Accelerating Genomics Research with OpenCL and FPGAs

Chris Rauer, Software Engineer, Altera, Now Part of Intel
Nicholas Finamore Jr., Sales and Marketing Associate, Altera, Now Part of Intel

WP-01262-1.0 White Paper

With the emergence of second-generation sequencing equipment, gene sequencing costs have decreased rapidly, leading to a dramatic increase in the availability of genome sequence data. Correlating the variations in genomes enables advances in a wide range of medical research, including personalized care. Because each human genome comprises over three billion base pairs, whole genomic sequencing requires significant processing power, storage capacity, and network bandwidth. In particular, variant calling is an extremely computationally intensive function. 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 Haplotype Caller algorithm using Altera®, now part of Intel, FPGAs programmed with Altera's SDK for OpenCL.

Introduction

Genomic variant discovery appears to be a straightforward problem: map reads to a reference sequence and at every position, count the mismatches and construe the genotype variants. However, multiple error sources in the sequence data make this process much more complex. These errors include:

- Amplification biases that occur during wet lab preparation
- Machine errors during library sequencing
- 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 errors modes.*" (1) Because of all these errors, variant discovery becomes a computationally intensive undertaking. Modern variant caller algorithms require up to several days of computation time using standard microprocessors.

Heterogeneous Computing and the OpenCL Computing Language

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

101 Innovation Drive, San Jose, CA 95134 www.altera.com

ISO 9001:2008 Registered

OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, digital signal processors (DSPs), FPGAs, and other multi-core processors. The OpenCL framework includes a language based on standard ANSI C99 for programming these devices and application programming interfaces (APIs) to control the platform and execute programs on the compute devices. The Khronos Group [(2)], a non-profit organization, manages the standard. An advantage of OpenCL is portability of programs from one vendor's accelerator device to another. Several vendors, including Altera, provide compilers for OpenCL. To claim conformance to the OpenCL standard, the vendor's compiler must accurately compile and execute a suite of over 8,500 OpenCL programs. [(8)] Altera, Intel, AMD, and Nvidia provide OpenCL conformant compilers.

FPGA Technology

FPGAs are reconfigurable integrated circuits consisting of programmable routing networks linking together logic array blocks, embedded memory blocks, and DSP blocks. In contrast, CPUs and GPUs contain fixed data paths and topologies that process program instructions. FPGA resources can 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 through put performance and power efficiency by reducing the amount of superfluous functional units in silicon.*" [(3)] The FPGA architecture can be utilized to do a certain types of computing problems very efficiently.

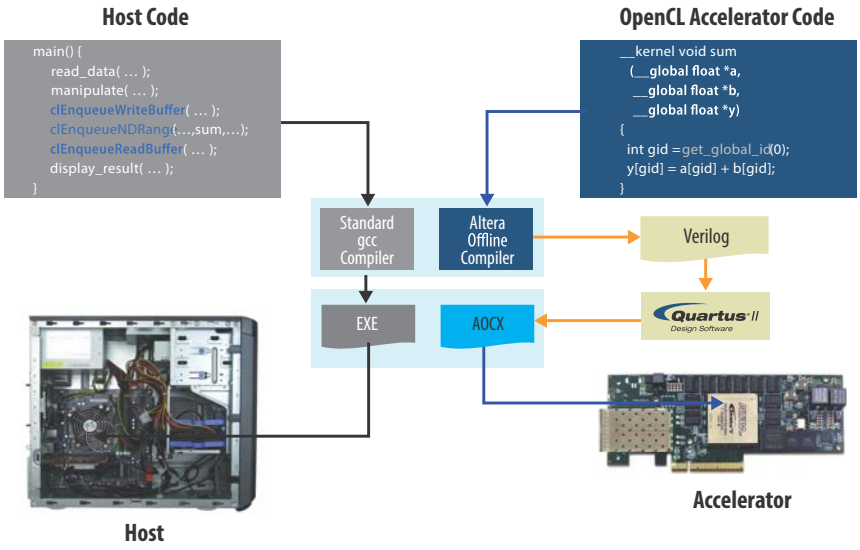
Programming with FPGAs

Traditionally, hardware developers have designed and verified digital circuits on FPGAs at the register-transfer level (RTL) using hardware description languages (HDLs) such as Verilog HDL and VHDL. While these traditional methods are effective to ensure efficient use of the devices, they are impractical for implementing complex algorithms such as gene sequencing. In early 2012, Altera introduced the Altera SDK for OpenCL, a software development kit that allows use of the OpenCL programming language to program Altera's FPGA as computing accelerator devices. In late 2014 Xilinx Corporation, another leading FPGA vendor, announced they were also developing a compiler for OpenCL. Altera's SDK for OpenCL has been utilized for a wide array of algorithms in a variety of computing fields.

Altera's SDK for OpenCL

The Altera SDK for OpenCL uses the same programming model as other vendors' compilers. Figure 1 illustrates the 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. FPGA programming is performed purely with OpenCL.

Figure 1. Altera SDK for OpenCL Use Model



The OpenCL program consists of a host program (intended to run on a standard CPU) and the kernel code (intended to run on the accelerator, in this case the FPGA). Using the standard IDE and GCC compiler, a programmer writes and compiles their host code. Using OpenCL API they communicate with the OpenCL kernel. In a separate **.cl** file, the programmer writes with OpenCL C following the appropriate optimization guidelines for the FPGA. Using the Altera offline compiler, the OpenCL kernel file is compiled and runs the Quartus® II or Quartus Prime software in the background to produce an **.aocx** file. During runtime, the Altera offline compiled executable is downloaded to the FPGA. All of the typical tools and processes that an FPGA designer would typically deal with abstract away; all development happens in the familiar software programmer's environment. *(4)*

FPGA Devices Used

For the purposes of this experiment, the Haplotype Caller code is partitioned to run on both the host and FPGA to optimize performance. The OpenCL complier was initially targeted to build the code for an Altera Stratix® V FGPA and then re-targeted at Altera's more advanced Arria® 10 FPGA. The Stratix V FPGA is part of Altera's high-end family of devices and has been shipping in volume production since 2012. The product line is built with a 28 nm silicon process technology from TSMC Corp. Altera is also shipping Arria 10 devices using a more advanced 20 nm process technology with more logic elements, DSP blocks, and memory. Additionally, Arria 10 FPGAs run at higher frequencies. Arria 10 FPGAs have advanced, hardened, floating-point elements that make floating-point functions more efficient than when implemented using standard logic. These features let us use more computational blocks, leading to higher overall performance and significantly better performance per watt for the Haplotype Caller algorithm.

Genome Analysis

Genome Variant Discovery

The process of identifying differences between DNA sequences is called variant discovery. Identifying variation in DNA has become essential in a variety of medical research and personalized medical care. Research projects that compare hundreds or thousands of sequences are stifled by the amount of compute time and resource required. Therefore, accelerating variant discovery has become a pursuit of many in the medical and high-tech community.

Using a robust calling algorithm that compares sequences and leverages meta-information (such as base qualities scores variant discovery) can be performed on the appropriately processed data. To avoid missing any or limiting the proportion of false positives in the call set, it is preferable 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.*" [(5)]

The Genome Analysis Tool Kit

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

- The GATK Haplotype Caller function is the variant discovery algorithm.
- PairHMM is the main algorithm to compare sequences. It calls SNPs and indels simultaneously via local re-assembly of haplotypes in an active region.
- The Haplotype Caller defines ActiveRegions, determines haplotypes by re-assembly of the ActiveRegion, determines likelihoods of the haplotypes given the read data, and assigns sample genotypes. [(5)]

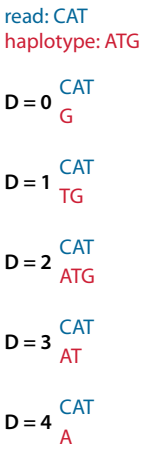
Using significant variation evidence, the areas to be further analyzed are identified and known as the ActiveRegions. The program then creates a De Brujin-like graph to reassemble the ActiveRegions and detect the possible haplotypes, which are then realigned using the Smith-Waterman algorithm. Using 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 matrix is then relegated to create the likelihoods of alleles for each potential variant site. [(5)]

PairHMM Algorithm Overview

Comparing two gene sequences is not as simple as comparing two regular strings because each sequence can have insertions, deletions, and mutations. The hidden Markov models in the PairHMM algorithm calculate the probability of a match with these possible changes. Also, the exact alignment is not known, therefore, a comparison must be done with each alignment.

The algorithm requires two gene sequences as input. 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 a gene string without any additional data. The PairHMM hidden Markov model equation compares the sequences and the result is passed to the next diagonal. The next diagonal compares the same two sequences again with a different alignment. (The different alignment is simply 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.

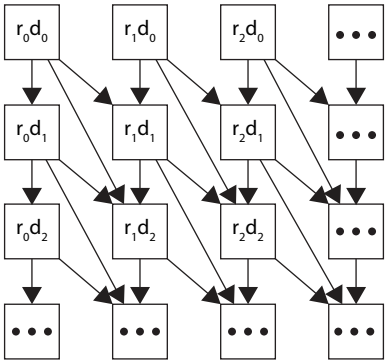
Figure 2. Comparing Each Alignment of 2 Small Sequences



Read length	3
Haplotype length	3
Number of diagonals	3 + 3 - 1 = 5
Number of PairHMM calculations	5 x 3 - 15

The hidden Markov comparison result is a single probability score, but it is very computationally expensive because it uses floating-point math. For a sequence of length n, the computation requirement is $O(n^2)$. Additionally, 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 summed 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 structure is called a systolic array and can be easily mapped to an FPGA fabric.

Figure 3. Using the PairHMM Calculation for 2 Sequence Characters



PairHMM FPGA Implementation

This algorithm can be optimized fairly well on a CPU using vector instructions. If the sequences are small, the comparison can be entirely performed using the CPUs internal level 1 cache memory. However, with larger sequences, the external memory bandwidth of the CPU may limit performance. A CPU may have clock speed of over 1 GHz, but the calculation still must be broken up into separate instructions. These algorithms also work extremely well on FPGAs. An FPGA typically has a lower clock speed, but it can take advantage of pipelined parallelism. Therefore, the FPGA can do hundreds of complex calculations in parallel and run one after the other each cycle inside the pipeline. Altera's OpenCL compiler analyzes the code and builds these pipelines automatically. The FPGA's lower clock speed typically enables applications to run while consuming much less power.

Experiment

The GATK Haplotype Caller Algorithm was initially written in Java. The algorithm was then converted to C++ by the Broad Institute. For our experiment, we ported the PairHMM algorithm (that was originally written by the Broad Institute [(5)]) from C++ to OpenCL. OpenCL is a C based language, which made porting the algorithm fairly straightforward. Additionally, the code was well optimized and required constant values were already pre-calculated. We tested the code for functional correctness with the emulator that is part of Altera's SDK for OpenCL; we used the test cases that came with the Broad Institute's C++ source code.

We targeted the code to a Stratix V D8 device on a Bittware Corp. (an Altera partner) board. [(9)] The compiler generated an **.aocx** file, which we loaded into the Stratix V device. We then took runtime performance measurements. We used the same methodology to run the algorithm on an Arria 10 device on an Altera development board with ES2 silicon.

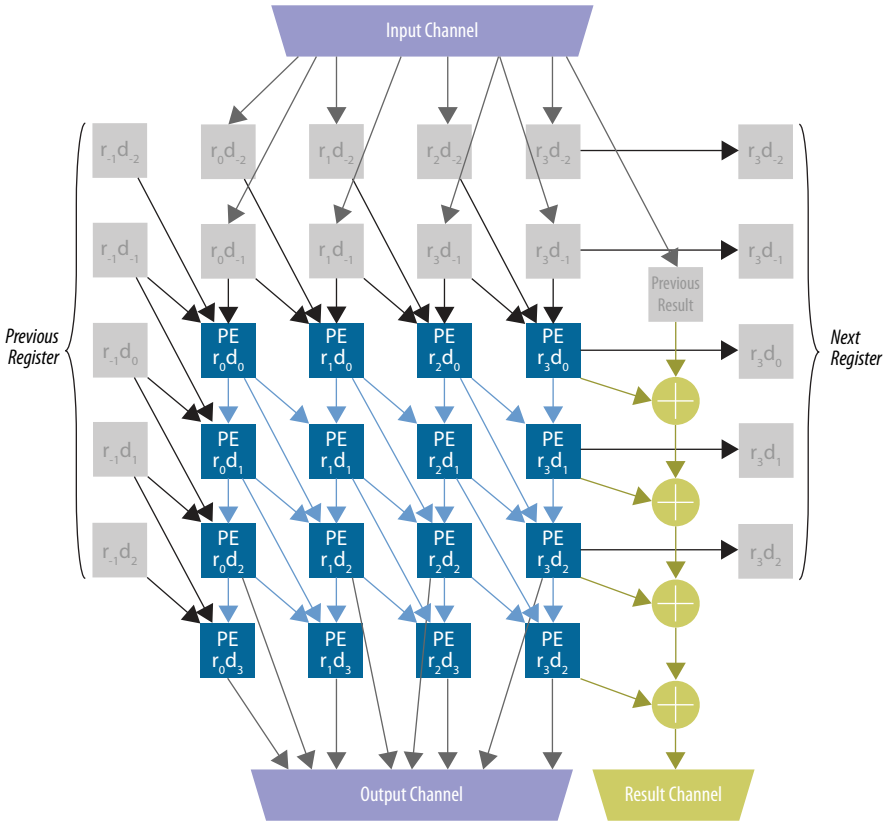
Choosing an Optimal Compute Grid Size

The FPGA accelerates this algorithm effectively because the algorithm can be mapped to a 2-D systolic array. In Figure 3, the top computations feed the bottom computations and the results trickle through the grid's compute units. Sizing this grid required experimentation. If the compute grid is too wide, the FPGAs M20k used blocks increase because the accesses is wider with shallower depth. If the depth needed is less than the physical depth of the M20k block, the block is underutilized. Several variables are loaded from memory for each column, which greatly amplifies the effect.

If the compute grid depth is too deep, too many haplotype characters are read from DDR memory simultaneously, which exhausts the DDR memory bandwidth. Additionally, adders are needed at the end of the row. As a result, increasing the depth linearly increases the number of these adders.

When the OpenCL compiler builds the functions for a Stratix V FPGA, it can fit 64 PairHMM processing elements in an 8 x 8 grid. This design is completely pipelined and can perform 64 full PairHMM calculations per clock cycle. Figure 4 shows a visual representation of a smaller 4 x 4 compute grid. When the same code is run through the compiler and re-targeted to an Arria 10 device, it produces an 8 x 26 grid, which is 3.25X larger than the Stratix V grid size. This difference is due to the Arria 10 device's hardened floating-point capability, as well as additional logic, memory, and switching fabric. The Arria 10 FPGA also runs at a somewhat higher frequency, adding to the performance gains.

Figure 4. Visual Representation of OpenCL 4x4 PairHMM Algorithm Implementation



Results

FPGA Utilization

Table 1 shows the FPGA compilation results. As expected, the Arria 10 FPGA fit more PairHMM algorithm computation blocks because of the hardened floating-point DSP. The Arria 10 DSP utilization is almost 100%. For the Stratix V device, the compute size is smaller because LEs and DSP blocks implement the floating-point functions. Also, the design only used 20% of the available DSP blocks because so many LEs were needed for floating-point operations. Using a larger FPGA would not have helped because the D8 device has the most DSP blocks. The $\mathbf{f_{MAX}}$ was not much better in the Arria 10 device due to early silicon and timing models.

Table 1. FPGA Compilation Results for PairHMM Algorithm

FPGA	PairHMM Compute Blocks	fMAX	Logic Utilization	DSP Utilization
Stratix V D8	64	207	209k/262k (80%)	388/1,963 (20%)
Arria 10 ES2 1150	208	213	210k/427k (49%)	1,513/1,518 (100%)

Performance

We compared the performance results to the initial Java-based runtime, the runtime on a 3.2 Ghz Intel Xeon CPU using the optimized AVX assembly language code, and the published results for the Nvidia K40 GPU. We used the *10 Second* Java test for comparison because it is the published runtime baseline comparison for different technologies. We also ran the algorithm on larger data sets, however, there are no published results to which we can compare our results.

Table 2. Performance Results Comparing Various Platforms to Original Java Code

Technology	Hardware	Runtime (ms)	Improvement over Baseline (X Fold)
Original Java version from GATK [7]	CPU running Java	10,800	1
Intel AVX (Single core)	Intel Xeon	138	78
NVidia GPU [7]	NVidia K40	70	154
Intel AVX 24 core [7]	Intel Xeon	15	720
Altera OpenCL	Stratix V D8	8.3	1,301
Altera OpenCL	Arria 10 ES2 1150	2.8	3,857

Conclusion

The Altera SDK for OpenCL allowed us to implement and test the GATK PairHMM algorithm effectively and easily. The Altera FPGA showed significant performance acceleration relative to other technologies. By porting the PairHMM algorithm from a Stratix V FPGA to the more advanced Arria 10 FPGA, we obtained a nearly 3X performance improvement. This experiment shows great promise when performance scaling these types of algorithms to future generations of FPGA technology.

Our recommendations for future work include:

- Incorporating the accelerated algorithms into the complete GATK
- Implementing compression algorithms in the FPGA to store and transport genome data more effectively
- Accelerating analysis engines, such as the GATK
- Porting the design to Stratix 10 devices to obtain further performance improvements

For additional performance improvements, we can further optimize the OpenCL code. For instance, in the current design, the result adder chain in Figure 4 is not 100% used every cycle. An improvement would be to multiplex one of the adders from the hidden Markov calculation to share the hardware (an optimization called resource folding). This improvement would free DSP resources that could be used to add more computation units, thereby increasing performance.

Acknowledgments

We would like 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 Mauricio Carneiro and Eric Banks from the Broad Institute for providing initial code and supporting us. We would also like to thank John Sotir and Richard Yang for making this project possible.

References

1. Van der Auwera, Geraldine A., et al. *From FastQ Data to High Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline* www.ncbi.nlm.nih.gov/pmc/articles/PMC4243306/
2. Altera SDK for OpenCL Programming Guide *documentation.altera.com/#/00015315-AA$NT00066862*
3. Settle, Sean, *High-performance Dynamic Programming on FPGAs with OpenCL*, 2013. ieee-hpec.org/2013/index_htm_files/29-High-performance-Settle-2876089.pdf
4. *OpenCL Specification*, Version 1.2, Khronos Group, 2012 www.khronos.org/registry/cl/specs/opencl-1.2.pdf
5. *The GATK Guide Book*, Version 3.4-46, Broad Institute, 2015. www.broadinstitute.org/gatk/guide/version-history
6. Carneiro, Mauricio, PairHMM (GitHub repository) github.com/MauricioCarneiro/PairHMM

7. Carneiro, Mauricio, *Acceleration Variant Calling*, Intel Genomic Sequencing Pipeline Workshop, Mount Sinai, 2013.
8. Altera SDK for OpenCL is First in Industry to Achieve Khronos Conformance for FPGAs, Altera Corporation, 2013.
https://www.khronos.org/news/permalink/altera-sdk-for-opencl-is-first-in-industry-to-achieve-khronos-conformance-f
9. Bittware S5-PCIe-HQ board
https://www.bittware.com

Document Revision History

Table 3 shows the revision history for this document.

Table 3. Document Revision History

Date	Version	Changes
March 2016	1.0	Initial release.