Optimizing the Rodinia Benchmark for FPGAs (Unrefereed Workshop Manuscript)

HAMID REZA ZOHOURI[†1] NAOYA MARUYAMA[†2]
AARON SMITH[†3] MOTOHIKO MATSUDA[†2] SATOSHI MATSUOKA[†1]

Abstract: We evaluate the performance of a sub-set of the benchmarks available in the Rodinia Suite, namely the integer benchmarks and some of the single-precision floating point benchmarks, using Altera's OpenCL SDK and the Terasic DE5-Net FPGA board, equipped with an Altera Stratix V GXA7 FPGA, and compare performance results with a modern CPU and GPU. The results are presented for multiple versions of each benchmark, each with a varying degree of optimization for FPGAs, ranging from direct ports from the initial OpenCL implementation to loop-pipelined kernels specifically optimized for FPGAs. Our results show that, while it is possible to use a common programming language available for other more-widely used accelerators in HPC, the implementation method optimal for FPGAs is significantly different from those for other accelerators such as GPUs. Specifically, we find that multi-threaded kernels typically used for GPUs do not perform as efficiently as those optimized with typical FPGA-specific optimizations such as sliding windows. However, by exploiting the FPGA-specific optimizations, FPGA with OpenCL shows promising performance. Our results using the Altera Stratix V 5SGXA7 FPGA show that it is possible to achieve similar or better performance in comparison to CPUs and up to only 1.5x slower than GPUs which means the FPGA provides much better power efficiency.

Keywords: FPGA, OpenCL, Rodinia Suite, CUDA, OpenMP

1. Introduction

Traditionally, FPGAs have been considered as a middle-ground between ASICs and general-purpose processors, offering better performance and performance per watt in comparison to general-purpose processors for a wide range of applications, but not as good as ASICs. On the other hand, FPGAs, due to their reconfigurable nature, can be used to run a wide variety of applications by reconfiguring them for each new application while in contrast, ASICs are designed for a specific algorithm/application and are considered to have very low flexibility. For example, the Catapult project at Microsoft demonstrates that it is possible to design an FPGA-based datacenter architecture for various workloads including web searches and machine learning [1].

Using FPGAs has always been considered very challenging due to the fact that Hardware Description Languages (HDL), like Verilog and VHDL, work in a completely different manner in comparison to software programming languages, mainly due to the fact that they have little to no high-level constructs and their programming model follows a parallel/data flow model rather than a sequential model like conventional software programming languages. Apart from this, very slow placement and routing (part of the compilation process for FPGAs) and debugging has always been considered as big challenges in FPGA-based programming.

Throughout the years, numerous attempts have been made to make FPGAs more attractive to software programmers using *C-to-Gates* or *C-to-hardware* converters which convert programs written in a software programming language to HDL, for use on FPGAs. This process is usually called High-level Synthesis (HLS). Notable example of C-to-Gates converters are AutoESL Autopilot [2], now acquired by Xilinx [3], Cadence Stratus High-Level Synthesis [4] and Synopsys Synphony C Compiler [5], all of which can convert C, C++ and System C to synthesizable HDL. The main drawback of HLS is that most converters focus on productivity and ease of programming for FPGAs and hence, usually suffer from performance issues. Also always, as part of the HLS flow, apart from verifying the original software code, an additional verification step is required after conversion to HDL to make sure the conversion has been done correctly [6].

Recently, to make FPGAs more attractive to software programmers, especially the HPC industry, both Altera [7] and Xilinx [8] have enabled the possibility of using OpenCL, a royalty-free, open source and portable programming language initially created by Apple in 2008 [9] and now maintained by Khronos Group [10], to program FPGAs. This new approach not only enables the possibility of porting existing OpenCL code for CPUs and GPUs onto FPGAs, but also since it is directly supported by FPGA manufacturers and is already embedded in their FPGA toolsets, is expected to have better performance and conversion quality in comparison to third-party converters. However, there are still few studies on benchmarking FPGA performance with OpenCL.

To evaluate the performance of Altera's OpenCL SDK on FPGAs and determine the effectiveness of using FPGAs in future HPC systems, we chose the Rodinia Suite [11] and ported some of the integer and floating-point benchmarks to the Terasic DE5-Net FPGA board, equipped with an Altera Stratix V GXA7 FPGA, using Altera's OpenCL SDK, and compared run time with CPUs and GPUs.

In addition to evaluating the original Rodinia benchmarks, we also explore the effectiveness of FPGA-specific parallelization and optimizations. The original Rodinia implementations are meant to be used for GPU-like highly multi-threaded processors. Although Altera FPGAs support executing such programs by pipelining the execution of multiple threads, it is likely that such programs will perform suboptimally due to barriers. A more FPGA-friendly parallelization, as recommended in the Altera programing and optimization guides [19, 20], is to parallelize loops by pipelining iterations, where data dependency across iterations can be resolved with sliding windows. Currently, we are creating a single-threaded version of each Rodinia benchmark as well as its optimized version with sliding windows. In this paper, we report our current status of evaluations with five benchmarks and three optimization case studies. We find that the

†1 Tokyo Institute of Technology
†2 RIKEN Advanced Institute for Computational Science
†3 Microsoft Research

original multi-threaded kernels do not perform as efficiently as those optimized with sliding windows, indicating the importance of FPGA-specific optimizations. However, with those optimizations, FPGA achieves highly promising performance Overall, our results using the Stratix V family of FPGAs show that it is possible to achieve similar or better performance in comparison to CPUs and up to only 1.5x slower than an Nvidia K20x GPU which means the FPGA provides much better power efficiency. Our study also indicates that, while the availability of a standardized common programming language such as OpenCL allows for portable programming across different accelerators including FPGAs and GPUs, the problem of performance portability is more critical. While many techniques such as auto-tuning have been presented to solve the performance portability issue, it is not clear such methods are effective for FPGAs since the time to generate the final routed logic gates usually takes hours rather than minutes for conventional processors.

The rest of the paper is organized as follows. In Section 2, we review related work. In section 3, our porting and optimization methodology is explained. In section 4, a description of each of the benchmarks alongside with the baseline optimization method is detailed. In Section 5, our FPGA-specific advanced optimization has been discussed. Our software and hardware platform is detailed in section 6. In section 7, optimization techniques and timing results are provided alongside with results of comparison with CPU and GPU. Section 8 is also dedicated to conclusion and future work.

2. Related Work

One of the earliest attempts in utilizing OpenCL for FPGA-based programming was presented in [12]. In this paper, the authors have presented a source-to-source converter called SOpenCL (Silicon-OpenCL) which is capable of producing synthesizable HDL code from OpenCL kernels. Similarly, in [13], the authors have presented another source-to-source converter capable of converting CUDA programs to synthesizable code for FPGAs. Both of these papers aim at creating a platform for automatic conversion of GPU code to synthesizable code for FPGAs.

In [14], the authors have introduced an OpenCL-based benchmark suite called OpenDwarfs and have presented their results from running 4 benchmarks (GEM, NW, SRAD and BFS) on a wide range of hardware including a Xilinx Virtex-6 LX760 FPGA. They have used SOpenCL for converting their OpenCL kernels to HDL code. Since the paper uses the kernels originally written for GPUs and does not discuss FPGA-specific optimizations in detail, it is not clear yet what performance can be obtained with FPGAs using OpenCL.

In [15], the authors have presented an implementation of the k-NN algorithm on the Terasic DE4 board with an Altera Stratix IV 4SGX530 FPGA, using Altera OpenCL SDK. Speed and performance per Watt comparison has also been provided with CPU and GPU. In [16], an implementation of Binomial Option Pricing has been presented on the same board using Altera OpenCL SDK v13.0 SP1 and compared with CPU and GPU.

In [17], Settle presented an implementation method of the Smith-Waterman algorithm using Altera's OpenCL SDK, which is similar to Needleman-Wunsch studied in our work. Our optimization using sliding windows is based on his method.

In [18], three image processing kernels (Canny, Sobel, and SURF) have been implemented on three OpenCL-capable FPGA boards (Gidel ProceV, Nallatech PCIe385-D5 and BittWare S5PH-Q) using Altera's OpenCL SDK, and area usage, operating frequency and productivity results have been compared with HDL implementations of the same kernels on one of the boards. In [19], Che et al. presented comparison of GPU, FPGA, and CPU for three benchmarks, including Needleman-Wunsch, which is also used as an evaluation workload in this paper. The paper compared the number of cycles on each processor for a given algorithm when implemented with CUDA, VHDL, and C with OpenMP for GPU, FPGA, and CPU, respectively. It did not investigate any processor-specific optimizations. Unlike their paper, we use OpenCL instead of the gate-level description language and show timing results with optimizations specific to each of the architectures.

3. Methodology

To understand the performance characteristics of FPGAs as an accelerator for a wide range of parallel applications, we use the Rodinia benchmark suite as a representative set of parallel computations [11]. It consists of twenty-one benchmarks, each of which represents one of the computation patterns compiled by [20], with implementations in two kinds of parallelism: fork-join coarse-grained parallelism based on OpenMP and fine-grained highly-multi-threaded parallelism based on CUDA and OpenCL. The former is intended to be used on multi-core CPUs, whereas the latter is for GPUs.

In this study, we use the multi-threaded OpenCL version as the baseline implementation for each benchmark and incrementally extend it with the optimizations suggested in the programming and optimization guides by Altera [21, 22]. We port the original benchmarks to comply with the restricted set of the OpenCL specification supported in the Altera OpenCL SDK. Specifically, since the SDK does not support the JIT compilation model, due to hours of compilation time, and the original Rodinia benchmarks all use JIT compilation, we rewrite the part of code that deals with loading of kernels, to use pre-compiled binary code.

To understand the effectiveness and necessity of optimizations, we first apply basic optimizations that give the compiler hints to better exploit the resources of a given FPGA more effectively. These optimizations do not require significant code restructuring, and therefore, the extension from the original version can be relatively straightforward. The basic optimizations evaluated in this paper include:

- No pointer aliasing using restrict keyword
- Loop unrolling
- Static setting of work-group sizes
- Kernel-wide SIMD code generation
- Kernel pipeline replication

The baseline and optimized versions use the fine-grained multi-threading model (thread-pipelined) since they are originally developed for GPUs. In contrast to GPUs, Altera's OpenCL Compiler does not automatically replicate processing logic to parallelize execution of multiple threads. Instead, multiple work-items and work-groups are processed in parallel by a pipeline that corresponds to the kernel function.

In addition to the explicit thread-based parallelization, Altera's OpenCL SDK provides another type of parallelism through automatic loop pipelining. Loops in an OpenCL kernel are processed in a pipeline-parallel fashion if the kernel is a single-threaded OpenCL task kernel (loop-pipelined). If data dependencies between loop iterations exist, a proper amount of stall cycles are inserted between iterations based on the compiler analysis. The programmer can optimize the pipeline efficiency by manually resolving data dependencies with sliding windows or shift registers implemented on registers and other on-chip memories. Such locality of data accesses can be more efficiently exploited with this model of parallelism since it does not require explicit barriers to attain memory consistency, which cause a large overhead due to pipeline flush. We create the baseline versions either by following the corresponding OpenMP version or by wrapping the multi-threaded OpenCL version with nested loops. As for the multi-thread evaluation, we evaluate the effectiveness of a set of non-restructuring basic optimizations when applied to the loop-based versions. Furthermore, we

attempt to improve the pipeline efficiency through, in most cases, a sliding window.

This paper reports an incomplete set of results since our porting and optimization studies are still ongoing. Specifically, we report the baseline versions of NW, NN, SRAD, Pathfinder and Hotspot benchmarks, as well as their respective version with the basic optimizations. For NW, Hotspot and Pathfinder, we present how their loop-pipelined versions can be optimized with sliding windows. Compared to our previous work, presented at SWoPP 2015, this work includes new benchmark results and discusses FPGA-specific optimizations.

4. Benchmarks

In this section we will describe the benchmarks used in our study.

4.1 Needleman-Wunsch

Needleman-Wunsch (NW) is a dynamic programming benchmark based on a sequence alignment algorithm. A pair of strings is organized as the top-most row and the left-most column of a 2-D matrix. The algorithm computes a score for each matrix element from the top-left position to the bottom-right position. Each score is computed based on its neighbor scores at the top, left, and top-left positions, resulting in diagonal data dependency. Wavefront parallelization is implemented in both the original OpenMP and CUDA/OpenCL versions. No floating-point arithmetic is used in this benchmark.

We create the thread-pipelined version with minimal changes from the original OpenCL version. For the loop-pipelined versions, our baseline version computes the matrix with doubly-nested loops, emulating a straightforward implementation without FPGA-specific adaptation. We also evaluate the effectiveness of basic optimizations for both thread-pipelined and loop-pipelined versions.

4.2 Hotspot

Hotspot is a structured grid benchmark. It simulates microprocessor temperature based on a stencil computation on 2-D structured grids. The stencil is an arithmetic computation using single-precision floating-point values. In the original CUDA version, the 2-D grid is decomposed into sub grids, each of which is computed by a thread block. The Rodinia implementations have an optimization that saves global memory accesses by redundantly computing wider halo regions.

Our thread-pipelined versions use the original OpenCL version. The baseline loop-pipelined version consists of doubly nested loops for the vertical and horizontal dimensions with stencil computations contained inside the inner loop.

4.3 Pathfinder

Pathfinder is a dynamic programming benchmark that attempts to find a path with smallest accumulated weight, from the bottom of a 2-D grid to its top. Movement direction is either straight ahead or diagonally ahead and calculation is done row by row. This benchmark has one kernel and is integer.

We use the original OpenCL kernel as the baseline thread-pipelined version. The baseline loop-pipelined version is based on the OpenMP version and unlike the thread-pipelined case, all the row-wise iterations are computed within a single kernel call to reduce host-device interaction.

4.4 Nearest Neighbor

Nearest Neighbor (NN) is a dense linear benchmark that finds the *k* nearest valid locations to a certain point of interest, or as per the author's description, "computes the nearest location to a specific latitude and longitude for a number of hurricanes". The kernel of this benchmark involves single-precision floating-point addition, multiplication and square root.

We use the original OpenCL kernel as the baseline thread-pipelined version. The baseline loop-pipelined version is created by wrapping the thread-pipelined kernel in a for loop.

4.5 SRAD

SRAD is a structured grid benchmark that processes 2-D medical images with PDE-based diffusion kernels. Similar to Hotspot, its computation involves stencil computations with single-precision floating-point values. Unlike Hotspot, it also includes reduction of grids, which presents the compiler with a different type of data dependency.

We use the original OpenCL kernel as the baseline for thread-pipelined versions and create loop-pipelined versions by wrapping the thread-pipelined version with doubly nested loops.

5. FPGA-specific Optimization Using Sliding Window

The loop-pipelined versions of the aforementioned benchmarks reflect implementations written in a straightforward way in OpenCL. Their experimental results will demonstrate a level of performance expected for programs written without FPGA-specific code restructured from its sequential CPU counterpart. While such results are useful as a performance baseline, it is also important to explore the potential performance limit for a given algorithm running on an FPGA. In this section, we show optimization methods for three of the benchmarks, NW, Hotspot, and Pathfinder, that attempt to improve pipeline throughputs by exploiting on-chip memory as sliding windows.

Note that our optimized implementations are still written in the standard OpenCL language. Directly expressing the algorithms in hardware description languages may allow us to achieve higher performances than those written in OpenCL, which is beyond the scope of this paper.

5.1 Needleman-Wunsch

The NW benchmark computes a 2-D matrix with a neighbor data dependency. The original OpenCL version computes the matrix elements with wavefront parallelization, where the inter-wave data dependency is resolved by using the OpenCL local memory. This method is also applicable on FPGAs, where the local memory is instantiated with the on-chip memory. However, the cost of thread synchronizations, even though it is limited to a work group, is known to be very expensive.

As suggested by the optimization guide [21,22], a more efficient way of using FPGA's on-chip memory is to implement sliding windows with loop pipelining. In fact, such an optimization for the Smith Waterman algorithm, which follows almost the same computation as Needleman-Wunsch, is presented by Settle [17]. We apply Settle's method to the NW benchmark and create an optimized loop-pipelined version as follows.

Each matrix element depends on the above, left, and top left elements. The baseline loop-pipelined version resolves the dependencies through the OpenCL global memory, which incurs significantly larger overhead than accesses to on-chip memory. Among the three dependencies, the left neighbor point can be obtained through a local register if consecutive row elements are computed by a single pipeline.

For the above and above-left elements, we use a sliding window that initially holds the top-most one row and pipeline the iterations of the vertical loop. Once the first element of the sliding window is computed by the first iteration of the loop, the next iteration can be started to compute the first element of the next row with one cycle delay, achieving the optimal pipeline throughput.

Since the size of the sliding window is limited to available FPGA resources, the OpenCL kernel uses 1-D column-wise blocking of size N, which is called multiple times for all the columns by the host code. In our current implementation, the accesses to the block boundary column still use the global memory. Although its cost is likely to be minor, it is possible to

avoid reading from global memory for the neighbor block by using remaining on-chip memory such as Altera's channel extension or the OpenCL pipes. We plan to investigate such further optimizations in our future study.

5.2 Hotspot

Hotspot iteratively updates a 2-D matrix using a 5-point stencil. Double buffering allows all the computations of one time step to be done in parallel. Exploiting data reuse for neighbor accesses is a well-known optimization for stencil computations, and is implemented in the original and our baseline thread-pipelined versions using OpenCL local memory. In FPGAs, since it is possible to achieve data reuse with a sliding window as shown in the example code distributed by Altera [23], we study its effectiveness in the Hotspot benchmark.

Similar to [23], we create a sliding window of size (N * 2 + P), where N represents the blocking size along the x dimension, and P represents the number of elements computed by a single iteration within each block. Our implementation assumes that N is divisible by P, and the loop across the sub region of N columns and the entire rows is pipelined with the maximum throughput. The host code calls the pipelined kernel multiple times to cover the entire 2-D matrix, and iterates this for the given number of time steps.

Although the original OpenCL version also implements a temporal blocking optimization for GPUs, our current implementation is limited to spatial blocking. Such aggressive blocking can be also effective for FPGAs since the benchmark algorithm is memory access intensive, and is a subject of our future study.

5.3 Pathfinder

The computation pattern and data dependency of Pathfinder is similar to those of NW. More specifically, it updates a 2-D matrix element using three neighbor elements located in the bottom, bottom left, and bottom right positions. The benchmark starts from the bottom row and iteratively computes all the rows by using the previous one line of row. Unlike NW, since it depends on both the bottom left and bottom right elements, it is not possible to use the same 1-D blocking method. Instead, we divide the problem space with sloped parallelograms so that the dependencies of each parallelogram can be resolved internally or using the previous parallelogram. We then use a sliding window of size N, where N represents the blocking factor along the row direction, and pipeline the loop across rows in a parallelogram with no pipeline stall. The kernel is then called by the host code multiple times to compute the whole problem space.

6. Software and Hardware Platform

Our hardware platforms consist of two machines, one for the FPGA board and one for CPU and GPU benchmarking. The FPGA board information are detailed in Table 1, and Table 2 contains information about the two systems.

Table 1 FPGA board Specifications

Manufacturer	Terasic
Board Name	DE5-Net
FPGA	Stratix V 5SGXA7
ALM	234,720
Register	938,880
M20K	2,560
DSP	256

Table 2 Test System

Machine	1	2
CPU	Intel i7-920 (4x 2.67 GHz)	Intel Xeon E5-2670 (8x 2.6 GHz)
GPU	--	Tesla K20X
FPGA Board	DE5-Net	--
Memory	12 GB DDR3	32 GB DDR3
OS	CentOS 6.6	CentOS 6.5

Our software platform consists of Altera Quartus v15.0.2, Altera OpenCL SDK v15.0.2 and Terasic Board Support Package 14.0/14.1 for the DE5-Net board. For GPU performance evaluation, we use the CUDA versions of the original Rodinia benchmark with CUDA v7.0.28 and Tesla K20X on machine 2. For CPU performance evaluation, we use the 8-core Xeon on machine 2 with GCC v4.9.2.

7. Results and Comparison

In this work we will only report kernel run time and disregard host to device memory transfer at this stage. Run times have been extracted using the default execution settings from the Rodinia Suite unless stated otherwise.

It should be noted that version 0 in all benchmarks is the original kernel from the Rodinia Suite, odd versions are single-threaded (loop-pipelined) with version 1 being the base-line version, and even versions are multi-threaded (thread-pipelined). Higher numbers reflect higher optimization effort (odd or even numbers are only comparable with their own group).

7.1 Needleman-Wunsch

Table 3 shows timing results for the NW benchmark. The length of strings is 2048 in our evaluation. Versions 2 and 3 represent the kernels with the basic optimizations. Specifically, version 2 adds 4-way SIMD and, if possible, restrict keyword for the input parameters to version 0 while version 3 uses only the restrict keyword on top of version 1. Version 5 is the loop-pipelined kernel with the advanced sliding window optimization.

Table 3 NW Results

Version	F_{max}	Run Time (ms)
0	277.23	258.265
1	243.48	1831.253
2	194.7	29.229
3	249.19	1818.112
5	148.06	3.721

As shown in the table, the sliding window-based version performs the most efficiently among all five versions. The baseline multi-threaded kernel (version 0) performs approximately 70x slower than the fastest (version 5), and while its performance is substantially improved with the basic optimizations as shown as version 2, the gap between the thread-pipelined and the optimized loop-pipelined versions is significant. Since they use different types of parallelism with different data-locality optimizations, optimizing performance requires the programmer to write completely different versions for FPGAs and GPU-like accelerators.

Figure 1 shows comparison of the fastest version of this benchmark (version 5) with CPU and GPU. Note that the rectangles represent execution time and lower is better.

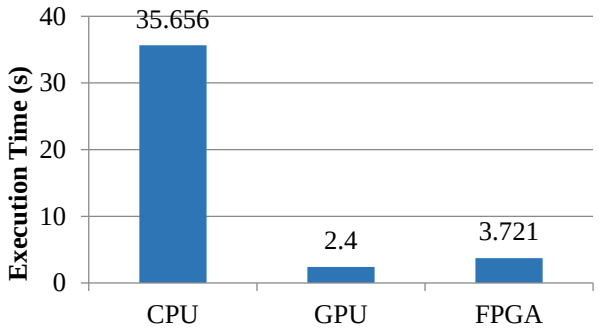

Figure 1 NW FPGA vs. CPU vs. GPU

In this benchmark, FPGA offers 9.6x speed-up against CPU and is only 1.55x slower than the K20X GPU. While the absolute performance of the FPGA is lower than the GPU, it indicates that the FPGA can achieve higher performance per Watt than the GPU.

7.2 Hotspot

Table 4 shows timing results for the Hotspot benchmark. The size of computed grids is 1024x1024 for our evaluation. For this benchmark, version 2 is a thread-pipelined version that builds on version 0 and uses 4-way SIMD and the restrict keyword for the input parameters if possible. Version 3 applies the restrict keyword and unrolling of the inner loop to the version 1 kernel, and version 5 uses the sliding window optimization.

Table 4 Hotspot Results

Version	F_{max}	Run Time (ms)
0	302.48	18.691
1	258.86	1635.237
2	269.685	6.562
3	196.19	9.729
5	227.84	1.701

Similar to the results of NW, there is a wide spectrum of performance depending on the parallelism and optimization used; however, the fastest performance is also achieved with the loop-pipelined version with the sliding window optimization. Although the thread-pipelined version with the basic optimization (version 2) performs better than the loop-pipelined version with the basic optimization (version 3), the effect of the sliding window optimization is significant as shown in version 5. These results indicate that for parallel programs exhibiting the structured grid pattern, the sliding window optimization should be used instead of the common method for GPUs using the local memory.

Figure 2 shows comparison of the fastest version of this benchmark (version 5) with CPU and GPU.

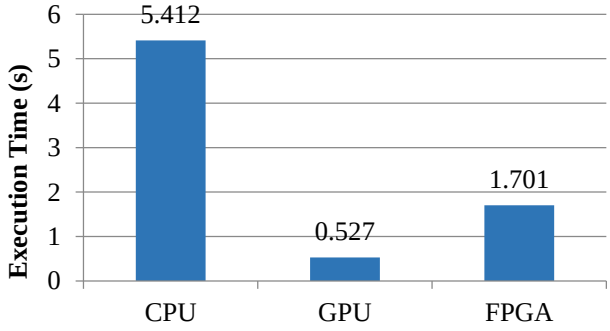

Figure 2 Hotspot FPGA vs. CPU vs. GPU

In this benchmark, the FPGA is 3.2x faster than CPU but is 3.2x slower than GPU. Even in this case, the FPGA most likely offers the best performance per watt.

7.3 Pathfinder

Table 5 shows timing results for the Pathfinder benchmark. In this benchmark, the lengths of each row and column are 100,000 and 100, respectively. For base-line single-threaded version, the for loop on the matrix rows was moved from host code to device code and the restrict keyword was added. Version 2, on the other hand, retains the structure of version 0 but uses the restrict keyword, 16-way SIMD, and 2 compute units; more compute units were not used for version 2 to avoid disabling the automatic local memory sharing between the compute units by the compiler, due to lack of enough on-chip memory. Version 3 uses loop unrolling both on the outer loop and the two inner loops. Version 4, in comparison to version 2, uses 4 compute units and does not have local memory sharing between compute units. Finally, version 5 uses the advanced sliding windows optimization.

Table 5 Pathfinder Results

Version	F_{max}	Run Time (ms)
0	302.48	151.227
1	285.87	41.927
2	164.74	115.337
3	194.32	18.477
4	148.76	120.607
5	142.49	4.569

As shown in Table 5, the results again demonstrate the importance of the FPGA-specific optimization. Compared to the original baseline code of version 0, the fastest version achieves more than 30x improvement of performance.
Figure 3 shows comparison of the fastest version of this benchmark (version 5) with CPU and GPU.

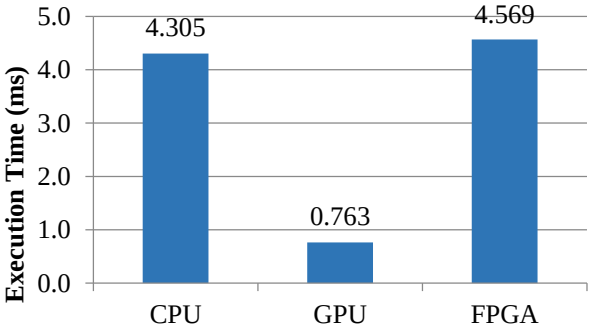

Figure 3 Pathfinder FPGA vs. CPU vs. GPU

In this benchmark FPGA offers nearly the same speed as CPU while but is 6x slower than GPU. Even in this case it is probable that the FPGA offers best power efficiency.

7.4 Nearest Neighbor

Table 6 shows performance results for the NN benchmark. For this benchmark, rather than using the original workload from the Rodinia Suite, we used a bigger workload with 33,554,432 hurricanes and a size of ~1.6 GB, generated by the input generator available in the benchmark.

Table 6 NN Results

Version	F_{max}	Run Time (ms)
0	304.59	110.249
1	303.58	124.306
2	253.16	18.099
3	215.7	21.433
4	227.37	18.534
5	229.41	20.166

Version 2 here uses strict, 16-wide SIMD and 3 compute units, version 3 uses strict and unroll factor of 48, version 4 is the same as version 2 but with 4 compute units and version 5 is the same as version 3 with unroll factor 64.

As the results show, this benchmark does not scale well with unrolling, most likely due to lack of enough memory bandwidth. Version 2 is the fastest version of this kernel. Here, the thread-pipelined kernels are slightly faster than their loop-pipelined counterparts due to working in parallel rather than in a pipelined fashion.

Figure 4 shows comparison of the fastest version of this benchmark (version 2) with GPU.

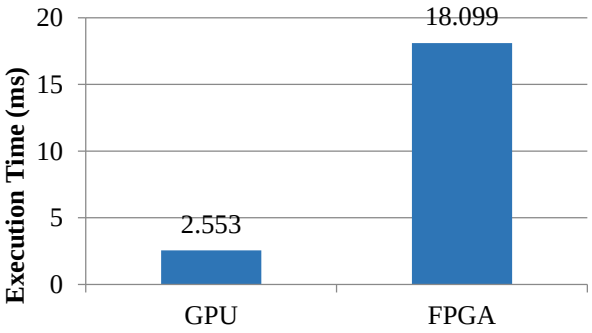

Figure 4 NN FPGA vs. GPU

The timing result we obtained from CPU using the OpenMP version of the benchmark is around 8700 ms since it uses formatted I/O routines within the innermost loop and hence, we find it is not a fair comparison with GPU and FPGA and avoided including CPU results in the chart. In this benchmark, FPGA is 7.44x slower than GPU.

7.5 SRAD

Table 7 shows performance results for the SRAD benchmark with 502x458 2-D matrices. In this benchmark, version 2 uses 2-way SIMD and restrict and version 3 uses restrict and unrolling.

Table 7 SRAD Results

Version	F_{max}	Run Time (s)
0	233.42	1.307
1	242.83	103.050
2	233.69	1.228
3	200.92	9.647

Figure 5 shows comparison of the fastest version of this benchmark (version 2) with CPU and GPU.

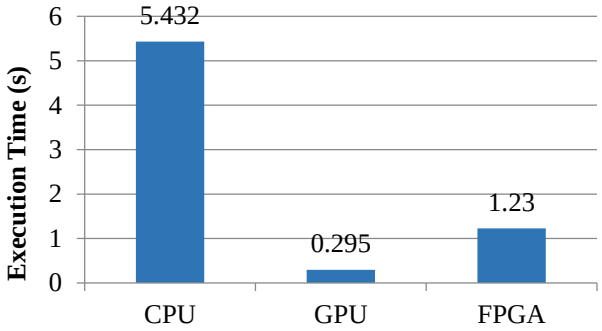

Figure 5 SRAD FPGA vs. CPU vs. GPU

In this benchmark, FPGA offers 4.4x speed-up against CPU while being 4.2x slower than GPU. This benchmark is the only benchmark that performs well on FPGAs with just simple optimizations.

8. Conclusion and Future Work

In this work we presented the results of porting a subset of the Rodinia benchmark suite to the FPGA platform using Altera's OpenCL SDK, and compared run time with an Nvidia Tesla K20X GPU and an Intel E5-2670 CPU.

Based on our findings, even though we could not match the speed of the K20X GPU in any of our benchmarks, we can predict that the FPGA would offer better power efficiency in comparison to GPU, in most cases. While we expect that using hardware description languages would allow us to achieve further better performance, it is still highly promising that such performance can be achieved on FPGAs, even with a "high-level" language such as OpenCL.

Our work is still ongoing and we will continue porting the rest of the benchmarks from the Rodinia Suite for FPGAs and apply more aggressive optimizations on the benchmarks that only have gone through baseline optimization so far. We are also evaluating methods to measure power consumption of all the platforms so that power efficiency of these platforms can be calculated and reported with acceptable accuracy.

Meanwhile, we are awaiting support for the Arria 10 FPGAs in Altera's OpenCL SDK so that we can evaluate the performance of floating-point benchmarks on this new FPGA family. The release of Startix 10 in near future can also potentially turn around the performance of FPGAs in floating-point benchmarks and pave the way for widespread adoption of this platform into future HPC systems.

9. References

[1] Putnam, A. Caulfield, A.M. Chung, E.S. Chiou, D. Constantinides, K. Demme, J. Esmaeilzadeh, H. Fowers, J. Gopal, G.P. Gray, J. Haselman, M. Hauck, S. Heil, S. Hormati, A. Kim, J.-Y. Lanka, S. Larus, J. Peterson, E. Pope, S. Smith, A. Thong, J. Xiao, P.Y. Burger, D.: A reconfigurable fabric for accelerating large-scale datacenter services, In *ACM/IEEE 41st International Symposium on Computer Architecture*, Minneapolis, MN, USA, pp. 13-24, (2014).

[2] Zhang, Z. Fan, Y. Jiang, W. Han, G. Yang, C. and Cong, J.: AutoPilot: A platform-based ESL synthesis system, In *High-Level Synthesis: From Algorithm to Digital Circuit*, Springer, pp. 99-112, (2008).

[3] Xilinx High-Level Synthesis (HLS) 2012.2, http://www.xilinx.com/tools/autoesl_instructions.htm

[4] Cadence Stratus High-Level Synthesis Datasheet, https://www.cadence.com/rl/Resources/datasheets/Stratus_ds.pdf

[5] Synopsis Synphony C Compiler Datasheet,

https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Documents/synphonyc-compiler-ds.pdf
[6] John Sanguinetti: High-level synthesis, verification and language, EE Times, http://www.eetimes.com/document.asp?doc_id=1276220
[7] Altera SDK for OpenCL, https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html
[8] Xilinx SDAccel, http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
[9] Apple Previews Mac OS X Snow Leopard to Developers, http://www.apple.com/pr/library/2008/06/09Apple-Previews-Mac-OS-X-Snow-Leopard-to-Developers.html
[10] Khronos OpenCL Working Group: *The OpenCL Specification: Version 1.0* (2010).
[11] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J., Lee, S.-H. and Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing, In *IEEE International Symposium on Workload Characterization*, Austin, TX, USA, pp. 44-54, (2009).
[12] Owaida, M., Bellas, N., Daloukas, K. and Antonopoulos, C.D.: Synthesis of Platform Architectures from OpenCL Programs, In *IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, Salt Lake City, UT, USA, pp. 186-193, (2011).
[13] Papakonstantinou, A. Gururaj, K. Stratton, J.A. Deming Chen Cong, J. Hwu, W.-M.W.: FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs, In *IEEE 7th Symposium on Application Specific Processors*, San Francisco, CA, USA, pp. 35-42, (2009).
[14] Krommydas, K. Wu-chun Feng Owaida, M. Antonopoulos, C.D. Bellas, N.: On the characterization of OpenCL dwarfs on fixed and reconfigurable platforms, In *IEEE 25th International Conference on Application-specific Systems, Architectures and Processors*, Zurich, Switzerland, pp 153-160, (2014).
[15] Yuliang Pu. Jun Peng. Letian Huang. Chen, J.: An Efficient KNN Algorithm Implemented on FPGA Based Heterogeneous Computing System Using OpenCL, In *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, Vancouver, BC, Canada, pp. 167-170, (2015).
[16] Morales, V.M. Horrein, P.-H. Baghdadi, A. Hochapfel, E. Vaton, S.: Energy-efficient FPGA implementation for binomial option pricing using OpenCL, In *Design, Automation and Test in Europe Conference and Exhibition,* Dresden, Germany, pp.1-6, (2014).
[17] Settle S.: High-performance Dynamic Programming on FPGAs with OpenCL, In *IEEE High Performance Extreme Computing Conference*, Waltham, MA, USA (2013).
[18] Hill, Kenneth. Craciun, Stefan. George, Alan. Lam, Herman: Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA, In *IEEE 26th International Conference on Application-specific Systems, Architectures and Processors*, Toronto, ON, Canada, pp. 189-193, (2015).
[19] Che S. Li J. Sheaffer J.W. Skadron K. Lach J.: Accelerating Compute-Intensive Applications with GPUs and FPGAs, In *Symposium on Application Specific Processors*, Anaheim, CA, USA, 2008, pp. 101-107, (2008).
[20] Asanovic K. Bodik R. Demmel J. Keaveny T. Keutzer K. Kubiatowicz J. Morgan N. Patterson D. Sen K. Wawrzynek J. Wessel D. and Yelick K.: A view of the parallel computing landscape, *Communications of the ACM,* Vol. 52, No. 10, pp. 56-67, (2009).
[21] Altera Corporation: Altera SDK for OpenCL: Programming Guide, https://www.altera.com/literature/hb/opencl-sdk/aocl_programming_guide.pdf
[22] Altera Corporation: Altera SDK for OpenCL: Best Practices Guide, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_optimization_guide.pdf
[23] Altera Corporation: Finite Difference Computation (3D) Design Example, https://www.altera.com/support/support-resources/design-examples/design-software/opencl/fdtd-3d.html