a 2D systolic array. In Figure 3, the top computations feed the bottom computations, and the results trickle through the compute units in the grid. Sizing this grid required a bit of experimentation. If the compute grid is too wide, the m20k blocks will increase, because the accesses will be wider, but the depth will be shallower. If the depth needed is less than the physical depth of the m20k block, the block will be underutilized. There are several variables that must be loaded from memory for each column, greatly amplifying this effect.

If the depth of the compute grid is too great, too many haplotype characters will have to be read from DDR memory simultaneously. This will exhaust the DDR memory bandwidth. Also, adders are needed at the end of row. As a result, increasing the depth will increase the number of these adders linearly.

When the OpenCL compiler builds the functions for an Intel Arria 10 FPGA, it is able to fit 208 PairHMM processing elements in an 8x26 grid. This is completely pipelined. Figure 4 shows a visual representation of a smaller 4x4 compute grid. Due to the hardened floating-point capability, along with additional logic, memory, and switching fabric within the more advanced device, the Intel Arria 10 FPGA can fit more processing elements than earlier devices.

4.2 Concatenating Reads and Haplotypes

In addition to optimizing the grid size, the input reads and haplotypes were concatenated to provide the kernel with a constant flow of inputs to minimize the latency of data transfer and global and local memory access. Additional logic was added to the kernel to ensure correct functionality for each read and haplotype pair. The stream of output results was then reassembled in the host to match the expected outcome of the application.

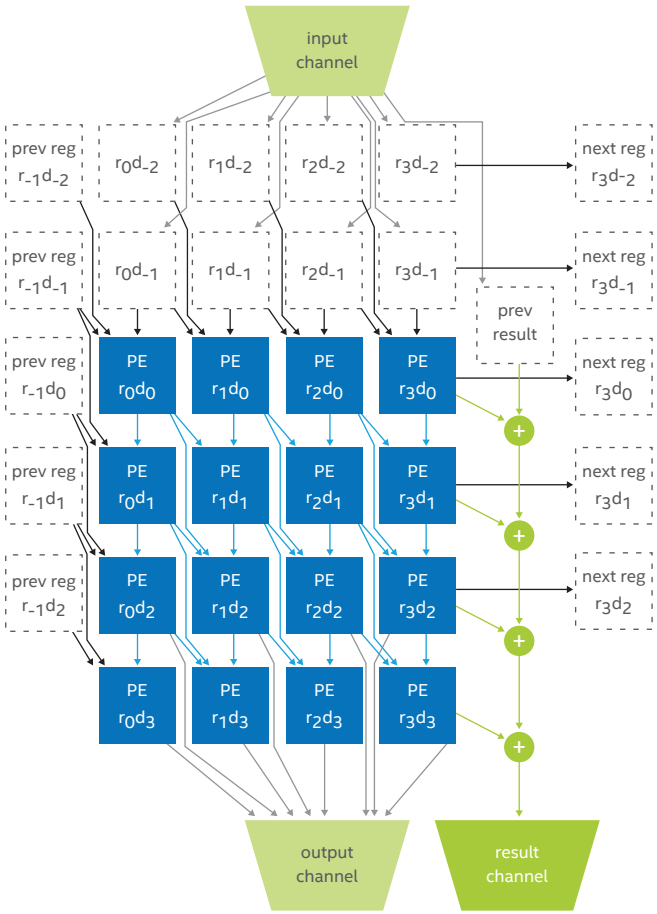

Figure 4. Visual representation of OpenCL™ 4x4 PairHMM algorithm implementation.

5 Results

5.1 FPGA Utilization

Table 1 shows FPGA resource utilization for the Intel Arria 10 device. As expected, the Intel Arria 10 FPGA was able to fit more computation blocks for the PairHMM algorithm because of the hardened floating-point DSP. The DSP utilization was almost 100 percent in the Intel Arria 10 FPGA because of this factor. Maximum frequency (fmax) for the Intel Arria 10 device was 232 MHz.

5.2 Performance

Cell updates per second (CUPS) is a performance measure commonly used in computational biology that represents the number of matrix cells updated each second. [7] For PairHMM, the number of cells in the matrix is the read length multiplied by the haplotype length. Therefore, the performance for one PairHMM calculation is as follows:

(read length × haplotype length) ÷ PairHMM time

Here, PairHMM time includes the time to prepare the data on the CPU, transfer the data to the FPGA, calculate the result on the FPGA, and then transfer the result back to the CPU. In this paper, the results are presented using Giga-CUPS (GCUPS) as the unit.

The functionality and performance measurement of the PairHMM algorithm was performed using data captured while running the GATK HaplotypeCaller application. PairHMM performance on similar whole-genome sequences was tested on multiple platforms. In total, 1,584,272,000 batches were used for computation on the platforms, as outlined in Table 2. The IBM and Xilinx results were presented by IBM Research in 2016. [8] The NVIDIA Tesla* K40 results presented in the third row were published by researchers at the Delft University of Technology. [9]

Table 1. FPGA compilation results for the PairHMM algorithm.

FIELD PROGRAMMABLE GATE ARRAY (FPGA)	PAIRHMM COMPUTE BLOCKS	FMAX (MHZ)	LOGIC UTILIZATION	DIGITAL SIGNAL PROCESSOR UTILIZATION
Intel® Arria® 10 GX FPGA	208	232	236k/427k (55%)	1508/1518 (99%)

Table 2. Performance results comparing various platforms.

IMPLEMENTATION	PEAK PERFORMANCE [GIGA-CUPS (GCUPS)]	AVERAGE PERFORMANCE (GCUPS)
One core [Intel® Advanced Vector Extensions (Intel® AVX) technology] on the Intel® Xeon® processor E5 v4 product family[a]	0.699	0.676
44 cores (Intel AVX technology) on the Intel Xeon processor E5 v4 product family[a]	22.0	21.2
NVIDIA Tesla* K40 [9]	12.79	N/A
Power8* eight-core 8284-22A [8]	0.809	N/A
Xilinx Kintex* Ultra Scale XCKU060 [8]	1.746	N/A
Intel® FPGA SDK for OpenCL™ technology on Intel® Arria® 10 GX FPGA development board[b]	44.43	33.8

[a] System configuration: Intel® Xeon® processor E5-2699 v4 @ 2.20 GHz, 2 sockets/22 cores per socket, 256 GB RAM, 2 TB Intel® SSD Data Center P3700 Series.

[b] System configuration (for last row of Table 2): Intel® Xeon® processor E5-2697 v2 @ 2.70 GHz, 2 sockets/12 cores per socket, 128 GB RAM, 2 TB Seagate HDD ST2000DM001*.

6 GATK Integration

The PairHMM FPGA accelerator has been integrated into the GATK through the Genomics Kernel Library (GKL) library. The GKL provides the GATK with a simple API, which is common for Intel® Advanced Vector Extensions (Intel® AVX) technology, OpenMP* with Intel AVX technology, and FPGA implementations of PairHMM. The GATK includes options to select a specific PairHMM implementation, as well as the capability to select the fastest available PairHMM implementation on the platform. By using the GKL approach, FPGA-accelerated PairHMM was added to the HaplotypeCaller and MuTect2 tools in GATK3 and GATK4 with few coding changes in the GATK.

7 Conclusion

The Intel FPGA SDK for OpenCL technology enabled simple, effective implementation and testing of the PairHMM algorithm for the GATK from the Broad Institute. The Altera FPGA shows significant performance acceleration relative to other technologies. Comparing the peak performance with IBM POWER8* and Xilinx platforms, the Intel Arria 10 device recorded speeds of up of 55x and 25x, respectively. Upon integration with the GATK Best Practices pipeline, the overall pipeline speed-up was 1.2x compared to the Intel AVX technology implementation. Possible future work includes the following:

- **Incorporate the accelerated algorithms** into the complete GATK.
- **Implement compression algorithms** in the FPGA to enable more effective storage and transportation of genome data along with acceleration of analysis engines such as the GATK.
- **Port to the recently announced Intel® Stratix® 10 FPGA** to potentially achieve further performance scaling.

Further optimization of the OpenCL code could be done as well. For instance, in the current design, the result adder chain in Figure 4 is not fully utilized every cycle. An improvement would be to mux one of the adders from the HMM calculation so that the hardware could be shared. This type of optimization, called resource folding, would allow chip area to be reduced so that freed DSP resources could be used to add more computation units, increasing performance.

8 Acknowledgments

The authors wish to thank Andrew Ling, the manager of the OpenCL compiler team in Toronto, and his team for their support with understanding some of the inner workings of the OpenCL compiler. We would also like to thank David Roazen, Louis Bergelson, Lee Lichtenstein, and Eric Banks from the Broad Institute, as well as John Sotir and Richard Yang, for making this project possible.

9 References

[1] Van der Auwera, Geraldine A., et al., "From FastQ Data to High Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline." http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4243306/.

[2] Altera SDK for OpenCL Programming Guide, Version 14.0, Altera Corporation (2014). https://go.altera.com/extranet2001/products/design_software2/opencl/files/aocl_programming_guide_14.0.pdf.

[3] "Altera SDK for OpenCL is First in Industry to Achieve Khronos Conformance for FPGAs," Altera Corporation (2013). http://www.prnewswire.com/news-releases/altera-sdk-for-opencl-is-first-in-industry-to-achieve-khronos-conformance-for-fpgas-227993801.html.

[4] Settle, Sean, "High-performance Dynamic Programming on FPGAs with OpenCL" (2013). http://ieee-hpec.org/2013/index_htm_files/29-High-performance-Settle-2876089.pdf.

[5] OpenCL Specification, Version 1.2, Khronos Group (2012). https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf.

[6] The GATK Guide Book, Version 3.4-46, Broad Institute (2015). https://www.broadinstitute.org/gatk/guide/version-history.

[7] Oliver, Timothy, Bertil Schmidt, et al. "Accelerating the Viterbi Algorithm for Profile Hidden Markov Models Using Reconfigurable Hardware" (2006). https://link.springer.com/chapter/10.1007/11758501_71.

[8] Ito, Megumi, and Moriyoshi Ohara, "A Power-efficient FPGA Accelerator: Systolic Array with Cache-coherent Interface for Pair-HMM Algorithm," IBM Research – Tokyo (2016). http://ieeexplore.ieee.org/document/7503681/?reload=true.

[9] Ren, Shanshan, Koen Bertels, and Zaid Al-Ars, "Exploration of Alternative GPU Implementations of the Pair-HMMs Forward Algorithm." https://ce-publications.et.tudelft.nl/publications/1562_exploration_of_alternative_gpu_implementations_of_the_pair.pdf.